

# Dart Server – Backend

Lehrbuch Teil B: Server, REST API, Datenbanken, Auth & Deployment

Lehrplan für Entwickler mit Vorkenntnissen in C++, JavaScript und Python

February 2026

# Contents

<b>1</b>	<b>Block 5: Server-Grundlagen</b>	<b>2</b>
1.1	Einheit 5.1: Dart auf dem Server . . . . .	2
1.2	Einheit 5.2: Shelf Framework Basics . . . . .	23
1.3	Einheit 5.3: Routing mit shelf_router . . . . .	47
1.4	Einheit 5.4: Middleware . . . . .	74
1.5	Einheit 5.5: Konfiguration & Environment . . . . .	100
1.6	Einheit 5.6: Projektstruktur . . . . .	126
<b>2</b>	<b>Block 6: REST API Entwicklung</b>	<b>158</b>
2.1	Einheit 6.1: REST Prinzipien & Design . . . . .	158
2.2	Einheit 6.2: JSON Serialisierung . . . . .	178
2.3	Einheit 6.3: Request Body Parsing . . . . .	200
2.4	Einheit 6.4: CRUD Operationen . . . . .	227
2.5	Einheit 6.5: Input Validierung . . . . .	259
2.6	Einheit 6.6: Error Handling . . . . .	295
2.7	Einheit 6.7: Pagination & Filtering . . . . .	329
2.8	Einheit 6.8: API Dokumentation . . . . .	360
<b>3</b>	<b>Block 7: Datenbanken</b>	<b>420</b>
3.1	Einheit 7.1: SQL Grundlagen & PostgreSQL . . . . .	420
3.2	Einheit 7.2: PostgreSQL mit Dart . . . . .	450
3.3	Einheit 7.3: Repository Pattern . . . . .	489
3.4	Einheit 7.4: Relationale Modellierung . . . . .	537
3.5	Einheit 7.5: Migrations . . . . .	561
3.6	Einheit 7.6: NoSQL mit MongoDB . . . . .	580
3.7	Einheit 7.7: Queries & Aggregationen . . . . .	602
3.8	Einheit 7.8: Caching mit Redis . . . . .	624
<b>4</b>	<b>Block 8: Authentifizierung &amp; Sicherheit</b>	<b>652</b>
4.1	Einheit 8.1: Passwort Hashing . . . . .	652
4.2	Einheit 8.2: JWT Authentication . . . . .	693
4.3	Einheit 8.3: Auth Middleware . . . . .	734
4.4	Einheit 8.4: OAuth 2.0 & Social Login . . . . .	776
4.5	Einheit 8.5: API Sicherheit . . . . .	820
4.6	Einheit 8.6: Auth Testing . . . . .	861
<b>5</b>	<b>Block 9: Produktion &amp; Abschlussprojekt</b>	<b>908</b>
5.1	Einheit 9.1: WebSockets & Real-time . . . . .	908
5.2	Einheit 9.2: Background Jobs & Scheduling . . . . .	962
5.3	Einheit 9.3: Logging & Monitoring . . . . .	1019
5.4	Einheit 9.4: Deployment & Docker . . . . .	1075
5.5	Einheit 9.5: Abschlussprojekt Backend . . . . .	1115

# Chapter 1

## Block 5: Server-Grundlagen

Dart auf dem Server, Shelf-Framework, Routing, Middleware und Projektstruktur.

### 1.1 Einheit 5.1: Dart auf dem Server

#### 1.1.0.1 Lernziele

Nach dieser Einheit kannst du: - Dart-Programme ohne Flutter ausführen - Die `dart:io`-Bibliothek für Server-Operationen nutzen - Einen einfachen HTTP-Server erstellen - Den Request/Response-Lifecycle verstehen

#### 1.1.0.2 Dart außerhalb von Flutter

Dart ist nicht nur für Flutter-Apps geeignet. Die Sprache wurde ursprünglich als allgemeine Programmiersprache entwickelt und eignet sich hervorragend für:

- **Command-Line Tools** (CLI)
- **Backend-Server** (REST APIs, WebSockets)
- **Automatisierungsskripte**
- **Microservices**

Dart-Projekt ohne Flutter erstellen

```
# Einfaches Dart-Projekt
dart create my_cli_tool

# Server-Projekt mit Shelf-Template
dart create -t server-shelf my_api

# Package-Projekt
dart create -t package my_package
```

Projektstruktur eines Server-Projekts

```
my_api/
+-- bin/
|   +-- server.dart      # Einstiegspunkt
+-- lib/
|   +-- my_api.dart      # Bibliotheks-Code
+-- test/
|   +-- my_api_test.dart # Tests
+-- pubspec.yaml         # Abhängigkeiten
+-- analysis_options.yaml
+-- README.md
```

Dart-Programm ausführen

```
# Direkt ausführen (JIT-Kompilierung)
dart run bin/server.dart

# Mit Argumenten
dart run bin/server.dart --port=8080

# Zu nativer Executable kompilieren (AOT)
dart compile exe bin/server.dart -o server
./server
```

### 1.1.0.3 Die dart:io Bibliothek

dart:io ist die Kernbibliothek für I/O-Operationen auf dem Server. Sie ist nur verfügbar, wenn Dart nativ läuft (nicht im Browser).

Wichtige Klassen in dart:io

Klasse	Zweck
HttpServer	HTTP-Server erstellen
HttpRequest	Eingehende Anfragen
HttpResponse	Antworten senden
File	Dateioperationen
Directory	Verzeichnisoperationen
Socket	TCP-Sockets
Platform	Plattform-Informationen

Plattform-Informationen abrufen

```
import 'dart:io';

void main() {
  print('Betriebssystem: ${Platform.operatingSystem}');
  print('Anzahl Prozessoren: ${Platform.numberOfProcessors}');
  print('Hostname: ${Platform.localHostname}');
  print('Dart-Version: ${Platform.version}');

  // Umgebungsvariablen
  final home = Platform.environment['HOME'];
  print('Home-Verzeichnis: $home');
}
```

Dateioperationen

```
import 'dart:io';

Future<void> main() async {
  // Datei lesen
  final file = File('config.json');

  if (await file.exists()) {
    final content = await file.readAsString();
```

```

    print('Inhalt: $content');
  }

  // Datei schreiben
  final logFile = File('app.log');
  await logFile.writeAsString(
    '${DateTime.now()}: Server gestartet\n',
    mode: FileMode.append,
  );

  // Verzeichnis auflisten
  final dir = Directory('.');
  await for (final entity in dir.list()) {
    print(entity.path);
  }
}

```

#### 1.1.0.4 HTTP-Grundlagen

Bevor wir einen Server bauen, müssen wir HTTP verstehen.

Das HTTP-Protokoll

HTTP (Hypertext Transfer Protocol) ist ein Request-Response-Protokoll:

1. **Client** sendet einen **Request** an den Server
2. **Server** verarbeitet den Request
3. **Server** sendet eine **Response** zurück

HTTP-Request-Struktur

```

GET /api/users HTTP/1.1
Host: localhost:8080
Accept: application/json
Authorization: Bearer token123

```

Bestandteile: - **Methode**: GET, POST, PUT, DELETE, PATCH, etc. - **Pfad**: /api/users - **HTTP-Version**: HTTP/1.1 - **Headers**: Metadaten (Host, Content-Type, etc.) - **Body**: Daten (bei POST/PUT)

HTTP-Response-Struktur

```

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 45

```

```
{"users": [{"id": 1, "name": "Max"}]}
```

Bestandteile: - **Statuscode**: 200, 404, 500, etc. - **Status-Text**: OK, Not Found, etc. - **Headers**: Metadaten - **Body**: Antwortdaten

Wichtige HTTP-Statuscodes

Code	Bedeutung	Verwendung
200	OK	Erfolgreiche GET/PUT-Anfrage

Code	Bedeutung	Verwendung
201	Created	Ressource erfolgreich erstellt
204	No Content	Erfolg ohne Rückgabedaten
400	Bad Request	Ungültige Anfrage
401	Unauthorized	Authentifizierung erforderlich
403	Forbidden	Keine Berechtigung
404	Not Found	Ressource nicht gefunden
500	Internal Server Error	Serverfehler

### HTTP-Methoden

Methode	Zweck	Idempotent	Body
GET	Daten abrufen	Ja	Nein
POST	Neue Ressource erstellen	Nein	Ja
PUT	Ressource ersetzen	Ja	Ja
PATCH	Ressource teilweise ändern	Nein	Ja
DELETE	Ressource löschen	Ja	Nein

#### 1.1.0.5 Einfacher HTTP-Server mit dart:io

Lass uns einen minimalen HTTP-Server bauen:

Minimaler Server

```
import 'dart:io';

Future<void> main() async {
  // Server auf localhost:8080 starten
  final server = await HttpServer.bind(
    InternetAddress.loopbackIPv4, // 127.0.0.1
    8080,
  );

  print('Server läuft auf http://localhost:8080');

  // Auf eingehende Requests warten
  await for (final request in server) {
    // Einfache Antwort senden
    request.response
      ..statusCode = HttpStatus.ok
      ..headers.contentType = ContentType.text
      ..write('Hallo vom Dart-Server!')
      ..close();
  }
}
```

Server mit einfachem Routing

```
import 'dart:io';
import 'dart:convert';
```

```
Future<void> main() async {
  final server = await HttpServer.bind(
    InternetAddress.anyIPv4,
    8080,
  );

  print('Server läuft auf http://localhost:8080');

  await for (final request in server) {
    await handleRequest(request);
  }
}

Future<void> handleRequest(HttpRequest request) async {
  final path = request.uri.path;
  final method = request.method;

  print('$method $path');

  // Einfaches Routing
  if (path == '/' && method == 'GET') {
    _sendJson(request.response, {'message': 'Willkommen!'});
  } else if (path == '/health' && method == 'GET') {
    _sendJson(request.response, {'status': 'ok'});
  } else if (path == '/echo' && method == 'POST') {
    await _handleEcho(request);
  } else {
    _sendNotFound(request.response);
  }
}

void _sendJson(HttpResponse response, Map<String, dynamic> data) {
  response
    ..statusCode = HttpStatus.ok
    ..headers.contentType = ContentType.json
    ..write(jsonEncode(data))
    ..close();
}

void _sendNotFound(HttpResponse response) {
  response
    ..statusCode = HttpStatus.notFound
    ..headers.contentType = ContentType.json
    ..write(jsonEncode({'error': 'Not Found'}))
    ..close();
}

Future<void> _handleEcho(HttpRequest request) async {
  // Request-Body lesen
  final body = await utf8.decoder.bind(request).join();
}
```

```
request.response
  ..statusCode = HttpStatus.ok
  ..headers.contentType = ContentType.json
  ..write(jsonEncode({
    'echo': body,
    'timestamp': DateTime.now().toIso8601String(),
  }))
  ..close();
}
```

Server mit Query-Parametern

```
import 'dart:io';
import 'dart:convert';

Future<void> main() async {
  final server = await HttpServer.bind(InternetAddress.anyIPv4, 8080);
  print('Server läuft auf http://localhost:8080');

  await for (final request in server) {
    if (request.uri.path == '/search') {
      handleSearch(request);
    } else {
      request.response
        ..statusCode = HttpStatus.notFound
        ..close();
    }
  }
}

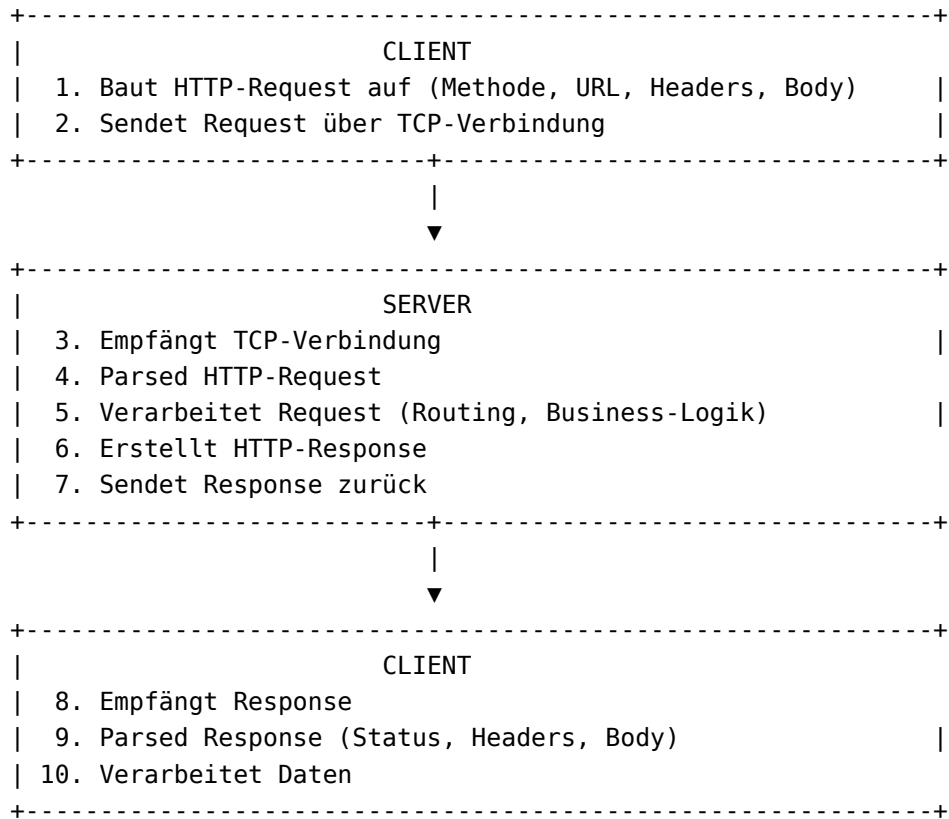
void handleSearch(HttpRequest request) {
  // Query-Parameter auslesen
  // URL: /search?q=dart&limit=10
  final params = request.uri.queryParameters;
  final query = params['q'] ?? '';
  final limit = int.tryParse(params['limit'] ?? '10') ?? 10;

  // Alle Query-Parameter (auch mehrfache)
  final allParams = request.uri.queryParametersAll;
  // /search?tag=dart&tag=flutter -> {'tag': ['dart', 'flutter']}

  request.response
    ..statusCode = HttpStatus.ok
    ..headers.contentType = ContentType.json
    ..write(jsonEncode({
      'query': query,
      'limit': limit,
      'allParams': allParams,
    }))
    ..close();
}
```

### 1.1.0.6 Request/Response Lifecycle

Der Lebenszyklus einer HTTP-Anfrage:



Request-Eigenschaften in dart:io

```

await for (final request in server) {
  // HTTP-Methode
  print('Methode: ${request.method}');

  // URI-Informationen
  print('Pfad: ${request.uri.path}');
  print('Query: ${request.uri.query}');
  print('Query-Params: ${request.uri.queryParameters}');

  // Headers
  print('Content-Type: ${request.headers.contentType}');
  print('Accept: ${request.headers.value('accept')}');
  print('User-Agent: ${request.headers.value('user-agent')}');

  // Alle Headers ausgeben
  request.headers.forEach((name, values) {
    print('$name: ${values.join(', ')}');
  });

  // Client-Informationen
  print('Client-IP: ${request.connectionInfo?.remoteAddress.address}');
  print('Client-Port: ${request.connectionInfo?.remotePort}');
}

```

```
// Body lesen (für POST/PUT)
if (request.method == 'POST') {
  final body = await utf8.decoder.bind(request).join();
  print('Body: $body');
}

// Response senden
request.response.close();
}
```

Response konfigurieren

```
void sendResponse(HttpRequest request) {
  final response = request.response;

  // Statuscode setzen
  response.statusCode = HttpStatus.ok; // 200

  // Headers setzen
  response.headers.contentType = ContentType.json;
  response.headers.add('X-Custom-Header', 'Wert');
  response.headers.add('Cache-Control', 'no-cache');

  // CORS-Headers (für Browser-Zugriff)
  response.headers.add('Access-Control-Allow-Origin', '*');

  // Body schreiben
  response.write({'data': 'Hello'});

  // Oder mit Bytes
  // response.add(utf8.encode({'data': 'Hello'}));

  // Response abschließen
  response.close();
}
```

### 1.1.0.7 Graceful Shutdown

Ein Server sollte sauber herunterfahren können:

```
import 'dart:io';

HttpServer? _server;

Future<void> main() async {
  // Signal-Handler für Ctrl+C (SIGINT)
  ProcessSignal.sigint.watch().listen((_) async {
    print('\nServer wird heruntergefahren...');
    await _server?.close();
    exit(0);
  });
}
```

```
_server = await HttpServer.bind(InternetAddress.anyIPv4, 8080);
print('Server läuft auf http://localhost:8080');
print('Drücke Ctrl+C zum Beenden');

await for (final request in _server!) {
  request.response
    ..write('Hello')
    ..close();
}
}
```

#### 1.1.0.8 Zusammenfassung

- **dart:io** ermöglicht Server-Entwicklung ohne Flutter
- **HttpServer** ist die Basis-Klasse für HTTP-Server
- Der **Request/Response-Lifecycle** folgt dem HTTP-Protokoll
- **Query-Parameter** werden über `uri.queryParameters` ausgelesen
- **Request-Body** wird als Stream gelesen
- Für produktive Server verwenden wir in der nächsten Einheit das **Shelf-Framework**

#### 1.1.0.9 Nächste Schritte

In der nächsten Einheit lernst du das **Shelf-Framework**, das eine elegantere Abstraktion für HTTP-Server bietet und Features wie Middleware, Routing und Testing erleichtert.

### 1.1.1 Übung

#### 1.1.1.1 Ziel

Erstelle einen einfachen HTTP-Server mit **dart:io**, der verschiedene Endpunkte bereitstellt und mit Requests umgehen kann.

#### 1.1.1.2 Aufgabe 1: Projekt-Setup (10 min)

Erstelle ein neues Dart-Projekt für deinen Server.

Schritte:

1. Erstelle ein neues Dart-Projekt:

```
dart create -t console simple_server
cd simple_server
```

2. Ersetze den Inhalt von `bin/simple_server.dart` mit einem minimalen Server:

```
import 'dart:io';

Future<void> main() async {
  final server = await HttpServer.bind(
    InternetAddress.loopbackIPv4,
    8080,
  );

  print('Server läuft auf http://localhost:8080');
```

```
    await for (final request in server) {  
      request.response  
        ..write('Hello World')  
        ..close();  
    }  
  }  
}
```

3. Starte den Server:

```
dart run
```

4. Teste mit curl oder Browser:

```
curl http://localhost:8080
```

### 1.1.1.3 Aufgabe 2: Einfaches Routing (20 min)

Erweitere den Server um verschiedene Endpunkte.

Anforderungen:

Implementiere folgende Routen:

Route	Methode	Response
/	GET	Willkommensnachricht als Text
/api/info	GET	JSON mit Server-Infos
/api/time	GET	JSON mit aktueller Uhrzeit
/api/echo	POST	Echo des Request-Bodys als JSON
Alles andere	*	404 Not Found

Erwartete Responses:

**GET /**

Willkommen beim Simple Server!

**GET /api/info**

```
{  
  "name": "Simple Server",  
  "version": "1.0.0",  
  "dart_version": "3.x.x"  
}
```

**GET /api/time**

```
{  
  "timestamp": "2024-01-15T14:30:00.000Z",  
  "timezone": "Europe/Berlin"  
}
```

**POST /api/echo** (mit Body {"message": "Hallo"})

```
{
  "received": {"message": "Hallo"},
  "timestamp": "2024-01-15T14:30:00.000Z"
}
```

#### 1.1.1.4 Aufgabe 3: Query-Parameter (15 min)

Füge einen neuen Endpunkt hinzu, der Query-Parameter verarbeitet.

Anforderungen:

**GET /api/greet**

Parameter	Pflicht	Default	Beschreibung
name	Nein	“Gast”	Name der Person
lang	Nein	“de”	Sprache (de/en)

Erwartete Responses:

**GET /api/greet?name=Max&lang=de**

```
{
  "greeting": "Hallo, Max!",
  "language": "de"
}
```

**GET /api/greet?name=Max&lang=en**

```
{
  "greeting": "Hello, Max!",
  "language": "en"
}
```

**GET /api/greet** (ohne Parameter)

```
{
  "greeting": "Hallo, Gast!",
  "language": "de"
}
```

#### 1.1.1.5 Aufgabe 4: Request-Logging (10 min)

Implementiere ein einfaches Logging für alle eingehenden Requests.

Anforderungen:

Bei jedem Request soll folgendes in der Konsole ausgegeben werden:

```
[2024-01-15 14:30:00] GET /api/info - 200 (12ms)
[2024-01-15 14:30:05] POST /api/echo - 200 (5ms)
[2024-01-15 14:30:10] GET /unknown - 404 (1ms)
```

Format: [Timestamp] Methode Pfad - Statuscode (Dauer)

Tipps:

- Speichere die Startzeit vor der Request-Verarbeitung
- Berechne die Dauer nach dem Senden der Response
- Nutze `DateTime.now()` für Timestamps

#### 1.1.1.6 Aufgabe 5: Graceful Shutdown (5 min)

Implementiere einen sauberen Server-Shutdown.

Anforderungen:

1. Der Server soll bei SIGINT (Ctrl+C) sauber herunterfahren
2. Vor dem Beenden soll eine Nachricht ausgegeben werden
3. Laufende Requests sollen noch abgeschlossen werden können

Erwartete Ausgabe:

Server läuft auf `http://localhost:8080`  
Drücke Ctrl+C zum Beenden

^C

[Shutdown] Server wird heruntergefahren...

[Shutdown] Auf Wiedersehen!

#### 1.1.1.7 Bonus-Aufgabe: Health Check mit Statistiken

Implementiere einen `/health`-Endpunkt, der Statistiken über den Server liefert.

Anforderungen:

**GET /health**

```
{
  "status": "healthy",
  "uptime_seconds": 3600,
  "requests_total": 150,
  "requests_per_minute": 2.5,
  "memory_usage_mb": 45.2
}
```

Tipps:

- Speichere die Startzeit des Servers
- Zähle alle eingehenden Requests
- Nutze `ProcessInfo.currentRss` für Speicherverbrauch

#### 1.1.1.8 Testen

Teste deinen Server mit curl:

```
# Aufgabe 2
curl http://localhost:8080/
curl http://localhost:8080/api/info
curl http://localhost:8080/api/time
curl -X POST http://localhost:8080/api/echo \
  -H "Content-Type: application/json" \
  -d '{"message": "Test"}'
```

```
# Aufgabe 3
curl "http://localhost:8080/api/greet"
curl "http://localhost:8080/api/greet?name=Max"
curl "http://localhost:8080/api/greet?name=Max&lang=en"

# 404 testen
curl http://localhost:8080/nicht-vorhanden

# Bonus
curl http://localhost:8080/health
```

### 1.1.1.9 Abgabe-Checkliste

- ☐ Server startet auf Port 8080
- ☐ Alle Routen aus Aufgabe 2 funktionieren
- ☐ Query-Parameter werden korrekt verarbeitet
- ☐ Logging zeigt alle Requests mit Dauer
- ☐ Graceful Shutdown funktioniert
- ☐ (Bonus) Health-Endpunkt mit Statistiken

## 1.1.2 Lösung

### 1.1.2.1 Vollständige Lösung

```
import 'dart:io';
import 'dart:convert';

// Globale Variablen für Statistiken
DateTime? _serverStartTime;
int _requestCount = 0;
HttpServer? _server;

Future<void> main() async {
  _serverStartTime = DateTime.now();

  // Signal-Handler für Ctrl+C
  ProcessSignal.sigint.watch().listen((_) async {
    print('\n[Shutdown] Server wird heruntergefahren...');
    await _server?.close();
    print('[Shutdown] Auf Wiedersehen!');
    exit(0);
  });

  _server = await HttpServer.bind(
    InternetAddress.loopbackIPv4,
    8080,
  );

  print('Server läuft auf http://localhost:8080');
  print('Drücke Ctrl+C zum Beenden\n');
```

```
await for (final request in _server!) {
  await _handleRequest(request);
}

Future<void> _handleRequest(HttpRequest request) async {
  final startTime = DateTime.now();
  _requestCount++;

  final path = request.uri.path;
  final method = request.method;

  try {
    // Routing
    if (path == '/' && method == 'GET') {
      _handleRoot(request);
    } else if (path == '/api/info' && method == 'GET') {
      _handleInfo(request);
    } else if (path == '/api/time' && method == 'GET') {
      _handleTime(request);
    } else if (path == '/api/echo' && method == 'POST') {
      await _handleEcho(request);
    } else if (path == '/api/greet' && method == 'GET') {
      _handleGreet(request);
    } else if (path == '/health' && method == 'GET') {
      _handleHealth(request);
    } else {
      _handleNotFound(request);
    }
  } catch (e) {
    _handleError(request, e);
  }

  // Logging
  final duration = DateTime.now().difference(startTime).inMilliseconds;
  final timestamp = _formatTimestamp(DateTime.now());
  final statusCode = request.response.statusCode;
  print('[${timestamp}] $method $path - $statusCode (${duration}ms)');
}

// Aufgabe 2: Routing

void _handleRoot(HttpRequest request) {
  request.response
    ..statusCode = HttpStatus.ok
    ..headers.contentType = ContentType.text
    ..write('Willkommen beim Simple Server!')
    ..close();
}

void _handleInfo(HttpRequest request) {
```

```
final info = {
  'name': 'Simple Server',
  'version': '1.0.0',
  'dart_version': Platform.version.split(' ').first,
};
_sendJson(request.response, info);
}

void _handleTime(HttpRequest request) {
  final now = DateTime.now();
  final timeInfo = {
    'timestamp': now.toUtc().toIso8601String(),
    'timezone': now.timeZoneName,
  };
  _sendJson(request.response, timeInfo);
}

Future<void> _handleEcho(HttpRequest request) async {
  try {
    final body = await utf8.decoder.bind(request).join();
    final data = body.isNotEmpty ? jsonDecode(body) : {};

    final response = {
      'received': data,
      'timestamp': DateTime.now().toUtc().toIso8601String(),
    };
    _sendJson(request.response, response);
  } catch (e) {
    _sendJson(
      request.response,
      {'error': 'Invalid JSON'},
      statusCode: HttpStatus.badRequest,
    );
  }
}

// Aufgabe 3: Query-Parameter

void _handleGreet(HttpRequest request) {
  final params = request.uri.queryParameters;
  final name = params['name'] ?? 'Gast';
  final lang = params['lang'] ?? 'de';

  String greeting;
  switch (lang) {
    case 'en':
      greeting = 'Hello, $name!';
      break;
    case 'de':
    default:
      greeting = 'Hallo, $name!';
  }
}
```

```
}

final response = {
  'greeting': greeting,
  'language': lang,
};
_sendJson(request.response, response);
}

// Bonus: Health Check

void _handleHealth(HttpRequest request) {
  final uptime = DateTime.now().difference(_serverStartTime!);
  final uptimeSeconds = uptime.inSeconds;
  final requestsPerMinute = uptimeSeconds > 0
    ? (_requestCount / (uptimeSeconds / 60)).toStringAsFixed(1)
    : '0.0';

  // Speicherverbrauch in MB
  final memoryBytes = ProcessInfo.currentRss;
  final memoryMb = (memoryBytes / (1024 * 1024)).toStringAsFixed(1);

  final health = {
    'status': 'healthy',
    'uptime_seconds': uptimeSeconds,
    'requests_total': _requestCount,
    'requests_per_minute': double.parse(requestsPerMinute),
    'memory_usage_mb': double.parse(memoryMb),
  };
  _sendJson(request.response, health);
}

// Error-Handler

void _handleNotFound(HttpRequest request) {
  _sendJson(
    request.response,
    {
      'error': 'Not Found',
      'path': request.uri.path,
      'method': request.method,
    },
    statusCode: HttpStatus.notFound,
  );
}

void _handleError(HttpRequest request, dynamic error) {
  _sendJson(
    request.response,
    {
      'error': 'Internal Server Error',
    },
  );
}
```

```

        'message': error.toString(),
      },
      statusCode: HttpStatus.internalServerError,
    );
  }

// Hilfsfunktionen

void _sendJson(
  HttpResponse response,
  Map<String, dynamic> data, {
    int statusCode = HttpStatus.ok,
  }) {
  response
    ..statusCode = statusCode
    ..headers.contentType = ContentType.json
    ..write(jsonEncode(data))
    ..close();
}

String _formatTimestamp(DateTime dt) {
  return '${dt.year}-${_pad(dt.month)}-${_pad(dt.day)} '
    '${_pad(dt.hour)}:${_pad(dt.minute)}:${_pad(dt.second)}';
}

String _pad(int n) => n.toString().padLeft(2, '0');

```

### 1.1.2.2 Erklärungen

#### Aufgabe 1: Projekt-Setup

Das Projekt wird mit `dart create -t console` erstellt. Die `-t console`-Option erstellt ein minimales Projekt ohne zusätzliche Abhängigkeiten.

#### Aufgabe 2: Routing

Das Routing erfolgt über einfache `if-else`-Bedingungen: - Jede Route prüft Pfad UND Methode - Am Ende steht `_handleNotFound` als Fallback - JSON-Responses werden über die Hilfsfunktion `_sendJson` gesendet

#### Aufgabe 3: Query-Parameter

Query-Parameter werden über `request.uri.queryParameters` ausgelesen: - Gibt eine `Map<String, String>` zurück - Bei fehlenden Parametern wird `null` zurückgegeben - Der `??`-Operator setzt Default-Werte

#### Aufgabe 4: Logging

Das Logging misst die Zeit vor und nach der Request-Verarbeitung: - `startTime` wird zu Beginn gespeichert - Nach `close()` wird die Differenz berechnet - Das Format ist konfigurierbar

#### Aufgabe 5: Graceful Shutdown

Der Shutdown wird über `ProcessSignal.sigint` implementiert: - `watch()` gibt einen Stream zurück - `listen()` registriert einen Handler - `server.close()` beendet den Server sauber

Bonus: Health Check

Der Health-Endpunkt sammelt verschiedene Metriken: - **Uptime**: Differenz zwischen Start und Jetzt - **Request-Count**: Globale Variable, die bei jedem Request erhöht wird - **Memory**: `ProcessInfo.currentRss` gibt den aktuellen Speicherverbrauch zurück

### 1.1.2.3 Test-Befehle

```
# Server starten
dart run bin/simple_server.dart

# In einem anderen Terminal testen:

# Root
curl http://localhost:8080/

# Info
curl http://localhost:8080/api/info

# Time
curl http://localhost:8080/api/time

# Echo
curl -X POST http://localhost:8080/api/echo \
  -H "Content-Type: application/json" \
  -d '{"message": "Hallo", "count": 42}'

# Greet (verschiedene Varianten)
curl "http://localhost:8080/api/greet"
curl "http://localhost:8080/api/greet?name=Max"
curl "http://localhost:8080/api/greet?name=Max&lang=en"
curl "http://localhost:8080/api/greet?name=Anna&lang=de"

# Health
curl http://localhost:8080/health

# 404 testen
curl http://localhost:8080/nicht-vorhanden
curl -X POST http://localhost:8080/api/info

# Shutdown testen: Drücke Ctrl+C im Server-Terminal
```

### 1.1.2.4 Beispiel-Ausgabe

Server-Konsole:

Server läuft auf `http://localhost:8080`  
Drücke `Ctrl+C` zum Beenden

```
[2024-01-15 14:30:00] GET / - 200 (2ms)
[2024-01-15 14:30:05] GET /api/info - 200 (1ms)
[2024-01-15 14:30:10] GET /api/time - 200 (1ms)
```



```
final body = await utf8.decoder.bind(request).join();
```

### 3. JSON-Fehler nicht abgefangen

```
// FALSCH: Kein Error-Handling
final data = jsonDecode(body);

// RICHTIG: Mit try-catch
try {
  final data = jsonDecode(body);
} catch (e) {
  // Fehler-Response senden
}
```

### 4. Content-Type nicht gesetzt

```
// FALSCH: Browser interpretiert als Text
response.write({'data': 'value'});

// RICHTIG: Content-Type setzen
response.headers.contentType = ContentType.json;
response.write({'data': 'value'});
```

## 1.1.3 Ressourcen

### 1.1.3.1 Offizielle Dokumentation

- Dart auf dem Server - Offizielle Übersicht
- dart:io Bibliothek - API-Referenz
- HttpServer Klasse - Detaillierte Dokumentation
- Write command-line apps - CLI-Tutorial

### 1.1.3.2 HTTP-Grundlagen

- HTTP-Übersicht (MDN) - Grundlagen
- HTTP-Statuscodes - Alle Codes erklärt
- HTTP-Methoden - GET, POST, PUT, etc.
- HTTP-Headers - Header-Referenz

### 1.1.3.3 Tools zum Testen

curl (Kommandozeile)

*# GET-Request*

```
curl http://localhost:8080
```

*# POST-Request mit JSON*

```
curl -X POST http://localhost:8080/api/data \
  -H "Content-Type: application/json" \
  -d '{"name": "Test"}'
```

*# Mit Headers*

```
curl -H "Authorization: Bearer token" http://localhost:8080/api/protected
```

```
# Verbose-Modus (zeigt Request/Response-Details)
curl -v http://localhost:8080
```

HTTPie (benutzerfreundliche Alternative zu curl)

```
# Installation
# macOS: brew install httpie
# Linux: apt install httpie

# GET-Request
http localhost:8080

# POST mit JSON
http POST localhost:8080/api/data name=Test

# Mit Headers
http localhost:8080/api/protected Authorization:"Bearer token"
```

GUI-Tools

- Postman - Beliebtes API-Testing-Tool
- Insomnia - Leichtgewichtige Alternative
- Bruno - Open-Source, Git-freundlich
- Thunder Client - VS Code Extension

#### 1.1.3.4 Dart-Pakete für Server-Entwicklung

Paket	Beschreibung	pub.dev
shelf	Modulares Web-Server-Framework	<a href="#">Link</a>
shelf_router	Routing für Shelf	<a href="#">Link</a>
dart_frog	Full-Stack Framework	<a href="#">Link</a>
serverpod	Backend-Framework mit ORM	<a href="#">Link</a>

#### 1.1.3.5 Videos & Tutorials

- Building a Dart Server - Einführung
- Dart Backend Development - Offizielles Tutorial

#### 1.1.3.6 Beispielprojekte

- shelf Beispiele
- Dart Samples

#### 1.1.3.7 Cheat Sheet: curl-Befehle

```
# Verschiedene HTTP-Methoden
curl -X GET http://localhost:8080/users
curl -X POST http://localhost:8080/users -d '{"name":"Max"}'
curl -X PUT http://localhost:8080/users/1 -d '{"name":"Updated"}'
curl -X DELETE http://localhost:8080/users/1
```

```
# JSON senden
curl -X POST http://localhost:8080/api \
  -H "Content-Type: application/json" \
  -d '{"key": "value"}'

# Response-Header anzeigen
curl -I http://localhost:8080

# Nur Statuscode anzeigen
curl -s -o /dev/null -w "%{http_code}" http://localhost:8080

# Mit Timeout
curl --max-time 5 http://localhost:8080

# Datei hochladen
curl -X POST http://localhost:8080/upload \
  -F "file=@/path/to/file.txt"
```

### 1.1.3.8 Cheat Sheet: Dart-Server

```
// Server starten
final server = await HttpServer.bind(
  InternetAddress.anyIPv4,
  8080,
);

// Request-Infos
request.method          // GET, POST, etc.
request.uri.path        // /api/users
request.uri.queryParameters // {key: value}
request.headers         // HttpHeaders

// Response senden
response.statusCode = 200;
response.headers.contentType = ContentType.json;
response.write({'data': 'value'});
response.close();

// Body lesen
final body = await utf8.decoder.bind(request).join();
final json = jsonDecode(body);
```

## 1.2 Einheit 5.2: Shelf Framework Basics

### 1.2.0.1 Lernziele

Nach dieser Einheit kannst du: - Die Shelf-Architektur verstehen - Handler und Middleware-Konzepte anwenden - Request/Response-Objekte von Shelf nutzen - Einen Shelf-Server aufsetzen und konfigurieren

### 1.2.0.2 Was ist Shelf?

**Shelf** ist das offizielle Web-Server-Framework für Dart. Es bietet eine elegante, funktionale Abstraktion für HTTP-Server.

Vorteile gegenüber dart:io

dart:io	Shelf
Low-Level API	High-Level Abstraktion
Manuelle Request-Verarbeitung	Funktionale Handler
Kein Middleware-Konzept	Middleware-Pipeline
Schwer testbar	Einfach testbar
Callback-basiert	Funktional/komposierbar

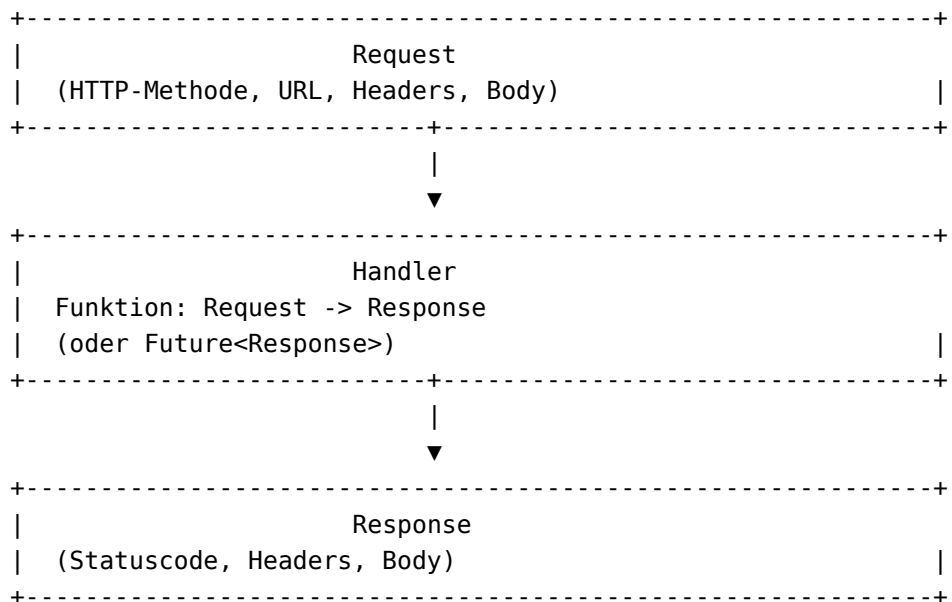
Installation

```
# pubspec.yaml
dependencies:
  shelf: ^1.4.0
  shelf_router: ^1.1.0 # Für Routing (nächste Einheit)

dev_dependencies:
  test: ^1.24.0
```

### 1.2.0.3 Shelf-Architektur

Shelf basiert auf drei Kernkonzepten:



Der Handler

Ein **Handler** ist eine Funktion, die einen **Request** nimmt und eine **Response** (oder **Future<Response>**) zurückgibt:

```
typedef Handler = FutureOr<Response> Function(Request request);
```

Das ist alles! Diese einfache Signatur macht Shelf extrem flexibel und testbar.

### 1.2.0.4 Erster Shelf-Server

Minimales Beispiel

```
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;

void main() async {
  // Handler definieren
  Response handler(Request request) {
    return Response.ok('Hallo von Shelf!');
  }

  // Server starten
  final server = await shelf_io.serve(handler, 'localhost', 8080);
  print('Server läuft auf http://${server.address.host}:${server.port}');
}
```

Handler als Closure

```
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;

void main() async {
  final handler = (Request request) {
    return Response.ok('Pfad: ${request.url.path}');
  };

  await shelf_io.serve(handler, 'localhost', 8080);
}
```

Async Handler

```
Future<Response> handler(Request request) async {
  // Simuliere Datenbankabfrage
  await Future.delayed(Duration(milliseconds: 100));

  return Response.ok('Daten geladen');
}
```

### 1.2.0.5 Request-Objekt

Das Request-Objekt enthält alle Informationen über die eingehende Anfrage.

Request-Eigenschaften

```
Response handler(Request request) {
  // HTTP-Methode
  print('Methode: ${request.method}'); // GET, POST, etc.

  // URL-Informationen
  print('URL: ${request.url}');          // Relative URL
  print('Pfad: ${request.url.path}');    // /api/users
  print('Query: ${request.url.query}');  // name=Max&age=25
}
```

```
print('Requested URL: ${request.requestedUri}'); // Absolute URL

// Headers
print('Content-Type: ${request.headers['content-type']}');
print('Accept: ${request.headers['accept']}');
print('Custom: ${request.headers['x-custom-header']}');

// Alle Headers
request.headers.forEach((key, value) {
  print('$key: $value');
});

return Response.ok('OK');
}
```

Request-Body lesen

```
import 'dart:convert';

Future<Response> handler(Request request) async {
  // Body als String
  final bodyString = await request.readAsString();

  // Body als JSON
  final bodyJson = jsonDecode(bodyString);

  // Oder direkt:
  // final bodyBytes = await request.read().toList();

  return Response.ok('Empfangen: $bodyJson');
}
```

Query-Parameter

```
Response handler(Request request) {
  // URL: /search?q=dart&limit=10
  final params = request.url.queryParameters;

  final query = params['q'] ?? '';
  final limit = int.tryParse(params['limit'] ?? '10') ?? 10;

  return Response.ok('Suche nach: $query (Limit: $limit)');
}
```

Request mit Context

Shelf ermöglicht es, Daten zwischen Middleware und Handlern zu teilen:

```
Response handler(Request request) {
  // Wert aus Context lesen (gesetzt von Middleware)
  final userId = request.context['userId'] as String?;

  return Response.ok('User: $userId');
}
```

```

}

// In Middleware:
Request requestWithContext = request.change(
  context: {'userId': '12345'},
);

```

### 1.2.0.6 Response-Objekt

Das **Response**-Objekt repräsentiert die HTTP-Antwort.

Response erstellen

```

// Einfache Text-Response
Response.ok('Hello World');

// Mit Statuscode
Response(200, body: 'OK');
Response(201, body: 'Created');
Response(204); // No Content

// Vordefinierte Konstruktoren
Response.ok('Success'); // 200
Response.movedPermanently('/new'); // 301
Response.found('/new'); // 302
Response.seeOther('/new'); // 303
Response.notModified(); // 304
Response.notFound('Not Found'); // 404
Response.forbidden('Forbidden'); // 403
Response.internalServerError(); // 500

```

Response mit Headers

```

Response handler(Request request) {
  return Response.ok(
    '{"data": "value"}',
    headers: {
      'content-type': 'application/json',
      'cache-control': 'no-cache',
      'x-custom-header': 'custom-value',
    },
  );
}

```

JSON-Response (Hilfsfunktion)

```

import 'dart:convert';

Response jsonResponse(
  Object? data, {
    int statusCode = 200,
    Map<String, String>? headers,
  }) {

```

```
return Response(  
  statusCode,  
  body: jsonEncode(data),  
  headers: {  
    'content-type': 'application/json; charset=utf-8',  
    ...?headers,  
  },  
);  
}  
  
// Verwendung:  
Response handler(Request request) {  
  return jsonResponse({  
    'users': [  
      {'id': 1, 'name': 'Max'},  
      {'id': 2, 'name': 'Anna'},  
    ],  
  });  
}
```

Response mit Stream-Body

Für große Dateien oder Streaming:

```
import 'dart:io';  
  
Response handler(Request request) {  
  final file = File('large_file.zip');  
  final stream = file.openRead();  
  
  return Response.ok(  
    stream,  
    headers: {  
      'content-type': 'application/zip',  
      'content-length': file.lengthSync().toString(),  
    },  
  );  
}
```

### 1.2.0.7 Pipeline

Die Pipeline ermöglicht das Verketteten von Middleware und Handlern:

```
import 'package:shelf/shelf.dart';  
import 'package:shelf/shelf_io.dart' as shelf_io;  
  
void main() async {  
  // Handler  
  Response handler(Request request) {  
    return Response.ok('Hello!');  
  }  
}
```

```
// Pipeline mit Middleware
final pipeline = Pipeline()
  .addMiddleware(logRequests())    // Built-in Logging
  .addHandler(handler);

await shelf_io.serve(pipeline, 'localhost', 8080);
}
```

Mehrere Middleware verketteten

```
final pipeline = Pipeline()
  .addMiddleware(logRequests())
  .addMiddleware(corsHeaders())
  .addMiddleware(authentication())
  .addHandler(router);
```

Die Middleware wird in der Reihenfolge ausgeführt, in der sie hinzugefügt wird.

### 1.2.0.8 Eingebaute Middleware

Shelf bietet einige nützliche Middleware-Funktionen:

`logRequests()`

Loggt alle eingehenden Requests:

```
import 'package:shelf/shelf.dart';

final pipeline = Pipeline()
  .addMiddleware(logRequests())
  .addHandler(handler);

// Ausgabe:
// 2024-01-15T14:30:00.000000 0:00:00.005000 GET      [200] /api/users
```

`createMiddleware()`

Erstellt einfache Middleware:

```
Middleware corsHeaders() {
  return createMiddleware(
    responseHandler: (response) {
      return response.change(headers: {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE',
        'Access-Control-Allow-Headers': 'Content-Type, Authorization',
      });
    },
  );
}
```

### 1.2.0.9 Server-Konfiguration

`serve()` Optionen

```
import 'package:shelf/shelf_io.dart' as shelf_io;

void main() async {
  final server = await shelf_io.serve(
    handler,
    'localhost', // oder '0.0.0.0' für alle Interfaces
    8080,
    poweredByHeader: 'My Dart Server', // Server-Header
    shared: true, // Shared-Mode für mehrere Isolates
  );

  // Server-Informationen
  print('Adresse: ${server.address.host}');
  print('Port: ${server.port}');
}
```

#### Graceful Shutdown

```
import 'dart:io';
import 'package:shelf/shelf_io.dart' as shelf_io;

HttpServer? _server;

void main() async {
  ProcessSignal.sigint.watch().listen((_) async {
    print('\nShutdown...');
    await _server?.close();
    exit(0);
  });

  _server = await shelf_io.serve(handler, 'localhost', 8080);
  print('Server läuft');
}
```

#### Port aus Umgebungsvariable

```
void main() async {
  final port = int.parse(Platform.environment['PORT'] ?? '8080');

  await shelf_io.serve(handler, '0.0.0.0', port);
  print('Server läuft auf Port $port');
}
```

#### 1.2.0.10 Beispiel: Vollständiger Server

```
import 'dart:convert';
import 'dart:io';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;

HttpServer? _server;
```

```
void main() async {
  // Shutdown-Handler
  ProcessSignal.sigint.watch().listen((_) async {
    print('\n[Shutdown] Server wird beendet...');
    await _server?.close();
    exit(0);
  });

  // Pipeline aufbauen
  final pipeline = Pipeline()
    .addMiddleware(logRequests())
    .addMiddleware(_corsHeaders())
    .addHandler(_router);

  // Server starten
  final port = int.parse(Platform.environment['PORT'] ?? '8080');
  _server = await shelf_io.serve(pipeline, '0.0.0.0', port);

  print('Server läuft auf http://localhost:$port');
}

// Einfaches Routing
Response _router(Request request) {
  final path = request.url.path;
  final method = request.method;

  if (path == '' && method == 'GET') {
    return Response.ok('Willkommen!');
  }

  if (path == 'api/health' && method == 'GET') {
    return _jsonResponse({'status': 'ok'});
  }

  if (path == 'api/echo' && method == 'POST') {
    return _handleEcho(request);
  }

  return Response.notFound('Not Found: $path');
}

Future<Response> _handleEcho(Request request) async {
  final body = await request.readAsString();
  try {
    final json = jsonDecode(body);
    return _jsonResponse({
      'echo': json,
      'timestamp': DateTime.now().toIso8601String(),
    });
  } catch (e) {
```

```
        return Response.badRequest(body: 'Invalid JSON');
    }
}

// Hilfsfunktionen
Response _jsonResponse(Object? data, {int statusCode = 200}) {
    return Response(
        statusCode,
        body: jsonEncode(data),
        headers: {'content-type': 'application/json'},
    );
}

Middleware _corsHeaders() {
    return createMiddleware(
        responseHandler: (response) => response.change(headers: {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE, OPTIONS',
            'Access-Control-Allow-Headers': 'Content-Type, Authorization',
        }),
    );
}
```

#### 1.2.0.11 Zusammenfassung

- **Shelf** ist das Standard-Framework für Dart-Server
- **Handler** sind einfache Funktionen: `Request -> Response`
- **Request** enthält alle Anfrage-Informationen
- **Response** wird über Konstruktoren erstellt
- **Pipeline** verkettet Middleware und Handler
- **logRequests()** ist eine nützliche eingebaute Middleware
- Der Server wird mit `shelf_io.serve()` gestartet

#### 1.2.0.12 Nächste Schritte

In der nächsten Einheit lernst du **shelf\_router** für deklaratives Routing mit URL-Parametern und Route-Gruppen.

### 1.2.1 Übung

#### 1.2.1.1 Ziel

Erstelle einen HTTP-Server mit dem Shelf-Framework, der verschiedene Endpunkte bereitstellt und Middleware verwendet.

#### 1.2.1.2 Aufgabe 1: Projekt-Setup (10 min)

Schritte:

1. Erstelle ein neues Dart-Projekt:

```
dart create -t server-shelf shelf_basics
cd shelf_basics
```

2. Überprüfe die `pubspec.yaml`:

```
dependencies:
  shelf: ^1.4.0
  shelf_router: ^1.1.0
```

3. Ersetze `bin/server.dart` mit einem minimalen Shelf-Server:

```
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;

void main() async {
  Response handler(Request request) {
    return Response.ok('Hello Shelf!');
  }

  final server = await shelf_io.serve(handler, 'localhost', 8080);
  print('Server running on http://${server.address.host}:${server.port}');
}
```

4. Starte und teste:

```
dart run bin/server.dart
curl http://localhost:8080
```

### 1.2.1.3 Aufgabe 2: Request-Handler (20 min)

Implementiere verschiedene Handler für unterschiedliche Endpunkte.

Anforderungen:

Erstelle einen Handler, der basierend auf dem Pfad verschiedene Responses zurückgibt:

Pfad	Methode	Response
/	GET	HTML-Willkommensseite
/api/status	GET	JSON mit Server-Status
/api/headers	GET	JSON mit allen Request-Headers
/api/echo	POST	JSON-Echo des Request-Bodys
Alles andere	*	404 Not Found

Erwartete Responses:

**GET /**

```
<!DOCTYPE html>
<html>
<head><title>Shelf Server</title></head>
<body><h1>Willkommen!</h1></body>
</html>
```

(Content-Type: text/html)

**GET /api/status**

```
{
  "status": "running",
  "version": "1.0.0",
  "timestamp": "2024-01-15T14:30:00.000Z"
}
```

**GET /api/headers**

```
{
  "headers": {
    "host": "localhost:8080",
    "user-agent": "curl/8.0.0",
    "accept": "*/*"
  }
}
```

**POST /api/echo** (mit Body)

```
{
  "method": "POST",
  "path": "/api/echo",
  "body": { ... },
  "timestamp": "2024-01-15T14:30:00.000Z"
}
```

#### 1.2.1.4 Aufgabe 3: JSON-Helper (10 min)

Erstelle eine Hilfsfunktion für JSON-Responses.

Anforderungen:

```
Response jsonResponse(Object? data, {int statusCode = 200});
```

Die Funktion soll: - Den Body als JSON encodieren - Den Content-Type auf `application/json`; `charset=utf-8` setzen - Den angegebenen Statuscode verwenden

Verwendungsbeispiel:

```
return jsonResponse({'status': 'ok'});
return jsonResponse({'error': 'Not found'}, statusCode: 404);
```

#### 1.2.1.5 Aufgabe 4: Custom Middleware (15 min)

Implementiere zwei eigene Middleware-Funktionen.

Middleware 1: Request-Timer

Misst die Verarbeitungszeit jedes Requests und fügt sie als Header hinzu.

```
Middleware requestTimer();
```

Der Response soll einen `X-Response-Time`-Header enthalten:

`X-Response-Time: 5ms`

Middleware 2: API-Key-Prüfung

Prüft, ob ein gültiger API-Key im Header vorhanden ist.

```
Middleware apiKeyAuth(String validApiKey);
```

- Wenn X-API-Key Header fehlt oder falsch: 401 Unauthorized
- Wenn korrekt: Request durchlassen
- Nur für /api/\* Pfade prüfen, / soll ohne Key funktionieren

### 1.2.1.6 Aufgabe 5: Pipeline zusammenbauen (5 min)

Kombiniere alles in einer Pipeline.

Anforderungen:

```
final pipeline = Pipeline()  
    .addMiddleware(logRequests())      // Eingebaut  
    .addMiddleware(requestTimer())     // Aufgabe 4  
    .addMiddleware(apiKeyAuth('secret-key-123')) // Aufgabe 4  
    .addHandler(router);               // Aufgabe 2
```

### 1.2.1.7 Testen

```
# Aufgabe 2: Endpunkte  
curl http://localhost:8080/  
curl http://localhost:8080/api/status  
curl http://localhost:8080/api/headers  
curl -X POST http://localhost:8080/api/echo \  
-H "Content-Type: application/json" \  
-d '{"test": "data"}'  
  
# 404  
curl http://localhost:8080/unknown  
  
# Aufgabe 4: Ohne API-Key (sollte 401 sein)  
curl http://localhost:8080/api/status  
  
# Mit API-Key  
curl -H "X-API-Key: secret-key-123" http://localhost:8080/api/status  
  
# Root ohne API-Key (sollte funktionieren)  
curl http://localhost:8080/  
  
# Response-Time Header prüfen  
curl -v -H "X-API-Key: secret-key-123" http://localhost:8080/api/status
```

### 1.2.1.8 Bonus-Aufgabe: Request-Context

Erweitere die API-Key-Middleware, um bei erfolgreicher Authentifizierung Informationen im Request-Context zu speichern.

Anforderungen:

1. Bei gültigem API-Key: Setze `request.context['authenticated'] = true`

2. Erstelle einen neuen Endpunkt GET `/api/whoami`:

```
{
  "authenticated": true,
  "api_key_prefix": "secre..."
}
```

### 1.2.1.9 Abgabe-Checkliste

- ☐ Server startet auf Port 8080
- ☐ Alle Endpunkte aus Aufgabe 2 funktionieren
- ☐ JSON-Helper-Funktion implementiert
- ☐ Request-Timer-Middleware funktioniert
- ☐ API-Key-Middleware funktioniert
- ☐ Pipeline ist korrekt aufgebaut
- ☐ (Bonus) Request-Context wird genutzt

## 1.2.2 Lösung

### 1.2.2.1 Vollständige Lösung

```
import 'dart:convert';
import 'dart:io';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;

void main() async {
  // Shutdown-Handler
  ProcessSignal.sigint.watch().listen((_) async {
    print('\nServer wird beendet...');
    exit(0);
  });

  // Pipeline aufbauen
  final pipeline = Pipeline()
    .addMiddleware(logRequests())
    .addMiddleware(requestTimer())
    .addMiddleware(apiKeyAuth('secret-key-123'))
    .addHandler(router);

  // Server starten
  final server = await shelf_io.serve(pipeline, 'localhost', 8080);
  print('Server running on http://${server.address.host}:${server.port}');
  print('API-Key für geschützte Routen: secret-key-123');
}

// =====
// Aufgabe 2: Router/Handler
// =====

Future<Response> router(Request request) async {
  final path = request.url.path;
```

```
final method = request.method;

// Root-Route
if (path == '' && method == 'GET') {
    return _handleRoot(request);
}

// API-Routen
if (path == 'api/status' && method == 'GET') {
    return _handleStatus(request);
}

if (path == 'api/headers' && method == 'GET') {
    return _handleHeaders(request);
}

if (path == 'api/echo' && method == 'POST') {
    return await _handleEcho(request);
}

// Bonus: Whoami
if (path == 'api/whoami' && method == 'GET') {
    return _handleWhoami(request);
}

// 404 für alles andere
return jsonResponse(
    {'error': 'Not Found', 'path': '/$path'},
    statusCode: 404,
);
}

Response _handleRoot(Request request) {
    const html = ''
    <!DOCTYPE html>
    <html>
    <head><title>Shelf Server</title></head>
    <body><h1>Willkommen!</h1></body>
    </html>
    '';

    return Response.ok(
        html,
        headers: {'content-type': 'text/html; charset=utf-8'},
    );
}

Response _handleStatus(Request request) {
    return jsonResponse({
        'status': 'running',
        'version': '1.0.0',
    });
}
```

```

        'timestamp': DateTime.now().toUtc().toIso8601String(),
    });
}

Response _handleHeaders(Request request) {
    final headers = <String, String>{};
    request.headers.forEach((key, value) {
        headers[key] = value;
    });

    return jsonResponse({'headers': headers});
}

Future<Response> _handleEcho(Request request) async {
    try {
        final bodyString = await request.readAsString();
        final body = bodyString.isNotEmpty ? jsonDecode(bodyString) : null;

        return jsonResponse({
            'method': request.method,
            'path': '/${request.url.path}',
            'body': body,
            'timestamp': DateTime.now().toUtc().toIso8601String(),
        });
    } catch (e) {
        return jsonResponse(
            {'error': 'Invalid JSON', 'details': e.toString()},
            statusCode: 400,
        );
    }
}

// Bonus: Whoami Handler
Response _handleWhoami(Request request) {
    final authenticated = request.context['authenticated'] as bool? ?? false;
    final apiKeyPrefix = request.context['apiKeyPrefix'] as String? ?? 'none';

    return jsonResponse({
        'authenticated': authenticated,
        'api_key_prefix': apiKeyPrefix,
    });
}

// =====
// Aufgabe 3: JSON-Helper
// =====

Response jsonResponse(Object? data, {int statusCode = 200}) {
    return Response(
        statusCode,
        body: jsonEncode(data),
    );
}

```

```
        headers: {'content-type': 'application/json; charset=utf-8'},
    );
}

// =====
// Aufgabe 4: Custom Middleware
// =====

// Middleware 1: Request-Timer
Middleware requestTimer() {
    return (Handler innerHandler) {
        return (Request request) async {
            final stopwatch = Stopwatch()..start();

            final response = await innerHandler(request);

            stopwatch.stop();

            return response.change(headers: {
                'X-Response-Time': '${stopwatch.elapsedMilliseconds}ms',
            });
        };
    };
}

// Middleware 2: API-Key-Authentifizierung
Middleware apiKeyAuth(String validApiKey) {
    return (Handler innerHandler) {
        return (Request request) async {
            final path = request.url.path;

            // Nur /api/* Pfade schützen
            if (!path.startsWith('api/')) {
                return innerHandler(request);
            }

            // API-Key aus Header lesen
            final apiKey = request.headers['x-api-key'];

            if (apiKey == null) {
                return jsonResponse(
                    {'error': 'Unauthorized', 'message': 'API-Key required'},
                    statusCode: 401,
                );
            }

            if (apiKey != validApiKey) {
                return jsonResponse(
                    {'error': 'Unauthorized', 'message': 'Invalid API-Key'},
                    statusCode: 401,
                );
            }
        };
    };
}
```

```

    }

    // Bonus: Context mit Auth-Infos anreichern
    final enrichedRequest = request.change(context: {
        'authenticated': true,
        'apiKeyPrefix': '${apiKey.substring(0, 4)}...',
    });

    return innerHandler(enrichedRequest);
};
};
}

```

### 1.2.2.2 Erklärungen

#### Aufgabe 2: Router

Der Router ist eine einfache Funktion, die basierend auf Pfad und Methode entscheidet:

```

Future<Response> router(Request request) async {
    final path = request.url.path; // Ohne führenden /
    final method = request.method;

    if (path == '' && method == 'GET') { ... }
    // ...
}

```

**Wichtig:** `request.url.path` enthält den Pfad OHNE führenden Slash! - URL: `http://localhost:8080/api/status` - `request.url.path` -> `api/status` (nicht `/api/status`)

#### Aufgabe 3: JSON-Helper

Die Funktion kapselt die JSON-Erstellung:

```

Response jsonResponse(Object? data, {int statusCode = 200}) {
    return Response(
        statusCode,
        body: jsonEncode(data),
        headers: {'content-type': 'application/json; charset=utf-8'},
    );
}

```

#### Aufgabe 4: Middleware

##### Request-Timer

Die Middleware misst die Zeit:

```

Middleware requestTimer() {
    return (Handler innerHandler) { // Nimmt den nächsten Handler
        return (Request request) async { // Gibt neuen Handler zurück
            final stopwatch = Stopwatch()..start();

            final response = await innerHandler(request); // Handler ausführen
        }
    };
}

```

```

    stopwatch.stop();

    return response.change(headers: {
      'X-Response-Time': '${stopwatch.elapsedMilliseconds}ms',
    });
  };
};
}

```

### API-Key-Auth

Die Auth-Middleware prüft nur API-Routen:

```

// Nur /api/* Pfade schützen
if (!path.startsWith('api/')) {
  return innerHandler(request); // Durchlassen
}

```

Und gibt bei fehlendem/falschem Key eine Error-Response zurück:

```

if (apiKey == null || apiKey != validApiKey) {
  return jsonResponse(..., statusCode: 401); // NICHT innerHandler aufrufen
}

```

### Bonus: Request-Context

Der Context wird über `request.change()` modifiziert:

```

final enrichedRequest = request.change(context: {
  'authenticated': true,
  'apiKeyPrefix': '${apiKey.substring(0, 4)}...',
});

return innerHandler(enrichedRequest); // Weiterreichen

```

Im Handler wird der Context gelesen:

```

final authenticated = request.context['authenticated'] as bool? ?? false;

```

#### 1.2.2.3 Test-Befehle

```

# Server starten
dart run bin/server.dart

# Root (kein API-Key nötig)
curl http://localhost:8080/

# Status ohne API-Key (401)
curl http://localhost:8080/api/status
# {"error": "Unauthorized", "message": "API-Key required"}

# Status mit API-Key
curl -H "X-API-Key: secret-key-123" http://localhost:8080/api/status

```

```
# {"status":"running","version":"1.0.0","timestamp":"..."}

# Headers
curl -H "X-API-Key: secret-key-123" http://localhost:8080/api/headers

# Echo
curl -X POST \
  -H "X-API-Key: secret-key-123" \
  -H "Content-Type: application/json" \
  -d '{"message": "Hello"}' \
  http://localhost:8080/api/echo

# Response-Time Header prüfen (verbose)
curl -v -H "X-API-Key: secret-key-123" http://localhost:8080/api/status
# Suche nach: < X-Response-Time: 2ms

# Whoami (Bonus)
curl -H "X-API-Key: secret-key-123" http://localhost:8080/api/whoami
# {"authenticated":true,"api_key_prefix":"secre..."}

# 404
curl -H "X-API-Key: secret-key-123" http://localhost:8080/api/unknown
# {"error":"Not Found","path":"/api/unknown"}
```

#### 1.2.2.4 Beispiel-Ausgabe

Server-Konsole:

Server running on http://localhost:8080  
API-Key für geschützte Routen: secret-key-123

```
2024-01-15T14:30:00.000000 0:00:00.002000 GET      [200] /
2024-01-15T14:30:05.000000 0:00:00.001000 GET      [401] /api/status
2024-01-15T14:30:10.000000 0:00:00.003000 GET      [200] /api/status
2024-01-15T14:30:15.000000 0:00:00.004000 POST     [200] /api/echo
```

#### 1.2.2.5 Häufige Fehler

1. Pfad mit/ohne Slash

```
// FALSCH
if (path == '/api/status') { ... }

// RICHTIG (request.url.path hat keinen führenden Slash)
if (path == 'api/status') { ... }
```

2. Middleware-Reihenfolge

```
// Logging sollte ZUERST kommen
final pipeline = Pipeline()
  .addMiddleware(logRequests())    // 1. Logging
  .addMiddleware(requestTimer())   // 2. Timer
```

```
.addMiddleware(apiKeyAuth(...))    // 3. Auth
.addHandler(router);
```

### 3. Async vergessen

```
// FALSCH
Response router(Request request) {
  final body = await request.readAsString(); // Error!
}

// RICHTIG
Future<Response> router(Request request) async {
  final body = await request.readAsString();
}
```

### 4. Context nicht weitergeben

```
// FALSCH: Neuer Request ohne Context-Daten
return innerHandler(request);

// RICHTIG: Request mit Context-Daten
final newRequest = request.change(context: {'key': 'value'});
return innerHandler(newRequest);
```

## 1.2.3 Ressourcen

### 1.2.3.1 Offizielle Dokumentation

- Shelf Package - pub.dev
- Shelf API-Dokumentation - API-Referenz
- Shelf GitHub Repository - Quellcode & Beispiele
- Write HTTP servers - Offizielles Tutorial

### 1.2.3.2 Verwandte Packages

Package	Beschreibung	Link
shelf	Core-Framework	pub.dev
shelf_router	Deklaratives Routing	pub.dev
shelf_static	Statische Dateien	pub.dev
shelf_cors_headers	CORS-Middleware	pub.dev
shelf_web_socket	WebSocket-Support	pub.dev

### 1.2.3.3 Code-Beispiele

#### Minimaler Server

```
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as io;

void main() async {
  var handler = (Request request) => Response.ok('Hello!');
```

```
    await io.serve(handler, 'localhost', 8080);  
  }
```

Mit Pipeline und Logging

```
import 'package:shelf/shelf.dart';  
import 'package:shelf/shelf_io.dart' as io;  
  
void main() async {  
  var handler = Pipeline()  
    .addMiddleware(logRequests())  
    .addHandler((req) => Response.ok('Hello!'));  
  
  await io.serve(handler, 'localhost', 8080);  
}
```

JSON-Response Helper

```
import 'dart:convert';  
import 'package:shelf/shelf.dart';  
  
Response json(Object? data, {int status = 200}) => Response(  
  status,  
  body: jsonEncode(data),  
  headers: {'content-type': 'application/json'},  
);
```

#### 1.2.3.4 Cheat Sheet: Request

```
// Methode & URL  
request.method          // GET, POST, etc.  
request.url             // Uri (relativ)  
request.url.path        // /api/users  
request.url.query       // name=Max  
request.requestedUri    // Vollständige Uri  
  
// Query-Parameter  
request.url.queryParameters // Map<String, String>  
request.url.queryParametersAll // Map<String, List<String>>  
  
// Headers  
request.headers['content-type']  
request.headers['authorization']  
  
// Body lesen  
await request.readAsString()  
await request.read().toList() // Stream<List<int>>  
  
// Context (von Middleware gesetzt)  
request.context['userId']
```

```
// Request modifizieren
request.change(context: {'key': 'value'})
```

### 1.2.3.5 Cheat Sheet: Response

```
// Konstruktoren
Response.ok('body') // 200
Response.created('location') // 201
Response.notFound('msg') // 404
Response.forbidden('msg') // 403
Response.internalServerError() // 500
Response(statusCode, body: 'x')

// Mit Headers
Response.ok('body', headers: {'x-custom': 'value'})

// Response modifizieren
response.change(headers: {'new': 'header'})

// Statuscode-Konstanten
200 // OK
201 // Created
204 // No Content
301 // Moved Permanently
302 // Found
400 // Bad Request
401 // Unauthorized
403 // Forbidden
404 // Not Found
500 // Internal Server Error
```

### 1.2.3.6 Cheat Sheet: Pipeline

```
final handler = Pipeline()
  .addMiddleware(logRequests())
  .addMiddleware(customMiddleware())
  .addHandler(myHandler);

await shelf_io.serve(handler, 'localhost', 8080);
```

### 1.2.3.7 Cheat Sheet: Middleware

```
// Einfache Middleware mit createMiddleware
Middleware myMiddleware() {
  return createMiddleware(
    requestHandler: (request) {
      // Vor dem Handler
      // Return Response um Handler zu überspringen
      // Return null um fortzufahren
    }
  );
}
```

```
        return null;
      },
      responseHandler: (response) {
        // Nach dem Handler
        return response.change(headers: {'x-processed': 'true'});
      },
    );
  }

// Manuelle Middleware
Middleware customMiddleware() {
  return (Handler innerHandler) {
    return (Request request) async {
      // Vor Handler
      print('Before: ${request.url}');

      final response = await innerHandler(request);

      // Nach Handler
      print('After: ${response.statusCode}');

      return response;
    };
  };
}
```

### 1.2.3.8 Test-Setup

```
import 'package:shelf/shelf.dart';
import 'package:test/test.dart';

void main() {
  test('handler returns ok', () async {
    final handler = (Request req) => Response.ok('Hello');

    final request = Request('GET', Uri.parse('http://localhost/'));
    final response = await handler(request);

    expect(response.statusCode, 200);
    expect(await response.readAsString(), 'Hello');
  });
}
```

### 1.2.3.9 Weiterführende Links

- [Dart Server Tutorial](#)
- [Shelf Best Practices](#)
- [Building REST APIs with Shelf](#)

## 1.3 Einheit 5.3: Routing mit shelf\_router

### 1.3.0.1 Lernziele

Nach dieser Einheit kannst du: - Das `shelf_router`-Package für deklaratives Routing nutzen  
- URL-Parameter und Query-Parameter extrahieren - Routes gruppieren und organisieren - Verschiedene HTTP-Methoden handhaben

### 1.3.0.2 Warum shelf\_router?

In der letzten Einheit haben wir Routing mit `if-else` implementiert. Das funktioniert, wird aber schnell unübersichtlich:

```
// Ohne shelf_router - wird schnell unübersichtlich
if (path == 'api/users' && method == 'GET') { ... }
if (path == 'api/users' && method == 'POST') { ... }
if (path.startsWith('api/users/') && method == 'GET') {
  final id = path.split('/').last; // Manuelles Parsen
  ...
}
```

**shelf\_router** bietet: - Deklarative Route-Definitionen - Automatisches URL-Parameter-Parsing  
- Methodenspezifische Handler - Route-Gruppen (Mounting)

Installation

```
dependencies:
  shelf: ^1.4.0
  shelf_router: ^1.1.0
```

### 1.3.0.3 Router Basics

Router erstellen

```
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';

final router = Router();

// Routes definieren
router.get('/', (Request request) {
  return Response.ok('Home');
});

router.get('/api/status', (Request request) {
  return Response.ok('OK');
});

// Router als Handler verwenden
final handler = Pipeline()
  .addMiddleware(logRequests())
  .addHandler(router.call);
```

HTTP-Methoden

```
final router = Router();

router.get('/users', _getUsers);
router.post('/users', _createUser);
router.put('/users/<id>', _updateUser);
router.patch('/users/<id>', _patchUser);
router.delete('/users/<id>', _deleteUser);

// Alle Methoden
router.all('/any', (Request request) {
  return Response.ok('Method: ${request.method}');
});
```

#### 1.3.0.4 URL-Parameter

URL-Parameter werden mit `<name>` definiert und automatisch extrahiert.

Einfache Parameter

```
// Route: /users/<id>
// URL:    /users/42
router.get('/users/<id>', (Request request, String id) {
  return Response.ok('User ID: $id');
});

// Mehrere Parameter
// Route: /projects/<projectId>/tasks/<taskId>
// URL:    /projects/abc/tasks/123
router.get('/projects/<projectId>/tasks/<taskId>',
  (Request request, String projectId, String taskId) {
  return Response.ok('Project: $projectId, Task: $taskId');
});
```

Parameter-Typen

Standardmäßig sind Parameter Strings. Konvertierung erfolgt manuell:

```
router.get('/users/<id>', (Request request, String id) async {
  final userId = int.tryParse(id);

  if (userId == null) {
    return Response.badRequest(body: 'Invalid user ID');
  }

  // Benutzer laden...
  return Response.ok('User: $userId');
});
```

Regex-Parameter

Für komplexere Muster:

```
// Nur Zahlen erlauben
router.get('/users/<id|[0-9]+>', (Request request, String id) {
```

```
// id ist garantiert nur Ziffern
return Response.ok('User: $id');
});

// UUID-Format
router.get('/items/<uuid|[a-f0-9-]{36}>', (Request request, String uuid) {
    return Response.ok('Item: $uuid');
});
```

#### Catch-All Parameter

Für Pfade mit beliebiger Tiefe:

```
// Alle Pfade unter /files/ abfangen
router.get('/files/<path|. *>', (Request request, String path) {
    // path = 'documents/2024/report.pdf'
    return Response.ok('File path: $path');
});
```

#### 1.3.0.5 Query-Parameter

Query-Parameter werden über `request.url.queryParameters` ausgelesen:

```
// URL: /search?q=dart&limit=10&sort=name
router.get('/search', (Request request) {
    final params = request.url.queryParameters;

    final query = params['q'] ?? '';
    final limit = int.tryParse(params['limit'] ?? '10') ?? 10;
    final sort = params['sort'] ?? 'id';

    return Response.ok('Search: $query, Limit: $limit, Sort: $sort');
});
```

#### Mehrfache Query-Parameter

```
// URL: /filter?tag=dart&tag=flutter&tag=server
router.get('/filter', (Request request) {
    final allParams = request.url.queryParametersAll;

    // {'tag': ['dart', 'flutter', 'server']}
    final tags = allParams['tag'] ?? [];

    return Response.ok('Tags: ${tags.join(', ')}');
});
```

#### Query-Parameter validieren

```
router.get('/users', (Request request) {
    final params = request.url.queryParameters;

    // Pflichtparameter prüfen
    if (!params.containsKey('status')) {
```

```

        return Response.badRequest(body: 'Missing required parameter: status');
    }

    // Gültige Werte prüfen
    final status = params['status']!;
    if (![ 'active', 'inactive', 'pending' ].contains(status)) {
        return Response.badRequest(body: 'Invalid status value');
    }

    return Response.ok('Status: $status');
});

```

### 1.3.0.6 Route-Gruppen (Mounting)

Für bessere Organisation können Router ineinander gemountet werden:

Sub-Router erstellen

```

// User-Routes
Router userRouter() {
    final router = Router();

    router.get('/', (Request request) {
        return Response.ok('List all users');
    });

    router.get('/:<id>', (Request request, String id) {
        return Response.ok('Get user $id');
    });

    router.post('/', (Request request) async {
        return Response.ok('Create user');
    });

    router.put('/:<id>', (Request request, String id) async {
        return Response.ok('Update user $id');
    });

    router.delete('/:<id>', (Request request, String id) {
        return Response.ok('Delete user $id');
    });

    return router;
}

// Product-Routes
Router productRouter() {
    final router = Router();

    router.get('/', (Request request) => Response.ok('List products'));
    router.get('/:<id>', (Request request, String id) => Response.ok('Product
↩ $id'));

```

```
    return router;
}
```

Router mounten

```
void main() async {
    final app = Router();

    // Sub-Router mounten
    app.mount('/api/users', userRouter().call);
    app.mount('/api/products', productRouter().call);

    // Root-Route
    app.get('/', (Request request) => Response.ok('API Home'));

    // Health-Check
    app.get('/health', (Request request) => Response.ok('OK'));

    final handler = Pipeline()
        .addMiddleware(logRequests())
        .addHandler(app.call);

    await shelf_io.serve(handler, 'localhost', 8080);
}
```

Resultierende Routes

```
GET / -> API Home
GET /health -> OK
GET /api/users -> List all users
GET /api/users/42 -> Get user 42
POST /api/users -> Create user
PUT /api/users/42 -> Update user 42
DELETE /api/users/42 -> Delete user 42
GET /api/products -> List products
GET /api/products/123 -> Product 123
```

### 1.3.0.7 Handler-Klassen

Für größere Projekte empfiehlt sich die Aufteilung in Klassen:

Controller-Pattern

```
class UserController {
    // Abhängigkeiten
    final UserRepository _repository;

    UserController(this._repository);

    // Handler als Router
    Router get router {
        final router = Router();
```

```
router.get('/', _list);
router.get('/:id', _getById);
router.post('/', _create);
router.put('/:id', _update);
router.delete('/:id', _delete);

return router;
}

Future<Response> _list(Request request) async {
  final users = await _repository.findAll();
  return jsonResponse(users);
}

Future<Response> _getById(Request request, String id) async {
  final user = await _repository.findById(id);
  if (user == null) {
    return Response.notFound('User not found');
  }
  return jsonResponse(user);
}

Future<Response> _create(Request request) async {
  final body = await request.readAsString();
  final data = jsonDecode(body);
  final user = await _repository.create(data);
  return jsonResponse(user, statusCode: 201);
}

Future<Response> _update(Request request, String id) async {
  final body = await request.readAsString();
  final data = jsonDecode(body);
  final user = await _repository.update(id, data);
  return jsonResponse(user);
}

Future<Response> _delete(Request request, String id) async {
  await _repository.delete(id);
  return Response(204); // No Content
}
}
```

Controller verwenden

```
void main() async {
  // Abhängigkeiten erstellen
  final userRepo = UserRepository();

  // Controller erstellen
  final userController = UserController(userRepo);
}
```

```
// Router aufbauen
final app = Router();
app.mount('/api/users', userController.router.call);

await shelf_io.serve(app.call, 'localhost', 8080);
}
```

#### 1.3.0.8 404 Handling

Wenn keine Route matcht, gibt shelf\_router automatisch 404 zurück. Du kannst das anpassen:

```
final router = Router();

// Normale Routes
router.get('/api/status', (Request request) => Response.ok('OK'));

// Catch-All für 404
router.all('/<ignored|. *>', (Request request, String ignored) {
  return Response.notFound(
    jsonEncode({
      'error': 'Not Found',
      'path': request.requestedUri.path,
      'method': request.method,
    }),
    headers: {'content-type': 'application/json'},
  );
});
```

#### 1.3.0.9 Beispiel: Vollständige API

```
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';

// In-Memory Datenbank
final _users = <String, Map<String, dynamic>>{};
var _nextId = 1;

void main() async {
  final app = Router();

  // API-Routen
  app.mount('/api/v1', apiRouter().call);

  // Root
  app.get('/', (Request r) => Response.ok('API Server v1.0'));

  // 404
```

```
app.all('/<path|. *>', (Request r, String p) {
    return jsonResponse({'error': 'Not Found'}, statusCode: 404);
});

final handler = Pipeline()
    .addMiddleware(logRequests())
    .addHandler(app.call);

await shelf_io.serve(handler, 'localhost', 8080);
print('Server: http://localhost:8080');
}

Router apiRouter() {
    final router = Router();

    // User CRUD
    router.get('/users', _listUsers);
    router.get('/users/<id>', _getUser);
    router.post('/users', _createUser);
    router.put('/users/<id>', _updateUser);
    router.delete('/users/<id>', _deleteUser);

    // Search
    router.get('/search', _search);

    return router;
}

Response _listUsers(Request request) {
    return jsonResponse({
        'users': _users.values.toList(),
        'count': _users.length,
    });
}

Response _getUser(Request request, String id) {
    final user = _users[id];
    if (user == null) {
        return jsonResponse({'error': 'User not found'}, statusCode: 404);
    }
    return jsonResponse(user);
}

Future<Response> _createUser(Request request) async {
    final body = jsonDecode(await request.readAsString());
    final id = '${_nextId++}';

    _users[id] = {
        'id': id,
        'name': body['name'],
        'email': body['email'],
    }
}
```

```

        'createdAt': DateTime.now().toIso8601String(),
    };

    return jsonResponse(_users[id], statusCode: 201);
}

Future<Response> _updateUser(Request request, String id) async {
    if (!_users.containsKey(id)) {
        return jsonResponse({'error': 'User not found'}, statusCode: 404);
    }

    final body = jsonDecode(await request.readAsString());
    _users[id] = {..._users[id]!, ...body, 'id': id};

    return jsonResponse(_users[id]);
}

Response _deleteUser(Request request, String id) {
    if (!_users.containsKey(id)) {
        return jsonResponse({'error': 'User not found'}, statusCode: 404);
    }

    _users.remove(id);
    return Response(204);
}

Response _search(Request request) {
    final params = request.url.queryParameters;
    final query = params['q']?.toLowerCase() ?? '';

    final results = _users.values.where((user) {
        final name = (user['name'] as String?)?.toLowerCase() ?? '';
        return name.contains(query);
    }).toList();

    return jsonResponse({'results': results, 'query': query});
}

Response jsonResponse(Object? data, {int statusCode = 200}) {
    return Response(
        statusCode,
        body: jsonEncode(data),
        headers: {'content-type': 'application/json'},
    );
}

```

#### 1.3.0.10 Zusammenfassung

- **shelf\_router** bietet deklaratives Routing
- **URL-Parameter** werden mit <name> definiert
- **Regex-Parameter** erlauben Validierung: <id|[0-9]+>

- **Query-Parameter** über `request.url.queryParameters`
- **Route-Gruppen** mit `router.mount('/prefix', subRouter.call)`
- **Controller-Pattern** für bessere Code-Organisation

#### 1.3.0.11 Nächste Schritte

In der nächsten Einheit lernst du **Middleware** im Detail: Logging, CORS, Error-Handling und eigene Middleware-Funktionen.

### 1.3.1 Übung

#### 1.3.1.1 Ziel

Erstelle eine REST API für eine Bücherverwaltung mit `shelf_router`. Du implementierst CRUD-Operationen, URL-Parameter, Query-Parameter und Route-Gruppen.

#### 1.3.1.2 Aufgabe 1: Projekt-Setup (5 min)

1. Erstelle ein neues Projekt oder verwende das vorherige
2. Stelle sicher, dass `shelf_router` installiert ist:

```
dependencies:  
  shelf: ^1.4.0  
  shelf_router: ^1.1.0
```

3. Erstelle die Grundstruktur:

```
import 'dart:convert';  
import 'package:shelf/shelf.dart';  
import 'package:shelf/shelf_io.dart' as shelf_io;  
import 'package:shelf_router/shelf_router.dart';  
  
void main() async {  
  final app = Router();  
  
  // Routes hier...  
  
  await shelf_io.serve(app.call, 'localhost', 8080);  
  print('Server: http://localhost:8080');  
}
```

#### 1.3.1.3 Aufgabe 2: Book CRUD API (25 min)

Implementiere eine vollständige CRUD-API für Bücher.

Datenmodell

Verwende eine In-Memory Map als "Datenbank":

```
final _books = <String, Map<String, dynamic>>{};  
var _nextId = 1;
```

Ein Buch hat folgende Felder: - `id` (String) - `title` (String) - `author` (String) - `year` (int) - `isbn` (String, optional)

## Endpunkte

Methode	Pfad	Beschreibung
GET	/api/books	Alle Bücher auflisten
GET	/api/books/<id>	Ein Buch abrufen
POST	/api/books	Neues Buch erstellen
PUT	/api/books/<id>	Buch aktualisieren
DELETE	/api/books/<id>	Buch löschen

## Erwartete Responses

**GET /api/books**

```
{
  "books": [
    {"id": "1", "title": "Clean Code", "author": "Robert C. Martin", "year": 2008}
  ],
  "total": 1
}
```

**GET /api/books/1**

```
{"id": "1", "title": "Clean Code", "author": "Robert C. Martin", "year": 2008}
```

**POST /api/books** (Request-Body: {"title": "...", "author": "...", "year": 2024}) - Status: 201 Created - Response: Das erstellte Buch mit ID

**PUT /api/books/1** - Status: 200 OK - Response: Das aktualisierte Buch

**DELETE /api/books/1** - Status: 204 No Content - Response: Leer

**Fehlerfall: Buch nicht gefunden** - Status: 404 - Response: {"error": "Book not found"}

**1.3.1.4 Aufgabe 3: Query-Parameter für Filterung (15 min)**

Erweitere den GET /api/books-Endpoint um Filterung.

## Parameter

Parameter	Beschreibung	Beispiel
author	Nach Autor filtern	?author=Martin
year	Nach Jahr filtern	?year=2008
q	Volltextsuche im Titel	?q=clean
limit	Max. Anzahl Ergebnisse	?limit=10
offset	Überspringen (Pagination)	?offset=20

## Beispiele

```
# Alle Bücher von "Martin"
GET /api/books?author=Martin
```

```
# Bücher aus 2020
```

```
GET /api/books?year=2020

# Suche nach "code" im Titel
GET /api/books?q=code

# Pagination: Seite 2 mit 10 Einträgen
GET /api/books?limit=10&offset=10

# Kombiniert
GET /api/books?author=Martin&limit=5
```

Response mit Pagination-Info

```
{
  "books": [...],
  "total": 50,
  "limit": 10,
  "offset": 0,
  "hasMore": true
}
```

#### 1.3.1.5 Aufgabe 4: Route-Gruppen (10 min)

Extrahiere die Book-Routes in einen eigenen Sub-Router.

Anforderungen

1. Erstelle eine Funktion Router `bookRouter()`
2. Mounte sie unter `/api/books`
3. Die Pfade im `bookRouter` sind dann relativ (ohne `/api/books` Prefix)

Struktur

```
Router bookRouter() {
  final router = Router();

  router.get('/', _listBooks);           // GET /api/books
  router.get('/:id', _getBook);          // GET /api/books/42
  router.post('/', _createBook);         // POST /api/books
  // ...

  return router;
}

void main() async {
  final app = Router();

  app.mount('/api/books', bookRouter().call);

  // Weitere Routes...
}
```

**1.3.1.6 Aufgabe 5: Zusätzliche Routes (5 min)**

Füge folgende Endpunkte hinzu:

Health Check

GET /health

Response: {"status": "ok", "timestamp": "..."}

API Info

GET /api

```
Response: {
  "name": "Book API",
  "version": "1.0.0",
  "endpoints": [
    "GET /api/books",
    "GET /api/books/:id",
    "POST /api/books",
    "PUT /api/books/:id",
    "DELETE /api/books/:id"
  ]
}
```

404 Handler

Alle nicht gefundenen Routes sollen ein JSON-Error zurückgeben:

```
{
  "error": "Not Found",
  "path": "/unknown/path",
  "method": "GET"
}
```

**1.3.1.7 Bonus: Nested Resource**

Füge Reviews zu Büchern hinzu.

Endpunkte

Methode	Pfad	Beschreibung
GET	/api/books/<bookId>/reviews	Reviews eines Buchs
POST	/api/books/<bookId>/reviews	Review hinzufügen
DELETE	/api/books/<bookId>/reviews/<reviewId>	Review löschen

Datenmodell

```
final _reviews = <String, List<Map<String, dynamic>>>{};
// Key: bookId, Value: Liste von Reviews
```

Ein Review:

```
{
  "id": "1",
  "bookId": "1",
}
```

```
"rating": 5,  
"comment": "Excellent book!",  
"createdAt": "2024-01-15T..."  
}
```

#### 1.3.1.8 Testen

```
# Books CRUD  
curl http://localhost:8080/api/books  
curl -X POST http://localhost:8080/api/books \  
-H "Content-Type: application/json" \  
-d '{"title": "Clean Code", "author": "Robert C. Martin", "year": 2008}'  
  
curl http://localhost:8080/api/books/1  
curl -X PUT http://localhost:8080/api/books/1 \  
-H "Content-Type: application/json" \  
-d '{"title": "Clean Code (Updated)", "author": "Robert C. Martin", "year":  
↪ 2008}'  
  
curl -X DELETE http://localhost:8080/api/books/1  
  
# Filter  
curl "http://localhost:8080/api/books?author=Martin"  
curl "http://localhost:8080/api/books?q=code&limit=5"  
  
# Health & Info  
curl http://localhost:8080/health  
curl http://localhost:8080/api  
  
# 404  
curl http://localhost:8080/not/found
```

#### 1.3.1.9 Abgabe-Checkliste

- ☐ Alle CRUD-Operationen funktionieren
- ☐ URL-Parameter werden korrekt extrahiert
- ☐ Query-Parameter für Filterung funktionieren
- ☐ Routes sind in Sub-Router organisiert
- ☐ Health und API-Info Endpunkte vorhanden
- ☐ 404-Handler gibt JSON zurück
- ☐ (Bonus) Nested Reviews funktionieren

### 1.3.2 Lösung

#### 1.3.2.1 Vollständige Lösung

```
import 'dart:convert';  
import 'package:shelf/shelf.dart';  
import 'package:shelf/shelf_io.dart' as shelf_io;  
import 'package:shelf_router/shelf_router.dart';
```

```
// =====  
// "Datenbank"  
// =====  
  
final _books = <String, Map<String, dynamic>>{};  
var _bookId = 1;  
  
final _reviews = <String, List<Map<String, dynamic>>>{};  
var _reviewId = 1;  
  
void main() async {  
  // Testdaten einfügen  
  _seedData();  
  
  // Router aufbauen  
  final app = Router();  
  
  // Health & API Info  
  app.get('/health', _healthHandler);  
  app.get('/api', _apiInfoHandler);  
  
  // Book Routes  
  app.mount('/api/books', bookRouter().call);  
  
  // 404 Handler (muss am Ende stehen!)  
  app.all('/<path|.*>', _notFoundHandler);  
  
  // Pipeline mit Logging  
  final handler = Pipeline()  
    .addMiddleware(logRequests())  
    .addHandler(app.call);  
  
  await shelf_io.serve(handler, 'localhost', 8080);  
  print('Server: http://localhost:8080');  
}  
  
void _seedData() {  
  _books['1'] = {  
    'id': '1',  
    'title': 'Clean Code',  
    'author': 'Robert C. Martin',  
    'year': 2008,  
    'isbn': '978-0132350884',  
  };  
  _books['2'] = {  
    'id': '2',  
    'title': 'The Pragmatic Programmer',  
    'author': 'David Thomas',  
    'year': 2019,  
    'isbn': '978-0135957059',  
  };  
}
```

```

};
_bookId = 3;
}

// =====
// Aufgabe 4: Book Router (Sub-Router)
// =====

Router bookRouter() {
    final router = Router();

    // CRUD
    router.get('/', _listBooks);
    router.get('/:id', _getBook);
    router.post('/', _createBook);
    router.put('/:id', _updateBook);
    router.delete('/:id', _deleteBook);

    // Bonus: Reviews
    router.get('/:bookId/reviews', _listReviews);
    router.post('/:bookId/reviews', _createReview);
    router.delete('/:bookId/reviews/:reviewId', _deleteReview);

    return router;
}

// =====
// Aufgabe 2 & 3: Book CRUD mit Filter
// =====

Response _listBooks(Request request) {
    final params = request.url.queryParameters;

    // Filter-Parameter
    final author = params['author']?.toLowerCase();
    final year = int.tryParse(params['year'] ?? '');
    final query = params['q']?.toLowerCase();
    final limit = int.tryParse(params['limit'] ?? '') ?? 100;
    final offset = int.tryParse(params['offset'] ?? '') ?? 0;

    // Filtern
    var results = _books.values.where((book) {
        if (author != null) {
            final bookAuthor = (book['author'] as String).toLowerCase();
            if (!bookAuthor.contains(author)) return false;
        }
        if (year != null && book['year'] != year) return false;
        if (query != null) {
            final title = (book['title'] as String).toLowerCase();
            if (!title.contains(query)) return false;
        }
    })
}

```

```
        return true;
    }).toList();

    final total = results.length;

    // Pagination
    if (offset > 0) {
        results = results.skip(offset).toList();
    }
    if (limit > 0) {
        results = results.take(limit).toList();
    }

    return jsonResponse({
        'books': results,
        'total': total,
        'limit': limit,
        'offset': offset,
        'hasMore': offset + results.length < total,
    });
}

Response _getBook(Request request, String id) {
    final book = _books[id];
    if (book == null) {
        return jsonResponse({'error': 'Book not found'}, statusCode: 404);
    }
    return jsonResponse(book);
}

Future<Response> _createBook(Request request) async {
    try {
        final body = jsonDecode(await request.readAsString());

        // Validierung
        if (body['title'] == null || body['author'] == null) {
            return jsonResponse(
                {'error': 'Missing required fields: title, author'},
                statusCode: 400,
            );
        }

        final id = '${_bookId++}';
        final book = {
            'id': id,
            'title': body['title'],
            'author': body['author'],
            'year': body['year'],
            'isbn': body['isbn'],
        };
    }
```

```

    _books[id] = book;
    return jsonResponse(book, statusCode: 201);
} catch (e) {
    return jsonResponse({'error': 'Invalid JSON'}, statusCode: 400);
}
}

Future<Response> _updateBook(Request request, String id) async {
    if (!_books.containsKey(id)) {
        return jsonResponse({'error': 'Book not found'}, statusCode: 404);
    }

    try {
        final body = jsonDecode(await request.readAsString());

        _books[id] = {
            ..._books[id]!,
            if (body['title'] != null) 'title': body['title'],
            if (body['author'] != null) 'author': body['author'],
            if (body['year'] != null) 'year': body['year'],
            if (body['isbn'] != null) 'isbn': body['isbn'],
            'id': id, // ID nicht überschreiben
        };

        return jsonResponse(_books[id]);
    } catch (e) {
        return jsonResponse({'error': 'Invalid JSON'}, statusCode: 400);
    }
}

Response _deleteBook(Request request, String id) {
    if (!_books.containsKey(id)) {
        return jsonResponse({'error': 'Book not found'}, statusCode: 404);
    }

    _books.remove(id);
    _reviews.remove(id); // Reviews auch löschen
    return Response(204);
}

// =====
// Bonus: Reviews
// =====

Response _listReviews(Request request, String bookId) {
    if (!_books.containsKey(bookId)) {
        return jsonResponse({'error': 'Book not found'}, statusCode: 404);
    }

    final reviews = _reviews[bookId] ?? [];
    return jsonResponse({

```

```

        'bookId': bookId,
        'reviews': reviews,
        'count': reviews.length,
    });
}

Future<Response> _createReview(Request request, String bookId) async {
  if (!_books.containsKey(bookId)) {
    return jsonResponse({'error': 'Book not found'}, statusCode: 404);
  }

  try {
    final body = jsonDecode(await request.readAsString());

    if (body['rating'] == null) {
      return jsonResponse(
        {'error': 'Missing required field: rating'},
        statusCode: 400,
      );
    }

    final rating = body['rating'] as int;
    if (rating < 1 || rating > 5) {
      return jsonResponse(
        {'error': 'Rating must be between 1 and 5'},
        statusCode: 400,
      );
    }

    final id = '${_reviewId++}';
    final review = {
      'id': id,
      'bookId': bookId,
      'rating': rating,
      'comment': body['comment'] ?? '',
      'createdAt': DateTime.now().toUtc().toIso8601String(),
    };

    _reviews.putIfAbsent(bookId, () => []);
    _reviews[bookId]!.add(review);

    return jsonResponse(review, statusCode: 201);
  } catch (e) {
    return jsonResponse({'error': 'Invalid JSON'}, statusCode: 400);
  }
}

Response _deleteReview(Request request, String bookId, String reviewId) {
  if (!_books.containsKey(bookId)) {
    return jsonResponse({'error': 'Book not found'}, statusCode: 404);
  }
}

```

```
final reviews = _reviews[bookId];
if (reviews == null) {
    return jsonResponse({'error': 'Review not found'}, statusCode: 404);
}

final index = reviews.indexWhere((r) => r['id'] == reviewId);
if (index == -1) {
    return jsonResponse({'error': 'Review not found'}, statusCode: 404);
}

reviews.removeAt(index);
return Response(204);
}

// =====
// Aufgabe 5: Health, API Info, 404
// =====

Response _healthHandler(Request request) {
    return jsonResponse({
        'status': 'ok',
        'timestamp': DateTime.now().toUtc().toIso8601String(),
    });
}

Response _apiInfoHandler(Request request) {
    return jsonResponse({
        'name': 'Book API',
        'version': '1.0.0',
        'endpoints': [
            'GET /api/books',
            'GET /api/books/:id',
            'POST /api/books',
            'PUT /api/books/:id',
            'DELETE /api/books/:id',
            'GET /api/books/:bookId/reviews',
            'POST /api/books/:bookId/reviews',
            'DELETE /api/books/:bookId/reviews/:reviewId',
        ],
    });
}

Response _notFoundHandler(Request request, String path) {
    return jsonResponse({
        'error': 'Not Found',
        'path': '/$path',
        'method': request.method,
    }, statusCode: 404);
}
```

```
// =====  
// Helper  
// =====  
  
Response jsonResponse(Object? data, {int statusCode = 200}) {  
  return Response(  
    statusCode,  
    body: jsonEncode(data),  
    headers: {'content-type': 'application/json; charset=utf-8'},  
  );  
}
```

### 1.3.2.2 Test-Befehle

```
# Server starten  
dart run bin/server.dart  
  
# === Books CRUD ===  
  
# Alle Bücher  
curl http://localhost:8080/api/books  
  
# Ein Buch  
curl http://localhost:8080/api/books/1  
  
# Neues Buch  
curl -X POST http://localhost:8080/api/books \  
  -H "Content-Type: application/json" \  
  -d '{"title": "Design Patterns", "author": "Gang of Four", "year": 1994}'  
  
# Buch aktualisieren  
curl -X PUT http://localhost:8080/api/books/1 \  
  -H "Content-Type: application/json" \  
  -d '{"title": "Clean Code (Updated)"}'  
  
# Buch löschen  
curl -X DELETE http://localhost:8080/api/books/3  
  
# === Filter ===  
  
# Nach Autor  
curl "http://localhost:8080/api/books?author=martin"  
  
# Nach Jahr  
curl "http://localhost:8080/api/books?year=2019"  
  
# Volltextsuche  
curl "http://localhost:8080/api/books?q=code"  
  
# Pagination
```

```
curl "http://localhost:8080/api/books?limit=1&offset=0"
curl "http://localhost:8080/api/books?limit=1&offset=1"

# Kombiniert
curl "http://localhost:8080/api/books?author=martin&limit=5"

# === Reviews (Bonus) ===

# Reviews eines Buchs
curl http://localhost:8080/api/books/1/reviews

# Review erstellen
curl -X POST http://localhost:8080/api/books/1/reviews \
  -H "Content-Type: application/json" \
  -d '{"rating": 5, "comment": "Excellent book!"}'

curl -X POST http://localhost:8080/api/books/1/reviews \
  -H "Content-Type: application/json" \
  -d '{"rating": 4, "comment": "Very good"}'

# Reviews auflisten
curl http://localhost:8080/api/books/1/reviews

# Review löschen
curl -X DELETE http://localhost:8080/api/books/1/reviews/1

# === Health & Info ===

curl http://localhost:8080/health
curl http://localhost:8080/api

# === 404 ===

curl http://localhost:8080/not/found
curl -X POST http://localhost:8080/api/unknown

# === Fehler testen ===

# Ungültiges JSON
curl -X POST http://localhost:8080/api/books \
  -H "Content-Type: application/json" \
  -d 'invalid json'

# Pflichtfelder fehlen
curl -X POST http://localhost:8080/api/books \
  -H "Content-Type: application/json" \
  -d '{} '

# Ungültiges Rating
curl -X POST http://localhost:8080/api/books/1/reviews \
  -H "Content-Type: application/json" \
```

```
-d '{"rating": 10}'
```

### 1.3.2.3 Beispiel-Ausgaben

```
$ curl http://localhost:8080/api/books
{
  "books": [
    {"id": "1", "title": "Clean Code", "author": "Robert C. Martin", "year": 2008, "isbn": "978-0132350884"},
    {"id": "2", "title": "The Pragmatic Programmer", "author": "David Thomas", "year": 2019, "isbn": "978-0135957059"}
  ],
  "total": 2,
  "limit": 100,
  "offset": 0,
  "hasMore": false
}

$ curl "http://localhost:8080/api/books?author=martin"
{
  "books": [
    {"id": "1", "title": "Clean Code", "author": "Robert C. Martin", "year": 2008, "isbn": "978-0132350884"}
  ],
  "total": 1,
  "limit": 100,
  "offset": 0,
  "hasMore": false
}

$ curl http://localhost:8080/api/books/99
{"error": "Book not found"}

$ curl http://localhost:8080/health
{"status": "ok", "timestamp": "2024-01-15T14:30:00.000Z"}
```

### 1.3.2.4 Wichtige Erkenntnisse

#### 1. Pfade im Sub-Router

Im `bookRouter()` sind die Pfade RELATIV zum Mount-Point:

```
// Im bookRouter():
router.get('/', ...); // wird zu: /api/books
router.get('/:id', ...); // wird zu: /api/books/42
```

#### 2. 404-Handler Position

Der 404-Handler MUSS am Ende stehen, weil `shelf_router` die erste passende Route nimmt:

```
// RICHTIG
app.get('/health', ...);
```

```
app.mount('/api/books', ...);
app.all('/<path|.*>', _notFound); // Am Ende!

// FALSCH
app.all('/<path|.*>', _notFound); // Fängt ALLES ab!
app.get('/health', ...);          // Wird nie erreicht
```

### 3. Mehrere URL-Parameter

Bei verschachtelten Ressourcen werden alle Parameter übergeben:

```
router.delete('/<bookId>/reviews/<reviewId>',
  (Request request, String bookId, String reviewId) {
    // Beide Parameter verfügbar
  });
```

### 4. Filter-Logik

Die Filter werden mit `where()` kombiniert:

```
var results = _books.values.where((book) {
  if (author != null && !book['author'].contains(author)) return false;
  if (year != null && book['year'] != year) return false;
  return true; // Alle Filter bestanden
});
```

## 1.3.3 Ressourcen

### 1.3.3.1 Offizielle Dokumentation

- shelf\_router Package - pub.dev
- shelf\_router API - API-Referenz
- shelf\_router GitHub - Quellcode

### 1.3.3.2 Cheat Sheet: Router

```
import 'package:shelf_router/shelf_router.dart';

final router = Router();

// HTTP-Methoden
router.get('/path', handler);
router.post('/path', handler);
router.put('/path', handler);
router.patch('/path', handler);
router.delete('/path', handler);
router.head('/path', handler);
router.options('/path', handler);
router.all('/path', handler); // Alle Methoden

// URL-Parameter
router.get('/users/<id>', (Request req, String id) => ...);
router.get('/a/<x>/b/<y>', (Request req, String x, String y) => ...);
```

```
// Regex-Parameter
router.get('/users/<id|[0-9]+>', handler);    // Nur Zahlen
router.get('/files/<path|. *>', handler);    // Catch-all

// Router mounten
router.mount('/api', subRouter.call);

// Als Handler verwenden
final handler = router.call;
// oder: router
```

### 1.3.3.3 Cheat Sheet: Parameter

```
// URL-Parameter
router.get('/users/<id>', (Request request, String id) {
    // /users/42 -> id = "42"
    return Response.ok('User: $id');
});

// Query-Parameter
router.get('/search', (Request request) {
    // /search?q=dart&limit=10
    final params = request.url.queryParameters;
    final q = params['q'];    // "dart"
    final limit = params['limit']; // "10"
    return Response.ok('...');
});

// Mehrfache Query-Parameter
router.get('/filter', (Request request) {
    // /filter?tag=a&tag=b
    final all = request.url.queryParametersAll;
    final tags = all['tag']; // ["a", "b"]
    return Response.ok('...');
});
```

### 1.3.3.4 Cheat Sheet: Route-Gruppen

```
// Sub-Router definieren
Router userRoutes() {
    final r = Router();
    r.get('/', listUsers);    // -> /api/users
    r.get('/<id>', getUser);  // -> /api/users/42
    r.post('/', createUser);  // -> /api/users
    return r;
}

// Mounten
```

```
final app = Router();
app.mount('/api/users', userRoutes().call);
```

### 1.3.3.5 Cheat Sheet: Controller-Pattern

```
class TodoController {
  final TodoRepository _repo;
  TodoController(this._repo);

  Router get router {
    final r = Router();
    r.get('/', list);
    r.get('/:id', getById);
    r.post('/', create);
    r.put('/:id', update);
    r.delete('/:id', delete);
    return r;
  }

  Response list(Request req) => ...;
  Response getById(Request req, String id) => ...;
  Future<Response> create(Request req) async => ...;
  Future<Response> update(Request req, String id) async => ...;
  Response delete(Request req, String id) => ...;
}

// Verwendung
final controller = TodoController(repo);
app.mount('/api/todos', controller.router.call);
```

### 1.3.3.6 Beispiel: CRUD-API

```
final router = Router();

// CREATE
router.post('/items', (Request request) async {
  final body = jsonDecode(await request.readAsString());
  // Item erstellen...
  return Response(201, body: jsonEncode(item));
});

// READ (alle)
router.get('/items', (Request request) {
  // Items laden...
  return Response.ok(jsonEncode(items));
});

// READ (einzeln)
router.get('/items/:id', (Request request, String id) {
```

```

// Item laden...
if (item == null) return Response.notFound('Not found');
return Response.ok(jsonEncode(item));
});

// UPDATE
router.put('/items/<id>', (Request request, String id) async {
  final body = jsonDecode(await request.readAsString());
  // Item aktualisieren...
  return Response.ok(jsonEncode(updated));
});

// DELETE
router.delete('/items/<id>', (Request request, String id) {
  // Item löschen...
  return Response(204); // No Content
});

```

### 1.3.3.7 Best Practices

#### 1. Route-Organisation

```

lib/
+-- routes/
|   +-- api_routes.dart
|   +-- user_routes.dart
|   +-- product_routes.dart
+-- controllers/
|   +-- user_controller.dart
|   +-- product_controller.dart
+-- main.dart

```

#### 2. Konsistente Pfade

```

// GUT: Konsistent, plural, lowercase
/api/users
/api/users/123
/api/products
/api/products/456/reviews

// SCHLECHT: Inkonsistent
/api/User
/api/getUsers
/api/product_list

```

#### 3. Versionierung

```

final app = Router();
app.mount('/api/v1', v1Router().call);
app.mount('/api/v2', v2Router().call);

```

#### 4. 404-Handler

```
// Am Ende des Routers
router.all('/<path|.*>', (Request req, String path) {
    return Response.notFound(jsonEncode({
        'error': 'Not Found',
        'path': '/$path',
    }));
});
```

### 1.3.3.8 Test-Befehle

```
# GET
curl http://localhost:8080/api/users
curl http://localhost:8080/api/users/42

# POST
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Max"}'

# PUT
curl -X PUT http://localhost:8080/api/users/42 \
  -H "Content-Type: application/json" \
  -d '{"name": "Updated"}'

# DELETE
curl -X DELETE http://localhost:8080/api/users/42

# Query-Parameter
curl "http://localhost:8080/api/search?q=test&limit=5"
```

### 1.3.3.9 Weiterführende Links

- RESTful API Design
- HTTP Methods
- URL Design Best Practices

## 1.4 Einheit 5.4: Middleware

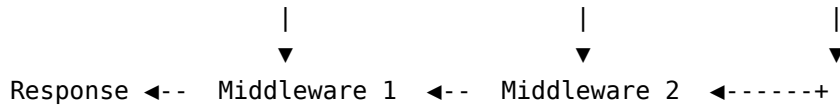
### 1.4.0.1 Lernziele

Nach dieser Einheit kannst du: - Das Middleware-Konzept verstehen und anwenden - Eigene Middleware für Logging, CORS und Auth erstellen - Middleware-Ketten aufbauen - Error-Handling-Middleware implementieren

### 1.4.0.2 Was ist Middleware?

**Middleware** ist Code, der zwischen dem Empfang eines Requests und dem Senden der Response ausgeführt wird. Sie ermöglicht die Verarbeitung von Requests/Responses an einem zentralen Ort.

Request --► Middleware 1 --► Middleware 2 --► Handler



Typische Middleware-Aufgaben

- **Logging:** Requests protokollieren
- **CORS:** Cross-Origin-Headers setzen
- **Authentication:** Tokens validieren
- **Authorization:** Berechtigungen prüfen
- **Compression:** Responses komprimieren
- **Rate Limiting:** Anfragelimit durchsetzen
- **Error Handling:** Fehler abfangen und formatieren

### 1.4.0.3 Middleware-Grundstruktur

In Shelf ist Middleware eine Funktion, die einen Handler nimmt und einen neuen Handler zurückgibt:

```
typedef Middleware = Handler Function(Handler innerHandler);
```

Einfaches Beispiel

```
Middleware simpleMiddleware() {
  return (Handler innerHandler) {
    return (Request request) async {
      // VOR dem Handler
      print('Request: ${request.method} ${request.url}');

      // Handler aufrufen
      final response = await innerHandler(request);

      // NACH dem Handler
      print('Response: ${response.statusCode}');

      return response;
    };
  };
}
```

Middleware verwenden

```
final handler = Pipeline()
  .addMiddleware(simpleMiddleware())
  .addHandler(myHandler);
```

### 1.4.0.4 createMiddleware Helper

Shelf bietet createMiddleware für einfache Fälle:

```
Middleware myMiddleware() {
  return createMiddleware(
    // Wird VOR dem Handler aufgerufen
    requestHandler: (Request request) {
```

```

    // Return Response um Handler zu überspringen
    // Return null um fortzufahren
    return null;
},

// Wird NACH dem Handler aufgerufen
responseHandler: (Response response) {
    // Response modifizieren
    return response.change(headers: {'X-Custom': 'value'});
},

// Fehler abfangen
errorHandler: (error, stackTrace) {
    print('Error: $error');
    return Response.internalServerError();
},
);
}

```

#### 1.4.0.5 Logging Middleware

##### Einfaches Logging

```

Middleware requestLogger() {
    return (Handler innerHandler) {
        return (Request request) async {
            final stopwatch = Stopwatch()..start();
            final timestamp = DateTime.now().toIso8601String();

            // Handler aufrufen
            final response = await innerHandler(request);

            stopwatch.stop();

            // Log-Ausgabe
            print('[${timestamp}] ${request.method.padRight(6)} '
                '${request.requestedUri.path} '
                '- ${response.statusCode} '
                '(${stopwatch.elapsedMilliseconds}ms)');

            return response;
        };
    };
}

```

##### Strukturiertes Logging mit JSON

```

Middleware jsonLogger() {
    return (Handler innerHandler) {
        return (Request request) async {
            final startTime = DateTime.now();
            final stopwatch = Stopwatch()..start();

```

```

    Response response;
    String? error;

    try {
        response = await innerHandler(request);
    } catch (e, stack) {
        error = e.toString();
        response = Response.internalServerError();
    }

    stopwatch.stop();

    final logEntry = {
        'timestamp': startTime.toUtc().toIso8601String(),
        'method': request.method,
        'path': request.requestedUri.path,
        'query': request.requestedUri.query,
        'status': response.statusCode,
        'duration_ms': stopwatch.elapsedMilliseconds,
        'client_ip': request.headers['x-forwarded-for'] ??
            request.headers['x-real-ip'],
        'user_agent': request.headers['user-agent'],
        if (error != null) 'error': error,
    };

    print(jsonEncode(logEntry));

    return response;
};
};
}

```

#### 1.4.0.6 CORS Middleware

Cross-Origin Resource Sharing (CORS) erlaubt Browser-Anfragen von anderen Domains.

Einfache CORS Middleware

```

Middleware corsHeaders({
    String origin = '*',
    List<String> methods = const ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
    List<String> headers = const ['Content-Type', 'Authorization'],
}) {
    return createMiddleware(
        requestHandler: (Request request) {
            // Preflight-Request (OPTIONS) beantworten
            if (request.method == 'OPTIONS') {
                return Response.ok('', headers: {
                    'Access-Control-Allow-Origin': origin,
                    'Access-Control-Allow-Methods': methods.join(', '),
                    'Access-Control-Allow-Headers': headers.join(', '),
                });
            }
        },
    );
}

```

```

        'Access-Control-Max-Age': '86400', // 24 Stunden Cache
    });
}
return null; // Weiter zum Handler
},
responseHandler: (Response response) {
    return response.change(headers: {
        'Access-Control-Allow-Origin': origin,
        'Access-Control-Allow-Methods': methods.join(', '),
        'Access-Control-Allow-Headers': headers.join(', '),
    });
},
);
}

```

Verwendung

```

final handler = Pipeline()
    .addMiddleware(corsHeaders(
        origin: 'https://example.com', // Oder '*' für alle
        methods: ['GET', 'POST'],
        headers: ['Content-Type', 'Authorization', 'X-Custom-Header'],
    ))
    .addHandler(router);

```

### 1.4.0.7 Authentication Middleware

Bearer Token Middleware

```

Middleware bearerAuth(String Function(String token) validateToken) {
    return (Handler innerHandler) {
        return (Request request) async {
            // Public Paths überspringen
            final publicPaths = ['/', '/health', '/api/auth/login'];
            if (publicPaths.contains(request.url.path)) {
                return innerHandler(request);
            }

            // Authorization Header prüfen
            final authHeader = request.headers['authorization'];

            if (authHeader == null) {
                return Response(401,
                    body: jsonEncode({'error': 'Missing Authorization header'}),
                    headers: {'content-type': 'application/json'});
            }

            if (!authHeader.startsWith('Bearer ')) {
                return Response(401,
                    body: jsonEncode({'error': 'Invalid Authorization format'}),
                    headers: {'content-type': 'application/json'});
            }
        }
    }
}

```

```

    final token = authHeader.substring(7); // "Bearer " entfernen

    try {
        final userId = validateToken(token);

        // User-ID in Context speichern
        final enrichedRequest = request.change(context: {
            'userId': userId,
            'authenticated': true,
        });

        return innerHandler(enrichedRequest);
    } catch (e) {
        return Response(401,
            body: jsonEncode({'error': 'Invalid token'}),
            headers: {'content-type': 'application/json'});
    }
};
};
}

```

#### API Key Middleware

```

Middleware apiKeyAuth(Set<String> validKeys) {
    return (Handler innerHandler) {
        return (Request request) async {
            final apiKey = request.headers['x-api-key'];

            if (apiKey == null || !validKeys.contains(apiKey)) {
                return Response(401,
                    body: jsonEncode({'error': 'Invalid or missing API key'}),
                    headers: {'content-type': 'application/json'});
            }

            return innerHandler(request);
        };
    };
}

```

#### 1.4.0.8 Error Handling Middleware

##### Globaler Error Handler

```

Middleware errorHandler() {
    return (Handler innerHandler) {
        return (Request request) async {
            try {
                return await innerHandler(request);
            } on FormatException catch (e) {
                return Response.badRequest(
                    body: jsonEncode({

```

```

        'error': 'Bad Request',
        'message': e.message,
    }},
    headers: {'content-type': 'application/json'},
);
} on NotFoundException catch (e) {
    return Response.notFound(
        jsonEncode({
            'error': 'Not Found',
            'message': e.message,
        })),
        headers: {'content-type': 'application/json'},
    );
} catch (e, stack) {
    // Unerwartete Fehler loggen
    print('Unhandled error: $e\n$stack');

    return Response.internalServerError(
        body: jsonEncode({
            'error': 'Internal Server Error',
            'message': 'An unexpected error occurred',
        })),
        headers: {'content-type': 'application/json'},
    );
}
};
};
}

// Custom Exceptions
class NotFoundException implements Exception {
    final String message;
    NotFoundException(this.message);
}

class UnauthorizedException implements Exception {
    final String message;
    UnauthorizedException(this.message);
}

```

Verwendung mit Custom Exceptions

```

Response getUser(Request request, String id) {
    final user = userRepository.findById(id);
    if (user == null) {
        throw NotFoundException('User $id not found');
    }
    return jsonResponse(user);
}

```

### 1.4.0.9 Request/Response Transformation

#### Request Body Parsing Middleware

```
Middleware jsonBodyParser() {
  return (Handler innerHandler) {
    return (Request request) async {
      if (request.method == 'GET' || request.method == 'DELETE') {
        return innerHandler(request);
      }

      final contentType = request.headers['content-type'];
      if (contentType?.contains('application/json') != true) {
        return innerHandler(request);
      }

      try {
        final body = await request.readAsString();
        final json = body.isNotEmpty ? jsonDecode(body) : {};

        // Parsed JSON in Context speichern
        final enrichedRequest = request.change(context: {
          'body': json,
        });

        return innerHandler(enrichedRequest);
      } catch (e) {
        return Response.badRequest(
          body: jsonEncode({'error': 'Invalid JSON body'}),
          headers: {'content-type': 'application/json'},
        );
      }
    };
  };
}

// Verwendung im Handler
Response createUser(Request request) {
  final body = request.context['body'] as Map<String, dynamic>;
  // body ist bereits geparkt!
}
```

#### Response Compression

```
import 'dart:io';

Middleware gzipCompression() {
  return (Handler innerHandler) {
    return (Request request) async {
      final response = await innerHandler(request);

      // Nur komprimieren wenn Client es akzeptiert
      final acceptEncoding = request.headers['accept-encoding'] ?? '';
    };
  };
}
```

```

    if (!acceptEncoding.contains('gzip')) {
        return response;
    }

    // Body komprimieren
    final body = await response.readAsString();
    final compressed = gzip.encode(utf8.encode(body));

    return Response(
        response.statusCode,
        body: compressed,
        headers: {
            ...response.headers,
            'content-encoding': 'gzip',
            'content-length': compressed.length.toString(),
        },
    );
};
}

```

#### 1.4.0.10 Middleware-Ketten

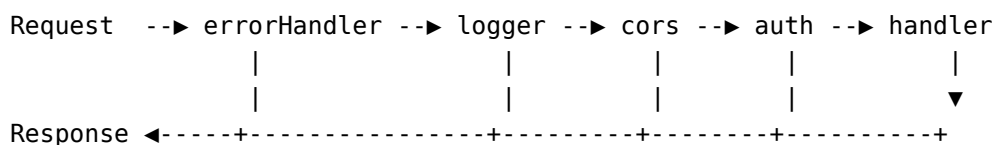
Die Reihenfolge der Middleware ist wichtig:

```

final handler = Pipeline()
  // 1. Error Handler (fängt alle Fehler)
  .addMiddleware(errorHandler())
  // 2. Logging (loggt alle Requests)
  .addMiddleware(requestLogger())
  // 3. CORS (vor Auth, damit OPTIONS funktioniert)
  .addMiddleware(corsHeaders())
  // 4. Authentication
  .addMiddleware(bearerAuth(validateToken))
  // 5. Body Parser
  .addMiddleware(jsonBodyParser())
  // 6. Handler
  .addHandler(router);

```

Ausführungsreihenfolge



#### 1.4.0.11 Bedingte Middleware

Middleware nur für bestimmte Pfade:

```

Middleware conditionalMiddleware(
  Middleware middleware, {

```

```
    required bool Function(Request) when,
  }) {
    return (Handler innerHandler) {
      final wrappedHandler = middleware(innerHandler);

      return (Request request) {
        if (when(request)) {
          return wrappedHandler(request);
        }
        return innerHandler(request);
      };
    };
  }

// Verwendung
final handler = Pipeline()
  .addMiddleware(conditionalMiddleware(
    bearerAuth(validate),
    when: (r) => r.url.path.startsWith('api/'),
  ))
  .addHandler(router);
```

#### 1.4.0.12 Zusammenfassung

- **Middleware** verarbeitet Requests/Responses zentral
- **createMiddleware** für einfache Fälle
- **Error Handling** sollte ganz außen sein
- **CORS** vor Authentication (wegen OPTIONS)
- **Context** zum Teilen von Daten zwischen Middleware und Handlern
- **Reihenfolge** ist entscheidend für korrektes Verhalten

#### 1.4.0.13 Nächste Schritte

In der nächsten Einheit lernst du **Konfiguration & Umgebungsvariablen** für unterschiedliche Environments (Development, Production).

### 1.4.1 Übung

#### 1.4.1.1 Ziel

Implementiere verschiedene Middleware-Komponenten für einen REST API Server: Logging, CORS, Authentication und Error Handling.

#### 1.4.1.2 Aufgabe 1: Logging Middleware (15 min)

Erstelle eine Logging-Middleware, die alle Requests protokolliert.

Anforderungen

Die Middleware soll für jeden Request folgendes ausgeben:

```
[2024-01-15 14:30:00] GET    /api/users    200  12ms
[2024-01-15 14:30:05] POST   /api/users    201   8ms
[2024-01-15 14:30:10] GET    /api/unknown  404   2ms
```

Format: [Timestamp] Methode Pfad Status Dauer

Bonus-Anforderungen

- Methode auf 6 Zeichen padden
- Pfad auf 20 Zeichen padden (oder kürzen mit ...)
- Farbige Ausgabe: Grün für 2xx, Gelb für 4xx, Rot für 5xx

```
Middleware requestLogger();
```

### 1.4.1.3 Aufgabe 2: CORS Middleware (15 min)

Erstelle eine konfigurierbare CORS-Middleware.

Anforderungen

```
Middleware corsMiddleware({
    String origin = '*',
    List<String> methods = const ['GET', 'POST', 'PUT', 'DELETE'],
    List<String> allowedHeaders = const ['Content-Type', 'Authorization'],
    int maxAge = 86400,
});
```

Die Middleware soll:

1. Bei OPTIONS-Requests (Preflight) direkt mit 200 antworten
2. CORS-Headers an alle Responses anhängen:
  - Access-Control-Allow-Origin
  - Access-Control-Allow-Methods
  - Access-Control-Allow-Headers
  - Access-Control-Max-Age

Test

```
# Preflight-Request
curl -X OPTIONS http://localhost:8080/api/test \
  -H "Origin: http://example.com" \
  -H "Access-Control-Request-Method: POST"

# Normaler Request
curl http://localhost:8080/api/test -v

# Prüfe Response-Headers
```

### 1.4.1.4 Aufgabe 3: Rate Limiting Middleware (15 min)

Erstelle eine Rate-Limiting-Middleware, die zu viele Requests ablehnt.

Anforderungen

```
Middleware rateLimiter({
    int maxRequests = 100,
    Duration window = const Duration(minutes: 1),
});
```

Die Middleware soll:

1. Requests pro IP-Adresse zählen

2. Bei Überschreitung des Limits: 429 Too Many Requests
3. Header hinzufügen:
  - X-RateLimit-Limit: Max. Requests
  - X-RateLimit-Remaining: Verbleibende Requests
  - X-RateLimit-Reset: Unix-Timestamp wann Reset

Beispiel-Response bei Limit erreicht

```
{
  "error": "Too Many Requests",
  "message": "Rate limit exceeded. Try again later.",
  "retryAfter": 45
}
```

Hinweis

Verwende eine Map zur Speicherung:

```
final _requests = <String, List<DateTime>>{};
```

#### 1.4.1.5 Aufgabe 4: Authentication Middleware (15 min)

Erstelle eine JWT-ähnliche Authentication-Middleware.

Anforderungen

```
Middleware authMiddleware({
  required String secret,
  List<String> publicPaths = const ['/health', '/api/auth/login'],
});
```

Die Middleware soll:

1. Public Paths ohne Authentifizierung durchlassen
2. Authorization: Bearer <token> Header prüfen
3. Token validieren (vereinfacht: base64(userId:secret) Format)
4. Bei gültigem Token: User-ID in Context speichern
5. Bei ungültigem Token: 401 Unauthorized

Token-Format (vereinfacht)

```
// Token generieren
final token = base64Encode(utf8.encode('user123:$secret'));

// Token validieren
final decoded = utf8.decode(base64Decode(token));
final parts = decoded.split(':');
if (parts[1] == secret) {
  return parts[0]; // userId
}
```

Test

```
# Ohne Token
curl http://localhost:8080/api/users
# 401 Unauthorized
```

```
# Mit Token
TOKEN=$(echo -n "user123:mysecret" | base64)
curl -H "Authorization: Bearer $TOKEN" http://localhost:8080/api/users
# 200 OK

# Public Path
curl http://localhost:8080/health
# 200 OK (ohne Token)
```

#### 1.4.1.6 Aufgabe 5: Error Handler Middleware (10 min)

Erstelle eine Error-Handling-Middleware.

Anforderungen

```
Middleware errorHandler();
```

Die Middleware soll:

1. Alle Exceptions abfangen
2. Verschiedene Exception-Typen unterschiedlich behandeln:
  - NotFoundException -> 404
  - ValidationException -> 400
  - UnauthorizedException -> 401
  - ForbiddenException -> 403
  - Alle anderen -> 500
3. JSON-Error-Response zurückgeben
4. Bei 500: Fehler loggen (aber nicht an Client senden)

Custom Exceptions

```
class ApiException implements Exception {
    final String message;
    final int statusCode;
    ApiException(this.message, this.statusCode);
}

class NotFoundException extends ApiException {
    NotFoundException(String message) : super(message, 404);
}

class ValidationException extends ApiException {
    ValidationException(String message) : super(message, 400);
}
```

Error-Response Format

```
{
  "error": "Not Found",
  "message": "User with ID 42 not found",
  "statusCode": 404
}
```

### 1.4.1.7 Aufgabe 6: Alles zusammenbauen (10 min)

Kombiniere alle Middleware-Komponenten in einer Pipeline.

Anforderungen

```
final handler = Pipeline()
  .addMiddleware(errorHandler())
  .addMiddleware(requestLogger())
  .addMiddleware(corsMiddleware())
  .addMiddleware(rateLimiter(maxRequests: 10, window: Duration(seconds: 30)))
  .addMiddleware(authMiddleware(secret: 'mysecret'))
  .addHandler(router);
```

Erstelle einen einfachen Router mit Test-Endpunkten:

- GET /health - Health Check (public)
- GET /api/users - User-Liste (auth required)
- GET /api/error - Wirft Exception zum Testen

### 1.4.1.8 Testen

```
# Server starten
dart run bin/server.dart

# Health (public)
curl http://localhost:8080/health

# Ohne Auth (401)
curl http://localhost:8080/api/users

# Mit Auth
TOKEN=$(echo -n "user123:mysecret" | base64)
curl -H "Authorization: Bearer $TOKEN" http://localhost:8080/api/users

# CORS Preflight
curl -X OPTIONS http://localhost:8080/api/users -v

# Rate Limit testen (schnell viele Requests)
for i in {1..15}; do curl http://localhost:8080/health; done

# Error testen
curl -H "Authorization: Bearer $TOKEN" http://localhost:8080/api/error
```

### 1.4.1.9 Abgabe-Checkliste

- ☐ Logging-Middleware zeigt alle Requests
- ☐ CORS-Middleware setzt korrekte Header
- ☐ Rate-Limiter blockiert bei zu vielen Requests
- ☐ Auth-Middleware validiert Tokens korrekt
- ☐ Error-Handler fängt Exceptions ab
- ☐ Pipeline ist korrekt aufgebaut

## 1.4.2 Lösung

### 1.4.2.1 Vollständige Lösung

```
import 'dart:convert';
import 'dart:io';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';

void main() async {
  // Router mit Test-Endpunkten
  final router = Router();

  router.get('/health', (Request r) => jsonResponse({'status': 'ok'}));

  router.get('/api/users', (Request r) {
    final userId = r.context['userId'];
    return jsonResponse({
      'users': [
        {'id': 1, 'name': 'Alice'},
        {'id': 2, 'name': 'Bob'},
      ],
      'requestedBy': userId,
    });
  });

  router.get('/api/error', (Request r) {
    throw NotFoundException('This is a test error');
  });

  router.get('/api/validate', (Request r) {
    throw ValidationException('Invalid input data');
  });

  // Pipeline aufbauen
  final handler = Pipeline()
    .addMiddleware(errorHandler())
    .addMiddleware(requestLogger())
    .addMiddleware(corsMiddleware())
    .addMiddleware(rateLimiter(maxRequests: 10, window: Duration(seconds: 30)))
    .addMiddleware(authMiddleware(secret: 'mysecret'))
    .addHandler(router.call);

  await shelf_io.serve(handler, 'localhost', 8080);
  print('Server: http://localhost:8080');
  print('Test-Token: ${base64Encode(utf8.encode('user123:mysecret'))}');
}

// =====
// Aufgabe 1: Logging Middleware
// =====
```

```

Middleware requestLogger() {
    return (Handler innerHandler) {
        return (Request request) async {
            final stopwatch = Stopwatch()..start();
            final timestamp = _formatTimestamp(DateTime.now());

            Response response;
            try {
                response = await innerHandler(request);
            } catch (e) {
                response = Response.internalServerError();
                rethrow;
            } finally {
                stopwatch.stop();

                final method = request.method.padRight(6);
                var path = '/${request.url.path}';
                if (path.length > 20) {
                    path = '${path.substring(0, 17)}...';
                }
                path = path.padRight(20);

                final status = response.statusCode;
                final duration = '${stopwatch.elapsedMilliseconds}ms'.padLeft(6);

                // Farbcodes (ANSI)
                final color = _getStatusColor(status);
                final reset = '\x1B[0m';

                print('[${timestamp}] $method $path $color$status$reset $duration');
            }

            return response;
        };
    };
}

String _formatTimestamp(DateTime dt) {
    return '${dt.year}-${_pad(dt.month)}-${_pad(dt.day)} '
        '${_pad(dt.hour)}:${_pad(dt.minute)}:${_pad(dt.second)}';
}

String _pad(int n) => n.toString().padLeft(2, '0');

String _getStatusColor(int status) {
    if (status >= 200 && status < 300) return '\x1B[32m'; // Grün
    if (status >= 400 && status < 500) return '\x1B[33m'; // Gelb
    if (status >= 500) return '\x1B[31m'; // Rot
    return '\x1B[0m'; // Reset
}

```

```
// =====
// Aufgabe 2: CORS Middleware
// =====

Middleware corsMiddleware({
    String origin = '*',
    List<String> methods = const ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
    List<String> allowedHeaders = const ['Content-Type', 'Authorization'],
    int maxAge = 86400,
}) {
    final corsHeaders = {
        'Access-Control-Allow-Origin': origin,
        'Access-Control-Allow-Methods': methods.join(', '),
        'Access-Control-Allow-Headers': allowedHeaders.join(', '),
        'Access-Control-Max-Age': maxAge.toString(),
    };

    return createMiddleware(
        requestHandler: (Request request) {
            // Preflight-Request (OPTIONS) direkt beantworten
            if (request.method == 'OPTIONS') {
                return Response.ok('', headers: corsHeaders);
            }
            return null;
        },
        responseHandler: (Response response) {
            // CORS-Headers an alle Responses anhängen
            return response.change(headers: corsHeaders);
        },
    );
}

// =====
// Aufgabe 3: Rate Limiting Middleware
// =====

final _requestCounts = <String, List<DateTime>>{};

Middleware rateLimiter({
    int maxRequests = 100,
    Duration window = const Duration(minutes: 1),
}) {
    return (Handler innerHandler) {
        return (Request request) async {
            // Client-IP ermitteln
            final clientId = request.headers['x-forwarded-for'] ??
                request.headers['x-real-ip'] ??
                'unknown';

            final now = DateTime.now();
```

```

    final windowStart = now.subtract(window);

    // Alte Einträge entfernen
    _requestCounts.putIfAbsent(clientIp, () => []);
    _requestCounts[clientIp]!.removeWhere((t) => t.isBefore(windowStart));

    final currentCount = _requestCounts[clientIp]!.length;
    final remaining = maxRequests - currentCount - 1;
    final resetTime = now.add(window).millisecondsSinceEpoch ~/ 1000;

    // Rate-Limit Headers
    final rateLimitHeaders = {
      'X-RateLimit-Limit': maxRequests.toString(),
      'X-RateLimit-Remaining': remaining.clamp(0, maxRequests).toString(),
      'X-RateLimit-Reset': resetTime.toString(),
    };

    // Limit erreicht?
    if (currentCount >= maxRequests) {
      final retryAfter = window.inSeconds -
        now.difference(_requestCounts[clientIp]!.first).inSeconds;

      return Response(
        429,
        body: jsonEncode({
          'error': 'Too Many Requests',
          'message': 'Rate limit exceeded. Try again later.',
          'retryAfter': retryAfter,
        }),
        headers: {
          'content-type': 'application/json',
          'Retry-After': retryAfter.toString(),
          ...rateLimitHeaders,
        },
      );
    }

    // Request zählen
    _requestCounts[clientIp]!.add(now);

    // Handler aufrufen und Headers hinzufügen
    final response = await innerHandler(request);
    return response.change(headers: rateLimitHeaders);
  };
};
}

// =====
// Aufgabe 4: Authentication Middleware
// =====

```

```
Middleware authMiddleware({
  required String secret,
  List<String> publicPaths = const ['/health', '/api/auth/login'],
}) {
  return (Handler innerHandler) {
    return (Request request) async {
      final path = '${request.url.path}';

      // Public Paths durchlassen
      if (publicPaths.any((p) => path == p || path.startsWith('$p/'))) {
        return innerHandler(request);
      }

      // Authorization Header prüfen
      final authHeader = request.headers['authorization'];

      if (authHeader == null) {
        return Response(
          401,
          body: jsonEncode({
            'error': 'Unauthorized',
            'message': 'Missing Authorization header',
          }),
          headers: {'content-type': 'application/json'},
        );
      }

      if (!authHeader.startsWith('Bearer ')) {
        return Response(
          401,
          body: jsonEncode({
            'error': 'Unauthorized',
            'message': 'Invalid Authorization format. Use: Bearer <token>',
          }),
          headers: {'content-type': 'application/json'},
        );
      }

      final token = authHeader.substring(7);

      try {
        // Token validieren (vereinfacht)
        final decoded = utf8.decode(base64Decode(token));
        final parts = decoded.split(':');

        if (parts.length != 2 || parts[1] != secret) {
          throw FormatException('Invalid token');
        }

        final userId = parts[0];
      }
    };
  };
}
```

```

        // User-ID in Context speichern
        final enrichedRequest = request.change(context: {
            'userId': userId,
            'authenticated': true,
        });

        return innerHandler(enrichedRequest);
    } catch (e) {
        return Response(
            401,
            body: jsonEncode({
                'error': 'Unauthorized',
                'message': 'Invalid or expired token',
            }),
            headers: {'content-type': 'application/json'},
        );
    }
};
};
}

// =====
// Aufgabe 5: Error Handler Middleware
// =====

Middleware errorHandler() {
    return (Handler innerHandler) {
        return (Request request) async {
            try {
                return await innerHandler(request);
            } on ApiException catch (e) {
                return Response(
                    e.statusCode,
                    body: jsonEncode({
                        'error': _getErrorName(e.statusCode),
                        'message': e.message,
                        'statusCode': e.statusCode,
                    }),
                    headers: {'content-type': 'application/json'},
                );
            } catch (e, stack) {
                // Unerwartete Fehler loggen (nicht an Client senden)
                print('Unhandled error: $e');
                print('Stack trace: $stack');

                return Response(
                    500,
                    body: jsonEncode({
                        'error': 'Internal Server Error',
                        'message': 'An unexpected error occurred',
                        'statusCode': 500,
                    }),
                    headers: {'content-type': 'application/json'},
                );
            }
        };
    };
}

```

```
        }},
        headers: {'content-type': 'application/json'},
    );
}
};
};
}

String _getErrorName(int statusCode) {
    switch (statusCode) {
        case 400:
            return 'Bad Request';
        case 401:
            return 'Unauthorized';
        case 403:
            return 'Forbidden';
        case 404:
            return 'Not Found';
        case 429:
            return 'Too Many Requests';
        default:
            return 'Error';
    }
}

// =====
// Custom Exceptions
// =====

class ApiException implements Exception {
    final String message;
    final int statusCode;
    ApiException(this.message, this.statusCode);
}

class NotFoundException extends ApiException {
    NotFoundException(String message) : super(message, 404);
}

class ValidationException extends ApiException {
    ValidationException(String message) : super(message, 400);
}

class UnauthorizedException extends ApiException {
    UnauthorizedException(String message) : super(message, 401);
}

class ForbiddenException extends ApiException {
    ForbiddenException(String message) : super(message, 403);
}
```

```
// =====  
// Helper  
// =====  
  
Response jsonResponse(Object? data, {int statusCode = 200}) {  
  return Response(  
    statusCode,  
    body: jsonEncode(data),  
    headers: {'content-type': 'application/json'},  
  );  
}
```

#### 1.4.2.2 Test-Befehle

```
# Server starten  
dart run bin/server.dart  
  
# Token generieren  
TOKEN=$(echo -n "user123:mysecret" | base64)  
echo "Token: $TOKEN"  
  
# === Health (public, keine Auth) ===  
curl http://localhost:8080/health  
  
# === Ohne Auth (401) ===  
curl http://localhost:8080/api/users  
  
# === Mit Auth (200) ===  
curl -H "Authorization: Bearer $TOKEN" http://localhost:8080/api/users  
  
# === CORS Preflight ===  
curl -X OPTIONS http://localhost:8080/api/users \  
  -H "Origin: http://example.com" \  
  -H "Access-Control-Request-Method: POST" -v  
  
# === Rate Limit testen ===  
for i in {1..12}; do  
  echo "Request $i:"  
  curl -s http://localhost:8080/health | head -1  
done  
  
# === Error Handler testen ===  
curl -H "Authorization: Bearer $TOKEN" http://localhost:8080/api/error  
curl -H "Authorization: Bearer $TOKEN" http://localhost:8080/api/validate  
  
# === Ungültiger Token ===  
curl -H "Authorization: Bearer invalid" http://localhost:8080/api/users
```

### 1.4.2.3 Beispiel-Ausgaben

Server-Konsole

Server: http://localhost:8080

Test-Token: dXNlcjEyMzpteXNlY3JldA==

[2024-01-15 14:30:00]	GET	/health	200	2ms
[2024-01-15 14:30:01]	GET	/api/users	401	1ms
[2024-01-15 14:30:02]	GET	/api/users	200	3ms
[2024-01-15 14:30:03]	OPTIONS	/api/users	200	1ms
[2024-01-15 14:30:04]	GET	/api/error	404	2ms

Response: /api/users (mit Auth)

```
{
  "users": [
    {"id": 1, "name": "Alice"},
    {"id": 2, "name": "Bob"}
  ],
  "requestedBy": "user123"
}
```

Response: Rate Limit erreicht

```
{
  "error": "Too Many Requests",
  "message": "Rate limit exceeded. Try again later.",
  "retryAfter": 25
}
```

Response: Error Handler

```
{
  "error": "Not Found",
  "message": "This is a test error",
  "statusCode": 404
}
```

### 1.4.2.4 Wichtige Erkenntnisse

#### 1. Middleware-Reihenfolge

```
Pipeline()
  .addMiddleware(errorHandler()) // 1. Ganz außen
  .addMiddleware(requestLogger()) // 2. Logging für alle
  .addMiddleware(corsMiddleware()) // 3. Vor Auth (OPTIONS!)
  .addMiddleware(rateLimiter()) // 4. Vor Auth
  .addMiddleware(authMiddleware()) // 5. Authentication
  .addHandler(router);
```

#### 2. Error Handler Position

Der Error Handler muss ganz außen sein, damit er Fehler aller anderen Middleware abfängt.

#### 3. CORS vor Auth

OPTIONS-Requests (Preflight) haben keinen Authorization-Header. Daher muss CORS vor Auth kommen.

#### 4. Context für Datenübertragung

```
// In Middleware
final newRequest = request.change(context: {'userId': userId});

// Im Handler
final userId = request.context['userId'];
```

#### 5. createMiddleware vs. manuelle Middleware

- **createMiddleware:** Für einfache Fälle (nur Response modifizieren)
- **Manuelle Middleware:** Für komplexe Logik (Rate Limiting, Auth)

### 1.4.3 Ressourcen

#### 1.4.3.1 Offizielle Dokumentation

- Shelf Middleware - API-Referenz
- createMiddleware - Helper-Funktion

#### 1.4.3.2 Nützliche Middleware-Packages

Package	Beschreibung	Link
shelf_cors_headers	CORS-Middleware	pub.dev
shelf_helmet	Security Headers	pub.dev
shelf_rate_limiter	Rate Limiting	pub.dev
shelf_static	Statische Dateien	pub.dev

#### 1.4.3.3 Cheat Sheet: Middleware-Struktur

```
// Vollständige Middleware-Signatur
Middleware myMiddleware() {
  return (Handler innerHandler) {
    return (Request request) async {
      // 1. VOR Handler (Request bearbeiten)

      // 2. Handler aufrufen
      final response = await innerHandler(request);

      // 3. NACH Handler (Response bearbeiten)

      return response;
    };
  };
}
```

**1.4.3.4 Cheat Sheet: createMiddleware**

```
Middleware simple() {
  return createMiddleware(
    // Request abfangen (return Response oder null)
    requestHandler: (Request request) => null,

    // Response modifizieren
    responseHandler: (Response response) => response,

    // Fehler behandeln
    errorHandler: (error, stack) => Response.internalServerError(),
  );
}
```

**1.4.3.5 Cheat Sheet: Häufige Patterns****Logging**

```
Middleware logger() => (Handler h) => (Request r) async {
  final sw = Stopwatch()..start();
  final res = await h(r);
  print('${r.method} ${r.url} - ${res.statusCode}
  ↪   (${sw.elapsedMilliseconds}ms)');
  return res;
};
```

**CORS**

```
Middleware cors() => createMiddleware(
  requestHandler: (r) => r.method == 'OPTIONS'
    ? Response.ok('', headers: {'Access-Control-Allow-Origin': '*'})
    : null,
  responseHandler: (r) => r.change(headers: {
    'Access-Control-Allow-Origin': '*',
  }),
);
```

**Auth**

```
Middleware auth(String Function(String) validate) => (Handler h) => (Request ↪
  ↪ r) async {
  final token = r.headers['authorization']?.replaceFirst('Bearer ', '');
  if (token == null) return Response(401);
  try {
    final userId = validate(token);
    return h(r.change(context: {'userId': userId}));
  } catch (e) {
    return Response(401);
  }
};
```

**Error Handler**

```
Middleware errors() => (Handler h) => (Request r) async {
    try {
        return await h(r);
    } catch (e, s) {
        print('Error: $e\n$s');
        return Response.internalServerError();
    }
};
```

#### Request Timer

```
Middleware timer() => (Handler h) => (Request r) async {
    final sw = Stopwatch()..start();
    final res = await h(r);
    return res.change(headers: {'X-Response-Time': '${sw.elapsedMilliseconds}ms'});
};
```

#### 1.4.3.6 Pipeline-Reihenfolge

```
Pipeline()
    .addMiddleware(errorHandler()) // 1. Außen: Fehler fangen
    .addMiddleware(logger())       // 2. Logging
    .addMiddleware(cors())         // 3. CORS (vor Auth!)
    .addMiddleware(auth())         // 4. Authentication
    .addMiddleware(bodyParser())   // 5. Body parsen
    .addHandler(router);           // 6. Handler
```

#### 1.4.3.7 Context verwenden

```
// In Middleware setzen
final newRequest = request.change(context: {
    'userId': '123',
    'roles': ['admin', 'user'],
});

// Im Handler lesen
final userId = request.context['userId'] as String?;
final roles = request.context['roles'] as List<String>;
```

#### 1.4.3.8 Bedingte Middleware

```
Middleware onlyFor(String pathPrefix, Middleware m) {
    return (Handler h) {
        final wrapped = m(h);
        return (Request r) {
            if (r.url.path.startsWith(pathPrefix)) {
                return wrapped(r);
            }
            return h(r);
        };
    };
}
```

```

    };
  };
}

// Verwendung
.addMiddleware(onlyFor('api/', authMiddleware()))

```

#### 1.4.3.9 Test-Middleware

```

import 'package:test/test.dart';

void main() {
  test('auth middleware rejects missing token', () async {
    final middleware = authMiddleware();
    final handler = middleware((r) => Response.ok('OK'));

    final request = Request('GET', Uri.parse('http://localhost/api/data'));
    final response = await handler(request);

    expect(response.statusCode, 401);
  });
}

```

## 1.5 Einheit 5.5: Konfiguration & Environment

### 1.5.0.1 Lernziele

Nach dieser Einheit kannst du: - Umgebungsvariablen in Dart lesen - **.env**-Dateien für lokale Entwicklung nutzen - Konfigurationsklassen erstellen - Verschiedene Umgebungen (dev/staging/prod) verwalten

### 1.5.0.2 Warum Konfiguration?

Anwendungen brauchen konfigurierbare Werte: - **Datenbankverbindungen** (Host, Port, Credentials) - **API-Keys** für externe Dienste - **Server-Port** und -Host - **Feature Flags** - **Logging-Level**

Diese Werte sollten NICHT im Code stehen: - Sicherheitsrisiko (Secrets im Git-Repository) - Keine Flexibilität (Code-Änderung für neue Umgebung) - Schlechte Wartbarkeit

### 1.5.0.3 Umgebungsvariablen in Dart

Lesen mit Platform.environment

```

import 'dart:io';

void main() {
  // Einzelne Variable lesen
  final port = Platform.environment['PORT'];
  print('PORT: $port');
}

```

```
// Mit Default-Wert
final host = Platform.environment['HOST'] ?? 'localhost';
print('HOST: $host');

// Alle Umgebungsvariablen
Platform.environment.forEach((key, value) {
  print('$key: $value');
});
}
```

Variablen setzen

```
# Einzelne Variable
export PORT=8080
dart run bin/server.dart

# Inline
PORT=8080 HOST=0.0.0.0 dart run bin/server.dart

# Mehrere Variablen
export DATABASE_URL=postgres://localhost:5432/mydb
export JWT_SECRET=supersecret
dart run bin/server.dart
```

#### 1.5.0.4 .env-Dateien

Für lokale Entwicklung sind .env-Dateien praktisch. Sie werden NICHT ins Git-Repository committed.

.env Datei Format

```
# .env
PORT=8080
HOST=localhost

# Datenbank
DATABASE_URL=postgres://user:pass@localhost:5432/mydb
DATABASE_POOL_SIZE=10

# Auth
JWT_SECRET=my-super-secret-key
JWT_EXPIRY=3600

# Externe APIs
STRIPE_API_KEY=sk_test_abc123
SENDGRID_API_KEY=SG.xyz789
```

.env laden mit dotenv Package

```
# pubspec.yaml
dependencies:
  dotenv: ^4.2.0
```

```
import 'package:dotenv/dotenv.dart' as dotenv;
import 'dart:io';

void main() {
  // .env laden (nur in Development!)
  dotenv.load();

  // Jetzt sind Variablen verfügbar
  final port = dotenv.env['PORT'] ?? Platform.environment['PORT'] ?? '8080';

  print('Port: $port');
}
```

Manuelle .env-Implementierung

Ohne externes Package:

```
import 'dart:io';

Future<Map<String, String>> loadEnvFile([String path = '.env']) async {
  final file = File(path);
  final env = <String, String>{};

  if (!await file.exists()) {
    return env;
  }

  final lines = await file.readAsLines();

  for (final line in lines) {
    // Kommentare und leere Zeilen überspringen
    final trimmed = line.trim();
    if (trimmed.isEmpty || trimmed.startsWith('#')) continue;

    // KEY=VALUE parsen
    final index = trimmed.indexOf('=');
    if (index == -1) continue;

    final key = trimmed.substring(0, index).trim();
    var value = trimmed.substring(index + 1).trim();

    // Quotes entfernen
    if ((value.startsWith('"') && value.endsWith('"')) ||
        (value.startsWith("'") && value.endsWith("'"))) {
      value = value.substring(1, value.length - 1);
    }

    env[key] = value;
  }

  return env;
}
```

### 1.5.0.5 Konfigurationsklassen

Statt überall `Platform.environment['KEY']` zu schreiben, kapsle die Konfiguration in einer Klasse.

Einfache Config-Klasse

```
class Config {
    final String host;
    final int port;
    final String databaseUrl;
    final String jwtSecret;
    final int jwtExpiry;
    final String environment;

    Config({
        required this.host,
        required this.port,
        required this.databaseUrl,
        required this.jwtSecret,
        this.jwtExpiry = 3600,
        this.environment = 'development',
    });

    /// Lädt Konfiguration aus Umgebungsvariablen
    factory Config.fromEnvironment() {
        return Config(
            host: _env('HOST', 'localhost'),
            port: int.parse(_env('PORT', '8080')),
            databaseUrl: _envRequired('DATABASE_URL'),
            jwtSecret: _envRequired('JWT_SECRET'),
            jwtExpiry: int.parse(_env('JWT_EXPIRY', '3600')),
            environment: _env('ENVIRONMENT', 'development'),
        );
    }

    bool get isDevelopment => environment == 'development';
    bool get isProduction => environment == 'production';
    bool get isStaging => environment == 'staging';

    static String _env(String key, String defaultValue) {
        return Platform.environment[key] ?? defaultValue;
    }

    static String _envRequired(String key) {
        final value = Platform.environment[key];
        if (value == null || value.isEmpty) {
            throw Exception('Required environment variable $key is not set');
        }
        return value;
    }
}
```

```
}
```

Verwendung

```
void main() async {
  final config = Config.fromEnvironment();

  print('Starting server...');
  print('Environment: ${config.environment}');
  print('Host: ${config.host}:${config.port}');

  if (config.isDevelopment) {
    print('Development mode - verbose logging enabled');
  }

  await shelf_io.serve(handler, config.host, config.port);
}
```

### 1.5.0.6 Umgebungsspezifische Konfiguration

Verschiedene .env-Dateien

```
project/
+-- .env                # Lokale Entwicklung (nicht im Git)
+-- .env.example        # Template (im Git)
+-- .env.staging        # Staging-Umgebung
+-- .env.production     # Produktion
```

.env.example

```
# .env.example - Kopiere zu .env und fülle aus
PORT=8080
HOST=localhost
DATABASE_URL=postgres://user:password@localhost:5432/dbname
JWT_SECRET=your-secret-here
ENVIRONMENT=development
```

.gitignore

```
# Umgebungsvariablen
.env
.env.local
.env.*.local

# Behalte das Example
!.env.example
```

Environment-Laden

```
Future<void> loadEnvironment() async {
  final env = Platform.environment['ENVIRONMENT'] ?? 'development';

  // Versuche umgebungsspezifische Datei zu laden
  final envFile = File('.env.$env');
```

```

if (await envFile.exists()) {
  await _loadEnvFile('.env.$env');
}

// Dann die Standard .env (überschreibt nicht)
await _loadEnvFile('.env');
}

```

### 1.5.0.7 Secrets sicher verwalten

Niemals in Git

```

# .gitignore
.env
*.pem
*.key
secrets/

```

In Produktion

Produktions-Secrets sollten über die Plattform injiziert werden:

```

# Docker
docker run -e JWT_SECRET=secret -e DATABASE_URL=... myapp

# Kubernetes
kubectl create secret generic app-secrets \
  --from-literal=JWT_SECRET=secret \
  --from-literal=DATABASE_URL=...

# Cloud (z.B. Railway, Fly.io, Heroku)
# Über Web-Interface oder CLI setzen

```

Secret-Validation

```

class Config {
  // ...

  void validate() {
    final errors = <String>[];

    if (jwtSecret.length < 32) {
      errors.add('JWT_SECRET should be at least 32 characters');
    }

    if (!databaseUrl.startsWith('postgres://')) {
      errors.add('DATABASE_URL must be a valid PostgreSQL URL');
    }

    if (isProduction && jwtSecret == 'development-secret') {
      errors.add('Cannot use development secret in production!');
    }
  }
}

```

```
        if (errors.isNotEmpty) {
            throw ConfigurationException(errors.join('\n'));
        }
    }
}

class ConfigurationException implements Exception {
    final String message;
    ConfigurationException(this.message);

    @override
    String toString() => 'ConfigurationException: $message';
}
```

### 1.5.0.8 Erweiterte Config-Klasse

Mit Sub-Konfigurationen

```
class Config {
    final ServerConfig server;
    final DatabaseConfig database;
    final AuthConfig auth;
    final String environment;

    Config({
        required this.server,
        required this.database,
        required this.auth,
        required this.environment,
    });

    factory Config.fromEnvironment() {
        return Config(
            server: ServerConfig.fromEnvironment(),
            database: DatabaseConfig.fromEnvironment(),
            auth: AuthConfig.fromEnvironment(),
            environment: _env('ENVIRONMENT', 'development'),
        );
    }

    static String _env(String key, [String? defaultValue]) {
        return Platform.environment[key] ?? defaultValue ?? '';
    }
}

class ServerConfig {
    final String host;
    final int port;

    ServerConfig({required this.host, required this.port});
}
```

```
factory ServerConfig.fromEnvironment() {
  return ServerConfig(
    host: Platform.environment['HOST'] ?? 'localhost',
    port: int.parse(Platform.environment['PORT'] ?? '8080'),
  );
}

class DatabaseConfig {
  final String url;
  final int poolSize;
  final Duration connectionTimeout;

  DatabaseConfig({
    required this.url,
    this.poolSize = 10,
    this.connectionTimeout = const Duration(seconds: 30),
  });

  factory DatabaseConfig.fromEnvironment() {
    return DatabaseConfig(
      url: Platform.environment['DATABASE_URL'] ?? '',
      poolSize: int.parse(Platform.environment['DB_POOL_SIZE'] ?? '10'),
      connectionTimeout: Duration(
        seconds: int.parse(Platform.environment['DB_TIMEOUT'] ?? '30'),
      ),
    );
  }
}

class AuthConfig {
  final String jwtSecret;
  final Duration tokenExpiry;
  final Duration refreshExpiry;

  AuthConfig({
    required this.jwtSecret,
    this.tokenExpiry = const Duration(hours: 1),
    this.refreshExpiry = const Duration(days: 7),
  });

  factory AuthConfig.fromEnvironment() {
    return AuthConfig(
      jwtSecret: Platform.environment['JWT_SECRET'] ?? '',
      tokenExpiry: Duration(
        seconds: int.parse(Platform.environment['JWT_EXPIRY'] ?? '3600'),
      ),
      refreshExpiry: Duration(
        days: int.parse(Platform.environment['REFRESH_EXPIRY_DAYS'] ?? '7'),
      ),
    );
  }
}
```

```
}  
}
```

### 1.5.0.9 Zusammenfassung

- **Umgebungsvariablen** über `Platform.environment` lesen
- **.env-Dateien** für lokale Entwicklung
- **Konfigurationsklassen** für typsichere Konfiguration
- **.env.example** ins Git, echte **.env** in **.gitignore**
- **Validation** für Produktions-Konfiguration
- **Secrets** niemals in Git committen

### 1.5.0.10 Nächste Schritte

In der nächsten Einheit lernst du **Projekt-Struktur & Architektur**: Wie du dein Backend-Projekt organisierst für bessere Wartbarkeit.

## 1.5.1 Übung

### 1.5.1.1 Ziel

Erstelle ein konfigurierbares Server-Projekt mit Umgebungsvariablen, .env-Dateien und einer typsicheren Konfigurationsklasse.

### 1.5.1.2 Aufgabe 1: .env-Loader implementieren (15 min)

Erstelle eine Funktion, die .env-Dateien lädt.

Anforderungen

```
Future<Map<String, String>> loadEnvFile(String path);
```

Die Funktion soll: 1. Die Datei lesen (falls vorhanden) 2. Kommentarzeilen (#) ignorieren 3. Leere Zeilen ignorieren 4. **KEY=VALUE** Format parsen 5. Quotes ( " und ' ) von Werten entfernen 6. Leere Map zurückgeben wenn Datei nicht existiert

Test-Datei (.env.test)

```
# Server  
PORT=3000  
HOST=localhost  
  
# Database  
DATABASE_URL="postgres://user:pass@localhost:5432/testdb"  
  
# Feature Flags  
DEBUG=true  
ENABLE_LOGGING='true'  
  
# Leerzeile ignorieren  
  
# Ungültige Zeile ohne = sollte ignoriert werden  
INVALID_LINE
```

### 1.5.1.3 Aufgabe 2: Config-Klasse erstellen (20 min)

Erstelle eine typsichere Konfigurationsklasse.

Anforderungen

```
class AppConfig {  
    // Server  
    final String host;  
    final int port;  
  
    // Database  
    final String databaseUrl;  
    final int databasePoolSize;  
  
    // Auth  
    final String jwtSecret;  
    final Duration jwtExpiry;  
  
    // Features  
    final bool debugMode;  
    final String logLevel;  
  
    // Environment  
    final String environment;  
  
    // Getter  
    bool get isDevelopment;  
    bool get isProduction;  
    bool get isStaging;  
}
```

Factory-Konstruktor

```
factory AppConfig.fromEnvironment(Map<String, String> env);
```

Anforderungen an fromEnvironment:

1. Pflichtfelder: DATABASE\_URL, JWT\_SECRET
2. Optionale Felder mit Defaults:
  - HOST: localhost
  - PORT: 8080
  - DATABASE\_POOL\_SIZE: 10
  - JWT\_EXPIRY: 3600 (Sekunden)
  - DEBUG: false
  - LOG\_LEVEL: info
  - ENVIRONMENT: development
3. Bei fehlendem Pflichtfeld: Exception werfen

### 1.5.1.4 Aufgabe 3: Config-Validation (10 min)

Füge eine validate()-Methode zur Config-Klasse hinzu.

Regeln

1. `jwtSecret` muss mindestens 32 Zeichen haben
2. `databaseUrl` muss mit `postgres://` oder `postgresql://` beginnen
3. `port` muss zwischen 1 und 65535 liegen
4. `logLevel` muss einer von: `debug`, `info`, `warning`, `error` sein
5. In Production darf `debugMode` nicht `true` sein
6. In Production darf `jwtSecret` nicht “development” enthalten

Validation-Fehler sammeln

```
class ConfigValidationException implements Exception {
    final List<String> errors;
    ConfigValidationException(this.errors);

    @override
    String toString() => 'ConfigValidationException:\n${errors.join('\n')}';
}
```

#### 1.5.1.5 Aufgabe 4: Environment-Loader (15 min)

Erstelle eine Funktion, die Umgebung und `.env` zusammenführt.

Anforderungen

```
Future<AppConfig> loadConfig() async {
    // 1. Basis-Environment laden
    final env = Map<String, String>.from(Platform.environment);

    // 2. .env-Datei laden (überschreibt nicht)
    final dotenv = await loadEnvFile('.env');
    for (final entry in dotenv.entries) {
        env.putIfAbsent(entry.key, () => entry.value);
    }

    // 3. Umgebungsspezifische .env laden
    // Wenn ENVIRONMENT=staging, lade .env.staging
    final environment = env['ENVIRONMENT'] ?? 'development';
    final envSpecific = await loadEnvFile('.env.$environment');
    // ...

    // 4. Config erstellen und validieren
    final config = AppConfig.fromEnvironment(env);
    config.validate();

    return config;
}
```

Priorität (höchste zuerst)

1. System-Umgebungsvariablen (`Platform.environment`)
2. `.env.{environment}`
3. `.env`

### 1.5.1.6 Aufgabe 5: Server mit Config starten (10 min)

Erstelle einen Server, der die Konfiguration nutzt.

main.dart

```
void main() async {
  try {
    final config = await loadConfig();

    print('Starting server...');
    print('Environment: ${config.environment}');
    print('Debug Mode: ${config.debugMode}');
    print('Log Level: ${config.logLevel}');

    final handler = Pipeline()
      .addMiddleware(logRequests())
      .addHandler(_router(config));

    await shelf_io.serve(handler, config.host, config.port);
    print('Server running on http://${config.host}:${config.port}');
  } on ConfigValidationException catch (e) {
    print('Configuration Error:');
    print(e);
    exit(1);
  } catch (e) {
    print('Failed to start server: $e');
    exit(1);
  }
}

Router _router(AppConfig config) {
  final router = Router();

  router.get('/health', (Request r) => Response.ok(jsonEncode({
    'status': 'ok',
    'environment': config.environment,
    'debug': config.debugMode,
  })));

  router.get('/config', (Request r) {
    // Nur in Development!
    if (!config.isDevelopment) {
      return Response.forbidden('Not allowed in production');
    }
    return Response.ok(jsonEncode({
      'host': config.host,
      'port': config.port,
      'environment': config.environment,
      'logLevel': config.logLevel,
      // NIEMALS Secrets ausgeben!
    })));
  });
}
```

```
    return router;
}
```

### 1.5.1.7 Test-Setup

.env

```
PORT=8080
HOST=localhost
DATABASE_URL=postgres://dev:dev@localhost:5432/devdb
JWT_SECRET=development-secret-key-that-is-long-enough
DEBUG=true
LOG_LEVEL=debug
ENVIRONMENT=development
```

.env.production

```
PORT=80
HOST=0.0.0.0
DATABASE_POOL_SIZE=50
DEBUG=false
LOG_LEVEL=warning
ENVIRONMENT=production
```

### 1.5.1.8 Testen

```
# Development (Standard)
dart run bin/server.dart

# Mit Umgebungsvariablen
PORT=3000 DEBUG=false dart run bin/server.dart

# Production simulieren
ENVIRONMENT=production \
DATABASE_URL=postgres://prod:prod@db:5432/proddb \
JWT_SECRET=production-secret-key-that-is-at-least-32-chars \
dart run bin/server.dart

# Validation-Fehler testen
JWT_SECRET=short dart run bin/server.dart

# Sollte Fehler zeigen

# Endpunkte testen
curl http://localhost:8080/health
curl http://localhost:8080/config # Nur in Development
```

### 1.5.1.9 Abgabe-Checkliste

- ☐ .env-Loader funktioniert korrekt
- ☐ Config-Klasse mit allen Feldern

- ☐ Pflichtfeld-Prüfung funktioniert
- ☐ Validation mit sinnvollen Regeln
- ☐ Environment-spezifische Dateien werden geladen
- ☐ Server nutzt Konfiguration
- ☐ /config Endpunkt nur in Development

## 1.5.2 Lösung

### 1.5.2.1 Vollständige Lösung

lib/config/env\_loader.dart

```
import 'dart:io';

/// Lädt eine .env-Datei und gibt die Werte als Map zurück
Future<Map<String, String>> loadEnvFile(String path) async {
  final file = File(path);
  final env = <String, String>{};

  if (!await file.exists()) {
    return env;
  }

  final lines = await file.readAsLines();

  for (final line in lines) {
    final trimmed = line.trim();

    // Kommentare und leere Zeilen überspringen
    if (trimmed.isEmpty || trimmed.startsWith('#')) {
      continue;
    }

    // KEY=VALUE Format parsen
    final equalsIndex = trimmed.indexOf('=');
    if (equalsIndex == -1) {
      continue; // Ungültige Zeile
    }

    final key = trimmed.substring(0, equalsIndex).trim();
    var value = trimmed.substring(equalsIndex + 1).trim();

    // Quotes entfernen
    if (value.length >= 2) {
      if ((value.startsWith('"') && value.endsWith('"')) ||
          (value.startsWith("'") && value.endsWith("'"))) {
        value = value.substring(1, value.length - 1);
      }
    }

    env[key] = value;
  }
}
```

```
    return env;
}
```

lib/config/app\_config.dart

```
import 'dart:io';

class AppConfig {
  // Server
  final String host;
  final int port;

  // Database
  final String databaseUrl;
  final int databasePoolSize;

  // Auth
  final String jwtSecret;
  final Duration jwtExpiry;

  // Features
  final bool debugMode;
  final String logLevel;

  // Environment
  final String environment;

  AppConfig({
    required this.host,
    required this.port,
    required this.databaseUrl,
    required this.databasePoolSize,
    required this.jwtSecret,
    required this.jwtExpiry,
    required this.debugMode,
    required this.logLevel,
    required this.environment,
  });

  // Environment-Getter
  bool get isDevelopment => environment == 'development';
  bool get isProduction => environment == 'production';
  bool get isStaging => environment == 'staging';

  /// Erstellt Config aus Environment-Map
  factory AppConfig.fromEnvironment(Map<String, String> env) {
    // Pflichtfelder prüfen
    final databaseUrl = env['DATABASE_URL'];
    if (databaseUrl == null || databaseUrl.isEmpty) {
      throw ConfigurationException('Required: DATABASE_URL is not set');
    }
  }
}
```

```

}

final jwtSecret = env['JWT_SECRET'];
if (jwtSecret == null || jwtSecret.isEmpty) {
    throw ConfigurationException('Required: JWT_SECRET is not set');
}

return AppConfig(
    host: env['HOST'] ?? 'localhost',
    port: int.tryParse(env['PORT'] ?? '') ?? 8080,
    databaseUrl: databaseUrl,
    databasePoolSize: int.tryParse(env['DATABASE_POOL_SIZE'] ?? '') ?? 10,
    jwtSecret: jwtSecret,
    jwtExpiry: Duration(
        seconds: int.tryParse(env['JWT_EXPIRY'] ?? '') ?? 3600,
    ),
    debugMode: env['DEBUG']?.toLowerCase() == 'true',
    logLevel: env['LOG_LEVEL'] ?? 'info',
    environment: env['ENVIRONMENT'] ?? 'development',
);
}

/// Validiert die Konfiguration
void validate() {
    final errors = <String>[];

    // JWT Secret Länge
    if (jwtSecret.length < 32) {
        errors.add('JWT_SECRET must be at least 32 characters (got
↵  ${jwtSecret.length})');
    }

    // Database URL Format
    if (!databaseUrl.startsWith('postgres://') &&
        !databaseUrl.startsWith('postgresql://')) {
        errors.add('DATABASE_URL must start with postgres:// or postgresql://');
    }

    // Port Range
    if (port < 1 || port > 65535) {
        errors.add('PORT must be between 1 and 65535 (got $port)');
    }

    // Log Level
    final validLogLevels = ['debug', 'info', 'warning', 'error'];
    if (!validLogLevels.contains(logLevel.toLowerCase())) {
        errors.add('LOG_LEVEL must be one of: ${validLogLevels.join(', ')} (got
↵  $logLevel)');
    }

    // Production-spezifische Regeln

```

```

    if (isProduction) {
      if (debugMode) {
        errors.add('DEBUG must be false in production');
      }
      if (jwtSecret.toLowerCase().contains('development') ||
          jwtSecret.toLowerCase().contains('dev') ||
          jwtSecret.toLowerCase().contains('test')) {
        errors.add('JWT_SECRET must not contain "development", "dev", or
↪ "test" in production');
      }
    }

    if (errors.isNotEmpty) {
      throw ConfigValidationException(errors);
    }
  }

  @override
  String toString() {
    return 'AppConfig('
      'environment: $environment, '
      'host: $host, '
      'port: $port, '
      'debugMode: $debugMode, '
      'logLevel: $logLevel'
      ')';
  }
}

class ConfigurationException implements Exception {
  final String message;
  ConfigurationException(this.message);

  @override
  String toString() => 'ConfigurationException: $message';
}

class ConfigValidationException implements Exception {
  final List<String> errors;
  ConfigValidationException(this.errors);

  @override
  String toString() =>
    'ConfigValidationException:\n - ${errors.join('\n - ')}';
}

lib/config/config_loader.dart

import 'dart:io';
import 'app_config.dart';
import 'env_loader.dart';

```

```
/// Lädt die Konfiguration aus Umgebungsvariablen und .env-Dateien
Future<AppConfig> loadConfig() async {
  // 1. Starte mit System-Umgebungsvariablen (höchste Priorität)
  final env = Map<String, String>.from(Platform.environment);

  // 2. Environment ermitteln
  final environment = env['ENVIRONMENT'] ?? 'development';

  // 3. Umgebungsspezifische .env laden (mittlere Priorität)
  final envSpecificPath = '.env.$environment';
  final envSpecific = await loadEnvFile(envSpecificPath);
  for (final entry in envSpecific.entries) {
    env.putIfAbsent(entry.key, () => entry.value);
  }

  // 4. Standard .env laden (niedrigste Priorität)
  final dotenv = await loadEnvFile('.env');
  for (final entry in dotenv.entries) {
    env.putIfAbsent(entry.key, () => entry.value);
  }

  // 5. Config erstellen
  final config = AppConfig.fromEnvironment(env);

  // 6. Validieren
  config.validate();

  return config;
}
```

bin/server.dart

```
import 'dart:convert';
import 'dart:io';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';

// Config importieren (Pfade anpassen je nach Projektstruktur)
import '../lib/config/app_config.dart';
import '../lib/config/config_loader.dart';

void main() async {
  try {
    // Config laden
    final config = await loadConfig();

    print('=' .padRight(50, '='));
    print('Starting server...');
    print('Environment: ${config.environment}');
```

```

    print('Host: ${config.host}');
    print('Port: ${config.port}');
    print('Debug Mode: ${config.debugMode}');
    print('Log Level: ${config.logLevel}');
    print('=' .padRight(50, '='));

    // Handler aufbauen
    final handler = Pipeline()
      .addMiddleware(logRequests())
      .addHandler(_router(config).call);

    // Server starten
    final server = await shelf_io.serve(handler, config.host, config.port);
    print('\nServer running on http://${server.address.host}:${server.port}');

    // Graceful Shutdown
    ProcessSignal.sigint.watch().listen((_) async {
      print('\nShutting down...');
      await server.close();
      exit(0);
    });
  } on ConfigurationException catch (e) {
    print('\n❌ Configuration Error:');
    print('  $e');
    exit(1);
  } on ConfigValidationException catch (e) {
    print('\n❌ Configuration Validation Failed:');
    print('  ${e.errors.join('\n  ')}');
    exit(1);
  } catch (e, stack) {
    print('\n❌ Failed to start server:');
    print('  $e');
    if (Platform.environment['DEBUG'] == 'true') {
      print('\nStack trace:');
      print(stack);
    }
    exit(1);
  }
}

Router _router(AppConfig config) {
  final router = Router();

  // Health Check
  router.get('/health', (Request request) {
    return Response.ok(
      jsonEncode({
        'status': 'ok',
        'environment': config.environment,
        'debug': config.debugMode,
        'timestamp': DateTime.now().toUtc().toIso8601String(),

```

```

    }},
    headers: {'content-type': 'application/json'},
  );
});

// Config Endpoint (nur Development!)
router.get('/config', (Request request) {
  if (!config.isDevelopment) {
    return Response.forbidden(
      jsonEncode({'error': 'Not allowed in ${config.environment}'}),
      headers: {'content-type': 'application/json'},
    );
  }

  // NIEMALS Secrets ausgeben!
  return Response.ok(
    jsonEncode({
      'host': config.host,
      'port': config.port,
      'environment': config.environment,
      'debugMode': config.debugMode,
      'logLevel': config.logLevel,
      'databasePoolSize': config.databasePoolSize,
      'jwtExpirySeconds': config.jwtExpiry.inSeconds,
      // Secrets werden NICHT ausgegeben
    }),
    headers: {'content-type': 'application/json'},
  );
});

// 404
router.all('/<path|.*>', (Request request, String path) {
  return Response.notFound(
    jsonEncode({'error': 'Not Found', 'path': '/$path'}),
    headers: {'content-type': 'application/json'},
  );
});

return router;
}

```

### 1.5.2.2 Test-Dateien

.env

```

# Development Environment
PORT=8080
HOST=localhost
DATABASE_URL=postgres://dev:devpass@localhost:5432/devdb
JWT_SECRET=development-secret-key-that-is-long-enough-32chars
DATABASE_POOL_SIZE=5

```

```
JWT_EXPIRY=3600
DEBUG=true
LOG_LEVEL=debug
ENVIRONMENT=development
```

.env.production

```
# Production Environment
PORT=80
HOST=0.0.0.0
DATABASE_POOL_SIZE=50
DEBUG=false
LOG_LEVEL=warning
ENVIRONMENT=production
# DATABASE_URL und JWT_SECRET müssen als System-Env gesetzt werden!
```

.env.example

```
# Copy this file to .env and fill in your values
PORT=8080
HOST=localhost
DATABASE_URL=postgres://user:password@localhost:5432/dbname
JWT_SECRET=your-secret-key-at-least-32-characters-long
DATABASE_POOL_SIZE=10
JWT_EXPIRY=3600
DEBUG=true
LOG_LEVEL=debug
ENVIRONMENT=development
```

### 1.5.2.3 Test-Befehle

```
# Development (Standard)
dart run bin/server.dart

# Output:
# =====
# Starting server...
# Environment: development
# Host: localhost
# Port: 8080
# Debug Mode: true
# Log Level: debug
# =====
# Server running on http://localhost:8080

# Endpunkte testen
curl http://localhost:8080/health
curl http://localhost:8080/config

# Mit überschriebenen Variablen
PORT=3000 LOG_LEVEL=info dart run bin/server.dart
```

```

# Production simulieren
ENVIRONMENT=production \
DATABASE_URL=postgres://prod:prod@db:5432/proddb \
JWT_SECRET=production-secret-key-that-is-definitely-long-enough \
dart run bin/server.dart

# Validation-Fehler testen
JWT_SECRET=short DATABASE_URL=invalid dart run bin/server.dart
# Output:
# ☐ Configuration Validation Failed:
#   JWT_SECRET must be at least 32 characters (got 5)
#   DATABASE_URL must start with postgres:// or postgresql://

# Debug in Production testen
ENVIRONMENT=production \
DEBUG=true \
DATABASE_URL=postgres://x:x@x:5432/x \
JWT_SECRET=development-secret-which-is-long-enough-32chars \
dart run bin/server.dart
# Output:
# ☐ Configuration Validation Failed:
#   DEBUG must be false in production
#   JWT_SECRET must not contain "development", "dev", or "test" in production

```

#### 1.5.2.4 Wichtige Erkenntnisse

##### 1. Priorität der Konfiguration

System-Env > .env.{environment} > .env

##### 2. Secrets niemals ausgeben

```

// FALSCH
return Response.ok(jsonEncode({
  'jwtSecret': config.jwtSecret, // NIEMALS!
  'databaseUrl': config.databaseUrl, // Enthält Passwort!
}));

// RICHTIG
return Response.ok(jsonEncode({
  'environment': config.environment,
  'debugMode': config.debugMode,
  // Keine Secrets
}));

```

##### 3. putIfAbsent vs. direkte Zuweisung

```

// putIfAbsent: Überschreibt NICHT existierende Werte
env.putIfAbsent('KEY', () => 'value');

// Direkte Zuweisung: Überschreibt immer
env['KEY'] = 'value';

```

#### 4. Frühe Validierung

Validiere die Konfiguration beim Start, nicht erst bei Verwendung:

```
void main() async {
  try {
    final config = await loadConfig();
    config.validate(); // Früh validieren!
    // ...
  } on ConfigValidationException catch (e) {
    print('Config invalid: $e');
    exit(1);
  }
}
```

### 1.5.3 Ressourcen

#### 1.5.3.1 Packages

Package	Beschreibung	Link
dotenv	.env-Dateien laden	<a href="#">pub.dev</a>
envied	Type-safe env mit Code-Gen	<a href="#">pub.dev</a>
args	Command-Line Arguments	<a href="#">pub.dev</a>

#### 1.5.3.2 Cheat Sheet: Umgebungsvariablen

```
import 'dart:io';

// Variable lesen
final value = Platform.environment['KEY'];

// Mit Default
final port = Platform.environment['PORT'] ?? '8080';

// Zu int konvertieren
final portInt = int.parse(Platform.environment['PORT'] ?? '8080');

// Pflichtfeld
final required = Platform.environment['KEY'] ??
  (throw Exception('KEY not set'));

// Boolean
final debug = Platform.environment['DEBUG'] == 'true';
```

#### 1.5.3.3 Cheat Sheet: .env Format

```
# Kommentar
KEY=value
PORT=8080
```

```

DATABASE_URL=postgres://user:pass@host:5432/db

# Mit Quotes (für Leerzeichen)
MESSAGE="Hello World"
PATH='some/path with/spaces'

# Mehrzeilige Werte (mit \n)
PRIVATE_KEY="-----BEGIN KEY-----\nbase64data\n-----END KEY-----"

```

#### 1.5.3.4 Cheat Sheet: Config-Klasse

```

class Config {
    final int port;
    final String dbUrl;

    Config({required this.port, required this.dbUrl});

    factory Config.fromEnv() {
        return Config(
            port: int.parse(Platform.environment['PORT'] ?? '8080'),
            dbUrl: Platform.environment['DATABASE_URL'] ?? '',
        );
    }
}

```

#### 1.5.3.5 Cheat Sheet: .env laden (manuell)

```

Future<Map<String, String>> loadEnv([String path = '.env']) async {
    final file = File(path);
    if (!await file.exists()) return {};

    final env = <String, String>{};
    for (final line in await file.readAsLines()) {
        final l = line.trim();
        if (l.isEmpty || l.startsWith('#')) continue;
        final i = l.indexOf('=');
        if (i == -1) continue;
        env[l.substring(0, i).trim()] = l.substring(i + 1).trim();
    }
    return env;
}

```

#### 1.5.3.6 Best Practices

##### 1. .env.example verwenden

```

# .env.example (im Git)
PORT=8080
DATABASE_URL=postgres://user:password@localhost:5432/dbname
JWT_SECRET=your-secret-here

```

## 2. Validierung

```
void validateConfig(Config config) {
  if (config.jwtSecret.length < 32) {
    throw Exception('JWT_SECRET too short');
  }
  if (config.isProduction && config.jwtSecret.contains('dev')) {
    throw Exception('Development secret in production!');
  }
}
```

## 3. Typsichere Defaults

```
class Config {
  static int _intEnv(String key, int def) =>
    int.tryParse(Platform.environment[key] ?? '') ?? def;

  static bool _boolEnv(String key, bool def) =>
    Platform.environment[key]?.toLowerCase() == 'true' || def;

  static Duration _durationEnv(String key, Duration def) =>
    Duration(seconds: _intEnv(key, def.inSeconds));
}
```

## 4. .gitignore

```
# Umgebungsvariablen
.env
.env.local
.env.*.local

# Secrets
*.pem
*.key
secrets/

# Behalte Example
!.env.example
```

## 1.5.3.7 Projektstruktur

```
project/
+-- .env                # Nicht im Git
+-- .env.example        # Im Git
+-- lib/
|   +-- config/
|       +-- config.dart
|       +-- database_config.dart
|       +-- auth_config.dart
+-- bin/
    +-- server.dart
```

### 1.5.3.8 Shell-Befehle

```
# Variable setzen
export PORT=8080

# Inline für einen Befehl
PORT=8080 dart run bin/server.dart

# Mehrere Variablen
export PORT=8080 HOST=0.0.0.0

# .env manuell laden (Bash)
export $(cat .env | xargs)

# Variable prüfen
echo $PORT
printenv PORT
```

### 1.5.3.9 Docker & Deployment

```
# Dockerfile
FROM dart:stable AS build
WORKDIR /app
COPY . .
RUN dart compile exe bin/server.dart -o server

FROM scratch
COPY --from=build /runtime/ /
COPY --from=build /app/server /app/server
# ENV wird beim Container-Start gesetzt
CMD ["/app/server"]
```

```
# Docker run mit Env
docker run -e PORT=8080 -e DATABASE_URL=... myapp

# Docker Compose
# docker-compose.yml
services:
  app:
    environment:
      - PORT=8080
      - DATABASE_URL=postgres://...
    env_file:
      - .env.production
```

## 1.6 Einheit 5.6: Projektstruktur

### 1.6.0.1 Lernziele

Nach dieser Einheit kannst du: - Ein Backend-Projekt sinnvoll strukturieren - Layered Architecture anwenden - Dependency Injection nutzen - Das Service- und Repository-Pattern implementieren

### 1.6.0.2 Warum Struktur wichtig ist

Ein gut strukturiertes Projekt: - Ist **wartbar** (Code leicht zu finden und ändern) - Ist **testbar** (Komponenten isoliert testbar) - Ist **skalierbar** (neue Features leicht hinzufügen) - Hat **klare Verantwortlichkeiten** (Single Responsibility)

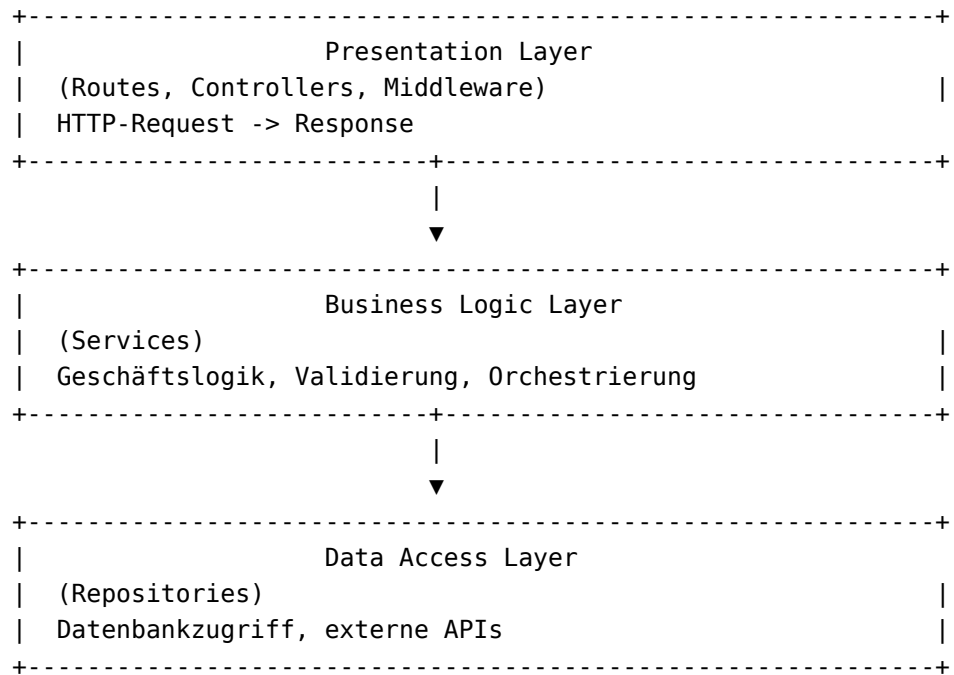
### 1.6.0.3 Empfohlene Projektstruktur

```
my_api/
+-- bin/
|   +-- server.dart          # Einstiegspunkt
+-- lib/
|   +-- app.dart             # App-Setup (Pipeline, Router)
|   +-- config/
|   |   +-- config.dart      # Konfigurationsklassen
|   |   +-- config_loader.dart
|   +-- middleware/
|   |   +-- auth_middleware.dart
|   |   +-- cors_middleware.dart
|   |   +-- error_middleware.dart
|   +-- routes/
|   |   +-- routes.dart      # Alle Routes zusammen
|   |   +-- user_routes.dart
|   |   +-- product_routes.dart
|   +-- controllers/
|   |   +-- user_controller.dart
|   |   +-- product_controller.dart
|   +-- services/
|   |   +-- user_service.dart
|   |   +-- auth_service.dart
|   +-- repositories/
|   |   +-- user_repository.dart
|   |   +-- product_repository.dart
|   +-- models/
|   |   +-- user.dart
|   |   +-- product.dart
|   +-- dto/                 # Data Transfer Objects
|   |   +-- create_user_dto.dart
|   |   +-- update_user_dto.dart
|   +-- utils/
|       +-- json_response.dart
|       +-- validators.dart
+-- test/
|   +-- unit/
|   +-- integration/
```

```
|  +-- helpers/
+-- pubspec.yaml
+-- analysis_options.yaml
+-- .env.example
+-- README.md
```

#### 1.6.0.4 Layered Architecture

Die Schichten einer Backend-Anwendung:



Regeln

1. **Höhere Schichten** dürfen **niedrigere Schichten** aufrufen
2. **Niedrigere Schichten** kennen **höhere Schichten** nicht
3. Jede Schicht hat eine **klare Verantwortung**

#### 1.6.0.5 Models / Entities

Models repräsentieren die Datenstrukturen:

```
// lib/models/user.dart
class User {
  final String id;
  final String email;
  final String name;
  final DateTime createdAt;
  final DateTime? updatedAt;

  User({
    required this.id,
    required this.email,
    required this.name,
    required this.createdAt,
    this.updatedAt,
```

```

});

// JSON-Serialisierung
factory User.fromJson(Map<String, dynamic> json) {
  return User(
    id: json['id'] as String,
    email: json['email'] as String,
    name: json['name'] as String,
    createdAt: DateTime.parse(json['created_at'] as String),
    updatedAt: json['updated_at'] != null
      ? DateTime.parse(json['updated_at'] as String)
      : null,
  );
}

Map<String, dynamic> toJson() {
  return {
    'id': id,
    'email': email,
    'name': name,
    'created_at': createdAt.toIso8601String(),
    'updated_at': updatedAt?.toIso8601String(),
  };
}

// Kopie mit geänderten Werten
User copyWith({
  String? email,
  String? name,
  DateTime? updatedAt,
}) {
  return User(
    id: id,
    email: email ?? this.email,
    name: name ?? this.name,
    createdAt: createdAt,
    updatedAt: updatedAt ?? this.updatedAt,
  );
}
}

```

#### 1.6.0.6 Repository Pattern

Repositories abstrahieren den Datenzugriff:

```

// lib/repositories/user_repository.dart

/// Interface für UserRepository
abstract class UserRepository {
  Future<List<User>> findAll();
  Future<User?> findById(String id);
}

```

```
Future<User?> findByEmail(String email);
Future<User> create(User user);
Future<User> update(User user);
Future<void> delete(String id);
}

/// In-Memory Implementierung (für Entwicklung/Tests)
class InMemoryUserRepository implements UserRepository {
    final _users = <String, User>{};
    var _nextId = 1;

    @override
    Future<List<User>> findAll() async {
        return _users.values.toList();
    }

    @override
    Future<User?> findById(String id) async {
        return _users[id];
    }

    @override
    Future<User?> findByEmail(String email) async {
        return _users.values.where((u) => u.email == email).firstOrNull;
    }

    @override
    Future<User> create(User user) async {
        final id = '${_nextId++}';
        final newUser = User(
            id: id,
            email: user.email,
            name: user.name,
            createdAt: DateTime.now(),
        );
        _users[id] = newUser;
        return newUser;
    }

    @override
    Future<User> update(User user) async {
        if (!_users.containsKey(user.id)) {
            throw NotFoundException('User ${user.id} not found');
        }
        final updated = user.copyWith(updatedAt: DateTime.now());
        _users[user.id] = updated;
        return updated;
    }

    @override
    Future<void> delete(String id) async {
```

```
    _users.remove(id);  
  }  
}
```

### 1.6.0.7 Service Pattern

Services enthalten die Geschäftslogik:

```
// lib/services/user_service.dart  
class UserService {  
  final UserRepository _repository;  
  final AuthService _authService;  
  
  UserService(this._repository, this._authService);  
  
  Future<List<User>> getAllUsers() {  
    return _repository.findAll();  
  }  
  
  Future<User> getUserById(String id) async {  
    final user = await _repository.findById(id);  
    if (user == null) {  
      throw NotFoundException('User $id not found');  
    }  
    return user;  
  }  
  
  Future<User> createUser(CreateUserDto dto) async {  
    // Validierung  
    _validateEmail(dto.email);  
    _validatePassword(dto.password);  
  
    // Prüfen ob Email bereits existiert  
    final existing = await _repository.findByEmail(dto.email);  
    if (existing != null) {  
      throw ValidationException('Email already in use');  
    }  
  
    // Passwort hashen  
    final passwordHash = await _authService.hashPassword(dto.password);  
  
    // User erstellen  
    final user = User(  
      id: '', // Wird vom Repository gesetzt  
      email: dto.email,  
      name: dto.name,  
      createdAt: DateTime.now(),  
    );  
  
    return _repository.create(user);  
  }  
}
```

```

Future<User> updateUser(String id, UpdateUserDto dto) async {
  final user = await getUserById(id);

  if (dto.email != null) {
    _validateEmail(dto.email!);
    final existing = await _repository.findByEmail(dto.email!);
    if (existing != null && existing.id != id) {
      throw ValidationException('Email already in use');
    }
  }

  final updated = user.copyWith(
    email: dto.email,
    name: dto.name,
  );

  return _repository.update(updated);
}

Future<void> deleteUser(String id) async {
  await getUserById(id); // Prüft ob User existiert
  await _repository.delete(id);
}

void _validateEmail(String email) {
  if (!email.contains('@')) {
    throw ValidationException('Invalid email format');
  }
}

void _validatePassword(String password) {
  if (password.length < 8) {
    throw ValidationException('Password must be at least 8 characters');
  }
}
}

```

#### 1.6.0.8 DTOs (Data Transfer Objects)

DTOs definieren die Struktur von Input-Daten:

```

// lib/dto/create_user_dto.dart
class CreateUserDto {
  final String email;
  final String name;
  final String password;

  CreateUserDto({
    required this.email,
    required this.name,

```

```
        required this.password,
    });

    factory CreateUserDto.fromJson(Map<String, dynamic> json) {
      return CreateUserDto(
        email: json['email'] as String? ?? '',
        name: json['name'] as String? ?? '',
        password: json['password'] as String? ?? '',
      );
    }
  }
}

// lib/dto/update_user_dto.dart
class UpdateUserDto {
  final String? email;
  final String? name;

  UpdateUserDto({this.email, this.name});

  factory UpdateUserDto.fromJson(Map<String, dynamic> json) {
    return UpdateUserDto(
      email: json['email'] as String?,
      name: json['name'] as String?,
    );
  }
}
```

### 1.6.0.9 Controllers

Controllers verbinden Routes mit Services:

```
// lib/controllers/user_controller.dart
class UserController {
  final UserService _service;

  UserController(this._service);

  Router get router {
    final router = Router();

    router.get('/', _list);
    router.get('/:id', _getById);
    router.post('/', _create);
    router.put('/:id', _update);
    router.delete('/:id', _delete);

    return router;
  }

  Future<Response> _list(Request request) async {
    final users = await _service.getAllUsers();
```

```

    return jsonResponse(users.map((u) => u.toJson()).toList());
  }

Future<Response> _getId(Request request, String id) async {
  final user = await _service.getUserById(id);
  return jsonResponse(user.toJson());
}

Future<Response> _create(Request request) async {
  final body = await _parseJson(request);
  final dto = CreateUserDto.fromJson(body);
  final user = await _service.createUser(dto);
  return jsonResponse(user.toJson(), statusCode: 201);
}

Future<Response> _update(Request request, String id) async {
  final body = await _parseJson(request);
  final dto = UpdateUserDto.fromJson(body);
  final user = await _service.updateUser(id, dto);
  return jsonResponse(user.toJson());
}

Future<Response> _delete(Request request, String id) async {
  await _service.deleteUser(id);
  return Response(204);
}

Future<Map<String, dynamic>> _parseJson(Request request) async {
  final body = await request.readAsString();
  return jsonDecode(body) as Map<String, dynamic>;
}
}

```

#### 1.6.0.10 Dependency Injection

Abhängigkeiten werden von außen übergeben:

```

// lib/app.dart
class App {
  final AppConfig config;
  final UserRepository userRepository;
  final AuthService authService;
  late final UserService userService;
  late final UserController userController;

  App(this.config)
    : userRepository = InMemoryUserRepository(),
      authService = AuthService(config.auth.jwtSecret) {
    // Services erstellen
    userService = UserService(userRepository, authService);
  }
}

```

```

    // Controllers erstellen
    userController = UserController(userService);
  }

  Handler get handler {
    final router = Router();

    // Routes mounten
    router.mount('/api/users', userController.router.call);
    router.get('/health', _healthCheck);

    // Pipeline
    return Pipeline()
      .addMiddleware(errorHandler())
      .addMiddleware(logRequests())
      .addMiddleware(corsMiddleware())
      .addMiddleware(authMiddleware(authService))
      .addHandler(router.call);
  }

  Response _healthCheck(Request request) {
    return jsonResponse({'status': 'ok'});
  }
}

// bin/server.dart
void main() async {
  final config = await loadConfig();
  final app = App(config);

  await shelf_io.serve(app.handler, config.server.host, config.server.port);
}

```

#### 1.6.0.11 Exceptions

Zentrale Exception-Definitionen:

```

// lib/utils/exceptions.dart
abstract class AppException implements Exception {
  final String message;
  final int statusCode;

  AppException(this.message, this.statusCode);
}

class NotFoundException extends AppException {
  NotFoundException(String message) : super(message, 404);
}

class ValidationException extends AppException {
  final List<String>? errors;
}

```

```

    ValidationException(String message, {this.errors}) : super(message, 400);
  }

class UnauthorizedException extends AppException {
  UnauthorizedException([String message = 'Unauthorized'])
    : super(message, 401);
}

class ForbiddenException extends AppException {
  ForbiddenException([String message = 'Forbidden']) : super(message, 403);
}

class ConflictException extends AppException {
  ConflictException(String message) : super(message, 409);
}

```

#### 1.6.0.12 Zusammenfassung

Schicht	Verantwortung	Beispiel
<b>Routes/Controllers</b>	HTTP-Handling	Request parsen, Response senden
<b>Services</b>	Geschäftslogik	Validierung, Orchestrierung
<b>Repositories</b>	Datenzugriff	CRUD-Operationen
<b>Models</b>	Datenstrukturen	User, Product
<b>DTOs</b>	Input-Strukturen	CreateUserDto

Vorteile dieser Struktur

1. **Testbarkeit:** Services können mit Mock-Repositories getestet werden
2. **Austauschbarkeit:** Repository-Implementierung kann gewechselt werden
3. **Klarheit:** Jede Datei hat einen klaren Zweck
4. **Wartbarkeit:** Änderungen sind lokal begrenzt

#### 1.6.0.13 Nächste Schritte

Im nächsten Block (Block 6) lernst du **REST API Entwicklung:** REST-Prinzipien, JSON-Serialisierung, CRUD-Operationen und mehr.

### 1.6.1 Übung

#### 1.6.1.1 Ziel

Refaktoriere eine bestehende “Monolith”-Anwendung in eine sauber strukturierte Layered Architecture mit Repositories, Services und Controllers.

#### 1.6.1.2 Ausgangssituation

Du hast folgenden Code in einer einzigen Datei:

```
// bin/server.dart (VORHER - alles in einer Datei)
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';

final _tasks = <String, Map<String, dynamic>>{};
var _nextId = 1;

void main() async {
  final router = Router();

  router.get('/api/tasks', (Request request) {
    return Response.ok(
      jsonEncode({'tasks': _tasks.values.toList()}),
      headers: {'content-type': 'application/json'},
    );
  });

  router.get('/api/tasks/<id>', (Request request, String id) {
    final task = _tasks[id];
    if (task == null) {
      return Response.notFound(jsonEncode({'error': 'Not found'}));
    }
    return Response.ok(jsonEncode(task));
  });

  router.post('/api/tasks', (Request request) async {
    final body = jsonDecode(await request.readAsString());
    if (body['title'] == null || body['title'].isEmpty) {
      return Response.badRequest(body: jsonEncode({'error': 'Title required'}));
    }
    final id = '${_nextId++}';
    _tasks[id] = {
      'id': id,
      'title': body['title'],
      'description': body['description'] ?? '',
      'completed': false,
      'createdAt': DateTime.now().toIso8601String(),
    };
    return Response(201, body: jsonEncode(_tasks[id]));
  });

  // ... mehr Code

  await shelf_io.serve(router.call, 'localhost', 8080);
}
```

### 1.6.1.3 Aufgabe 1: Projektstruktur anlegen (10 min)

Erstelle folgende Ordnerstruktur:

```
task_api/  
+-- bin/  
|   +-- server.dart  
+-- lib/  
|   +-- app.dart  
|   +-- config/  
|       +-- config.dart  
|   +-- controllers/  
|       +-- task_controller.dart  
|   +-- services/  
|       +-- task_service.dart  
|   +-- repositories/  
|       +-- task_repository.dart  
|   +-- models/  
|       +-- task.dart  
|   +-- dto/  
|       +-- create_task_dto.dart  
|       +-- update_task_dto.dart  
|   +-- utils/  
|       +-- exceptions.dart  
|       +-- json_response.dart  
+-- pubspec.yaml
```

#### 1.6.1.4 Aufgabe 2: Model erstellen (10 min)

Erstelle das Task-Model in `lib/models/task.dart`.

Anforderungen

```
class Task {  
  final String id;  
  final String title;  
  final String description;  
  final bool completed;  
  final DateTime createdAt;  
  final DateTime? completedAt;  
  
  // Konstruktor  
  // fromJson factory  
  // toJson method  
  // copyWith method  
}
```

#### 1.6.1.5 Aufgabe 3: Repository erstellen (15 min)

Erstelle das Repository in `lib/repositories/task_repository.dart`.

Interface

```
abstract class TaskRepository {  
  Future<List<Task>> findAll();  
  Future<Task?> findById(String id);  
  Future<List<Task>> findByCompleted(bool completed);  
}
```

```
Future<Task> create(Task task);
Future<Task> update(Task task);
Future<void> delete(String id);
}
```

InMemoryTaskRepository

Implementiere eine In-Memory-Version des Repositories.

#### 1.6.1.6 Aufgabe 4: DTOs erstellen (10 min)

Erstelle die DTOs für Input-Daten.

CreateTaskDto

```
class CreateTaskDto {
  final String title;
  final String? description;

  // fromJson
  // validate() method
}
```

UpdateTaskDto

```
class UpdateTaskDto {
  final String? title;
  final String? description;
  final bool? completed;

  // fromJson
}
```

#### 1.6.1.7 Aufgabe 5: Service erstellen (15 min)

Erstelle den Service in lib/services/task\_service.dart.

Anforderungen

```
class TaskService {
  final TaskRepository _repository;

  TaskService(this._repository);

  Future<List<Task>> getAllTasks({bool? completed});
  Future<Task> getTaskById(String id);
  Future<Task> createTask(CreateTaskDto dto);
  Future<Task> updateTask(String id, UpdateTaskDto dto);
  Future<Task> completeTask(String id);
  Future<void> deleteTask(String id);
}
```

Business-Logik

1. createTask: Validiere Title (nicht leer, max 200 Zeichen)

2. `completeTask`: Setze `completed = true` und `completedAt`
3. `getTaskById`: Werfe `NotFoundException` wenn nicht gefunden

### 1.6.1.8 Aufgabe 6: Controller erstellen (15 min)

Erstelle den Controller in `lib/controllers/task_controller.dart`.

Anforderungen

```
class TaskController {
  final TaskService _service;

  TaskController(this._service);

  Router get router {
    // GET    /           -> Liste aller Tasks
    // GET    /<id>       -> Einzelner Task
    // POST   /           -> Task erstellen
    // PUT    /<id>       -> Task aktualisieren
    // PATCH  /<id>/complete -> Task abschließen
    // DELETE /<id>       -> Task löschen
  }
}
```

Query-Parameter für Liste

```
GET /api/tasks?completed=true   -> Nur abgeschlossene
GET /api/tasks?completed=false -> Nur offene
GET /api/tasks                  -> Alle
```

### 1.6.1.9 Aufgabe 7: App zusammenbauen (10 min)

Erstelle `lib/app.dart` mit Dependency Injection.

```
class App {
  final AppConfig config;
  late final TaskRepository taskRepository;
  late final TaskService taskService;
  late final TaskController taskController;

  App(this.config) {
    // Abhängigkeiten erstellen
  }

  Handler get handler {
    // Pipeline mit Middleware
    // Router mit gemounteten Controllers
  }
}
```

`bin/server.dart`

```
void main() async {
  final config = AppConfig.fromEnvironment();
```

```
final app = App(config);

await shelf_io.serve(app.handler, config.host, config.port);
print('Server running on http://${config.host}:${config.port}');
}
```

#### 1.6.1.10 Aufgabe 8: Utils erstellen (5 min)

lib/utils/exceptions.dart

```
class NotFoundException extends AppException { ... }
class ValidationException extends AppException { ... }
```

lib/utils/json\_response.dart

```
Response jsonResponse(Object? data, {int statusCode = 200});
```

#### 1.6.1.11 Testen

*# Server starten*

```
dart run bin/server.dart
```

*# Tasks erstellen*

```
curl -X POST http://localhost:8080/api/tasks \
  -H "Content-Type: application/json" \
  -d '{"title": "Learn Dart"}'
```

```
curl -X POST http://localhost:8080/api/tasks \
  -H "Content-Type: application/json" \
  -d '{"title": "Build API", "description": "REST API with Shelf"}'
```

*# Alle Tasks*

```
curl http://localhost:8080/api/tasks
```

*# Nur offene*

```
curl "http://localhost:8080/api/tasks?completed=false"
```

*# Task abschließen*

```
curl -X PATCH http://localhost:8080/api/tasks/1/complete
```

*# Nur abgeschlossene*

```
curl "http://localhost:8080/api/tasks?completed=true"
```

*# Task aktualisieren*

```
curl -X PUT http://localhost:8080/api/tasks/2 \
  -H "Content-Type: application/json" \
  -d '{"title": "Build REST API"}'
```

*# Task löschen*

```
curl -X DELETE http://localhost:8080/api/tasks/1
```

```
# Fehler testen
curl -X POST http://localhost:8080/api/tasks \
  -H "Content-Type: application/json" \
  -d '{"title": ""}'

curl http://localhost:8080/api/tasks/999
```

#### 1.6.1.12 Abgabe-Checkliste

- ☐ Projektstruktur angelegt
- ☐ Task-Model mit JSON-Serialisierung
- ☐ TaskRepository Interface und InMemory-Implementierung
- ☐ DTOs für Create und Update
- ☐ TaskService mit Geschäftslogik
- ☐ TaskController mit allen Endpunkten
- ☐ App-Klasse mit Dependency Injection
- ☐ Error-Handler-Middleware
- ☐ Alle Endpunkte funktionieren
- ☐ Filter-Parameter funktioniert

### 1.6.2 Lösung

#### 1.6.2.1 Vollständige Lösung

lib/models/task.dart

```
class Task {
  final String id;
  final String title;
  final String description;
  final bool completed;
  final DateTime createdAt;
  final DateTime? completedAt;

  Task({
    required this.id,
    required this.title,
    this.description = '',
    this.completed = false,
    required this.createdAt,
    this.completedAt,
  });

  factory Task.fromJson(Map<String, dynamic> json) {
    return Task(
      id: json['id'] as String,
      title: json['title'] as String,
      description: json['description'] as String? ?? '',
      completed: json['completed'] as bool? ?? false,
      createdAt: DateTime.parse(json['createdAt'] as String),
      completedAt: json['completedAt'] != null
        ? DateTime.parse(json['completedAt'] as String)
```

```

        : null,
    );
}

Map<String, dynamic> toJson() {
    return {
        'id': id,
        'title': title,
        'description': description,
        'completed': completed,
        'createdAt': createdAt.toIso8601String(),
        if (completedAt != null) 'completedAt': completedAt!.toIso8601String(),
    };
}

Task copyWith({
    String? title,
    String? description,
    bool? completed,
    DateTime? completedAt,
}) {
    return Task(
        id: id,
        title: title ?? this.title,
        description: description ?? this.description,
        completed: completed ?? this.completed,
        createdAt: createdAt,
        completedAt: completedAt ?? this.completedAt,
    );
}
}

```

lib/dto/create\_task\_dto.dart

```

import '../utils/exceptions.dart';

class CreateTaskDto {
    final String title;
    final String? description;

    CreateTaskDto({
        required this.title,
        this.description,
    });

    factory CreateTaskDto.fromJson(Map<String, dynamic> json) {
        return CreateTaskDto(
            title: json['title'] as String ?? '',
            description: json['description'] as String?,
        );
    }
}

```

```
void validate() {
  final errors = <String>[];

  if (title.isEmpty) {
    errors.add('Title is required');
  }
  if (title.length > 200) {
    errors.add('Title must be at most 200 characters');
  }

  if (errors.isNotEmpty) {
    throw ValidationException('Invalid task data', errors: errors);
  }
}
```

lib/dto/update\_task\_dto.dart

```
class UpdateTaskDto {
  final String? title;
  final String? description;
  final bool? completed;

  UpdateTaskDto({
    this.title,
    this.description,
    this.completed,
  });

  factory UpdateTaskDto.fromJson(Map<String, dynamic> json) {
    return UpdateTaskDto(
      title: json['title'] as String?,
      description: json['description'] as String?,
      completed: json['completed'] as bool?,
    );
  }
}
```

lib/repositories/task\_repository.dart

```
import '../models/task.dart';

abstract class TaskRepository {
  Future<List<Task>> findAll();
  Future<Task?> findById(String id);
  Future<List<Task>> findByCompleted(bool completed);
  Future<Task> create(Task task);
  Future<Task> update(Task task);
  Future<void> delete(String id);
}
```

```
class InMemoryTaskRepository implements TaskRepository {
  final _tasks = <String, Task>{};
  var _nextId = 1;

  @override
  Future<List<Task>> findAll() async {
    return _tasks.values.toList()
      ..sort((a, b) => b.createdAt.compareTo(a.createdAt));
  }

  @override
  Future<Task?> findById(String id) async {
    return _tasks[id];
  }

  @override
  Future<List<Task>> findByCompleted(bool completed) async {
    return _tasks.values
      .where((t) => t.completed == completed)
      .toList()
      ..sort((a, b) => b.createdAt.compareTo(a.createdAt));
  }

  @override
  Future<Task> create(Task task) async {
    final id = '${_nextId++}';
    final newTask = Task(
      id: id,
      title: task.title,
      description: task.description,
      completed: task.completed,
      createdAt: DateTime.now(),
    );
    _tasks[id] = newTask;
    return newTask;
  }

  @override
  Future<Task> update(Task task) async {
    _tasks[task.id] = task;
    return task;
  }

  @override
  Future<void> delete(String id) async {
    _tasks.remove(id);
  }
}
```

lib/services/task\_service.dart

```
import '../models/task.dart';
import '../dto/create_task_dto.dart';
import '../dto/update_task_dto.dart';
import '../repositories/task_repository.dart';
import '../utils/exceptions.dart';

class TaskService {
  final TaskRepository _repository;

  TaskService(this._repository);

  Future<List<Task>> getAllTasks({bool? completed}) async {
    if (completed != null) {
      return _repository.findByCompleted(completed);
    }
    return _repository.findAll();
  }

  Future<Task> getTaskById(String id) async {
    final task = await _repository.findById(id);
    if (task == null) {
      throw NotFoundException('Task $id not found');
    }
    return task;
  }

  Future<Task> createTask(CreateTaskDto dto) async {
    // Validierung
    dto.validate();

    // Task erstellen
    final task = Task(
      id: '', // Wird vom Repository gesetzt
      title: dto.title,
      description: dto.description ?? '',
      createdAt: DateTime.now(),
    );

    return _repository.create(task);
  }

  Future<Task> updateTask(String id, UpdateTaskDto dto) async {
    final task = await getTaskById(id);

    // Validierung
    if (dto.title != null) {
      if (dto.title!.isEmpty) {
        throw ValidationException('Title cannot be empty');
      }
      if (dto.title!.length > 200) {
        throw ValidationException('Title must be at most 200 characters');
      }
    }
  }
}
```

```

    }
  }

  final updated = task.copyWith(
    title: dto.title,
    description: dto.description,
    completed: dto.completed,
    completedAt: dto.completed == true && !task.completed
      ? DateTime.now()
      : task.completedAt,
  );

  return _repository.update(updated);
}

Future<Task> completeTask(String id) async {
  final task = await getTaskById(id);

  if (task.completed) {
    throw ValidationException('Task is already completed');
  }

  final completed = task.copyWith(
    completed: true,
    completedAt: DateTime.now(),
  );

  return _repository.update(completed);
}

Future<void> deleteTask(String id) async {
  await getTaskById(id); // Prüft ob Task existiert
  await _repository.delete(id);
}
}

```

lib/controllers/task\_controller.dart

```

import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';

import '../services/task_service.dart';
import '../dto/create_task_dto.dart';
import '../dto/update_task_dto.dart';
import '../utils/json_response.dart';

class TaskController {
  final TaskService _service;

  TaskController(this._service);
}

```

```
Router get router {
    final router = Router();

    router.get('/', _list);
    router.get('/:id', _getById);
    router.post('/', _create);
    router.put('/:id', _update);
    router.patch('/:id/complete', _complete);
    router.delete('/:id', _delete);

    return router;
}

Future<Response> _list(Request request) async {
    // Query-Parameter für Filter
    final completedParam = request.url.queryParameters['completed'];
    bool? completed;
    if (completedParam == 'true') {
        completed = true;
    } else if (completedParam == 'false') {
        completed = false;
    }

    final tasks = await _service.getAllTasks(completed: completed);

    return jsonResponse({
        'tasks': tasks.map((t) => t.toJson()).toList(),
        'total': tasks.length,
        if (completed != null) 'filter': {'completed': completed},
    });
}

Future<Response> _getById(Request request, String id) async {
    final task = await _service.getTaskById(id);
    return jsonResponse(task.toJson());
}

Future<Response> _create(Request request) async {
    final body = await _parseJson(request);
    final dto = CreateTaskDto.fromJson(body);
    final task = await _service.createTask(dto);
    return jsonResponse(task.toJson(), statusCode: 201);
}

Future<Response> _update(Request request, String id) async {
    final body = await _parseJson(request);
    final dto = UpdateTaskDto.fromJson(body);
    final task = await _service.updateTask(id, dto);
    return jsonResponse(task.toJson());
}
```

```
Future<Response> _complete(Request request, String id) async {
  final task = await _service.completeTask(id);
  return jsonResponse(task.toJson());
}

Future<Response> _delete(Request request, String id) async {
  await _service.deleteTask(id);
  return Response(204);
}

Future<Map<String, dynamic>> _parseJson(Request request) async {
  final body = await request.readAsString();
  if (body.isEmpty) return {};
  return jsonDecode(body) as Map<String, dynamic>;
}
}
```

lib/utils/exceptions.dart

```
abstract class AppException implements Exception {
  final String message;
  final int statusCode;

  AppException(this.message, this.statusCode);

  @override
  String toString() => message;
}

class NotFoundException extends AppException {
  NotFoundException(String message) : super(message, 404);
}

class ValidationException extends AppException {
  final List<String>? errors;

  ValidationException(String message, {this.errors}) : super(message, 400);

  @override
  String toString() {
    if (errors != null && errors!.isNotEmpty) {
      return '$message: ${errors!.join(', ')}';
    }
    return message;
  }
}

class UnauthorizedException extends AppException {
  UnauthorizedException([String message = 'Unauthorized'])
    : super(message, 401);
}
```

```
}
```

```
lib/utils/json_response.dart
```

```
import 'dart:convert';
import 'package:shelf/shelf.dart';

Response jsonResponse(Object? data, {int statusCode = 200}) {
  return Response(
    statusCode,
    body: jsonEncode(data),
    headers: {'content-type': 'application/json; charset=utf-8'},
  );
}
```

```
lib/config/config.dart
```

```
import 'dart:io';

class AppConfig {
  final String host;
  final int port;
  final String environment;

  AppConfig({
    required this.host,
    required this.port,
    required this.environment,
  });

  factory AppConfig.fromEnvironment() {
    return AppConfig(
      host: Platform.environment['HOST'] ?? 'localhost',
      port: int.parse(Platform.environment['PORT'] ?? '8080'),
      environment: Platform.environment['ENVIRONMENT'] ?? 'development',
    );
  }

  bool get isDevelopment => environment == 'development';
}
```

```
lib/app.dart
```

```
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';

import 'config/config.dart';
import 'controllers/task_controller.dart';
import 'services/task_service.dart';
import 'repositories/task_repository.dart';
import 'utils/exceptions.dart';
```

```
import 'utils/json_response.dart';

class App {
  final AppConfig config;
  late final TaskRepository taskRepository;
  late final TaskService taskService;
  late final TaskController taskController;

  App(this.config) {
    // Repositories (Data Layer)
    taskRepository = InMemoryTaskRepository();

    // Services (Business Layer)
    taskService = TaskService(taskRepository);

    // Controllers (Presentation Layer)
    taskController = TaskController(taskService);
  }

  Handler get handler {
    final router = Router();

    // Health Check
    router.get('/health', _healthCheck);

    // API Routes
    router.mount('/api/tasks', taskController.router.call);

    // 404
    router.all('/<path|.*>', _notFound);

    // Pipeline mit Middleware
    return Pipeline()
      .addMiddleware(_errorHandler())
      .addMiddleware(logRequests())
      .addHandler(router.call);
  }

  Response _healthCheck(Request request) {
    return jsonResponse({
      'status': 'ok',
      'environment': config.environment,
      'timestamp': DateTime.now().toUtc().toIso8601String(),
    });
  }

  Response _notFound(Request request, String path) {
    return jsonResponse({
      'error': 'Not Found',
      'path': '/$path',
    }, statusCode: 404);
  }
}
```

```

}

Middleware _errorHandler() {
  return (Handler innerHandler) {
    return (Request request) async {
      try {
        return await innerHandler(request);
      } on AppException catch (e) {
        return jsonResponse({
          'error': _getErrorName(e.statusCode),
          'message': e.message,
          if (e is ValidationException && e.errors != null)
            'errors': e.errors,
        }, statusCode: e.statusCode);
      } on FormatException catch (e) {
        return jsonResponse({
          'error': 'Bad Request',
          'message': 'Invalid JSON: ${e.message}',
        }, statusCode: 400);
      } catch (e, stack) {
        print('Unhandled error: $e\n$stack');
        return jsonResponse({
          'error': 'Internal Server Error',
          'message': 'An unexpected error occurred',
        }, statusCode: 500);
      }
    };
  };
}

String _getErrorName(int statusCode) {
  switch (statusCode) {
    case 400: return 'Bad Request';
    case 401: return 'Unauthorized';
    case 403: return 'Forbidden';
    case 404: return 'Not Found';
    case 409: return 'Conflict';
    default: return 'Error';
  }
}
}

```

bin/server.dart

```

import 'dart:io';
import 'package:shelf/shelf_io.dart' as shelf_io;

import '../lib/app.dart';
import '../lib/config/config.dart';

void main() async {

```

```
final config = AppConfig.fromEnvironment();
final app = App(config);

final server = await shelf_io.serve(app.handler, config.host, config.port);

print('Task API Server');
print('Environment: ${config.environment}');
print('Running on http://${server.address.host}:${server.port}');

// Graceful Shutdown
ProcessSignal.sigint.watch().listen((_) async {
  print('\nShutting down...');
  await server.close();
  exit(0);
});
}
```

### 1.6.2.2 Test-Befehle

```
# Server starten
dart run bin/server.dart

# Tasks erstellen
curl -X POST http://localhost:8080/api/tasks \
  -H "Content-Type: application/json" \
  -d '{"title": "Learn Dart", "description": "Study Dart basics"}'

curl -X POST http://localhost:8080/api/tasks \
  -H "Content-Type: application/json" \
  -d '{"title": "Build API"}'

# Alle Tasks
curl http://localhost:8080/api/tasks

# Einzelner Task
curl http://localhost:8080/api/tasks/1

# Task abschließen
curl -X PATCH http://localhost:8080/api/tasks/1/complete

# Nur abgeschlossene
curl "http://localhost:8080/api/tasks?completed=true"

# Nur offene
curl "http://localhost:8080/api/tasks?completed=false"

# Task aktualisieren
curl -X PUT http://localhost:8080/api/tasks/2 \
  -H "Content-Type: application/json" \
  -d '{"title": "Build REST API", "description": "Using Shelf"}'
```

```
# Task löschen
curl -X DELETE http://localhost:8080/api/tasks/1

# Fehler: Leerer Titel
curl -X POST http://localhost:8080/api/tasks \
  -H "Content-Type: application/json" \
  -d '{"title": ""}'

# Fehler: Task nicht gefunden
curl http://localhost:8080/api/tasks/999

# Health Check
curl http://localhost:8080/health
```

### 1.6.2.3 Architektur-Übersicht

```
Request -> Router -> Controller -> Service -> Repository -> Data
                                   ↓           ↓
                               Validation   CRUD Operations
                                   ↓
Response <- Controller <- Service <- Model
```

Die klare Trennung ermöglicht: - **Unit-Tests**: Services mit Mock-Repositories testen - **Aus-tauschbarkeit**: InMemory -> PostgreSQL ohne Controller-Änderung - **Wartbarkeit**: Jede Datei hat einen klaren Zweck

## 1.6.3 Ressourcen

### 1.6.3.1 Architektur-Patterns

- Clean Architecture - Robert C. Martin
- Repository Pattern - Martin Fowler
- Service Layer - Martin Fowler

### 1.6.3.2 Dart-spezifisch

- Effective Dart: Design
- Dart Package Layout

### 1.6.3.3 Cheat Sheet: Projektstruktur

```
project/
+-- bin/                # Einstiegspunkte
|   +-- server.dart
+-- lib/
|   +-- app.dart        # App-Setup
|   +-- config/         # Konfiguration
|   +-- middleware/     # HTTP-Middleware
|   +-- routes/         # Route-Definitionen
|   +-- controllers/    # Request-Handler
|   +-- services/       # Geschäftslogik
|   +-- repositories/   # Datenzugriff
```

```

|   +-- models/           # Datenstrukturen
|   +-- dto/              # Input-Strukturen
|   +-- utils/            # Hilfsfunktionen
+-- test/
|   +-- unit/
|   +-- integration/
+-- pubspec.yaml

```

#### 1.6.3.4 Cheat Sheet: Layered Architecture

```

+-----+
| Presentation (Routes/Controllers)|
+-----+
| Business Logic (Services)        |
+-----+
| Data Access (Repositories)       |
+-----+

```

#### 1.6.3.5 Cheat Sheet: Repository Pattern

```

// Interface
abstract class UserRepository {
    Future<List<User>> findAll();
    Future<User?> findById(String id);
    Future<User> create(User user);
    Future<User> update(User user);
    Future<void> delete(String id);
}

// Implementierung
class PostgresUserRepository implements UserRepository {
    final Database _db;
    PostgresUserRepository(this._db);

    @override
    Future<List<User>> findAll() async {
        final result = await _db.query('SELECT * FROM users');
        return result.map(User.fromRow).toList();
    }
    // ...
}

// In-Memory für Tests
class InMemoryUserRepository implements UserRepository {
    final _users = <String, User>{};
    // ...
}

```

### 1.6.3.6 Cheat Sheet: Service Pattern

```
class UserService {
    final UserRepository _repo;
    final EmailService _email;

    UserService(this._repo, this._email);

    Future<User> createUser(CreateUserDto dto) async {
        // 1. Validierung
        _validate(dto);

        // 2. Business-Logik
        final existing = await _repo.findByEmail(dto.email);
        if (existing != null) throw ConflictException('Email exists');

        // 3. Daten speichern
        final user = await _repo.create(User.fromDto(dto));

        // 4. Side-Effects
        await _email.sendWelcome(user.email);

        return user;
    }
}
```

### 1.6.3.7 Cheat Sheet: Controller

```
class UserController {
    final UserService _service;
    UserController(this._service);

    Router get router {
        final r = Router();
        r.get('/', list);
        r.get('/:id', getById);
        r.post('/', create);
        r.put('/:id', update);
        r.delete('/:id', delete);
        return r;
    }

    Future<Response> list(Request req) async {
        final users = await _service.getAll();
        return jsonResponse(users);
    }
    // ...
}
```

### 1.6.3.8 Cheat Sheet: Dependency Injection

```
class App {  
  // Repositories (unterste Schicht)  
  final UserRepository userRepo;  
  final ProductRepository productRepo;  
  
  // Services (mittlere Schicht)  
  late final UserService userService;  
  late final ProductService productService;  
  
  // Controllers (oberste Schicht)  
  late final UserController userController;  
  
  App(Database db)  
    : userRepo = PostgresUserRepository(db),  
      productRepo = PostgresProductRepository(db) {  
    userService = UserService(userRepo);  
    productService = ProductService(productRepo);  
    userController = UserController(userService);  
  }  
}
```

### 1.6.3.9 Cheat Sheet: DTOs

```
// Input (vom Client)  
class CreateUserDto {  
  final String email;  
  final String name;  
  final String password;  
  
  factory CreateUserDto.fromJson(Map<String, dynamic> json) => CreateUserDto(  
    email: json['email'] ?? '',  
    name: json['name'] ?? '',  
    password: json['password'] ?? '',  
  );  
}  
  
// Output (zum Client)  
class UserResponse {  
  final String id;  
  final String email;  
  final String name;  
  // Kein Passwort!  
  
  Map<String, dynamic> toJson() => {  
    'id': id,  
    'email': email,  
    'name': name,  
  };  
}
```

### 1.6.3.10 Cheat Sheet: Exceptions

```
abstract class AppException implements Exception {
  final String message;
  final int statusCode;
  AppException(this.message, this.statusCode);
}

class NotFoundException extends AppException {
  NotFoundException(String msg) : super(msg, 404);
}

class ValidationException extends AppException {
  ValidationException(String msg) : super(msg, 400);
}

class UnauthorizedException extends AppException {
  UnauthorizedException([String msg = 'Unauthorized']) : super(msg, 401);
}
```

### 1.6.3.11 Best Practices

#### 1. Eine Datei pro Klasse

```
lib/models/
+-- user.dart
+-- product.dart
+-- order.dart
```

#### 2. Barrel-Files für Exports

```
// lib/models/models.dart
export 'user.dart';
export 'product.dart';
export 'order.dart';
```

#### 3. Klare Namenskonventionen

UserRepository	(nicht: UserRepo, UsersRepository)
UserService	(nicht: UserSvc, UsersService)
UserController	(nicht: UserCtrl)
CreateUserDto	(nicht: CreateUser, UserCreateDto)

#### 4. Interfaces für Abstraktion

```
// Immer ein Interface
abstract class UserRepository { ... }

// Dann Implementierungen
class PostgresUserRepository implements UserRepository { ... }
class MongoUserRepository implements UserRepository { ... }
class InMemoryUserRepository implements UserRepository { ... }
```

# Chapter 2

## Block 6: REST API Entwicklung

REST-Prinzipien, JSON, CRUD, Validierung, Error Handling und API-Dokumentation.

### 2.1 Einheit 6.1: REST Prinzipien & Design

#### 2.1.0.1 Lernziele

Nach dieser Einheit kannst du: - Die REST-Architektur und ihre Prinzipien verstehen - HTTP-Methoden korrekt einsetzen - Ressourcen sinnvoll benennen - API-Versionierung implementieren

#### 2.1.0.2 Was ist REST?

**REST** (Representational State Transfer) ist ein Architekturstil für verteilte Systeme. Eine REST-konforme API wird als **RESTful API** bezeichnet.

Die 6 REST-Prinzipien

1. **Client-Server**: Klare Trennung zwischen Client und Server
2. **Stateless**: Jeder Request enthält alle nötigen Informationen
3. **Cacheable**: Responses können gecacht werden
4. **Uniform Interface**: Einheitliche Schnittstelle
5. **Layered System**: Schichten zwischen Client und Server möglich
6. **Code on Demand** (optional): Server kann Code zum Client senden

#### 2.1.0.3 Ressourcen und URIs

In REST dreht sich alles um **Ressourcen**. Eine Ressource ist ein Objekt oder eine Sammlung von Objekten.

Ressourcen-Naming Best Practices

# RICHTIG: Nomen, Plural, lowercase

GET /api/users

GET /api/products

GET /api/orders

# FALSCH: Verben, Singular, CamelCase

GET /api/getUsers

GET /api/User

GET /api/fetchProducts

Hierarchische Ressourcen

# Benutzer

GET /api/users

GET /api/users/123

# Alle Benutzer

# Benutzer mit ID 123

# Bestellungen eines Benutzers

GET /api/users/123/orders # Alle Bestellungen von User 123

GET /api/users/123/orders/456 # Bestellung 456 von User 123

# Produkte in einer Kategorie

GET /api/categories/electronics/products

Konsistente Namenskonventionen

Konvention	Beispiel	Empfehlung
kebab-case	/user-profiles	Empfohlen
snake_case	/user_profiles	Akzeptabel
camelCase	/userProfiles	Vermeiden

### 2.1.0.4 HTTP-Methoden

CRUD-Operationen

Operation	HTTP-Methode	Beispiel	Beschreibung
Create	POST	POST /users	Neuen User erstellen
Read	GET	GET /users/123	User abrufen
Update (komplett)	PUT	PUT /users/123	User komplett ersetzen
Update (teilweise)	PATCH	PATCH /users/123	User teilweise ändern
Delete	DELETE	DELETE /users/123	User löschen

Methoden-Eigenschaften

Methode	Safe	Idempotent	Request Body
GET	Ja	Ja	Nein
POST	Nein	Nein	Ja
PUT	Nein	Ja	Ja
PATCH	Nein	Nein	Ja
DELETE	Nein	Ja	Nein

- **Safe:** Verändert keine Daten auf dem Server
- **Idempotent:** Mehrfaches Ausführen hat denselben Effekt wie einmaliges

PUT vs. PATCH

```
// PUT: Komplettes Ersetzen
// Alle Felder müssen gesendet werden
PUT /api/users/123
{
  "name": "Max Mustermann",
  "email": "max@example.com",
  "age": 30,
  "city": "Berlin"
}
```

```
// PATCH: Teilweises Update
// Nur geänderte Felder senden
PATCH /api/users/123
{
  "city": "München"
}
```

### 2.1.0.5 HTTP-Statuscodes

2xx - Erfolg

Code	Name	Verwendung
200	OK	Erfolgreiche GET, PUT, PATCH, DELETE
201	Created	Ressource erfolgreich erstellt (POST)
204	No Content	Erfolg ohne Response-Body

4xx - Client-Fehler

Code	Name	Verwendung
400	Bad Request	Ungültige Anfrage (Validierungsfehler)
401	Unauthorized	Authentifizierung erforderlich
403	Forbidden	Keine Berechtigung
404	Not Found	Ressource nicht gefunden
409	Conflict	Konflikt (z.B. Duplikat)
422	Unprocessable Entity	Semantisch ungültig

5xx - Server-Fehler

Code	Name	Verwendung
500	Internal Server Error	Unerwarteter Serverfehler
502	Bad Gateway	Upstream-Server-Fehler
503	Service Unavailable	Server überlastet/Wartung

### 2.1.0.6 Request/Response-Format

JSON als Standard

```
// Request
POST /api/users
Content-Type: application/json

{
  "name": "Max Mustermann",
  "email": "max@example.com"
}

// Response
```

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: /api/users/123

{
  "id": "123",
  "name": "Max Mustermann",
  "email": "max@example.com",
  "createdAt": "2024-01-15T10:30:00Z"
}
```

#### Konsistente Response-Struktur

```
// Einzelne Ressource
{
  "id": "123",
  "name": "Max",
  "email": "max@example.com"
}

// Liste von Ressourcen
{
  "data": [
    {"id": "1", "name": "Max"},
    {"id": "2", "name": "Anna"}
  ],
  "meta": {
    "total": 50,
    "page": 1,
    "perPage": 10
  }
}

// Fehler
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input data",
    "details": [
      {"field": "email", "message": "Invalid email format"}
    ]
  }
}
```

#### 2.1.0.7 API-Versionierung

APIs entwickeln sich weiter. Versionierung ermöglicht Breaking Changes ohne bestehende Clients zu brechen.

##### Strategien

1. URL-Pfad (empfohlen)

GET /api/v1/users  
GET /api/v2/users

```
final app = Router();
app.mount('/api/v1', v1Router().call);
app.mount('/api/v2', v2Router().call);
```

## 2. Query-Parameter

GET /api/users?version=1  
GET /api/users?version=2

## 3. Header

GET /api/users  
Accept: application/vnd.myapi.v1+json

Versionierung in Dart

```
Router v1Router() {
  final router = Router();
  router.get('/users', (Request r) => Response.ok('v1 users'));
  return router;
}

Router v2Router() {
  final router = Router();
  router.get('/users', (Request r) => Response.ok('v2 users with pagination'));
  return router;
}

void main() async {
  final app = Router();

  // Beide Versionen verfügbar
  app.mount('/api/v1', v1Router().call);
  app.mount('/api/v2', v2Router().call);

  // Default auf neueste Version
  app.mount('/api', v2Router().call);

  await shelf_io.serve(app.call, 'localhost', 8080);
}
```

### 2.1.0.8 HATEOAS (Hypermedia)

**HATEOAS** (Hypermedia as the Engine of Application State) bedeutet, dass Responses Links zu verwandten Ressourcen enthalten.

```
// Response mit HATEOAS
{
  "id": "123",
  "name": "Max",
  "email": "max@example.com",
  "_links": {
```

```
    "self": {"href": "/api/users/123"},
    "orders": {"href": "/api/users/123/orders"},
    "profile": {"href": "/api/users/123/profile"}
  }
}

// Collection mit Links
{
  "data": [...],
  "_links": {
    "self": {"href": "/api/users?page=2"},
    "first": {"href": "/api/users?page=1"},
    "prev": {"href": "/api/users?page=1"},
    "next": {"href": "/api/users?page=3"},
    "last": {"href": "/api/users?page=10"}
  }
}
```

### 2.1.0.9 Best Practices

#### 1. Konsistente Benennung

```
// GUT
/api/users
/api/user-profiles
/api/order-items

// SCHLECHT
/api/Users
/api/userProfile
/api/order_items
```

#### 2. Keine Verben in URLs

```
// GUT: HTTP-Methode drückt Aktion aus
POST /api/users          # User erstellen
DELETE /api/users/123    # User löschen

// SCHLECHT: Verben in URL
POST /api/createUser
GET /api/deleteUser/123
```

#### 3. Filter über Query-Parameter

```
// GUT
GET /api/users?status=active&role=admin

// SCHLECHT
GET /api/users/active/admin
GET /api/activeAdminUsers
```

#### 4. HTTP-Statuscodes korrekt verwenden

```
// GUT
POST /api/users -> 201 Created
GET /api/users/999 -> 404 Not Found
DELETE /api/users/123 -> 204 No Content

// SCHLECHT
POST /api/users -> 200 OK (sollte 201 sein)
GET /api/users/999 -> 200 OK mit {"error": "not found"}
```

#### 2.1.0.10 Zusammenfassung

Prinzip	Umsetzung
Ressourcen	Nomen, Plural, lowercase
Aktionen	HTTP-Methoden (GET, POST, PUT, PATCH, DELETE)
Statuscodes	2xx Erfolg, 4xx Client-Fehler, 5xx Server-Fehler
Format	JSON mit konsistenter Struktur
Versionierung	URL-Pfad (/api/v1/...)

#### 2.1.0.11 Nächste Schritte

In der nächsten Einheit lernst du **JSON-Serialisierung** in Dart: Wie du Objekte in JSON umwandelst und umgekehrt.

### 2.1.1 Übung

#### 2.1.1.1 Ziel

Entwirf und implementiere eine RESTful API für einen Online-Shop mit korrekten Ressourcen-Namen, HTTP-Methoden und Statuscodes.

#### 2.1.1.2 Aufgabe 1: API-Design (15 min)

Entwirf die API-Struktur für einen Online-Shop mit folgenden Entitäten: - Produkte (Products) - Kategorien (Categories) - Bestellungen (Orders) - Bestellpositionen (Order Items)

Anforderungen

Schreibe für jede Entität die REST-Endpunkte auf:

Aktion	Methode	Pfad	Statuscode
Alle Produkte	?	?	?
Ein Produkt	?	?	?
Produkt erstellen	?	?	?
...	...	...	...

Zusätzliche Endpunkte

- Produkte einer Kategorie
- Bestellungen eines Kunden
- Positionen einer Bestellung

### 2.1.1.3 Aufgabe 2: Basis-Implementation (20 min)

Implementiere die Produkt-Endpunkte.

Datenmodell

```
class Product {  
    final String id;  
    final String name;  
    final String description;  
    final double price;  
    final String categoryId;  
    final int stock;  
    final DateTime createdAt;  
}
```

Endpunkte

```
GET    /api/v1/products      // Alle Produkte  
GET    /api/v1/products/:id // Ein Produkt  
POST   /api/v1/products      // Produkt erstellen  
PUT    /api/v1/products/:id // Produkt ersetzen  
PATCH /api/v1/products/:id // Produkt aktualisieren  
DELETE /api/v1/products/:id // Produkt löschen
```

Korrekte Statuscodes

- GET Liste: 200 OK
- GET Einzel: 200 OK oder 404 Not Found
- POST: 201 Created mit Location-Header
- PUT/PATCH: 200 OK oder 404 Not Found
- DELETE: 204 No Content oder 404 Not Found

### 2.1.1.4 Aufgabe 3: Verschachtelte Ressourcen (15 min)

Implementiere verschachtelte Ressourcen.

Kategorie-Produkte

```
GET /api/v1/categories/:categoryId/products
```

Liefert alle Produkte einer Kategorie.

Response

```
{  
  "category": {  
    "id": "electronics",  
    "name": "Elektronik"  
  },  
  "products": [  
    {"id": "1", "name": "Laptop", "price": 999.99},  
    {"id": "2", "name": "Smartphone", "price": 599.99}  
  ],  
  "total": 2  
}
```

### 2.1.1.5 Aufgabe 4: API-Versionierung (10 min)

Implementiere zwei API-Versionen.

v1: Einfache Produktliste

```
{
  "products": [
    {"id": "1", "name": "Laptop", "price": 999.99}
  ]
}
```

v2: Mit Pagination und Meta-Daten

```
{
  "data": [
    {"id": "1", "name": "Laptop", "price": 999.99}
  ],
  "meta": {
    "total": 50,
    "page": 1,
    "perPage": 10,
    "totalPages": 5
  },
  "links": {
    "self": "/api/v2/products?page=1",
    "next": "/api/v2/products?page=2",
    "last": "/api/v2/products?page=5"
  }
}
```

### 2.1.1.6 Aufgabe 5: Location-Header (5 min)

Bei POST-Requests soll der Location-Header gesetzt werden.

Request

POST /api/v1/products

Content-Type: application/json

```
{"name": "Neues Produkt", "price": 49.99}
```

Response

HTTP/1.1 201 Created

Location: /api/v1/products/123

Content-Type: application/json

```
{"id": "123", "name": "Neues Produkt", "price": 49.99, ...}
```

### 2.1.1.7 Testen

# Alle Produkte

```
curl http://localhost:8080/api/v1/products
```

```

# Ein Produkt
curl http://localhost:8080/api/v1/products/1

# Produkt erstellen
curl -X POST http://localhost:8080/api/v1/products \
  -H "Content-Type: application/json" \
  -d '{"name": "Tablet", "price": 399.99, "categoryId": "electronics"}'

# Produkt aktualisieren (PATCH)
curl -X PATCH http://localhost:8080/api/v1/products/1 \
  -H "Content-Type: application/json" \
  -d '{"price": 899.99}'

# Produkt ersetzen (PUT)
curl -X PUT http://localhost:8080/api/v1/products/1 \
  -H "Content-Type: application/json" \
  -d '{"name": "Laptop Pro", "price": 1299.99, "categoryId": "electronics",
    ↪   "stock": 50}'

# Produkt löschen
curl -X DELETE http://localhost:8080/api/v1/products/1

# Produkte einer Kategorie
curl http://localhost:8080/api/v1/categories/electronics/products

# API v2 mit Pagination
curl "http://localhost:8080/api/v2/products?page=1&perPage=10"

# Location-Header prüfen
curl -i -X POST http://localhost:8080/api/v1/products \
  -H "Content-Type: application/json" \
  -d '{"name": "Test"}'

```

### 2.1.1.8 Abgabe-Checkliste

- ☐ Alle CRUD-Endpunkte für Produkte
- ☐ Korrekte HTTP-Methoden
- ☐ Korrekte Statuscodes (200, 201, 204, 404)
- ☐ Verschachtelte Ressourcen funktionieren
- ☐ Zwei API-Versionen (v1, v2)
- ☐ Location-Header bei POST
- ☐ Konsistente Response-Struktur

## 2.1.2 Lösung

### 2.1.2.1 Aufgabe 1: API-Design

Produkte

Aktion	Methode	Pfad	Statuscode
Alle Produkte	GET	/api/v1/products	200

Aktion	Methode	Pfad	Statuscode
Ein Produkt	GET	/api/v1/products/:id	200 / 404
Erstellen	POST	/api/v1/products	201
Ersetzen	PUT	/api/v1/products/:id	200 / 404
Aktualisieren	PATCH	/api/v1/products/:id	200 / 404
Löschen	DELETE	/api/v1/products/:id	204 / 404

### Kategorien

Aktion	Methode	Pfad	Statuscode
Alle Kategorien	GET	/api/v1/categories	200
Eine Kategorie	GET	/api/v1/categories/:id	200 / 404
Produkte einer Kategorie	GET	/api/v1/categories/:id/products	200 / 404

### Bestellungen

Aktion	Methode	Pfad	Statuscode
Alle Bestellungen	GET	/api/v1/orders	200
Eine Bestellung	GET	/api/v1/orders/:id	200 / 404
Bestellung erstellen	POST	/api/v1/orders	201
Positionen einer Bestellung	GET	/api/v1/orders/:id/items	200 / 404

#### 2.1.2.2 Vollständige Lösung

```
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';

// =====
// Models
// =====

class Product {
  final String id;
  final String name;
  final String description;
  final double price;
  final String categoryId;
  final int stock;
  final DateTime createdAt;
```

```

Product({
    required this.id,
    required this.name,
    this.description = '',
    required this.price,
    required this.categoryId,
    this.stock = 0,
    required this.createdAt,
});

Map<String, dynamic> toJson() => {
    'id': id,
    'name': name,
    'description': description,
    'price': price,
    'categoryId': categoryId,
    'stock': stock,
    'createdAt': createdAt.toIso8601String(),
};

Product copyWith({
    String? name,
    String? description,
    double? price,
    String? categoryId,
    int? stock,
}) => Product(
    id: id,
    name: name ?? this.name,
    description: description ?? this.description,
    price: price ?? this.price,
    categoryId: categoryId ?? this.categoryId,
    stock: stock ?? this.stock,
    createdAt: createdAt,
);
}

class Category {
    final String id;
    final String name;

    Category({required this.id, required this.name});

    Map<String, dynamic> toJson() => {'id': id, 'name': name};
}

// =====
// Data Store
// =====

```

```

final _products = <String, Product>{};
final _categories = <String, Category>{};
var _nextProductId = 1;

void _seedData() {
  _categories['electronics'] = Category(id: 'electronics', name: 'Elektronik');
  _categories['clothing'] = Category(id: 'clothing', name: 'Kleidung');

  _products['1'] = Product(
    id: '1', name: 'Laptop', price: 999.99,
    categoryId: 'electronics', stock: 50, createdAt: DateTime.now(),
  );
  _products['2'] = Product(
    id: '2', name: 'Smartphone', price: 599.99,
    categoryId: 'electronics', stock: 100, createdAt: DateTime.now(),
  );
  _products['3'] = Product(
    id: '3', name: 'T-Shirt', price: 29.99,
    categoryId: 'clothing', stock: 200, createdAt: DateTime.now(),
  );
  _nextProductId = 4;
}

// =====
// Main
// =====

void main() async {
  _seedData();

  final app = Router();

  // API Versionen
  app.mount('/api/v1', v1Router().call);
  app.mount('/api/v2', v2Router().call);

  // Health
  app.get('/health', (r) => jsonResponse({'status': 'ok'}));

  final handler = Pipeline()
    .addMiddleware(logRequests())
    .addHandler(app.call);

  await shelf_io.serve(handler, 'localhost', 8080);
  print('Server: http://localhost:8080');
}

// =====
// API v1
// =====

```

```
Router v1Router() {
    final router = Router();

    // Products CRUD
    router.get('/products', _v1ListProducts);
    router.get('/products/<id>', _v1GetProduct);
    router.post('/products', _v1CreateProduct);
    router.put('/products/<id>', _v1ReplaceProduct);
    router.patch('/products/<id>', _v1UpdateProduct);
    router.delete('/products/<id>', _v1DeleteProduct);

    // Categories
    router.get('/categories', _listCategories);
    router.get('/categories/<id>', _getCategory);
    router.get('/categories/<id>/products', _getCategoryProducts);

    return router;
}

Response _v1ListProducts(Request request) {
    return jsonResponse({
        'products': _products.values.map((p) => p.toJson()).toList(),
    });
}

Response _v1GetProduct(Request request, String id) {
    final product = _products[id];
    if (product == null) {
        return jsonResponse({'error': 'Product not found'}, statusCode: 404);
    }
    return jsonResponse(product.toJson());
}

Future<Response> _v1CreateProduct(Request request) async {
    final body = jsonDecode(await request.readAsString());

    final id = '${_nextProductId++}';
    final product = Product(
        id: id,
        name: body['name'] ?? '',
        description: body['description'] ?? '',
        price: (body['price'] ?? 0).toDouble(),
        categoryId: body['categoryId'] ?? '',
        stock: body['stock'] ?? 0,
        createdAt: DateTime.now(),
    );

    _products[id] = product;

    // 201 Created mit Location-Header
    return Response(
```

```
201,
body: jsonEncode(product.toJson()),
headers: {
  'content-type': 'application/json',
  'location': '/api/v1/products/$id',
},
);
}

Future<Response> _v1ReplaceProduct(Request request, String id) async {
  if (!_products.containsKey(id)) {
    return jsonResponse({'error': 'Product not found'}, statusCode: 404);
  }

  final body = jsonDecode(await request.readAsString());
  final existing = _products[id]!;

  // PUT: Komplettes Ersetzen - alle Felder müssen vorhanden sein
  final product = Product(
    id: id,
    name: body['name'] ?? '',
    description: body['description'] ?? '',
    price: (body['price'] ?? 0).toDouble(),
    categoryId: body['categoryId'] ?? '',
    stock: body['stock'] ?? 0,
    createdAt: existing.createdAt,
  );

  _products[id] = product;
  return jsonResponse(product.toJson());
}

Future<Response> _v1UpdateProduct(Request request, String id) async {
  final product = _products[id];
  if (product == null) {
    return jsonResponse({'error': 'Product not found'}, statusCode: 404);
  }

  final body = jsonDecode(await request.readAsString());

  // PATCH: Nur übergebene Felder aktualisieren
  final updated = product.copyWith(
    name: body['name'],
    description: body['description'],
    price: body['price']?.toDouble(),
    categoryId: body['categoryId'],
    stock: body['stock'],
  );

  _products[id] = updated;
  return jsonResponse(updated.toJson());
}
```

```
}

Response _v1DeleteProduct(Request request, String id) {
    if (!_products.containsKey(id)) {
        return jsonResponse({'error': 'Product not found'}, statusCode: 404);
    }

    _products.remove(id);
    return Response(204); // No Content
}

// Categories
Response _listCategories(Request request) {
    return jsonResponse({
        'categories': _categories.values.map((c) => c.toJson()).toList(),
    });
}

Response _getCategory(Request request, String id) {
    final category = _categories[id];
    if (category == null) {
        return jsonResponse({'error': 'Category not found'}, statusCode: 404);
    }
    return jsonResponse(category.toJson());
}

Response _getCategoryProducts(Request request, String id) {
    final category = _categories[id];
    if (category == null) {
        return jsonResponse({'error': 'Category not found'}, statusCode: 404);
    }

    final products = _products.values
        .where((p) => p.categoryId == id)
        .map((p) => p.toJson())
        .toList();

    return jsonResponse({
        'category': category.toJson(),
        'products': products,
        'total': products.length,
    });
}

// =====
// API v2 (mit Pagination)
// =====

Router v2Router() {
    final router = Router();
    router.get('/products', _v2ListProducts);
}
```

```

// Weitere v2 Endpunkte...
return router;
}

Response _v2ListProducts(Request request) {
    final params = request.url.queryParameters;
    final page = int.tryParse(params['page'] ?? '1') ?? 1;
    final perPage = int.tryParse(params['perPage'] ?? '10') ?? 10;

    final allProducts = _products.values.toList();
    final total = allProducts.length;
    final totalPages = (total / perPage).ceil();

    final start = (page - 1) * perPage;
    final end = start + perPage;
    final pageProducts = allProducts
        .skip(start)
        .take(perPage)
        .map((p) => p.toJson())
        .toList();

    return jsonResponse({
        'data': pageProducts,
        'meta': {
            'total': total,
            'page': page,
            'perPage': perPage,
            'totalPages': totalPages,
        },
        'links': {
            'self': '/api/v2/products?page=$page&perPage=$perPage',
            if (page > 1) 'prev': '/api/v2/products?page=${page - 1}&perPage=$perPage',
            if (page < totalPages) 'next': '/api/v2/products?page=${page +
↵ 1}&perPage=$perPage',
            'first': '/api/v2/products?page=1&perPage=$perPage',
            'last': '/api/v2/products?page=$totalPages&perPage=$perPage',
        },
    });
}

// =====
// Helper
// =====

Response jsonResponse(Object? data, {int statusCode = 200}) {
    return Response(
        statusCode,
        body: jsonEncode(data),
        headers: {'content-type': 'application/json'},
    );
}

```

### 2.1.2.3 Test-Befehle

```
# v1: Alle Produkte
curl http://localhost:8080/api/v1/products

# v1: Ein Produkt
curl http://localhost:8080/api/v1/products/1

# v1: Produkt erstellen (mit Location-Header)
curl -i -X POST http://localhost:8080/api/v1/products \
  -H "Content-Type: application/json" \
  -d '{"name": "Tablet", "price": 399.99, "categoryId": "electronics"}'

# v1: PATCH (teilweise aktualisieren)
curl -X PATCH http://localhost:8080/api/v1/products/1 \
  -H "Content-Type: application/json" \
  -d '{"price": 899.99}'

# v1: PUT (komplett ersetzen)
curl -X PUT http://localhost:8080/api/v1/products/1 \
  -H "Content-Type: application/json" \
  -d '{"name": "Laptop Pro", "price": 1299.99, "categoryId": "electronics",
    ↪   "stock": 25}'

# v1: Löschen
curl -X DELETE http://localhost:8080/api/v1/products/3

# Kategorien
curl http://localhost:8080/api/v1/categories
curl http://localhost:8080/api/v1/categories/electronics/products

# v2: Mit Pagination
curl "http://localhost:8080/api/v2/products?page=1&perPage=2"

# 404 testen
curl http://localhost:8080/api/v1/products/999
```

### 2.1.2.4 Wichtige Erkenntnisse

PUT vs. PATCH

```
// PUT: Komplett ersetzen
// Alle Felder werden überschrieben (auch mit Default-Werten)
final product = Product(
  id: id,
  name: body['name'] ?? '', // Leerer String wenn nicht angegeben!
  ...
);
```

```
// PATCH: Nur übergebene Felder
// copyWith überschreibt nur wenn Wert != null
final updated = product.copyWith(
  name: body['name'], // null = keine Änderung
  ...
);
```

Location-Header bei 201

```
return Response(
  201, // Created
  body: jsonEncode(product.toJson()),
  headers: {
    'content-type': 'application/json',
    'location': '/api/v1/products/$id', // URL der neuen Ressource
  },
);
```

204 No Content bei DELETE

```
// Kein Body bei erfolgreicher Löschung
return Response(204);
```

## 2.1.3 Ressourcen

### 2.1.3.1 Offizielle Dokumentation & Standards

- HTTP/1.1 Specification (RFC 7231) - HTTP-Methoden
- HTTP Status Codes (MDN)
- REST API Tutorial - Umfassende Einführung

### 2.1.3.2 Best Practice Guides

- Microsoft REST API Guidelines
- Google API Design Guide
- JSON API Specification

### 2.1.3.3 Cheat Sheet: HTTP-Methoden

Methode	CRUD	Idempotent	Safe	Body
GET	Read	Ja	Ja	Nein
POST	Create	Nein	Nein	Ja
PUT	Replace	Ja	Nein	Ja
PATCH	Update	Nein	Nein	Ja
DELETE	Delete	Ja	Nein	Nein

### 2.1.3.4 Cheat Sheet: Statuscodes

2xx Erfolg:

- |                |                        |
|----------------|------------------------|
| 200 OK         | - Erfolgreiche Anfrage |
| 201 Created    | - Ressource erstellt   |
| 204 No Content | - Erfolg ohne Body     |

**4xx Client-Fehler:**

400 Bad Request	- Ungültige Anfrage
401 Unauthorized	- Nicht authentifiziert
403 Forbidden	- Keine Berechtigung
404 Not Found	- Nicht gefunden
409 Conflict	- Konflikt/Duplikat
422 Unprocessable	- Semantisch ungültig

**5xx Server-Fehler:**

500 Internal Error	- Serverfehler
503 Unavailable	- Service nicht verfügbar

**2.1.3.5 Cheat Sheet: URL-Design**

## # Ressourcen (Nomen, Plural)

GET	/users	# Liste
GET	/users/123	# Einzeln
POST	/users	# Erstellen
PUT	/users/123	# Ersetzen
PATCH	/users/123	# Aktualisieren
DELETE	/users/123	# Löschen

## # Verschachtelte Ressourcen

```
GET /users/123/orders
GET /users/123/orders/456
```

## # Filter (Query-Parameter)

```
GET /users?status=active&sort=name
```

## # Versionierung

```
GET /api/v1/users
GET /api/v2/users
```

**2.1.3.6 Cheat Sheet: Response-Struktur**

```
// Einzelne Ressource
{
  "id": "123",
  "name": "Max",
  "email": "max@example.com"
}

// Liste
{
  "data": [...],
  "meta": {
    "total": 100,
    "page": 1,
    "perPage": 10
  }
}
```

```
}

// Fehler
{
  "error": {
    "code": "NOT_FOUND",
    "message": "User not found"
  }
}
```

### 2.1.3.7 Naming Conventions

```
# GUT
/api/users
/api/user-profiles
/api/order-items

# SCHLECHT
/api/getUsers      (Verb)
/api/User          (Singular, CamelCase)
/api/user_profile  (snake_case mit Singular)
```

### 2.1.3.8 Dart: Basis-Router

```
Router apiRouter() {
  final router = Router();

  // CRUD für Users
  router.get('/users', listUsers);
  router.get('/users/<id>', getUser);
  router.post('/users', createUser);
  router.put('/users/<id>', replaceUser);
  router.patch('/users/<id>', updateUser);
  router.delete('/users/<id>', deleteUser);

  return router;
}
```

### 2.1.3.9 Tools

- Postman - API Testing
- Insomnia - REST Client
- Swagger Editor - API Design
- JSON Formatter - JSON validieren

## 2.2 Einheit 6.2: JSON Serialisierung

### 2.2.0.1 Lernziele

Nach dieser Einheit kannst du: - JSON in Dart parsen und generieren - Model-Klassen mit `fromJson/toJson` erstellen - Verschachtelte Objekte und Listen serialisieren - Das

json\_serializable Package für Code-Generierung nutzen

### 2.2.0.2 JSON Grundlagen in Dart

dart:convert

```
import 'dart:convert';

void main() {
  // JSON String -> Dart Map
  final jsonString = '{"name": "Max", "age": 30}';
  final map = jsonDecode(jsonString) as Map<String, dynamic>;
  print(map['name']); // Max

  // Dart Map -> JSON String
  final data = {'name': 'Anna', 'age': 25};
  final json = jsonEncode(data);
  print(json); // {"name":"Anna","age":25}
}
```

JSON-Typen in Dart

JSON	Dart
string	String
number	int oder double
boolean	bool
null	null
array	List<dynamic>
object	Map<String, dynamic>

### 2.2.0.3 Model-Klassen

Einfache Model-Klasse

```
class User {
  final String id;
  final String name;
  final String email;
  final int age;

  User({
    required this.id,
    required this.name,
    required this.email,
    required this.age,
  });

  // JSON -> User
  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      id: json['id'] as String,
      name: json['name'] as String,
```

```

        email: json['email'] as String,
        age: json['age'] as int,
    );
}

// User -> JSON
Map<String, dynamic> toJson() {
    return {
        'id': id,
        'name': name,
        'email': email,
        'age': age,
    };
}
}

```

Verwendung

```

// JSON parsen
final jsonString = '{"id": "1", "name": "Max", "email": "max@example.com",
↪  ↪  "age": 30}';
final map = jsonDecode(jsonString) as Map<String, dynamic>;
final user = User.fromJson(map);

// Zu JSON serialisieren
final json = jsonEncode(user.toJson());

```

#### 2.2.0.4 Optionale Felder

Mit Null-Safety

```

class Product {
    final String id;
    final String name;
    final double price;
    final String? description; // Optional
    final String? imageUrl;    // Optional

    Product({
        required this.id,
        required this.name,
        required this.price,
        this.description,
        this.imageUrl,
    });

    factory Product.fromJson(Map<String, dynamic> json) {
        return Product(
            id: json['id'] as String,
            name: json['name'] as String,
            price: (json['price'] as num).toDouble(),
            description: json['description'] as String?,

```

```

        imageUrl: json['image_url'] as String?,
    );
}

Map<String, dynamic> toJson() {
    return {
        'id': id,
        'name': name,
        'price': price,
        if (description != null) 'description': description,
        if (imageUrl != null) 'image_url': imageUrl,
    };
}
}

```

### Default-Werte

```

factory Product.fromJson(Map<String, dynamic> json) {
    return Product(
        id: json['id'] as String,
        name: json['name'] as String,
        price: (json['price'] as num?)?.toDouble() ?? 0.0,
        description: json['description'] as String? ?? '',
        stock: json['stock'] as int? ?? 0,
    );
}

```

### 2.2.0.5 Verschachtelte Objekte

#### Einfache Verschachtelung

```

class Address {
    final String street;
    final String city;
    final String zipCode;

    Address({required this.street, required this.city, required this.zipCode});

    factory Address.fromJson(Map<String, dynamic> json) {
        return Address(
            street: json['street'] as String,
            city: json['city'] as String,
            zipCode: json['zip_code'] as String,
        );
    }

    Map<String, dynamic> toJson() => {
        'street': street,
        'city': city,
        'zip_code': zipCode,
    };
}

```

```
class User {
    final String id;
    final String name;
    final Address address; // Verschachteltes Objekt

    User({required this.id, required this.name, required this.address});

    factory User.fromJson(Map<String, dynamic> json) {
        return User(
            id: json['id'] as String,
            name: json['name'] as String,
            address: Address.fromJson(json['address'] as Map<String, dynamic>),
        );
    }

    Map<String, dynamic> toJson() => {
        'id': id,
        'name': name,
        'address': address.toJson(),
    };
}
```

JSON-Beispiel

```
{
  "id": "123",
  "name": "Max Mustermann",
  "address": {
    "street": "Hauptstraße 1",
    "city": "Berlin",
    "zip_code": "10115"
  }
}
```

### 2.2.0.6 Listen serialisieren

Liste von Objekten

```
class Order {
    final String id;
    final List<OrderItem> items;
    final double total;

    Order({required this.id, required this.items, required this.total});

    factory Order.fromJson(Map<String, dynamic> json) {
        return Order(
            id: json['id'] as String,
            items: (json['items'] as List<dynamic>)
                .map((item) => OrderItem.fromJson(item as Map<String, dynamic>))
                .toList(),
        );
    }
}
```

```

        total: (json['total'] as num).toDouble(),
    );
}

Map<String, dynamic> toJson() => {
    'id': id,
    'items': items.map((item) => item.toJson()).toList(),
    'total': total,
};
}

class OrderItem {
    final String productId;
    final int quantity;
    final double price;

    OrderItem({required this.productId, required this.quantity, required
↪ this.price});

    factory OrderItem.fromJson(Map<String, dynamic> json) {
        return OrderItem(
            productId: json['product_id'] as String,
            quantity: json['quantity'] as int,
            price: (json['price'] as num).toDouble(),
        );
    }

    Map<String, dynamic> toJson() => {
        'product_id': productId,
        'quantity': quantity,
        'price': price,
    };
}

```

### 2.2.0.7 DateTime-Serialisierung

ISO 8601 Format

```

class Event {
    final String id;
    final String title;
    final DateTime startTime;
    final DateTime? endTime;

    Event({
        required this.id,
        required this.title,
        required this.startTime,
        this.endTime,
    });
}

```

```
factory Event.fromJson(Map<String, dynamic> json) {
  return Event(
    id: json['id'] as String,
    title: json['title'] as String,
    startTime: DateTime.parse(json['start_time'] as String),
    endTime: json['end_time'] != null
      ? DateTime.parse(json['end_time'] as String)
      : null,
  );
}

Map<String, dynamic> toJson() => {
  'id': id,
  'title': title,
  'start_time': startTime.toIso8601String(),
  if (endTime != null) 'end_time': endTime!.toIso8601String(),
};
}
```

#### 2.2.0.8 Enums serialisieren

```
enum OrderStatus { pending, processing, shipped, delivered, cancelled }

class Order {
  final String id;
  final OrderStatus status;

  Order({required this.id, required this.status});

  factory Order.fromJson(Map<String, dynamic> json) {
    return Order(
      id: json['id'] as String,
      status: OrderStatus.values.firstWhere(
        (e) => e.name == json['status'],
        orElse: () => OrderStatus.pending,
      ),
    );
  }

  Map<String, dynamic> toJson() => {
    'id': id,
    'status': status.name,
  };
}
```

#### 2.2.0.9 json\_serializable (Code-Generierung)

Für größere Projekte empfiehlt sich die automatische Code-Generierung.

Setup

```
# pubspec.yaml
dependencies:
  json_annotation: ^4.8.0

dev_dependencies:
  build_runner: ^2.4.0
  json_serializable: ^6.7.0
```

Model mit Annotations

```
import 'package:json_annotation/json_annotation.dart';

part 'user.g.dart';

@JsonSerializable()
class User {
  final String id;
  final String name;
  final String email;

  @JsonKey(name: 'created_at')
  final DateTime createdAt;

  @JsonKey(includeIfNull: false)
  final String? bio;

  User({
    required this.id,
    required this.name,
    required this.email,
    required this.createdAt,
    this.bio,
  });

  factory User.fromJson(Map<String, dynamic> json) => _$UserFromJson(json);
  Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

Code generieren

```
# Einmalig generieren
dart run build_runner build

# Watch-Modus (automatisch bei Änderungen)
dart run build_runner watch
```

### 2.2.0.10 Fehlerbehandlung

Sichere Konvertierung

```
class User {
  factory User.fromJson(Map<String, dynamic> json) {
```

```

try {
    return User(
        id: json['id'] as String? ?? '',
        name: json['name'] as String? ?? 'Unknown',
        age: _parseAge(json['age']),
    );
} catch (e) {
    throw FormatException('Invalid User JSON: $e');
}

static int _parseAge(dynamic value) {
    if (value is int) return value;
    if (value is String) return int.tryParse(value) ?? 0;
    return 0;
}
}

```

### 2.2.0.11 Zusammenfassung

Aufgabe	Lösung
JSON parsen	<code>jsonDecode(string)</code>
JSON generieren	<code>jsonEncode(map)</code>
Model von JSON	<code>factory Model.fromJson(Map)</code>
Model zu JSON	<code>Map toJson()</code>
Verschachtelt	Rekursiv <code>fromJson/toJson</code> aufrufen
Listen	<code>.map().toList()</code>
DateTime	<code>DateTime.parse()</code> / <code>.toIso8601String()</code>
Automatisch	<code>json_serializable</code> Package

### 2.2.0.12 Nächste Schritte

In der nächsten Einheit lernst du **Request Body Parsing**: Wie du JSON-Daten aus HTTP-Requests liest und verarbeitest.

## 2.2.1 Übung

### 2.2.1.1 Ziel

Erstelle Model-Klassen mit JSON-Serialisierung für ein Blog-System.

### 2.2.1.2 Aufgabe 1: Einfache Model-Klasse (15 min)

Erstelle eine `Author`-Klasse.

JSON-Format

```

{
  "id": "author-123",
  "name": "Max Mustermann",
  "email": "max@example.com",

```

```
"bio": "Dart-Entwickler seit 2020",  
"avatar_url": "https://example.com/avatar.jpg"  
}
```

Anforderungen

- `id`, `name`, `email` sind Pflichtfelder
- `bio` und `avatarUrl` sind optional
- Beachte: JSON verwendet `snake_case`, Dart `camelCase`

### 2.2.1.3 Aufgabe 2: Model mit DateTime (10 min)

Erstelle eine `Post`-Klasse für Blog-Artikel.

JSON-Format

```
{  
  "id": "post-456",  
  "title": "Einführung in Dart",  
  "content": "Dart ist eine moderne Programmiersprache...",  
  "author_id": "author-123",  
  "published_at": "2024-01-15T10:30:00Z",  
  "updated_at": "2024-01-16T14:00:00Z",  
  "tags": ["dart", "programming", "tutorial"],  
  "view_count": 1250  
}
```

Anforderungen

- `updatedAt` ist optional (kann null sein)
- `tags` ist eine Liste von Strings
- `viewCount` hat Default-Wert 0

### 2.2.1.4 Aufgabe 3: Verschachtelte Objekte (15 min)

Erstelle eine `Comment`-Klasse und erweitere `Post`.

Comment JSON

```
{  
  "id": "comment-789",  
  "post_id": "post-456",  
  "author": {  
    "id": "author-999",  
    "name": "Anna Schmidt",  
    "email": "anna@example.com"  
  },  
  "content": "Toller Artikel!",  
  "created_at": "2024-01-15T12:00:00Z"  
}
```

Post mit eingebettetem Author

```
{  
  "id": "post-456",
```

```
"title": "Einführung in Dart",
"author": {
  "id": "author-123",
  "name": "Max Mustermann",
  "email": "max@example.com"
},
"content": "...",
"published_at": "2024-01-15T10:30:00Z"
}
```

Erstelle eine neue Klasse `PostWithAuthor` oder erweitere `Post`.

#### 2.2.1.5 Aufgabe 4: Listen von Objekten (10 min)

Erstelle eine Funktion zum Parsen einer API-Response.

API-Response

```
{
  "posts": [
    {"id": "1", "title": "Post 1", ...},
    {"id": "2", "title": "Post 2", ...}
  ],
  "meta": {
    "total": 50,
    "page": 1,
    "per_page": 10
  }
}
```

Funktion

```
class PostListResponse {
  final List<Post> posts;
  final int total;
  final int page;
  final int perPage;

  // fromJson implementieren
}
```

#### 2.2.1.6 Aufgabe 5: Enum-Serialisierung (10 min)

Füge einen Status zu Posts hinzu.

Status-Enum

```
enum PostStatus { draft, published, archived }
```

JSON

```
{
  "id": "post-456",
  "title": "...",
```

```
"status": "published"
}
```

### 2.2.1.7 Bonus: json\_serializable

Erstelle die Models mit json\_serializable und Code-Generierung.

```
@JsonSerializable()
class Author {
  final String id;
  final String name;

  @JsonKey(name: 'avatar_url')
  final String? avatarUrl;

  Author({required this.id, required this.name, this.avatarUrl});

  factory Author.fromJson(Map<String, dynamic> json) => _$AuthorFromJson(json);
  Map<String, dynamic> toJson() => _$AuthorToJson(this);
}
```

### 2.2.1.8 Testen

```
void main() {
  // Author testen
  final authorJson = {
    'id': 'author-1',
    'name': 'Max',
    'email': 'max@example.com',
    'bio': 'Developer',
    'avatar_url': 'https://...',
  };

  final author = Author.fromJson(authorJson);
  print(author.name);
  print(jsonEncode(author.toJson()));

  // Post testen
  final postJson = {
    'id': 'post-1',
    'title': 'Hello',
    'content': 'World',
    'author_id': 'author-1',
    'published_at': '2024-01-15T10:00:00Z',
    'tags': ['dart', 'json'],
    'view_count': 100,
  };

  final post = Post.fromJson(postJson);
  print(post.title);
}
```

```
print(post.tags);

// Verschachtelt testen
final commentJson = {
  'id': 'comment-1',
  'post_id': 'post-1',
  'author': {'id': 'a-1', 'name': 'Anna', 'email': 'a@b.com'},
  'content': 'Nice!',
  'created_at': '2024-01-15T12:00:00Z',
};

final comment = Comment.fromJson(commentJson);
print(comment.author.name);
}
```

### 2.2.1.9 Abgabe-Checkliste

- ☐ Author-Klasse mit fromJson/toJson
- ☐ Post-Klasse mit DateTime und Listen
- ☐ Comment-Klasse mit verschachteltem Author
- ☐ PostListResponse für API-Response
- ☐ PostStatus-Enum serialisiert
- ☐ (Bonus) json\_serializable Setup

## 2.2.2 Lösung

### 2.2.2.1 Vollständige Lösung

```
import 'dart:convert';

// =====
// Aufgabe 1: Author
// =====

class Author {
  final String id;
  final String name;
  final String email;
  final String? bio;
  final String? avatarUrl;

  Author({
    required this.id,
    required this.name,
    required this.email,
    this.bio,
    this.avatarUrl,
  });

  factory Author.fromJson(Map<String, dynamic> json) {
    return Author(
```

```
        id: json['id'] as String,
        name: json['name'] as String,
        email: json['email'] as String,
        bio: json['bio'] as String?,
        avatarUrl: json['avatar_url'] as String?,
    );
}

Map<String, dynamic> toJson() {
    return {
        'id': id,
        'name': name,
        'email': email,
        if (bio != null) 'bio': bio,
        if (avatarUrl != null) 'avatar_url': avatarUrl,
    };
}
}

// =====
// Aufgabe 2: Post mit DateTime
// =====

class Post {
    final String id;
    final String title;
    final String content;
    final String authorId;
    final DateTime publishedAt;
    final DateTime? updatedAt;
    final List<String> tags;
    final int viewCount;
    final PostStatus status;

    Post({
        required this.id,
        required this.title,
        required this.content,
        required this.authorId,
        required this.publishedAt,
        this.updatedAt,
        this.tags = const [],
        this.viewCount = 0,
        this.status = PostStatus.draft,
    });

    factory Post.fromJson(Map<String, dynamic> json) {
        return Post(
            id: json['id'] as String,
            title: json['title'] as String,
            content: json['content'] as String,
```

```

        authorId: json['author_id'] as String,
        publishedAt: DateTime.parse(json['published_at'] as String),
        updatedAt: json['updated_at'] != null
            ? DateTime.parse(json['updated_at'] as String)
            : null,
        tags: (json['tags'] as List<dynamic>?)
            ?.map((e) => e as String)
            .toList() ?? [],
        viewCount: json['view_count'] as int? ?? 0,
        status: PostStatus.values.firstWhere(
            (e) => e.name == json['status'],
            orElse: () => PostStatus.draft,
        ),
    );
}

Map<String, dynamic> toJson() {
    return {
        'id': id,
        'title': title,
        'content': content,
        'author_id': authorId,
        'published_at': publishedAt.toIso8601String(),
        if (updatedAt != null) 'updated_at': updatedAt!.toIso8601String(),
        'tags': tags,
        'view_count': viewCount,
        'status': status.name,
    };
}
}

// =====
// Aufgabe 5: PostStatus Enum
// =====

enum PostStatus { draft, published, archived }

// =====
// Aufgabe 3: Comment mit verschachteltem Author
// =====

class Comment {
    final String id;
    final String postId;
    final Author author;
    final String content;
    final DateTime createdAt;

    Comment({
        required this.id,
        required this.postId,

```

```
        required this.author,
        required this.content,
        required this.createdAt,
    });

    factory Comment.fromJson(Map<String, dynamic> json) {
        return Comment(
            id: json['id'] as String,
            postId: json['post_id'] as String,
            author: Author.fromJson(json['author'] as Map<String, dynamic>),
            content: json['content'] as String,
            createdAt: DateTime.parse(json['created_at'] as String),
        );
    }

    Map<String, dynamic> toJson() {
        return {
            'id': id,
            'post_id': postId,
            'author': author.toJson(),
            'content': content,
            'created_at': createdAt.toIso8601String(),
        };
    }
}

// =====
// Post mit eingebettetem Author
// =====

class PostWithAuthor {
    final String id;
    final String title;
    final String content;
    final Author author;
    final DateTime publishedAt;
    final List<String> tags;

    PostWithAuthor({
        required this.id,
        required this.title,
        required this.content,
        required this.author,
        required this.publishedAt,
        this.tags = const [],
    });

    factory PostWithAuthor.fromJson(Map<String, dynamic> json) {
        return PostWithAuthor(
            id: json['id'] as String,
            title: json['title'] as String,
```

```
        content: json['content'] as String,
        author: Author.fromJson(json['author'] as Map<String, dynamic>),
        publishedAt: DateTime.parse(json['published_at'] as String),
        tags: (json['tags'] as List<dynamic>?)
            ?.map((e) => e as String)
            .toList() ?? [],
    );
}

Map<String, dynamic> toJson() {
    return {
        'id': id,
        'title': title,
        'content': content,
        'author': author.toJson(),
        'published_at': publishedAt.toIso8601String(),
        'tags': tags,
    };
}
}

// =====
// Aufgabe 4: API Response mit Liste
// =====

class PostListResponse {
    final List<Post> posts;
    final int total;
    final int page;
    final int perPage;

    PostListResponse({
        required this.posts,
        required this.total,
        required this.page,
        required this.perPage,
    });

    factory PostListResponse.fromJson(Map<String, dynamic> json) {
        final meta = json['meta'] as Map<String, dynamic>;

        return PostListResponse(
            posts: (json['posts'] as List<dynamic>)
                .map((e) => Post.fromJson(e as Map<String, dynamic>))
                .toList(),
            total: meta['total'] as int,
            page: meta['page'] as int,
            perPage: meta['per_page'] as int,
        );
    }
}
```

```
Map<String, dynamic> toJson() {
  return {
    'posts': posts.map((p) => p.toJson()).toList(),
    'meta': {
      'total': total,
      'page': page,
      'per_page': perPage,
    },
  };
}

int get totalPages => (total / perPage).ceil();
bool get hasNextPage => page < totalPages;
bool get hasPrevPage => page > 1;
}

// =====
// Test
// =====

void main() {
  // Author testen
  print('=== Author ===');
  final authorJson = {
    'id': 'author-1',
    'name': 'Max Mustermann',
    'email': 'max@example.com',
    'bio': 'Dart Developer',
    'avatar_url': 'https://example.com/avatar.jpg',
  };
  final author = Author.fromJson(authorJson);
  print('Name: ${author.name}');
  print('JSON: ${jsonEncode(author.toJson())}');

  // Post testen
  print('\n=== Post ===');
  final postJson = {
    'id': 'post-1',
    'title': 'Hello Dart',
    'content': 'Dart is awesome!',
    'author_id': 'author-1',
    'published_at': '2024-01-15T10:00:00Z',
    'tags': ['dart', 'tutorial'],
    'view_count': 100,
    'status': 'published',
  };
  final post = Post.fromJson(postJson);
  print('Title: ${post.title}');
  print('Tags: ${post.tags}');
  print('Status: ${post.status}');
```

```
// Comment testen
print('\n=== Comment ===');
final commentJson = {
  'id': 'comment-1',
  'post_id': 'post-1',
  'author': {
    'id': 'author-2',
    'name': 'Anna',
    'email': 'anna@example.com',
  },
  'content': 'Great article!',
  'created_at': '2024-01-15T12:00:00Z',
};
final comment = Comment.fromJson(commentJson);
print('Author: ${comment.author.name}');
print('Content: ${comment.content}');

// PostListResponse testen
print('\n=== PostListResponse ===');
final responseJson = {
  'posts': [postJson, postJson],
  'meta': {
    'total': 50,
    'page': 1,
    'per_page': 10,
  },
};
final response = PostListResponse.fromJson(responseJson);
print('Posts: ${response.posts.length}');
print('Total: ${response.total}');
print('Has next: ${response.hasNextPage}');
}
```

### 2.2.2.2 Ausgabe

=== Author ===

Name: Max Mustermann

JSON: {"id":"author-1","name":"Max Mustermann","email":"max@example.com","bio":"Dart Developer"}

=== Post ===

Title: Hello Dart

Tags: [dart, tutorial]

Status: PostStatus.published

=== Comment ===

Author: Anna

Content: Great article!

=== PostListResponse ===

Posts: 2

Total: 50

Has next: true

### 2.2.2.3 Wichtige Patterns

snake\_case camelCase

```
// JSON: avatar_url -> Dart: avatarUrl
avatarUrl: json['avatar_url'] as String?,

// Dart: avatarUrl -> JSON: avatar_url
'avatar_url': avatarUrl,
```

Optionale Felder

```
// Nur wenn nicht null ausgeben
if (bio != null) 'bio': bio,
```

Listen parsen

```
tags: (json['tags'] as List<dynamic>?)
      ?.map((e) => e as String)
      .toList() ?? [],
```

Enum-Handling

```
// Parse mit Fallback
status: PostStatus.values.firstWhere(
  (e) => e.name == json['status'],
  orElse: () => PostStatus.draft,
),

// Serialize
'status': status.name,
```

## 2.2.3 Ressourcen

### 2.2.3.1 Offizielle Dokumentation

- [dart:convert](#)
- [json\\_serializable](#)
- [json\\_annotation](#)
- [JSON and serialization \(Flutter\)](#)

### 2.2.3.2 Cheat Sheet: Basics

```
import 'dart:convert';

// JSON String -> Map
final map = jsonDecode('{"key": "value"}') as Map<String, dynamic>;

// Map -> JSON String
final json = jsonEncode({'key': 'value'});
```

```
// Pretty Print
final pretty = JsonEncoder.withIndent('  ').convert(data);
```

### 2.2.3.3 Cheat Sheet: Model-Klasse

```
class User {
    final String id;
    final String name;
    final String? email; // Optional

    User({required this.id, required this.name, this.email});

    factory User.fromJson(Map<String, dynamic> json) => User(
        id: json['id'] as String,
        name: json['name'] as String,
        email: json['email'] as String?,
    );

    Map<String, dynamic> toJson() => {
        'id': id,
        'name': name,
        if (email != null) 'email': email,
    };
}
```

### 2.2.3.4 Cheat Sheet: Verschachtelte Objekte

```
factory Parent.fromJson(Map<String, dynamic> json) => Parent(
    child: Child.fromJson(json['child'] as Map<String, dynamic>),
);

Map<String, dynamic> toJson() => {
    'child': child.toJson(),
};
```

### 2.2.3.5 Cheat Sheet: Listen

```
// JSON -> List<Model>
final items = (json['items'] as List)
    .map((e) => Item.fromJson(e as Map<String, dynamic>))
    .toList();

// List<Model> -> JSON
'items': items.map((e) => e.toJson()).toList(),
```

### 2.2.3.6 Cheat Sheet: DateTime

```
// Parse
final date = DateTime.parse(json['date'] as String);

// Serialize
'date': date.toIso8601String(),
```

### 2.2.3.7 Cheat Sheet: Enums

```
// Parse
final status = Status.values.firstWhere(
  (e) => e.name == json['status'],
  orElse: () => Status.unknown,
);

// Serialize
'status': status.name,
```

### 2.2.3.8 Cheat Sheet: json\_serializable

```
# pubspec.yaml
dependencies:
  json_annotation: ^4.8.0
dev_dependencies:
  build_runner: ^2.4.0
  json_serializable: ^6.7.0
```

```
import 'package:json_annotation/json_annotation.dart';
part 'model.g.dart';

@JsonSerializable()
class Model {
  final String id;

  @JsonKey(name: 'created_at')
  final DateTime createdAt;

  @JsonKey(includeIfNull: false)
  final String? optional;

  Model({required this.id, required this.createdAt, this.optional});

  factory Model.fromJson(Map<String, dynamic> json) => _$ModelFromJson(json);
  Map<String, dynamic> toJson() => _$ModelToJson(this);
}
```

```
# Code generieren
dart run build_runner build
```

```
# Watch mode
```

```
dart run build_runner watch
```

### 2.2.3.9 Typsichere Konvertierung

```
// Sicher zu int
int parseId(dynamic value) {
  if (value is int) return value;
  if (value is String) return int.tryParse(value) ?? 0;
  return 0;
}

// Sicher zu double
double parsePrice(dynamic value) {
  if (value is double) return value;
  if (value is int) return value.toDouble();
  if (value is String) return double.tryParse(value) ?? 0.0;
  return 0.0;
}
```

### 2.2.3.10 JSON Tools

- JSON Formatter
- JSON to Dart
- quicktype - JSON zu Code

## 2.3 Einheit 6.3: Request Body Parsing

### 2.3.0.1 Lernziele

Nach dieser Einheit kannst du: - Request Bodies aus HTTP-Anfragen auslesen - JSON-Daten aus POST/PUT/PATCH-Requests parsen - Content-Type Header korrekt verarbeiten - Fehlerhafte Request Bodies behandeln

### 2.3.0.2 Request Body Grundlagen

HTTP Request Anatomie

```
POST /api/users HTTP/1.1
Host: localhost:8080
Content-Type: application/json
Content-Length: 45
```

```
{"name": "Max", "email": "max@example.com"}
```

Ein HTTP-Request besteht aus: 1. **Request Line**: Methode, Pfad, HTTP-Version 2. **Headers**: Metadaten (Content-Type, Authorization, etc.) 3. **Body**: Die eigentlichen Daten (bei POST, PUT, PATCH)

### 2.3.0.3 Body lesen mit Shelf

Einfaches Body-Lesen

```
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';

Future<Response> handlePost(Request request) async {
  // Body als String lesen
  final bodyString = await request.readAsString();
  print('Received: $bodyString');

  // JSON parsen
  final json = jsonDecode(bodyString) as Map<String, dynamic>;

  return Response.ok('Received: ${json['name']}');
}
```

Body kann nur einmal gelesen werden!

```
Future<Response> handleRequest(Request request) async {
  // Erstes Lesen funktioniert
  final body1 = await request.readAsString();

  // Zweites Lesen gibt leeren String!
  final body2 = await request.readAsString();
  print(body2); // ""

  return Response.ok('OK');
}
```

**Wichtig:** Der Request Body ist ein Stream und kann nur einmal konsumiert werden!

#### 2.3.0.4 JSON Body Parsing

Sichere JSON-Verarbeitung

```
Future<Response> createUser(Request request) async {
  // 1. Content-Type prüfen
  final contentType = request.headers['content-type'];
  if (contentType == null || !contentType.contains('application/json')) {
    return Response(
      415, // Unsupported Media Type
      body: jsonEncode({'error': 'Content-Type must be application/json'}),
      headers: {'content-type': 'application/json'},
    );
  }

  // 2. Body lesen
  final bodyString = await request.readAsString();

  // 3. Leeren Body abfangen
  if (bodyString.isEmpty) {
    return Response(
      400,
```

```

        body: jsonEncode({'error': 'Request body is empty'}),
        headers: {'content-type': 'application/json'},
    );
}

// 4. JSON parsen (mit Fehlerbehandlung)
Map<String, dynamic> json;
try {
    json = jsonDecode(bodyString) as Map<String, dynamic>;
} on FormatException catch (e) {
    return Response(
        400,
        body: jsonEncode({'error': 'Invalid JSON: ${e.message}'}),
        headers: {'content-type': 'application/json'},
    );
}

// 5. Daten verarbeiten
final name = json['name'] as String?;
final email = json['email'] as String?;

if (name == null || email == null) {
    return Response(
        400,
        body: jsonEncode({'error': 'Missing required fields: name, email'}),
        headers: {'content-type': 'application/json'},
    );
}

// Erfolg
return Response(
    201,
    body: jsonEncode({'id': '123', 'name': name, 'email': email}),
    headers: {'content-type': 'application/json'},
);
}

```

### 2.3.0.5 Body Parser Middleware

Wiederverwendbare Middleware

```

import 'dart:convert';
import 'package:shelf/shelf.dart';

/// Middleware die JSON Body parst und in request.context speichert
Middleware jsonBodyParser() {
    return (Handler innerHandler) {
        return (Request request) async {
            // Nur bei relevanten Methoden
            if (!['POST', 'PUT', 'PATCH'].contains(request.method)) {
                return innerHandler(request);
            }

```

```

    }

    // Content-Type prüfen
    final contentType = request.headers['content-type'] ?? '';
    if (!contentType.contains('application/json')) {
        return innerHandler(request);
    }

    // Body lesen und parsen
    final bodyString = await request.readAsString();

    if (bodyString.isEmpty) {
        // Leerer Body als leeres Objekt
        final updated = request.change(context: {
            ...request.context,
            'body': <String, dynamic>{},
        });
        return innerHandler(updated);
    }

    try {
        final json = jsonDecode(bodyString);
        final updated = request.change(context: {
            ...request.context,
            'body': json,
        });
        return innerHandler(updated);
    } on FormatException {
        return Response(
            400,
            body: jsonEncode({'error': 'Invalid JSON in request body'}),
            headers: {'content-type': 'application/json'},
        );
    }
};
}

```

Verwendung der Middleware

```

void main() async {
    final router = Router();

    router.post('/users', (Request request) {
        // Body ist bereits geparkt im Context
        final body = request.context['body'] as Map<String, dynamic>;

        final name = body['name'];
        final email = body['email'];

        return Response.ok(jsonEncode({

```

```
        'message': 'User $name created',
    }));
});

final handler = Pipeline()
    .addMiddleware(logRequests())
    .addMiddleware(jsonBodyParser()) // Body Parser aktivieren
    .addHandler(router.call);

await shelf_io.serve(handler, 'localhost', 8080);
}
```

### 2.3.0.6 Extension für einfacheren Zugriff

Request Extension

```
extension RequestBody on Request {
    /// Gibt den geparseten JSON Body zurück
    Map<String, dynamic> get jsonBody {
        final body = context['body'];
        if (body is Map<String, dynamic>) {
            return body;
        }
        return {};
    }

    /// Prüft ob ein JSON Body vorhanden ist
    bool get hasJsonBody => context.containsKey('body');
}

// Verwendung
router.post('/users', (Request request) {
    final body = request.jsonBody;
    final name = body['name'];
    // ...
});
```

### 2.3.0.7 Model aus Body erstellen

Direkte Konvertierung

```
class CreateUserRequest {
    final String name;
    final String email;
    final String? phone;

    CreateUserRequest({
        required this.name,
        required this.email,
        this.phone,
    });
}
```

```

factory CreateUserRequest.fromJson(Map<String, dynamic> json) {
  return CreateUserRequest(
    name: json['name'] as String,
    email: json['email'] as String,
    phone: json['phone'] as String?,
  );
}

// Im Handler
router.post('/users', (Request request) async {
  final body = await request.readAsString();
  final json = jsonDecode(body) as Map<String, dynamic>;

  final createRequest = CreateUserRequest.fromJson(json);

  // User erstellen...
  final user = User(
    id: generateId(),
    name: createRequest.name,
    email: createRequest.email,
    phone: createRequest.phone,
  );

  return Response(201, body: jsonEncode(user.toJson()));
});

```

### 2.3.0.8 Verschiedene Content-Types

Form Data (URL-encoded)

```

Future<Response> handleFormData(Request request) async {
  final contentType = request.headers['content-type'] ?? '';

  if (contentType.contains('application/x-www-form-urlencoded')) {
    final body = await request.readAsString();
    // username=max&password=secret
    final params = Uri.splitQueryString(body);

    final username = params['username'];
    final password = params['password'];

    return Response.ok('Username: $username');
  }

  return Response(415, body: 'Unsupported Media Type');
}

```

Multipart Form Data (Datei-Upload)

```
// Für Datei-Uploads wird meist ein spezielles Package verwendet
// z.B. shelf_multipart

import 'package:shelf_multipart/shelf_multipart.dart';

Future<Response> handleFileUpload(Request request) async {
  if (!request.isMultipart) {
    return Response(400, body: 'Expected multipart request');
  }

  await for (final part in request.parts) {
    final contentDisposition = part.headers['content-disposition'];
    final filename = // ... aus contentDisposition extrahieren

    final bytes = await part.readBytes();
    // Datei speichern...
  }

  return Response.ok('File uploaded');
}
```

### 2.3.0.9 Fehlerbehandlung Best Practices

Strukturierte Fehler-Responses

```
Response badRequest(String message, {Map<String, dynamic>? details}) {
  return Response(
    400,
    body: jsonEncode({
      'error': {
        'code': 'BAD_REQUEST',
        'message': message,
        if (details != null) 'details': details,
      },
    }),
    headers: {'content-type': 'application/json'},
  );
}

// Verwendung
if (json['name'] == null) {
  return badRequest('Missing required field', details: {
    'field': 'name',
    'expected': 'string',
  });
}
```

### 2.3.0.10 Zusammenfassung

Aufgabe	Lösung
Body lesen	<code>await request.readAsString()</code>
JSON parsen	<code>jsonDecode(bodyString)</code>
Content-Type prüfen	<code>request.headers['content-type']</code>
Wiederverwendbar	Middleware mit <code>request.context</code>
Fehlerbehandlung	try/catch mit passenden HTTP-Codes

HTTP Status Codes für Body-Fehler

Code	Bedeutung	Verwendung
400	Bad Request	Ungültiges JSON, fehlende Felder
415	Unsupported Media Type	Falscher Content-Type
422	Unprocessable Entity	Valides JSON, aber ungültige Daten

### 2.3.0.11 Nächste Schritte

In der nächsten Einheit lernst du **CRUD-Operationen**: Wie du alle grundlegenden Datenbankoperationen (Create, Read, Update, Delete) über REST-Endpoints implementierst.

## 2.3.1 Übung

### 2.3.1.1 Ziel

Implementiere einen Server, der verschiedene Request Bodies verarbeitet.

### 2.3.1.2 Aufgabe 1: Einfacher JSON-Handler (15 min)

Erstelle einen Endpoint der einen neuen Task erstellt.

Anforderungen

- POST `/api/tasks`
- Erwartet JSON-Body mit `title` (required) und `description` (optional)
- Gibt den erstellten Task mit generierter `id` zurück
- Status 201 bei Erfolg

Beispiel Request

```
curl -X POST http://localhost:8080/api/tasks \
  -H "Content-Type: application/json" \
  -d '{"title": "Einkaufen", "description": "Milch und Brot"}'
```

Erwartete Response

```
{
  "id": "task-1",
  "title": "Einkaufen",
  "description": "Milch und Brot",
  "completed": false,
  "createdAt": "2024-01-15T10:30:00Z"
}
```

### 2.3.1.3 Aufgabe 2: Fehlerbehandlung (15 min)

Erweitere den Handler mit robuster Fehlerbehandlung.

Zu behandelnde Fehler

1. **Kein JSON Content-Type** -> 415 Unsupported Media Type
2. **Leerer Body** -> 400 Bad Request
3. **Ungültiges JSON** -> 400 Bad Request
4. **Fehlendes Pflichtfeld** -> 400 Bad Request

Fehler-Response Format

```
{
  "error": {
    "code": "MISSING_FIELD",
    "message": "Required field 'title' is missing"
  }
}
```

Test-Fälle

```
# Ohne Content-Type
curl -X POST http://localhost:8080/api/tasks -d '{"title": "Test"}'

# Leerer Body
curl -X POST http://localhost:8080/api/tasks -H "Content-Type: application/json"

# Ungültiges JSON
curl -X POST http://localhost:8080/api/tasks -H "Content-Type: application/json" -d '{invalid}' ↵

# Fehlendes Pflichtfeld
curl -X POST http://localhost:8080/api/tasks -H "Content-Type: application/json" -d '{}'
```

### 2.3.1.4 Aufgabe 3: Body Parser Middleware (15 min)

Erstelle eine wiederverwendbare Middleware für JSON Body Parsing.

Anforderungen

- Parst JSON Body nur für POST, PUT, PATCH
- Speichert geparstes JSON in `request.context['body']`
- Gibt 400 bei ungültigem JSON zurück
- Ignoriert Requests ohne JSON Content-Type

Verwendung

```
final handler = Pipeline()
  .addMiddleware(logRequests())
  .addMiddleware(jsonBodyParser()) // Deine Middleware
  .addHandler(router.call);

// Im Handler:
router.post('/tasks', (Request request) {
```

```
final body = request.context['body'] as Map<String, dynamic>;
final title = body['title'];
// ...
});
```

#### 2.3.1.5 Aufgabe 4: Request Extension (10 min)

Erstelle eine Extension für einfacheren Zugriff auf den Body.

Extension

```
extension RequestJsonBody on Request {
  /// Gibt den JSON Body zurück (oder leeres Map)
  Map<String, dynamic> get json => ???;

  /// Extrahiert ein Feld mit Typ-Sicherheit
  T? field<T>(String key) => ???;

  /// Extrahiert ein Pflichtfeld (wirft Exception wenn fehlt)
  String requireString(String key) => ???;
}
```

Verwendung

```
router.post('/tasks', (Request request) {
  final title = request.requireString('title');
  final description = request.field<String>('description');
  final priority = request.field<int>('priority') ?? 0;
  // ...
});
```

#### 2.3.1.6 Aufgabe 5: Komplexer Body (15 min)

Implementiere einen Handler für verschachtelte JSON-Strukturen.

Endpoint

POST /api/orders - Neue Bestellung erstellen

Request Body

```
{
  "customer": {
    "name": "Max Mustermann",
    "email": "max@example.com",
    "address": {
      "street": "Hauptstraße 1",
      "city": "Berlin",
      "zip": "10115"
    }
  },
  "items": [
    {"productId": "prod-1", "quantity": 2},
    {"productId": "prod-2", "quantity": 1}
  ]
}
```

```
],  
  "notes": "Bitte klingeln"  
}
```

Anforderungen

- Alle Customer-Felder sind Pflicht (außer address.street ist optional)
- Items muss mindestens 1 Element haben
- Quantity muss > 0 sein
- Notes ist optional

Response bei Erfolg (201)

```
{  
  "orderId": "order-123",  
  "customer": { ... },  
  "items": [ ... ],  
  "itemCount": 3,  
  "createdAt": "2024-01-15T10:30:00Z"  
}
```

### 2.3.1.7 Bonus: Form Data Handler

Implementiere einen Login-Endpoint der Form-Daten akzeptiert.

Endpoint

POST /login Content-Type: application/x-www-form-urlencoded

Body

username=max&password=secret123

Response

```
{  
  "success": true,  
  "message": "Welcome, max!"  
}
```

### 2.3.1.8 Testen

```
void main() async {  
  final app = Router();  
  
  // Deine Endpoints hier...  
  
  final handler = Pipeline()  
    .addMiddleware(logRequests())  
    .addMiddleware(jsonBodyParser())  
    .addHandler(app.call);  
  
  await shelf_io.serve(handler, 'localhost', 8080);  
}
```

```
print('Server: http://localhost:8080');  
}
```

### 2.3.1.9 Abgabe-Checkliste

- ☐ Task-Endpoint erstellt und getestet
- ☐ Alle Fehler werden korrekt behandelt
- ☐ JSON Body Parser Middleware funktioniert
- ☐ Request Extension implementiert
- ☐ Komplexer Order-Body wird validiert
- ☐ (Bonus) Form Data Login funktioniert

## 2.3.2 Lösung

### 2.3.2.1 Vollständige Lösung

```
import 'dart:convert';  
import 'package:shelf/shelf.dart';  
import 'package:shelf/shelf_io.dart' as shelf_io;  
import 'package:shelf_router/shelf_router.dart';  
  
// =====  
// Aufgabe 3: Body Parser Middleware  
// =====  
  
Middleware jsonBodyParser() {  
  return (Handler innerHandler) {  
    return (Request request) async {  
      // Nur bei relevanten Methoden  
      if (!['POST', 'PUT', 'PATCH'].contains(request.method)) {  
        return innerHandler(request);  
      }  
  
      // Content-Type prüfen  
      final contentType = request.headers['content-type'] ?? '';  
      if (!contentType.contains('application/json')) {  
        return innerHandler(request);  
      }  
  
      // Body lesen  
      final bodyString = await request.readAsString();  
  
      // Leerer Body  
      if (bodyString.isEmpty) {  
        final updated = request.change(context: {  
          ...request.context,  
          'body': <String, dynamic>{},  
        });  
        return innerHandler(updated);  
      }  
    }  
  }  
}
```

```
// JSON parsen
try {
    final json = jsonDecode(bodyString);
    final updated = request.change(context: {
        ...request.context,
        'body': json,
    });
    return innerHandler(updated);
} on FormatException catch (e) {
    return ErrorResponse(400, 'INVALID_JSON', 'Invalid JSON: ${e.message}');
}
};
};
}

// =====
// Aufgabe 4: Request Extension
// =====

extension RequestJsonBody on Request {
    /// Gibt den JSON Body zurück (oder leeres Map)
    Map<String, dynamic> get json {
        final body = context['body'];
        if (body is Map<String, dynamic>) {
            return body;
        }
        return {};
    }

    /// Extrahiert ein Feld mit Typ-Sicherheit
    T? field<T>(String key) {
        final value = json[key];
        if (value is T) {
            return value;
        }
        return null;
    }

    /// Extrahiert ein Pflichtfeld (wirft Exception wenn fehlt)
    String requireString(String key) {
        final value = json[key];
        if (value is String && value.isNotEmpty) {
            return value;
        }
        throw MissingFieldException(key);
    }

    /// Extrahiert eine Pflicht-Map
    Map<String, dynamic> requireMap(String key) {
        final value = json[key];
        if (value is Map<String, dynamic>) {
```

```

        return value;
    }
    throw MissingFieldException(key);
}

/// Extrahiert eine Pflicht-Liste
List<dynamic> requireList(String key) {
    final value = json[key];
    if (value is List && value.isNotEmpty) {
        return value;
    }
    throw MissingFieldException(key);
}
}

class MissingFieldException implements Exception {
    final String field;
    MissingFieldException(this.field);

    @override
    String toString() => 'Missing required field: $field';
}

// =====
// Helper Functions
// =====

Response jsonResponse(Object? data, {int statusCode = 200}) {
    return Response(
        statusCode,
        body: jsonEncode(data),
        headers: {'content-type': 'application/json'},
    );
}

Response ErrorResponse(int statusCode, String code, String message) {
    return Response(
        statusCode,
        body: jsonEncode({
            'error': {
                'code': code,
                'message': message,
            },
        }),
        headers: {'content-type': 'application/json'},
    );
}

// =====
// Data Storage
// =====

```

```
final _tasks = <String, Map<String, dynamic>>{};
var _nextTaskId = 1;

final _orders = <String, Map<String, dynamic>>{};
var _nextOrderId = 1;

// =====
// Aufgabe 1 & 2: Task Handler
// =====

Future<Response> createTask(Request request) async {
  // Aufgabe 2: Content-Type prüfen
  final contentType = request.headers['content-type'] ?? '';
  if (!contentType.contains('application/json')) {
    return ErrorResponse(415, 'UNSUPPORTED_MEDIA_TYPE',
      'Content-Type must be application/json');
  }

  // Body ist bereits durch Middleware geparkt
  final body = request.json;

  // Aufgabe 2: Leerer Body
  if (body.isEmpty && request.context['body'] == null) {
    return ErrorResponse(400, 'EMPTY_BODY', 'Request body is empty');
  }

  // Aufgabe 2: Pflichtfeld prüfen
  final title = body['title'] as String?;
  if (title == null || title.isEmpty) {
    return ErrorResponse(400, 'MISSING_FIELD',
      "Required field 'title' is missing");
  }

  final description = body['description'] as String?;

  // Task erstellen
  final id = 'task-${_nextTaskId++}';
  final task = {
    'id': id,
    'title': title,
    'description': description,
    'completed': false,
    'createdAt': DateTime.now().toUtc().toIso8601String(),
  };

  _tasks[id] = task;

  return jsonResponse(task, statusCode: 201);
}
```

```
// Alternative mit Extension
Future<Response> createTaskWithExtension(Request request) async {
    final contentType = request.headers['content-type'] ?? '';
    if (!contentType.contains('application/json')) {
        return ErrorResponse(415, 'UNSUPPORTED_MEDIA_TYPE',
            'Content-Type must be application/json');
    }

    try {
        final title = request.requireString('title');
        final description = request.field<String>('description');
        final priority = request.field<int>('priority') ?? 0;

        final id = 'task-${_nextTaskId++}';
        final task = {
            'id': id,
            'title': title,
            'description': description,
            'priority': priority,
            'completed': false,
            'createdAt': DateTime.now().toUtc().toIso8601String(),
        };

        _tasks[id] = task;
        return jsonResponse(task, statusCode: 201);
    } on MissingFieldException catch (e) {
        return ErrorResponse(400, 'MISSING_FIELD',
            "Required field '${e.field}' is missing");
    }
}

// =====
// Aufgabe 5: Komplexer Order Handler
// =====

Future<Response> createOrder(Request request) async {
    final contentType = request.headers['content-type'] ?? '';
    if (!contentType.contains('application/json')) {
        return ErrorResponse(415, 'UNSUPPORTED_MEDIA_TYPE',
            'Content-Type must be application/json');
    }

    try {
        // Customer validieren
        final customer = request.requireMap('customer');

        final customerName = customer['name'] as String?;
        if (customerName == null || customerName.isEmpty) {
            return ErrorResponse(400, 'MISSING_FIELD',
                "Required field 'customer.name' is missing");
        }
    }
}
```

```
}

final customerEmail = customer['email'] as String?;
if (customerEmail == null || customerEmail.isEmpty) {
    return ErrorResponse(400, 'MISSING_FIELD',
        "Required field 'customer.email' is missing");
}

final address = customer['address'] as Map<String, dynamic>?;
if (address == null) {
    return ErrorResponse(400, 'MISSING_FIELD',
        "Required field 'customer.address' is missing");
}

final city = address['city'] as String?;
if (city == null || city.isEmpty) {
    return ErrorResponse(400, 'MISSING_FIELD',
        "Required field 'customer.address.city' is missing");
}

final zip = address['zip'] as String?;
if (zip == null || zip.isEmpty) {
    return ErrorResponse(400, 'MISSING_FIELD',
        "Required field 'customer.address.zip' is missing");
}

// Items validieren
final items = request.json['items'] as List<dynamic>?;
if (items == null || items.isEmpty) {
    return ErrorResponse(400, 'MISSING_FIELD',
        "Required field 'items' must have at least 1 element");
}

var totalQuantity = 0;
final validatedItems = <Map<String, dynamic>>[];

for (var i = 0; i < items.length; i++) {
    final item = items[i] as Map<String, dynamic>?;
    if (item == null) {
        return ErrorResponse(400, 'INVALID_ITEM',
            "Item at index $i is invalid");
    }

    final productId = item['productId'] as String?;
    if (productId == null || productId.isEmpty) {
        return ErrorResponse(400, 'MISSING_FIELD',
            "Required field 'items[$i].productId' is missing");
    }

    final quantity = item['quantity'] as int?;
    if (quantity == null || quantity <= 0) {
```

```

        return ErrorResponse(400, 'INVALID_VALUE',
            "Field 'items[$i].quantity' must be greater than 0");
    }

    totalQuantity += quantity;
    validatedItems.add({
        'productId': productId,
        'quantity': quantity,
    });
}

// Optional: Notes
final notes = request.field<String>('notes');

// Order erstellen
final orderId = 'order-${_nextOrderId++}';
final order = {
    'orderId': orderId,
    'customer': {
        'name': customerName,
        'email': customerEmail,
        'address': {
            'street': address['street'] as String? ?? '',
            'city': city,
            'zip': zip,
        },
    },
    'items': validatedItems,
    'itemCount': totalQuantity,
    if (notes != null) 'notes': notes,
    'createdAt': DateTime.now().toUtc().toIso8601String(),
};

_orders[orderId] = order;

return jsonResponse(order, statusCode: 201);

} on MissingFieldException catch (e) {
    return ErrorResponse(400, 'MISSING_FIELD',
        "Required field '${e.field}' is missing");
}
}

// =====
// Bonus: Form Data Handler
// =====

Future<Response> handleLogin(Request request) async {
    final contentType = request.headers['content-type'] ?? '';

    if (!contentType.contains('application/x-www-form-urlencoded')) {

```

```
        return ErrorResponse(415, 'UNSUPPORTED_MEDIA_TYPE',
            'Content-Type must be application/x-www-form-urlencoded');
    }

    final body = await request.readAsString();

    if (body.isEmpty) {
        return ErrorResponse(400, 'EMPTY_BODY', 'Request body is empty');
    }

    final params = Uri.splitQueryString(body);

    final username = params['username'];
    final password = params['password'];

    if (username == null || username.isEmpty) {
        return ErrorResponse(400, 'MISSING_FIELD',
            "Required field 'username' is missing");
    }

    if (password == null || password.isEmpty) {
        return ErrorResponse(400, 'MISSING_FIELD',
            "Required field 'password' is missing");
    }

    // Einfache Demo-Validierung
    if (password.length < 6) {
        return ErrorResponse(401, 'INVALID_CREDENTIALS',
            'Invalid username or password');
    }

    return jsonResponse({
        'success': true,
        'message': 'Welcome, $username!',
    });
}

// =====
// Main
// =====

void main() async {
    final app = Router();

    // Task Endpoints
    app.post('/api/tasks', createTask);
    app.post('/api/tasks/v2', createTaskWithExtension);

    // Order Endpoint
    app.post('/api/orders', createOrder);
}
```

```

// Login Endpoint (Form Data)
app.post('/login', handleLogin);

// Liste aller Tasks (für Tests)
app.get('/api/tasks', (Request request) {
  return jsonResponse({
    'tasks': _tasks.values.toList(),
    'total': _tasks.length,
  });
});

// Liste aller Orders (für Tests)
app.get('/api/orders', (Request request) {
  return jsonResponse({
    'orders': _orders.values.toList(),
    'total': _orders.length,
  });
});

final handler = Pipeline()
  .addMiddleware(logRequests())
  .addMiddleware(jsonBodyParser())
  .addHandler(app.call);

await shelf_io.serve(handler, 'localhost', 8080);
print('Server: http://localhost:8080');
print('');
print('Test-Befehle:');
print('');
print('# Task erstellen');
print('curl -X POST http://localhost:8080/api/tasks \\');
print('  -H "Content-Type: application/json" \\');
print('  -d \\{"title": "Einkaufen", "description": "Milch"}\\');
print('');
print('# Order erstellen');
print('curl -X POST http://localhost:8080/api/orders \\');
print('  -H "Content-Type: application/json" \\');
print('  -d                                     ↪
↪ \\{"customer":{"name":"Max","email":"max@test.de","address":{"city":"Berlin","zip":"10115"}
print('');
print('# Login (Form Data)');
print('curl -X POST http://localhost:8080/login \\');
print('  -H "Content-Type: application/x-www-form-urlencoded" \\');
print('  -d "username=max&password=secret123"');
}

```

### 2.3.2.2 Test-Befehle

```

# === Aufgabe 1: Task erstellen ===
curl -X POST http://localhost:8080/api/tasks \

```

```
-H "Content-Type: application/json" \  
-d '{"title": "Einkaufen", "description": "Milch und Brot"}'  
  
# === Aufgabe 2: Fehler testen ===  
  
# Ohne Content-Type -> 415  
curl -X POST http://localhost:8080/api/tasks \  
-d '{"title": "Test"}'  
  
# Leerer Body -> 400  
curl -X POST http://localhost:8080/api/tasks \  
-H "Content-Type: application/json" \  
-d ''  
  
# Ungültiges JSON -> 400  
curl -X POST http://localhost:8080/api/tasks \  
-H "Content-Type: application/json" \  
-d '{invalid}'  
  
# Fehlendes Pflichtfeld -> 400  
curl -X POST http://localhost:8080/api/tasks \  
-H "Content-Type: application/json" \  
-d '{"description": "ohne title"}'  
  
# === Aufgabe 5: Order erstellen ===  
  
# Erfolgreiche Order  
curl -X POST http://localhost:8080/api/orders \  
-H "Content-Type: application/json" \  
-d '{  
  "customer": {  
    "name": "Max Mustermann",  
    "email": "max@example.com",  
    "address": {  
      "street": "Hauptstraße 1",  
      "city": "Berlin",  
      "zip": "10115"  
    }  
  },  
  "items": [  
    {"productId": "prod-1", "quantity": 2},  
    {"productId": "prod-2", "quantity": 1}  
  ],  
  "notes": "Bitte klingeln"  
}'  
  
# Fehlende Items -> 400  
curl -X POST http://localhost:8080/api/orders \  
-H "Content-Type: application/json" \  
-d '{  
  "customer": {"name": "Max", "email": "max@test.de", "address": {"city":  
↪ "Berlin", "zip": "10115"}},  
↪
```

```

    "items": []
  }'

# Ungültige Quantity -> 400
curl -X POST http://localhost:8080/api/orders \
  -H "Content-Type: application/json" \
  -d '{
    "customer": {"name": "Max", "email": "max@test.de", "address": {"city":
↪ "Berlin", "zip": "10115"}},
↪ "items": [{"productId": "p1", "quantity": 0}]
  }'

# === Bonus: Login ===
curl -X POST http://localhost:8080/login \
  -H "Content-Type: application/x-www-form-urlencoded" \
  -d "username=max&password=secret123"

# === Listen abrufen ===
curl http://localhost:8080/api/tasks
curl http://localhost:8080/api/orders

```

### 2.3.2.3 Ausgabe-Beispiele

Task erstellen (201)

```

{
  "id": "task-1",
  "title": "Einkaufen",
  "description": "Milch und Brot",
  "completed": false,
  "createdAt": "2024-01-15T10:30:00.000Z"
}

```

Fehler: Missing Field (400)

```

{
  "error": {
    "code": "MISSING_FIELD",
    "message": "Required field 'title' is missing"
  }
}

```

Order erstellen (201)

```

{
  "orderId": "order-1",
  "customer": {
    "name": "Max Mustermann",
    "email": "max@example.com",
    "address": {
      "street": "Hauptstraße 1",
      "city": "Berlin",

```

```

        "zip": "10115"
    }
},
"items": [
    {"productId": "prod-1", "quantity": 2},
    {"productId": "prod-2", "quantity": 1}
],
"itemCount": 3,
"notes": "Bitte klingeln",
"createdAt": "2024-01-15T10:30:00.000Z"
}

```

Login Erfolg (200)

```

{
  "success": true,
  "message": "Welcome, max!"
}

```

### 2.3.2.4 Wichtige Patterns

Middleware speichert geparseten Body

```

final updated = request.change(context: {
    ...request.context,
    'body': json, // Hier speichern
});
return innerHandler(updated);

// Später im Handler:
final body = request.context['body'] as Map<String, dynamic>;

```

Extension für sauberen Zugriff

```

extension RequestJsonBody on Request {
    Map<String, dynamic> get json {
        final body = context['body'];
        return body is Map<String, dynamic> ? body : {};
    }
}

// Verwendung
final title = request.json['title'];

```

Strukturierte Fehler-Responses

```

Response ErrorResponse(int statusCode, String code, String message) {
    return Response(
        statusCode,
        body: jsonEncode({
            'error': {
                'code': code,
                'message': message,
            }
        })
    );
}

```

```
    },
  )),
  headers: {'content-type': 'application/json'},
);
}
```

### 2.3.3 Ressourcen

#### 2.3.3.1 Offizielle Dokumentation

- Shelf Request API
- dart:convert
- HTTP Content-Type

#### 2.3.3.2 Cheat Sheet: Body lesen

```
// Body als String
final body = await request.readAsString();

// Body als Bytes
final bytes = await request.read().toList();
final allBytes = bytes.expand((b) => b).toList();

// Achtung: Body kann nur einmal gelesen werden!
```

#### 2.3.3.3 Cheat Sheet: JSON Body

```
import 'dart:convert';

Future<Response> handler(Request request) async {
  // Body lesen und parsen
  final body = await request.readAsString();
  final json = jsonDecode(body) as Map<String, dynamic>;

  // Felder extrahieren
  final name = json['name'] as String;
  final age = json['age'] as int;
  final email = json['email'] as String?; // Optional

  return Response.ok('OK');
}
```

#### 2.3.3.4 Cheat Sheet: Content-Type prüfen

```
final contentType = request.headers['content-type'] ?? '';

if (contentType.contains('application/json')) {
  // JSON verarbeiten
} else if (contentType.contains('application/x-www-form-urlencoded')) {
  // Form-Daten verarbeiten
}
```

```
} else if (contentType.contains('multipart/form-data')) {  
    // Datei-Upload verarbeiten  
} else {  
    return Response(415); // Unsupported Media Type  
}
```

### 2.3.3.5 Cheat Sheet: Sichere Extraktion

```
// Mit Default-Werten  
final name = json['name'] as String? ?? 'Unknown';  
final age = json['age'] as int? ?? 0;  
  
// Mit Null-Check  
final name = json['name'] as String?;  
if (name == null) {  
    return badRequest('name is required');  
}  
  
// Typ-sichere Konvertierung  
int parseId(dynamic value) {  
    if (value is int) return value;  
    if (value is String) return int.tryParse(value) ?? 0;  
    return 0;  
}
```

### 2.3.3.6 Cheat Sheet: Body Parser Middleware

```
Middleware jsonBodyParser() {  
    return (Handler handler) {  
        return (Request request) async {  
            if (!['POST', 'PUT', 'PATCH'].contains(request.method)) {  
                return handler(request);  
            }  
  
            final contentType = request.headers['content-type'] ?? '';  
            if (!contentType.contains('application/json')) {  
                return handler(request);  
            }  
  
            final body = await request.readAsString();  
            if (body.isEmpty) {  
                return handler(request.change(context: {  
                    ...request.context,  
                    'body': <String, dynamic>{},  
                }));  
            }  
  
            try {  
                final json = jsonDecode(body);  
            }  
        }  
    }  
}
```

```
        return handler(request.change(context: {
            ...request.context,
            'body': json,
        }));
    } on FormatException {
        return Response(400,
            body: '{"error": "Invalid JSON"}',
            headers: {'content-type': 'application/json'},
        );
    }
};
};
}

// Verwendung
final handler = Pipeline()
    .addMiddleware(jsonBodyParser())
    .addHandler(router.call);
```

### 2.3.3.7 Cheat Sheet: Form Data

```
// application/x-www-form-urlencoded
// Body: username=max&password=secret

final body = await request.readAsString();
final params = Uri.splitQueryString(body);

final username = params['username']; // "max"
final password = params['password']; // "secret"
```

### 2.3.3.8 Cheat Sheet: Fehler-Responses

```
// 400 Bad Request - Allgemeiner Client-Fehler
Response badRequest(String message) => Response(400,
    body: jsonEncode({'error': message}),
    headers: {'content-type': 'application/json'},
);

// 415 Unsupported Media Type - Falscher Content-Type
Response unsupportedMediaType() => Response(415,
    body: jsonEncode({'error': 'Content-Type must be application/json'}),
    headers: {'content-type': 'application/json'},
);

// 422 Unprocessable Entity - Semantische Fehler
Response unprocessableEntity(String message) => Response(422,
    body: jsonEncode({'error': message}),
    headers: {'content-type': 'application/json'},
);
```

### 2.3.3.9 Cheat Sheet: Request Extension

```
extension RequestBody on Request {
  Map<String, dynamic> get jsonBody {
    final body = context['body'];
    if (body is Map<String, dynamic>) return body;
    return {};
  }

  T? getField<T>(String key) {
    return jsonBody[key] as T?;
  }

  String requireField(String key) {
    final value = jsonBody[key] as String?;
    if (value == null) {
      throw ArgumentError('Missing required field: $key');
    }
    return value;
  }
}
```

### 2.3.3.10 HTTP Content-Types

Content-Type	Verwendung
application/json	REST APIs, strukturierte Daten
application/x-www-form-urlencoded	HTML-Formulare
multipart/form-data	Datei-Uploads
text/plain	Einfacher Text
application/xml	XML-Daten

### 2.3.3.11 HTTP Status Codes für Body-Fehler

Code	Name	Verwendung
400	Bad Request	Ungültiges JSON, Syntax-Fehler
415	Unsupported Media Type	Content-Type nicht unterstützt
422	Unprocessable Entity	Valide Syntax, aber ungültige Semantik

### 2.3.3.12 Test-Befehle

```
# JSON Body senden
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Max", "email": "max@example.com"}'

# Form Data senden
curl -X POST http://localhost:8080/login \
  -H "Content-Type: application/x-www-form-urlencoded" \
```

```

-d "username=max&password=secret"

# Datei hochladen
curl -X POST http://localhost:8080/upload \
  -F "file=@/path/to/file.pdf"

# Ohne Content-Type (sollte 415 geben)
curl -X POST http://localhost:8080/api/users \
  -d '{"name": "Max"}'

# Ungültiges JSON (sollte 400 geben)
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{invalid json}'

```

### 2.3.3.13 Debugging-Tipps

```

// Body und Headers loggen
Future<Response> debugHandler(Request request) async {
  print('Method: ${request.method}');
  print('Path: ${request.url.path}');
  print('Headers: ${request.headers}');

  final body = await request.readAsString();
  print('Body: $body');

  return Response.ok('Debug logged');
}

```

## 2.4 Einheit 6.4: CRUD Operationen

### 2.4.0.1 Lernziele

Nach dieser Einheit kannst du: - Alle CRUD-Operationen (Create, Read, Update, Delete) implementieren - Die passenden HTTP-Methoden und Statuscodes verwenden - Eine vollständige REST-API für eine Ressource erstellen - PUT vs. PATCH korrekt einsetzen

### 2.4.0.2 CRUD Übersicht

CRUD beschreibt die vier grundlegenden Operationen für persistente Daten:

Operation	HTTP-Methode	Pfad	Beschreibung
Create	POST	/items	Neue Ressource erstellen
Read	GET	/items/:id	Eine Ressource lesen
Read All	GET	/items	Alle Ressourcen listen
Update	PUT/PATCH	/items/:id	Ressource aktualisieren
Delete	DELETE	/items/:id	Ressource löschen

### 2.4.0.3 Projekt-Setup

Model-Klasse

```
class Todo {
  final String id;
  final String title;
  final String? description;
  final bool completed;
  final DateTime createdAt;
  final DateTime? updatedAt;

  Todo({
    required this.id,
    required this.title,
    this.description,
    this.completed = false,
    required this.createdAt,
    this.updatedAt,
  });

  factory Todo.fromJson(Map<String, dynamic> json) {
    return Todo(
      id: json['id'] as String,
      title: json['title'] as String,
      description: json['description'] as String?,
      completed: json['completed'] as bool? ?? false,
      createdAt: DateTime.parse(json['created_at'] as String),
      updatedAt: json['updated_at'] != null
        ? DateTime.parse(json['updated_at'] as String)
        : null,
    );
  }

  Map<String, dynamic> toJson() => {
    'id': id,
    'title': title,
    if (description != null) 'description': description,
    'completed': completed,
    'created_at': createdAt.toIso8601String(),
    if (updatedAt != null) 'updated_at': updatedAt!.toIso8601String(),
  };

  Todo copyWith({
    String? title,
    String? description,
    bool? completed,
    DateTime? updatedAt,
  }) => Todo(
    id: id,
    title: title ?? this.title,
    description: description ?? this.description,
```

```

    completed: completed ?? this.completed,
    createdAt: createdAt,
    updatedAt: updatedAt ?? this.updatedAt,
  );
}

```

#### In-Memory Storage

```

class TodoRepository {
  final _todos = <String, Todo>{};
  var _nextId = 1;

  String _generateId() => 'todo-${_nextId++}';

  List<Todo> findAll() => _todos.values.toList();

  Todo? findById(String id) => _todos[id];

  Todo create(String title, String? description) {
    final id = _generateId();
    final todo = Todo(
      id: id,
      title: title,
      description: description,
      createdAt: DateTime.now().toUtc(),
    );
    _todos[id] = todo;
    return todo;
  }

  Todo? update(String id, {String? title, String? description, bool?
↩ completed}) {
    ↪
    final existing = _todos[id];
    if (existing == null) return null;

    final updated = existing.copyWith(
      title: title,
      description: description,
      completed: completed,
      updatedAt: DateTime.now().toUtc(),
    );
    _todos[id] = updated;
    return updated;
  }

  bool delete(String id) => _todos.remove(id) != null;
}

```

#### 2.4.0.4 CREATE (POST)

Neue Ressource erstellen

```
final repo = TodoRepository();

Future<Response> createTodo(Request request) async {
  final body = request.json;

  // Validierung
  final title = body['title'] as String?;
  if (title == null || title.isEmpty) {
    return Response(400,
      body: jsonEncode({'error': 'title is required'}),
      headers: {'content-type': 'application/json'},
    );
  }

  final description = body['description'] as String?;

  // Erstellen
  final todo = repo.create(title, description);

  // 201 Created mit Location-Header
  return Response(201,
    body: jsonEncode(todo.toJson()),
    headers: {
      'content-type': 'application/json',
      'location': '/api/todos/${todo.id}',
    },
  );
}
```

#### Wichtige Punkte

- **Status 201 Created** bei Erfolg
- **Location-Header** mit URL der neuen Ressource
- Validierung der Pflichtfelder
- Generierte ID zurückgeben

#### 2.4.0.5 READ (GET)

Einzelne Ressource lesen

```
Response getTodo(Request request, String id) {
  final todo = repo.findById(id);

  if (todo == null) {
    return Response(404,
      body: jsonEncode({'error': 'Todo not found'}),
      headers: {'content-type': 'application/json'},
    );
  }

  return Response.ok(
    jsonEncode(todo.toJson()),
  );
}
```

```
    headers: {'content-type': 'application/json'},
  );
}
```

Alle Ressourcen listen

```
Response listTodos(Request request) {
  final todos = repo.findAll();

  return Response.ok(
    jsonEncode({
      'data': todos.map((t) => t.toJson()).toList(),
      'total': todos.length,
    }),
    headers: {'content-type': 'application/json'},
  );
}
```

Mit Query-Parametern filtern

```
Response listTodos(Request request) {
  var todos = repo.findAll();

  // Filter: ?completed=true
  final completedParam = request.url.queryParameters['completed'];
  if (completedParam != null) {
    final completed = completedParam == 'true';
    todos = todos.where((t) => t.completed == completed).toList();
  }

  // Sortierung: ?sort=created_at&order=desc
  final sort = request.url.queryParameters['sort'];
  final order = request.url.queryParameters['order'] ?? 'asc';

  if (sort == 'created_at') {
    todos.sort((a, b) => order == 'desc'
      ? b.createdAt.compareTo(a.createdAt)
      : a.createdAt.compareTo(b.createdAt));
  } else if (sort == 'title') {
    todos.sort((a, b) => order == 'desc'
      ? b.title.compareTo(a.title)
      : a.title.compareTo(b.title));
  }

  return Response.ok(
    jsonEncode({
      'data': todos.map((t) => t.toJson()).toList(),
      'total': todos.length,
    }),
    headers: {'content-type': 'application/json'},
  );
}
```

### 2.4.0.6 UPDATE (PUT vs. PATCH)

PUT - Vollständiges Ersetzen

PUT ersetzt die **gesamte** Ressource. Alle Felder müssen angegeben werden.

```
Future<Response> replaceTodo(Request request, String id) async {
    final existing = repo.findById(id);
    if (existing == null) {
        return Response(404,
            body: jsonEncode({'error': 'Todo not found'}),
            headers: {'content-type': 'application/json'},
        );
    }

    final body = request.json;

    // Bei PUT müssen alle Felder vorhanden sein
    final title = body['title'] as String?;
    if (title == null || title.isEmpty) {
        return Response(400,
            body: jsonEncode({'error': 'title is required'}),
            headers: {'content-type': 'application/json'},
        );
    }

    // Alle Felder werden überschrieben (auch mit null/default)
    final updated = repo.update(
        id,
        title: title,
        description: body['description'] as String?,
        completed: body['completed'] as bool? ?? false,
    );

    return Response.ok(
        jsonEncode(updated!.toJson()),
        headers: {'content-type': 'application/json'},
    );
}
```

PATCH - Teilweise Aktualisierung

PATCH aktualisiert nur die **angegebenen** Felder.

```
Future<Response> updateTodo(Request request, String id) async {
    final existing = repo.findById(id);
    if (existing == null) {
        return Response(404,
            body: jsonEncode({'error': 'Todo not found'}),
            headers: {'content-type': 'application/json'},
        );
    }
}
```

```
final body = request.json;

// Nur übergebene Felder aktualisieren
final updated = repo.update(
    id,
    title: body['title'] as String?,
    description: body['description'] as String?,
    completed: body['completed'] as bool?,
);

return Response.ok(
    jsonEncode(updated!.toJson()),
    headers: {'content-type': 'application/json'},
);
}
```

Unterschied PUT vs. PATCH

```
# Ursprünglicher Todo:
{
  "id": "1",
  "title": "Einkaufen",
  "description": "Milch und Brot",
  "completed": false
}

# PUT /todos/1 mit {"title": "Kochen", "completed": true}
# -> description wird auf null gesetzt!
{
  "id": "1",
  "title": "Kochen",
  "description": null,
  "completed": true
}

# PATCH /todos/1 mit {"completed": true}
# -> Nur completed wird geändert, Rest bleibt!
{
  "id": "1",
  "title": "Einkaufen",
  "description": "Milch und Brot",
  "completed": true
}
```

#### 2.4.0.7 DELETE

Ressource löschen

```
Response deleteTodo(Request request, String id) {
    final deleted = repo.delete(id);
```

```
if (!deleted) {
  return Response(404,
    body: jsonEncode({'error': 'Todo not found'}),
    headers: {'content-type': 'application/json'},
  );
}

// 204 No Content - Erfolg ohne Body
return Response(204);
}
```

Alternative: Soft Delete

```
Response softDeleteTodo(Request request, String id) {
  final existing = repo.findById(id);
  if (existing == null) {
    return Response(404,
      body: jsonEncode({'error': 'Todo not found'}),
      headers: {'content-type': 'application/json'},
    );
  }

  // Statt löschen: als gelöscht markieren
  repo.update(id, deletedAt: DateTime.now().toUtc());

  return Response(204);
}
```

#### 2.4.0.8 Vollständiger Router

```
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';

void main() async {
  final repo = TodoRepository();
  final api = TodoApi(repo);

  final router = Router();

  // CRUD Routen
  router.get('/api/todos', api.listTodos);
  router.get('/api/todos/<id>', api.getTodo);
  router.post('/api/todos', api.createTodo);
  router.put('/api/todos/<id>', api.replaceTodo);
  router.patch('/api/todos/<id>', api.updateTodo);
  router.delete('/api/todos/<id>', api.deleteTodo);

  final handler = Pipeline()
    .addMiddleware(logRequests())
```

```

        .addMiddleware(jsonBodyParser())
        .addHandler(router.call);

    await shelf_io.serve(handler, 'localhost', 8080);
    print('Server: http://localhost:8080');
}

class TodoApi {
    final TodoRepository repo;

    TodoApi(this.repo);

    Response listTodos(Request request) { /* ... */ }
    Response getTodo(Request request, String id) { /* ... */ }
    Future<Response> createTodo(Request request) async { /* ... */ }
    Future<Response> replaceTodo(Request request, String id) async { /* ... */ }
    Future<Response> updateTodo(Request request, String id) async { /* ... */ }
    Response deleteTodo(Request request, String id) { /* ... */ }
}

```

#### 2.4.0.9 HTTP Statuscodes Übersicht

Operation	Erfolg	Nicht gefunden	Validierungsfehler
GET (one)	200 OK	404 Not Found	-
GET (all)	200 OK	-	-
POST	201 Created	-	400 Bad Request
PUT	200 OK	404 Not Found	400 Bad Request
PATCH	200 OK	404 Not Found	400 Bad Request
DELETE	204 No Content	404 Not Found	-

#### 2.4.0.10 Idempotenz

**Idempotent** bedeutet: Mehrfaches Ausführen hat das gleiche Ergebnis.

Methode	Idempotent?	Erklärung
GET	Ja	Liest nur, ändert nichts
PUT	Ja	Gleiches Ergebnis bei wiederholtem Aufruf
DELETE	Ja	Nach dem Löschen: 404 (aber Ressource bleibt gelöscht)
POST	<b>Nein</b>	Jeder Aufruf erstellt neue Ressource
PATCH	Meistens	Hängt von der Implementation ab

#### 2.4.0.11 Zusammenfassung

Operation	Methode	Pfad	Status
Create	POST	/api/todos	201 + Location
Read One	GET	/api/todos/:id	200 / 404
Read All	GET	/api/todos	200

Operation	Methode	Pfad	Status
Replace	PUT	/api/todos/:id	200 / 404
Update	PATCH	/api/todos/:id	200 / 404
Delete	DELETE	/api/todos/:id	204 / 404

### 2.4.0.12 Nächste Schritte

In der nächsten Einheit lernst du **Input-Validierung**: Wie du Eingabedaten systematisch prüfst und aussagekräftige Fehlermeldungen zurückgibst.

## 2.4.1 Übung

### 2.4.1.1 Ziel

Implementiere eine vollständige REST-API für eine Notizen-Anwendung.

### 2.4.1.2 Aufgabe 1: Model & Repository (15 min)

Erstelle die Datenstrukturen für Notizen.

Note Model

```
class Note {
    final String id;
    final String title;
    final String content;
    final String? category;
    final List<String> tags;
    final bool pinned;
    final DateTime createdAt;
    final DateTime? updatedAt;

    // Constructor, fromJson, toJson, copyWith implementieren
}
```

Repository Interface

```
class NoteRepository {
    // In-Memory Storage
    final _notes = <String, Note>{};

    List<Note> findAll();
    Note? findById(String id);
    Note create({required String title, required String content, String?
↪ category, List<String> tags});
    Note? update(String id, {String? title, String? content, String? category,
↪ List<String>? tags, bool? pinned});
    bool delete(String id);
}
```

### 2.4.1.3 Aufgabe 2: CREATE Endpoint (10 min)

Implementiere den POST-Endpoint für neue Notizen.

## Anforderungen

- POST /api/notes
- title und content sind Pflichtfelder
- category und tags sind optional
- pinned default: false
- Status 201 mit Location-Header bei Erfolg
- Status 400 bei fehlenden Pflichtfeldern

## Request Body

```
{
  "title": "Meeting Notes",
  "content": "Agenda: ...",
  "category": "work",
  "tags": ["meeting", "important"]
}
```

## Response (201)

```
{
  "id": "note-1",
  "title": "Meeting Notes",
  "content": "Agenda: ...",
  "category": "work",
  "tags": ["meeting", "important"],
  "pinned": false,
  "createdAt": "2024-01-15T10:00:00Z"
}
```

**2.4.1.4 Aufgabe 3: READ Endpoints (15 min)**

Implementiere die GET-Endpoints.

## Liste aller Notizen

- GET /api/notes
- Optional: Filter per Query-Parameter
  - ?category=work - Nach Kategorie filtern
  - ?pinned=true - Nur gepinnte Notizen
  - ?tag=meeting - Nach Tag filtern

## Response

```
{
  "data": [
    {"id": "note-1", "title": "Meeting Notes", ...},
    {"id": "note-2", "title": "Ideas", ...}
  ],
  "total": 2
}
```

## Einzelne Notiz

- GET /api/notes/:id
- Status 404 wenn nicht gefunden

### 2.4.1.5 Aufgabe 4: UPDATE Endpoints (15 min)

Implementiere PUT und PATCH.

PUT /api/notes/:id (Replace)

- Ersetzt die komplette Notiz
- **title** und **content** sind Pflicht
- Andere Felder werden auf Default zurückgesetzt wenn nicht angegeben

PATCH /api/notes/:id (Partial Update)

- Aktualisiert nur die angegebenen Felder
- Perfekt für "Pin/Unpin":

```
curl -X PATCH http://localhost:8080/api/notes/1 \
-H "Content-Type: application/json" \
-d '{"pinned": true}'
```

Unterschied demonstrieren

```
# Original: {"title": "Test", "content": "Hello", "category": "work",
↪  "pinned": false}

# PUT ohne category:
# -> {"title": "New", "content": "World", "category": null, "pinned": false}

# PATCH mit nur pinned:
# -> {"title": "Test", "content": "Hello", "category": "work", "pinned": true}
```

### 2.4.1.6 Aufgabe 5: DELETE Endpoint (5 min)

Implementiere den DELETE-Endpoint.

DELETE /api/notes/:id

- Status 204 (No Content) bei Erfolg
- Status 404 wenn nicht gefunden

### 2.4.1.7 Aufgabe 6: Sub-Ressourcen (Bonus, 15 min)

Erweitere die API um Kommentare zu Notizen.

Comment Model

```
class Comment {
    final String id;
    final String noteId;
    final String author;
    final String text;
    final DateTime createdAt;
}
```

Endpoints

Methode	Pfad	Beschreibung
GET	/api/notes/:noteId/comments	Alle Kommentare einer Notiz
POST	/api/notes/:noteId/comments	Kommentar hinzufügen
DELETE	/api/notes/:noteId/comments/:commentId	Kommentar löschen

#### 2.4.1.8 Vollständige API-Übersicht

Methode	Pfad	Beschreibung	Status
GET	/api/notes	Alle Notizen	200
GET	/api/notes/:id	Eine Notiz	200/404
POST	/api/notes	Notiz erstellen	201/400
PUT	/api/notes/:id	Notiz ersetzen	200/400/404
PATCH	/api/notes/:id	Notiz aktualisieren	200/404
DELETE	/api/notes/:id	Notiz löschen	204/404

#### 2.4.1.9 Testen

```
# CREATE
curl -X POST http://localhost:8080/api/notes \
  -H "Content-Type: application/json" \
  -d '{"title": "Test", "content": "Hello World"}'

# READ ALL
curl http://localhost:8080/api/notes

# READ ONE
curl http://localhost:8080/api/notes/note-1

# UPDATE (PATCH)
curl -X PATCH http://localhost:8080/api/notes/note-1 \
  -H "Content-Type: application/json" \
  -d '{"pinned": true}'

# UPDATE (PUT)
curl -X PUT http://localhost:8080/api/notes/note-1 \
  -H "Content-Type: application/json" \
  -d '{"title": "Updated", "content": "New content"}'

# DELETE
curl -X DELETE http://localhost:8080/api/notes/note-1

# FILTER
curl "http://localhost:8080/api/notes?pinned=true"
curl "http://localhost:8080/api/notes?category=work"
```

#### 2.4.1.10 Abgabe-Checkliste

- ☐ Note Model mit allen Feldern
- ☐ NoteRepository mit allen Methoden

- ☐ POST /api/notes funktioniert (201 + Location)
- ☐ GET /api/notes listet alle Notizen
- ☐ GET /api/notes/:id gibt einzelne Notiz zurück
- ☐ PUT /api/notes/:id ersetzt komplett
- ☐ PATCH /api/notes/:id aktualisiert teilweise
- ☐ DELETE /api/notes/:id löscht (204)
- ☐ 404 bei nicht existierenden Notizen
- ☐ 400 bei fehlenden Pflichtfeldern
- ☐ (Bonus) Filtering per Query-Parameter
- ☐ (Bonus) Comment Sub-Ressourcen

## 2.4.2 Lösung

### 2.4.2.1 Vollständige Lösung

```
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';

// =====
// Aufgabe 1: Models
// =====

class Note {
  final String id;
  final String title;
  final String content;
  final String? category;
  final List<String> tags;
  final bool pinned;
  final DateTime createdAt;
  final DateTime? updatedAt;

  Note({
    required this.id,
    required this.title,
    required this.content,
    this.category,
    this.tags = const [],
    this.pinned = false,
    required this.createdAt,
    this.updatedAt,
  });

  factory Note.fromJson(Map<String, dynamic> json) {
    return Note(
      id: json['id'] as String,
      title: json['title'] as String,
      content: json['content'] as String,
      category: json['category'] as String?,
    );
  }
}
```

```

        tags: (json['tags'] as List<dynamic>?)
            ?.map((e) => e as String)
            .toList() ?? [],
        pinned: json['pinned'] as bool? ?? false,
        createdAt: DateTime.parse(json['createdAt'] as String),
        updatedAt: json['updatedAt'] != null
            ? DateTime.parse(json['updatedAt'] as String)
            : null,
    );
}

Map<String, dynamic> toJson() => {
    'id': id,
    'title': title,
    'content': content,
    if (category != null) 'category': category,
    'tags': tags,
    'pinned': pinned,
    'createdAt': createdAt.toIso8601String(),
    if (updatedAt != null) 'updatedAt': updatedAt!.toIso8601String(),
};

Note copyWith({
    String? title,
    String? content,
    String? category,
    List<String>? tags,
    bool? pinned,
    DateTime? updatedAt,
}) => Note(
    id: id,
    title: title ?? this.title,
    content: content ?? this.content,
    category: category ?? this.category,
    tags: tags ?? this.tags,
    pinned: pinned ?? this.pinned,
    createdAt: createdAt,
    updatedAt: updatedAt ?? this.updatedAt,
);
}

class Comment {
    final String id;
    final String noteId;
    final String author;
    final String text;
    final DateTime createdAt;

    Comment({
        required this.id,
        required this.noteId,

```

```
        required this.author,
        required this.text,
        required this.createdAt,
    });

    Map<String, dynamic> toJson() => {
        'id': id,
        'noteId': noteId,
        'author': author,
        'text': text,
        'createdAt': createdAt.toIso8601String(),
    };
}

// =====
// Aufgabe 1: Repository
// =====

class NoteRepository {
    final _notes = <String, Note>{};
    var _nextId = 1;

    String _generateId() => 'note-${_nextId++}';

    List<Note> findAll() => _notes.values.toList();

    Note? findById(String id) => _notes[id];

    Note create({
        required String title,
        required String content,
        String? category,
        List<String> tags = const [],
        bool pinned = false,
    }) {
        final id = _generateId();
        final note = Note(
            id: id,
            title: title,
            content: content,
            category: category,
            tags: tags,
            pinned: pinned,
            createdAt: DateTime.now().toUtc(),
        );
        _notes[id] = note;
        return note;
    }

    Note? update(
        String id, {
```

```
String? title,
String? content,
String? category,
List<String>? tags,
bool? pinned,
}) {
    final existing = _notes[id];
    if (existing == null) return null;

    final updated = existing.copyWith(
        title: title,
        content: content,
        category: category,
        tags: tags,
        pinned: pinned,
        updatedAt: DateTime.now().toUtc(),
    );
    _notes[id] = updated;
    return updated;
}

Note? replace(
    String id, {
    required String title,
    required String content,
    String? category,
    List<String> tags = const [],
    bool pinned = false,
}) {
    final existing = _notes[id];
    if (existing == null) return null;

    final replaced = Note(
        id: id,
        title: title,
        content: content,
        category: category,
        tags: tags,
        pinned: pinned,
        createdAt: existing.createdAt,
        updatedAt: DateTime.now().toUtc(),
    );
    _notes[id] = replaced;
    return replaced;
}

bool delete(String id) => _notes.remove(id) != null;
}

class CommentRepository {
    final _comments = <String, Comment>{};
```

```
var _nextId = 1;

String _generateId() => 'comment-${_nextId++}';

List<Comment> findByNoteId(String noteId) =>
    _comments.values.where((c) => c.noteId == noteId).toList();

Comment? findById(String id) => _comments[id];

Comment create({
    required String noteId,
    required String author,
    required String text,
}) {
    final id = _generateId();
    final comment = Comment(
        id: id,
        noteId: noteId,
        author: author,
        text: text,
        createdAt: DateTime.now().toUtc(),
    );
    _comments[id] = comment;
    return comment;
}

bool delete(String id) => _comments.remove(id) != null;
}

// =====
// Helper Functions
// =====

Response jsonResponse(Object? data, {int statusCode = 200}) {
    return Response(
        statusCode,
        body: jsonEncode(data),
        headers: {'content-type': 'application/json'},
    );
}

Response notFound(String message) =>
    jsonResponse({'error': message}, statusCode: 404);

Response badRequest(String message) =>
    jsonResponse({'error': message}, statusCode: 400);

Response created(Object data, String location) {
    return Response(201,
        body: jsonEncode(data),
        headers: {
```

```

        'content-type': 'application/json',
        'location': location,
    },
);
}

extension JsonRequest on Request {
    Map<String, dynamic> get json {
        final body = context['body'];
        return body is Map<String, dynamic> ? body : {};
    }
}

Middleware jsonBodyParser() {
    return (Handler handler) {
        return (Request request) async {
            if (!['POST', 'PUT', 'PATCH'].contains(request.method)) {
                return handler(request);
            }
            final contentType = request.headers['content-type'] ?? '';
            if (!contentType.contains('application/json')) {
                return handler(request);
            }
            final body = await request.readAsString();
            if (body.isEmpty) {
                return handler(request.change(context: {...request.context, 'body':
↵ <String, dynamic>{}}));
            }
            try {
                final json = jsonDecode(body);
                return handler(request.change(context: {...request.context, 'body':
↵ json}));
            } on FormatException {
                return badRequest('Invalid JSON');
            }
        };
    };
}

// =====
// API Handlers
// =====

class NotesApi {
    final NoteRepository noteRepo;
    final CommentRepository commentRepo;

    NotesApi(this.noteRepo, this.commentRepo);

    // ===== Aufgabe 3: READ =====

```

```
Response listNotes(Request request) {
    var notes = noteRepo.findAll();

    // Filter: ?category=work
    final category = request.url.queryParameters['category'];
    if (category != null) {
        notes = notes.where((n) => n.category == category).toList();
    }

    // Filter: ?pinned=true
    final pinnedParam = request.url.queryParameters['pinned'];
    if (pinnedParam != null) {
        final pinned = pinnedParam == 'true';
        notes = notes.where((n) => n.pinned == pinned).toList();
    }

    // Filter: ?tag=meeting
    final tag = request.url.queryParameters['tag'];
    if (tag != null) {
        notes = notes.where((n) => n.tags.contains(tag)).toList();
    }

    return jsonResponse({
        'data': notes.map((n) => n.toJson()).toList(),
        'total': notes.length,
    });
}

Response getNote(Request request, String id) {
    final note = noteRepo.findById(id);
    if (note == null) {
        return notFound('Note not found');
    }
    return jsonResponse(note.toJson());
}

// ===== Aufgabe 2: CREATE =====

Response createNote(Request request) {
    final body = request.json;

    // Validierung
    final title = body['title'] as String?;
    final content = body['content'] as String?;

    if (title == null || title.isEmpty) {
        return badRequest('title is required');
    }
    if (content == null || content.isEmpty) {
        return badRequest('content is required');
    }
}
```

```
final category = body['category'] as String?;
final tags = (body['tags'] as List<dynamic>?)
    ?.map((e) => e as String)
    .toList() ?? [];
final pinned = body['pinned'] as bool? ?? false;

final note = noteRepo.create(
    title: title,
    content: content,
    category: category,
    tags: tags,
    pinned: pinned,
);

return created(note.toJson(), '/api/notes/${note.id}');
}

// ===== Aufgabe 4: UPDATE =====

Response replaceNote(Request request, String id) {
    if (noteRepo.findById(id) == null) {
        return notFound('Note not found');
    }

    final body = request.json;

    // PUT: Alle Pflichtfelder validieren
    final title = body['title'] as String?;
    final content = body['content'] as String?;

    if (title == null || title.isEmpty) {
        return badRequest('title is required');
    }
    if (content == null || content.isEmpty) {
        return badRequest('content is required');
    }

    final category = body['category'] as String?;
    final tags = (body['tags'] as List<dynamic>?)
        ?.map((e) => e as String)
        .toList() ?? [];
    final pinned = body['pinned'] as bool? ?? false;

    final note = noteRepo.replace(
        id,
        title: title,
        content: content,
        category: category,
        tags: tags,
        pinned: pinned,
    );
}
```

```
);

return jsonResponse(note!.toJson());
}

Response updateNote(Request request, String id) {
  if (noteRepo.findById(id) == null) {
    return notFound('Note not found');
  }

  final body = request.json;

  // PATCH: Nur angegebene Felder
  final tags = (body['tags'] as List<dynamic>?)
    ?.map((e) => e as String)
    .toList();

  final note = noteRepo.update(
    id,
    title: body['title'] as String?,
    content: body['content'] as String?,
    category: body['category'] as String?,
    tags: tags,
    pinned: body['pinned'] as bool?,
  );

  return jsonResponse(note!.toJson());
}

// ===== Aufgabe 5: DELETE =====

Response deleteNote(Request request, String id) {
  if (!noteRepo.delete(id)) {
    return notFound('Note not found');
  }
  return Response(204);
}

// ===== Aufgabe 6: Comments =====

Response listComments(Request request, String noteId) {
  if (noteRepo.findById(noteId) == null) {
    return notFound('Note not found');
  }

  final comments = commentRepo.findByNoteId(noteId);
  return jsonResponse({
    'data': comments.map((c) => c.toJson()).toList(),
    'total': comments.length,
  });
}
```

```

Response createComment(Request request, String noteId) {
    if (noteRepo.findById(noteId) == null) {
        return notFound('Note not found');
    }

    final body = request.json;

    final author = body['author'] as String?;
    final text = body['text'] as String?;

    if (author == null || author.isEmpty) {
        return badRequest('author is required');
    }
    if (text == null || text.isEmpty) {
        return badRequest('text is required');
    }

    final comment = commentRepo.create(
        noteId: noteId,
        author: author,
        text: text,
    );

    return created(comment.toJson(),
        ↵ ↵
        '/api/notes/$noteId/comments/${comment.id}');
}

Response deleteComment(Request request, String noteId, String commentId) {
    if (noteRepo.findById(noteId) == null) {
        return notFound('Note not found');
    }

    final comment = commentRepo.findById(commentId);
    if (comment == null || comment.noteId != noteId) {
        return notFound('Comment not found');
    }

    commentRepo.delete(commentId);
    return Response(204);
}

// =====
// Main
// =====

void main() async {
    final noteRepo = NoteRepository();
    final commentRepo = CommentRepository();
    final api = NotesApi(noteRepo, commentRepo);
}

```

```
// Seed-Daten
noteRepo.create(
  title: 'Welcome',
  content: 'This is your first note!',
  tags: ['welcome', 'getting-started'],
  pinned: true,
);
noteRepo.create(
  title: 'Meeting Notes',
  content: 'Agenda: ...',
  category: 'work',
  tags: ['meeting'],
);

final router = Router();

// Notes CRUD
router.get('/api/notes', api.listNotes);
router.get('/api/notes/<id>', api.getNote);
router.post('/api/notes', api.createNote);
router.put('/api/notes/<id>', api.replaceNote);
router.patch('/api/notes/<id>', api.updateNote);
router.delete('/api/notes/<id>', api.deleteNote);

// Comments (Sub-Ressource)
router.get('/api/notes/<noteId>/comments', api.listComments);
router.post('/api/notes/<noteId>/comments', api.createComment);
router.delete('/api/notes/<noteId>/comments/<commentId>', api.deleteComment);

final handler = Pipeline()
  .addMiddleware(logRequests())
  .addMiddleware(jsonBodyParser())
  .addHandler(router.call);

await shelf_io.serve(handler, 'localhost', 8080);
print('Server: http://localhost:8080');
print('');
print('Endpoints:');
print('  GET    /api/notes');
print('  GET    /api/notes/:id');
print('  POST   /api/notes');
print('  PUT    /api/notes/:id');
print('  PATCH  /api/notes/:id');
print('  DELETE /api/notes/:id');
print('  GET    /api/notes/:id/comments');
print('  POST   /api/notes/:id/comments');
print('  DELETE /api/notes/:id/comments/:commentId');
}
```

### 2.4.2.2 Test-Befehle

```
# ===== CREATE =====
curl -X POST http://localhost:8080/api/notes \
  -H "Content-Type: application/json" \
  -d '{"title": "Test Note", "content": "Hello World", "tags": ["test"]}'

# ===== READ ALL =====
curl http://localhost:8080/api/notes

# Mit Filter
curl "http://localhost:8080/api/notes?pinned=true"
curl "http://localhost:8080/api/notes?category=work"
curl "http://localhost:8080/api/notes?tag=meeting"

# ===== READ ONE =====
curl http://localhost:8080/api/notes/note-1

# 404 Test
curl http://localhost:8080/api/notes/not-exists

# ===== UPDATE (PATCH) =====
# Nur pinned ändern
curl -X PATCH http://localhost:8080/api/notes/note-1 \
  -H "Content-Type: application/json" \
  -d '{"pinned": false}'

# Nur title ändern
curl -X PATCH http://localhost:8080/api/notes/note-2 \
  -H "Content-Type: application/json" \
  -d '{"title": "Updated Meeting Notes"}'

# ===== UPDATE (PUT) =====
# Komplette ersetzen (category wird null weil nicht angegeben!)
curl -X PUT http://localhost:8080/api/notes/note-2 \
  -H "Content-Type: application/json" \
  -d '{"title": "Replaced Note", "content": "New content"}'

# ===== DELETE =====
curl -X DELETE http://localhost:8080/api/notes/note-3 -w "\nStatus:
↳  %{http_code}\n"

# ===== COMMENTS =====
# Kommentar erstellen
curl -X POST http://localhost:8080/api/notes/note-1/comments \
  -H "Content-Type: application/json" \
  -d '{"author": "Max", "text": "Great note!"}'

# Kommentare auflisten
curl http://localhost:8080/api/notes/note-1/comments
```

*# Kommentar löschen*

```
curl -X DELETE http://localhost:8080/api/notes/note-1/comments/comment-1
```

### 2.4.2.3 Ausgabe-Beispiele

Liste aller Notizen (200)

```
{
  "data": [
    {
      "id": "note-1",
      "title": "Welcome",
      "content": "This is your first note!",
      "tags": ["welcome", "getting-started"],
      "pinned": true,
      "createdAt": "2024-01-15T10:00:00.000Z"
    },
    {
      "id": "note-2",
      "title": "Meeting Notes",
      "content": "Agenda: ...",
      "category": "work",
      "tags": ["meeting"],
      "pinned": false,
      "createdAt": "2024-01-15T10:00:00.000Z"
    }
  ],
  "total": 2
}
```

Notiz erstellt (201)

```
{
  "id": "note-3",
  "title": "Test Note",
  "content": "Hello World",
  "tags": ["test"],
  "pinned": false,
  "createdAt": "2024-01-15T10:30:00.000Z"
}
```

PUT vs PATCH Unterschied

*# Original:*

```
{ "id": "note-2", "title": "Meeting", "content": "...", "category": "work",
  ↪  "pinned": false} ↵
```

*# Nach PATCH mit {"pinned": true}:*

```
{ "id": "note-2", "title": "Meeting", "content": "...", "category": "work",
  ↪  "pinned": true} ↵
```

*# -> Nur pinned geändert*

```
# Nach PUT mit {"title": "New", "content": "..."}:  
{ "id": "note-2", "title": "New", "content": "...", "pinned": false }  
# -> category ist weg (null), pinned auf default
```

#### 2.4.2.4 Wichtige Erkenntnisse

Repository-Pattern

```
// Trennung von Datenzugriff und API-Logik  
class NoteRepository {  
    List<Note> findAll();  
    Note? findById(String id);  
    Note create(...);  
    Note? update(...);  
    bool delete(String id);  
}  
  
// API-Handler nutzt Repository  
class NotesApi {  
    final NoteRepository repo;  
    // ...  
}
```

copyWith für PATCH

```
// Nur nicht-null Werte werden überschrieben  
Note copyWith({  
    String? title,  
    String? content,  
}) => Note(  
    id: id,  
    title: title ?? this.title, // Behalte alten Wert wenn null  
    content: content ?? this.content,  
);
```

Location-Header bei 201

```
Response created(Object data, String location) {  
    return Response(201,  
        body: jsonEncode(data),  
        headers: {  
            'content-type': 'application/json',  
            'location': location, // URL der neuen Ressource  
        },  
    );  
}
```

### 2.4.3 Ressourcen

#### 2.4.3.1 Offizielle Dokumentation

- HTTP Request Methods
- HTTP Status Codes

- REST API Design

### 2.4.3.2 Cheat Sheet: HTTP Methoden

Methode	Verwendung	Body?	Idempotent?
GET	Lesen	Nein	Ja
POST	Erstellen	Ja	Nein
PUT	Ersetzen	Ja	Ja
PATCH	Teilupdate	Ja	Meistens
DELETE	Löschen	Nein	Ja

### 2.4.3.3 Cheat Sheet: CRUD Mapping

```
// CREATE
router.post('/api/items', createItem);
// -> 201 Created + Location Header

// READ (all)
router.get('/api/items', listItems);
// -> 200 OK

// READ (one)
router.get('/api/items/<id>', getItem);
// -> 200 OK oder 404 Not Found

// UPDATE (replace)
router.put('/api/items/<id>', replaceItem);
// -> 200 OK oder 404 Not Found

// UPDATE (partial)
router.patch('/api/items/<id>', updateItem);
// -> 200 OK oder 404 Not Found

// DELETE
router.delete('/api/items/<id>', deleteItem);
// -> 204 No Content oder 404 Not Found
```

### 2.4.3.4 Cheat Sheet: Response Helper

```
Response jsonResponse(Object? data, {int statusCode = 200}) {
  return Response(
    statusCode,
    body: jsonEncode(data),
    headers: {'content-type': 'application/json'},
  );
}

Response notFound(String message) {
  return jsonResponse({'error': message}, statusCode: 404);
}
```

```
}

Response badRequest(String message) {
    return jsonResponse({'error': message}, statusCode: 400);
}

Response created(Object data, String location) {
    return Response(201,
        body: jsonEncode(data),
        headers: {
            'content-type': 'application/json',
            'location': location,
        },
    );
}

Response noContent() => Response(204);
```

#### 2.4.3.5 Cheat Sheet: Model mit copyWith

```
class Item {
    final String id;
    final String name;
    final double price;
    final bool active;

    Item({
        required this.id,
        required this.name,
        required this.price,
        this.active = true,
    });

    // Für PATCH: nur angegebene Felder ändern
    Item copyWith({
        String? name,
        double? price,
        bool? active,
    }) => Item(
        id: id, // ID bleibt immer gleich
        name: name ?? this.name,
        price: price ?? this.price,
        active: active ?? this.active,
    );

    Map<String, dynamic> toJson() => {
        'id': id,
        'name': name,
        'price': price,
        'active': active,
    };
}
```

```
};  
}
```

### 2.4.3.6 Cheat Sheet: Repository Pattern

```
abstract class Repository<T> {  
    List<T> findAll();  
    T? findById(String id);  
    T create(T item);  
    T? update(String id, T item);  
    bool delete(String id);  
}  
  
class InMemoryRepository<T> implements Repository<T> {  
    final _items = <String, T>{};  
  
    @override  
    List<T> findAll() => _items.values.toList();  
  
    @override  
    T? findById(String id) => _items[id];  
  
    @override  
    T create(T item) {  
        // ID-Generierung hier oder im Aufrufer  
        _items[item.id] = item;  
        return item;  
    }  
  
    @override  
    T? update(String id, T item) {  
        if (!_items.containsKey(id)) return null;  
        _items[id] = item;  
        return item;  
    }  
  
    @override  
    bool delete(String id) => _items.remove(id) != null;  
}
```

### 2.4.3.7 Cheat Sheet: PUT vs PATCH

```
// PUT: Komplette ersetzen (alle Felder nötig)  
Future<Response> replaceItem(Request request, String id) async {  
    final body = request.json;  
  
    // Validierung: alle Felder müssen da sein  
    if (body['name'] == null || body['price'] == null) {  
        return badRequest('name and price are required');  
    }  
}
```

```

}

// Komplette neues Objekt erstellen
final item = Item(
    id: id,
    name: body['name'],
    price: body['price'],
    active: body['active'] ?? true, // Default wenn nicht angegeben
);

return jsonResponse(repo.update(id, item)!.toJson());
}

// PATCH: Teilweise aktualisieren (nur angegebene Felder)
Future<Response> updateItem(Request request, String id) async {
    final existing = repo.findById(id);
    if (existing == null) return notFound('Item not found');

    final body = request.json;

    // Nur angegebene Felder überschreiben
    final updated = existing.copyWith(
        name: body['name'],
        price: body['price'],
        active: body['active'],
    );

    return jsonResponse(repo.update(id, updated)!.toJson());
}

```

### 2.4.3.8 HTTP Statuscodes

Erfolg (2xx)

Code	Name	Verwendung
200	OK	Erfolgreiche GET/PUT/PATCH
201	Created	Erfolgreiche POST
204	No Content	Erfolgreiche DELETE

Client-Fehler (4xx)

Code	Name	Verwendung
400	Bad Request	Ungültige Daten
404	Not Found	Ressource existiert nicht
409	Conflict	Konflikt (z.B. Duplikat)
422	Unprocessable Entity	Semantischer Fehler

Server-Fehler (5xx)

Code	Name	Verwendung
500	Internal Server Error	Unerwarteter Fehler
503	Service Unavailable	Server überlastet

### 2.4.3.9 Test-Befehle

```
# CREATE
curl -X POST http://localhost:8080/api/items \
  -H "Content-Type: application/json" \
  -d '{"name": "Test", "price": 9.99}'

# READ (all)
curl http://localhost:8080/api/items

# READ (one)
curl http://localhost:8080/api/items/1

# UPDATE (PUT - replace)
curl -X PUT http://localhost:8080/api/items/1 \
  -H "Content-Type: application/json" \
  -d '{"name": "Updated", "price": 19.99, "active": true}'

# UPDATE (PATCH - partial)
curl -X PATCH http://localhost:8080/api/items/1 \
  -H "Content-Type: application/json" \
  -d '{"price": 14.99}'

# DELETE
curl -X DELETE http://localhost:8080/api/items/1

# DELETE mit Response-Code anzeigen
curl -X DELETE http://localhost:8080/api/items/1 -w "\n%{http_code}\n"
```

### 2.4.3.10 Best Practices

- Konsistente URL-Struktur**
  - Plural für Collections: `/api/users`
  - ID für einzelne Ressourcen: `/api/users/:id`
- Korrekte Statuscodes**
  - Nicht 200 für alles
  - 201 bei Erstellung
  - 204 bei Löschung
- Location-Header bei POST**
  - URL der neuen Ressource zurückgeben
- Idempotenz beachten**
  - PUT/DELETE sollten idempotent sein
- Konsistente Fehler-Responses**
  - Immer JSON zurückgeben
  - Aussagekräftige Fehlermeldungen

## 2.5 Einheit 6.5: Input Validierung

### 2.5.0.1 Lernziele

Nach dieser Einheit kannst du: - Eingabedaten systematisch validieren - Validierungsregeln definieren und anwenden - Aussagekräftige Fehlermeldungen zurückgeben - Wiederverwendbare Validatoren erstellen

### 2.5.0.2 Warum Input-Validierung?

Gefahren ohne Validierung

```
// Ohne Validierung
Future<Response> createUser(Request request) async {
    final body = request.json;

    // Was passiert hier wenn...
    // - email ist keine gültige E-Mail?
    // - age ist negativ?
    // - password hat nur 1 Zeichen?
    // - name enthält <script>-Tags?

    final user = User(
        name: body['name'],
        email: body['email'],
        age: body['age'],
        password: body['password'],
    );

    // Speichern mit ungültigen Daten!
    await db.save(user);
}
```

Validierung schützt vor

1. **Ungültigen Daten** in der Datenbank
2. **Security-Problemen** (SQL Injection, XSS)
3. **Unerwarteten Fehlern** im Backend
4. **Schlechter Benutzererfahrung** (späte Fehlermeldungen)

### 2.5.0.3 Einfache Validierung

Inline-Validierung

```
Future<Response> createUser(Request request) async {
    final body = request.json;

    // Pflichtfelder
    final name = body['name'] as String?;
    if (name == null || name.isEmpty) {
        return badRequest('name is required');
    }

    final email = body['email'] as String?;
```

```

if (email == null || email.isEmpty) {
    return badRequest('email is required');
}

// Format-Validierung
if (!email.contains('@')) {
    return badRequest('email must be a valid email address');
}

// Längen-Validierung
final password = body['password'] as String?;
if (password == null || password.length < 8) {
    return badRequest('password must be at least 8 characters');
}

// Bereichs-Validierung
final age = body['age'] as int?;
if (age != null && (age < 0 || age > 150)) {
    return badRequest('age must be between 0 and 150');
}

// ... User erstellen
}

```

**Problem:** Code wird schnell unübersichtlich!

#### 2.5.0.4 Strukturierte Validierung

ValidationResult Klasse

```

class ValidationResult {
    final bool isValid;
    final List<ValidationError> errors;

    ValidationResult.valid()
        : isValid = true,
          errors = const [];

    ValidationResult.invalid(this.errors) : isValid = false;

    factory ValidationResult.fromErrors(List<ValidationError> errors) {
        if (errors.isEmpty) {
            return ValidationResult.valid();
        }
        return ValidationResult.invalid(errors);
    }

    Map<String, dynamic> toJson() => {
        'valid': isValid,
        'errors': errors.map((e) => e.toJson()).toList(),
    };
}

```

```
class ValidationError {
    final String field;
    final String message;
    final String code;

    ValidationError({
        required this.field,
        required this.message,
        required this.code,
    });

    Map<String, dynamic> toJson() => {
        'field': field,
        'message': message,
        'code': code,
    };
}
```

Validator-Klasse

```
class Validator {
    final List<ValidationError> _errors = [];
    final Map<String, dynamic> data;

    Validator(this.data);

    // Pflichtfeld
    Validator required(String field, {String? message}) {
        final value = data[field];
        if (value == null || (value is String && value.isEmpty)) {
            _errors.add(ValidationError(
                field: field,
                message: message ?? '$field is required',
                code: 'REQUIRED',
            ));
        }
        return this;
    }

    // String-Länge
    Validator minLength(String field, int length, {String? message}) {
        final value = data[field];
        if (value is String && value.length < length) {
            _errors.add(ValidationError(
                field: field,
                message: message ?? '$field must be at least $length characters',
                code: 'MIN_LENGTH',
            ));
        }
        return this;
    }
}
```

```
}

Validator maxLength(String field, int length, {String? message}) {
  final value = data[field];
  if (value is String && value.length > length) {
    _errors.add(ValidationError(
      field: field,
      message: message ?? '$field must be at most $length characters',
      code: 'MAX_LENGTH',
    ));
  }
  return this;
}

// E-Mail Format
Validator email(String field, {String? message}) {
  final value = data[field];
  if (value is String && value.isNotEmpty) {
    final emailRegex = RegExp(r'^[\w-\.]@([\w-]+\.)+[\w-]{2,4}$');
    if (!emailRegex.hasMatch(value)) {
      _errors.add(ValidationError(
        field: field,
        message: message ?? '$field must be a valid email',
        code: 'INVALID_EMAIL',
      ));
    }
  }
  return this;
}

// Zahlenbereich
Validator range(String field, {int? min, int? max, String? message}) {
  final value = data[field];
  if (value is num) {
    if (min != null && value < min) {
      _errors.add(ValidationError(
        field: field,
        message: message ?? '$field must be at least $min',
        code: 'MIN_VALUE',
      ));
    }
    if (max != null && value > max) {
      _errors.add(ValidationError(
        field: field,
        message: message ?? '$field must be at most $max',
        code: 'MAX_VALUE',
      ));
    }
  }
  return this;
}
```

```

// Enum-Werte
Validator oneOf(String field, List<String> values, {String? message}) {
  final value = data[field];
  if (value is String && !values.contains(value)) {
    _errors.add(ValidationError(
      field: field,
      message: message ?? '$field must be one of: ${values.join(', ')}',
      code: 'INVALID_VALUE',
    ));
  }
  return this;
}

// Regex-Pattern
Validator pattern(String field, RegExp regex, {String? message}) {
  final value = data[field];
  if (value is String && value.isNotEmpty && !regex.hasMatch(value)) {
    _errors.add(ValidationError(
      field: field,
      message: message ?? '$field has invalid format',
      code: 'INVALID_FORMAT',
    ));
  }
  return this;
}

// Custom Validation
Validator custom(String field, bool Function(dynamic) validator, {String? ↵
↵ message, String code = 'CUSTOM'}) {
  final value = data[field];
  if (value != null && !validator(value)) {
    _errors.add(ValidationError(
      field: field,
      message: message ?? '$field is invalid',
      code: code,
    ));
  }
  return this;
}

ValidationResult validate() {
  return ValidationResult.fromErrors(_errors);
}
}

```

### Verwendung

```

Future<Response> createUser(Request request) async {
  final body = request.json;

```

```

final result = Validator(body)
    .required('name')
    .minLength('name', 2)
    .maxLength('name', 100)
    .required('email')
    .email('email')
    .required('password')
    .minLength('password', 8)
    .range('age', min: 0, max: 150)
    .oneOf('role', ['user', 'admin', 'moderator'])
    .validate();

if (!result.isValid) {
    return Response(400,
        body: jsonEncode({
            'error': 'Validation failed',
            'details': result.errors.map((e) => e.toJson()).toList(),
        }),
        headers: {'content-type': 'application/json'},
    );
}

// Validierte Daten verwenden
final user = User(
    name: body['name'],
    email: body['email'],
    password: body['password'],
    age: body['age'],
    role: body['role'] ?? 'user',
);

// ...
}

```

### 2.5.0.5 Request-spezifische Validierung

Unterschiedliche Regeln für Create/Update

```

class UserValidator {
    static ValidationResult validateCreate(Map<String, dynamic> data) {
        return Validator(data)
            .required('name')
            .minLength('name', 2)
            .required('email')
            .email('email')
            .required('password')
            .minLength('password', 8)
            .validate();
    }

    static ValidationResult validateUpdate(Map<String, dynamic> data) {

```

```
// Bei Update: Felder optional, aber wenn vorhanden dann validieren
final v = Validator(data);

if (data.containsKey('name')) {
    v.minLength('name', 2).maxLength('name', 100);
}

if (data.containsKey('email')) {
    v.email('email');
}

if (data.containsKey('password')) {
    v.minLength('password', 8);
}

return v.validate();
}
}

// Verwendung
router.post('/users', (Request request) {
    final result = UserValidator.validateCreate(request.json);
    if (!result.isValid) {
        return validationError(result);
    }
    // ...
});

router.patch('/users/<id>', (Request request, String id) {
    final result = UserValidator.validateUpdate(request.json);
    if (!result.isValid) {
        return validationError(result);
    }
    // ...
});
```

### 2.5.0.6 Fehler-Response Format

RFC 7807 Problem Details

```
Response validationError(ValidationResult result) {
    return Response(400,
        body: jsonEncode({
            'type': 'https://api.example.com/errors/validation',
            'title': 'Validation Error',
            'status': 400,
            'detail': 'One or more fields failed validation',
            'errors': result.errors.map((e) => {
                return {
                    'field': e.field,
                    'message': e.message,
```

```

        'code': e.code,
    };
    }).toList(),
  )),
  headers: {
    'content-type': 'application/problem+json',
  },
);
}

```

Beispiel-Response

```

{
  "type": "https://api.example.com/errors/validation",
  "title": "Validation Error",
  "status": 400,
  "detail": "One or more fields failed validation",
  "errors": [
    {
      "field": "email",
      "message": "email must be a valid email",
      "code": "INVALID_EMAIL"
    },
    {
      "field": "password",
      "message": "password must be at least 8 characters",
      "code": "MIN_LENGTH"
    }
  ]
}

```

### 2.5.0.7 Verschachtelte Objekte validieren

```

Validator nestedValidator(String parentField, Map<String, dynamic>? nested) {
  return Validator(nested ?? {});
}

```

```

Future<Response> createOrder(Request request) async {
  final body = request.json;

```

```

  // Haupt-Validierung
  final mainResult = Validator(body)
    .required('customer')
    .required('items')
    .validate();

```

```

  if (!mainResult.isValid) {
    return validationError(mainResult);
  }

```

```

  // Verschachtelte Validierung: Customer

```

```
final customer = body['customer'] as Map<String, dynamic>;
final customerResult = Validator(customer ?? {})
    .required('name')
    .required('email')
    .email('email')
    .validate();

if (!customerResult.isValid) {
    // Fehler mit Prefix versehen
    final prefixedErrors = customerResult.errors.map((e) => ValidationError(
        field: 'customer.${e.field}',
        message: e.message,
        code: e.code,
    )).toList();
    return validationError(ValidationResult.invalid(prefixedErrors));
}

// Array-Validierung: Items
final items = body['items'] as List<dynamic>;
if (items == null || items.isEmpty) {
    return badRequest('items must not be empty');
}

for (var i = 0; i < items.length; i++) {
    final item = items[i] as Map<String, dynamic>;
    final itemResult = Validator(item ?? {})
        .required('productId')
        .required('quantity')
        .range('quantity', min: 1)
        .validate();

    if (!itemResult.isValid) {
        final prefixedErrors = itemResult.errors.map((e) => ValidationError(
            field: 'items[$i].${e.field}',
            message: e.message,
            code: e.code,
        )).toList();
        return validationError(ValidationResult.invalid(prefixedErrors));
    }
}

// Alle Validierungen bestanden
// ...
}
```

### 2.5.0.8 Zusammenfassung

Regel	Code
Pflichtfeld	<code>.required('field')</code>
Min-Länge	<code>.minLength('field', 8)</code>

Regel	Code
Max-Länge	<code>.maxLength('field', 100)</code>
E-Mail	<code>.email('field')</code>
Zahlenbereich	<code>.range('field', min: 0, max: 100)</code>
Erlaubte Werte	<code>.oneOf('field', ['a', 'b', 'c'])</code>
Regex-Pattern	<code>.pattern('field', regex)</code>
Custom	<code>.custom('field', (v) =&gt; v &gt; 0)</code>

### 2.5.0.9 Nächste Schritte

In der nächsten Einheit lernst du **Error Handling & HTTP-Statuscodes**: Wie du Fehler systematisch behandelst und die richtigen HTTP-Codes verwendest.

## 2.5.1 Übung

### 2.5.1.1 Ziel

Implementiere eine robuste Validierung für eine Benutzerregistrierung-API.

### 2.5.1.2 Aufgabe 1: Validator-Klasse (20 min)

Erstelle eine wiederverwendbare Validator-Klasse.

Anforderungen

```
class Validator {
    Validator required(String field, {String? message});
    Validator minLength(String field, int length, {String? message});
    Validator maxLength(String field, int length, {String? message});
    Validator email(String field, {String? message});
    Validator range(String field, {int? min, int? max, String? message});
    Validator oneOf(String field, List<String> values, {String? message});
    Validator pattern(String field, RegExp regex, {String? message});
    ValidationResult validate();
}

class ValidationResult {
    final bool isValid;
    final List<ValidationError> errors;
}

class ValidationError {
    final String field;
    final String message;
    final String code;
}
```

Test

```
final result = Validator({'name': '', 'age': 200})
    .required('name')
    .range('age', min: 0, max: 150)
    .validate();
```

```
print(result.isValid); // false
print(result.errors.length); // 2
```

### 2.5.1.3 Aufgabe 2: User Registration (15 min)

Erstelle einen Registrierungs-Endpoint mit vollständiger Validierung.

Endpoint

POST /api/register

Request Body

```
{
  "username": "max_mustermann",
  "email": "max@example.com",
  "password": "SecurePass123!",
  "confirmPassword": "SecurePass123!",
  "age": 25,
  "role": "user"
}
```

Validierungsregeln

Feld	Regeln
username	Pflicht, 3-20 Zeichen, alphanumerisch + underscore
email	Pflicht, gültiges E-Mail-Format
password	Pflicht, min. 8 Zeichen, min. 1 Großbuchstabe, 1 Zahl
confirmPassword	Muss mit password übereinstimmen
age	Optional, 13-120
role	Optional, default "user", erlaubt: user, admin

Erfolg (201)

```
{
  "id": "user-1",
  "username": "max_mustermann",
  "email": "max@example.com",
  "age": 25,
  "role": "user",
  "createdAt": "2024-01-15T10:00:00Z"
}
```

Fehler (400)

```
{
  "error": "Validation failed",
  "details": [
    {
      "field": "username",
      "message": "username must be between 3 and 20 characters",
    }
  ]
}
```

```

    "code": "LENGTH"
  },
  {
    "field": "password",
    "message": "password must contain at least one uppercase letter",
    "code": "WEAK_PASSWORD"
  }
]
}

```

### 2.5.1.4 Aufgabe 3: Passwort-Validierung (10 min)

Erstelle einen speziellen Passwort-Validator.

Anforderungen

```

Validator password(String field, {
  int minLength = 8,
  bool requireUppercase = true,
  bool requireLowercase = true,
  bool requireDigit = true,
  bool requireSpecial = false,
});

```

Fehlermeldungen

- “password must be at least 8 characters”
- “password must contain at least one uppercase letter”
- “password must contain at least one lowercase letter”
- “password must contain at least one digit”
- “password must contain at least one special character”

### 2.5.1.5 Aufgabe 4: Custom Validators (10 min)

Implementiere benutzerdefinierte Validierungsregeln.

Beispiele

```

// Passwörter müssen übereinstimmen
Validator matches(String field, String otherField, {String? message});

// Feld darf nicht vorhanden sein wenn anderes Feld gesetzt
Validator excludesWith(String field, String otherField, {String? message});

// Datum muss in der Zukunft liegen
Validator futureDate(String field, {String? message});

// Eindeutigkeit (mit Callback)
Validator unique(String field, Future<bool> Function(String) checkExists,
  ↪ {String? message});

```

Verwendung

```
final result = Validator(body)
    .required('password')
    .required('confirmPassword')
    .matches('confirmPassword', 'password', message: 'Passwords must match')
    .validate();
```

### 2.5.1.6 Aufgabe 5: Verschachtelte Validierung (15 min)

Validiere komplexe, verschachtelte Objekte.

Request Body

```
{
  "company": {
    "name": "Acme Corp",
    "taxId": "DE123456789",
    "address": {
      "street": "Hauptstraße 1",
      "city": "Berlin",
      "zip": "10115",
      "country": "DE"
    }
  },
  "contact": {
    "name": "Max Mustermann",
    "email": "max@acme.com",
    "phone": "+49 30 12345678"
  },
  "employees": [
    {"name": "Anna", "role": "developer"},
    {"name": "Bob", "role": "designer"}
  ]
}
```

Validierungsregeln

**Company:** - name: Pflicht, 2-100 Zeichen - taxId: Pflicht, Format: 2 Buchstaben + 9 Ziffern

**Address:** - street: Optional - city: Pflicht - zip: Pflicht, 5 Ziffern - country: Pflicht, 2 Buchstaben (ISO-Code)

**Contact:** - name: Pflicht - email: Pflicht, gültiges Format - phone: Optional

**Employees:** - Mindestens 1 Mitarbeiter - Jeder braucht name (Pflicht) und role (Pflicht, oneOf: developer, designer, manager)

Fehlermeldungen mit Pfaden

```
{
  "errors": [
    {"field": "company.address.zip", "message": "zip must be 5 digits"},
    {"field": "employees[1].role", "message": "role must be one of:
      ↪ developer, designer, manager"}
  ]
}
```

### 2.5.1.7 Aufgabe 6: Validator-Middleware (Bonus, 10 min)

Erstelle eine Middleware, die automatisch validiert.

Verwendung

```
// Schema definieren
final userSchema = {
  'name': [required(), minLength(2)],
  'email': [required(), email()],
  'age': [range(min: 0, max: 150)],
};

// Middleware anwenden
router.post('/users',
  validate(userSchema), // Middleware
  createUser,           // Handler
);
```

Middleware

```
Middleware validate(Map<String, List<ValidationRule>> schema) {
  return (Handler handler) {
    return (Request request) async {
      final body = request.json;
      final errors = <ValidationError>[];

      for (final entry in schema.entries) {
        final field = entry.key;
        final rules = entry.value;

        for (final rule in rules) {
          final error = rule.validate(field, body[field]);
          if (error != null) {
            errors.add(error);
            break; // Stop at first error for this field
          }
        }
      }

      if (errors.isNotEmpty) {
        return validationError(ValidationResult.invalid(errors));
      }

      return handler(request);
    };
  };
}
```

### 2.5.1.8 Testen

```
# Erfolgreiche Registrierung
curl -X POST http://localhost:8080/api/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "max_m",
    "email": "max@test.de",
    "password": "SecurePass1",
    "confirmPassword": "SecurePass1"
  }'

# Validierungsfehler
curl -X POST http://localhost:8080/api/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "ab",
    "email": "invalid",
    "password": "weak"
  }'

# Passwörter stimmen nicht überein
curl -X POST http://localhost:8080/api/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "max_m",
    "email": "max@test.de",
    "password": "SecurePass1",
    "confirmPassword": "DifferentPass"
  }'
```

### 2.5.1.9 Abgabe-Checkliste

- ☐ Validator-Klasse mit allen Basis-Methoden
- ☐ ValidationResult und ValidationError implementiert
- ☐ User Registration Endpoint funktioniert
- ☐ Passwort-Validierung mit Komplexitätsregeln
- ☐ Custom Validator: matches() funktioniert
- ☐ Verschachtelte Objekte werden validiert
- ☐ Fehler-Response im korrekten Format
- ☐ (Bonus) Validator-Middleware

## 2.5.2 Lösung

### 2.5.2.1 Vollständige Lösung

```
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';
```

```
// =====
// Aufgabe 1: Validation Classes
// =====

class ValidationError {
    final String field;
    final String message;
    final String code;

    ValidationError({
        required this.field,
        required this.message,
        required this.code,
    });

    Map<String, dynamic> toJson() => {
        'field': field,
        'message': message,
        'code': code,
    };
}

class ValidationResult {
    final bool isValid;
    final List<ValidationError> errors;

    ValidationResult.valid()
        : isValid = true,
          errors = const [];

    ValidationResult.invalid(this.errors) : isValid = false;

    factory ValidationResult.fromErrors(List<ValidationError> errors) {
        if (errors.isEmpty) {
            return ValidationResult.valid();
        }
        return ValidationResult.invalid(errors);
    }
}

class Validator {
    final List<ValidationError> _errors = [];
    final Map<String, dynamic> data;
    final String _prefix;

    Validator(this.data, {String prefix = ''}) : _prefix = prefix;

    String _fieldName(String field) => _prefix.isEmpty ? field : '$_prefix.$field';

    // Pflichtfeld
    Validator required(String field, {String? message}) {
```

```

    final value = data[field];
    if (value == null || (value is String && value.isEmpty)) {
        _errors.add(ValidationError(
            field: _fieldName(field),
            message: message ?? '${_fieldName(field)} is required',
            code: 'REQUIRED',
        ));
    }
    return this;
}

// String-Länge
Validator minLength(String field, int length, {String? message}) {
    final value = data[field];
    if (value is String && value.isNotEmpty && value.length < length) {
        _errors.add(ValidationError(
            field: _fieldName(field),
            message: message ?? '${_fieldName(field)} must be at least $length
↪ characters',
↪ code: 'MIN_LENGTH',
        ));
    }
    return this;
}

Validator maxLength(String field, int length, {String? message}) {
    final value = data[field];
    if (value is String && value.length > length) {
        _errors.add(ValidationError(
            field: _fieldName(field),
            message: message ?? '${_fieldName(field)} must be at most $length
↪ characters',
↪ code: 'MAX_LENGTH',
        ));
    }
    return this;
}

// E-Mail
Validator email(String field, {String? message}) {
    final value = data[field];
    if (value is String && value.isNotEmpty) {
        final regex = RegExp(r'^[\w-\.\.]+@([\w-]+\.\.)+[\w-]{2,4}$');
        if (!regex.hasMatch(value)) {
            _errors.add(ValidationError(
                field: _fieldName(field),
                message: message ?? '${_fieldName(field)} must be a valid email',
                code: 'INVALID_EMAIL',
            ));
        }
    }
}

```

```
    return this;
}

// Zahlenbereich
Validator range(String field, {int? min, int? max, String? message}) {
    final value = data[field];
    if (value is num) {
        if (min != null && value < min) {
            _errors.add(ValidationError(
                field: _fieldName(field),
                message: message ?? '${_fieldName(field)} must be at least $min',
                code: 'MIN_VALUE',
            ));
        }
        if (max != null && value > max) {
            _errors.add(ValidationError(
                field: _fieldName(field),
                message: message ?? '${_fieldName(field)} must be at most $max',
                code: 'MAX_VALUE',
            ));
        }
    }
    return this;
}

// Erlaubte Werte
Validator oneOf(String field, List<String> values, {String? message}) {
    final value = data[field];
    if (value is String && value.isNotEmpty && !values.contains(value)) {
        _errors.add(ValidationError(
            field: _fieldName(field),
            message: message ?? '${_fieldName(field)} must be one of:
↪ ${values.join(', ')}',
            code: 'INVALID_VALUE',
        ));
    }
    return this;
}

// Regex-Pattern
Validator pattern(String field, RegExp regex, {String? message, String code ↪
↪ = 'INVALID_FORMAT'}) {
    final value = data[field];
    if (value is String && value.isNotEmpty && !regex.hasMatch(value)) {
        _errors.add(ValidationError(
            field: _fieldName(field),
            message: message ?? '${_fieldName(field)} has invalid format',
            code: code,
        ));
    }
    return this;
}
```

```

}

// =====
// Aufgabe 3: Passwort-Validierung
// =====

Validator password(String field, {
  int minLength = 8,
  bool requireUppercase = true,
  bool requireLowercase = true,
  bool requireDigit = true,
  bool requireSpecial = false,
}) {
  final value = data[field];
  if (value is String && value.isNotEmpty) {
    if (value.length < minLength) {
      _errors.add(ValidationError(
        field: _fieldName(field),
        message: '${_fieldName(field)} must be at least $minLength characters',
        code: 'WEAK_PASSWORD',
      ));
    }
    if (requireUppercase && !RegExp(r'[A-Z]').hasMatch(value)) {
      _errors.add(ValidationError(
        field: _fieldName(field),
        message: '${_fieldName(field)} must contain at least one uppercase ↵
↵ letter',
        code: 'WEAK_PASSWORD',
      ));
    }
    if (requireLowercase && !RegExp(r'[a-z]').hasMatch(value)) {
      _errors.add(ValidationError(
        field: _fieldName(field),
        message: '${_fieldName(field)} must contain at least one lowercase ↵
↵ letter',
        code: 'WEAK_PASSWORD',
      ));
    }
    if (requireDigit && !RegExp(r'\d').hasMatch(value)) {
      _errors.add(ValidationError(
        field: _fieldName(field),
        message: '${_fieldName(field)} must contain at least one digit',
        code: 'WEAK_PASSWORD',
      ));
    }
    if (requireSpecial && !RegExp(r'[!@#$$%^&*(),.?":{}|<>]').hasMatch(value)) {
      _errors.add(ValidationError(
        field: _fieldName(field),
        message: '${_fieldName(field)} must contain at least one special ↵
↵ character',
        code: 'WEAK_PASSWORD',
      ));
    }
  }
}

```

```

    ));
    }
    }
    return this;
}

// =====
// Aufgabe 4: Custom Validators
// =====

Validator matches(String field, String otherField, {String? message}) {
    final value = data[field];
    final otherValue = data[otherField];
    if (value != null && otherValue != null && value != otherValue) {
        _errors.add(ValidationError(
            field: _fieldName(field),
            message: message ?? '${_fieldName(field)} must match
↪  ${_fieldName(otherField)}',
            code: 'MISMATCH',
        ));
    }
    return this;
}

Validator custom(
    String field,
    bool Function(dynamic) validator, {
    String? message,
    String code = 'CUSTOM',
}) {
    final value = data[field];
    if (value != null && !validator(value)) {
        _errors.add(ValidationError(
            field: _fieldName(field),
            message: message ?? '${_fieldName(field)} is invalid',
            code: code,
        ));
    }
    return this;
}

// =====
// Aufgabe 5: Verschachtelte Validierung
// =====

Validator nested(String field, void Function(Validator) validateNested) {
    final value = data[field];
    if (value is Map<String, dynamic>) {
        final nestedValidator = Validator(value, prefix: _fieldName(field));
        validateNested(nestedValidator);
        _errors.addAll(nestedValidator._errors);
    }
}

```

```

    }
    return this;
}

Validator array(String field, void Function(Validator, int) validateItem, ↪
↪ {int? minItems}) {
    final value = data[field];
    if (value is List) {
        if (minItems != null && value.length < minItems) {
            _errors.add(ValidationError(
                field: _fieldName(field),
                message: '${_fieldName(field)} must have at least $minItems items',
                code: 'MIN_ITEMS',
            ));
        }
        for (var i = 0; i < value.length; i++) {
            final item = value[i];
            if (item is Map<String, dynamic>) {
                final itemValidator = Validator(item, prefix: ↪
↪ '${_fieldName(field)}[${i}]');
                validateItem(itemValidator, i);
                _errors.addAll(itemValidator._errors);
            }
        }
    }
    return this;
}

ValidationResult validate() {
    return ValidationResult.fromErrors(_errors);
}
}

// =====
// Helper Functions
// =====

Response jsonResponse(Object? data, {int statusCode = 200}) {
    return Response(
        statusCode,
        body: jsonEncode(data),
        headers: {'content-type': 'application/json'},
    );
}

Response validationError(ValidationResult result) {
    return Response(400,
        body: jsonEncode({
            'error': 'Validation failed',
            'details': result.errors.map((e) => e.toJson()).toList(),
        })),

```

```

        headers: {'content-type': 'application/json'},
    );
}

extension RequestJson on Request {
    Map<String, dynamic> get json {
        final body = context['body'];
        return body is Map<String, dynamic> ? body : {};
    }
}

Middleware jsonBodyParser() {
    return (Handler handler) {
        return (Request request) async {
            if (!['POST', 'PUT', 'PATCH'].contains(request.method)) {
                return handler(request);
            }
            final contentType = request.headers['content-type'] ?? '';
            if (!contentType.contains('application/json')) {
                return handler(request);
            }
            final body = await request.readAsString();
            if (body.isEmpty) {
                return handler(request.change(context: {...request.context, 'body':
↵ <String, dynamic>{}}));
            }
            try {
                final json = jsonDecode(body);
                return handler(request.change(context: {...request.context, 'body':
↵ json}));
            } on FormatException {
                return Response(400, body: '{"error": "Invalid JSON"}', headers:
↵ {'content-type': 'application/json'});
            }
        };
    };
}

// =====
// Storage
// =====

final _users = <String, Map<String, dynamic>>{};
var _nextUserId = 1;

// =====
// Aufgabe 2: User Registration
// =====

Response registerUser(Request request) {
    final body = request.json;

```

```
// Username Pattern: alphanumerisch + underscore
final usernamePattern = RegExp(r'^[a-zA-Z0-9_]+$');

final result = Validator(body)
    .required('username')
    .minLength('username', 3)
    .maxLength('username', 20)
    .pattern('username', usernamePattern,
        message: 'username must contain only letters, numbers, and ↵
↳ underscores',
        code: 'INVALID_USERNAME')
    .required('email')
    .email('email')
    .required('password')
    .password('password',
        minLength: 8,
        requireUppercase: true,
        requireLowercase: true,
        requireDigit: true)
    .required('confirmPassword')
    .matches('confirmPassword', 'password', message: 'Passwords must match')
    .range('age', min: 13, max: 120)
    .oneOf('role', ['user', 'admin'])
    .validate();

if (!result.isValid) {
    return validationError(result);
}

// User erstellen
final id = 'user-${_nextUserId++}';
final user = {
    'id': id,
    'username': body['username'],
    'email': body['email'],
    'age': body['age'],
    'role': body['role'] ?? 'user',
    'createdAt': DateTime.now().toUtc().toIso8601String(),
};

_users[id] = user;

return Response(201,
    body: jsonEncode(user),
    headers: {
        'content-type': 'application/json',
        'location': '/api/users/$id',
    },
);
}
```

```
// =====
// Aufgabe 5: Verschachtelte Validierung
// =====

Response createCompany(Request request) {
    final body = request.json;

    // Tax ID Pattern: 2 Buchstaben + 9 Ziffern
    final taxIdPattern = RegExp(r'^[A-Z]{2}\d{9}$');
    // ZIP Pattern: 5 Ziffern
    final zipPattern = RegExp(r'^\d{5}$');
    // Country Pattern: 2 Buchstaben
    final countryPattern = RegExp(r'^[A-Z]{2}$');

    final result = Validator(body)
        .required('company')
        .nested('company', (v) {
            v.required('name')
                .minLength('name', 2)
                .maxLength('name', 100)
                .required('taxId')
                .pattern('taxId', taxIdPattern,
                    message: 'taxId must be 2 letters followed by 9 digits')
                .required('address')
                .nested('address', (addr) {
                    addr.required('city')
                        .required('zip')
                        .pattern('zip', zipPattern, message: 'zip must be 5 digits')
                        .required('country')
                        .pattern('country', countryPattern, message: 'country must be 2 ↵
↵ letter ISO code');
                });
            });
        .required('contact')
        .nested('contact', (v) {
            v.required('name')
                .required('email')
                .email('email');
            });
        .required('employees')
        .array('employees', (v, i) {
            v.required('name')
                .required('role')
                .oneOf('role', ['developer', 'designer', 'manager']);
        }, minItems: 1)
        .validate();

    if (!result.isValid) {
        return validationError(result);
    }
}
```

```
return jsonResponse({
    'message': 'Company created successfully',
    'data': body,
}, statusCode: 201);
}

// =====
// Aufgabe 6: Validator Middleware (Bonus)
// =====

typedef ValidationRule = ValidationError? Function(String field, dynamic value);

ValidationRule requiredRule({String? message}) {
    return (field, value) {
        if (value == null || (value is String && value.isEmpty)) {
            return ValidationError(
                field: field,
                message: message ?? '$field is required',
                code: 'REQUIRED',
            );
        }
        return null;
    };
}

ValidationRule minLengthRule(int length, {String? message}) {
    return (field, value) {
        if (value is String && value.isNotEmpty && value.length < length) {
            return ValidationError(
                field: field,
                message: message ?? '$field must be at least $length characters',
                code: 'MIN_LENGTH',
            );
        }
        return null;
    };
}

ValidationRule emailRule({String? message}) {
    return (field, value) {
        if (value is String && value.isNotEmpty) {
            final regex = RegExp(r'^[\w-\.] + @([\w-]+\w-)+[\w-]{2,4}$');
            if (!regex.hasMatch(value)) {
                return ValidationError(
                    field: field,
                    message: message ?? '$field must be a valid email',
                    code: 'INVALID_EMAIL',
                );
            }
        }
    };
}
```

```

        return null;
    };
}

Middleware validateSchema(Map<String, List<ValidationRule>> schema) {
    return (Handler handler) {
        return (Request request) async {
            final body = request.json;
            final errors = <ValidationError>[];

            for (final entry in schema.entries) {
                final field = entry.key;
                final rules = entry.value;

                for (final rule in rules) {
                    final error = rule(field, body[field]);
                    if (error != null) {
                        errors.add(error);
                        break; // First error per field
                    }
                }
            }

            if (errors.isNotEmpty) {
                return validationError(ValidationResult.invalid(errors));
            }

            return handler(request);
        };
    };
}

// =====
// Main
// =====

void main() async {
    final router = Router();

    // User Registration
    router.post('/api/register', registerUser);

    // Company mit verschachtelter Validierung
    router.post('/api/companies', createCompany);

    // Bonus: Schema-basierte Validierung
    final productSchema = {
        'name': [requiredRule(), minLengthRule(2)],
        'email': [requiredRule(), emailRule()],
    };
};

```

```

router.post('/api/products', Pipeline()
  .addMiddleware(validateSchema(productSchema))
  .addHandler((request) {
    return jsonResponse({'message': 'Product created'}, statusCode: 201);
  }));

// Users auflisten
router.get('/api/users', (request) {
  return jsonResponse({
    'data': _users.values.toList(),
    'total': _users.length,
  });
});

final handler = Pipeline()
  .addMiddleware(logRequests())
  .addMiddleware(jsonBodyParser())
  .addHandler(router.call);

await shelf_io.serve(handler, 'localhost', 8080);
print('Server: http://localhost:8080');
print('');
print('Endpoints:');
print('  POST /api/register - User Registration');
print('  POST /api/companies - Company with nested validation');
print('  POST /api/products - Schema-based validation (bonus)');
print('  GET  /api/users - List users');
}

```

### 2.5.2.2 Test-Befehle

```

# ===== Erfolgreiche Registrierung =====
curl -X POST http://localhost:8080/api/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "max_mustermann",
    "email": "max@example.com",
    "password": "SecurePass1",
    "confirmPassword": "SecurePass1",
    "age": 25
  }'

# ===== Validierungsfehler =====

# Username zu kurz + ungültige E-Mail
curl -X POST http://localhost:8080/api/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "ab",
    "email": "invalid-email",

```

```
    "password": "SecurePass1",
    "confirmPassword": "SecurePass1"
  }'

# Schwaches Passwort (keine Großbuchstaben, keine Ziffern)
curl -X POST http://localhost:8080/api/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "max_m",
    "email": "max@test.de",
    "password": "weakpass",
    "confirmPassword": "weakpass"
  }'

# Passwörter stimmen nicht überein
curl -X POST http://localhost:8080/api/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "max_m",
    "email": "max@test.de",
    "password": "SecurePass1",
    "confirmPassword": "DifferentPass"
  }'

# Ungültiger Role-Wert
curl -X POST http://localhost:8080/api/register \
  -H "Content-Type: application/json" \
  -d '{
    "username": "max_m",
    "email": "max@test.de",
    "password": "SecurePass1",
    "confirmPassword": "SecurePass1",
    "role": "superadmin"
  }'

# ===== Verschachtelte Validierung =====

# Erfolgreiche Company
curl -X POST http://localhost:8080/api/companies \
  -H "Content-Type: application/json" \
  -d '{
    "company": {
      "name": "Acme Corp",
      "taxId": "DE123456789",
      "address": {
        "street": "Hauptstraße 1",
        "city": "Berlin",
        "zip": "10115",
        "country": "DE"
      }
    }
  },
```

```

    "contact": {
      "name": "Max Mustermann",
      "email": "max@acme.com"
    },
    "employees": [
      {"name": "Anna", "role": "developer"},
      {"name": "Bob", "role": "designer"}
    ]
  }'

```

#### # Fehler in verschachtelten Feldern

```

curl -X POST http://localhost:8080/api/companies \
-H "Content-Type: application/json" \
-d '{
  "company": {
    "name": "A",
    "taxId": "invalid",
    "address": {
      "city": "Berlin",
      "zip": "123",
      "country": "Germany"
    }
  },
  "contact": {
    "email": "invalid"
  },
  "employees": []
}'

```

#### # ===== Bonus: Schema-Validierung =====

```

curl -X POST http://localhost:8080/api/products \
-H "Content-Type: application/json" \
-d '{"name": "A", "email": "invalid"}'

```

### 2.5.2.3 Ausgabe-Beispiele

Erfolgreiche Registrierung (201)

```

{
  "id": "user-1",
  "username": "max_mustermann",
  "email": "max@example.com",
  "age": 25,
  "role": "user",
  "createdAt": "2024-01-15T10:00:00.000Z"
}

```

Validierungsfehler (400)

```

{
  "error": "Validation failed",
  "details": [

```

```
{
  "field": "username",
  "message": "username must be at least 3 characters",
  "code": "MIN_LENGTH"
},
{
  "field": "email",
  "message": "email must be a valid email",
  "code": "INVALID_EMAIL"
}
]
```

Verschachtelte Fehler (400)

```
{
  "error": "Validation failed",
  "details": [
    {
      "field": "company.name",
      "message": "company.name must be at least 2 characters",
      "code": "MIN_LENGTH"
    },
    {
      "field": "company.taxId",
      "message": "taxId must be 2 letters followed by 9 digits",
      "code": "INVALID_FORMAT"
    },
    {
      "field": "company.address.zip",
      "message": "zip must be 5 digits",
      "code": "INVALID_FORMAT"
    },
    {
      "field": "contact.name",
      "message": "contact.name is required",
      "code": "REQUIRED"
    },
    {
      "field": "employees",
      "message": "employees must have at least 1 items",
      "code": "MIN_ITEMS"
    }
  ]
}
```

#### 2.5.2.4 Wichtige Patterns

Fluent API für Validierung

```
final result = Validator(body)
    .required('name')
```

```
.minLength('name', 2)
.required('email')
.email('email')
.validate();
```

Prefix für verschachtelte Felder

```
Validator(data, prefix: 'company.address')
// Fehler-Feld wird: company.address.city
```

Passwort-Validierung mit mehreren Regeln

```
.password('password',
    minLength: 8,
    requireUppercase: true,
    requireDigit: true)
```

## 2.5.3 Ressourcen

### 2.5.3.1 Offizielle Dokumentation

- RFC 7807 Problem Details
- OWASP Input Validation

### 2.5.3.2 Cheat Sheet: Inline-Validierung

```
// Pflichtfeld
final name = body['name'] as String?;
if (name == null || name.isEmpty) {
    return badRequest('name is required');
}

// E-Mail Format
final emailRegex = RegExp(r'^[\w-\.]@([\w-]+\.)+[\w-]{2,4}$');
if (!emailRegex.hasMatch(email)) {
    return badRequest('invalid email format');
}

// String-Länge
if (password.length < 8) {
    return badRequest('password must be at least 8 characters');
}

// Zahlenbereich
if (age < 0 || age > 150) {
    return badRequest('age must be between 0 and 150');
}

// Erlaubte Werte
if (!['draft', 'published'].contains(status)) {
    return badRequest('status must be draft or published');
}
```

**2.5.3.3 Cheat Sheet: Validator-Klasse**

```

class Validator {
    final List<ValidationError> _errors = [];
    final Map<String, dynamic> data;

    Validator(this.data);

    Validator required(String field, {String? message}) {
        final value = data[field];
        if (value == null || (value is String && value.isEmpty)) {
            _errors.add(ValidationError(field: field, message: message ?? '$field
↪ is required', code: 'REQUIRED'));
        }
        return this;
    }

    Validator minLength(String field, int length, {String? message}) {
        final value = data[field];
        if (value is String && value.length < length) {
            _errors.add(ValidationError(field: field, message: message ?? '$field
↪ must be at least $length characters', code: 'MIN_LENGTH'));
        }
        return this;
    }

    Validator maxLength(String field, int length, {String? message}) {
        final value = data[field];
        if (value is String && value.length > length) {
            _errors.add(ValidationError(field: field, message: message ?? '$field
↪ must be at most $length characters', code: 'MAX_LENGTH'));
        }
        return this;
    }

    Validator email(String field, {String? message}) {
        final value = data[field];
        if (value is String && value.isNotEmpty) {
            final regex = RegExp(r'^[\w-\.]@([\w-]+\.)+[\w-]{2,4}$');
            if (!regex.hasMatch(value)) {
                _errors.add(ValidationError(field: field, message: message ?? '$field
↪ must be a valid email', code: 'INVALID_EMAIL'));
            }
        }
        return this;
    }

    Validator range(String field, {int? min, int? max}) {
        final value = data[field];
        if (value is num) {
            if (min != null && value < min) {

```

```

        _errors.add(ValidationError(field: field, message: '$field must be at ↵
↵ least $min', code: 'MIN_VALUE'));
    }
    if (max != null && value > max) {
        _errors.add(ValidationError(field: field, message: '$field must be at ↵
↵ most $max', code: 'MAX_VALUE'));
    }
}
return this;
}

Validator oneOf(String field, List<String> values) {
    final value = data[field];
    if (value is String && !values.contains(value)) {
        _errors.add(ValidationError(field: field, message: '$field must be one ↵
↵ of: ${values.join(', ')}', code: 'INVALID_VALUE'));
    }
    return this;
}

ValidationResult validate() => ValidationResult.fromErrors(_errors);
}

```

#### 2.5.3.4 Cheat Sheet: Verwendung

```

final result = Validator(body)
    .required('name')
    .minLength('name', 2)
    .maxLength('name', 100)
    .required('email')
    .email('email')
    .required('password')
    .minLength('password', 8)
    .range('age', min: 0, max: 150)
    .oneOf('role', ['user', 'admin'])
    .validate();

if (!result.isValid) {
    return Response(400,
        body: jsonEncode({
            'error': 'Validation failed',
            'details': result.errors.map((e) => e.toJson()).toList(),
        }),
        headers: {'content-type': 'application/json'},
    );
}

```

### 2.5.3.5 Cheat Sheet: Fehler-Response

```
Response validationError(ValidationResult result) {
    return Response(400,
        body: jsonEncode({
            'type': 'validation_error',
            'title': 'Validation Error',
            'status': 400,
            'errors': result.errors.map((e) => {
                return {
                    'field': e.field,
                    'message': e.message,
                    'code': e.code,
                };
            }).toList(),
        )),
        headers: {'content-type': 'application/json'},
    );
}
```

### 2.5.3.6 Regex-Patterns

```
// E-Mail
final email = RegExp(r'^[\w-\.]@([\w-]+\.)+[\w-]{2,4}$');

// URL
final url = RegExp(r'^https?:\/\/[\w-]+(\.[\w-]+)+[/#?]?.*$');

// Telefon (einfach)
final phone = RegExp(r'^\+?[\d\s-]{10,}$');

// UUID
final uuid =
    ↪️ RegExp(r'^[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}$');

// Slug (URL-freundlich)
final slug = RegExp(r'^[a-z0-9]+(?:-[a-z0-9]+)*$');

// Nur Buchstaben
final alpha = RegExp(r'^[a-zA-Z]+$');

// Alphanumerisch
final alphaNum = RegExp(r'^[a-zA-Z0-9]+$');

// Datum (YYYY-MM-DD)
final date = RegExp(r'^\d{4}-\d{2}-\d{2}$');
```

### 2.5.3.7 Validierungsregeln nach Feldtyp

String-Felder

Regel	Beschreibung
required	Darf nicht leer sein
minLength	Mindestlänge
maxLength	Maximallänge
pattern	Regex-Match
email	E-Mail-Format
url	URL-Format
oneOf	Erlaubte Werte

#### Numerische Felder

Regel	Beschreibung
required	Muss vorhanden sein
min	Mindestwert
max	Maximalwert
integer	Muss Ganzzahl sein
positive	Muss > 0 sein

#### Array-Felder

Regel	Beschreibung
required	Darf nicht leer sein
minItems	Mindestanzahl Elemente
maxItems	Maximalanzahl Elemente
unique	Keine Duplikate

#### 2.5.3.8 HTTP Statuscodes

Code	Verwendung
400	Bad Request - Syntaxfehler
422	Unprocessable Entity - Semantische Fehler

#### 2.5.3.9 Beispiel: Vollständige Validierung

```
class ProductValidator {
    static ValidationResult validateCreate(Map<String, dynamic> data) {
        return Validator(data)
            .required('name')
            .minLength('name', 2)
            .maxLength('name', 200)
            .required('price')
            .range('price', min: 0)
            .required('category')
            .oneOf('category', ['electronics', 'clothing', 'food'])
            .maxLength('description', 2000)
            .range('stock', min: 0)
    }
}
```

```

        .validate();
    }

    static ValidationResult validateUpdate(Map<String, dynamic> data) {
        final v = Validator(data);

        if (data.containsKey('name')) {
            v.minLength('name', 2).maxLength('name', 200);
        }
        if (data.containsKey('price')) {
            v.range('price', min: 0);
        }
        if (data.containsKey('category')) {
            v.oneOf('category', ['electronics', 'clothing', 'food']);
        }
        if (data.containsKey('description')) {
            v.maxLength('description', 2000);
        }
        if (data.containsKey('stock')) {
            v.range('stock', min: 0);
        }

        return v.validate();
    }
}

```

### 2.5.3.10 Test-Beispiele

#### # Fehlende Pflichtfelder

```

curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{}'

```

#### # Ungültige E-Mail

```

curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Max", "email": "invalid", "password": "12345678"}'

```

#### # Zu kurzes Passwort

```

curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Max", "email": "max@test.de", "password": "123"}'

```

#### # Ungültiger Enum-Wert

```

curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Max", "email": "max@test.de", "password": "12345678", "role": ↵
↵  "superadmin"}'

```

## 2.6 Einheit 6.6: Error Handling

### 2.6.0.1 Lernziele

Nach dieser Einheit kannst du: - HTTP-Statuscodes korrekt einsetzen - Fehler systematisch behandeln und loggen - Einheitliche Fehler-Responses erstellen - Globale Fehlerbehandlung mit Middleware implementieren

### 2.6.0.2 HTTP-Statuscodes Übersicht

2xx - Erfolg

Code	Name	Verwendung
200	OK	Standard-Erfolg (GET, PUT, PATCH)
201	Created	Ressource erstellt (POST)
204	No Content	Erfolg ohne Body (DELETE)

4xx - Client-Fehler

Code	Name	Verwendung
400	Bad Request	Ungültige Syntax, fehlende Felder
401	Unauthorized	Nicht authentifiziert
403	Forbidden	Authentifiziert, aber keine Berechtigung
404	Not Found	Ressource existiert nicht
405	Method Not Allowed	HTTP-Methode nicht unterstützt
409	Conflict	Konflikt (z.B. Duplikat)
415	Unsupported Media Type	Content-Type nicht unterstützt
422	Unprocessable Entity	Valide Syntax, semantischer Fehler
429	Too Many Requests	Rate Limit überschritten

5xx - Server-Fehler

Code	Name	Verwendung
500	Internal Server Error	Unerwarteter Fehler
502	Bad Gateway	Upstream-Server-Fehler
503	Service Unavailable	Überlastet/Wartung
504	Gateway Timeout	Upstream-Timeout

### 2.6.0.3 Fehler-Response Format

Einheitliches Format

```
class ApiError {
    final String code;
    final String message;
    final Map<String, dynamic>? details;
    final String? traceId;

    ApiError({
        required this.code,
```

```
        required this.message,
        this.details,
        this.traceId,
    });

    Map<String, dynamic> toJson() => {
        'error': {
            'code': code,
            'message': message,
            if (details != null) 'details': details,
            if (traceId != null) 'traceId': traceId,
        },
    };
}
```

### Beispiel-Responses

```
// 400 Bad Request
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Validation failed",
    "details": {
      "fields": [
        {"field": "email", "message": "Invalid email format"}
      ]
    }
  }
}

// 404 Not Found
{
  "error": {
    "code": "NOT_FOUND",
    "message": "User with ID '123' not found"
  }
}

// 500 Internal Server Error
{
  "error": {
    "code": "INTERNAL_ERROR",
    "message": "An unexpected error occurred",
    "traceId": "abc123-def456"
  }
}
```

#### 2.6.0.4 Exception-Klassen

##### Benutzerdefinierte Exceptions

```
abstract class ApiException implements Exception {
    int get statusCode;
    String get code;
    String get message;
    Map<String, dynamic>? get details => null;

    Response toResponse({String? traceId}) {
        return Response(
            statusCode,
            body: jsonEncode({
                'error': {
                    'code': code,
                    'message': message,
                    if (details != null) 'details': details,
                    if (traceId != null) 'traceId': traceId,
                },
            }),
            headers: {'content-type': 'application/json'},
        );
    }
}

class NotFoundException extends ApiException {
    final String resourceType;
    final String resourceId;

    NotFoundException(this.resourceType, this.resourceId);

    @override
    int get statusCode => 404;

    @override
    String get code => 'NOT_FOUND';

    @override
    String get message => '$resourceType with ID \'$resourceId\' not found';
}

class ValidationException extends ApiException {
    final List<ValidationError> errors;

    ValidationException(this.errors);

    @override
    int get statusCode => 400;

    @override
    String get code => 'VALIDATION_ERROR';

    @override
    String get message => 'Validation failed';
}
```

```
@override
Map<String, dynamic>? get details => {
  'fields': errors.map((e) => e.toJson()).toList(),
};
}

class UnauthorizedException extends ApiException {
  @override
  int get statusCode => 401;

  @override
  String get code => 'UNAUTHORIZED';

  @override
  String get message => 'Authentication required';
}

class ForbiddenException extends ApiException {
  @override
  int get statusCode => 403;

  @override
  String get code => 'FORBIDDEN';

  @override
  String get message => 'You do not have permission to access this resource';
}

class ConflictException extends ApiException {
  final String conflictMessage;

  ConflictException(this.conflictMessage);

  @override
  int get statusCode => 409;

  @override
  String get code => 'CONFLICT';

  @override
  String get message => conflictMessage;
}
```

#### Verwendung

```
Response getUser(Request request, String id) {
  final user = userRepo.findById(id);
  if (user == null) {
    throw NotFoundException('User', id);
  }
}
```

```
    return jsonResponse(user.toJson());
}

Response createUser(Request request) {
    final body = request.json;

    // Validierung
    final result = UserValidator.validate(body);
    if (!result.isValid) {
        throw ValidationException(result.errors);
    }

    // Duplikat-Check
    if (userRepo.existsByEmail(body['email'])) {
        throw ConflictException('A user with this email already exists');
    }

    // User erstellen...
}
```

### 2.6.0.5 Globale Fehlerbehandlung

#### Error Handler Middleware

```
Middleware errorHandler() {
    return (Handler innerHandler) {
        return (Request request) async {
            try {
                return await innerHandler(request);
            } on ApiException catch (e) {
                // Bekannte API-Fehler
                final traceId = _generateTraceId();
                _logError(request, e, traceId);
                return e.toResponse(traceId: traceId);
            } on FormatException catch (e) {
                // JSON-Parsing-Fehler
                return Response(400,
                    body: jsonEncode({
                        'error': {
                            'code': 'INVALID_JSON',
                            'message': 'Invalid JSON: ${e.message}',
                        },
                    }),
                    headers: {'content-type': 'application/json'},
                );
            } catch (e, stackTrace) {
                // Unerwartete Fehler
                final traceId = _generateTraceId();
                _logError(request, e, traceId, stackTrace: stackTrace);

                // In Produktion: Keine Details preisgeben
            }
        };
    };
}
```

```

        return Response(500,
            body: jsonEncode({
                'error': {
                    'code': 'INTERNAL_ERROR',
                    'message': 'An unexpected error occurred',
                    'traceId': traceId,
                },
            }),
            headers: {'content-type': 'application/json'},
        );
    }
};
};
}

String _generateTraceId() {
    final random = Random();
    return List.generate(16, (_) => random.nextInt(16).toRadixString(16)).join();
}

void _logError(Request request, Object error, String traceId, {StackTrace? ↵
    ↵ stackTrace}) {
    final timestamp = DateTime.now().toIso8601String();
    print('[${timestamp}] ERROR [${traceId}] ${request.method} ${request.url}');
    print('  Error: $error');
    if (stackTrace != null) {
        print('  Stack: $stackTrace');
    }
}
}

```

### Pipeline-Setup

```

void main() async {
    final router = Router();

    // Routes definieren...

    final handler = Pipeline()
        .addMiddleware(logRequests())
        .addMiddleware(errorHandler()) // Fehlerbehandlung NACH logging
        .addMiddleware(jsonBodyParser())
        .addHandler(router.call);

    await shelf_io.serve(handler, 'localhost', 8080);
}

```

#### 2.6.0.6 Fehler-Logging

##### Strukturiertes Logging

```
enum LogLevel { debug, info, warn, error }
```

```
class Logger {
    final String name;

    Logger(this.name);

    void debug(String message, [Map<String, dynamic>? context]) {
        _log(LogLevel.debug, message, context);
    }

    void info(String message, [Map<String, dynamic>? context]) {
        _log(LogLevel.info, message, context);
    }

    void warn(String message, [Map<String, dynamic>? context]) {
        _log(LogLevel.warn, message, context);
    }

    void error(String message, [Object? error, StackTrace? stackTrace]) {
        _log(LogLevel.error, message, {
            if (error != null) 'error': error.toString(),
            if (stackTrace != null) 'stack': stackTrace.toString(),
        });
    }

    void _log(LogLevel level, String message, Map<String, dynamic>? context) {
        final timestamp = DateTime.now().toIso8601String();
        final levelStr = level.name.toUpperCase().padRight(5);
        final contextStr = context != null ? ' $context' : '';
        print('[$timestamp] $levelStr [$name] $message$contextStr');
    }
}

// Verwendung
final logger = Logger('UserService');

void createUser(Map<String, dynamic> data) {
    logger.info('Creating user', {'email': data['email']});

    try {
        // ...
        logger.info('User created', {'userId': user.id});
    } catch (e, stack) {
        logger.error('Failed to create user', e, stack);
        rethrow;
    }
}
```

### 2.6.0.7 Entwicklung vs. Produktion

Unterschiedliche Fehler-Details

```

final isProduction = Platform.environment['ENV'] == 'production';

Middleware errorHandler() {
  return (Handler innerHandler) {
    return (Request request) async {
      try {
        return await innerHandler(request);
      } catch (e, stackTrace) {
        final traceId = _generateTraceId();

        if (isProduction) {
          // Produktion: Minimale Details
          return Response(500,
            body: jsonEncode({
              'error': {
                'code': 'INTERNAL_ERROR',
                'message': 'An unexpected error occurred',
                'traceId': traceId,
              },
            })),
            headers: {'content-type': 'application/json'},
          );
        } else {
          // Entwicklung: Volle Details
          return Response(500,
            body: jsonEncode({
              'error': {
                'code': 'INTERNAL_ERROR',
                'message': e.toString(),
                'traceId': traceId,
                'stack': stackTrace.toString().split('\n').take(10).toList(),
              },
            })),
            headers: {'content-type': 'application/json'},
          );
        }
      }
    };
  };
}

```

#### 2.6.0.8 404 Handler für unbekannte Routen

```

Handler notFoundHandler() {
  return (Request request) {
    return Response(404,
      body: jsonEncode({
        'error': {
          'code': 'NOT_FOUND',
          'message': 'Endpoint ${request.method} ${request.url.path} not found',

```

```

        },
    })),
    headers: {'content-type': 'application/json'},
);
};
}

// Verwendung
final handler = Pipeline()
    .addMiddleware(errorHandler())
    .addHandler(Cascade()
        .add(router.call)
        .add(notFoundHandler()) // Fallback für unbekannte Routen
        .handler);

```

### 2.6.0.9 Method Not Allowed (405)

```

// Shelf_router gibt automatisch 404 zurück
// Für 405 müssen wir selbst prüfen

Middleware methodNotAllowed(Map<String, List<String>> allowedMethods) {
    return (Handler handler) {
        return (Request request) {
            final path = request.url.path;
            final method = request.method;

            for (final entry in allowedMethods.entries) {
                // Einfacher Path-Match (für komplexere Patterns: Regex)
                if (path.startsWith(entry.key)) {
                    if (!entry.value.contains(method)) {
                        return Response(405,
                            headers: {
                                'allow': entry.value.join(', '),
                                'content-type': 'application/json',
                            },
                            body: jsonEncode({
                                'error': {
                                    'code': 'METHOD_NOT_ALLOWED',
                                    'message': 'Method $method not allowed for $path',
                                    'allowed': entry.value,
                                },
                            }
                        ));
                    }
                }
            }

            return handler(request);
        };
    };
}

```

```
}

```

### 2.6.0.10 Zusammenfassung

Situation	Statuscode	Error Code
Ressource nicht gefunden	404	NOT_FOUND
Validierungsfehler	400	VALIDATION_ERROR
Ungültiges JSON	400	INVALID_JSON
Nicht authentifiziert	401	UNAUTHORIZED
Keine Berechtigung	403	FORBIDDEN
Duplikat	409	CONFLICT
Server-Fehler	500	INTERNAL_ERROR

### 2.6.0.11 Nächste Schritte

In der nächsten Einheit lernst du **Pagination & Filtering**: Wie du große Datenmengen in Seiten aufteilst und filterbar machst.

## 2.6.1 Übung

### 2.6.1.1 Ziel

Implementiere ein robustes Error Handling System für eine REST-API.

### 2.6.1.2 Aufgabe 1: Exception-Klassen (15 min)

Erstelle eine Hierarchie von API-Exceptions.

Basis-Klasse

```
abstract class ApiException implements Exception {
    int get statusCode;
    String get code;
    String get message;
    Map<String, dynamic>? get details => null;

    Response toResponse({String? traceId});
}
```

Zu implementierende Exceptions

Exception	Status	Code	Beispiel
NotFoundException	404	NOT_FOUND	User nicht gefunden
BadRequestException	400	BAD_REQUEST	Ungültige Anfrage
ValidationException	400	VALIDATION_ERROR	Felder ungültig
UnauthorizedException	401	UNAUTHORIZED	Nicht eingeloggt
ForbiddenException	403	FORBIDDEN	Keine Berechtigung
ConflictException	409	CONFLICT	Email existiert

Beispiel-Verwendung

```

throw NotFoundException('User', '123');
// -> {"error": {"code": "NOT_FOUND", "message": "User with ID '123' not found"}}

throw ConflictException('A user with this email already exists');
// -> {"error": {"code": "CONFLICT", "message": "A user with this email
↪ already exists"}}

```

### 2.6.1.3 Aufgabe 2: Error Handler Middleware (15 min)

Erstelle eine Middleware, die alle Exceptions abfängt.

Anforderungen

1. `ApiException` -> entsprechender Statuscode
2. `FormatException` -> 400 Bad Request
3. Andere Exceptions -> 500 Internal Server Error
4. Trace-ID für jede Fehler-Response generieren
5. Alle Fehler loggen

Template

```

Middleware errorHandler({bool isDevelopment = false}) {
  return (Handler innerHandler) {
    return (Request request) async {
      try {
        return await innerHandler(request);
      } on ApiException catch (e) {
        // TODO: Bekannte API-Fehler behandeln
      } on FormatException catch (e) {
        // TODO: JSON-Parsing-Fehler behandeln
      } catch (e, stackTrace) {
        // TODO: Unbekannte Fehler behandeln
        // In Development: Stack Trace einschließen
        // In Production: Nur generische Meldung
      }
    };
  };
}

```

### 2.6.1.4 Aufgabe 3: Fehler-Response Helpers (10 min)

Erstelle Helper-Funktionen für häufige Fehler.

Zu implementieren

```

Response notFound(String resource, String id);
Response badRequest(String message);
Response unauthorized({String? message});
Response forbidden({String? message});
Response conflict(String message);
Response validationError(List<ValidationError> errors);
Response internalError({String? traceId});

```

Erwartete Responses

```
// notFound('User', '123')
{
  "error": {
    "code": "NOT_FOUND",
    "message": "User with ID '123' not found"
  }
}

// validationError([...])
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Validation failed",
    "details": {
      "fields": [
        {
          "field": "email",
          "message": "Invalid email format",
          "code": "INVALID_EMAIL"
        }
      ]
    }
  }
}
```

#### 2.6.1.5 Aufgabe 4: Vollständiger API-Server (20 min)

Baue eine API mit korrektem Error Handling.

Endpoints

Methode	Pfad	Beschreibung
GET	/api/users	Alle User listen
GET	/api/users/:id	User abrufen (404 wenn nicht gefunden)
POST	/api/users	User erstellen (400 bei Validierung, 409 bei Duplikat)
PUT	/api/users/:id	User aktualisieren (404 wenn nicht gefunden)
DELETE	/api/users/:id	User löschen (404 wenn nicht gefunden)

Validierungsregeln für POST/PUT

- **name:** Pflicht, 2-100 Zeichen
- **email:** Pflicht, gültiges Format, eindeutig
- **age:** Optional, 0-150

Test-Szenarien

```
# 200 OK
curl http://localhost:8080/api/users

# 201 Created
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
```

```

-d '{"name": "Max", "email": "max@test.de"}'

# 400 Bad Request (Validierung)
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "A"}'

# 404 Not Found
curl http://localhost:8080/api/users/not-exists

# 409 Conflict (Duplikat)
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Max2", "email": "max@test.de"}'

```

### 2.6.1.6 Aufgabe 5: 404 Fallback Handler (5 min)

Erstelle einen Handler für unbekannte Routen.

Anforderung

Alle Requests, die keiner Route entsprechen, sollen eine JSON-Fehler-Response bekommen.

```

{
  "error": {
    "code": "NOT_FOUND",
    "message": "Endpoint GET /api/unknown not found"
  }
}

```

Tipp: Cascade

```

final handler = Cascade()
  .add(router.call)
  .add(notFoundHandler())
  .handler;

```

### 2.6.1.7 Aufgabe 6: Request Logging mit Fehlern (Bonus, 10 min)

Erweitere das Logging um Fehlerinformationen.

Format

```

[2024-01-15T10:30:00Z] INFO GET /api/users 200 45ms
[2024-01-15T10:30:01Z] ERROR POST /api/users 400 12ms {"code": "VALIDATION_ERROR"}
[2024-01-15T10:30:02Z] ERROR GET /api/users/999 404 5ms {"code": "NOT_FOUND"}
[2024-01-15T10:30:03Z] ERROR POST /api/users 500 123ms {"code": "INTERNAL_ERROR", "traceId": "

```

Middleware

```

Middleware requestLogger() {
  return (Handler handler) {
    return (Request request) async {
      final stopwatch = Stopwatch()..start();

```

```
    final response = await handler(request);
    stopwatch.stop();

    final level = response.statusCode >= 400 ? 'ERROR' : 'INFO';
    // TODO: Logging implementieren

    return response;
  };
};
}
```

### 2.6.1.8 Testen

```
# Erfolgreiche Requests
curl http://localhost:8080/api/users
curl http://localhost:8080/api/users/user-1

# Fehler-Szenarien
curl http://localhost:8080/api/users/not-exists # 404
curl http://localhost:8080/api/unknown # 404 (Route nicht gefunden)
curl -X POST http://localhost:8080/api/users -d '{} ' # 400

# Ungültiges JSON
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{invalid}' # 400

# Mit Response-Headers
curl -i http://localhost:8080/api/users/not-exists
```

### 2.6.1.9 Abgabe-Checkliste

- ☐ Alle Exception-Klassen implementiert
- ☐ Error Handler Middleware funktioniert
- ☐ ApiException -> korrekter Statuscode
- ☐ FormatException -> 400
- ☐ Unbekannte Fehler -> 500 (ohne Details in Produktion)
- ☐ Trace-ID wird generiert
- ☐ Fehler werden geloggt
- ☐ 404 Handler für unbekannte Routen
- ☐ API-Endpoints mit korrektem Error Handling
- ☐ (Bonus) Request Logging mit Fehlerinfos

## 2.6.2 Lösung

### 2.6.2.1 Vollständige Lösung

```
import 'dart:convert';
import 'dart:math';
import 'package:shelf/shelf.dart';
```

```
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';

// =====
// Aufgabe 1: Exception-Klassen
// =====

abstract class ApiException implements Exception {
  int get statusCode;
  String get code;
  String get message;
  Map<String, dynamic>? get details => null;

  Response toResponse({String? traceId}) {
    return Response(
      statusCode,
      body: jsonEncode({
        'error': {
          'code': code,
          'message': message,
          if (details != null) 'details': details,
          if (traceId != null) 'traceId': traceId,
        },
      }),
      headers: {'content-type': 'application/json'},
    );
  }
}

class NotFoundException extends ApiException {
  final String resourceType;
  final String resourceId;

  NotFoundException(this.resourceType, this.resourceId);

  @override
  int get statusCode => 404;

  @override
  String get code => 'NOT_FOUND';

  @override
  String get message => '$resourceType with ID \'$resourceId\' not found';
}

class BadRequestException extends ApiException {
  @override
  final String message;

  BadRequestException(this.message);
}
```

```
@override
int get statusCode => 400;

@override
String get code => 'BAD_REQUEST';
}

class ValidationError {
  final String field;
  final String message;
  final String code;

  ValidationError({required this.field, required this.message, required
↵  this.code});

  Map<String, dynamic> toJson() => {'field': field, 'message': message,
↵  'code': code};
}

class ValidationException extends ApiException {
  final List<ValidationError> errors;

  ValidationException(this.errors);

  @override
  int get statusCode => 400;

  @override
  String get code => 'VALIDATION_ERROR';

  @override
  String get message => 'Validation failed';

  @override
  Map<String, dynamic>? get details => {
    'fields': errors.map((e) => e.toJson()).toList(),
  };
}

class UnauthorizedException extends ApiException {
  final String? customMessage;

  UnauthorizedException([this.customMessage]);

  @override
  int get statusCode => 401;

  @override
  String get code => 'UNAUTHORIZED';

  @override
```

```

    String get message => customMessage ?? 'Authentication required';
}

class ForbiddenException extends ApiException {
    final String? customMessage;

    ForbiddenException([this.customMessage]);

    @override
    int get statusCode => 403;

    @override
    String get code => 'FORBIDDEN';

    @override
    String get message => customMessage ?? 'You do not have permission to ↵
↵  access this resource';
}

class ConflictException extends ApiException {
    @override
    final String message;

    ConflictException(this.message);

    @override
    int get statusCode => 409;

    @override
    String get code => 'CONFLICT';
}

// =====
// Aufgabe 2: Error Handler Middleware
// =====

String _generateTraceId() {
    final random = Random();
    return List.generate(16, (_) => random.nextInt(16).toRadixString(16)).join();
}

void _logError(String traceId, Request request, Object error, {StackTrace? ↵
↵  stackTrace}) {
    final timestamp = DateTime.now().toIso8601String();
    print('[${timestamp}] ERROR [${traceId}] ${request.method} ${request.url}');
    print(' Error: $error');
    if (stackTrace != null) {
        // Nur erste 5 Zeilen des Stack Traces
        final lines = stackTrace.toString().split('\n').take(5);
        for (final line in lines) {
            print(' $line');
        }
    }
}

```

```

    }
  }
}

Middleware errorHandler({bool isDevelopment = false}) {
  return (Handler innerHandler) {
    return (Request request) async {
      try {
        return await innerHandler(request);
      } on ApiException catch (e) {
        // Bekannte API-Fehler
        final traceId = _generateTraceId();
        if (e.statusCode >= 500) {
          _logError(traceId, request, e);
        }
        return e.toResponse(traceId: traceId);
      } on FormatException catch (e) {
        // JSON-Parsing-Fehler
        final traceId = _generateTraceId();
        return Response(400,
          body: jsonEncode({
            'error': {
              'code': 'INVALID_JSON',
              'message': 'Invalid JSON: ${e.message}',
              'traceId': traceId,
            },
          })),
          headers: {'content-type': 'application/json'},
        );
      } catch (e, stackTrace) {
        // Unbekannte Fehler
        final traceId = _generateTraceId();
        _logError(traceId, request, e, stackTrace: stackTrace);

        if (isDevelopment) {
          // Entwicklung: Volle Details
          return Response(500,
            body: jsonEncode({
              'error': {
                'code': 'INTERNAL_ERROR',
                'message': e.toString(),
                'traceId': traceId,
                'stack': stackTrace.toString().split('\n').take(10).toList(),
              },
            })),
            headers: {'content-type': 'application/json'},
          );
        } else {
          // Produktion: Minimale Details
          return Response(500,
            body: jsonEncode({

```

```

        'error': {
            'code': 'INTERNAL_ERROR',
            'message': 'An unexpected error occurred',
            'traceId': traceId,
        },
    )),
    headers: {'content-type': 'application/json'},
);
    }
}
};
};
}

// =====
// Aufgabe 3: Helper-Funktionen
// =====

Response notFound(String resource, String id) {
    return Response(404,
        body: jsonEncode({
            'error': {
                'code': 'NOT_FOUND',
                'message': '$resource with ID \'$id\' not found',
            },
        )),
        headers: {'content-type': 'application/json'},
    );
}

Response badRequest(String message) {
    return Response(400,
        body: jsonEncode({
            'error': {
                'code': 'BAD_REQUEST',
                'message': message,
            },
        )),
        headers: {'content-type': 'application/json'},
    );
}

Response unauthorized({String? message}) {
    return Response(401,
        body: jsonEncode({
            'error': {
                'code': 'UNAUTHORIZED',
                'message': message ?? 'Authentication required',
            },
        )),
        headers: {'content-type': 'application/json'},
    );
}

```

```
);
}

Response forbidden({String? message}) {
  return Response(403,
    body: jsonEncode({
      'error': {
        'code': 'FORBIDDEN',
        'message': message ?? 'Access denied',
      },
    }),
    headers: {'content-type': 'application/json'},
  );
}

Response conflict(String message) {
  return Response(409,
    body: jsonEncode({
      'error': {
        'code': 'CONFLICT',
        'message': message,
      },
    }),
    headers: {'content-type': 'application/json'},
  );
}

Response validationError(List<ValidationError> errors) {
  return Response(400,
    body: jsonEncode({
      'error': {
        'code': 'VALIDATION_ERROR',
        'message': 'Validation failed',
        'details': {
          'fields': errors.map((e) => e.toJson()).toList(),
        },
      },
    }),
    headers: {'content-type': 'application/json'},
  );
}

Response internalError({String? traceId}) {
  return Response(500,
    body: jsonEncode({
      'error': {
        'code': 'INTERNAL_ERROR',
        'message': 'An unexpected error occurred',
        if (traceId != null) 'traceId': traceId,
      },
    }),
  );
}
```

```

        headers: {'content-type': 'application/json'},
    );
}

Response jsonResponse(Object? data, {int statusCode = 200}) {
    return Response(
        statusCode,
        body: jsonEncode(data),
        headers: {'content-type': 'application/json'},
    );
}

// =====
// Aufgabe 5: 404 Fallback Handler
// =====

Handler notFoundHandler() {
    return (Request request) {
        return Response(404,
            body: jsonEncode({
                'error': {
                    'code': 'NOT_FOUND',
                    'message': 'Endpoint ${request.method} ${request.url.path} not found',
                },
            }),
            headers: {'content-type': 'application/json'},
        );
    };
}

// =====
// Aufgabe 6: Request Logging (Bonus)
// =====

Middleware requestLogger() {
    return (Handler handler) {
        return (Request request) async {
            final stopwatch = Stopwatch()..start();
            final response = await handler(request);
            stopwatch.stop();

            final timestamp = DateTime.now().toIso8601String();
            final level = response.statusCode >= 400 ? 'ERROR' : 'INFO';
            final duration = '${stopwatch.elapsedMilliseconds}ms';

            var logLine = '[${timestamp}] $level ${request.method}
↳ ${request.url.path} ${response.statusCode} $duration';
↳

            // Bei Fehlern: Error-Details extrahieren (wenn möglich)
            if (response.statusCode >= 400) {
                // Response Body kann nur einmal gelesen werden, daher clonen

```

```

        final body = await response.readAsString();
        try {
            final json = jsonDecode(body) as Map<String, dynamic>;
            final error = json['error'] as Map<String, dynamic>?;
            if (error != null) {
                logLine += ' {"code": "${error['code']}"}';
                if (error['traceId'] != null) {
                    logLine += ', "traceId": "${error['traceId']}"}';
                }
                logLine += '}';
            }
        } catch (_) {}
        // Response neu erstellen
        return response.change(body: body);
    }

    print(logLine);
    return response;
};
};
}

// =====
// Aufgabe 4: API Server
// =====

extension RequestJson on Request {
    Map<String, dynamic> get json {
        final body = context['body'];
        return body is Map<String, dynamic> ? body : {};
    }
}

Middleware jsonBodyParser() {
    return (Handler handler) {
        return (Request request) async {
            if (!['POST', 'PUT', 'PATCH'].contains(request.method)) {
                return handler(request);
            }
            final contentType = request.headers['content-type'] ?? '';
            if (!contentType.contains('application/json')) {
                return handler(request);
            }
            final body = await request.readAsString();
            if (body.isEmpty) {
                return handler(request.change(context: {...request.context, 'body':
↵ <String, dynamic>{}}));
            }
            final json = jsonDecode(body); // FormatException wird von errorHandler ↵
↵ gefangen
            return handler(request.change(context: {...request.context, 'body':
↵ json}));

```

```

    };
  };
}

// User Storage
final _users = <String, Map<String, dynamic>>{};
var _nextUserId = 1;

void _seedUsers() {
  _users['user-1'] = {
    'id': 'user-1',
    'name': 'Max Mustermann',
    'email': 'max@example.com',
    'age': 30,
    'createdAt': DateTime.now().toIso8601String(),
  };
  _nextUserId = 2;
}

// Validation
List<ValidationError> validateUser(Map<String, dynamic> data, {bool isUpdate ↵
  ↵ = false, String? excludeEmail}) {
  final errors = <ValidationError>[];

  // Name
  final name = data['name'] as String?;
  if (!isUpdate || data.containsKey('name')) {
    if (name == null || name.isEmpty) {
      errors.add(ValidationError(field: 'name', message: 'name is required', ↵
  ↵ code: 'REQUIRED'));
    } else if (name.length < 2) {
      errors.add(ValidationError(field: 'name', message: 'name must be at ↵
  ↵ least 2 characters', code: 'MIN_LENGTH'));
    } else if (name.length > 100) {
      errors.add(ValidationError(field: 'name', message: 'name must be at ↵
  ↵ most 100 characters', code: 'MAX_LENGTH'));
    }
  }

  // Email
  final email = data['email'] as String?;
  if (!isUpdate || data.containsKey('email')) {
    if (email == null || email.isEmpty) {
      errors.add(ValidationError(field: 'email', message: 'email is ↵
  ↵ required', code: 'REQUIRED'));
    } else {
      final emailRegex = RegExp(r'^[\w-\.]@([\w-]+\.)+[\w-]{2,4}$');
      if (!emailRegex.hasMatch(email)) {
        errors.add(ValidationError(field: 'email', message: 'email must be a ↵
  ↵ valid email', code: 'INVALID_EMAIL'));
      }
    }
  }
}

```

```
    }
  }

  // Age (optional)
  final age = data['age'] as int?;
  if (age != null && (age < 0 || age > 150)) {
    errors.add(ValidationError(field: 'age', message: 'age must be between 0
↵ and 150', code: 'RANGE'));
  }

  return errors;
}

bool emailExists(String email, {String? excludeId}) {
  return _users.values.any((u) => u['email'] == email && u['id'] != excludeId);
}

// Handlers
Response listUsers(Request request) {
  return jsonResponse({
    'data': _users.values.toList(),
    'total': _users.length,
  });
}

Response getUser(Request request, String id) {
  final user = _users[id];
  if (user == null) {
    throw NotFoundException('User', id);
  }
  return jsonResponse(user);
}

Response createUser(Request request) {
  final body = request.json;

  // Validierung
  final errors = validateUser(body);
  if (errors.isNotEmpty) {
    throw ValidationException(errors);
  }

  // Duplikat-Check
  final email = body['email'] as String;
  if (emailExists(email)) {
    throw ConflictException('A user with email \'$email\' already exists');
  }

  // User erstellen
  final id = 'user-${_nextUserId++}';
  final user = {
```

```

        'id': id,
        'name': body['name'],
        'email': email,
        if (body['age'] != null) 'age': body['age'],
        'createdAt': DateTime.now().toIso8601String(),
    };
    _users[id] = user;

    return Response(201,
        body: jsonEncode(user),
        headers: {
            'content-type': 'application/json',
            'location': '/api/users/$id',
        },
    );
}

Response updateUser(Request request, String id) {
    final existing = _users[id];
    if (existing == null) {
        throw NotFoundException('User', id);
    }

    final body = request.json;

    // Validierung
    final errors = validateUser(body, isUpdate: true);
    if (errors.isNotEmpty) {
        throw ValidationException(errors);
    }

    // Duplikat-Check (wenn Email geändert wird)
    final email = body['email'] as String?;
    if (email != null && emailExists(email, excludeId: id)) {
        throw ConflictException('A user with email \'$email\' already exists');
    }

    // Update
    final updated = {
        ...existing,
        if (body['name'] != null) 'name': body['name'],
        if (body['email'] != null) 'email': body['email'],
        if (body.containsKey('age')) 'age': body['age'],
        'updatedAt': DateTime.now().toIso8601String(),
    };
    _users[id] = updated;

    return jsonResponse(updated);
}

Response deleteUser(Request request, String id) {

```

```
    if (_users.remove(id) == null) {
        throw NotFoundException('User', id);
    }
    return Response(204);
}

// Simulierter Fehler für Tests
Response simulateError(Request request) {
    throw Exception('This is a simulated internal error');
}

// =====
// Main
// =====

void main() async {
    _seedUsers();

    final router = Router();

    // User CRUD
    router.get('/api/users', listUsers);
    router.get('/api/users/<id>', getUser);
    router.post('/api/users', createUser);
    router.put('/api/users/<id>', updateUser);
    router.delete('/api/users/<id>', deleteUser);

    // Test-Endpoint für 500 Error
    router.get('/api/error', simulateError);

    // Pipeline mit Error Handler
    final handler = Pipeline()
        .addMiddleware(requestLogger())
        .addMiddleware(errorHandler(isDevelopment: true))
        .addMiddleware(jsonBodyParser())
        .addHandler(Cascade()
            .add(router.call)
            .add(notFoundHandler())
            .handler);

    await shelf_io.serve(handler, 'localhost', 8080);
    print('Server: http://localhost:8080');
    print('');
    print('Test-Befehle:');
    print('  curl http://localhost:8080/api/users');
    print('  curl http://localhost:8080/api/users/user-1');
    print('  curl http://localhost:8080/api/users/not-exists # 404');
    print('  curl http://localhost:8080/api/unknown # 404 route');
    print('  curl http://localhost:8080/api/error # 500');
}
```

### 2.6.2.2 Test-Befehle

```
# ===== Erfolgreiche Requests =====

# Liste alle User
curl http://localhost:8080/api/users

# Einen User abrufen
curl http://localhost:8080/api/users/user-1

# User erstellen
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Anna Schmidt", "email": "anna@test.de", "age": 25}'

# User aktualisieren
curl -X PUT http://localhost:8080/api/users/user-1 \
  -H "Content-Type: application/json" \
  -d '{"name": "Max Updated"}'

# User löschen
curl -X DELETE http://localhost:8080/api/users/user-2

# ===== Fehler-Szenarien =====

# 404 - User nicht gefunden
curl http://localhost:8080/api/users/not-exists

# 404 - Route nicht gefunden
curl http://localhost:8080/api/unknown

# 400 - Validierungsfehler
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "A"}'

# 400 - Ungültige Email
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Test", "email": "invalid"}'

# 400 - Ungültiges JSON
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{invalid json}'

# 409 - Duplikat
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{"name": "Max Copy", "email": "max@example.com"}'
```

```
# 500 - Interner Fehler
curl http://localhost:8080/api/error

# Mit Response-Headers anzeigen
curl -i http://localhost:8080/api/users/not-exists
```

### 2.6.2.3 Ausgabe-Beispiele

#### 404 Not Found

```
{
  "error": {
    "code": "NOT_FOUND",
    "message": "User with ID 'not-exists' not found",
    "traceId": "a1b2c3d4e5f6g7h8"
  }
}
```

#### 400 Validation Error

```
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Validation failed",
    "details": {
      "fields": [
        {
          "field": "name",
          "message": "name must be at least 2 characters",
          "code": "MIN_LENGTH"
        },
        {
          "field": "email",
          "message": "email is required",
          "code": "REQUIRED"
        }
      ]
    },
    "traceId": "b2c3d4e5f6g7h8i9"
  }
}
```

#### 409 Conflict

```
{
  "error": {
    "code": "CONFLICT",
    "message": "A user with email 'max@example.com' already exists",
    "traceId": "c3d4e5f6g7h8i9j0"
  }
}
```

## 500 Internal Error (Development)

```
{
  "error": {
    "code": "INTERNAL_ERROR",
    "message": "Exception: This is a simulated internal error",
    "traceId": "d4e5f6g7h8i9j0k1",
    "stack": [
      "#0      simulateError (file:///app/main.dart:123:3)",
      "#1      Router.call (package:shelf_router/router.dart:45:12)",
      "... "
    ]
  }
}
```

## 500 Internal Error (Production)

```
{
  "error": {
    "code": "INTERNAL_ERROR",
    "message": "An unexpected error occurred",
    "traceId": "d4e5f6g7h8i9j0k1"
  }
}
```

## 2.6.2.4 Log-Ausgabe

```
[2024-01-15T10:30:00.000Z] INFO GET /api/users 200 5ms
[2024-01-15T10:30:01.000Z] INFO GET /api/users/user-1 200 2ms
[2024-01-15T10:30:02.000Z] ERROR GET /api/users/not-exists 404 3ms {"code": "NOT_FOUND"}
[2024-01-15T10:30:03.000Z] ERROR POST /api/users 400 8ms {"code": "VALIDATION_ERROR"}
[2024-01-15T10:30:04.000Z] ERROR GET /api/error 500 1ms {"code": "INTERNAL_ERROR", "traceId":
```

## 2.6.2.5 Wichtige Patterns

## Exception-Hierarchie

```
abstract class ApiException {
  int get statusCode;
  String get code;
  String get message;
  Response toResponse();
}

class NotFoundException extends ApiException { ... }
class ValidationException extends ApiException { ... }
```

## Cascade für 404 Fallback

```
final handler = Cascade()
  .add(router.call)           // Erst Router versuchen
  .add(notFoundHandler())    // Dann 404 Handler
  .handler;
```

## Error Handler Middleware

```
Middleware errorHandler() {
  return (handler) => (request) async {
    try {
      return await handler(request);
    } on ApiException catch (e) {
      return e.toResponse();
    } catch (e) {
      return internalError();
    }
  };
}
```

### 2.6.3 Ressourcen

#### 2.6.3.1 Offizielle Dokumentation

- [HTTP Status Codes \(MDN\)](#)
- [RFC 7807 Problem Details](#)
- [REST API Error Handling](#)

#### 2.6.3.2 Cheat Sheet: HTTP-Statuscodes

##### Erfolg (2xx)

```
// 200 OK - Standard-Erfolg
return Response.ok(jsonEncode(data));

// 201 Created - Ressource erstellt
return Response(201,
  body: jsonEncode(data),
  headers: {'location': '/api/items/$id'},
);

// 204 No Content - Erfolg ohne Body
return Response(204);
```

##### Client-Fehler (4xx)

```
// 400 Bad Request
return Response(400, body: '{"error": "Invalid request"}');

// 401 Unauthorized
return Response(401, body: '{"error": "Authentication required"}');

// 403 Forbidden
return Response(403, body: '{"error": "Access denied"}');

// 404 Not Found
return Response(404, body: '{"error": "Resource not found"}');

// 409 Conflict
```

```
return Response(409, body: '{"error": "Resource already exists"}');

// 422 Unprocessable Entity
return Response(422, body: '{"error": "Validation failed"}');

// 429 Too Many Requests
return Response(429,
  headers: {'retry-after': '60'},
  body: '{"error": "Rate limit exceeded"}',
);
```

Server-Fehler (5xx)

```
// 500 Internal Server Error
return Response(500, body: '{"error": "Internal server error"}');

// 503 Service Unavailable
return Response(503,
  headers: {'retry-after': '300'},
  body: '{"error": "Service temporarily unavailable"}',
);
```

### 2.6.3.3 Cheat Sheet: ApiException Klassen

```
abstract class ApiException implements Exception {
  int get statusCode;
  String get code;
  String get message;

  Response toResponse() => Response(statusCode,
    body: jsonEncode({'error': {'code': code, 'message': message}}),
    headers: {'content-type': 'application/json'},
  );
}

class NotFoundException extends ApiException {
  final String resource;
  NotFoundException(this.resource);

  @override int get statusCode => 404;
  @override String get code => 'NOT_FOUND';
  @override String get message => '$resource not found';
}

class BadRequestException extends ApiException {
  @override final String message;
  BadRequestException(this.message);

  @override int get statusCode => 400;
  @override String get code => 'BAD_REQUEST';
}
```

```

class UnauthorizedException extends ApiException {
    @override int get statusCode => 401;
    @override String get code => 'UNAUTHORIZED';
    @override String get message => 'Authentication required';
}

class ForbiddenException extends ApiException {
    @override int get statusCode => 403;
    @override String get code => 'FORBIDDEN';
    @override String get message => 'Access denied';
}

class ConflictException extends ApiException {
    @override final String message;
    ConflictException(this.message);

    @override int get statusCode => 409;
    @override String get code => 'CONFLICT';
}

```

#### 2.6.3.4 Cheat Sheet: Error Handler Middleware

```

Middleware errorHandler() {
    return (Handler handler) {
        return (Request request) async {
            try {
                return await handler(request);
            } on ApiException catch (e) {
                return e.toResponse();
            } on FormatException {
                return Response(400,
                    body: '{"error": {"code": "INVALID_JSON", "message": "Invalid JSON"}}',
                    headers: {'content-type': 'application/json'},
                );
            } catch (e, stack) {
                print('ERROR: $e\n$stack');
                return Response(500,
                    body: '{"error": {"code": "INTERNAL_ERROR", "message": "An
↪ unexpected error occurred"}}',
                    headers: {'content-type': 'application/json'},
                );
            }
        };
    };
}

```

### 2.6.3.5 Cheat Sheet: Fehler-Response Format

```
// Standard-Format
Response ErrorResponse(int status, String code, String message, {Map<String, ↵
↳ dynamic>? details}) {
    return Response(status,
        body: jsonEncode({
            'error': {
                'code': code,
                'message': message,
                if (details != null) 'details': details,
            },
        }),
        headers: {'content-type': 'application/json'},
    );
}

// RFC 7807 Problem Details
Response problemDetails(int status, String type, String title, String detail) {
    return Response(status,
        body: jsonEncode({
            'type': type,
            'title': title,
            'status': status,
            'detail': detail,
        }),
        headers: {'content-type': 'application/problem+json'},
    );
}
```

### 2.6.3.6 Cheat Sheet: Logger

```
class Logger {
    static void info(String message) {
        print('[${DateTime.now()}] INFO: $message');
    }

    static void error(String message, [Object? error, StackTrace? stack]) {
        print('[${DateTime.now()}] ERROR: $message');
        if (error != null) print('  Error: $error');
        if (stack != null) print('  Stack: $stack');
    }

    static void warn(String message) {
        print('[${DateTime.now()}] WARN: $message');
    }
}
```

### 2.6.3.7 HTTP Status Quick Reference

Code	Name	Wann verwenden
200	OK	GET erfolgreich, PUT/PATCH erfolgreich
201	Created	POST erfolgreich (neue Ressource)
204	No Content	DELETE erfolgreich
400	Bad Request	Ungültige Eingabe, fehlendes Feld
401	Unauthorized	Kein/ungültiger Token
403	Forbidden	Token gültig, aber keine Berechtigung
404	Not Found	Ressource existiert nicht
405	Method Not Allowed	z.B. DELETE auf read-only
409	Conflict	Duplikat, Race Condition
415	Unsupported Media Type	Falscher Content-Type
422	Unprocessable Entity	Semantischer Fehler
429	Too Many Requests	Rate Limit
500	Internal Server Error	Unerwarteter Fehler
503	Service Unavailable	Wartung, Überlastung

### 2.6.3.8 Fehlercode-Konventionen

```
// Konstanten für Error Codes
class ErrorCodes {
    static const notFound = 'NOT_FOUND';
    static const badRequest = 'BAD_REQUEST';
    static const validationError = 'VALIDATION_ERROR';
    static const unauthorized = 'UNAUTHORIZED';
    static const forbidden = 'FORBIDDEN';
    static const conflict = 'CONFLICT';
    static const internalError = 'INTERNAL_ERROR';
    static const rateLimited = 'RATE_LIMITED';
}
```

### 2.6.3.9 Best Practices

- Konsistentes Format**
  - Immer JSON zurückgeben
  - Einheitliche Fehlerstruktur
- Aussagekräftige Meldungen**
  - Für Menschen lesbar
  - Konkrete Hinweise zur Behebung
- Sicherheit**
  - Keine Stack Traces in Produktion
  - Keine internen Details preisgeben
- Logging**
  - Alle Fehler loggen
  - Trace-ID für Debugging
- Richtige Statuscodes**
  - 4xx für Client-Fehler
  - 5xx für Server-Fehler

### 2.6.3.10 Test-Befehle

```
# 404 testen
curl http://localhost:8080/api/users/not-exists

# 400 testen (ungültiges JSON)
curl -X POST http://localhost:8080/api/users \
  -H "Content-Type: application/json" \
  -d '{invalid}'

# 401 testen
curl http://localhost:8080/api/protected

# Response-Code anzeigen
curl -w "\nStatus: %{http_code}\n" http://localhost:8080/api/users/123
```

## 2.7 Einheit 6.7: Pagination & Filtering

### 2.7.0.1 Lernziele

Nach dieser Einheit kannst du: - Große Datenmengen in Seiten aufteilen (Pagination) - Daten nach Kriterien filtern - Sortierung implementieren - Suchfunktionen bereitstellen

### 2.7.0.2 Warum Pagination?

Problem ohne Pagination

```
// Alle 1.000.000 Produkte laden
GET /api/products

// Probleme:
// - Sehr lange Ladezeit
// - Hoher Speicherverbrauch (Server & Client)
// - Netzwerk-Überlastung
// - Schlechte User Experience
```

Lösung mit Pagination

```
// Nur 20 Produkte pro Seite
GET /api/products?page=1&perPage=20

// Vorteile:
// - Schnelle Antwortzeiten
// - Geringer Speicherverbrauch
// - Bessere UX
```

### 2.7.0.3 Offset-basierte Pagination

Implementierung

```
Response listProducts(Request request) {
  // Parameter aus Query String
```

```
final params = request.url.queryParameters;
final page = int.tryParse(params['page'] ?? '1') ?? 1;
final perPage = int.tryParse(params['perPage'] ?? '20') ?? 20;

// Limits setzen
final safePerPage = perPage.clamp(1, 100); // Max 100 pro Seite
final safePage = page.clamp(1, 1000);      // Max Seite 1000

// Daten abrufen
final allProducts = productRepo.findAll();
final total = allProducts.length;

// Offset berechnen
final offset = (safePage - 1) * safePerPage;
final products = allProducts.skip(offset).take(safePerPage).toList();

// Pagination-Metadaten
final totalPages = (total / safePerPage).ceil();

return jsonResponse({
  'data': products.map((p) => p.toJson()).toList(),
  'meta': {
    'total': total,
    'page': safePage,
    'perPage': safePerPage,
    'totalPages': totalPages,
    'hasNextPage': safePage < totalPages,
    'hasPrevPage': safePage > 1,
  },
});
}
```

Response-Format

```
{
  "data": [
    {"id": "1", "name": "Product 1", ...},
    {"id": "2", "name": "Product 2", ...}
  ],
  "meta": {
    "total": 150,
    "page": 1,
    "perPage": 20,
    "totalPages": 8,
    "hasNextPage": true,
    "hasPrevPage": false
  }
}
```

### 2.7.0.4 HATEOAS Links

Navigation-Links

```
Response listProducts(Request request) {
    // ... Pagination wie oben ...

    final baseUrl = '/api/products';

    return jsonResponse({
        'data': products.map((p) => p.toJson()).toList(),
        'meta': {
            'total': total,
            'page': safePage,
            'perPage': safePerPage,
            'totalPages': totalPages,
        },
        'links': {
            'self': '$baseUrl?page=$safePage&perPage=$safePerPage',
            'first': '$baseUrl?page=1&perPage=$safePerPage',
            'last': '$baseUrl?page=$totalPages&perPage=$safePerPage',
            if (safePage > 1)
                'prev': '$baseUrl?page=${safePage - 1}&perPage=$safePerPage',
            if (safePage < totalPages)
                'next': '$baseUrl?page=${safePage + 1}&perPage=$safePerPage',
        },
    });
}
```

Response mit Links

```
{
  "data": [...],
  "meta": {...},
  "links": {
    "self": "/api/products?page=2&perPage=20",
    "first": "/api/products?page=1&perPage=20",
    "last": "/api/products?page=8&perPage=20",
    "prev": "/api/products?page=1&perPage=20",
    "next": "/api/products?page=3&perPage=20"
  }
}
```

### 2.7.0.5 Cursor-basierte Pagination

Besser für Echtzeit-Daten und sehr große Datensätze.

Implementierung

```
Response listPosts(Request request) {
    final params = request.url.queryParameters;
    final limit = int.tryParse(params['limit'] ?? '20') ?? 20;
    final cursor = params['cursor']; // ID des letzten Elements
```

```
// Limit begrenzen
final safeLimit = limit.clamp(1, 100);

List<Post> posts;

if (cursor != null) {
    // Nach dem Cursor-Element holen
    final cursorIndex = postRepo.findIndexById(cursor);
    posts = postRepo.findAll()
        .skip(cursorIndex + 1)
        .take(safeLimit)
        .toList();
} else {
    // Von Anfang
    posts = postRepo.findAll().take(safeLimit).toList();
}

// Nächster Cursor
final nextCursor = posts.isNotEmpty ? posts.last.id : null;
final hasMore = posts.length == safeLimit;

return jsonResponse({
    'data': posts.map((p) => p.toJson()).toList(),
    'meta': {
        'limit': safeLimit,
        'hasMore': hasMore,
    },
    'cursors': {
        if (cursor != null) 'before': cursor,
        if (nextCursor != null && hasMore) 'after': nextCursor,
    },
});
}
```

Verwendung

```
# Erste Seite
GET /api/posts?limit=20

# Nächste Seite mit Cursor
GET /api/posts?limit=20&cursor=post-20
```

### 2.7.0.6 Filtering

Einfache Filter

```
Response listProducts(Request request) {
    final params = request.url.queryParameters;

    var products = productRepo.findAll();

    // Filter: ?category=electronics
```

```

final category = params['category'];
if (category != null) {
    products = products.where((p) => p.category == category).toList();
}

// Filter: ?inStock=true
final inStock = params['inStock'];
if (inStock == 'true') {
    products = products.where((p) => p.stock > 0).toList();
}

// Filter: ?minPrice=10&maxPrice=100
final minPrice = double.tryParse(params['minPrice'] ?? '');
final maxPrice = double.tryParse(params['maxPrice'] ?? '');

if (minPrice != null) {
    products = products.where((p) => p.price >= minPrice).toList();
}
if (maxPrice != null) {
    products = products.where((p) => p.price <= maxPrice).toList();
}

// ... Pagination ...
}

```

Request

```
GET /api/products?category=electronics&minPrice=50&maxPrice=200&inStock=true
```

### 2.7.0.7 Sortierung

Implementierung

```

Response listProducts(Request request) {
    final params = request.url.queryParameters;

    var products = productRepo.findAll();

    // Sortierung: ?sort=price&order=asc
    final sortField = params['sort'];
    final sortOrder = params['order'] ?? 'asc';

    if (sortField != null) {
        products = _sortProducts(products, sortField, sortOrder);
    }

    // ... Pagination ...
}

List<Product> _sortProducts(List<Product> products, String field, String order) {
    final sorted = [...products];
}

```

```
sorted.sort((a, b) {
    int comparison;

    switch (field) {
        case 'name':
            comparison = a.name.compareTo(b.name);
            break;
        case 'price':
            comparison = a.price.compareTo(b.price);
            break;
        case 'createdAt':
            comparison = a.createdAt.compareTo(b.createdAt);
            break;
        case 'stock':
            comparison = a.stock.compareTo(b.stock);
            break;
        default:
            comparison = 0;
    }

    return order == 'desc' ? -comparison : comparison;
});

return sorted;
}
```

Request

```
# Nach Preis aufsteigend
GET /api/products?sort=price&order=asc

# Nach Name absteigend
GET /api/products?sort=name&order=desc

# Mehrere Sortierungen (optional)
GET /api/products?sort=category,price&order=asc,desc
```

### 2.7.0.8 Suche

Einfache Textsuche

```
Response searchProducts(Request request) {
    final params = request.url.queryParameters;
    final query = params['q']?.toLowerCase();

    if (query == null || query.isEmpty) {
        return badRequest('Search query is required');
    }

    var products = productRepo.findAll();

    // Suche in Name und Beschreibung
```

```
products = products.where((p) =>
    p.name.toLowerCase().contains(query) ||
    (p.description?.toLowerCase().contains(query) ?? false)
).toList();

// ... Pagination ...
}
```

Request

```
GET /api/products/search?q=laptop
```

Erweiterte Suche

```
Response advancedSearch(Request request) {
    final params = request.url.queryParameters;

    var products = productRepo.findAll();

    // Textsuche
    final q = params['q']?.toLowerCase();
    if (q != null && q.isNotEmpty) {
        products = products.where((p) =>
            p.name.toLowerCase().contains(q) ||
            (p.description?.toLowerCase().contains(q) ?? false)
        ).toList();
    }

    // Kategorie
    final category = params['category'];
    if (category != null) {
        products = products.where((p) => p.category == category).toList();
    }

    // Preisbereich
    final minPrice = double.tryParse(params['minPrice'] ?? '');
    final maxPrice = double.tryParse(params['maxPrice'] ?? '');
    if (minPrice != null) {
        products = products.where((p) => p.price >= minPrice).toList();
    }
    if (maxPrice != null) {
        products = products.where((p) => p.price <= maxPrice).toList();
    }

    // Rating
    final minRating = double.tryParse(params['minRating'] ?? '');
    if (minRating != null) {
        products = products.where((p) => p.rating >= minRating).toList();
    }

    // Tags (mehrere möglich)
    final tags = params['tags']?.split(',');
}
```

```

if (tags != null && tags.isNotEmpty) {
    products = products.where((p) =>
        tags.any((tag) => p.tags.contains(tag))
    ).toList();
}

// ... Sortierung und Pagination ...
}

```

Request

```

GET /api/products/search?q=laptop&category=electronics&minPrice=500&maxPrice_ ↵
↵ =1500&minRating=4&tags=gaming,portable

```

### 2.7.0.9 Wiederverwendbare Pagination-Klasse

```

class PaginatedResponse<T> {
    final List<T> data;
    final int total;
    final int page;
    final int perPage;

    PaginatedResponse({
        required this.data,
        required this.total,
        required this.page,
        required this.perPage,
    });

    int get totalPages => (total / perPage).ceil();
    bool get hasNextPage => page < totalPages;
    bool get hasPrevPage => page > 1;

    Map<String, dynamic> toJson(Map<String, dynamic> Function(T) itemToJson) => {
        'data': data.map(itemToJson).toList(),
        'meta': {
            'total': total,
            'page': page,
            'perPage': perPage,
            'totalPages': totalPages,
            'hasNextPage': hasNextPage,
            'hasPrevPage': hasPrevPage,
        },
    };
}

// Verwendung
final paginated = PaginatedResponse(
    data: products,
    total: totalCount,
    page: page,

```

```

    perPage: perPage,
  );

  return jsonResponse(paginated.toJson((p) => p.toJson()));

```

### 2.7.0.10 Zusammenfassung

Feature	Query-Parameter	Beispiel
Seite	page	?page=2
Pro Seite	perPage	?perPage=50
Sortierfeld	sort	?sort=price
Sortierrichtung	order	?order=desc
Filter	Feldname	?category=electronics
Bereich	min/max	?minPrice=10&maxPrice=100
Suche	q	?q=laptop
Cursor	cursor	?cursor=abc123

### 2.7.0.11 Nächste Schritte

In der nächsten Einheit lernst du **API-Dokumentation**: Wie du deine API mit OpenAPI/Swagger dokumentierst.

## 2.7.1 Übung

### 2.7.1.1 Ziel

Implementiere eine Product-API mit vollständiger Pagination, Filtering und Suche.

### 2.7.1.2 Aufgabe 1: Basis-Pagination (15 min)

Erstelle einen Endpoint mit Offset-basierter Pagination.

Endpoint

GET /api/products

Query-Parameter

Parameter	Default	Beschreibung
page	1	Aktuelle Seite
perPage	20	Elemente pro Seite (max 100)

Response

```

{
  "data": [
    { "id": "1", "name": "Laptop", "price": 999.99, ... }
  ],
  "meta": {
    "total": 150,
    "page": 1,

```

```

    "perPage": 20,
    "totalPages": 8,
    "hasNextPage": true,
    "hasPrevPage": false
  }
}

```

Anforderungen

- **page** muss mindestens 1 sein
- **perPage** muss zwischen 1 und 100 liegen
- Bei leerer Liste: leeres Array, total: 0

### 2.7.1.3 Aufgabe 2: HATEOAS Links (10 min)

Erweitere die Response um Navigation-Links.

Response

```

{
  "data": [...],
  "meta": {...},
  "links": {
    "self": "/api/products?page=2&perPage=20",
    "first": "/api/products?page=1&perPage=20",
    "last": "/api/products?page=8&perPage=20",
    "prev": "/api/products?page=1&perPage=20",
    "next": "/api/products?page=3&perPage=20"
  }
}

```

Regeln

- **prev** nur wenn **page** > 1
- **next** nur wenn **page** < **totalPages**
- Filter/Sort-Parameter in Links übernehmen

### 2.7.1.4 Aufgabe 3: Filtering (15 min)

Implementiere Filter für die Product-Liste.

Filter

Parameter	Typ	Beispiel
<b>category</b>	string	?category=electronics
<b>inStock</b>	boolean	?inStock=true
<b>minPrice</b>	number	?minPrice=50
<b>maxPrice</b>	number	?maxPrice=200
<b>brand</b>	string	?brand=Apple

Kombinierte Filter

```
# Alle Apple-Produkte zwischen 100€ und 500€, die auf Lager sind
GET /api/products?brand=Apple&minPrice=100&maxPrice=500&inStock=true
```

Response

Filter-Infos in Meta aufnehmen:

```
{
  "data": [...],
  "meta": {
    "total": 15,
    "filtered": true,
    "appliedFilters": {
      "brand": "Apple",
      "minPrice": 100,
      "maxPrice": 500,
      "inStock": true
    },
    ...
  }
}
```

#### 2.7.1.5 Aufgabe 4: Sortierung (10 min)

Implementiere Sortierung.

Parameter

Parameter	Werte	Default
sort	name, price, createdAt, rating	createdAt
order	asc, desc	desc

Beispiele

```
# Nach Preis aufsteigend
GET /api/products?sort=price&order=asc

# Nach Name absteigend
GET /api/products?sort=name&order=desc

# Neueste zuerst (Default)
GET /api/products
```

Validierung

- Unbekannte Sort-Felder ignorieren oder Fehler
- Ungültige Order-Werte auf **asc** setzen

#### 2.7.1.6 Aufgabe 5: Suche (15 min)

Implementiere einen Such-Endpoint.

Endpoint

GET `/api/products/search`

Parameter

Parameter	Beschreibung
q	Suchbegriff (Pflicht)
Alle Filter	Wie bei List
Pagination	Wie bei List

Suchlogik

- Suche in: `name`, `description`, `brand`
- Case-insensitive
- Partial Match (enthält)

Response

```
{
  "data": [...],
  "meta": {
    "query": "laptop",
    "total": 25,
    ...
  }
}
```

Fehler

- 400 wenn q fehlt oder leer

### 2.7.1.7 Aufgabe 6: Cursor Pagination (Bonus, 15 min)

Implementiere Cursor-basierte Pagination für einen Feed-Endpoint.

Endpoint

GET `/api/feed`

Parameter

Parameter	Beschreibung
limit	Anzahl (default 20, max 100)
cursor	ID des letzten Elements

Response

```
{
  "data": [
    {"id": "post-50", ...},
    {"id": "post-49", ...}
  ],
  "meta": {
    "limit": 20,
    "hasMore": true
  }
}
```

```
},  
"cursors": {  
  "before": "post-50",  
  "after": "post-31"  
}  
}
```

Verwendung

*# Erste Seite*

```
curl http://localhost:8080/api/feed?limit=20
```

*# Nächste Seite*

```
curl http://localhost:8080/api/feed?limit=20&cursor=post-31
```

### 2.7.1.8 Test-Daten

Erstelle Seed-Daten für mindestens 50 Produkte:

```
void _seedProducts() {  
  final categories = ['electronics', 'clothing', 'home', 'sports'];  
  final brands = ['Apple', 'Samsung', 'Nike', 'Adidas', 'Sony'];  
  
  for (var i = 1; i <= 50; i++) {  
    _products['product-$i'] = Product(  
      id: 'product-$i',  
      name: 'Product $i',  
      description: 'Description for product $i',  
      price: (Random().nextDouble() * 1000).roundToDouble(),  
      category: categories[i % categories.length],  
      brand: brands[i % brands.length],  
      stock: Random().nextInt(100),  
      rating: 1 + Random().nextDouble() * 4,  
      createdAt: DateTime.now().subtract(Duration(days: i)),  
    );  
  }  
}
```

### 2.7.1.9 Testen

*# Basis-Pagination*

```
curl "http://localhost:8080/api/products"
```

```
curl "http://localhost:8080/api/products?page=2&perPage=10"
```

*# Filter*

```
curl "http://localhost:8080/api/products?category=electronics"
```

```
curl "http://localhost:8080/api/products?minPrice=100&maxPrice=500"
```

```
curl "http://localhost:8080/api/products?inStock=true&brand=Apple"
```

*# Sortierung*

```
curl "http://localhost:8080/api/products?sort=price&order=asc"
```

```

curl "http://localhost:8080/api/products?sort=rating&order=desc"

# Suche
curl "http://localhost:8080/api/products/search?q=laptop"
curl "http://localhost:8080/api/products/search?q=phone&category=electronics"

# Kombiniert
curl ' '
  ↪ "http://localhost:8080/api/products?category=electronics&sort=price&order=asc&page=1&perPage=10"

# Cursor (Bonus)
curl "http://localhost:8080/api/feed?limit=10"
curl "http://localhost:8080/api/feed?limit=10&cursor=post-41"

```

#### 2.7.1.10 Abgabe-Checkliste

- ☐ Offset Pagination funktioniert
- ☐ Meta-Daten korrekt (total, totalPages, hasNext, hasPrev)
- ☐ HATEOAS Links vorhanden
- ☐ Filter: category, inStock, minPrice, maxPrice, brand
- ☐ Filter können kombiniert werden
- ☐ Sortierung: sort + order Parameter
- ☐ Suche: /search?q= Endpoint
- ☐ Suche durchsucht name, description, brand
- ☐ Filter in Links übernommen
- ☐ (Bonus) Cursor Pagination implementiert

### 2.7.2 Lösung

#### 2.7.2.1 Vollständige Lösung

```

import 'dart:convert';
import 'dart:math';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';

// =====
// Model
// =====

class Product {
  final String id;
  final String name;
  final String? description;
  final double price;
  final String category;
  final String brand;
  final int stock;
  final double rating;
  final DateTime createdAt;
}

```

```
Product({
    required this.id,
    required this.name,
    this.description,
    required this.price,
    required this.category,
    required this.brand,
    required this.stock,
    required this.rating,
    required this.createdAt,
});

Map<String, dynamic> toJson() => {
    'id': id,
    'name': name,
    if (description != null) 'description': description,
    'price': price,
    'category': category,
    'brand': brand,
    'stock': stock,
    'rating': rating,
    'inStock': stock > 0,
    'createdAt': createdAt.toIso8601String(),
};

}

class Post {
    final String id;
    final String content;
    final DateTime createdAt;

    Post({required this.id, required this.content, required this.createdAt});

    Map<String, dynamic> toJson() => {
        'id': id,
        'content': content,
        'createdAt': createdAt.toIso8601String(),
    };
}

// =====
// Storage & Seed Data
// =====

final _products = <String, Product>{};
final _posts = <String, Post>{};

void _seedData() {
    final random = Random(42);
    final categories = ['electronics', 'clothing', 'home', 'sports', 'books'];
```

```

final brands = ['Apple', 'Samsung', 'Nike', 'Adidas', 'Sony', 'LG', 'Puma'];
final adjectives = ['Premium', 'Classic', 'Pro', 'Ultra', 'Mega', 'Super'];

for (var i = 1; i <= 50; i++) {
    final category = categories[i % categories.length];
    final brand = brands[i % brands.length];
    final adj = adjectives[random.nextInt(adjectives.length)];

    _products['product-$i'] = Product(
        id: 'product-$i',
        name: '$brand $adj ${category.substring(0,
↵ 1).toUpperCase()}${category.substring(1)} $i',
        description: 'High-quality $category product from $brand. Model number
↵ $i.',
        price: (random.nextDouble() * 1000 + 10).roundToDouble(),
        category: category,
        brand: brand,
        stock: random.nextInt(100),
        rating: 1 + random.nextDouble() * 4,
        createdAt: DateTime.now().subtract(Duration(days: i)),
    );
}

for (var i = 1; i <= 100; i++) {
    _posts['post-$i'] = Post(
        id: 'post-$i',
        content: 'This is post number $i with some interesting content.',
        createdAt: DateTime.now().subtract(Duration(hours: i)),
    );
}
}

// =====
// Helper Functions
// =====

Response jsonResponse(Object? data, {int statusCode = 200}) {
    return Response(
        statusCode,
        body: jsonEncode(data),
        headers: {'content-type': 'application/json'},
    );
}

Response badRequest(String message) {
    return jsonResponse({'error': message}, statusCode: 400);
}

// =====
// Aufgabe 1-4: Product List mit Pagination, Filter, Sort
// =====

```

```
Response listProducts(Request request) {
    final params = request.url.queryParameters;

    // ===== Aufgabe 1: Pagination =====
    final page = (int.tryParse(params['page'] ?? '1') ?? 1).clamp(1, 1000);
    final perPage = (int.tryParse(params['perPage'] ?? '20') ?? 20).clamp(1, 100);

    // ===== Aufgabe 3: Filtering =====
    var products = _products.values.toList();
    final appliedFilters = <String, dynamic>{};

    // Filter: category
    final category = params['category'];
    if (category != null) {
        products = products.where((p) => p.category == category).toList();
        appliedFilters['category'] = category;
    }

    // Filter: brand
    final brand = params['brand'];
    if (brand != null) {
        products = products.where((p) => p.brand == brand).toList();
        appliedFilters['brand'] = brand;
    }

    // Filter: inStock
    final inStock = params['inStock'];
    if (inStock == 'true') {
        products = products.where((p) => p.stock > 0).toList();
        appliedFilters['inStock'] = true;
    } else if (inStock == 'false') {
        products = products.where((p) => p.stock == 0).toList();
        appliedFilters['inStock'] = false;
    }

    // Filter: minPrice
    final minPrice = double.tryParse(params['minPrice'] ?? '');
    if (minPrice != null) {
        products = products.where((p) => p.price >= minPrice).toList();
        appliedFilters['minPrice'] = minPrice;
    }

    // Filter: maxPrice
    final maxPrice = double.tryParse(params['maxPrice'] ?? '');
    if (maxPrice != null) {
        products = products.where((p) => p.price <= maxPrice).toList();
        appliedFilters['maxPrice'] = maxPrice;
    }

    // Filter: minRating
```

```
final minRating = double.tryParse(params['minRating'] ?? '');
if (minRating != null) {
    products = products.where((p) => p.rating >= minRating).toList();
    appliedFilters['minRating'] = minRating;
}

// ===== Aufgabe 4: Sortierung =====
final sort = params['sort'] ?? 'createdAt';
final order = params['order'] ?? 'desc';

final validSortFields = ['name', 'price', 'createdAt', 'rating', 'stock'];
if (validSortFields.contains(sort)) {
    products = [...products]..sort((a, b) {
        int cmp;
        switch (sort) {
            case 'name':
                cmp = a.name.compareTo(b.name);
                break;
            case 'price':
                cmp = a.price.compareTo(b.price);
                break;
            case 'rating':
                cmp = a.rating.compareTo(b.rating);
                break;
            case 'stock':
                cmp = a.stock.compareTo(b.stock);
                break;
            case 'createdAt':
            default:
                cmp = a.createdAt.compareTo(b.createdAt);
        }
        return order == 'desc' ? -cmp : cmp;
    });
}

// ===== Pagination berechnen =====
final total = products.length;
final totalPages = total > 0 ? (total / perPage).ceil() : 1;
final offset = (page - 1) * perPage;
final pagedProducts = products.skip(offset).take(perPage).toList();

// ===== Aufgabe 2: HATEOAS Links =====
final baseUrl = '/api/products';
final queryParams = <String, String>{};

// Filter/Sort in Links übernehmen
if (category != null) queryParams['category'] = category;
if (brand != null) queryParams['brand'] = brand;
if (inStock != null) queryParams['inStock'] = inStock;
if (minPrice != null) queryParams['minPrice'] = minPrice.toString();
if (maxPrice != null) queryParams['maxPrice'] = maxPrice.toString();
```

```

if (sort != 'createdAt') queryParams['sort'] = sort;
if (order != 'desc') queryParams['order'] = order;

String buildUrl(int p) {
    final params = {...queryParams, 'page': p.toString(), 'perPage':
↪ perPage.toString()};
↪ final query = params.entries.map((e) => '${e.key}=${e.value}').join('&');
    return '$baseUrl?$query';
}

final links = <String, String>{
    'self': buildUrl(page),
    'first': buildUrl(1),
    'last': buildUrl(totalPages),
};
if (page > 1) links['prev'] = buildUrl(page - 1);
if (page < totalPages) links['next'] = buildUrl(page + 1);

// ===== Response =====
return jsonResponse({
    'data': pagedProducts.map((p) => p.toJson()).toList(),
    'meta': {
        'total': total,
        'page': page,
        'perPage': perPage,
        'totalPages': totalPages,
        'hasNextPage': page < totalPages,
        'hasPrevPage': page > 1,
        if (appliedFilters.isNotEmpty) 'filtered': true,
        if (appliedFilters.isNotEmpty) 'appliedFilters': appliedFilters,
        'sort': sort,
        'order': order,
    },
    'links': links,
});
}

// =====
// Aufgabe 5: Suche
// =====

Response searchProducts(Request request) {
    final params = request.url.queryParameters;

    // Suchbegriff (Pflicht)
    final query = params['q']?.toLowerCase();
    if (query == null || query.isEmpty) {
        return badRequest('Search query (q) is required');
    }

    // Pagination

```

```
final page = (int.tryParse(params['page'] ?? '1') ?? 1).clamp(1, 1000);
final perPage = (int.tryParse(params['perPage'] ?? '20') ?? 20).clamp(1, 100);

// Suche in name, description, brand
var products = _products.values.where((p) =>
    p.name.toLowerCase().contains(query) ||
    (p.description?.toLowerCase().contains(query) ?? false) ||
    p.brand.toLowerCase().contains(query)
).toList();

// Zusätzliche Filter
final category = params['category'];
if (category != null) {
    products = products.where((p) => p.category == category).toList();
}

final brand = params['brand'];
if (brand != null) {
    products = products.where((p) => p.brand == brand).toList();
}

final inStock = params['inStock'];
if (inStock == 'true') {
    products = products.where((p) => p.stock > 0).toList();
}

final minPrice = double.tryParse(params['minPrice'] ?? '');
if (minPrice != null) {
    products = products.where((p) => p.price >= minPrice).toList();
}

final maxPrice = double.tryParse(params['maxPrice'] ?? '');
if (maxPrice != null) {
    products = products.where((p) => p.price <= maxPrice).toList();
}

// Sortierung (Relevanz standardmäßig = nach Name-Match zuerst)
final sort = params['sort'];
final order = params['order'] ?? 'asc';

if (sort != null) {
    products = [...products]..sort((a, b) {
        int cmp;
        switch (sort) {
            case 'name':
                cmp = a.name.compareTo(b.name);
                break;
            case 'price':
                cmp = a.price.compareTo(b.price);
                break;
            case 'rating':
```

```

        cmp = a.rating.compareTo(b.rating);
        break;
    default:
        cmp = a.createdAt.compareTo(b.createdAt);
    }
    return order == 'desc' ? -cmp : cmp;
});
}

// Pagination
final total = products.length;
final totalPages = total > 0 ? (total / perPage).ceil() : 1;
final offset = (page - 1) * perPage;
final pagedProducts = products.skip(offset).take(perPage).toList();

return jsonResponse({
    'data': pagedProducts.map((p) => p.toJson()).toList(),
    'meta': {
        'query': query,
        'total': total,
        'page': page,
        'perPage': perPage,
        'totalPages': totalPages,
        'hasNextPage': page < totalPages,
        'hasPrevPage': page > 1,
    },
});
}

// =====
// Aufgabe 6: Cursor Pagination (Bonus)
// =====

Response listFeed(Request request) {
    final params = request.url.queryParameters;

    // Parameter
    final limit = (int.tryParse(params['limit'] ?? '20') ?? 20).clamp(1, 100);
    final cursor = params['cursor'];

    // Posts nach createdAt sortiert (neueste zuerst)
    var allPosts = _posts.values.toList()
        ..sort((a, b) => b.createdAt.compareTo(a.createdAt));

    List<Post> posts;
    String? beforeCursor;

    if (cursor != null) {
        // Finde Index des Cursor-Posts
        final cursorIndex = allPosts.indexWhere((p) => p.id == cursor);
        if (cursorIndex == -1) {

```

```

        return badRequest('Invalid cursor');
    }
    beforeCursor = cursor;
    posts = allPosts.skip(cursorIndex + 1).take(limit).toList();
} else {
    posts = allPosts.take(limit).toList();
}

final hasMore = posts.length == limit;
final afterCursor = posts.isNotEmpty ? posts.last.id : null;

return jsonResponse({
    'data': posts.map((p) => p.toJson()).toList(),
    'meta': {
        'limit': limit,
        'hasMore': hasMore,
        'count': posts.length,
    },
    'cursors': {
        if (beforeCursor != null) 'before': beforeCursor,
        if (hasMore && afterCursor != null) 'after': afterCursor,
    },
});
}

// =====
// Zusätzliche Endpoints
// =====

Response getCategories(Request request) {
    final categories = _products.values.map((p) =>
↪   p.category).toSet().toList()..sort();
    return jsonResponse({
        'data': categories,
        'total': categories.length,
    });
}

Response getBrands(Request request) {
    final brands = _products.values.map((p) => p.brand).toSet().toList()..sort();
    return jsonResponse({
        'data': brands,
        'total': brands.length,
    });
}

Response getProduct(Request request, String id) {
    final product = _products[id];
    if (product == null) {
        return jsonResponse({'error': 'Product not found'}, statusCode: 404);
    }
}

```

```

    return jsonResponse(product.toJson());
}

// =====
// Main
// =====

void main() async {
  _seedData();

  final router = Router();

  // Products
  router.get('/api/products', listProducts);
  router.get('/api/products/search', searchProducts);
  router.get('/api/products/<id>', getProduct);

  // Meta-Endpoints
  router.get('/api/categories', getCategories);
  router.get('/api/brands', getBrands);

  // Feed (Cursor Pagination)
  router.get('/api/feed', listFeed);

  final handler = Pipeline()
    .addMiddleware(logRequests())
    .addHandler(router.call);

  await shelf_io.serve(handler, 'localhost', 8080);
  print('Server: http://localhost:8080');
  print('');
  print('Endpoints:');
  print('  GET /api/products');
  print('  GET /api/products/search?q=...');
  print('  GET /api/products/:id');
  print('  GET /api/categories');
  print('  GET /api/brands');
  print('  GET /api/feed (Cursor Pagination)');
  print('');
  print('Test-Befehle:');
  print('  curl "http://localhost:8080/api/products?page=1&perPage=5"');
  print('  curl                                     ↪
↪ "http://localhost:8080/api/products?category=electronics&sort=price&order=asc"');
  print('  curl "http://localhost:8080/api/products/search?q=apple"');
  print('  curl "http://localhost:8080/api/feed?limit=10"');
}

```

### 2.7.2.2 Test-Befehle

```

# ===== Basis Pagination =====
curl "http://localhost:8080/api/products"
curl "http://localhost:8080/api/products?page=1&perPage=5"
curl "http://localhost:8080/api/products?page=2&perPage=5"
curl "http://localhost:8080/api/products?page=10&perPage=10"

# ===== Filtering =====
curl "http://localhost:8080/api/products?category=electronics"
curl "http://localhost:8080/api/products?brand=Apple"
curl "http://localhost:8080/api/products?inStock=true"
curl "http://localhost:8080/api/products?minPrice=100&maxPrice=500"
curl "http://localhost:8080/api/products?minRating=4"

# Kombinierte Filter
curl ↵
  ↪ "http://localhost:8080/api/products?category=electronics&brand=Apple&inStock=true"
curl "http://localhost:8080/api/products?minPrice=50&maxPrice=200&inStock=true"

# ===== Sortierung =====
curl "http://localhost:8080/api/products?sort=price&order=asc"
curl "http://localhost:8080/api/products?sort=price&order=desc"
curl "http://localhost:8080/api/products?sort=name&order=asc"
curl "http://localhost:8080/api/products?sort=rating&order=desc"

# Filter + Sort + Pagination
curl ↵
  ↪ "http://localhost:8080/api/products?category=electronics&sort=price&order=asc&page=1&perPage=5"

# ===== Suche =====
curl "http://localhost:8080/api/products/search?q=apple"
curl "http://localhost:8080/api/products/search?q=premium"
curl "http://localhost:8080/api/products/search?q=electronics"
curl "http://localhost:8080/api/products/search?q=apple&category=electronics"
curl ↵
  ↪ "http://localhost:8080/api/products/search?q=product&minPrice=100&maxPrice=300"

# ===== Cursor Pagination =====
# Erste Seite
curl "http://localhost:8080/api/feed?limit=10"

# Nächste Seite (Cursor aus vorheriger Response verwenden)
curl "http://localhost:8080/api/feed?limit=10&cursor=post-10"
curl "http://localhost:8080/api/feed?limit=10&cursor=post-20"

# ===== Meta-Endpoints =====
curl "http://localhost:8080/api/categories"
curl "http://localhost:8080/api/brands"

```

### 2.7.2.3 Ausgabe-Beispiele

Pagination Response

```
{
  "data": [
    {
      "id": "product-1",
      "name": "Apple Premium Electronics 1",
      "price": 523.0,
      "category": "electronics",
      "brand": "Apple",
      "stock": 42,
      "rating": 3.8,
      "inStock": true,
      "createdAt": "2024-01-14T10:00:00.000Z"
    }
  ],
  "meta": {
    "total": 50,
    "page": 1,
    "perPage": 5,
    "totalPages": 10,
    "hasNextPage": true,
    "hasPrevPage": false,
    "sort": "createdAt",
    "order": "desc"
  },
  "links": {
    "self": "/api/products?page=1&perPage=5",
    "first": "/api/products?page=1&perPage=5",
    "last": "/api/products?page=10&perPage=5",
    "next": "/api/products?page=2&perPage=5"
  }
}
```

Filtered Response

```
{
  "data": [...],
  "meta": {
    "total": 8,
    "page": 1,
    "perPage": 20,
    "totalPages": 1,
    "hasNextPage": false,
    "hasPrevPage": false,
    "filtered": true,
    "appliedFilters": {
      "category": "electronics",
      "brand": "Apple",
      "inStock": true
    }
  },
  "sort": "createdAt",
  "order": "desc"
}
```

```
},  
"links": {...}  
}
```

Search Response

```
{  
  "data": [  
    {"id": "product-1", "name": "Apple Premium Electronics 1", ...}  
  ],  
  "meta": {  
    "query": "apple",  
    "total": 8,  
    "page": 1,  
    "perPage": 20,  
    "totalPages": 1,  
    "hasNextPage": false,  
    "hasPrevPage": false  
  }  
}
```

Cursor Pagination Response

```
{  
  "data": [  
    {"id": "post-1", "content": "...", "createdAt": "..."},  
    {"id": "post-2", "content": "...", "createdAt": "..."}  
  ],  
  "meta": {  
    "limit": 10,  
    "hasMore": true,  
    "count": 10  
  },  
  "cursors": {  
    "after": "post-10"  
  }  
}
```

#### 2.7.2.4 Wichtige Patterns

Filter in Links übernehmen

```
// Alle Filter/Sort Parameter sammeln  
final queryParams = <String, String>{};  
if (category != null) queryParams['category'] = category;  
if (sort != 'createdAt') queryParams['sort'] = sort;  
  
// URL mit Parametern bauen  
String buildUrl(int page) {  
  final params = {...queryParams, 'page': page.toString()};  
  final query = params.entries.map((e) => '${e.key}=${e.value}').join('&');  
  return '$baseUrl?$query';  
}
```

```
}
```

Sichere Parameter-Extraktion

```
final page = (int.tryParse(params['page'] ?? '1') ?? 1).clamp(1, 1000);  
final perPage = (int.tryParse(params['perPage'] ?? '20') ?? 20).clamp(1, 100);
```

Cursor-Position finden

```
final cursorIndex = allPosts.indexWhere((p) => p.id == cursor);  
if (cursorIndex == -1) {  
    return badRequest('Invalid cursor');  
}  
posts = allPosts.skip(cursorIndex + 1).take(limit).toList();
```

## 2.7.3 Ressourcen

### 2.7.3.1 Offizielle Dokumentation

- REST API Pagination
- JSON:API Pagination
- Cursor Pagination

### 2.7.3.2 Cheat Sheet: Offset Pagination

```
Response listItems(Request request) {  
    final params = request.url.queryParameters;  
  
    // Parameter mit Defaults  
    final page = int.tryParse(params['page'] ?? '1') ?? 1;  
    final perPage = int.tryParse(params['perPage'] ?? '20') ?? 20;  
  
    // Limits  
    final safePage = page.clamp(1, 1000);  
    final safePerPage = perPage.clamp(1, 100);  
  
    // Daten holen  
    final allItems = repo.findAll();  
    final total = allItems.length;  
  
    // Slice  
    final offset = (safePage - 1) * safePerPage;  
    final items = allItems.skip(offset).take(safePerPage).toList();  
  
    // Metadaten  
    final totalPages = (total / safePerPage).ceil();  
  
    return jsonResponse({  
        'data': items.map((i) => i.toJson()).toList(),  
        'meta': {  
            'total': total,  
            'page': safePage,  

```

```
        'perPage': safePerPage,
        'totalPages': totalPages,
        'hasNextPage': safePage < totalPages,
        'hasPrevPage': safePage > 1,
    },
    });
}
```

### 2.7.3.3 Cheat Sheet: Cursor Pagination

```
Response listItems(Request request) {
    final params = request.url.queryParameters;
    final limit = (int.tryParse(params['limit'] ?? '20') ?? 20).clamp(1, 100);
    final cursor = params['cursor'];

    List<Item> items;
    if (cursor != null) {
        items = repo.findAfterId(cursor, limit);
    } else {
        items = repo.findAll().take(limit).toList();
    }

    final nextCursor = items.isNotEmpty ? items.last.id : null;
    final hasMore = items.length == limit;

    return jsonResponse({
        'data': items.map((i) => i.toJson()).toList(),
        'meta': {'limit': limit, 'hasMore': hasMore},
        'cursors': {
            if (cursor != null) 'before': cursor,
            if (hasMore && nextCursor != null) 'after': nextCursor,
        },
    });
}
```

### 2.7.3.4 Cheat Sheet: Filtering

```
Response listProducts(Request request) {
    final params = request.url.queryParameters;
    var items = repo.findAll();

    // Exakter Match
    final category = params['category'];
    if (category != null) {
        items = items.where((i) => i.category == category).toList();
    }

    // Boolean
    if (params['active'] == 'true') {
```

```
    items = items.where((i) => i.active).toList();
}

// Bereich
final minPrice = double.tryParse(params['minPrice'] ?? '');
final maxPrice = double.tryParse(params['maxPrice'] ?? '');
if (minPrice != null) {
    items = items.where((i) => i.price >= minPrice).toList();
}
if (maxPrice != null) {
    items = items.where((i) => i.price <= maxPrice).toList();
}

// Liste (z.B. ?tags=a,b,c)
final tags = params['tags']?.split(',');
if (tags != null && tags.isNotEmpty) {
    items = items.where((i) => tags.any((t) => i.tags.contains(t))).toList();
}

// ... pagination ...
}
```

#### 2.7.3.5 Cheat Sheet: Sortierung

```
Response listItems(Request request) {
    final params = request.url.queryParameters;
    var items = repo.findAll();

    final sort = params['sort'];
    final order = params['order'] ?? 'asc';

    if (sort != null) {
        items = [...items]..sort((a, b) {
            final cmp = switch (sort) {
                'name' => a.name.compareTo(b.name),
                'price' => a.price.compareTo(b.price),
                'createdAt' => a.createdAt.compareTo(b.createdAt),
                _ => 0,
            };
            return order == 'desc' ? -cmp : cmp;
        });
    }

    // ... pagination ...
}
```

#### 2.7.3.6 Cheat Sheet: Suche

```

Response searchItems(Request request) {
    final q = request.url.queryParameters['q']?.toLowerCase();

    if (q == null || q.isEmpty) {
        return badRequest('Search query required');
    }

    final items = repo.findAll().where((i) =>
        i.name.toLowerCase().contains(q) ||
        (i.description?.toLowerCase().contains(q) ?? false)
    ).toList();

    // ... pagination ...
}

```

### 2.7.3.7 Cheat Sheet: HATEOAS Links

```

Map<String, String> buildLinks(String baseUrl, int page, int perPage, int ↵
    ↵ totalPages) {
    return {
        'self': '$baseUrl?page=$page&perPage=$perPage',
        'first': '$baseUrl?page=1&perPage=$perPage',
        'last': '$baseUrl?page=$totalPages&perPage=$perPage',
        if (page > 1) 'prev': '$baseUrl?page=${page - 1}&perPage=$perPage',
        if (page < totalPages) 'next': '$baseUrl?page=${page + 1}&perPage=$perPage',
    };
}

```

### 2.7.3.8 Cheat Sheet: PaginatedResponse Klasse

```

class PaginatedResponse<T> {
    final List<T> data;
    final int total;
    final int page;
    final int perPage;

    PaginatedResponse({
        required this.data,
        required this.total,
        required this.page,
        required this.perPage,
    });

    int get totalPages => (total / perPage).ceil();
    bool get hasNext => page < totalPages;
    bool get hasPrev => page > 1;

    Map<String, dynamic> toJson(Object Function(T) toJsonFn) => {
        'data': data.map(toJsonFn).toList(),
    }
}

```

```

    'meta': {
      'total': total,
      'page': page,
      'perPage': perPage,
      'totalPages': totalPages,
      'hasNextPage': hasNext,
      'hasPrevPage': hasPrev,
    },
  };
}

```

### 2.7.3.9 Query-Parameter Übersicht

Parameter	Beschreibung	Beispiel
page	Aktuelle Seite	?page=2
perPage	Elemente pro Seite	?perPage=50
limit	Anzahl (Cursor)	?limit=20
cursor	Cursor-Position	?cursor=abc
sort	Sortierfeld	?sort=price
order	asc/desc	?order=desc
q	Suchbegriff	?q=laptop
[field]	Filter	?category=tech
min[field]	Minimum	?minPrice=10
max[field]	Maximum	?maxPrice=100

### 2.7.3.10 Offset vs. Cursor Pagination

Aspekt	Offset	Cursor
Einfachheit	Einfach	Komplexer
Performance	Bei großen Offsets langsam	Konsistent schnell
Echtzeit-Daten	Probleme bei Änderungen	Stabil
Beliebige Seiten	Möglich	Nicht möglich
Use Case	Klassische Listen	Feeds, Timelines

### 2.7.3.11 Test-Befehle

```

# Pagination
curl "http://localhost:8080/api/products?page=1&perPage=10"
curl "http://localhost:8080/api/products?page=2&perPage=10"

# Cursor
curl "http://localhost:8080/api/posts?limit=10"
curl "http://localhost:8080/api/posts?limit=10&cursor=post-10"

# Filter
curl "http://localhost:8080/api/products?category=electronics"
curl "http://localhost:8080/api/products?minPrice=50&maxPrice=200"

```

```

curl "http://localhost:8080/api/products?inStock=true"

# Sortierung
curl "http://localhost:8080/api/products?sort=price&order=asc"
curl "http://localhost:8080/api/products?sort=name&order=desc"

# Suche
curl "http://localhost:8080/api/products/search?q=laptop"

# Kombiniert
curl "http://localhost:8080/api/products?category=electronics&sort=price&order=asc&page=1&perPage=10"

```

### 2.7.3.12 Best Practices

1. **Limits setzen:** Max perPage begrenzen (z.B. 100)
2. **Defaults definieren:** page=1, perPage=20
3. **Metadaten mitliefern:** total, totalPages, hasNext
4. **HATEOAS Links:** Navigation vereinfachen
5. **Konsistente Benennung:** camelCase oder snake\_case
6. **Cursor für Feeds:** Bei Echtzeit-Daten bevorzugen

## 2.8 Einheit 6.8: API Dokumentation

### 2.8.0.1 Lernziele

Nach dieser Einheit kannst du: - OpenAPI/Swagger-Spezifikationen verstehen und schreiben  
 - API-Dokumentation automatisch generieren - Interaktive API-Dokumentation bereitstellen -  
 Best Practices für API-Dokumentation anwenden

### 2.8.0.2 Warum API-Dokumentation?

Ohne Dokumentation

Entwickler: "Wie rufe ich den Login-Endpoint auf?"

Backend-Dev: "POST /api/auth mit email und password im Body"

Entwickler: "Welches Format? Was kommt zurück? Welche Fehler gibt es?"

Backend-Dev: "Moment, ich schau im Code nach..."

Mit Dokumentation

- **Self-Service:** Entwickler finden alle Infos selbst
- **Konsistenz:** Eine zentrale Quelle der Wahrheit
- **Testbarkeit:** Endpoints direkt in der Doku testen
- **Onboarding:** Neue Teammitglieder schneller produktiv

### 2.8.0.3 OpenAPI Spezifikation

**OpenAPI** (früher Swagger) ist der Industriestandard für REST-API-Dokumentation.

Grundstruktur (YAML)

```

openapi: 3.0.3
info:

```

```
title: Produkt-API
description: REST API für Produktverwaltung
version: 1.0.0
contact:
  name: API Support
  email: api@example.com

servers:
- url: http://localhost:8080
  description: Lokaler Entwicklungsserver
- url: https://api.example.com
  description: Produktionsserver

paths:
  /api/products:
    get:
      summary: Alle Produkte abrufen
      # ... Endpoint-Details
```

Basis-Felder

Feld	Beschreibung
openapi	Version der OpenAPI-Spezifikation
info	API-Metadaten (Titel, Version, Beschreibung)
servers	Verfügbare Server-URLs
paths	Alle API-Endpoints
components	Wiederverwendbare Schemas, Parameter

#### 2.8.0.4 Endpoints dokumentieren

GET-Endpoint mit Parametern

```
paths:
  /api/products:
    get:
      summary: Produkte auflisten
      description: Gibt eine paginierte Liste aller Produkte zurück
      operationId: listProducts
      tags:
        - Products
      parameters:
        - name: page
          in: query
          description: Seitennummer
          required: false
          schema:
            type: integer
            default: 1
            minimum: 1
        - name: perPage
          in: query
```

```

    description: Elemente pro Seite
    required: false
    schema:
      type: integer
      default: 20
      minimum: 1
      maximum: 100
  - name: category
    in: query
    description: Nach Kategorie filtern
    required: false
    schema:
      type: string
      enum: [electronics, clothing, home, sports]
responses:
  '200':
    description: Erfolgreiche Antwort
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ProductList'
  '400':
    description: Ungültige Parameter
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'

```

POST-Endpoint mit Request Body

```

paths:
  /api/products:
    post:
      summary: Neues Produkt erstellen
      description: Erstellt ein neues Produkt und gibt es zurück
      operationId: createProduct
      tags:
        - Products
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ProductCreate'
            example:
              name: "Laptop Pro 15"
              description: "High-End Laptop mit 16GB RAM"
              price: 1299.99
              category: "electronics"
              brand: "TechBrand"
      responses:

```

```

'201':
  description: Produkt erfolgreich erstellt
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/Product'
  headers:
    Location:
      description: URL des neuen Produkts
      schema:
        type: string
        example: /api/products/123
'400':
  description: Validierungsfehler
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/ValidationError'
'409':
  description: Produkt existiert bereits

```

#### Pfad-Parameter

```

paths:
  /api/products/{id}:
    get:
      summary: Einzelnes Produkt abrufen
      operationId: getProduct
      tags:
        - Products
      parameters:
        - name: id
          in: path
          description: Produkt-ID
          required: true
          schema:
            type: string
            example: product-123
      responses:
        '200':
          description: Produkt gefunden
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Product'
        '404':
          description: Produkt nicht gefunden
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'

```

```
put:
  summary: Produkt aktualisieren
  operationId: updateProduct
  tags:
    - Products
  parameters:
    - name: id
      in: path
      required: true
      schema:
        type: string
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ProductUpdate'
  responses:
    '200':
      description: Produkt aktualisiert
    '404':
      description: Produkt nicht gefunden

delete:
  summary: Produkt löschen
  operationId: deleteProduct
  tags:
    - Products
  parameters:
    - name: id
      in: path
      required: true
      schema:
        type: string
  responses:
    '204':
      description: Produkt gelöscht
    '404':
      description: Produkt nicht gefunden
```

### 2.8.0.5 Schemas definieren

Components für Wiederverwendung

```
components:
  schemas:
    Product:
      type: object
      required:
        - id
```

```
- name
- price
- category
properties:
  id:
    type: string
    description: Eindeutige Produkt-ID
    example: product-123
  name:
    type: string
    description: Produktname
    minLength: 2
    maxLength: 100
    example: "Laptop Pro 15"
  description:
    type: string
    description: Produktbeschreibung
    nullable: true
    example: "High-End Laptop mit 16GB RAM"
  price:
    type: number
    format: double
    description: Preis in Euro
    minimum: 0.01
    example: 1299.99
  category:
    type: string
    enum: [electronics, clothing, home, sports, books]
    example: electronics
  brand:
    type: string
    example: TechBrand
  stock:
    type: integer
    minimum: 0
    default: 0
    example: 42
  inStock:
    type: boolean
    readOnly: true
    example: true
  createdAt:
    type: string
    format: date-time
    readOnly: true
    example: "2024-01-15T10:30:00Z"

ProductCreate:
  type: object
  required:
    - name
```

```
- price
- category
properties:
  name:
    type: string
    minLength: 2
    maxLength: 100
  description:
    type: string
    nullable: true
  price:
    type: number
    minimum: 0.01
  category:
    type: string
    enum: [electronics, clothing, home, sports, books]
  brand:
    type: string
  stock:
    type: integer
    minimum: 0
    default: 0

ProductUpdate:
  type: object
  properties:
    name:
      type: string
      minLength: 2
      maxLength: 100
    description:
      type: string
      nullable: true
    price:
      type: number
      minimum: 0.01
    category:
      type: string
      enum: [electronics, clothing, home, sports, books]
    brand:
      type: string
    stock:
      type: integer
      minimum: 0

ProductList:
  type: object
  properties:
    data:
      type: array
    items:
```

```
    $ref: '#/components/schemas/Product'
  meta:
    $ref: '#/components/schemas/PaginationMeta'
  links:
    $ref: '#/components/schemas/PaginationLinks'

PaginationMeta:
  type: object
  properties:
    total:
      type: integer
      example: 150
    page:
      type: integer
      example: 1
    perPage:
      type: integer
      example: 20
    totalPages:
      type: integer
      example: 8
    hasNextPage:
      type: boolean
      example: true
    hasPrevPage:
      type: boolean
      example: false

PaginationLinks:
  type: object
  properties:
    self:
      type: string
      example: "/api/products?page=1&perPage=20"
    first:
      type: string
      example: "/api/products?page=1&perPage=20"
    last:
      type: string
      example: "/api/products?page=8&perPage=20"
    prev:
      type: string
      nullable: true
    next:
      type: string
      nullable: true

Error:
  type: object
  required:
    - error
```

```
properties:
  error:
    type: string
    description: Fehlermeldung
    example: "Resource not found"
  code:
    type: string
    description: Fehlercode
    example: "NOT_FOUND"

ValidationError:
  type: object
  properties:
    error:
      type: string
      example: "Validation failed"
    details:
      type: array
      items:
        type: object
        properties:
          field:
            type: string
            example: "email"
          message:
            type: string
            example: "Invalid email format"
```

### 2.8.0.6 Authentifizierung dokumentieren

#### API-Key

```
components:
  securitySchemes:
    ApiKeyAuth:
      type: apiKey
      in: header
      name: X-API-Key
      description: API-Schlüssel im Header

security:
  - ApiKeyAuth: []

paths:
  /api/products:
    get:
      # Öffentlicher Endpoint (keine Auth nötig)
      security: []
      # ...

  post:
```

```
# Erfordert API-Key
security:
  - ApiKeyAuth: []
# ...
```

Bearer Token (JWT)

```
components:
  securitySchemes:
    BearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
      description: JWT Token im Authorization Header

paths:
  /api/auth/login:
    post:
      summary: Benutzer einloggen
      security: []
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              required: [email, password]
              properties:
                email:
                  type: string
                  format: email
                password:
                  type: string
                  format: password
      responses:
        '200':
          description: Login erfolgreich
          content:
            application/json:
              schema:
                type: object
                properties:
                  token:
                    type: string
                    description: JWT Token
                  expiresIn:
                    type: integer
                    description: Gültigkeit in Sekunden
        '401':
          description: Ungültige Anmeldedaten
```

```
/api/users/me:
  get:
    summary: Aktuellen Benutzer abrufen
    security:
      - BearerAuth: []
    responses:
      '200':
        description: Benutzerinfo
      '401':
        description: Nicht authentifiziert
```

### 2.8.0.7 OpenAPI in Dart bereitstellen

Statische OpenAPI-Datei

```
import 'dart:io';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';
import 'package:shelf_static/shelf_static.dart';

void main() async {
  final router = Router();

  // API-Endpoints
  router.mount('/api', apiRouter().call);

  // OpenAPI Spec als JSON/YAML
  router.get('/openapi.yaml', (Request r) async {
    final file = File('openapi.yaml');
    final content = await file.readAsString();
    return Response.ok(
      content,
      headers: {'content-type': 'application/x-yaml'},
    );
  });

  router.get('/openapi.json', (Request r) async {
    final file = File('openapi.json');
    final content = await file.readAsString();
    return Response.ok(
      content,
      headers: {'content-type': 'application/json'},
    );
  });

  // Swagger UI
  router.mount('/docs/', createStaticHandler('swagger-ui/'));

  await shelf_io.serve(router.call, 'localhost', 8080);
  print('API: http://localhost:8080/api');
```

```
print('Docs: http://localhost:8080/docs/');
print('Spec: http://localhost:8080/openapi.yaml');
}
```

### Inline OpenAPI Spec

```
const openApiSpec = `
openapi: 3.0.3
info:
  title: Product API
  version: 1.0.0
paths:
  /api/products:
    get:
      summary: List products
      responses:
        '200':
          description: Success
`;

router.get('/openapi.yaml', (Request r) {
  return Response.ok(
    openApiSpec,
    headers: {'content-type': 'application/x-yaml'},
  );
});
```

### 2.8.0.8 Swagger UI einbinden

#### Download und Setup

```
# Swagger UI herunterladen
curl -L https://github.com/swagger-api/swagger-ui/archive/refs/tags/v5.11.0.tar.gz
  ↪
  ↪ -O swagger-ui.tar.gz
tar -xzf swagger-ui.tar.gz
cp -r swagger-ui-5.11.0/dist swagger-ui

# Index.html anpassen (URL ändern)
sed -i 's|https://petstore.swagger.io/v2/swagger.json|openapi.yaml|g'
  ↪ swagger-ui/swagger-initializer.js
```

swagger-initializer.js anpassen

```
window.onload = function() {
  window.ui = SwaggerUIBundle({
    url: "/openapi.yaml",
    dom_id: '#swagger-ui',
    presets: [
      SwaggerUIBundle.presets.apis,
      SwaggerUIStandalonePreset
    ],
  },
```

```
    layout: "StandaloneLayout"  
  });  
};
```

Mit Shelf Static

```
import 'package:shelf_static/shelf_static.dart';  
  
// In pubspec.yaml:  
// dependencies:  
//   shelf_static: ^1.1.0  
  
final router = Router();  
  
// Swagger UI Dateien servieren  
router.mount('/docs/', createStaticHandler(  
  'swagger-ui',  
  defaultDocument: 'index.html',  
));
```

### 2.8.0.9 Tags für Gruppierung

```
tags:  
  - name: Products  
    description: Produktverwaltung  
  - name: Categories  
    description: Kategorieverwaltung  
  - name: Auth  
    description: Authentifizierung  
  
paths:  
  /api/products:  
    get:  
      tags: [Products]  
      summary: Produkte auflisten  
  
  /api/products/{id}:  
    get:  
      tags: [Products]  
      summary: Einzelnes Produkt  
  
  /api/categories:  
    get:  
      tags: [Categories]  
      summary: Kategorien auflisten  
  
  /api/auth/login:  
    post:  
      tags: [Auth]  
      summary: Einloggen
```

### 2.8.0.10 Beispiele hinzufügen

Request/Response-Beispiele

```
paths:
  /api/products:
    post:
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ProductCreate'
            examples:
              laptop:
                summary: Laptop erstellen
                value:
                  name: "MacBook Pro 14"
                  description: "Apple M3 Pro Chip"
                  price: 2499.00
                  category: "electronics"
                  brand: "Apple"
                  stock: 50
              shirt:
                summary: T-Shirt erstellen
                value:
                  name: "Basic T-Shirt"
                  price: 19.99
                  category: "clothing"
                  brand: "BasicBrand"
      responses:
        '201':
          content:
            application/json:
              examples:
                success:
                  summary: Erfolgreich erstellt
                  value:
                    id: "product-456"
                    name: "MacBook Pro 14"
                    price: 2499.00
                    category: "electronics"
                    inStock: true
                    createdAt: "2024-01-15T12:00:00Z"
```

### 2.8.0.11 Vollständiges Beispiel

```
openapi: 3.0.3
info:
  title: E-Commerce API
  description: |
    REST API für einen Online-Shop.
```

```
## Features
- Produktverwaltung mit CRUD-Operationen
- Pagination und Filtering
- Volltextsuche
- Authentifizierung via JWT

## Rate Limits
- 100 Requests pro Minute für authentifizierte Benutzer
- 20 Requests pro Minute für anonyme Benutzer
version: 1.0.0
contact:
  name: API Team
  email: api@shop.example.com
license:
  name: MIT

servers:
- url: http://localhost:8080
  description: Development
- url: https://api.shop.example.com
  description: Production

tags:
- name: Products
  description: Produkt-Endpoints
- name: Search
  description: Such-Endpoints
- name: Auth
  description: Authentifizierung

paths:
/api/products:
  get:
    tags: [Products]
    summary: Produkte auflisten
    operationId: listProducts
    parameters:
      - $ref: '#/components/parameters/PageParam'
      - $ref: '#/components/parameters/PerPageParam'
      - name: category
        in: query
        schema:
          type: string
      - name: sort
        in: query
        schema:
          type: string
          enum: [name, price, createdAt]
      - name: order
        in: query
        schema:
```

```
    type: string
    enum: [asc, desc]
    default: asc
  responses:
    '200':
      description: Liste der Produkte
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ProductList'

  post:
    tags: [Products]
    summary: Produkt erstellen
    operationId: createProduct
    security:
      - BearerAuth: []
    requestBody:
      required: true
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ProductCreate'
    responses:
      '201':
        description: Erstellt
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Product'
      '400':
        $ref: '#/components/responses/ValidationError'
      '401':
        $ref: '#/components/responses/Unauthorized'

/api/products/{id}:
  parameters:
    - $ref: '#/components/parameters/ProductId'
  get:
    tags: [Products]
    summary: Produkt abrufen
    operationId: getProduct
    responses:
      '200':
        description: Produkt gefunden
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Product'
      '404':
        $ref: '#/components/responses/NotFound'
```

```
/api/products/search:
  get:
    tags: [Search]
    summary: Produkte suchen
    operationId: searchProducts
    parameters:
      - name: q
        in: query
        required: true
        description: Suchbegriff
        schema:
          type: string
          minLength: 2
      - $ref: '#/components/parameters/PageParam'
      - $ref: '#/components/parameters/PerPageParam'
    responses:
      '200':
        description: Suchergebnisse
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/ProductList'
      '400':
        description: Suchbegriff fehlt

components:
  securitySchemes:
    BearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT

  parameters:
    ProductId:
      name: id
      in: path
      required: true
      schema:
        type: string
      example: product-123

    PageParam:
      name: page
      in: query
      schema:
        type: integer
        default: 1
        minimum: 1

    PerPageParam:
```

```
name: perPage
in: query
schema:
  type: integer
  default: 20
  minimum: 1
  maximum: 100

responses:
  NotFound:
    description: Ressource nicht gefunden
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'

  ValidationError:
    description: Validierungsfehler
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ValidationError'

  Unauthorized:
    description: Nicht authentifiziert
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'

schemas:
  Product:
    type: object
    properties:
      id:
        type: string
      name:
        type: string
      price:
        type: number
      category:
        type: string
      inStock:
        type: boolean
      createdAt:
        type: string
        format: date-time

  ProductCreate:
    type: object
    required: [name, price, category]
```

```
properties:
  name:
    type: string
  price:
    type: number
    minimum: 0.01
  category:
    type: string

ProductList:
  type: object
  properties:
    data:
      type: array
      items:
        $ref: '#/components/schemas/Product'
    meta:
      $ref: '#/components/schemas/PaginationMeta'

PaginationMeta:
  type: object
  properties:
    total:
      type: integer
    page:
      type: integer
    totalPages:
      type: integer

Error:
  type: object
  properties:
    error:
      type: string

ValidationError:
  type: object
  properties:
    error:
      type: string
    details:
      type: array
      items:
        type: object
        properties:
          field:
            type: string
          message:
            type: string
```

### 2.8.0.12 Best Practices

#### 1. Konsistente Benennung

```
# GUT: camelCase für Parameter und Felder
parameters:
  - name: perPage
    in: query

# SCHLECHT: Inkonsistent
parameters:
  - name: per_page # snake_case
  - name: PerPage # PascalCase
```

#### 2. Aussagekräftige Beschreibungen

```
# GUT
parameters:
  - name: minPrice
    description: Mindestpreis in Euro (inklusive)
    schema:
      type: number
      minimum: 0
      example: 10.50

# SCHLECHT
parameters:
  - name: minPrice
    schema:
      type: number
```

#### 3. Beispiele überall

```
# GUT
properties:
  email:
    type: string
    format: email
    example: "max@example.com"
  createdAt:
    type: string
    format: date-time
    example: "2024-01-15T10:30:00Z"
```

#### 4. Fehler-Responses dokumentieren

```
# ALLE möglichen Fehler dokumentieren
responses:
  '200':
    description: Erfolg
  '400':
    description: Ungültige Eingabe
  '401':
    description: Nicht authentifiziert
```

```
'403':
  description: Keine Berechtigung
'404':
  description: Nicht gefunden
'409':
  description: Konflikt (z.B. Duplikat)
'500':
  description: Serverfehler
```

### 2.8.0.13 Zusammenfassung

Aspekt	Empfehlung
Format	OpenAPI 3.0 (YAML oder JSON)
UI	Swagger UI oder ReDoc
Schemas	In <code>components</code> definieren und referenzieren
Beispiele	Überall wo möglich hinzufügen
Tags	Endpoints logisch gruppieren
Auth	In <code>securitySchemes</code> dokumentieren

### 2.8.0.14 Nächste Schritte

Dies war die letzte Einheit von Block 6. Du hast nun alle Grundlagen für die Entwicklung professioneller REST-APIs mit Dart gelernt:

- REST-Prinzipien und API-Design
- JSON-Serialisierung
- Request Bodies verarbeiten
- CRUD-Operationen
- Input-Validierung
- Error Handling
- Pagination & Filtering
- API-Dokumentation

Setze dieses Wissen in deinem eigenen Projekt um!

## 2.8.1 Übung

### 2.8.1.1 Ziel

Erstelle eine vollständige OpenAPI-Dokumentation für die Produkt-API und stelle sie mit Swagger UI bereit.

### 2.8.1.2 Aufgabe 1: OpenAPI-Grundstruktur (10 min)

Erstelle eine Datei `openapi.yaml` mit der Grundstruktur.

Anforderungen

- OpenAPI Version 3.0.3
- API-Titel: "Product API"
- Version: 1.0.0
- Beschreibung: Kurze API-Beschreibung
- Server: localhost:8080 (Development)

Erwartete Struktur

```
openapi: 3.0.3
info:
  title: ...
  description: ...
  version: ...
servers:
  - url: ...
    description: ...
```

### 2.8.1.3 Aufgabe 2: Schemas definieren (15 min)

Definiere die Schemas für Product, ProductCreate und Error.

Product-Schema

Feld	Typ	Required	Beschreibung
id	string	Ja	Eindeutige ID
name	string	Ja	Produktname (2-100 Zeichen)
description	string	Nein	Beschreibung
price	number	Ja	Preis (min 0.01)
category	string	Ja	Enum: electronics, clothing, home
stock	integer	Nein	Lagerbestand (min 0, default 0)
inStock	boolean	ReadOnly	Automatisch berechnet
createdAt	date-time	ReadOnly	Erstellungsdatum

ProductCreate-Schema

Wie Product, aber ohne id, inStock, createdAt

Error-Schema

```
properties:
  error:
    type: string
  code:
    type: string
```

### 2.8.1.4 Aufgabe 3: GET /api/products dokumentieren (15 min)

Dokumentiere den List-Endpoint mit allen Query-Parametern.

Parameter

Parameter	Typ	Default	Beschreibung
page	integer	1	Seitennummer (min 1)
perPage	integer	20	Pro Seite (1-100)
category	string	-	Kategorie-Filter
sort	string	createdAt	Sortierfeld
order	string	desc	asc oder desc

Response 200

Definiere **ProductList**-Schema mit: - **data**: Array von Product - **meta**: PaginationMeta (total, page, perPage, totalPages) - **links**: PaginationLinks (self, first, last, prev, next)

Response 400

Verwende Error-Schema

#### 2.8.1.5 Aufgabe 4: CRUD-Endpoints dokumentieren (20 min)

Dokumentiere alle Product-Endpoints.

POST /api/products

- Summary: Produkt erstellen
- Request Body: ProductCreate
- Response 201: Product + Location Header
- Response 400: Validierungsfehler

GET /api/products/{id}

- Path Parameter: id (required)
- Response 200: Product
- Response 404: Nicht gefunden

PUT /api/products/{id}

- Path Parameter: id
- Request Body: ProductUpdate (alle Felder optional)
- Response 200: Product
- Response 404: Nicht gefunden

DELETE /api/products/{id}

- Path Parameter: id
- Response 204: Kein Content
- Response 404: Nicht gefunden

#### 2.8.1.6 Aufgabe 5: Such-Endpoint dokumentieren (10 min)

GET /api/products/search

- Query Parameter: q (required, minLength 2)
- Zusätzlich: page, perPage, category (wie bei List)
- Response 200: ProductList
- Response 400: Wenn q fehlt

#### 2.8.1.7 Aufgabe 6: Wiederverwendbare Components (10 min)

Refactore die Spec mit \$ref:

Parameters

```
components:
  parameters:
    ProductId:
      name: id
      in: path
      required: true
      schema:
```

```

    type: string

  PageParam:
    name: page
    in: query
    schema:
      type: integer
      default: 1

  PerPageParam:
    name: perPage
    in: query
    schema:
      type: integer
      default: 20
      maximum: 100

```

### Responses

```

components:
  responses:
    NotFound:
      description: Ressource nicht gefunden
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'

    ValidationError:
      description: Validierungsfehler
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ValidationError'

```

### Verwendung

```

paths:
  /api/products/{id}:
    parameters:
      - $ref: '#/components/parameters/ProductId'
    get:
      responses:
        '404':
          $ref: '#/components/responses/NotFound'

```

#### 2.8.1.8 Aufgabe 7: Tags hinzufügen (5 min)

Gruppieren Endpoints mit Tags:

- Products: Alle CRUD-Endpoints
- Search: Such-Endpoint

- Meta: Kategorien und Brands

```
tags:
  - name: Products
    description: Produktverwaltung (CRUD)
  - name: Search
    description: Produktsuche
  - name: Meta
    description: Metadaten (Kategorien, Marken)
```

### 2.8.1.9 Aufgabe 8: Swagger UI einrichten (15 min)

Stelle die Dokumentation mit Swagger UI bereit.

Schritte

1. OpenAPI-Spec als Endpoint bereitstellen:

```
router.get('/openapi.yaml', (Request r) async {
  final content = await File('openapi.yaml').readAsString();
  return Response.ok(content, headers: {
    'content-type': 'application/x-yaml',
  });
});
```

2. Swagger UI einbinden (Option A - CDN):

```
router.get('/docs', (Request r) {
  return Response.ok(''
<!DOCTYPE html>
<html>
<head>
  <title>API Docs</title>
  <link rel="stylesheet"
↪ href="https://unpkg.com/swagger-ui-dist@5/swagger-ui.css">
</head>
<body>
  <div id="swagger-ui"></div>
  <script
↪ src="https://unpkg.com/swagger-ui-dist@5/swagger-ui-bundle.js"></script>
  <script>
    SwaggerUIBundle({
      url: '/openapi.yaml',
      dom_id: '#swagger-ui',
    });
  </script>
</body>
</html>
'', headers: {'content-type': 'text/html'});
});
```

3. Server starten und testen:

```
curl http://localhost:8080/openapi.yaml
# Browser: http://localhost:8080/docs
```

### 2.8.1.10 Aufgabe 9: Beispiele hinzufügen (10 min)

Füge Beispiele für Request und Response hinzu.

Request-Beispiele

```
requestBody:
  content:
    application/json:
      examples:
        laptop:
          summary: Laptop erstellen
          value:
            name: "Gaming Laptop"
            description: "RTX 4080, 32GB RAM"
            price: 2499.99
            category: "electronics"
            stock: 10
        shirt:
          summary: T-Shirt erstellen
          value:
            name: "Basic T-Shirt"
            price: 19.99
            category: "clothing"
```

Response-Beispiele

```
responses:
  '200':
    content:
      application/json:
        examples:
          singleProduct:
            summary: Einzelnes Produkt
            value:
              id: "product-123"
              name: "Gaming Laptop"
              price: 2499.99
              inStock: true
```

### 2.8.1.11 Aufgabe 10: Authentifizierung dokumentieren (Bonus, 10 min)

Füge JWT-Authentifizierung hinzu.

Security Scheme

```
components:
  securitySchemes:
    BearerAuth:
      type: http
```

```
scheme: bearer
bearerFormat: JWT
description: JWT Token aus /api/auth/login
```

Auf Endpoints anwenden

```
paths:
  /api/products:
    get:
      # Öffentlich - kein security
      security: []

    post:
      # Erfordert Auth
      security:
        - BearerAuth: []

  /api/products/{id}:
    put:
      security:
        - BearerAuth: []
    delete:
      security:
        - BearerAuth: []
```

Login-Endpoint

```
/api/auth/login:
  post:
    tags: [Auth]
    summary: JWT Token erhalten
    security: []
    requestBody:
      content:
        application/json:
          schema:
            type: object
            required: [email, password]
            properties:
              email:
                type: string
                format: email
              password:
                type: string
                format: password
    responses:
      '200':
        description: Login erfolgreich
        content:
          application/json:
            schema:
              type: object
```

```

    properties:
      token:
        type: string
      expiresIn:
        type: integer
  '401':
    description: Ungültige Anmeldedaten

```

### 2.8.1.12 Testen

OpenAPI Spec validieren

```

# Online Validator
# https://editor.swagger.io - Spec einfügen

# Oder mit CLI
npm install -g @apidevtools/swagger-cli
swagger-cli validate openapi.yaml

```

Swagger UI prüfen

1. Server starten: `dart run bin/server.dart`
2. Browser öffnen: `http://localhost:8080/docs`
3. Endpoints durchklicken
4. “Try it out” testen

### 2.8.1.13 Abgabe-Checkliste

- ☐ openapi.yaml mit korrekter Grundstruktur
- ☐ Product, ProductCreate, ProductUpdate Schemas
- ☐ Error und ValidationError Schemas
- ☐ GET /api/products mit Pagination-Parametern
- ☐ POST /api/products dokumentiert
- ☐ GET/PUT/DELETE /api/products/{id} dokumentiert
- ☐ GET /api/products/search dokumentiert
- ☐ Wiederverwendbare Parameters und Responses
- ☐ Tags für Gruppierung
- ☐ Swagger UI funktioniert
- ☐ Mindestens 2 Request/Response-Beispiele
- ☐ (Bonus) JWT-Authentifizierung dokumentiert

## 2.8.2 Lösung

### 2.8.2.1 openapi.yaml

```

openapi: 3.0.3
info:
  title: Product API
  description: |
    REST API für Produktverwaltung.

  ## Features

```

```

- CRUD-Operationen für Produkte
- Pagination und Filtering
- Volltextsuche
- Kategorien und Marken

## Authentifizierung
Schreibende Operationen (POST, PUT, DELETE) erfordern einen JWT-Token.
Token über POST /api/auth/login erhalten.
version: 1.0.0
contact:
  name: API Support
  email: api@example.com

servers:
- url: http://localhost:8080
  description: Development Server

tags:
- name: Products
  description: Produktverwaltung (CRUD)
- name: Search
  description: Produktsuche
- name: Meta
  description: Metadaten (Kategorien, Marken)
- name: Auth
  description: Authentifizierung

paths:
# =====
# Products
# =====
/api/products:
  get:
    tags: [Products]
    summary: Produkte auflisten
    description: Gibt eine paginierte Liste aller Produkte zurück.
    ↪ Unterstützt Filtering und Sortierung.
    operationId: listProducts
    parameters:
      - $ref: '#/components/parameters/PageParam'
      - $ref: '#/components/parameters/PerPageParam'
      - name: category
        in: query
        description: Nach Kategorie filtern
        schema:
          type: string
          enum: [electronics, clothing, home, sports, books]
      - name: brand
        in: query
        description: Nach Marke filtern
        schema:

```

```
    type: string
- name: inStock
  in: query
  description: Nur verfügbare Produkte
  schema:
    type: boolean
- name: minPrice
  in: query
  description: Mindestpreis
  schema:
    type: number
    minimum: 0
- name: maxPrice
  in: query
  description: Höchstpreis
  schema:
    type: number
- name: sort
  in: query
  description: Sortierfeld
  schema:
    type: string
    enum: [name, price, createdAt, rating, stock]
    default: createdAt
- name: order
  in: query
  description: Sortierrichtung
  schema:
    type: string
    enum: [asc, desc]
    default: desc
responses:
  '200':
    description: Liste der Produkte
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ProductList'
        examples:
          withProducts:
            summary: Mit Produkten
            value:
              data:
                - id: "product-1"
                  name: "Laptop Pro"
                  price: 1299.99
                  category: "electronics"
                  inStock: true
              meta:
                total: 50
                page: 1
```

```
        perPage: 20
        totalPages: 3
        hasNextPage: true
        hasPrevPage: false
      links:
        self: "/api/products?page=1&perPage=20"
        next: "/api/products?page=2&perPage=20"
    empty:
      summary: Keine Produkte
      value:
        data: []
        meta:
          total: 0
          page: 1
          perPage: 20
          totalPages: 0
          hasNextPage: false
          hasPrevPage: false
    '400':
      $ref: '#/components/responses/BadRequest'

post:
  tags: [Products]
  summary: Produkt erstellen
  description: Erstellt ein neues Produkt. Erfordert Authentifizierung.
  operationId: createProduct
  security:
    - BearerAuth: []
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ProductCreate'
        examples:
          laptop:
            summary: Laptop erstellen
            value:
              name: "Gaming Laptop X"
              description: "High-End Gaming Laptop mit RTX 4080"
              price: 2499.99
              category: "electronics"
              brand: "TechBrand"
              stock: 25
          shirt:
            summary: T-Shirt erstellen
            value:
              name: "Basic Cotton T-Shirt"
              description: "100% Baumwolle, verschiedene Farben"
              price: 24.99
              category: "clothing"
```

```
        brand: "FashionCo"
        stock: 100
responses:
  '201':
    description: Produkt erstellt
    headers:
      Location:
        description: URL des neuen Produkts
        schema:
          type: string
          example: /api/products/product-123
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Product'
  '400':
    $ref: '#/components/responses/ValidationError'
  '401':
    $ref: '#/components/responses/Unauthorized'
  '409':
    description: Produkt existiert bereits
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'

/api/products/{id}:
  parameters:
    - $ref: '#/components/parameters/ProductId'

  get:
    tags: [Products]
    summary: Produkt abrufen
    description: Gibt ein einzelnes Produkt anhand seiner ID zurück
    operationId: getProduct
    responses:
      '200':
        description: Produkt gefunden
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Product'
            example:
              id: "product-123"
              name: "Gaming Laptop X"
              description: "High-End Gaming Laptop"
              price: 2499.99
              category: "electronics"
              brand: "TechBrand"
              stock: 25
              inStock: true
```

```

        rating: 4.5
        createdAt: "2024-01-15T10:30:00Z"
      '404':
        $ref: '#/components/responses/NotFound'

put:
  tags: [Products]
  summary: Produkt aktualisieren
  description: Aktualisiert ein bestehendes Produkt. Nur geänderte Felder ↪
↪ senden.
  operationId: updateProduct
  security:
    - BearerAuth: []
  requestBody:
    required: true
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ProductUpdate'
        examples:
          priceUpdate:
            summary: Preis ändern
            value:
              price: 1999.99
          fullUpdate:
            summary: Mehrere Felder ändern
            value:
              name: "Gaming Laptop X Pro"
              price: 2799.99
              stock: 15
  responses:
    '200':
      description: Produkt aktualisiert
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Product'
    '400':
      $ref: '#/components/responses/ValidationError'
    '401':
      $ref: '#/components/responses/Unauthorized'
    '404':
      $ref: '#/components/responses/NotFound'

delete:
  tags: [Products]
  summary: Produkt löschen
  description: Löscht ein Produkt permanent
  operationId: deleteProduct
  security:
    - BearerAuth: []

```

```

responses:
  '204':
    description: Produkt gelöscht
  '401':
    $ref: '#/components/responses/Unauthorized'
  '404':
    $ref: '#/components/responses/NotFound'

# =====
# Search
# =====
/api/products/search:
  get:
    tags: [Search]
    summary: Produkte suchen
    description: |
      Volltextsuche in Produkten.
      Durchsucht: name, description, brand
    operationId: searchProducts
    parameters:
      - name: q
        in: query
        required: true
        description: Suchbegriff (mindestens 2 Zeichen)
        schema:
          type: string
          minLength: 2
        example: laptop
      - $ref: '#/components/parameters/PageParam'
      - $ref: '#/components/parameters/PerPageParam'
      - name: category
        in: query
        description: Zusätzlicher Kategorie-Filter
        schema:
          type: string
      - name: minPrice
        in: query
        schema:
          type: number
      - name: maxPrice
        in: query
        schema:
          type: number
    responses:
      '200':
        description: Suchergebnisse
        content:
          application/json:
            schema:
              allOf:
                - $ref: '#/components/schemas/ProductList'

```

```

        - type: object
        properties:
          meta:
            type: object
            properties:
              query:
                type: string
                example: "laptop"
      example:
        data:
          - id: "product-1"
            name: "Gaming Laptop"
            price: 1499.99
        meta:
          query: "laptop"
          total: 5
          page: 1
          perPage: 20
    '400':
      description: Suchbegriff fehlt oder zu kurz
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
          example:
            error: "Search query (q) is required"
            code: "MISSING_QUERY"

# =====
# Meta
# =====
/api/categories:
  get:
    tags: [Meta]
    summary: Kategorien abrufen
    description: Liste aller verfügbaren Produktkategorien
    operationId: getCategories
    responses:
      '200':
        description: Liste der Kategorien
        content:
          application/json:
            schema:
              type: object
              properties:
                data:
                  type: array
                  items:
                    type: string
                total:
                  type: integer

```

```

    example:
      data: ["books", "clothing", "electronics", "home", "sports"]
      total: 5

/api/brands:
  get:
    tags: [Meta]
    summary: Marken abrufen
    description: Liste aller verfügbaren Marken
    operationId: getBrands
    responses:
      '200':
        description: Liste der Marken
        content:
          application/json:
            schema:
              type: object
              properties:
                data:
                  type: array
                  items:
                    type: string
                total:
                  type: integer
            example:
              data: ["Adidas", "Apple", "Nike", "Samsung", "Sony"]
              total: 5

# =====
# Auth
# =====
/api/auth/login:
  post:
    tags: [Auth]
    summary: Einloggen
    description: JWT Token erhalten für authentifizierte Requests
    operationId: login
    security: []
    requestBody:
      required: true
    content:
      application/json:
        schema:
          type: object
          required: [email, password]
          properties:
            email:
              type: string
              format: email
              example: admin@example.com
            password:

```

```

        type: string
        format: password
        example: secret123
responses:
  '200':
    description: Login erfolgreich
    content:
      application/json:
        schema:
          type: object
          properties:
            token:
              type: string
              description: JWT Token
            expiresIn:
              type: integer
              description: Gültigkeit in Sekunden
          example:
            token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
            expiresIn: 3600
  '401':
    description: Ungültige Anmeldedaten
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'
        example:
          error: "Invalid credentials"
          code: "INVALID_CREDENTIALS"

# =====
# Components
# =====

components:
  securitySchemes:
    BearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT
      description: |
        JWT Token im Authorization Header.
        Format: `Authorization: Bearer <token>`

        Token über POST /api/auth/login erhalten.

parameters:
  ProductId:
    name: id
    in: path
    required: true
    description: Eindeutige Produkt-ID

```

```
    schema:
      type: string
    example: product-123

  PageParam:
    name: page
    in: query
    description: Seitennummer (ab 1)
    schema:
      type: integer
      default: 1
      minimum: 1
    example: 1

  PerPageParam:
    name: perPage
    in: query
    description: Elemente pro Seite
    schema:
      type: integer
      default: 20
      minimum: 1
      maximum: 100
    example: 20

  responses:
    NotFound:
      description: Ressource nicht gefunden
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
          example:
            error: "Product not found"
            code: "NOT_FOUND"

    BadRequest:
      description: Ungültige Anfrage
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
          example:
            error: "Invalid parameters"
            code: "BAD_REQUEST"

    ValidationError:
      description: Validierungsfehler
      content:
        application/json:
          schema:
```

```
    $ref: '#/components/schemas/ValidationError'
  example:
    error: "Validation failed"
    code: "VALIDATION_ERROR"
    details:
      - field: "price"
        message: "Price must be greater than 0"
      - field: "name"
        message: "Name is required"

Unauthorized:
  description: Nicht authentifiziert
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/Error'
      example:
        error: "Authentication required"
        code: "UNAUTHORIZED"

schemas:
  Product:
    type: object
    description: Vollständiges Produktobjekt
    required:
      - id
      - name
      - price
      - category
    properties:
      id:
        type: string
        description: Eindeutige Produkt-ID
        readOnly: true
        example: product-123
      name:
        type: string
        description: Produktname
        minLength: 2
        maxLength: 100
        example: "Gaming Laptop X"
      description:
        type: string
        description: Produktbeschreibung
        nullable: true
        example: "High-End Gaming Laptop mit RTX 4080"
      price:
        type: number
        format: double
        description: Preis in Euro
        minimum: 0.01
```

```
    example: 2499.99
  category:
    type: string
    description: Produktkategorie
    enum: [electronics, clothing, home, sports, books]
    example: electronics
  brand:
    type: string
    description: Markenname
    example: TechBrand
  stock:
    type: integer
    description: Lagerbestand
    minimum: 0
    default: 0
    example: 25
  inStock:
    type: boolean
    description: Ob auf Lager (berechnet aus stock > 0)
    readOnly: true
    example: true
  rating:
    type: number
    description: Durchschnittliche Bewertung (1-5)
    minimum: 1
    maximum: 5
    example: 4.5
  createdAt:
    type: string
    format: date-time
    description: Erstellungsdatum
    readOnly: true
    example: "2024-01-15T10:30:00Z"
```

```
ProductCreate:
  type: object
  description: Daten zum Erstellen eines Produkts
  required:
    - name
    - price
    - category
  properties:
    name:
      type: string
      minLength: 2
      maxLength: 100
      example: "Neues Produkt"
    description:
      type: string
      nullable: true
      example: "Produktbeschreibung"
```

```
    price:
      type: number
      minimum: 0.01
      example: 99.99
    category:
      type: string
      enum: [electronics, clothing, home, sports, books]
      example: electronics
    brand:
      type: string
      example: "BrandName"
    stock:
      type: integer
      minimum: 0
      default: 0
      example: 10

ProductUpdate:
  type: object
  description: Daten zum Aktualisieren eines Produkts. Alle Felder optional.
  properties:
    name:
      type: string
      minLength: 2
      maxLength: 100
    description:
      type: string
      nullable: true
    price:
      type: number
      minimum: 0.01
    category:
      type: string
      enum: [electronics, clothing, home, sports, books]
    brand:
      type: string
    stock:
      type: integer
      minimum: 0

ProductList:
  type: object
  description: Paginierte Liste von Produkten
  properties:
    data:
      type: array
      items:
        $ref: '#/components/schemas/Product'
    meta:
      $ref: '#/components/schemas/PaginationMeta'
  links:
```

```
$ref: '#/components/schemas/PaginationLinks'
```

PaginationMeta:

- type: **object**
- description: **Pagination-Metadaten**
- properties:
  - total:
    - type: **integer**
    - description: **Gesamtanzahl der Elemente**
    - example: **150**
  - page:
    - type: **integer**
    - description: **Aktuelle Seite**
    - example: **1**
  - perPage:
    - type: **integer**
    - description: **Elemente pro Seite**
    - example: **20**
  - totalPages:
    - type: **integer**
    - description: **Gesamtanzahl der Seiten**
    - example: **8**
  - hasNextPage:
    - type: **boolean**
    - description: **Gibt es eine nächste Seite?**
    - example: **true**
  - hasPrevPage:
    - type: **boolean**
    - description: **Gibt es eine vorherige Seite?**
    - example: **false**
  - filtered:
    - type: **boolean**
    - description: **Wurden Filter angewendet?**
    - example: **false**
  - appliedFilters:
    - type: **object**
    - description: **Angewendete Filter**
    - additionalProperties: **true**

PaginationLinks:

- type: **object**
- description: **HATEOAS Navigation-Links**
- properties:
  - self:
    - type: **string**
    - description: **Link zur aktuellen Seite**
    - example: **"/api/products?page=1&perPage=20"**
  - first:
    - type: **string**
    - description: **Link zur ersten Seite**
    - example: **"/api/products?page=1&perPage=20"**

```
last:
  type: string
  description: Link zur letzten Seite
  example: "/api/products?page=8&perPage=20"
prev:
  type: string
  nullable: true
  description: Link zur vorherigen Seite (null wenn erste Seite)
next:
  type: string
  nullable: true
  description: Link zur nächsten Seite (null wenn letzte Seite)
  example: "/api/products?page=2&perPage=20"

Error:
  type: object
  description: Fehlerobjekt
  required:
    - error
  properties:
    error:
      type: string
      description: Fehlermeldung
      example: "Resource not found"
    code:
      type: string
      description: Fehlercode
      example: "NOT_FOUND"

ValidationError:
  type: object
  description: Validierungsfehler mit Details
  properties:
    error:
      type: string
      example: "Validation failed"
    code:
      type: string
      example: "VALIDATION_ERROR"
    details:
      type: array
      items:
        type: object
        properties:
          field:
            type: string
            description: Feldname
            example: "email"
          message:
            type: string
            description: Fehlerbeschreibung
```

```
example: "Invalid email format"
```

### 2.8.2.2 Dart Server mit Swagger UI

```
import 'dart:convert';
import 'dart:io';
import 'dart:math';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as shelf_io;
import 'package:shelf_router/shelf_router.dart';

// =====
// Models (gekürzt)
// =====

class Product {
  final String id;
  final String name;
  final String? description;
  final double price;
  final String category;
  final String brand;
  final int stock;
  final double rating;
  final DateTime createdAt;

  Product({
    required this.id,
    required this.name,
    this.description,
    required this.price,
    required this.category,
    required this.brand,
    required this.stock,
    required this.rating,
    required this.createdAt,
  });

  Map<String, dynamic> toJson() => {
    'id': id,
    'name': name,
    if (description != null) 'description': description,
    'price': price,
    'category': category,
    'brand': brand,
    'stock': stock,
    'inStock': stock > 0,
    'rating': rating,
    'createdAt': createdAt.toIso8601String(),
  };
}
```

```

}

// =====
// Storage
// =====

final _products = <String, Product>{};

void _seedData() {
  final random = Random(42);
  final categories = ['electronics', 'clothing', 'home', 'sports', 'books'];
  final brands = ['Apple', 'Samsung', 'Nike', 'Adidas', 'Sony'];

  for (var i = 1; i <= 50; i++) {
    _products['product-$i'] = Product(
      id: 'product-$i',
      name: '${brands[i % brands.length]} Product $i',
      description: 'Description for product $i',
      price: (random.nextDouble() * 1000 + 10).roundToDouble(),
      category: categories[i % categories.length],
      brand: brands[i % brands.length],
      stock: random.nextInt(100),
      rating: 1 + random.nextDouble() * 4,
      createdAt: DateTime.now().subtract(Duration(days: i)),
    );
  }
}

// =====
// Helpers
// =====

Response jsonResponse(Object? data, {int statusCode = 200}) {
  return Response(
    statusCode,
    body: jsonEncode(data),
    headers: {'content-type': 'application/json'},
  );
}

// =====
// API Handlers (vereinfacht)
// =====

Response listProducts(Request request) {
  final params = request.url.queryParameters;
  final page = (int.tryParse(params['page'] ?? '1') ?? 1).clamp(1, 1000);
  final perPage = (int.tryParse(params['perPage'] ?? '20') ?? 20).clamp(1, 100);

  var products = _products.values.toList();

```

```

// Filter
final category = params['category'];
if (category != null) {
    products = products.where((p) => p.category == category).toList();
}

// Sortierung
final sort = params['sort'] ?? 'createdAt';
final order = params['order'] ?? 'desc';

products = [...products]..sort((a, b) {
    int cmp = switch (sort) {
        'name' => a.name.compareTo(b.name),
        'price' => a.price.compareTo(b.price),
        _ => a.createdAt.compareTo(b.createdAt),
    };
    return order == 'desc' ? -cmp : cmp;
});

// Pagination
final total = products.length;
final totalPages = total > 0 ? (total / perPage).ceil() : 1;
final offset = (page - 1) * perPage;
final pagedProducts = products.skip(offset).take(perPage).toList();

return jsonResponse({
    'data': pagedProducts.map((p) => p.toJson()).toList(),
    'meta': {
        'total': total,
        'page': page,
        'perPage': perPage,
        'totalPages': totalPages,
        'hasNextPage': page < totalPages,
        'hasPrevPage': page > 1,
    },
    'links': {
        'self': '/api/products?page=$page&perPage=$perPage',
        'first': '/api/products?page=1&perPage=$perPage',
        'last': '/api/products?page=$totalPages&perPage=$perPage',
        if (page > 1) 'prev': '/api/products?page=${page - 1}&perPage=$perPage',
        if (page < totalPages) 'next': '/api/products?page=${page +
↵ 1}&perPage=$perPage',
    },
});
}

Response getProduct(Request request, String id) {
    final product = _products[id];
    if (product == null) {
        return jsonResponse({'error': 'Product not found', 'code': 'NOT_FOUND'},
↵ statusCode: 404);

```

```

    }
    return jsonResponse(product.toJson());
}

Response getCategories(Request request) {
    final categories = _products.values.map((p) =>
        ↪ p.category).toSet().toList()..sort();
    ↪ return jsonResponse({'data': categories, 'total': categories.length});
}

Response getBrands(Request request) {
    final brands = _products.values.map((p) => p.brand).toSet().toList()..sort();
    return jsonResponse({'data': brands, 'total': brands.length});
}

// =====
// OpenAPI & Swagger UI
// =====

Future<Response> serveOpenApi(Request request) async {
    try {
        final file = File('openapi.yaml');
        if (!await file.exists()) {
            return Response.notFound('openapi.yaml not found');
        }
        final content = await file.readAsString();
        return Response.ok(content, headers: {
            'content-type': 'application/x-yaml',
            'access-control-allow-origin': '*',
        });
    } catch (e) {
        return Response.internalServerError(body: 'Error reading openapi.yaml');
    }
}

Response serveSwaggerUI(Request request) {
    const html = '''
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Product API - Documentation</title>
    <link rel="stylesheet" type="text/css"
    ↪ href="https://unpkg.com/swagger-ui-dist@5/swagger-ui.css">
    <style>
        html { box-sizing: border-box; overflow-y: scroll; }
        *, *:before, *:after { box-sizing: inherit; }
        body { margin: 0; background: #fafafa; }
        .swagger-ui .topbar { display: none; }
    </style>

```

```

</head>
<body>
  <div id="swagger-ui"></div>
  <script
    ↪ src="https://unpkg.com/swagger-ui-dist@5/swagger-ui-bundle.js"></script>
  <script
    ↪ src="https://unpkg.com/swagger-ui-dist@5/swagger-ui-standalone-preset.js"></script>
  <script>
    window.onload = function() {
      window.ui = SwaggerUIBundle({
        url: "/openapi.yaml",
        dom_id: '#swagger-ui',
        deepLinking: true,
        presets: [
          SwaggerUIBundle.presets.apis,
          SwaggerUIStandalonePreset
        ],
        plugins: [
          SwaggerUIBundle.plugins.DownloadUrl
        ],
        layout: "StandaloneLayout",
        defaultModelsExpandDepth: 1,
        defaultModelExpandDepth: 1,
        docExpansion: "list",
        filter: true,
        showExtensions: true,
        showCommonExtensions: true
      });
    };
  </script>
</body>
</html>
''';

    return Response.ok(html, headers: {'content-type': 'text/html;
    ↪ charset=utf-8'});
  }

  // =====
  // Main
  // =====

  void main() async {
    _seedData();

    final router = Router();

    // API Endpoints
    router.get('/api/products', listProducts);
    router.get('/api/products/<id>', getProduct);
    router.get('/api/categories', getCategories);

```

```
router.get('/api/brands', getBrands);

// OpenAPI Spec
router.get('/openapi.yaml', serveOpenApi);
router.get('/openapi.json', serveOpenApi); // Auch als JSON-Pfad

// Swagger UI
router.get('/docs', serveSwaggerUI);
router.get('/docs/', serveSwaggerUI);

// CORS für Swagger UI
final handler = Pipeline()
    .addMiddleware(logRequests())
    .addMiddleware((handler) {
        return (request) async {
            if (request.method == 'OPTIONS') {
                return Response.ok('', headers: {
                    'access-control-allow-origin': '*',
                    'access-control-allow-methods': 'GET, POST, PUT, DELETE, OPTIONS',
                    'access-control-allow-headers': 'Content-Type, Authorization',
                });
            }
            final response = await handler(request);
            return response.change(headers: {
                ...response.headers,
                'access-control-allow-origin': '*',
            });
        };
    })
    .addHandler(router.call);

await shelf_io.serve(handler, 'localhost', 8080);

print('');
print(
    print(
        print(
            Product API Server Running
        )
    )
);
print(
    print(
        print(
            API: http://localhost:8080/api
        )
        print(
            Docs: http://localhost:8080/docs
        )
        print(
            OpenAPI: http://localhost:8080/openapi.yaml
        )
    )
);
print('');
print('');
```

### 2.8.2.3 Test-Befehle

```
# Server starten
dart run bin/server.dart

# OpenAPI Spec abrufen
```

```

curl http://localhost:8080/openapi.yaml

# Swagger UI öffnen
open http://localhost:8080/docs

# API testen
curl http://localhost:8080/api/products
curl http://localhost:8080/api/products/product-1
curl http://localhost:8080/api/categories
curl http://localhost:8080/api/brands

# Mit Filtern
curl 'http://localhost:8080/api/products?category=electronics&sort=price&order=asc'

```

#### 2.8.2.4 Projektstruktur

```

project/
+-- bin/
|   +-- server.dart      # Server-Code
+-- openapi.yaml         # OpenAPI Spezifikation
+-- pubspec.yaml

pubspec.yaml
name: product_api
description: Product API with OpenAPI documentation

environment:
  sdk: ^3.0.0

dependencies:
  shelf: ^1.4.1
  shelf_router: ^1.1.4

```

#### 2.8.2.5 Wichtige Patterns

Wiederverwendbare Parameter

```

# Definition
components:
  parameters:
    PageParam:
      name: page
      in: query
      schema:
        type: integer
        default: 1

# Verwendung
paths:
  /api/products:

```

```
get:
  parameters:
    - $ref: '#/components/parameters/PageParam'
```

Wiederverwendbare Responses

```
# Definition
components:
  responses:
    NotFound:
      description: Not found
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'

# Verwendung
responses:
  '404':
    $ref: '#/components/responses/NotFound'
```

Schema-Vererbung

```
ProductList:
  allOf:
    - $ref: '#/components/schemas/PaginatedResponse'
    - type: object
      properties:
        data:
          type: array
          items:
            $ref: '#/components/schemas/Product'
```

## 2.8.3 Ressourcen

### 2.8.3.1 Offizielle Dokumentation

- OpenAPI Specification
- Swagger Documentation
- Swagger Editor - Online-Editor
- OpenAPI Generator

### 2.8.3.2 Tools

Editoren

- Swagger Editor - Browser-basiert
- Stoplight Studio - Desktop App
- VS Code OpenAPI Extension

UI-Generatoren

- Swagger UI - Interaktive Dokumentation
- ReDoc - Lesbare Dokumentation

- RapiDoc - Moderne Alternative

Validierung

```
# swagger-cli installieren
npm install -g @apidevtools/swagger-cli

# Spec validieren
swagger-cli validate openapi.yaml
```

### 2.8.3.3 Cheat Sheet: Grundstruktur

```
openapi: 3.0.3
info:
  title: API Name
  description: API Beschreibung
  version: 1.0.0
  contact:
    name: Support
    email: api@example.com

servers:
  - url: http://localhost:8080
    description: Development

tags:
  - name: Resources
    description: Resource endpoints

paths:
  /api/resource:
    get:
      # ...

components:
  schemas:
    # ...
  parameters:
    # ...
  responses:
    # ...
  securitySchemes:
    # ...
```

### 2.8.3.4 Cheat Sheet: Parameter

```
parameters:
  # Query Parameter
  - name: page
    in: query
    description: Seitennummer
```

```
    required: false
    schema:
      type: integer
      default: 1
      minimum: 1

# Path Parameter
- name: id
  in: path
  description: Ressourcen-ID
  required: true
  schema:
    type: string

# Header Parameter
- name: X-API-Key
  in: header
  required: true
  schema:
    type: string

# Cookie Parameter
- name: session
  in: cookie
  schema:
    type: string
```

### 2.8.3.5 Cheat Sheet: Request Body

```
requestBody:
  required: true
  description: Daten zum Erstellen
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/CreateRequest'
      examples:
        example1:
          summary: Beispiel 1
          value:
            name: "Test"
            price: 99.99
        example2:
          summary: Beispiel 2
          value:
            name: "Test 2"
```

### 2.8.3.6 Cheat Sheet: Responses

```
responses:
  '200':
    description: Erfolg
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Resource'

  '201':
    description: Erstellt
    headers:
      Location:
        description: URL der neuen Ressource
        schema:
          type: string

  '204':
    description: Kein Inhalt

  '400':
    description: Ungültige Anfrage
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'

  '401':
    description: Nicht authentifiziert

  '403':
    description: Keine Berechtigung

  '404':
    description: Nicht gefunden

  '409':
    description: Konflikt

  '422':
    description: Validierungsfehler
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ValidationError'

  '500':
    description: Serverfehler
```

### 2.8.3.7 Cheat Sheet: Schemas

```
components:
  schemas:
    # Einfaches Schema
    User:
      type: object
      required:
        - id
        - email
      properties:
        id:
          type: string
          readOnly: true
        email:
          type: string
          format: email
        name:
          type: string
          minLength: 2
          maxLength: 100
        age:
          type: integer
          minimum: 0
          maximum: 150
        role:
          type: string
          enum: [user, admin, moderator]
          default: user
        tags:
          type: array
          items:
            type: string
      metadata:
        type: object
        additionalProperties: true
      createdAt:
        type: string
        format: date-time
        readOnly: true

    # Nullable
    NullableField:
      type: string
      nullable: true

    # OneOf (einer von)
    Response:
      oneOf:
        - $ref: '#/components/schemas/Success'
        - $ref: '#/components/schemas/Error'
```

```
# AllOf (Kombination)
DetailedUser:
  allOf:
    - $ref: '#/components/schemas/User'
    - type: object
      properties:
        address:
          $ref: '#/components/schemas/Address'
```

### 2.8.3.8 Cheat Sheet: Authentifizierung

```
components:
  securitySchemes:
    # API Key
    ApiKeyAuth:
      type: apiKey
      in: header
      name: X-API-Key

    # Bearer Token
    BearerAuth:
      type: http
      scheme: bearer
      bearerFormat: JWT

    # Basic Auth
    BasicAuth:
      type: http
      scheme: basic

    # OAuth 2.0
    OAuth2:
      type: oauth2
      flows:
        authorizationCode:
          authorizationUrl: https://auth.example.com/authorize
          tokenUrl: https://auth.example.com/token
          scopes:
            read: Read access
            write: Write access

    # Global anwenden
  security:
    - BearerAuth: []

# Pro Endpoint
paths:
  /public:
    get:
```

```

    security: [] # Keine Auth

/private:
  get:
    security:
      - BearerAuth: []

```

### 2.8.3.9 Cheat Sheet: Wiederverwendung

```

components:
  # Parameter
  parameters:
    PageParam:
      name: page
      in: query
      schema:
        type: integer
        default: 1

  # Responses
  responses:
    NotFound:
      description: Not found
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'

# Verwendung mit $ref
paths:
  /items:
    get:
      parameters:
        - $ref: '#/components/parameters/PageParam'
      responses:
        '404':
          $ref: '#/components/responses/NotFound'

```

### 2.8.3.10 Cheat Sheet: Swagger UI in Dart

```

// Inline Swagger UI
Response serveSwaggerUI(Request request) {
  return Response.ok('''
<!DOCTYPE html>
<html>
<head>
  <title>API Docs</title>
  <link rel="stylesheet"
    ↪ href="https://unpkg.com/swagger-ui-dist@5/swagger-ui.css">

```

↪

```

</head>
<body>
  <div id="swagger-ui"></div>
  <script
    ↪ src="https://unpkg.com/swagger-ui-dist@5/swagger-ui-bundle.js"></script>
  <script>
    SwaggerUIBundle({
      url: '/openapi.yaml',
      dom_id: '#swagger-ui',
    });
  </script>
</body>
</html>
'', headers: {'content-type': 'text/html'}));
}

// OpenAPI Spec servieren
Future<Response> serveOpenApi(Request request) async {
  final content = await File('openapi.yaml').readAsString();
  return Response.ok(content, headers: {
    'content-type': 'application/x-yaml',
  });
}

// Router
router.get('/docs', serveSwaggerUI);
router.get('/openapi.yaml', serveOpenApi);

```

### 2.8.3.11 Datentypen

OpenAPI	Dart	Format
string	String	-
string	String	date (2024-01-15)
string	String	date-time (ISO 8601)
string	String	email
string	String	uri
string	String	uuid
string	String	password
integer	int	int32
integer	int	int64
number	double	float
number	double	double
boolean	bool	-
array	List	-
object	Map / Class	-

### 2.8.3.12 Best Practices

1. **Konsistente Benennung:** camelCase für Felder
2. **Aussagekräftige Beschreibungen:** Jedes Feld dokumentieren

3. **Beispiele überall:** examples für Requests und Responses
4. **Fehler dokumentieren:** Alle möglichen Fehlercodes
5. **Wiederverwendung:** Components nutzen
6. **Tags:** Endpoints logisch gruppieren
7. **Versionierung:** Im Info-Block und URL
8. **Validierung:** Schema mit Constraints

### 2.8.3.13 Validierung der Spec

```
# Online
# https://editor.swagger.io

# CLI
npm install -g @apidevtools/swagger-cli
swagger-cli validate openapi.yaml

# Spectral (Linting)
npm install -g @stoplight/spectral-cli
spectral lint openapi.yaml
```

### 2.8.3.14 Code-Generierung

```
# Dart Client generieren
npx @openapitools/openapi-generator-cli generate \
  -i openapi.yaml \
  -g dart \
  -o ./generated/client

# Server Stub generieren
npx @openapitools/openapi-generator-cli generate \
  -i openapi.yaml \
  -g dart-shelf \
  -o ./generated/server
```

### 2.8.3.15 Alternative UIs

ReDoc

```
<!DOCTYPE html>
<html>
<head>
  <title>API Docs</title>
  <link href=
    ↪ "https://fonts.googleapis.com/css?family=Montserrat:300,400,700|Roboto:300,400,700"
    ↪
    ↪ rel="stylesheet">
  <style>body { margin: 0; padding: 0; }</style>
</head>
<body>
  <redoc spec-url='/openapi.yaml'></redoc>
  <script
    ↪ src="https://cdn.redoc.ly/redoc/latest/bundles/redoc.standalone.js"
    ↪
  ></script>
```

```
</body>
</html>
```

RapiDoc

```
<!DOCTYPE html>
<html>
<head>
  <title>API Docs</title>
  <script type="module"
    ↪ src="https://unpkg.com/rapidoc/dist/rapidoc-min.js"></script>
</head>
<body>
  <rapidoc spec-url="/openapi.yaml" theme="dark"></rapidoc>
</body>
</html>
```

# Chapter 3

## Block 7: Datenbanken

SQL, PostgreSQL, Repository Pattern, MongoDB, Redis und Caching.

### 3.1 Einheit 7.1: SQL Grundlagen & PostgreSQL

#### 3.1.0.1 Lernziele

Nach dieser Einheit kannst du: - Die Grundlagen relationaler Datenbanken verstehen - SQL-Syntax für CRUD-Operationen anwenden - Tabellen erstellen und verknüpfen - Einfache Abfragen mit WHERE, ORDER BY und LIMIT schreiben

#### 3.1.0.2 Relationale Datenbanken

Was ist eine relationale Datenbank?

Eine **relationale Datenbank** speichert Daten in **Tabellen** (Relations), die aus **Zeilen** (Datensätze) und **Spalten** (Attribute) bestehen.

```
+-----+
|                users                |
+-----+-----+-----+
| id | name          | email                |
+-----+-----+-----+
| 1  | Max Müller   | max@example.com      |
| 2  | Anna Schmidt | anna@example.com     |
| 3  | Tom Weber    | tom@example.com      |
+-----+-----+-----+
```

Wichtige Begriffe

Begriff	Beschreibung
<b>Tabelle</b>	Sammlung von zusammengehörigen Daten
<b>Zeile (Row)</b>	Ein einzelner Datensatz
<b>Spalte (Column)</b>	Ein Attribut/Feld
<b>Primary Key</b>	Eindeutiger Identifikator für jede Zeile
<b>Foreign Key</b>	Verweis auf Primary Key einer anderen Tabelle
<b>Schema</b>	Struktur der Datenbank (Tabellen, Spalten, Typen)

Populäre relationale Datenbanken

Datenbank	Eigenschaften
<b>PostgreSQL</b>	Open Source, feature-reich, sehr stabil
<b>MySQL/MariaDB</b>	Weit verbreitet, performant

Datenbank	Eigenschaften
<b>SQLite</b>	Eingebettet, dateibasiert, kein Server nötig
<b>SQL Server</b>	Microsoft, Enterprise-Fokus

### 3.1.0.3 SQL Syntax Grundlagen

**SQL** (Structured Query Language) ist die Standardsprache für relationale Datenbanken.

Datentypen

SQL-Typ	Beschreibung	Beispiel
INTEGER	Ganzzahlen	42
BIGINT	Große Ganzzahlen	9223372036854775807
DECIMAL(p,s)	Präzise Dezimalzahlen	DECIMAL(10,2) für Geld
REAL/FLOAT	Fließkommazahlen	3.14159
VARCHAR(n)	Text mit max. Länge	VARCHAR(255)
TEXT	Text unbegrenzt	Lange Beschreibungen
BOOLEAN	Wahrheitswert	TRUE, FALSE
DATE	Datum	'2024-01-15'
TIMESTAMP	Datum + Zeit	'2024-01-15 10:30:00'
UUID	Universally Unique ID	'550e8400-e29b-...'

### 3.1.0.4 Tabellen erstellen (CREATE)

Einfache Tabelle

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Mit Constraints

```
CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  description TEXT,
  price DECIMAL(10, 2) NOT NULL CHECK (price > 0),
  stock INTEGER NOT NULL DEFAULT 0 CHECK (stock >= 0),
  category_id INTEGER REFERENCES categories(id),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Constraint-Typen

Constraint	Beschreibung
PRIMARY KEY	Eindeutiger Identifikator
NOT NULL	Wert muss vorhanden sein

Constraint	Beschreibung
UNIQUE	Wert muss einzigartig sein
CHECK	Bedingung muss erfüllt sein
DEFAULT	Standardwert wenn nicht angegeben
REFERENCES	Foreign Key zu anderer Tabelle

### 3.1.0.5 Daten einfügen (INSERT)

Einzelner Datensatz

```
INSERT INTO users (name, email)
VALUES ('Max Müller', 'max@example.com');
```

Mehrere Datensätze

```
INSERT INTO users (name, email)
VALUES
    ('Anna Schmidt', 'anna@example.com'),
    ('Tom Weber', 'tom@example.com'),
    ('Lisa Fischer', 'lisa@example.com');
```

Mit Rückgabe der ID

```
INSERT INTO users (name, email)
VALUES ('Max Müller', 'max@example.com')
RETURNING id;

-- Oder alle Felder
INSERT INTO users (name, email)
VALUES ('Max Müller', 'max@example.com')
RETURNING *;
```

### 3.1.0.6 Daten abfragen (SELECT)

Alle Spalten

```
SELECT * FROM users;
```

Bestimmte Spalten

```
SELECT name, email FROM users;
```

Mit Alias

```
SELECT
    name AS username,
    email AS mail_address
FROM users;
```

Mit Bedingungen (WHERE)

```
-- Exakter Vergleich
SELECT * FROM users WHERE id = 1;
```

```
-- Textvergleich
SELECT * FROM users WHERE name = 'Max Müller';

-- Größer/Kleiner
SELECT * FROM products WHERE price > 100;
SELECT * FROM products WHERE price BETWEEN 50 AND 150;

-- IN (Liste von Werten)
SELECT * FROM products WHERE category IN ('electronics', 'books');

-- LIKE (Textmuster)
SELECT * FROM users WHERE email LIKE '%@example.com';
SELECT * FROM users WHERE name LIKE 'Max%'; -- Beginnt mit Max

-- NULL-Prüfung
SELECT * FROM products WHERE description IS NULL;
SELECT * FROM products WHERE description IS NOT NULL;

-- Logische Operatoren
SELECT * FROM products
WHERE price > 100 AND stock > 0;

SELECT * FROM users
WHERE name = 'Max' OR name = 'Anna';
```

### 3.1.0.7 Sortierung und Limits

#### ORDER BY

```
-- Aufsteigend (Standard)
SELECT * FROM products ORDER BY price ASC;

-- Absteigend
SELECT * FROM products ORDER BY price DESC;

-- Mehrere Spalten
SELECT * FROM products
ORDER BY category ASC, price DESC;

-- NULL-Handling
SELECT * FROM products
ORDER BY description NULLS LAST;
```

#### LIMIT und OFFSET

```
-- Erste 10 Datensätze
SELECT * FROM products LIMIT 10;

-- Seite 2 (Datensätze 11-20)
SELECT * FROM products LIMIT 10 OFFSET 10;

-- Top 5 teuerste Produkte
```

```
SELECT * FROM products
ORDER BY price DESC
LIMIT 5;
```

### 3.1.0.8 Daten aktualisieren (UPDATE)

Einzelnes Feld

```
UPDATE users
SET email = 'newemail@example.com'
WHERE id = 1;
```

Mehrere Felder

```
UPDATE products
SET
    price = 29.99,
    stock = stock + 10,
    updated_at = CURRENT_TIMESTAMP
WHERE id = 42;
```

Mit Bedingung

```
-- Alle Produkte in Kategorie um 10% reduzieren
UPDATE products
SET price = price * 0.9
WHERE category = 'electronics';
```

Mit RETURNING

```
UPDATE users
SET name = 'Max Mustermann'
WHERE id = 1
RETURNING *;
```

### 3.1.0.9 Daten löschen (DELETE)

Einzelner Datensatz

```
DELETE FROM users WHERE id = 1;
```

Mit Bedingung

```
-- Alle inaktiven User löschen
DELETE FROM users
WHERE last_login < '2023-01-01';
```

Alle Datensätze (Vorsicht!)

```
DELETE FROM users; -- Löscht ALLE Zeilen

-- Schneller für große Tabellen:
TRUNCATE TABLE users;
```

Mit RETURNING

```
DELETE FROM users
WHERE id = 1
RETURNING *;
```

### 3.1.0.10 Aggregatfunktionen

```
-- Anzahl
SELECT COUNT(*) FROM users;
SELECT COUNT(*) FROM products WHERE stock > 0;

-- Summe
SELECT SUM(price) FROM products;

-- Durchschnitt
SELECT AVG(price) FROM products;

-- Minimum/Maximum
SELECT MIN(price), MAX(price) FROM products;

-- Kombiniert
SELECT
    COUNT(*) AS total_products,
    SUM(price * stock) AS total_value,
    AVG(price) AS avg_price,
    MIN(price) AS cheapest,
    MAX(price) AS most_expensive
FROM products;
```

### 3.1.0.11 Gruppierung (GROUP BY)

```
-- Anzahl Produkte pro Kategorie
SELECT category, COUNT(*) AS count
FROM products
GROUP BY category;

-- Durchschnittspreis pro Kategorie
SELECT category, AVG(price) AS avg_price
FROM products
GROUP BY category
ORDER BY avg_price DESC;

-- Mit HAVING (Filter für Gruppen)
SELECT category, COUNT(*) AS count
FROM products
GROUP BY category
HAVING COUNT(*) > 5;
```

### 3.1.0.12 Einfache JOINS

Tabellen verknüpfen

```
-- Kategorien-Tabelle
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);

-- Produkte mit category_id (Foreign Key)
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    category_id INTEGER REFERENCES categories(id)
);
```

INNER JOIN

```
-- Produkte mit Kategoriennamen
SELECT
    p.id,
    p.name AS product_name,
    c.name AS category_name
FROM products p
INNER JOIN categories c ON p.category_id = c.id;
```

LEFT JOIN

```
-- Alle Produkte, auch ohne Kategorie
SELECT
    p.id,
    p.name AS product_name,
    c.name AS category_name
FROM products p
LEFT JOIN categories c ON p.category_id = c.id;
```

JOIN-Typen Übersicht

JOIN-Typ	Beschreibung
INNER JOIN	Nur übereinstimmende Zeilen
LEFT JOIN	Alle linken + übereinstimmende rechte
RIGHT JOIN	Alle rechten + übereinstimmende linke
FULL JOIN	Alle Zeilen aus beiden Tabellen

### 3.1.0.13 Tabellen ändern (ALTER)

Spalte hinzufügen

```
ALTER TABLE users
ADD COLUMN phone VARCHAR(20);
```

Spalte ändern

```
ALTER TABLE users
ALTER COLUMN name TYPE VARCHAR(200);
```

Spalte löschen

```
ALTER TABLE users
DROP COLUMN phone;
```

Constraint hinzufügen

```
ALTER TABLE products
ADD CONSTRAINT positive_price CHECK (price > 0);
```

### 3.1.0.14 Tabelle löschen (DROP)

```
-- Tabelle löschen (Fehler wenn nicht existiert)
DROP TABLE users;

-- Sicher löschen
DROP TABLE IF EXISTS users;

-- Mit abhängigen Objekten
DROP TABLE users CASCADE;
```

### 3.1.0.15 Best Practices

#### 1. Namenskonventionen

```
-- Tabellen: snake_case, Plural
CREATE TABLE user_profiles (...);
CREATE TABLE order_items (...);

-- Spalten: snake_case
CREATE TABLE users (
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    created_at TIMESTAMP
);
```

#### 2. Immer Primary Keys verwenden

```
-- Mit SERIAL (auto-increment)
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    ...
);

-- Mit UUID
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    ...
);
```

## 3. Foreign Keys definieren

```
-- Explizite Referenz
CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id),
    ...
);
```

## 4. Indizes für häufige Abfragen

```
-- Index auf häufig gesuchte Spalte
CREATE INDEX idx_users_email ON users(email);

-- Zusammengesetzter Index
CREATE INDEX idx_products_category_price
ON products(category, price);
```

## 3.1.0.16 Zusammenfassung

Operation	SQL-Befehl
Tabelle erstellen	CREATE TABLE
Daten einfügen	INSERT INTO ... VALUES
Daten abfragen	SELECT ... FROM ... WHERE
Daten aktualisieren	UPDATE ... SET ... WHERE
Daten löschen	DELETE FROM ... WHERE
Tabelle ändern	ALTER TABLE
Tabelle löschen	DROP TABLE

## 3.1.0.17 Nächste Schritte

In der nächsten Einheit lernst du, wie du **PostgreSQL mit Dart** verwendest: Verbindung herstellen, Queries ausführen und Ergebnisse verarbeiten.

## 3.1.1 Übung

## 3.1.1.1 Ziel

Übe die grundlegenden SQL-Befehle mit einer kleinen Produktdatenbank.

## 3.1.1.2 Vorbereitung

Option A: PostgreSQL lokal

```
# PostgreSQL installieren (Arch Linux)
sudo pacman -S postgresql

# Datenbank initialisieren
sudo -u postgres initdb -D /var/lib/postgres/data

# Service starten
sudo systemctl start postgresql
```

```
# Benutzer und Datenbank erstellen
sudo -u postgres createuser -s $USER
createdb shop_db
```

Option B: Docker

```
# PostgreSQL Container starten
docker run --name postgres-shop \
  -e POSTGRES_PASSWORD=secret \
  -e POSTGRES_DB=shop_db \
  -p 5432:5432 \
  -d postgres:16

# Verbinden
docker exec -it postgres-shop psql -U postgres -d shop_db
```

Option C: Online SQL Editor

- SQLite Online
- DB Fiddle (PostgreSQL wählen)

### 3.1.1.3 Aufgabe 1: Tabellen erstellen (15 min)

Erstelle die folgenden Tabellen für einen Online-Shop.

categories

Spalte	Typ	Constraints
id	SERIAL	PRIMARY KEY
name	VARCHAR(50)	NOT NULL, UNIQUE
description	TEXT	-

products

Spalte	Typ	Constraints
id	SERIAL	PRIMARY KEY
name	VARCHAR(100)	NOT NULL
description	TEXT	-
price	DECIMAL(10,2)	NOT NULL, CHECK > 0
stock	INTEGER	NOT NULL, DEFAULT 0, CHECK >= 0
category_id	INTEGER	REFERENCES categories(id)
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP

customers

Spalte	Typ	Constraints
id	SERIAL	PRIMARY KEY
name	VARCHAR(100)	NOT NULL
email	VARCHAR(255)	NOT NULL, UNIQUE

Spalte	Typ	Constraints
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP

### 3.1.1.4 Aufgabe 2: Daten einfügen (10 min)

Kategorien einfügen

Füge mindestens 4 Kategorien ein: - Electronics - Clothing - Books - Home & Garden

Produkte einfügen

Füge mindestens 10 Produkte ein, verteilt auf die Kategorien:

```
-- Beispiel
INSERT INTO products (name, description, price, stock, category_id)
VALUES ('Laptop Pro 15', 'High-end laptop with 16GB RAM', 1299.99, 25, 1);
```

Achte auf: - Verschiedene Preisklassen (10€ - 2000€) - Verschiedene Lagerbestände (0 - 100) - Mindestens ein Produkt ohne Kategorie (NULL)

Kunden einfügen

Füge 5 Kunden ein.

### 3.1.1.5 Aufgabe 3: Einfache Abfragen (10 min)

Schreibe SQL-Queries für:

1. **Alle Produkte anzeigen**
  - Sortiert nach Name
2. **Produkte über 100€**
  - Nur Name und Preis anzeigen
3. **Produkte auf Lager**
  - Nur Produkte mit stock > 0
4. **Günstigstes und teuerstes Produkt**
  - Verwende MIN/MAX
5. **Anzahl Produkte pro Kategorie**
  - Mit GROUP BY

### 3.1.1.6 Aufgabe 4: Komplexere Abfragen (15 min)

4.1 Filter kombinieren

Finde alle Produkte die: - Zur Kategorie "Electronics" gehören - Mehr als 50€ kosten - Auf Lager sind (stock > 0)

4.2 Textsuche

Finde alle Produkte deren Name oder Beschreibung "Pro" enthält.

4.3 Preisbereiche

Kategorisiere Produkte nach Preis: - Günstig: < 50€ - Mittel: 50€ - 200€ - Premium: > 200€

Hint: Verwende CASE WHEN

```
SELECT
    name,
    price,
    CASE
        WHEN price < 50 THEN 'Günstig'
        WHEN price <= 200 THEN 'Mittel'
        ELSE 'Premium'
    END AS price_category
FROM products;
```

#### 4.4 Top 5 teuerste Produkte

Mit Kategorienamen (JOIN erforderlich).

### 3.1.1.7 Aufgabe 5: JOINS (15 min)

#### 5.1 Produkte mit Kategorienamen

Zeige alle Produkte mit ihrem Kategorienamen an. Produkte ohne Kategorie sollen auch erscheinen (LEFT JOIN).

#### 5.2 Kategorien mit Produktanzahl

Zeige für jede Kategorie: - Kategorienamen - Anzahl Produkte - Durchschnittspreis - Gesamtwert (Summe: price \* stock)

#### 5.3 Leere Kategorien finden

Finde Kategorien, die keine Produkte haben.

Hint: LEFT JOIN + WHERE ... IS NULL

### 3.1.1.8 Aufgabe 6: UPDATE und DELETE (10 min)

#### 6.1 Preis erhöhen

Erhöhe den Preis aller Electronics-Produkte um 10%.

```
UPDATE products
SET price = price * 1.10
WHERE category_id = (SELECT id FROM categories WHERE name = 'Electronics');
```

#### 6.2 Stock auffüllen

Setze den Stock aller Produkte mit Stock = 0 auf 10.

#### 6.3 Produkt umbenennen

Ändere den Namen eines Produkts deiner Wahl.

#### 6.4 Produkt löschen

Lösche ein Produkt (aber behalte die Query als Kommentar).

### 3.1.1.9 Aufgabe 7: Views erstellen (Bonus, 10 min)

#### 7.1 Produktübersicht View

Erstelle eine View product\_overview:

```
CREATE VIEW product_overview AS
SELECT
    p.id,
    p.name AS product_name,
    p.price,
    p.stock,
    c.name AS category_name,
    p.price * p.stock AS total_value
FROM products p
LEFT JOIN categories c ON p.category_id = c.id;
```

## 7.2 View verwenden

```
-- View abfragen
SELECT * FROM product_overview;

-- Filtern
SELECT * FROM product_overview
WHERE category_name = 'Electronics';
```

## 7.3 Low Stock View

Erstelle eine View `low_stock_products` für Produkte mit `stock < 5`.

### 3.1.1.10 Aufgabe 8: Indizes (Bonus, 5 min)

#### 8.1 Index erstellen

```
-- Index auf häufig gesuchte Spalte
CREATE INDEX idx_products_name ON products(name);

-- Index auf Foreign Key
CREATE INDEX idx_products_category ON products(category_id);

-- Zusammengesetzter Index
CREATE INDEX idx_products_category_price ON products(category_id, price);
```

#### 8.2 Indizes anzeigen

```
-- PostgreSQL
SELECT indexname, tablename
FROM pg_indexes
WHERE schemaname = 'public';
```

### 3.1.1.11 Testen

```
-- Tabellen auflisten
\d

-- Tabellenstruktur anzeigen
\d products
```

```
-- Alle Produkte
SELECT * FROM products;

-- Produkte mit Kategorie
SELECT p.name, c.name AS category
FROM products p
LEFT JOIN categories c ON p.category_id = c.id;
```

### 3.1.1.12 Abgabe-Checkliste

- ☐ 3 Tabellen erstellt (categories, products, customers)
- ☐ Mindestens 4 Kategorien eingefügt
- ☐ Mindestens 10 Produkte eingefügt
- ☐ 5 Kunden eingefügt
- ☐ Alle einfachen Abfragen (Aufgabe 3) funktionieren
- ☐ Komplexere Abfragen mit CASE WHEN
- ☐ JOINS funktionieren
- ☐ UPDATE und DELETE getestet
- ☐ (Bonus) Views erstellt
- ☐ (Bonus) Indizes erstellt

### 3.1.1.13 SQL-Datei speichern

Speichere alle deine SQL-Befehle in einer Datei `shop_setup.sql`:

```
-- shop_setup.sql

-- Tabellen erstellen
CREATE TABLE categories (...);
CREATE TABLE products (...);
CREATE TABLE customers (...);

-- Daten einfügen
INSERT INTO categories ...
INSERT INTO products ...
INSERT INTO customers ...

-- Abfragen
SELECT ...
```

So kannst du die Datenbank jederzeit neu aufsetzen:

```
psql -d shop_db -f shop_setup.sql
```

## 3.1.2 Lösung

### 3.1.2.1 Aufgabe 1: Tabellen erstellen

```
-- Kategorien-Tabelle
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,
```

```
description TEXT
);

-- Produkte-Tabelle
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10, 2) NOT NULL CHECK (price > 0),
    stock INTEGER NOT NULL DEFAULT 0 CHECK (stock >= 0),
    category_id INTEGER REFERENCES categories(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Kunden-Tabelle
CREATE TABLE customers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

### 3.1.2.2 Aufgabe 2: Daten einfügen

Kategorien

```
INSERT INTO categories (name, description) VALUES
('Electronics', 'Electronic devices and accessories'),
('Clothing', 'Fashion and apparel'),
('Books', 'Books and magazines'),
('Home & Garden', 'Home improvement and gardening');
```

Produkte

```
INSERT INTO products (name, description, price, stock, category_id) VALUES
-- Electronics (category_id = 1)
('Laptop Pro 15', 'High-end laptop with 16GB RAM', 1299.99, 25, 1),
('Wireless Mouse', 'Ergonomic wireless mouse', 49.99, 100, 1),
('USB-C Hub', '7-in-1 USB-C hub', 79.99, 50, 1),
('Smartphone X', 'Latest smartphone with 5G', 899.00, 30, 1),

-- Clothing (category_id = 2)
('T-Shirt Basic', '100% cotton t-shirt', 19.99, 200, 2),
('Jeans Classic', 'Blue denim jeans', 59.99, 75, 2),
('Winter Jacket', 'Warm winter jacket', 149.99, 20, 2),

-- Books (category_id = 3)
('Clean Code', 'Programming best practices', 39.99, 45, 3),
('Flutter Complete', 'Learn Flutter development', 49.99, 30, 3),

-- Home & Garden (category_id = 4)
('Garden Tools Set', 'Complete garden tool set', 89.99, 15, 4),
```

```

('LED Lamp', 'Energy-efficient LED lamp', 29.99, 60, 4),

-- Ohne Kategorie
('Mystery Box', 'Surprise item', 9.99, 0, NULL);

```

Kunden

```

INSERT INTO customers (name, email) VALUES
('Max Müller', 'max@example.com'),
('Anna Schmidt', 'anna@example.com'),
('Tom Weber', 'tom@example.com'),
('Lisa Fischer', 'lisa@example.com'),
('Jan Becker', 'jan@example.com');

```

### 3.1.2.3 Aufgabe 3: Einfache Abfragen

3.1 Alle Produkte sortiert nach Name

```

SELECT * FROM products
ORDER BY name ASC;

```

3.2 Produkte über 100€

```

SELECT name, price
FROM products
WHERE price > 100
ORDER BY price DESC;

```

Ergebnis:

name	price
Laptop Pro 15	1299.99
Smartphone X	899.00
Winter Jacket	149.99

3.3 Produkte auf Lager

```

SELECT name, stock
FROM products
WHERE stock > 0
ORDER BY stock DESC;

```

3.4 Günstigstes und teuerstes Produkt

```

SELECT
  MIN(price) AS cheapest,
  MAX(price) AS most_expensive
FROM products;

```

Ergebnis:

cheapest	most_expensive
9.99	1299.99

-- Oder mit Produktnamen:

```
SELECT name, price FROM products WHERE price = (SELECT MIN(price) FROM products)
UNION
SELECT name, price FROM products WHERE price = (SELECT MAX(price) FROM products);
```

### 3.5 Anzahl Produkte pro Kategorie

```
SELECT
    c.name AS category,
    COUNT(p.id) AS product_count
FROM categories c
LEFT JOIN products p ON c.id = p.category_id
GROUP BY c.id, c.name
ORDER BY product_count DESC;
```

Ergebnis:

category	product_count
Electronics	4
Clothing	3
Books	2
Home & Garden	2

#### 3.1.2.4 Aufgabe 4: Komplexere Abfragen

##### 4.1 Filter kombinieren

```
SELECT p.name, p.price, p.stock
FROM products p
JOIN categories c ON p.category_id = c.id
WHERE c.name = 'Electronics'
    AND p.price > 50
    AND p.stock > 0
ORDER BY p.price DESC;
```

Ergebnis:

name	price	stock
Laptop Pro 15	1299.99	25
Smartphone X	899.00	30
USB-C Hub	79.99	50

##### 4.2 Textsuche

```
SELECT name, description
FROM products
WHERE name ILIKE '%pro%'
    OR description ILIKE '%pro%';
```

Ergebnis:

name	description
------	-------------

Laptop Pro 15 | High-end laptop with 16GB RAM

#### 4.3 Preisbereiche

```
SELECT
  name,
  price,
  CASE
    WHEN price < 50 THEN 'Günstig'
    WHEN price <= 200 THEN 'Mittel'
    ELSE 'Premium'
  END AS price_category
FROM products
ORDER BY price;
```

Ergebnis:

name	price	price_category
Mystery Box	9.99	Günstig
T-Shirt Basic	19.99	Günstig
LED Lamp	29.99	Günstig
Clean Code	39.99	Günstig
Wireless Mouse	49.99	Günstig
Flutter Complete	49.99	Günstig
Jeans Classic	59.99	Mittel
USB-C Hub	79.99	Mittel
Garden Tools Set	89.99	Mittel
Winter Jacket	149.99	Mittel
Smartphone X	899.00	Premium
Laptop Pro 15	1299.99	Premium

#### 4.4 Top 5 teuerste Produkte mit Kategorie

```
SELECT
  p.name AS product,
  p.price,
  COALESCE(c.name, 'Keine Kategorie') AS category
FROM products p
LEFT JOIN categories c ON p.category_id = c.id
ORDER BY p.price DESC
LIMIT 5;
```

Ergebnis:

product	price	category
Laptop Pro 15	1299.99	Electronics
Smartphone X	899.00	Electronics
Winter Jacket	149.99	Clothing
Garden Tools	89.99	Home & Garden
USB-C Hub	79.99	Electronics

### 3.1.2.5 Aufgabe 5: JOINS

#### 5.1 Produkte mit Kategorienamen

```
SELECT
    p.id,
    p.name AS product_name,
    p.price,
    p.stock,
    COALESCE(c.name, 'Keine Kategorie') AS category_name
FROM products p
LEFT JOIN categories c ON p.category_id = c.id
ORDER BY category_name, p.name;
```

#### 5.2 Kategorien mit Statistiken

```
SELECT
    c.name AS category,
    COUNT(p.id) AS product_count,
    ROUND(AVG(p.price), 2) AS avg_price,
    SUM(p.price * p.stock) AS total_value
FROM categories c
LEFT JOIN products p ON c.id = p.category_id
GROUP BY c.id, c.name
ORDER BY total_value DESC NULLS LAST;
```

Ergebnis:

category	product_count	avg_price	total_value
Electronics	4	582.24	68747.00
Clothing	3	76.66	11498.50
Books	2	44.99	3299.25
Home & Garden	2	59.99	3149.25

#### 5.3 Leere Kategorien

```
SELECT c.name AS empty_category
FROM categories c
LEFT JOIN products p ON c.id = p.category_id
WHERE p.id IS NULL;
```

Ergebnis (falls vorhanden):

```
empty_category
-----
(keine Ergebnisse wenn alle Kategorien Produkte haben)
```

Zum Testen, füge eine leere Kategorie hinzu:

```
INSERT INTO categories (name) VALUES ('Sports');
```

-- Jetzt nochmal:

```
SELECT c.name AS empty_category
FROM categories c
LEFT JOIN products p ON c.id = p.category_id
```

```
WHERE p.id IS NULL;
```

```
-- Ergebnis:
-- empty_category
-- -----
-- Sports
```

### 3.1.2.6 Aufgabe 6: UPDATE und DELETE

6.1 Preis erhöhen (Electronics +10%)

```
UPDATE products
SET price = price * 1.10
WHERE category_id = (SELECT id FROM categories WHERE name = 'Electronics')
RETURNING name, price;
```

Ergebnis:

name	price
Laptop Pro 15	1429.99
Wireless Mouse	54.99
USB-C Hub	87.99
Smartphone X	988.90

6.2 Stock auffüllen

```
UPDATE products
SET stock = 10
WHERE stock = 0
RETURNING name, stock;
```

6.3 Produkt umbenennen

```
UPDATE products
SET name = 'Laptop Pro 15 (2024 Edition)'
WHERE name = 'Laptop Pro 15'
RETURNING *;
```

6.4 Produkt löschen

```
-- Mystery Box löschen
DELETE FROM products
WHERE name = 'Mystery Box'
RETURNING *;
```

### 3.1.2.7 Aufgabe 7: Views (Bonus)

7.1 Produktübersicht View

```
CREATE VIEW product_overview AS
SELECT
    p.id,
    p.name AS product_name,
```

```
p.price,
p.stock,
COALESCE(c.name, 'Keine Kategorie') AS category_name,
p.price * p.stock AS total_value,
CASE
    WHEN p.stock = 0 THEN 'Ausverkauft'
    WHEN p.stock < 10 THEN 'Wenig auf Lager'
    ELSE 'Auf Lager'
END AS stock_status
FROM products p
LEFT JOIN categories c ON p.category_id = c.id;
```

## 7.2 View verwenden

```
-- Alle Produkte
SELECT * FROM product_overview;

-- Nur Electronics
SELECT * FROM product_overview
WHERE category_name = 'Electronics'
ORDER BY price DESC;

-- Ausverkaufte Produkte
SELECT product_name, price
FROM product_overview
WHERE stock_status = 'Ausverkauft';
```

## 7.3 Low Stock View

```
CREATE VIEW low_stock_products AS
SELECT
    p.name,
    p.stock,
    c.name AS category
FROM products p
LEFT JOIN categories c ON p.category_id = c.id
WHERE p.stock < 5
ORDER BY p.stock ASC;

-- Verwenden
SELECT * FROM low_stock_products;
```

### 3.1.2.8 Aufgabe 8: Indizes (Bonus)

```
-- Index auf Produktname (für Suche)
CREATE INDEX idx_products_name ON products(name);

-- Index auf Foreign Key (für JOINS)
CREATE INDEX idx_products_category ON products(category_id);

-- Zusammengesetzter Index (für Filter + Sort)
```

```

CREATE INDEX idx_products_category_price ON products(category_id, price);

-- Index auf Email (für Login)
CREATE INDEX idx_customers_email ON customers(email);

-- Indizes anzeigen
SELECT
    indexname,
    tablename,
    indexdef
FROM pg_indexes
WHERE schemaname = 'public'
ORDER BY tablename, indexname;

```

### 3.1.2.9 Vollständiges Setup-Skript

```

-- shop_setup.sql
-- Vollständiges Setup für die Shop-Datenbank

-- Alte Tabellen löschen (falls vorhanden)
DROP VIEW IF EXISTS product_overview CASCADE;
DROP VIEW IF EXISTS low_stock_products CASCADE;
DROP TABLE IF EXISTS products CASCADE;
DROP TABLE IF EXISTS customers CASCADE;
DROP TABLE IF EXISTS categories CASCADE;

-- =====
-- Tabellen erstellen
-- =====

CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,
    description TEXT
);

CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10, 2) NOT NULL CHECK (price > 0),
    stock INTEGER NOT NULL DEFAULT 0 CHECK (stock >= 0),
    category_id INTEGER REFERENCES categories(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE customers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,

```

```

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- =====
-- Indizes erstellen
-- =====

CREATE INDEX idx_products_name ON products(name);
CREATE INDEX idx_products_category ON products(category_id);
CREATE INDEX idx_products_category_price ON products(category_id, price);
CREATE INDEX idx_customers_email ON customers(email);

-- =====
-- Daten einfügen
-- =====

INSERT INTO categories (name, description) VALUES
    ('Electronics', 'Electronic devices and accessories'),
    ('Clothing', 'Fashion and apparel'),
    ('Books', 'Books and magazines'),
    ('Home & Garden', 'Home improvement and gardening');

INSERT INTO products (name, description, price, stock, category_id) VALUES
    ('Laptop Pro 15', 'High-end laptop with 16GB RAM', 1299.99, 25, 1),
    ('Wireless Mouse', 'Ergonomic wireless mouse', 49.99, 100, 1),
    ('USB-C Hub', '7-in-1 USB-C hub', 79.99, 50, 1),
    ('Smartphone X', 'Latest smartphone with 5G', 899.00, 30, 1),
    ('T-Shirt Basic', '100% cotton t-shirt', 19.99, 200, 2),
    ('Jeans Classic', 'Blue denim jeans', 59.99, 75, 2),
    ('Winter Jacket', 'Warm winter jacket', 149.99, 20, 2),
    ('Clean Code', 'Programming best practices', 39.99, 45, 3),
    ('Flutter Complete', 'Learn Flutter development', 49.99, 30, 3),
    ('Garden Tools Set', 'Complete garden tool set', 89.99, 15, 4),
    ('LED Lamp', 'Energy-efficient LED lamp', 29.99, 60, 4),
    ('Mystery Box', 'Surprise item', 9.99, 0, NULL);

INSERT INTO customers (name, email) VALUES
    ('Max Müller', 'max@example.com'),
    ('Anna Schmidt', 'anna@example.com'),
    ('Tom Weber', 'tom@example.com'),
    ('Lisa Fischer', 'lisa@example.com'),
    ('Jan Becker', 'jan@example.com');

-- =====
-- Views erstellen
-- =====

CREATE VIEW product_overview AS
SELECT
    p.id,
    p.name AS product_name,
```

```
p.price,
p.stock,
COALESCE(c.name, 'Keine Kategorie') AS category_name,
p.price * p.stock AS total_value,
CASE
    WHEN p.stock = 0 THEN 'Ausverkauft'
    WHEN p.stock < 10 THEN 'Wenig auf Lager'
    ELSE 'Auf Lager'
END AS stock_status
FROM products p
LEFT JOIN categories c ON p.category_id = c.id;

CREATE VIEW low_stock_products AS
SELECT
    p.name,
    p.stock,
    c.name AS category
FROM products p
LEFT JOIN categories c ON p.category_id = c.id
WHERE p.stock < 5
ORDER BY p.stock ASC;

-- =====
-- Überprüfung
-- =====

SELECT 'Kategorien:' AS info, COUNT(*) AS count FROM categories
UNION ALL
SELECT 'Produkte:', COUNT(*) FROM products
UNION ALL
SELECT 'Kunden:', COUNT(*) FROM customers;
```

Ausführen

```
psql -d shop_db -f shop_setup.sql
```

### 3.1.3 Ressourcen

#### 3.1.3.1 Offizielle Dokumentation

- PostgreSQL Documentation
- PostgreSQL Tutorial
- SQLite Documentation
- MySQL Documentation

#### 3.1.3.2 Online-Tools

- SQLite Online
- DB Fiddle
- SQL Fiddle
- Explain PostgreSQL - Query-Analyse

### 3.1.3.3 Cheat Sheet: Datentypen

SQL-Typ	PostgreSQL	Beschreibung
INTEGER	INT4	4-Byte Integer
BIGINT	INT8	8-Byte Integer
SERIAL	INT4 + Sequence	Auto-Increment
DECIMAL(p,s)	NUMERIC	Präzise Dezimalzahl
REAL	FLOAT4	4-Byte Float
VARCHAR(n)	-	Variable Länge bis n
TEXT	-	Unbegrenzte Länge
BOOLEAN	BOOL	true/false
DATE	-	Datum (YYYY-MM-DD)
TIMESTAMP	-	Datum + Zeit
UUID	-	128-bit UUID

### 3.1.3.4 Cheat Sheet: CREATE TABLE

```

CREATE TABLE table_name (
  -- Auto-increment ID
  id SERIAL PRIMARY KEY,

  -- Pflichtfeld
  name VARCHAR(100) NOT NULL,

  -- Eindeutig
  email VARCHAR(255) UNIQUE NOT NULL,

  -- Mit Check
  price DECIMAL(10,2) CHECK (price > 0),

  -- Mit Default
  stock INTEGER DEFAULT 0,

  -- Nullable
  description TEXT,

  -- Foreign Key
  category_id INTEGER REFERENCES categories(id),

  -- Timestamp
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

### 3.1.3.5 Cheat Sheet: INSERT

```

-- Einzelne
INSERT INTO users (name, email)
VALUES ('Max', 'max@example.com');

```

```
-- Mehrere
INSERT INTO users (name, email) VALUES
    ('Max', 'max@example.com'),
    ('Anna', 'anna@example.com');

-- Mit Rückgabe
INSERT INTO users (name, email)
VALUES ('Max', 'max@example.com')
RETURNING id;

-- Aus anderer Tabelle
INSERT INTO archive (name, email)
SELECT name, email FROM users WHERE active = false;
```

### 3.1.3.6 Cheat Sheet: SELECT

```
-- Grundform
SELECT spalten FROM tabelle WHERE bedingung;

-- Alle Spalten
SELECT * FROM users;

-- Mit Alias
SELECT name AS username FROM users;

-- Distinct
SELECT DISTINCT category FROM products;

-- Limit und Offset
SELECT * FROM products LIMIT 10 OFFSET 20;

-- Sortierung
SELECT * FROM products ORDER BY price DESC;

-- Gruppierung
SELECT category, COUNT(*) FROM products GROUP BY category;
```

### 3.1.3.7 Cheat Sheet: WHERE-Klauseln

```
-- Vergleich
WHERE price > 100
WHERE price BETWEEN 50 AND 200
WHERE price IN (10, 20, 30)

-- Text
WHERE name = 'Max'
WHERE name LIKE 'M%'           -- beginnt mit M
WHERE name LIKE '%a'           -- endet mit a
WHERE name LIKE '%ax%'         -- enthält ax
```

```
WHERE name ILIKE '%max%'      -- case-insensitive

-- NULL
WHERE description IS NULL
WHERE description IS NOT NULL

-- Logik
WHERE price > 100 AND stock > 0
WHERE category = 'A' OR category = 'B'
WHERE NOT active
```

### 3.1.3.8 Cheat Sheet: JOIN

```
-- INNER JOIN (nur Übereinstimmungen)
SELECT p.name, c.name
FROM products p
INNER JOIN categories c ON p.category_id = c.id;

-- LEFT JOIN (alle links + Übereinstimmungen)
SELECT p.name, c.name
FROM products p
LEFT JOIN categories c ON p.category_id = c.id;

-- Mehrere JOINS
SELECT o.id, u.name, p.name
FROM orders o
JOIN users u ON o.user_id = u.id
JOIN products p ON o.product_id = p.id;
```

### 3.1.3.9 Cheat Sheet: Aggregatfunktionen

```
COUNT(*)      -- Anzahl Zeilen
COUNT(column) -- Anzahl nicht-NULL Werte
SUM(column)   -- Summe
AVG(column)   -- Durchschnitt
MIN(column)   -- Minimum
MAX(column)   -- Maximum

-- Beispiel
SELECT
    category,
    COUNT(*) AS total,
    AVG(price) AS avg_price,
    SUM(stock) AS total_stock
FROM products
GROUP BY category
HAVING COUNT(*) > 5;
```

### 3.1.3.10 Cheat Sheet: UPDATE

```
-- Einzelnes Feld
UPDATE users SET email = 'new@example.com' WHERE id = 1;

-- Mehrere Felder
UPDATE products
SET price = price * 1.1, updated_at = NOW()
WHERE category = 'electronics';

-- Mit Subquery
UPDATE products
SET category_id = (SELECT id FROM categories WHERE name = 'New')
WHERE category_id IS NULL;

-- Mit RETURNING
UPDATE users SET name = 'Max' WHERE id = 1 RETURNING *;
```

### 3.1.3.11 Cheat Sheet: DELETE

```
-- Mit Bedingung
DELETE FROM users WHERE id = 1;

-- Alle löschen
DELETE FROM users;

-- Schneller alle löschen
TRUNCATE TABLE users;

-- Mit RETURNING
DELETE FROM users WHERE id = 1 RETURNING *;
```

### 3.1.3.12 Cheat Sheet: ALTER TABLE

```
-- Spalte hinzufügen
ALTER TABLE users ADD COLUMN phone VARCHAR(20);

-- Spalte ändern
ALTER TABLE users ALTER COLUMN name TYPE VARCHAR(200);

-- Spalte löschen
ALTER TABLE users DROP COLUMN phone;

-- NOT NULL setzen
ALTER TABLE users ALTER COLUMN email SET NOT NULL;

-- Default setzen
ALTER TABLE users ALTER COLUMN active SET DEFAULT true;

-- Constraint hinzufügen
```

```
ALTER TABLE products ADD CONSTRAINT price_positive CHECK (price > 0);

-- Foreign Key hinzufügen
ALTER TABLE products ADD CONSTRAINT fk_category
    FOREIGN KEY (category_id) REFERENCES categories(id);
```

### 3.1.3.13 Cheat Sheet: Indizes

```
-- Einfacher Index
CREATE INDEX idx_name ON table(column);

-- Unique Index
CREATE UNIQUE INDEX idx_email ON users(email);

-- Zusammengesetzter Index
CREATE INDEX idx_cat_price ON products(category_id, price);

-- Index löschen
DROP INDEX idx_name;

-- Indizes anzeigen (PostgreSQL)
SELECT indexname, tablename FROM pg_indexes WHERE schemaname = 'public';
```

### 3.1.3.14 Cheat Sheet: Views

```
-- View erstellen
CREATE VIEW active_users AS
SELECT id, name, email FROM users WHERE active = true;

-- View verwenden
SELECT * FROM active_users;

-- View ersetzen
CREATE OR REPLACE VIEW active_users AS ...;

-- View löschen
DROP VIEW active_users;
```

### 3.1.3.15 PostgreSQL CLI (psql)

```
# Verbinden
psql -d database_name

# Als User
psql -U username -d database_name

# Remote
psql -h hostname -U username -d database_name
```

psql-Befehle

Befehl	Beschreibung
\l	Datenbanken auflisten
\c dbname	Zu Datenbank verbinden
\dt	Tabellen auflisten
\d table	Tabellenstruktur
\di	Indizes auflisten
\dv	Views auflisten
\du	Benutzer auflisten
\q	Beenden
\i file.sql	SQL-Datei ausführen
\timing	Query-Zeiten anzeigen

### 3.1.3.16 Docker PostgreSQL

```
# Container starten
docker run --name postgres \
  -e POSTGRES_PASSWORD=secret \
  -e POSTGRES_DB=mydb \
  -p 5432:5432 \
  -d postgres:16

# Verbinden
docker exec -it postgres psql -U postgres -d mydb

# Logs
docker logs postgres

# Stoppen
docker stop postgres

# Löschen
docker rm postgres
```

### 3.1.3.17 Best Practices

1. Immer Primary Key verwenden
2. Foreign Keys definieren
3. Indizes für häufige Queries
4. snake\_case für Namen
5. Plural für Tabellennamen
6. NOT NULL wo möglich
7. Transaktionen für zusammengehörige Änderungen
8. EXPLAIN für Performance-Analyse

### 3.1.3.18 Transaktionen

```
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
```

```
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;

-- Bei Fehler
ROLLBACK;
```

## 3.2 Einheit 7.2: PostgreSQL mit Dart

### 3.2.0.1 Lernziele

Nach dieser Einheit kannst du: - PostgreSQL-Verbindungen in Dart herstellen - SQL-Queries ausführen und Ergebnisse verarbeiten - Prepared Statements und Parametrisierung nutzen - Connection Pools für bessere Performance verwenden

### 3.2.0.2 Das postgres Package

Installation

```
# pubspec.yaml
dependencies:
  postgres: ^3.1.0
```

```
dart pub get
```

Warum postgres?

- Offizielles PostgreSQL-Package für Dart
- Asynchron mit Futures
- Connection Pooling
- Prepared Statements
- Typ-sichere Parameter

### 3.2.0.3 Verbindung herstellen

Einfache Verbindung

```
import 'package:postgres/postgres.dart';

Future<void> main() async {
  // Verbindung konfigurieren
  final connection = await Connection.open(
    Endpoint(
      host: 'localhost',
      port: 5432,
      database: 'shop_db',
      username: 'postgres',
      password: 'secret',
    ),
    settings: ConnectionSettings(
      sslMode: SslMode.disable, // Für lokale Entwicklung
    ),
  );
}
```

```

    print('Connected to PostgreSQL!');

    // Verbindung nutzen...

    // Verbindung schließen
    await connection.close();
}

```

Mit Connection String

```

final connection = await Connection.open(
  Endpoint(
    host: 'localhost',
    database: 'shop_db',
    username: 'postgres',
    password: 'secret',
  ),
);

```

Umgebungsvariablen nutzen

```

import 'dart:io';

final connection = await Connection.open(
  Endpoint(
    host: Platform.environment['DB_HOST'] ?? 'localhost',
    port: int.parse(Platform.environment['DB_PORT'] ?? '5432'),
    database: Platform.environment['DB_NAME'] ?? 'shop_db',
    username: Platform.environment['DB_USER'] ?? 'postgres',
    password: Platform.environment['DB_PASSWORD'] ?? 'secret',
  ),
);

```

### 3.2.0.4 Queries ausführen

SELECT - Daten abfragen

```

Future<void> listProducts(Connection conn) async {
  final result = await conn.execute('SELECT * FROM products');

  for (final row in result) {
    print('ID: ${row[0]}, Name: ${row[1]}, Price: ${row[2]}');
  }
}

```

Mit benannten Spalten

```

Future<void> listProducts(Connection conn) async {
  final result = await conn.execute('SELECT id, name, price FROM products');

  for (final row in result) {
    // Zugriff über Index

```

```

    final id = row[0] as int;
    final name = row[1] as String;
    final price = row[2] as double;

    print('$id: $name - €$price');
  }
}

```

Result als Map

```

Future<void> listProducts(Connection conn) async {
  final result = await conn.execute(
    Sql.named('SELECT id, name, price FROM products'),
  );

  for (final row in result) {
    print(row.toColumnMap());
    // {id: 1, name: 'Laptop', price: 1299.99}
  }
}

```

### 3.2.0.5 Parametrisierte Queries

**WICHTIG:** Niemals Strings direkt in SQL einbauen (SQL Injection!)

Falsch (SQL Injection Gefahr!)

```

// NIEMALS SO MACHEN!
final name = ''; DROP TABLE products; --";
await conn.execute("SELECT * FROM products WHERE name = '$name'");

```

Richtig: Positionale Parameter

```

Future<List<Map<String, dynamic>>> findByCategory(
  Connection conn,
  String category,
) async {
  final result = await conn.execute(
    Sql.named('SELECT * FROM products WHERE category = @category'),
    parameters: {'category': category},
  );

  return result.map((row) => row.toColumnMap()).toList();
}

```

Mehrere Parameter

```

Future<List<Map<String, dynamic>>> findProducts(
  Connection conn, {
    required String category,
    required double minPrice,
    required double maxPrice,
  }) async {

```

```

final result = await conn.execute(
  Sql.named('''
    SELECT * FROM products
    WHERE category = @category
      AND price >= @minPrice
      AND price <= @maxPrice
    ORDER BY price
  '''),
  parameters: {
    'category': category,
    'minPrice': minPrice,
    'maxPrice': maxPrice,
  },
);

return result.map((row) => row.toColumnMap()).toList();
}

```

### 3.2.0.6 INSERT, UPDATE, DELETE

INSERT mit RETURNING

```

Future<int> createProduct(
  Connection conn, {
    required String name,
    required double price,
    required int categoryId,
  }) async {
  final result = await conn.execute(
    Sql.named('''
      INSERT INTO products (name, price, category_id)
      VALUES (@name, @price, @categoryId)
      RETURNING id
    '''),
    parameters: {
      'name': name,
      'price': price,
      'categoryId': categoryId,
    },
  );

  // ID der neuen Zeile
  return result.first[0] as int;
}

```

UPDATE

```

Future<int> updatePrice(
  Connection conn, {
    required int id,
    required double newPrice,
  }) async {

```

```

final result = await conn.execute(
  Sql.named('''
    UPDATE products
    SET price = @price, updated_at = NOW()
    WHERE id = @id
  '''),
  parameters: {
    'id': id,
    'price': newPrice,
  },
);

// Anzahl betroffener Zeilen
return result.affectedRows;
}

```

## DELETE

```

Future<bool> deleteProduct(Connection conn, int id) async {
  final result = await conn.execute(
    Sql.named('DELETE FROM products WHERE id = @id'),
    parameters: {'id': id},
  );

  return result.affectedRows > 0;
}

```

### 3.2.0.7 Transaktionen

#### Einfache Transaktion

```

Future<void> transferMoney(
  Connection conn, {
    required int fromId,
    required int toId,
    required double amount,
  }) async {
  await conn.runTx((tx) async {
    // Vom Sender abziehen
    await tx.execute(
      Sql.named('UPDATE accounts SET balance = balance - @amount WHERE id = ↵
↵ @id'),
      parameters: {'id': fromId, 'amount': amount},
    );

    // Zum Empfänger hinzufügen
    await tx.execute(
      Sql.named('UPDATE accounts SET balance = balance + @amount WHERE id = ↵
↵ @id'),
      parameters: {'id': toId, 'amount': amount},
    );
  });
}

```

```
    print('Transfer completed');
}
```

Mit Fehlerbehandlung

```
Future<bool> createOrder(
    Connection conn, {
        required int userId,
        required List<OrderItem> items,
    }) async {
    try {
        await conn.runTx((tx) async {
            // Bestellung erstellen
            final orderResult = await tx.execute(
                Sql.named('''
                    INSERT INTO orders (user_id, total)
                    VALUES (@userId, @total)
                    RETURNING id
                '''),
                parameters: {
                    'userId': userId,
                    'total': items.fold(0.0, (sum, i) => sum + i.price * i.quantity),
                },
            );

            final orderId = orderResult.first[0] as int;

            // Bestellpositionen einfügen
            for (final item in items) {
                await tx.execute(
                    Sql.named('''
                        INSERT INTO order_items (order_id, product_id, quantity, price)
                        VALUES (@orderId, @productId, @quantity, @price)
                    '''),
                    parameters: {
                        'orderId': orderId,
                        'productId': item.productId,
                        'quantity': item.quantity,
                        'price': item.price,
                    },
                );

                // Stock reduzieren
                await tx.execute(
                    Sql.named('''
                        UPDATE products
                        SET stock = stock - @quantity
                        WHERE id = @productId
                    '''),
                    parameters: {
```

```
        'productId': item.productId,  
        'quantity': item.quantity,  
      },  
    );  
  }  
});  
  
  return true;  
} catch (e) {  
  print('Order failed: $e');  
  return false;  
}  
}
```

### 3.2.0.8 Connection Pool

Für bessere Performance bei vielen gleichzeitigen Anfragen.

Pool erstellen

```
import 'package:postgres/postgres.dart';  
  
Future<Pool> createPool() async {  
  final pool = Pool.withEndpoints(  
    [  
      Endpoint(  
        host: 'localhost',  
        database: 'shop_db',  
        username: 'postgres',  
        password: 'secret',  
      ),  
    ],  
    settings: PoolSettings(  
      maxConnectionCount: 10,  
      sslMode: SslMode.disable,  
    ),  
  );  
  
  return pool;  
}
```

Pool verwenden

```
Future<void> main() async {  
  final pool = await createPool();  
  
  // Pool-Verbindung nutzen  
  final products = await pool.execute('SELECT * FROM products');  
  print('Found ${products.length} products');  
  
  // Oder mit Callback  
  await pool.run((conn) async {
```

```

    final result = await conn.execute('SELECT COUNT(*) FROM products');
    print('Total: ${result.first[0]}');
  });

  // Pool schließen
  await pool.close();
}

```

In einer Server-Anwendung

```

late Pool dbPool;

Future<void> initDatabase() async {
  dbPool = Pool.withEndpoints(
    [
      Endpoint(
        host: 'localhost',
        database: 'shop_db',
        username: 'postgres',
        password: 'secret',
      ),
    ],
    settings: PoolSettings(maxConnectionCount: 10),
  );
}

// In Request-Handler
Future<Response> handleGetProducts(Request request) async {
  final result = await dbPool.execute('SELECT * FROM products');
  final products = result.map((row) => row.toColumnMap()).toList();
  return jsonResponse(products);
}

```

### 3.2.0.9 Ergebnisse auf Models mappen

Product Model

```

class Product {
  final int id;
  final String name;
  final String? description;
  final double price;
  final int stock;
  final int? categoryId;
  final DateTime createdAt;

  Product({
    required this.id,
    required this.name,
    this.description,
    required this.price,
    required this.stock,
  }) {}
}

```

```

    this.categoryId,
    required this.createdAt,
  });

  factory Product.fromRow(ResultRow row) {
    final map = row.toColumnMap();
    return Product(
      id: map['id'] as int,
      name: map['name'] as String,
      description: map['description'] as String?,
      price: (map['price'] as num).toDouble(),
      stock: map['stock'] as int,
      categoryId: map['category_id'] as int?,
      createdAt: map['created_at'] as DateTime,
    );
  }

  Map<String, dynamic> toJson() => {
    'id': id,
    'name': name,
    'description': description,
    'price': price,
    'stock': stock,
    'categoryId': categoryId,
    'createdAt': createdAt.toIso8601String(),
  };
}

```

### Repository Pattern

```

class ProductRepository {
  final Pool _pool;

  ProductRepository(this._pool);

  Future<List<Product>> findAll() async {
    final result = await _pool.execute(
      'SELECT * FROM products ORDER BY created_at DESC',
    );
    return result.map(Product.fromRow).toList();
  }

  Future<Product?> findById(int id) async {
    final result = await _pool.execute(
      Sql.named('SELECT * FROM products WHERE id = @id'),
      parameters: {'id': id},
    );

    if (result.isEmpty) return null;
    return Product.fromRow(result.first);
  }
}

```

```
Future<int> create(ProductCreate data) async {
  final result = await _pool.execute(
    Sql.named('''
      INSERT INTO products (name, description, price, stock, category_id)
      VALUES (@name, @description, @price, @stock, @categoryId)
      RETURNING id
    '''),
    parameters: {
      'name': data.name,
      'description': data.description,
      'price': data.price,
      'stock': data.stock,
      'categoryId': data.categoryId,
    },
  );
  return result.first[0] as int;
}

Future<bool> update(int id, ProductUpdate data) async {
  final result = await _pool.execute(
    Sql.named('''
      UPDATE products
      SET name = COALESCE(@name, name),
          description = COALESCE(@description, description),
          price = COALESCE(@price, price),
          stock = COALESCE(@stock, stock)
      WHERE id = @id
    '''),
    parameters: {
      'id': id,
      'name': data.name,
      'description': data.description,
      'price': data.price,
      'stock': data.stock,
    },
  );
  return result.affectedRows > 0;
}

Future<bool> delete(int id) async {
  final result = await _pool.execute(
    Sql.named('DELETE FROM products WHERE id = @id'),
    parameters: {'id': id},
  );
  return result.affectedRows > 0;
}
}
```

### 3.2.0.10 Fehlerbehandlung

```
import 'package:postgres/postgres.dart';

Future<void> safeQuery(Pool pool) async {
  try {
    await pool.execute('SELECT * FROM nonexistent_table');
  } on ServerException catch (e) {
    print('PostgreSQL Error: ${e.message}');
    print('Code: ${e.code}');
    // z.B. 42P01 = undefined_table
  } on SocketException catch (e) {
    print('Connection Error: $e');
  } catch (e) {
    print('Unknown Error: $e');
  }
}
```

Häufige Fehlercodes

Code	Bedeutung
23505	unique_violation (Duplikat)
23503	foreign_key_violation
23502	not_null_violation
42P01	undefined_table
42703	undefined_column

### 3.2.0.11 Zusammenfassung

Operation	Methode
Verbindung	Connection.open()
Query	conn.execute(Sql.named(...))
Parameter	parameters: {'key': value}
Transaktion	conn.runTx((tx) async {...})
Pool	Pool.withEndpoints([...])

### 3.2.0.12 Nächste Schritte

In der nächsten Einheit lernst du das **Repository Pattern**: Wie du Datenbankzugriffe sauber von der Geschäftslogik trennst.

## 3.2.1 Übung

### 3.2.1.1 Ziel

Implementiere eine Dart-Anwendung, die mit PostgreSQL kommuniziert und CRUD-Operationen auf einer Produktdatenbank durchführt.

### 3.2.1.2 Vorbereitung

PostgreSQL starten

```
# Option A: Docker
docker run --name postgres-shop \
  -e POSTGRES_PASSWORD=secret \
  -e POSTGRES_DB=shop_db \
  -p 5432:5432 \
  -d postgres:16

# Option B: Lokal
sudo systemctl start postgresql
createdb shop_db
```

Projekt erstellen

```
mkdir postgres_dart_demo
cd postgres_dart_demo
dart create -t console .
```

Dependencies

```
# pubspec.yaml
name: postgres_dart_demo
environment:
  sdk: ^3.0.0

dependencies:
  postgres: ^3.1.0
```

```
dart pub get
```

### 3.2.1.3 Aufgabe 1: Verbindung herstellen (10 min)

Erstelle `bin/main.dart` mit einer Datenbankverbindung.

Anforderungen

1. Verbindung zu `localhost:5432`, Datenbank `shop_db`
2. Ausgabe bei erfolgreicher Verbindung
3. Verbindung am Ende schließen

Vorlage

```
import 'package:postgres/postgres.dart';

Future<void> main() async {
  print('Connecting to PostgreSQL...');

  // TODO: Connection erstellen

  print('Connected!');

  // TODO: Connection schließen

  print('Disconnected.');
```

Test

```
dart run
# Ausgabe:
# Connecting to PostgreSQL...
# Connected!
# Disconnected.
```

### 3.2.1.4 Aufgabe 2: Tabellen erstellen (10 min)

Erstelle die Tabellen `categories` und `products` via Dart.

SQL für Tabellen

```
CREATE TABLE IF NOT EXISTS categories (
  id SERIAL PRIMARY KEY,
  name VARCHAR(50) NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS products (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  description TEXT,
  price DECIMAL(10, 2) NOT NULL,
  stock INTEGER NOT NULL DEFAULT 0,
  category_id INTEGER REFERENCES categories(id),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Implementierung

```
Future<void> createTables(Connection conn) async {
  // TODO: Tabellen erstellen
}
```

### 3.2.1.5 Aufgabe 3: Daten einfügen (15 min)

Implementiere Funktionen zum Einfügen von Daten.

3.1 Kategorie einfügen

```
Future<int> insertCategory(Connection conn, String name) async {
  // TODO: INSERT mit RETURNING id
  // Parametrisierte Query verwenden!
}
```

3.2 Produkt einfügen

```
Future<int> insertProduct(
  Connection conn, {
    required String name,
    String? description,
    required double price,
```

```
    int stock = 0,  
    int? categoryId,  
  }) async {  
    // TODO: INSERT mit RETURNING id  
  }
```

### 3.3 Seed Data

```
Future<void> seedData(Connection conn) async {  
  // Kategorien  
  final electronicsId = await insertCategory(conn, 'Electronics');  
  final clothingId = await insertCategory(conn, 'Clothing');  
  
  // Produkte  
  await insertProduct(  
    conn,  
    name: 'Laptop Pro',  
    description: 'High-end laptop',  
    price: 1299.99,  
    stock: 25,  
    categoryId: electronicsId,  
  );  
  
  // TODO: Weitere Produkte einfügen  
}
```

#### 3.2.1.6 Aufgabe 4: Daten abfragen (15 min)

##### 4.1 Alle Produkte

```
Future<void> listAllProducts(Connection conn) async {  
  // TODO: SELECT * FROM products  
  // Ausgabe: ID, Name, Preis  
}
```

##### 4.2 Produkt nach ID

```
Future<Map<String, dynamic>?> getProductById(Connection conn, int id) async {  
  // TODO: SELECT mit WHERE id = @id  
  // Return null wenn nicht gefunden  
}
```

##### 4.3 Produkte nach Kategorie

```
Future<List<Map<String, dynamic>>> getProductsByCategory(  
  Connection conn,  
  String categoryName,  
) async {  
  // TODO: JOIN mit categories  
  // Filter nach category name  
}
```

##### 4.4 Produkte mit Preisspanne

```
Future<List<Map<String, dynamic>>> getProductsByPriceRange(
  Connection conn, {
    required double minPrice,
    required double maxPrice,
  }) async {
  // TODO: SELECT mit BETWEEN oder >= AND <=
}
```

### 3.2.1.7 Aufgabe 5: Daten aktualisieren (10 min)

#### 5.1 Preis ändern

```
Future<bool> updateProductPrice(
  Connection conn, {
    required int id,
    required double newPrice,
  }) async {
  // TODO: UPDATE mit RETURNING oder affectedRows prüfen
}
```

#### 5.2 Stock erhöhen

```
Future<int> increaseStock(
  Connection conn, {
    required int productId,
    required int amount,
  }) async {
  // TODO: UPDATE stock = stock + amount
  // Return neuen Stock-Wert
}
```

### 3.2.1.8 Aufgabe 6: Daten löschen (5 min)

```
Future<bool> deleteProduct(Connection conn, int id) async {
  // TODO: DELETE mit affectedRows > 0
}
```

### 3.2.1.9 Aufgabe 7: Transaktionen (15 min)

Implementiere eine Bestellung mit Transaktion.

Szenario

1. Neue Bestellung erstellen
2. Stock der bestellten Produkte reduzieren
3. Bei Fehler: Rollback

Tabelle für Bestellungen

```
CREATE TABLE IF NOT EXISTS orders (
  id SERIAL PRIMARY KEY,
  total DECIMAL(10, 2) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
);

CREATE TABLE IF NOT EXISTS order_items (
    id SERIAL PRIMARY KEY,
    order_id INTEGER REFERENCES orders(id),
    product_id INTEGER REFERENCES products(id),
    quantity INTEGER NOT NULL,
    price DECIMAL(10, 2) NOT NULL
);
```

Implementierung

```
class OrderItem {
    final int productId;
    final int quantity;
    final double price;

    OrderItem({
        required this.productId,
        required this.quantity,
        required this.price,
    });
}

Future<int?> createOrder(Connection conn, List<OrderItem> items) async {
    // TODO: Transaktion mit conn.runTx()
    // 1. Order erstellen
    // 2. Order Items einfügen
    // 3. Stock reduzieren
    // Return: Order ID oder null bei Fehler
}
```

### 3.2.1.10 Aufgabe 8: Product Model (10 min)

Erstelle ein Product-Model mit fromRow Factory.

Product Klasse

```
class Product {
    final int id;
    final String name;
    final String? description;
    final double price;
    final int stock;
    final int? categoryId;
    final DateTime createdAt;

    Product({
        required this.id,
        required this.name,
        this.description,
        required this.price,
```

```

        required this.stock,
        this.categoryId,
        required this.createdAt,
    });

    // TODO: factory Product.fromRow(ResultRow row)

    // TODO: Map<String, dynamic> toJson()
}

```

Verwendung

```

Future<List<Product>> getAllProducts(Connection conn) async {
    final result = await conn.execute('SELECT * FROM products');
    return result.map(Product.fromRow).toList();
}

```

### 3.2.1.11 Aufgabe 9: Connection Pool (Bonus, 10 min)

Refactore die Anwendung mit einem Connection Pool.

```

late Pool dbPool;

Future<void> initPool() async {
    dbPool = Pool.withEndpoints(
        [
            Endpoint(
                host: 'localhost',
                database: 'shop_db',
                username: 'postgres',
                password: 'secret',
            ),
        ],
        settings: PoolSettings(
            maxConnectionCount: 5,
            sslMode: SslMode.disable,
        ),
    );
}

Future<void> closePool() async {
    await dbPool.close();
}

// Nutzung
Future<List<Product>> getAllProducts() async {
    final result = await dbPool.execute('SELECT * FROM products');
    return result.map(Product.fromRow).toList();
}

```

### 3.2.1.12 Testen

Main-Funktion zum Testen

```
Future<void> main() async {
  final conn = await Connection.open(
    Endpoint(
      host: 'localhost',
      database: 'shop_db',
      username: 'postgres',
      password: 'secret',
    ),
    settings: ConnectionSettings(sslMode: SslMode.disable),
  );

  try {
    // Tabellen erstellen
    await createTables(conn);
    print('Tables created');

    // Seed Data
    await seedData(conn);
    print('Data seeded');

    // Alle Produkte
    print('\n--- All Products ---');
    await listAllProducts(conn);

    // Produkt nach ID
    print('\n--- Product by ID ---');
    final product = await getProductById(conn, 1);
    print(product);

    // Preis ändern
    print('\n--- Update Price ---');
    await updateProductPrice(conn, id: 1, newPrice: 1199.99);
    final updated = await getProductById(conn, 1);
    print('New price: ${updated?['price']}');

    // Bestellung erstellen
    print('\n--- Create Order ---');
    final orderId = await createOrder(conn, [
      OrderItem(productId: 1, quantity: 2, price: 1199.99),
    ]);
    print('Order ID: $orderId');

  } finally {
    await conn.close();
  }
}
```

### 3.2.1.13 Abgabe-Checkliste

- ☐ Verbindung zu PostgreSQL funktioniert
- ☐ Tabellen werden erstellt (IF NOT EXISTS)
- ☐ Kategorien können eingefügt werden
- ☐ Produkte können eingefügt werden (mit Parametern)
- ☐ Alle Produkte können abgefragt werden
- ☐ Produkt nach ID finden funktioniert
- ☐ Produkte nach Kategorie filtern
- ☐ Produkte nach Preisspanne filtern
- ☐ Preis aktualisieren funktioniert
- ☐ Stock erhöhen funktioniert
- ☐ Produkt löschen funktioniert
- ☐ Transaktion für Bestellung implementiert
- ☐ Product Model mit fromRow erstellt
- ☐ (Bonus) Connection Pool implementiert

## 3.2.2 Lösung

### 3.2.2.1 Vollständige Lösung

```
import 'package:postgres/postgres.dart';

// =====
// Models
// =====

class Product {
  final int id;
  final String name;
  final String? description;
  final double price;
  final int stock;
  final int? categoryId;
  final DateTime createdAt;

  Product({
    required this.id,
    required this.name,
    this.description,
    required this.price,
    required this.stock,
    this.categoryId,
    required this.createdAt,
  });

  factory Product.fromRow(ResultRow row) {
    final map = row.toColumnMap();
    return Product(
      id: map['id'] as int,
      name: map['name'] as String,
      description: map['description'] as String?,
    );
  }
}
```

```

        price: (map['price'] as num).toDouble(),
        stock: map['stock'] as int,
        categoryId: map['category_id'] as int?,
        createdAt: map['created_at'] as DateTime,
    );
}

Map<String, dynamic> toJson() => {
    'id': id,
    'name': name,
    'description': description,
    'price': price,
    'stock': stock,
    'categoryId': categoryId,
    'createdAt': createdAt.toIso8601String(),
};

@override
String toString() => 'Product($id: $name, €$price, stock: $stock)';
}

class OrderItem {
    final int productId;
    final int quantity;
    final double price;

    OrderItem({
        required this.productId,
        required this.quantity,
        required this.price,
    });

    double get total => price * quantity;
}

// =====
// Aufgabe 1: Verbindung herstellen
// =====

Future<Connection> connect() async {
    return await Connection.open(
        Endpoint(
            host: 'localhost',
            port: 5432,
            database: 'shop_db',
            username: 'postgres',
            password: 'secret',
        ),
        settings: ConnectionSettings(
            sslMode: SslMode.disable,
        ),
    );
}

```

```

    );
}

// =====
// Aufgabe 2: Tabellen erstellen
// =====

Future<void> createTables(Connection conn) async {
    await conn.execute('''
        CREATE TABLE IF NOT EXISTS categories (
            id SERIAL PRIMARY KEY,
            name VARCHAR(50) NOT NULL UNIQUE
        )
    ''');

    await conn.execute('''
        CREATE TABLE IF NOT EXISTS products (
            id SERIAL PRIMARY KEY,
            name VARCHAR(100) NOT NULL,
            description TEXT,
            price DECIMAL(10, 2) NOT NULL,
            stock INTEGER NOT NULL DEFAULT 0,
            category_id INTEGER REFERENCES categories(id),
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    ''');

    await conn.execute('''
        CREATE TABLE IF NOT EXISTS orders (
            id SERIAL PRIMARY KEY,
            total DECIMAL(10, 2) NOT NULL,
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
        )
    ''');

    await conn.execute('''
        CREATE TABLE IF NOT EXISTS order_items (
            id SERIAL PRIMARY KEY,
            order_id INTEGER REFERENCES orders(id),
            product_id INTEGER REFERENCES products(id),
            quantity INTEGER NOT NULL,
            price DECIMAL(10, 2) NOT NULL
        )
    ''');
}

// =====
// Aufgabe 3: Daten einfügen
// =====

Future<int> insertCategory(Connection conn, String name) async {

```

```
final result = await conn.execute(
  Sql.named('''
    INSERT INTO categories (name)
    VALUES (@name)
    ON CONFLICT (name) DO UPDATE SET name = EXCLUDED.name
    RETURNING id
  '''),
  parameters: {'name': name},
);
return result.first[0] as int;
}

Future<int> insertProduct(
  Connection conn, {
    required String name,
    String? description,
    required double price,
    int stock = 0,
    int? categoryId,
  }) async {
  final result = await conn.execute(
    Sql.named('''
      INSERT INTO products (name, description, price, stock, category_id)
      VALUES (@name, @description, @price, @stock, @categoryId)
      RETURNING id
    '''),
    parameters: {
      'name': name,
      'description': description,
      'price': price,
      'stock': stock,
      'categoryId': categoryId,
    },
  );
  return result.first[0] as int;
}

Future<void> seedData(Connection conn) async {
  // Alte Daten löschen
  await conn.execute('DELETE FROM order_items');
  await conn.execute('DELETE FROM orders');
  await conn.execute('DELETE FROM products');
  await conn.execute('DELETE FROM categories');

  // Kategorien
  final electronicsId = await insertCategory(conn, 'Electronics');
  final clothingId = await insertCategory(conn, 'Clothing');
  final booksId = await insertCategory(conn, 'Books');

  // Produkte
  await insertProduct(
```

```
    conn,
    name: 'Laptop Pro 15',
    description: 'High-end laptop with 16GB RAM',
    price: 1299.99,
    stock: 25,
    categoryId: electronicsId,
  );

await insertProduct(
  conn,
  name: 'Wireless Mouse',
  description: 'Ergonomic wireless mouse',
  price: 49.99,
  stock: 100,
  categoryId: electronicsId,
);

await insertProduct(
  conn,
  name: 'USB-C Hub',
  description: '7-in-1 USB-C hub',
  price: 79.99,
  stock: 50,
  categoryId: electronicsId,
);

await insertProduct(
  conn,
  name: 'T-Shirt Basic',
  description: '100% cotton t-shirt',
  price: 19.99,
  stock: 200,
  categoryId: clothingId,
);

await insertProduct(
  conn,
  name: 'Clean Code',
  description: 'Programming best practices by Robert C. Martin',
  price: 39.99,
  stock: 45,
  categoryId: booksId,
);

await insertProduct(
  conn,
  name: 'Mystery Box',
  description: 'Surprise item!',
  price: 9.99,
  stock: 5,
  categoryId: null,
```

```

    );
}

// =====
// Aufgabe 4: Daten abfragen
// =====

Future<void> listAllProducts(Connection conn) async {
    final result = await conn.execute('''
        SELECT id, name, price, stock
        FROM products
        ORDER BY name
    ''');

    for (final row in result) {
        print('${row[0]}: ${row[1]} - €${row[2]} (Stock: ${row[3]})');
    }
}

Future<Map<String, dynamic>?> getProductById(Connection conn, int id) async {
    final result = await conn.execute(
        Sql.named('SELECT * FROM products WHERE id = @id'),
        parameters: {'id': id},
    );

    if (result.isEmpty) return null;
    return result.first.toColumnMap();
}

Future<List<Map<String, dynamic>>> getProductsByCategory(
    Connection conn,
    String categoryName,
) async {
    final result = await conn.execute(
        Sql.named('''
            SELECT p.*, c.name AS category_name
            FROM products p
            JOIN categories c ON p.category_id = c.id
            WHERE c.name = @categoryName
            ORDER BY p.name
        '''),
        parameters: {'categoryName': categoryName},
    );

    return result.map((row) => row.toColumnMap()).toList();
}

Future<List<Map<String, dynamic>>> getProductsByPriceRange(
    Connection conn, {
    required double minPrice,
    required double maxPrice,

```

```

}) async {
  final result = await conn.execute(
    Sql.named('''
      SELECT * FROM products
      WHERE price >= @minPrice AND price <= @maxPrice
      ORDER BY price
    '''),
    parameters: {
      'minPrice': minPrice,
      'maxPrice': maxPrice,
    },
  );

  return result.map((row) => row.toColumnMap()).toList();
}

// Mit Product Model
Future<List<Product>> getAllProducts(Connection conn) async {
  final result = await conn.execute(
    'SELECT * FROM products ORDER BY created_at DESC',
  );
  return result.map(Product.fromRow).toList();
}

// =====
// Aufgabe 5: Daten aktualisieren
// =====

Future<bool> updateProductPrice(
  Connection conn, {
    required int id,
    required double newPrice,
  }) async {
  final result = await conn.execute(
    Sql.named('''
      UPDATE products
      SET price = @price
      WHERE id = @id
    '''),
    parameters: {
      'id': id,
      'price': newPrice,
    },
  );
  return result.affectedRows > 0;
}

Future<int> increaseStock(
  Connection conn, {
    required int productId,
    required int amount,
  }) async {
  final result = await conn.execute(
    Sql.named('''
      UPDATE products
      SET stock = stock + @amount
      WHERE id = @productId
    '''),
    parameters: {
      'productId': productId,
      'amount': amount,
    },
  );
  return result.affectedRows;
}

```

```

}) async {
  final result = await conn.execute(
    Sql.named('''
      UPDATE products
      SET stock = stock + @amount
      WHERE id = @productId
      RETURNING stock
    '''),
    parameters: {
      'productId': productId,
      'amount': amount,
    },
  );

  if (result.isEmpty) {
    throw Exception('Product not found: $productId');
  }

  return result.first[0] as int;
}

// =====
// Aufgabe 6: Daten löschen
// =====

Future<bool> deleteProduct(Connection conn, int id) async {
  final result = await conn.execute(
    Sql.named('DELETE FROM products WHERE id = @id'),
    parameters: {'id': id},
  );
  return result.affectedRows > 0;
}

// =====
// Aufgabe 7: Transaktionen
// =====

Future<int?> createOrder(Connection conn, List<OrderItem> items) async {
  if (items.isEmpty) {
    throw ArgumentError('Order must have at least one item');
  }

  try {
    return await conn.runTx((tx) async {
      // Gesamtsumme berechnen
      final total = items.fold(0.0, (sum, item) => sum + item.total);

      // 1. Order erstellen
      final orderResult = await tx.execute(
        Sql.named('''
          INSERT INTO orders (total)

```

```
        VALUES (@total)
        RETURNING id
        ''),
    parameters: {'total': total},
);

final orderId = orderResult.first[0] as int;

// 2. Für jedes Item
for (final item in items) {
    // Stock prüfen
    final stockResult = await tx.execute(
        Sql.named('SELECT stock FROM products WHERE id = @id'),
        parameters: {'id': item.productId},
    );

    if (stockResult.isEmpty) {
        throw Exception('Product not found: ${item.productId}');
    }

    final currentStock = stockResult.first[0] as int;
    if (currentStock < item.quantity) {
        throw Exception(
            'Not enough stock for product ${item.productId}: '
            'need ${item.quantity}, have $currentStock',
        );
    }

    // Order Item einfügen
    await tx.execute(
        Sql.named(''
            INSERT INTO order_items (order_id, product_id, quantity, price)
            VALUES (@orderId, @productId, @quantity, @price)
            '''),
        parameters: {
            'orderId': orderId,
            'productId': item.productId,
            'quantity': item.quantity,
            'price': item.price,
        },
    );

    // Stock reduzieren
    await tx.execute(
        Sql.named(''
            UPDATE products
            SET stock = stock - @quantity
            WHERE id = @productId
            '''),
        parameters: {
            'productId': item.productId,
```

```

        'quantity': item.quantity,
    },
    );
}

    return orderId;
});
} catch (e) {
    print('Order failed: $e');
    return null;
}
}

// =====
// Aufgabe 9: Connection Pool
// =====

late Pool dbPool;

Future<void> initPool() async {
    dbPool = Pool.withEndpoints(
        [
            Endpoint(
                host: 'localhost',
                database: 'shop_db',
                username: 'postgres',
                password: 'secret',
            ),
        ],
        settings: PoolSettings(
            maxConnectionCount: 5,
            sslMode: SslMode.disable,
        ),
    );
}

Future<void> closePool() async {
    await dbPool.close();
}

Future<List<Product>> getAllProductsFromPool() async {
    final result = await dbPool.execute('SELECT * FROM products');
    return result.map(Product.fromRow).toList();
}

// =====
// Main
// =====

Future<void> main() async {
    print('Connecting to PostgreSQL...');

```

```
final conn = await connect();
print('Connected!\n');

try {
  // Tabellen erstellen
  print('Creating tables...');
  await createTables(conn);
  print('Tables created.\n');

  // Seed Data
  print('Seeding data...');
  await seedData(conn);
  print('Data seeded.\n');

  // ===== Aufgabe 4: Abfragen =====

  print('=== All Products ===');
  await listAllProducts(conn);

  print('\n=== Product by ID (1) ===');
  final product = await getProductById(conn, 1);
  if (product != null) {
    print('Found: ${product['name']} - €${product['price']}');
  }

  print('\n=== Products in Electronics ===');
  final electronics = await getProductsByCategory(conn, 'Electronics');
  for (final p in electronics) {
    print('  ${p['name']}: €${p['price']}');
  }

  print('\n=== Products €10-€50 ===');
  final affordable = await getProductsByPriceRange(
    conn,
    minPrice: 10,
    maxPrice: 50,
  );
  for (final p in affordable) {
    print('  ${p['name']}: €${p['price']}');
  }

  // ===== Aufgabe 5: Update =====

  print('\n=== Update Price ===');
  final updated = await updateProductPrice(conn, id: 1, newPrice: 1199.99);
  print('Updated: $updated');
  final updatedProduct = await getProductById(conn, 1);
  print('New price: €${updatedProduct?['price']}');

  print('\n=== Increase Stock ===');
  final newStock = await increaseStock(conn, productId: 1, amount: 5);
```

```
print('New stock for product 1: $newStock');

// ===== Aufgabe 7: Transaktion =====

print('\n=== Create Order ===');
// Produkt-Info für Bestellung holen
final productForOrder = await getProductById(conn, 1);
if (productForOrder != null) {
    print('Before order - Stock: ${productForOrder['stock']}');

    final orderId = await createOrder(conn, [
        OrderItem(
            productId: 1,
            quantity: 2,
            price: (productForOrder['price'] as num).toDouble(),
        ),
    ]);

    if (orderId != null) {
        print('Order created with ID: $orderId');

        final afterOrder = await getProductById(conn, 1);
        print('After order - Stock: ${afterOrder?['stock']}');
    }
}

// ===== Mit Product Model =====

print('\n=== All Products (as Models) ===');
final allProducts = await getAllProducts(conn);
for (final p in allProducts) {
    print(p);
}

// ===== Aufgabe 6: Delete =====

print('\n=== Delete Product ===');
// Produkt ohne Kategorie löschen
final result = await conn.execute(
    Sql.named('SELECT id FROM products WHERE category_id IS NULL LIMIT 1'),
);
if (result.isNotEmpty) {
    final idToDelete = result.first[0] as int;
    final deleted = await deleteProduct(conn, idToDelete);
    print('Deleted product $idToDelete: $deleted');
}

// ===== Pool Demo =====

print('\n=== Pool Demo ===');
await initPool();
```

```
    final poolProducts = await getAllProductsFromPool();
    print('Products from pool: ${poolProducts.length}');
    await closePool();
    print('Pool closed.');
```

```
  } catch (e, stackTrace) {
    print('Error: $e');
    print(stackTrace);
  } finally {
    await conn.close();
    print('\nDisconnected.');
```

```
  }
}
```

### 3.2.2.2 Ausgabe

Connecting to PostgreSQL...  
Connected!

Creating tables...  
Tables created.

Seeding data...  
Data seeded.

=== All Products ===  
5: Clean Code - €39.99 (Stock: 45)  
1: Laptop Pro 15 - €1299.99 (Stock: 25)  
6: Mystery Box - €9.99 (Stock: 5)  
4: T-Shirt Basic - €19.99 (Stock: 200)  
3: USB-C Hub - €79.99 (Stock: 50)  
2: Wireless Mouse - €49.99 (Stock: 100)

=== Product by ID (1) ===  
Found: Laptop Pro 15 - €1299.99

=== Products in Electronics ===  
Laptop Pro 15: €1299.99  
USB-C Hub: €79.99  
Wireless Mouse: €49.99

=== Products €10-€50 ===  
T-Shirt Basic: €19.99  
Clean Code: €39.99  
Wireless Mouse: €49.99

=== Update Price ===  
Updated: true  
New price: €1199.99

=== Increase Stock ===

New stock for product 1: 30

=== Create Order ===

Before order - Stock: 30

Order created with ID: 1

After order - Stock: 28

=== All Products (as Models) ===

Product(6: Mystery Box, €9.99, stock: 5)

Product(5: Clean Code, €39.99, stock: 45)

Product(4: T-Shirt Basic, €19.99, stock: 200)

Product(3: USB-C Hub, €79.99, stock: 50)

Product(2: Wireless Mouse, €49.99, stock: 100)

Product(1: Laptop Pro 15, €1199.99, stock: 28)

=== Delete Product ===

Deleted product 6: true

=== Pool Demo ===

Products from pool: 5

Pool closed.

Disconnected.

### 3.2.2.3 Projektstruktur

```
postgres_dart_demo/
+-- bin/
|   +-- main.dart
+-- lib/
|   +-- models/
|   |   +-- product.dart
|   +-- repositories/
|   |   +-- product_repository.dart
|   +-- database.dart
+-- pubspec.yaml

lib/database.dart

import 'package:postgres/postgres.dart';

class Database {
  static Pool? _pool;

  static Future<Pool> get pool async {
    _pool ??= Pool.withEndpoints(
      [
        Endpoint(
          host: 'localhost',
          database: 'shop_db',
          username: 'postgres',
          password: 'secret',
        ),
      ],
    ),
  }
}
```

```

    ],
    settings: PoolSettings(
      maxConnectionCount: 10,
      sslMode: SslMode.disable,
    ),
  );
  return _pool!;
}

static Future<void> close() async {
  await _pool?.close();
  _pool = null;
}
}

```

lib/repositories/product\_repository.dart

```

import 'package:postgres/postgres.dart';
import '../models/product.dart';

class ProductRepository {
  final Pool _pool;

  ProductRepository(this._pool);

  Future<List<Product>> findAll() async {
    final result = await _pool.execute(
      'SELECT * FROM products ORDER BY created_at DESC',
    );
    return result.map(Product.fromRow).toList();
  }

  Future<Product?> findById(int id) async {
    final result = await _pool.execute(
      Sql.named('SELECT * FROM products WHERE id = @id'),
      parameters: {'id': id},
    );
    if (result.isEmpty) return null;
    return Product.fromRow(result.first);
  }

  Future<int> create({
    required String name,
    String? description,
    required double price,
    int stock = 0,
    int? categoryId,
  }) async {
    final result = await _pool.execute(
      Sql.named(''
        INSERT INTO products (name, description, price, stock, category_id)

```

```

        VALUES (@name, @description, @price, @stock, @categoryId)
        RETURNING id
    '''),
    parameters: {
        'name': name,
        'description': description,
        'price': price,
        'stock': stock,
        'categoryId': categoryId,
    },
);
return result.first[0] as int;
}

Future<bool> delete(int id) async {
    final result = await _pool.execute(
        Sql.named('DELETE FROM products WHERE id = @id'),
        parameters: {'id': id},
    );
    return result.affectedRows > 0;
}
}

```

### 3.2.3 Ressourcen

#### 3.2.3.1 Offizielle Dokumentation

- postgres Package (pub.dev)
- postgres API Documentation
- PostgreSQL Documentation

#### 3.2.3.2 Cheat Sheet: Verbindung

```

import 'package:postgres/postgres.dart';

// Einfache Verbindung
final conn = await Connection.open(
    Endpoint(
        host: 'localhost',
        port: 5432,
        database: 'mydb',
        username: 'postgres',
        password: 'secret',
    ),
    settings: ConnectionSettings(
        sslMode: SslMode.disable,
    ),
);

// Verbindung schließen
await conn.close();

```

### 3.2.3.3 Cheat Sheet: Connection Pool

```
// Pool erstellen
final pool = Pool.withEndpoints(
  [
    Endpoint(
      host: 'localhost',
      database: 'mydb',
      username: 'postgres',
      password: 'secret',
    ),
  ],
  settings: PoolSettings(
    maxConnectionCount: 10,
    sslMode: SslMode.disable,
  ),
);

// Query ausführen
final result = await pool.execute('SELECT * FROM users');

// Pool schließen
await pool.close();
```

### 3.2.3.4 Cheat Sheet: Queries

```
// Einfache Query
final result = await conn.execute('SELECT * FROM users');

// Mit benannten Parametern
final result = await conn.execute(
  Sql.named('SELECT * FROM users WHERE id = @id'),
  parameters: {'id': 1},
);

// Mehrere Parameter
final result = await conn.execute(
  Sql.named('''
    SELECT * FROM products
    WHERE category = @category
    AND price >= @minPrice
    AND price <= @maxPrice
  '''),
  parameters: {
    'category': 'electronics',
    'minPrice': 50.0,
    'maxPrice': 500.0,
  },
);
```

### 3.2.3.5 Cheat Sheet: Ergebnisse verarbeiten

```
// Iteration
for (final row in result) {
    print('ID: ${row[0]}, Name: ${row[1]}');
}

// Als Map
for (final row in result) {
    final map = row.toColumnMap();
    print(map['name']);
}

// Zu Liste konvertieren
final products = result.map((row) => row.toColumnMap()).toList();

// Mit Model
final products = result.map(Product.fromRow).toList();
```

### 3.2.3.6 Cheat Sheet: INSERT

```
// INSERT mit RETURNING
final result = await conn.execute(
    Sql.named('''
        INSERT INTO products (name, price)
        VALUES (@name, @price)
        RETURNING id
    '''),
    parameters: {
        'name': 'New Product',
        'price': 99.99,
    },
);

final newId = result.first[0] as int;
```

### 3.2.3.7 Cheat Sheet: UPDATE

```
final result = await conn.execute(
    Sql.named('''
        UPDATE products
        SET price = @price
        WHERE id = @id
    '''),
    parameters: {
        'id': 1,
        'price': 89.99,
    },
);
```

```
final rowsAffected = result.affectedRows;
```

### 3.2.3.8 Cheat Sheet: DELETE

```
final result = await conn.execute(
  Sql.named('DELETE FROM products WHERE id = @id'),
  parameters: {'id': 1},
);

final deleted = result.affectedRows > 0;
```

### 3.2.3.9 Cheat Sheet: Transaktionen

```
await conn.runTx((tx) async {
  // Alle Queries in der Transaktion
  await tx.execute(
    Sql.named('UPDATE accounts SET balance = balance - @amount WHERE id = ↵
↵ @from'),
    parameters: {'from': 1, 'amount': 100},
  );

  await tx.execute(
    Sql.named('UPDATE accounts SET balance = balance + @amount WHERE id = @to'),
    parameters: {'to': 2, 'amount': 100},
  );

  // Automatischer COMMIT am Ende
  // Automatischer ROLLBACK bei Exception
});
```

### 3.2.3.10 Cheat Sheet: Model Mapping

```
class Product {
  final int id;
  final String name;
  final double price;

  Product({
    required this.id,
    required this.name,
    required this.price,
  });

  factory Product.fromRow(ResultRow row) {
    final map = row.toColumnMap();
    return Product(
      id: map['id'] as int,
      name: map['name'] as String,
      price: (map['price'] as num).toDouble(),
    );
  }
}
```

```

    );
  }

  Map<String, dynamic> toJson() => {
    'id': id,
    'name': name,
    'price': price,
  };
}

```

### 3.2.3.11 Cheat Sheet: Fehlerbehandlung

```

try {
  await conn.execute('...');
} on ServerException catch (e) {
  print('PostgreSQL Error: ${e.message}');
  print('Code: ${e.code}');

  switch (e.code) {
    case '23505':
      print('Duplicate entry');
      break;
    case '23503':
      print('Foreign key violation');
      break;
    case '23502':
      print('Not null violation');
      break;
  }
} on SocketException catch (e) {
  print('Connection failed: $e');
} catch (e) {
  print('Unknown error: $e');
}

```

### 3.2.3.12 Cheat Sheet: Datentypen

PostgreSQL	Dart	Konvertierung
INTEGER	int	row[0] as int
BIGINT	int	row[0] as int
DECIMAL	num	(row[0] as num).toDouble()
TEXT	String	row[0] as String
VARCHAR	String	row[0] as String
BOOLEAN	bool	row[0] as bool
TIMESTAMP	DateTime	row[0] as DateTime
DATE	DateTime	row[0] as DateTime
UUID	String	row[0] as String
JSONB	Map	row[0] as Map

### 3.2.3.13 Umgebungsvariablen

```
import 'dart:io';

final conn = await Connection.open(
  Endpoint(
    host: Platform.environment['DB_HOST'] ?? 'localhost',
    port: int.parse(Platform.environment['DB_PORT'] ?? '5432'),
    database: Platform.environment['DB_NAME'] ?? 'mydb',
    username: Platform.environment['DB_USER'] ?? 'postgres',
    password: Platform.environment['DB_PASSWORD'] ?? '',
  ),
);
```

```
# .env oder Shell
export DB_HOST=localhost
export DB_PORT=5432
export DB_NAME=shop_db
export DB_USER=postgres
export DB_PASSWORD=secret

dart run
```

### 3.2.3.14 Docker Compose

```
# docker-compose.yml
version: '3.8'

services:
  postgres:
    image: postgres:16
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: secret
      POSTGRES_DB: shop_db
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

volumes:
  postgres_data:
```

```
docker-compose up -d
```

### 3.2.3.15 Best Practices

1. **Immer parametrisierte Queries** - Nie Strings konkatenieren
2. **Connection Pool** für Server-Anwendungen
3. **Transaktionen** für zusammengehörige Änderungen
4. **Models** für Typ-Sicherheit

5. **Fehlerbehandlung** mit spezifischen Catches
6. **Umgebungsvariablen** für Credentials
7. **Connection schließen** im finally-Block

### 3.2.3.16 Häufige Fehler

Problem	Lösung
Connection refused	PostgreSQL läuft? Port korrekt?
Authentication failed	Username/Password prüfen
SSL required	<code>sslMode: SslMode.require</code>
Table not found	Schema/Tabelle existiert?
Type mismatch	Typ-Konvertierung prüfen

## 3.3 Einheit 7.3: Repository Pattern

### 3.3.0.1 Lernziele

Nach dieser Einheit kannst du: - Das Repository Pattern verstehen und anwenden - Datenbankzugriffe von der Geschäftslogik trennen - Testbare Repositories mit Interfaces erstellen - Unit-of-Work Pattern für Transaktionen nutzen

### 3.3.0.2 Was ist das Repository Pattern?

Problem ohne Repository

```
// Direkte Datenbankzugriffe überall im Code
class ProductService {
    final Pool _db;

    Future<Product?> getProduct(int id) async {
        final result = await _db.execute(
            Sql.named('SELECT * FROM products WHERE id = @id'),
            parameters: {'id': id},
        );
        if (result.isEmpty) return null;
        return Product.fromRow(result.first);
    }

    Future<void> updateStock(int id, int quantity) async {
        // SQL direkt im Service - schwer zu testen!
        await _db.execute(
            Sql.named('UPDATE products SET stock = stock - @qty WHERE id = @id'),
            parameters: {'id': id, 'qty': quantity},
        );
    }
}
```

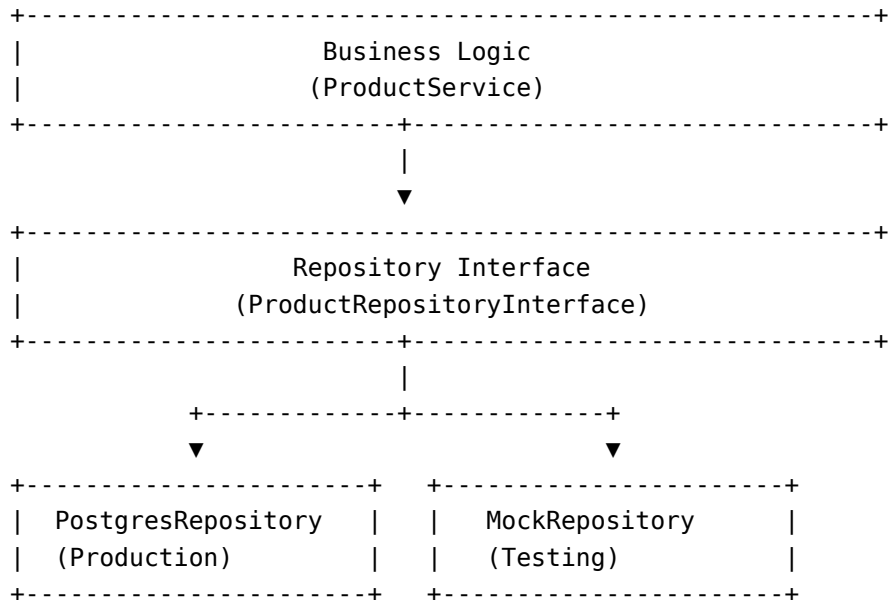
Probleme

- SQL-Code verstreut in der gesamten Anwendung
- Schwer zu testen (Datenbank-Abhängigkeit)
- Duplizierter Code

- Schwer zu warten bei Schema-Änderungen

### 3.3.0.3 Die Lösung: Repository Pattern

Das **Repository Pattern** abstrahiert den Datenzugriff hinter einem Interface.



### 3.3.0.4 Repository Interface

Generisches Interface

```

abstract class Repository<T, ID> {
    Future<List<T>> findAll();
    Future<T?> findById(ID id);
    Future<T> create(T entity);
    Future<T?> update(ID id, T entity);
    Future<bool> delete(ID id);
}
  
```

Spezifisches Interface

```

abstract class ProductRepository {
    Future<List<Product>> findAll();
    Future<Product?> findById(int id);
    Future<List<Product>> findByCategory(String category);
    Future<List<Product>> findByPriceRange(double min, double max);
    Future<List<Product>> search(String query);

    Future<Product> create(ProductCreate data);
    Future<Product?> update(int id, ProductUpdate data);
    Future<bool> delete(int id);

    Future<bool> updateStock(int id, int quantity);
}
  
```

### 3.3.0.5 PostgreSQL Implementation

```
import 'package:postgres/postgres.dart';

class PostgresProductRepository implements ProductRepository {
  final Pool _pool;

  PostgresProductRepository(this._pool);

  @override
  Future<List<Product>> findAll() async {
    final result = await _pool.execute(
      'SELECT * FROM products ORDER BY created_at DESC',
    );
    return result.map(Product.fromRow).toList();
  }

  @override
  Future<Product?> findById(int id) async {
    final result = await _pool.execute(
      Sql.named('SELECT * FROM products WHERE id = @id'),
      parameters: {'id': id},
    );

    if (result.isEmpty) return null;
    return Product.fromRow(result.first);
  }

  @override
  Future<List<Product>> findByCategory(String category) async {
    final result = await _pool.execute(
      Sql.named('''
        SELECT p.* FROM products p
        JOIN categories c ON p.category_id = c.id
        WHERE c.name = @category
        ORDER BY p.name
      '''),
      parameters: {'category': category},
    );
    return result.map(Product.fromRow).toList();
  }

  @override
  Future<List<Product>> findByPriceRange(double min, double max) async {
    final result = await _pool.execute(
      Sql.named('''
        SELECT * FROM products
        WHERE price >= @min AND price <= @max
        ORDER BY price
      '''),
      parameters: {'min': min, 'max': max},
    );
  }
}
```

```

    );
    return result.map(Product.fromRow).toList();
}

@Override
Future<List<Product>> search(String query) async {
    final result = await _pool.execute(
        Sql.named('''
            SELECT * FROM products
            WHERE name ILIKE @query
            OR description ILIKE @query
            ORDER BY name
        '''),
        parameters: {'query': '%$query%'},
    );
    return result.map(Product.fromRow).toList();
}

@Override
Future<Product> create(ProductCreate data) async {
    final result = await _pool.execute(
        Sql.named('''
            INSERT INTO products (name, description, price, stock, category_id)
            VALUES (@name, @description, @price, @stock, @categoryId)
            RETURNING *
        '''),
        parameters: {
            'name': data.name,
            'description': data.description,
            'price': data.price,
            'stock': data.stock,
            'categoryId': data.categoryId,
        },
    );
    return Product.fromRow(result.first);
}

@Override
Future<Product?> update(int id, ProductUpdate data) async {
    final result = await _pool.execute(
        Sql.named('''
            UPDATE products SET
            name = COALESCE(@name, name),
            description = COALESCE(@description, description),
            price = COALESCE(@price, price),
            stock = COALESCE(@stock, stock),
            category_id = COALESCE(@categoryId, category_id),
            updated_at = NOW()
            WHERE id = @id
            RETURNING *
        '''),

```

```

        parameters: {
            'id': id,
            'name': data.name,
            'description': data.description,
            'price': data.price,
            'stock': data.stock,
            'categoryId': data.categoryId,
        },
    );

    if (result.isEmpty) return null;
    return Product.fromRow(result.first);
}

@override
Future<bool> delete(int id) async {
    final result = await _pool.execute(
        Sql.named('DELETE FROM products WHERE id = @id'),
        parameters: {'id': id},
    );
    return result.affectedRows > 0;
}

@override
Future<bool> updateStock(int id, int quantity) async {
    final result = await _pool.execute(
        Sql.named('''
            UPDATE products
            SET stock = stock + @quantity
            WHERE id = @id AND stock + @quantity >= 0
        '''),
        parameters: {'id': id, 'quantity': quantity},
    );
    return result.affectedRows > 0;
}
}

```

### 3.3.0.6 DTOs (Data Transfer Objects)

Create DTO

```

class ProductCreate {
    final String name;
    final String? description;
    final double price;
    final int stock;
    final int? categoryId;

    ProductCreate({
        required this.name,
        this.description,
    })

```

```
        required this.price,
        this.stock = 0,
        this.categoryId,
    });

    factory ProductCreate.fromJson(Map<String, dynamic> json) {
        return ProductCreate(
            name: json['name'] as String,
            description: json['description'] as String?,
            price: (json['price'] as num).toDouble(),
            stock: json['stock'] as int? ?? 0,
            categoryId: json['categoryId'] as int?,
        );
    }
}
```

Update DTO

```
class ProductUpdate {
    final String? name;
    final String? description;
    final double? price;
    final int? stock;
    final int? categoryId;

    ProductUpdate({
        this.name,
        this.description,
        this.price,
        this.stock,
        this.categoryId,
    });

    factory ProductUpdate.fromJson(Map<String, dynamic> json) {
        return ProductUpdate(
            name: json['name'] as String?,
            description: json['description'] as String?,
            price: (json['price'] as num?)?.toDouble(),
            stock: json['stock'] as int?,
            categoryId: json['categoryId'] as int?,
        );
    }

    bool get isEmpty =>
        name == null &&
        description == null &&
        price == null &&
        stock == null &&
        categoryId == null;
}
```

### 3.3.0.7 Service Layer

Der Service verwendet das Repository für Geschäftslogik.

```
class ProductService {
    final ProductRepository _repository;

    ProductService(this._repository);

    Future<List<Product>> getAllProducts() {
        return _repository.findAll();
    }

    Future<Product?> getProduct(int id) {
        return _repository.findById(id);
    }

    Future<Product> createProduct(ProductCreate data) async {
        // Geschäftslogik / Validierung
        if (data.price <= 0) {
            throw ValidationException('Price must be positive');
        }
        if (data.name.length < 2) {
            throw ValidationException('Name must be at least 2 characters');
        }

        return _repository.create(data);
    }

    Future<Product?> updateProduct(int id, ProductUpdate data) async {
        if (data.isEmpty) {
            throw ValidationException('No data to update');
        }

        final existing = await _repository.findById(id);
        if (existing == null) {
            throw NotFoundException('Product not found: $id');
        }

        return _repository.update(id, data);
    }

    Future<bool> deleteProduct(int id) async {
        final existing = await _repository.findById(id);
        if (existing == null) {
            throw NotFoundException('Product not found: $id');
        }

        return _repository.delete(id);
    }

    Future<bool> reduceStock(int id, int quantity) async {
```

```
    if (quantity <= 0) {
        throw ValidationException('Quantity must be positive');
    }

    final product = await _repository.findById(id);
    if (product == null) {
        throw NotFoundException('Product not found: $id');
    }

    if (product.stock < quantity) {
        throw InsufficientStockException(
            'Not enough stock: need $quantity, have ${product.stock}',
        );
    }

    return _repository.updateStock(id, -quantity);
}
```

#### 3.3.0.8 Mock Repository für Tests

```
class MockProductRepository implements ProductRepository {
    final List<Product> _products = [];
    int _nextId = 1;

    @override
    Future<List<Product>> findAll() async {
        return List.from(_products);
    }

    @override
    Future<Product?> findById(int id) async {
        try {
            return _products.firstWhere((p) => p.id == id);
        } catch (_) {
            return null;
        }
    }

    @override
    Future<List<Product>> findByCategory(String category) async {
        // Vereinfacht für Tests
        return _products.where((p) => p.categoryId != null).toList();
    }

    @override
    Future<List<Product>> findByPriceRange(double min, double max) async {
        return _products.where((p) => p.price >= min && p.price <= max).toList();
    }
}
```

```
@Override
Future<List<Product>> search(String query) async {
    final q = query.toLowerCase();
    return _products.where((p) =>
        p.name.toLowerCase().contains(q) ||
        (p.description?.toLowerCase().contains(q) ?? false)
    ).toList();
}

@Override
Future<Product> create(ProductCreate data) async {
    final product = Product(
        id: _nextId++,
        name: data.name,
        description: data.description,
        price: data.price,
        stock: data.stock,
        categoryId: data.categoryId,
        createdAt: DateTime.now(),
    );
    _products.add(product);
    return product;
}

@Override
Future<Product?> update(int id, ProductUpdate data) async {
    final index = _products.indexWhere((p) => p.id == id);
    if (index == -1) return null;

    final existing = _products[index];
    final updated = Product(
        id: existing.id,
        name: data.name ?? existing.name,
        description: data.description ?? existing.description,
        price: data.price ?? existing.price,
        stock: data.stock ?? existing.stock,
        categoryId: data.categoryId ?? existing.categoryId,
        createdAt: existing.createdAt,
    );

    _products[index] = updated;
    return updated;
}

@Override
Future<bool> delete(int id) async {
    final lengthBefore = _products.length;
    _products.removeWhere((p) => p.id == id);
    return _products.length < lengthBefore;
}
```

```

@override
Future<bool> updateStock(int id, int quantity) async {
  final product = await findById(id);
  if (product == null) return false;
  if (product.stock + quantity < 0) return false;

  await update(id, ProductUpdate(stock: product.stock + quantity));
  return true;
}

// Test-Hilfsmethoden
void clear() => _products.clear();
void seed(List<Product> products) {
  _products.addAll(products);
  _nextId = products.map((p) => p.id).reduce((a, b) => a > b ? a : b) + 1;
}
}

```

### 3.3.0.9 Unit Tests

```

import 'package:test/test.dart';

void main() {
  late MockProductRepository repository;
  late ProductService service;

  setUp(() {
    repository = MockProductRepository();
    service = ProductService(repository);
  });

  group('ProductService', () {
    test('createProduct - valid data - returns product', () async {
      final data = ProductCreate(
        name: 'Test Product',
        price: 99.99,
        stock: 10,
      );

      final product = await service.createProduct(data);

      expect(product.name, equals('Test Product'));
      expect(product.price, equals(99.99));
      expect(product.id, isPositive);
    });

    test('createProduct - negative price - throws', () async {
      final data = ProductCreate(
        name: 'Test',
        price: -10,

```

```

    );

    expect(
        () => service.createProduct(data),
        throwsA(isA<ValidationException>()),
    );
});

test('reduceStock - sufficient stock - reduces', () async {
    // Arrange
    await repository.create(ProductCreate(
        name: 'Test',
        price: 10,
        stock: 50,
    ));

    // Act
    final success = await service.reduceStock(1, 10);

    // Assert
    expect(success, isTrue);
    final product = await repository.findById(1);
    expect(product?.stock, equals(40));
});

test('reduceStock - insufficient stock - throws', () async {
    await repository.create(ProductCreate(
        name: 'Test',
        price: 10,
        stock: 5,
    ));

    expect(
        () => service.reduceStock(1, 10),
        throwsA(isA<InsufficientStockException>()),
    );
});
});
}

```

### 3.3.0.10 Unit of Work Pattern

Für Transaktionen über mehrere Repositories.

```

abstract class UnitOfWork {
    ProductRepository get products;
    OrderRepository get orders;
    CustomerRepository get customers;

    Future<T> transaction<T>(Future<T> Function() action);
}

```

```

class PostgresUnitOfWork implements UnitOfWork {
    final Pool _pool;

    late final ProductRepository _products;
    late final OrderRepository _orders;
    late final CustomerRepository _customers;

    PostgresUnitOfWork(this._pool) {
        _products = PostgresProductRepository(_pool);
        _orders = PostgresOrderRepository(_pool);
        _customers = PostgresCustomerRepository(_pool);
    }

    @override
    ProductRepository get products => _products;

    @override
    OrderRepository get orders => _orders;

    @override
    CustomerRepository get customers => _customers;

    @override
    Future<T> transaction<T>(Future<T> Function() action) async {
        // Hier würde die eigentliche Transaktionslogik sein
        // In diesem Fall vereinfacht
        return await action();
    }
}

```

#### Verwendung

```

class OrderService {
    final UnitOfWork _uow;

    OrderService(this._uow);

    Future<Order> createOrder(int customerId, List<OrderItem> items) async {
        return await _uow.transaction(() async {
            // Kunde prüfen
            final customer = await _uow.customers.findById(customerId);
            if (customer == null) {
                throw NotFoundException('Customer not found');
            }

            // Stock prüfen und reduzieren
            for (final item in items) {
                final product = await _uow.products.findById(item.productId);
                if (product == null) {
                    throw NotFoundException('Product not found: ${item.productId}');
                }
            }
        });
    }
}

```

```
    }
    if (product.stock < item.quantity) {
        throw InsufficientStockException('Not enough stock');
    }

    await _uow.products.updateStock(item.productId, -item.quantity);
}

// Bestellung erstellen
return await _uow.orders.create(OrderCreate(
    customerId: customerId,
    items: items,
));
});
}
}
```

### 3.3.0.11 Dependency Injection

```
// Container für Dependencies
class Container {
    static late Pool _pool;
    static late ProductRepository _productRepository;
    static late ProductService _productService;

    static Future<void> init() async {
        _pool = Pool.withEndpoints([
            Endpoint(
                host: 'localhost',
                database: 'shop_db',
                username: 'postgres',
                password: 'secret',
            ),
        ]);

        _productRepository = PostgresProductRepository(_pool);
        _productService = ProductService(_productRepository);
    }

    static ProductService get productService => _productService;

    static Future<void> dispose() async {
        await _pool.close();
    }
}

// Verwendung
void main() async {
    await Container.init();
}
```

```

final products = await Container.productService.getAllProducts();

await Container.dispose();
}

```

### 3.3.0.12 Zusammenfassung

Konzept	Beschreibung
Repository	Abstrahiert Datenzugriff
Interface	Ermöglicht Austauschbarkeit
DTO	Trennt Input von Entity
Service	Enthält Geschäftslogik
Mock	Ermöglicht Unit Tests
Unit of Work	Koordiniert Transaktionen

### 3.3.0.13 Nächste Schritte

In der nächsten Einheit lernst du **Relationale Modellierung**: Wie du komplexe Beziehungen zwischen Tabellen modellierst und mit JOINS abfragst.

## 3.3.1 Übung

### 3.3.1.1 Ziel

Implementiere ein vollständiges Repository-System für einen Online-Shop mit Products, Categories und Customers.

### 3.3.1.2 Aufgabe 1: Models erstellen (15 min)

Erstelle die Model-Klassen im Ordner `lib/models/`.

Product

```

// lib/models/product.dart
class Product {
  final int id;
  final String name;
  final String? description;
  final double price;
  final int stock;
  final int? categoryId;
  final DateTime createdAt;
  final DateTime? updatedAt;

  // TODO: Konstruktor
  // TODO: factory Product.fromRow(ResultRow row)
  // TODO: Map<String, dynamic> toJson()
  // TODO: copyWith method
}

```

Category

```
// lib/models/category.dart
class Category {
  final int id;
  final String name;
  final String? description;

  // TODO: Implementieren
}
```

Customer

```
// lib/models/customer.dart
class Customer {
  final int id;
  final String name;
  final String email;
  final DateTime createdAt;

  // TODO: Implementieren
}
```

### 3.3.1.3 Aufgabe 2: DTOs erstellen (10 min)

ProductCreate

```
// lib/dtos/product_create.dart
class ProductCreate {
  final String name;
  final String? description;
  final double price;
  final int stock;
  final int? categoryId;

  // TODO: Konstruktor mit required fields
  // TODO: factory fromJson(Map<String, dynamic> json)
  // TODO: Validierung in einem validate() method
}
```

ProductUpdate

```
// lib/dtos/product_update.dart
class ProductUpdate {
  final String? name;
  final String? description;
  final double? price;
  final int? stock;
  final int? categoryId;

  // TODO: Alle Felder optional
  // TODO: bool get isEmpty
  // TODO: factory fromJson
}
```

### 3.3.1.4 Aufgabe 3: Repository Interfaces (10 min)

Base Repository

```
// lib/repositories/base_repository.dart
abstract class BaseRepository<T, ID> {
  Future<List<T>> findAll();
  Future<T?> findById(ID id);
  Future<bool> delete(ID id);
}
```

Product Repository

```
// lib/repositories/product_repository.dart
abstract class ProductRepository extends BaseRepository<Product, int> {
  Future<List<Product>> findByCategory(int categoryId);
  Future<List<Product>> findByPriceRange(double min, double max);
  Future<List<Product>> search(String query);
  Future<List<Product>> findLowStock(int threshold);

  Future<Product> create(ProductCreate data);
  Future<Product?> update(int id, ProductUpdate data);
  Future<bool> updateStock(int id, int delta);
}
```

Category Repository

```
// lib/repositories/category_repository.dart
abstract class CategoryRepository extends BaseRepository<Category, int> {
  Future<Category?> findByName(String name);
  Future<Category> create(String name, String? description);
  Future<Category?> update(int id, String? name, String? description);
}
```

### 3.3.1.5 Aufgabe 4: PostgreSQL Implementation (25 min)

PostgresProductRepository

```
// lib/repositories/postgres/postgres_product_repository.dart
class PostgresProductRepository implements ProductRepository {
  final Pool _pool;

  PostgresProductRepository(this._pool);

  @override
  Future<List<Product>> findAll() async {
    // TODO: SELECT * FROM products ORDER BY created_at DESC
  }

  @override
  Future<Product?> findById(int id) async {
    // TODO: SELECT mit WHERE id = @id
  }
}
```

```
@override
Future<List<Product>> findByCategory(int categoryId) async {
  // TODO: SELECT mit WHERE category_id = @categoryId
}

@override
Future<List<Product>> findByPriceRange(double min, double max) async {
  // TODO: SELECT mit BETWEEN oder >= AND <=
}

@override
Future<List<Product>> search(String query) async {
  // TODO: ILIKE Suche in name und description
}

@override
Future<List<Product>> findLowStock(int threshold) async {
  // TODO: SELECT WHERE stock < @threshold
}

@override
Future<Product> create(ProductCreate data) async {
  // TODO: INSERT ... RETURNING *
}

@override
Future<Product?> update(int id, ProductUpdate data) async {
  // TODO: UPDATE mit COALESCE für optionale Felder
}

@override
Future<bool> delete(int id) async {
  // TODO: DELETE ... affectedRows > 0
}

@override
Future<bool> updateStock(int id, int delta) async {
  // TODO: UPDATE stock = stock + @delta WHERE stock + @delta >= 0
}
}
```

#### 3.3.1.6 Aufgabe 5: Mock Repository (15 min)

```
// lib/repositories/mock/mock_product_repository.dart
class MockProductRepository implements ProductRepository {
  final List<Product> _products = [];
  int _nextId = 1;

  // TODO: Alle Methoden implementieren
  // Verwende _products als In-Memory-Speicher
}
```

```
// Test-Hilfsmethoden
void clear() {
  _products.clear();
  _nextId = 1;
}

void seed(List<Product> products) {
  _products.addAll(products);
  if (products.isNotEmpty) {
    _nextId = products.map((p) => p.id).reduce((a, b) => a > b ? a : b) + 1;
  }
}

int get count => _products.length;
}
```

### 3.3.1.7 Aufgabe 6: Service Layer (20 min)

```
// lib/services/product_service.dart
class ProductService {
  final ProductRepository _repository;
  final CategoryRepository _categoryRepository;

  ProductService(this._repository, this._categoryRepository);

  Future<List<Product>> getAllProducts() {
    return _repository.findAll();
  }

  Future<Product?> getProduct(int id) {
    return _repository.findById(id);
  }

  Future<Product> createProduct(ProductCreate data) async {
    // TODO: Validierung
    // - Name mindestens 2 Zeichen
    // - Price > 0
    // - Stock >= 0
    // - Wenn categoryId angegeben, muss Kategorie existieren

    // TODO: Repository aufrufen
  }

  Future<Product?> updateProduct(int id, ProductUpdate data) async {
    // TODO: Prüfen ob Produkt existiert
    // TODO: Validierung der neuen Werte
    // TODO: Repository aufrufen
  }
}
```

```

Future<bool> deleteProduct(int id) async {
  // TODO: Prüfen ob Produkt existiert
  // TODO: Repository aufrufen
}

Future<bool> adjustStock(int id, int delta) async {
  // TODO: Produkt laden
  // TODO: Prüfen ob neuer Stock >= 0
  // TODO: Repository aufrufen
}

Future<List<Product>> searchProducts(String query) {
  if (query.length < 2) {
    throw ValidationException('Search query must be at least 2 characters');
  }
  return _repository.search(query);
}
}

```

#### Exceptions

```

// lib/exceptions.dart
class ValidationException implements Exception {
  final String message;
  ValidationException(this.message);
  @override
  String toString() => 'ValidationException: $message';
}

class NotFoundException implements Exception {
  final String message;
  NotFoundException(this.message);
  @override
  String toString() => 'NotFoundException: $message';
}

class InsufficientStockException implements Exception {
  final String message;
  InsufficientStockException(this.message);
  @override
  String toString() => 'InsufficientStockException: $message';
}

```

#### 3.3.1.8 Aufgabe 7: Unit Tests (20 min)

```

// test/product_service_test.dart
import 'package:test/test.dart';

void main() {
  late MockProductRepository productRepo;
  late MockCategoryRepository categoryRepo;
}

```

```
late ProductService service;

setUp(() {
  productRepo = MockProductRepository();
  categoryRepo = MockCategoryRepository();
  service = ProductService(productRepo, categoryRepo);
});

group('createProduct', () {
  test('valid data creates product', () async {
    // TODO: Arrange, Act, Assert
  });

  test('empty name throws ValidationException', () async {
    // TODO
  });

  test('negative price throws ValidationException', () async {
    // TODO
  });

  test('invalid categoryId throws ValidationException', () async {
    // TODO: categoryRepo.findById returns null
  });
});

group('adjustStock', () {
  test('positive delta increases stock', () async {
    // TODO
  });

  test('negative delta decreases stock', () async {
    // TODO
  });

  test('delta causing negative stock throws', () async {
    // TODO
  });

  test('non-existent product throws NotFoundException', () async {
    // TODO
  });
});

group('searchProducts', () {
  test('short query throws ValidationException', () async {
    // TODO: query.length < 2
  });

  test('valid query returns results', () async {
    // TODO
  });
});
```

```
    });  
  });  
}
```

### 3.3.1.9 Aufgabe 8: Dependency Injection (10 min)

```
// lib/container.dart  
class Container {  
  static Pool? _pool;  
  static ProductRepository? _productRepository;  
  static CategoryRepository? _categoryRepository;  
  static ProductService? _productService;  
  
  static Future<void> init({  
    required String host,  
    required String database,  
    required String username,  
    required String password,  
    int port = 5432,  
  }) async {  
    // TODO: Pool erstellen  
    // TODO: Repositories erstellen  
    // TODO: Services erstellen  
  }  
  
  static ProductService get productService {  
    if (_productService == null) {  
      throw StateError('Container not initialized');  
    }  
    return _productService!;  
  }  
  
  // TODO: Getter für andere Services  
  
  static Future<void> dispose() async {  
    await _pool?.close();  
    _pool = null;  
    _productRepository = null;  
    _categoryRepository = null;  
    _productService = null;  
  }  
}
```

### 3.3.1.10 Aufgabe 9: Integration Test (Bonus, 15 min)

```
// test/integration/product_repository_test.dart  
import 'package:test/test.dart';  
  
void main() {
```

```

late Pool pool;
late PostgresProductRepository repository;

setUpAll(() async {
  pool = Pool.withEndpoints([
    Endpoint(
      host: 'localhost',
      database: 'shop_test_db',
      username: 'postgres',
      password: 'secret',
    ),
  ]);

  repository = PostgresProductRepository(pool);

  // Tabellen erstellen
  await pool.execute('''
    CREATE TABLE IF NOT EXISTS products (...)
  ''');
});

setUp(() async {
  // Daten vor jedem Test löschen
  await pool.execute('DELETE FROM products');
});

tearDownAll(() async {
  await pool.close();
});

test('create and findById', () async {
  final created = await repository.create(ProductCreate(
    name: 'Test Product',
    price: 99.99,
  ));

  final found = await repository.findById(created.id);

  expect(found, isNotNull);
  expect(found!.name, equals('Test Product'));
});

// TODO: Weitere Tests
}

```

### 3.3.1.11 Projektstruktur

```

lib/
+-- models/
|   +-- product.dart
|   +-- category.dart

```

```
| +-- customer.dart
+-- dtos/
| +-- product_create.dart
| +-- product_update.dart
+-- repositories/
| +-- base_repository.dart
| +-- product_repository.dart
| +-- category_repository.dart
| +-- postgres/
| | +-- postgres_product_repository.dart
| | +-- postgres_category_repository.dart
| +-- mock/
|     +-- mock_product_repository.dart
|     +-- mock_category_repository.dart
+-- services/
| +-- product_service.dart
+-- exceptions.dart
+-- container.dart

test/
+-- product_service_test.dart
+-- integration/
    +-- product_repository_test.dart
```

#### 3.3.1.12 Abgabe-Checkliste

- ☐ Product, Category, Customer Models mit fromRow
- ☐ ProductCreate und ProductUpdate DTOs
- ☐ ProductRepository Interface
- ☐ CategoryRepository Interface
- ☐ PostgresProductRepository Implementierung
- ☐ MockProductRepository Implementierung
- ☐ ProductService mit Validierung
- ☐ Custom Exceptions (Validation, NotFound, InsufficientStock)
- ☐ Mindestens 5 Unit Tests für ProductService
- ☐ Container für Dependency Injection
- ☐ (Bonus) Integration Tests für Repository

### 3.3.2 Lösung

#### 3.3.2.1 Projektstruktur

```
lib/
+-- models/
| +-- product.dart
| +-- category.dart
| +-- customer.dart
+-- dtos/
| +-- product_create.dart
| +-- product_update.dart
+-- repositories/
| +-- product_repository.dart
| +-- category_repository.dart
```

```
| +-- postgres/
| | +-- postgres_product_repository.dart
| | +-- postgres_category_repository.dart
| +-- mock/
|     +-- mock_product_repository.dart
|     +-- mock_category_repository.dart
+-- services/
|   +-- product_service.dart
+-- exceptions.dart
+-- container.dart
```

### 3.3.2.2 Models

lib/models/product.dart

```
import 'package:postgres/postgres.dart';

class Product {
  final int id;
  final String name;
  final String? description;
  final double price;
  final int stock;
  final int? categoryId;
  final DateTime createdAt;
  final DateTime? updatedAt;

  Product({
    required this.id,
    required this.name,
    this.description,
    required this.price,
    required this.stock,
    this.categoryId,
    required this.createdAt,
    this.updatedAt,
  });

  factory Product.fromRow(ResultRow row) {
    final map = row.toColumnMap();
    return Product(
      id: map['id'] as int,
      name: map['name'] as String,
      description: map['description'] as String?,
      price: (map['price'] as num).toDouble(),
      stock: map['stock'] as int,
      categoryId: map['category_id'] as int?,
      createdAt: map['created_at'] as DateTime,
      updatedAt: map['updated_at'] as DateTime?,
    );
  }
}
```

```

Map<String, dynamic> toJson() => {
  'id': id,
  'name': name,
  'description': description,
  'price': price,
  'stock': stock,
  'categoryId': categoryId,
  'inStock': stock > 0,
  'createdAt': createdAt.toIso8601String(),
  'updatedAt': updatedAt?.toIso8601String(),
};

Product copyWith({
  int? id,
  String? name,
  String? description,
  double? price,
  int? stock,
  int? categoryId,
  DateTime? createdAt,
  DateTime? updatedAt,
}) {
  return Product(
    id: id ?? this.id,
    name: name ?? this.name,
    description: description ?? this.description,
    price: price ?? this.price,
    stock: stock ?? this.stock,
    categoryId: categoryId ?? this.categoryId,
    createdAt: createdAt ?? this.createdAt,
    updatedAt: updatedAt ?? this.updatedAt,
  );
}

@override
String toString() => 'Product($id: $name, €$price, stock: $stock)';
}

```

lib/models/category.dart

```

import 'package:postgres/postgres.dart';

class Category {
  final int id;
  final String name;
  final String? description;

  Category({
    required this.id,
    required this.name,
    this.description,
  }) {}
}

```

```
});

factory Category.fromRow(ResultRow row) {
  final map = row.toColumnMap();
  return Category(
    id: map['id'] as int,
    name: map['name'] as String,
    description: map['description'] as String?,
  );
}

Map<String, dynamic> toJson() => {
  'id': id,
  'name': name,
  'description': description,
};
}
```

### 3.3.2.3 DTOs

lib/dtos/product\_create.dart

```
class ProductCreate {
  final String name;
  final String? description;
  final double price;
  final int stock;
  final int? categoryId;

  ProductCreate({
    required this.name,
    this.description,
    required this.price,
    this.stock = 0,
    this.categoryId,
  });

  factory ProductCreate.fromJson(Map<String, dynamic> json) {
    return ProductCreate(
      name: json['name'] as String,
      description: json['description'] as String?,
      price: (json['price'] as num).toDouble(),
      stock: json['stock'] as int? ?? 0,
      categoryId: json['categoryId'] as int?,
    );
  }

  List<String> validate() {
    final errors = <String>[];

    if (name.trim().length < 2) {
```

```
        errors.add('Name must be at least 2 characters');
    }
    if (name.length > 100) {
        errors.add('Name must be at most 100 characters');
    }
    if (price <= 0) {
        errors.add('Price must be positive');
    }
    if (stock < 0) {
        errors.add('Stock cannot be negative');
    }

    return errors;
}
}
```

lib/dtos/product\_update.dart

```
class ProductUpdate {
    final String? name;
    final String? description;
    final double? price;
    final int? stock;
    final int? categoryId;

    ProductUpdate({
        this.name,
        this.description,
        this.price,
        this.stock,
        this.categoryId,
    });

    factory ProductUpdate.fromJson(Map<String, dynamic> json) {
        return ProductUpdate(
            name: json['name'] as String?,
            description: json['description'] as String?,
            price: (json['price'] as num?)?.toDouble(),
            stock: json['stock'] as int?,
            categoryId: json['categoryId'] as int?,
        );
    }

    bool get isEmpty =>
        name == null &&
        description == null &&
        price == null &&
        stock == null &&
        categoryId == null;

    List<String> validate() {
```

```
final errors = <String>[];

if (name != null && name!.trim().length < 2) {
  errors.add('Name must be at least 2 characters');
}
if (price != null && price! <= 0) {
  errors.add('Price must be positive');
}
if (stock != null && stock! < 0) {
  errors.add('Stock cannot be negative');
}

return errors;
}
}
```

### 3.3.2.4 Repository Interfaces

lib/repositories/product\_repository.dart

```
import '../models/product.dart';
import '../dtos/product_create.dart';
import '../dtos/product_update.dart';

abstract class ProductRepository {
  Future<List<Product>> findAll();
  Future<Product?> findById(int id);
  Future<List<Product>> findByCategory(int categoryId);
  Future<List<Product>> findByPriceRange(double min, double max);
  Future<List<Product>> search(String query);
  Future<List<Product>> findLowStock(int threshold);

  Future<Product> create(ProductCreate data);
  Future<Product?> update(int id, ProductUpdate data);
  Future<bool> delete(int id);
  Future<bool> updateStock(int id, int delta);
}
```

lib/repositories/category\_repository.dart

```
import '../models/category.dart';

abstract class CategoryRepository {
  Future<List<Category>> findAll();
  Future<Category?> findById(int id);
  Future<Category?> findByName(String name);
  Future<Category> create(String name, String? description);
  Future<Category?> update(int id, String? name, String? description);
  Future<bool> delete(int id);
}
```

### 3.3.2.5 PostgreSQL Implementations

lib/repositories/postgres/postgres\_product\_repository.dart

```
import 'package:postgres/postgres.dart';
import '../models/product.dart';
import '../dtos/product_create.dart';
import '../dtos/product_update.dart';
import '../product_repository.dart';

class PostgresProductRepository implements ProductRepository {
  final Pool _pool;

  PostgresProductRepository(this._pool);

  @override
  Future<List<Product>> findAll() async {
    final result = await _pool.execute(
      'SELECT * FROM products ORDER BY created_at DESC',
    );
    return result.map(Product.fromRow).toList();
  }

  @override
  Future<Product?> findById(int id) async {
    final result = await _pool.execute(
      Sql.named('SELECT * FROM products WHERE id = @id'),
      parameters: {'id': id},
    );
    if (result.isEmpty) return null;
    return Product.fromRow(result.first);
  }

  @override
  Future<List<Product>> findByCategoryId(int categoryId) async {
    final result = await _pool.execute(
      Sql.named('''
        SELECT * FROM products
        WHERE category_id = @categoryId
        ORDER BY name
      '''),
      parameters: {'categoryId': categoryId},
    );
    return result.map(Product.fromRow).toList();
  }

  @override
  Future<List<Product>> findByPriceRange(double min, double max) async {
    final result = await _pool.execute(
      Sql.named('''
        SELECT * FROM products
        WHERE price >= @min AND price <= @max
      '''),
      parameters: {'min': min, 'max': max},
    );
    return result.map(Product.fromRow).toList();
  }
}
```

```
        ORDER BY price
    '''),
    parameters: {'min': min, 'max': max},
);
return result.map(Product.fromRow).toList();
}

@override
Future<List<Product>> search(String query) async {
    final result = await _pool.execute(
        Sql.named('''
            SELECT * FROM products
            WHERE name ILIKE @query OR description ILIKE @query
            ORDER BY name
        '''),
        parameters: {'query': '%$query%'},
    );
    return result.map(Product.fromRow).toList();
}

@override
Future<List<Product>> findLowStock(int threshold) async {
    final result = await _pool.execute(
        Sql.named('''
            SELECT * FROM products
            WHERE stock < @threshold
            ORDER BY stock ASC
        '''),
        parameters: {'threshold': threshold},
    );
    return result.map(Product.fromRow).toList();
}

@override
Future<Product> create(ProductCreate data) async {
    final result = await _pool.execute(
        Sql.named('''
            INSERT INTO products (name, description, price, stock, category_id)
            VALUES (@name, @description, @price, @stock, @categoryId)
            RETURNING *
        '''),
        parameters: {
            'name': data.name,
            'description': data.description,
            'price': data.price,
            'stock': data.stock,
            'categoryId': data.categoryId,
        },
    );
    return Product.fromRow(result.first);
}
```

```
@override
Future<Product?> update(int id, ProductUpdate data) async {
  final result = await _pool.execute(
    Sql.named('''
      UPDATE products SET
        name = COALESCE(@name, name),
        description = COALESCE(@description, description),
        price = COALESCE(@price, price),
        stock = COALESCE(@stock, stock),
        category_id = COALESCE(@categoryId, category_id),
        updated_at = NOW()
      WHERE id = @id
      RETURNING *
    '''),
    parameters: {
      'id': id,
      'name': data.name,
      'description': data.description,
      'price': data.price,
      'stock': data.stock,
      'categoryId': data.categoryId,
    },
  );
  if (result.isEmpty) return null;
  return Product.fromRow(result.first);
}

@override
Future<bool> delete(int id) async {
  final result = await _pool.execute(
    Sql.named('DELETE FROM products WHERE id = @id'),
    parameters: {'id': id},
  );
  return result.affectedRows > 0;
}

@override
Future<bool> updateStock(int id, int delta) async {
  final result = await _pool.execute(
    Sql.named('''
      UPDATE products
      SET stock = stock + @delta, updated_at = NOW()
      WHERE id = @id AND stock + @delta >= 0
    '''),
    parameters: {'id': id, 'delta': delta},
  );
  return result.affectedRows > 0;
}
}
```

### 3.3.2.6 Mock Implementations

lib/repositories/mock/mock\_product\_repository.dart

```
import '../..models/product.dart';
import '../..dtos/product_create.dart';
import '../..dtos/product_update.dart';
import '../product_repository.dart';

class MockProductRepository implements ProductRepository {
  final List<Product> _products = [];
  int _nextId = 1;

  @override
  Future<List<Product>> findAll() async {
    return List.from(_products)
      ..sort((a, b) => b.createdAt.compareTo(a.createdAt));
  }

  @override
  Future<Product?> findById(int id) async {
    try {
      return _products.firstWhere((p) => p.id == id);
    } catch (_) {
      return null;
    }
  }

  @override
  Future<List<Product>> findByCategory(int categoryId) async {
    return _products
      .where((p) => p.categoryId == categoryId)
      .toList()
      ..sort((a, b) => a.name.compareTo(b.name));
  }

  @override
  Future<List<Product>> findByPriceRange(double min, double max) async {
    return _products
      .where((p) => p.price >= min && p.price <= max)
      .toList()
      ..sort((a, b) => a.price.compareTo(b.price));
  }

  @override
  Future<List<Product>> search(String query) async {
    final q = query.toLowerCase();
    return _products
      .where((p) =>
        p.name.toLowerCase().contains(q) ||
        (p.description?.toLowerCase().contains(q) ?? false))
      .toList();
  }
}
```

```
        ..sort((a, b) => a.name.compareTo(b.name));
    }

    @override
    Future<List<Product>> findLowStock(int threshold) async {
        return _products.where((p) => p.stock < threshold).toList()
            ..sort((a, b) => a.stock.compareTo(b.stock));
    }

    @override
    Future<Product> create(ProductCreate data) async {
        final product = Product(
            id: _nextId++,
            name: data.name,
            description: data.description,
            price: data.price,
            stock: data.stock,
            categoryId: data.categoryId,
            createdAt: DateTime.now(),
        );
        _products.add(product);
        return product;
    }

    @override
    Future<Product?> update(int id, ProductUpdate data) async {
        final index = _products.indexWhere((p) => p.id == id);
        if (index == -1) return null;

        final existing = _products[index];
        final updated = existing.copyWith(
            name: data.name ?? existing.name,
            description: data.description ?? existing.description,
            price: data.price ?? existing.price,
            stock: data.stock ?? existing.stock,
            categoryId: data.categoryId ?? existing.categoryId,
            updatedAt: DateTime.now(),
        );

        _products[index] = updated;
        return updated;
    }

    @override
    Future<bool> delete(int id) async {
        final lengthBefore = _products.length;
        _products.removeWhere((p) => p.id == id);
        return _products.length < lengthBefore;
    }

    @override
```

```

Future<bool> updateStock(int id, int delta) async {
  final product = await findById(id);
  if (product == null) return false;

  final newStock = product.stock + delta;
  if (newStock < 0) return false;

  await update(id, ProductUpdate(stock: newStock));
  return true;
}

// Test helpers
void clear() {
  _products.clear();
  _nextId = 1;
}

void seed(List<Product> products) {
  _products.addAll(products);
  if (products.isNotEmpty) {
    _nextId = products.map((p) => p.id).reduce((a, b) => a > b ? a : b) + 1;
  }
}

int get count => _products.length;
}

```

### 3.3.2.7 Exceptions

lib/exceptions.dart

```

class ValidationException implements Exception {
  final String message;
  final List<String> errors;

  ValidationException(this.message, [this.errors = const []]);

  @override
  String toString() => 'ValidationException: $message';
}

class NotFoundException implements Exception {
  final String message;

  NotFoundException(this.message);

  @override
  String toString() => 'NotFoundException: $message';
}

class InsufficientStockException implements Exception {

```

```
final String message;

InsufficientStockException(this.message);

@override
String toString() => 'InsufficientStockException: $message';
}
```

### 3.3.2.8 Service Layer

lib/services/product\_service.dart

```
import '../models/product.dart';
import '../dtos/product_create.dart';
import '../dtos/product_update.dart';
import '../repositories/product_repository.dart';
import '../repositories/category_repository.dart';
import '../exceptions.dart';

class ProductService {
  final ProductRepository _productRepository;
  final CategoryRepository _categoryRepository;

  ProductService(this._productRepository, this._categoryRepository);

  Future<List<Product>> getAllProducts() {
    return _productRepository.findAll();
  }

  Future<Product?> getProduct(int id) {
    return _productRepository.findById(id);
  }

  Future<Product> createProduct(ProductCreate data) async {
    // Validate
    final errors = data.validate();
    if (errors.isNotEmpty) {
      throw ValidationException('Invalid product data', errors);
    }

    // Check category exists if specified
    if (data.categoryId != null) {
      final category = await _categoryRepository.findById(data.categoryId!);
      if (category == null) {
        throw ValidationException('Category not found: ${data.categoryId}');
      }
    }

    return _productRepository.create(data);
  }
}
```

```
Future<Product?> updateProduct(int id, ProductUpdate data) async {
    if (data.isEmpty) {
        throw ValidationException('No data to update');
    }

    // Check product exists
    final existing = await _productRepository.findById(id);
    if (existing == null) {
        throw NotFoundException('Product not found: $id');
    }

    // Validate update data
    final errors = data.validate();
    if (errors.isNotEmpty) {
        throw ValidationException('Invalid update data', errors);
    }

    // Check category if changing
    if (data.categoryId != null) {
        final category = await _categoryRepository.findById(data.categoryId!);
        if (category == null) {
            throw ValidationException('Category not found: ${data.categoryId}');
        }
    }

    return _productRepository.update(id, data);
}

Future<bool> deleteProduct(int id) async {
    final existing = await _productRepository.findById(id);
    if (existing == null) {
        throw NotFoundException('Product not found: $id');
    }

    return _productRepository.delete(id);
}

Future<bool> adjustStock(int id, int delta) async {
    final product = await _productRepository.findById(id);
    if (product == null) {
        throw NotFoundException('Product not found: $id');
    }

    final newStock = product.stock + delta;
    if (newStock < 0) {
        throw InsufficientStockException(
            'Insufficient stock: have ${product.stock}, need ${-delta}',
        );
    }

    return _productRepository.updateStock(id, delta);
}
```

```
}

Future<List<Product>> searchProducts(String query) {
  if (query.trim().length < 2) {
    throw ValidationException('Search query must be at least 2 characters');
  }
  return _productRepository.search(query.trim());
}

Future<List<Product>> getLowStockProducts({int threshold = 10}) {
  return _productRepository.findLowStock(threshold);
}
}
```

### 3.3.2.9 Unit Tests

test/product\_service\_test.dart

```
import 'package:test/test.dart';
import 'package:shop/models/product.dart';
import 'package:shop/dtos/product_create.dart';
import 'package:shop/dtos/product_update.dart';
import 'package:shop/repositories/mock/mock_product_repository.dart';
import 'package:shop/repositories/mock/mock_category_repository.dart';
import 'package:shop/services/product_service.dart';
import 'package:shop/exceptions.dart';

void main() {
  late MockProductRepository productRepo;
  late MockCategoryRepository categoryRepo;
  late ProductService service;

  setUp(() {
    productRepo = MockProductRepository();
    categoryRepo = MockCategoryRepository();
    service = ProductService(productRepo, categoryRepo);
  });

  group('createProduct', () {
    test('valid data creates product', () async {
      final data = ProductCreate(
        name: 'Test Product',
        price: 99.99,
        stock: 10,
      );

      final product = await service.createProduct(data);

      expect(product.name, equals('Test Product'));
      expect(product.price, equals(99.99));
      expect(product.stock, equals(10));
    });
  });
}
```

```
    expect(product.id, isPositive);
  });

test('short name throws ValidationException', () async {
  final data = ProductCreate(name: 'A', price: 10);

  expect(
    () => service.createProduct(data),
    throwsA(isA<ValidationException>()),
  );
});

test('negative price throws ValidationException', () async {
  final data = ProductCreate(name: 'Test', price: -10);

  expect(
    () => service.createProduct(data),
    throwsA(isA<ValidationException>()),
  );
});

test('invalid categoryId throws ValidationException', () async {
  final data = ProductCreate(
    name: 'Test',
    price: 10,
    categoryId: 999,
  );

  expect(
    () => service.createProduct(data),
    throwsA(isA<ValidationException>()),
  );
});

group('updateProduct', () {
  test('valid update succeeds', () async {
    // Arrange
    await productRepo.create(ProductCreate(name: 'Test', price: 10));

    // Act
    final updated = await service.updateProduct(
      1,
      ProductUpdate(price: 20),
    );

    // Assert
    expect(updated?.price, equals(20));
  });

  test('empty update throws ValidationException', () async {
```

```
    await productRepo.create(ProductCreate(name: 'Test', price: 10));

    expect(
      () => service.updateProduct(1, ProductUpdate()),
      throwsA(isA<ValidationException>()),
    );
  });

  test('non-existent product throws NotFoundException', () async {
    expect(
      () => service.updateProduct(999, ProductUpdate(price: 20)),
      throwsA(isA<NotFoundException>()),
    );
  });
});

group('adjustStock', () {
  test('positive delta increases stock', () async {
    await productRepo.create(ProductCreate(name: 'Test', price: 10, stock: ↵
    ↵ 50));

    final success = await service.adjustStock(1, 10);

    expect(success, isTrue);
    final product = await productRepo.findById(1);
    expect(product?.stock, equals(60));
  });

  test('negative delta decreases stock', () async {
    await productRepo.create(ProductCreate(name: 'Test', price: 10, stock: ↵
    ↵ 50));

    final success = await service.adjustStock(1, -10);

    expect(success, isTrue);
    final product = await productRepo.findById(1);
    expect(product?.stock, equals(40));
  });

  test('delta causing negative stock throws', () async {
    await productRepo.create(ProductCreate(name: 'Test', price: 10, stock: 5));

    expect(
      () => service.adjustStock(1, -10),
      throwsA(isA<InsufficientStockException>()),
    );
  });

  test('non-existent product throws NotFoundException', () async {
    expect(
      () => service.adjustStock(999, 10),
```

```

        throwsA(isA<NotFoundException>()),
    );
  });
});

group('searchProducts', () {
  test('short query throws ValidationException', () async {
    expect(
      () => service.searchProducts('a'),
      throwsA(isA<ValidationException>()),
    );
  });

  test('valid query returns matching products', () async {
    await productRepo.create(ProductCreate(name: 'Laptop Pro', price: 1000));
    await productRepo.create(ProductCreate(name: 'Mouse', price: 50));

    final results = await service.searchProducts('Laptop');

    expect(results, hasLength(1));
    expect(results.first.name, equals('Laptop Pro'));
  });
});

group('deleteProduct', () {
  test('existing product is deleted', () async {
    await productRepo.create(ProductCreate(name: 'Test', price: 10));

    final success = await service.deleteProduct(1);

    expect(success, isTrue);
    expect(productRepo.count, equals(0));
  });

  test('non-existent product throws NotFoundException', () async {
    expect(
      () => service.deleteProduct(999),
      throwsA(isA<NotFoundException>()),
    );
  });
});
}

```

### 3.3.2.10 Container

lib/container.dart

```

import 'package:postgres/postgres.dart';
import 'repositories/product_repository.dart';
import 'repositories/category_repository.dart';
import 'repositories/postgres/postgres_product_repository.dart';

```

```
import 'repositories/postgres/postgres_category_repository.dart';
import 'services/product_service.dart';

class Container {
  static Pool? _pool;
  static ProductRepository? _productRepository;
  static CategoryRepository? _categoryRepository;
  static ProductService? _productService;

  static Future<void> init({
    required String host,
    required String database,
    required String username,
    required String password,
    int port = 5432,
  }) async {
    _pool = Pool.withEndpoints(
      [
        Endpoint(
          host: host,
          port: port,
          database: database,
          username: username,
          password: password,
        ),
      ],
      settings: PoolSettings(
        maxConnectionCount: 10,
        sslMode: SslMode.disable,
      ),
    );

    _productRepository = PostgresProductRepository(_pool!);
    _categoryRepository = PostgresCategoryRepository(_pool!);
    _productService = ProductService(_productRepository!, _categoryRepository!);
  }

  static ProductRepository get productRepository {
    if (_productRepository == null) {
      throw StateError('Container not initialized');
    }
    return _productRepository!;
  }

  static CategoryRepository get categoryRepository {
    if (_categoryRepository == null) {
      throw StateError('Container not initialized');
    }
    return _categoryRepository!;
  }
}
```

```
static ProductService get productService {
  if (_productService == null) {
    throw StateError('Container not initialized');
  }
  return _productService!;
}

static Future<void> dispose() async {
  await _pool?.close();
  _pool = null;
  _productRepository = null;
  _categoryRepository = null;
  _productService = null;
}
}
```

### 3.3.2.11 Verwendung

```
import 'package:shop/container.dart';
import 'package:shop/dtos/product_create.dart';

Future<void> main() async {
  await Container.init(
    host: 'localhost',
    database: 'shop_db',
    username: 'postgres',
    password: 'secret',
  );

  try {
    final service = Container.productService;

    // Produkt erstellen
    final product = await service.createProduct(ProductCreate(
      name: 'New Laptop',
      price: 1299.99,
      stock: 25,
    ));
    print('Created: $product');

    // Alle Produkte
    final products = await service.getAllProducts();
    print('Total products: ${products.length}');

    // Suchen
    final results = await service.searchProducts('Laptop');
    print('Search results: ${results.length}');

  } finally {
    await Container.dispose();
  }
}
```

```
}  
}
```

### 3.3.3 Ressourcen

#### 3.3.3.1 Konzepte

- Repository Pattern (Martin Fowler)
- Unit of Work Pattern
- Data Transfer Object (DTO)

#### 3.3.3.2 Cheat Sheet: Repository Interface

```
// Generisches Interface  
abstract class Repository<T, ID> {  
    Future<List<T>> findAll();  
    Future<T?> findById(ID id);  
    Future<T> create(T entity);  
    Future<T?> update(ID id, T entity);  
    Future<bool> delete(ID id);  
}  
  
// Spezifisches Interface  
abstract class ProductRepository {  
    Future<List<Product>> findAll();  
    Future<Product?> findById(int id);  
    Future<List<Product>> findByCategory(int categoryId);  
    Future<List<Product>> search(String query);  
  
    Future<Product> create(ProductCreate data);  
    Future<Product?> update(int id, ProductUpdate data);  
    Future<bool> delete(int id);  
}
```

#### 3.3.3.3 Cheat Sheet: DTOs

```
// Create DTO - required fields  
class ProductCreate {  
    final String name;  
    final double price;  
    final int stock;  
  
    ProductCreate({  
        required this.name,  
        required this.price,  
        this.stock = 0,  
    });  
  
    factory ProductCreate.fromJson(Map<String, dynamic> json) =>  
        ProductCreate(  
            name: json['name'],  
            price: json['price'],  
            stock: json['stock'] ?? 0,  
        );  
}
```

```

        name: json['name'],
        price: json['price'],
        stock: json['stock'] ?? 0,
    );
}

// Update DTO - all optional
class ProductUpdate {
    final String? name;
    final double? price;
    final int? stock;

    ProductUpdate({this.name, this.price, this.stock});

    bool get isEmpty => name == null && price == null && stock == null;

    factory ProductUpdate.fromJson(Map<String, dynamic> json) =>
        ProductUpdate(
            name: json['name'],
            price: json['price'],
            stock: json['stock'],
        );
}

```

### 3.3.3.4 Cheat Sheet: PostgreSQL Repository

```

class PostgresProductRepository implements ProductRepository {
    final Pool _pool;

    PostgresProductRepository(this._pool);

    @override
    Future<List<Product>> findAll() async {
        final result = await _pool.execute('SELECT * FROM products');
        return result.map(Product.fromRow).toList();
    }

    @override
    Future<Product?> findById(int id) async {
        final result = await _pool.execute(
            Sql.named('SELECT * FROM products WHERE id = @id'),
            parameters: {'id': id},
        );
        if (result.isEmpty) return null;
        return Product.fromRow(result.first);
    }

    @override
    Future<Product> create(ProductCreate data) async {
        final result = await _pool.execute(

```

```

        Sql.named('''
            INSERT INTO products (name, price, stock)
            VALUES (@name, @price, @stock)
            RETURNING *
        '''),
        parameters: {
            'name': data.name,
            'price': data.price,
            'stock': data.stock,
        },
    );
    return Product.fromRow(result.first);
}

@override
Future<Product?> update(int id, ProductUpdate data) async {
    final result = await _pool.execute(
        Sql.named('''
            UPDATE products SET
                name = COALESCE(@name, name),
                price = COALESCE(@price, price),
                stock = COALESCE(@stock, stock)
            WHERE id = @id
            RETURNING *
        '''),
        parameters: {
            'id': id,
            'name': data.name,
            'price': data.price,
            'stock': data.stock,
        },
    );
    if (result.isEmpty) return null;
    return Product.fromRow(result.first);
}

@override
Future<bool> delete(int id) async {
    final result = await _pool.execute(
        Sql.named('DELETE FROM products WHERE id = @id'),
        parameters: {'id': id},
    );
    return result.affectedRows > 0;
}
}

```

### 3.3.3.5 Cheat Sheet: Mock Repository

```

class MockProductRepository implements ProductRepository {
    final List<Product> _products = [];
}

```

```
int _nextId = 1;

@override
Future<List<Product>> findAll() async => List.from(_products);

@override
Future<Product?> findById(int id) async {
  try {
    return _products.firstWhere((p) => p.id == id);
  } catch (_) {
    return null;
  }
}

@override
Future<Product> create(ProductCreate data) async {
  final product = Product(
    id: _nextId++,
    name: data.name,
    price: data.price,
    stock: data.stock,
    createdAt: DateTime.now(),
  );
  _products.add(product);
  return product;
}

// Test helpers
void clear() => _products.clear();
void seed(List<Product> products) => _products.addAll(products);
}
```

### 3.3.3.6 Cheat Sheet: Service Layer

```
class ProductService {
  final ProductRepository _repository;

  ProductService(this._repository);

  Future<Product> createProduct(ProductCreate data) async {
    // 1. Validierung
    if (data.name.length < 2) {
      throw ValidationException('Name too short');
    }
    if (data.price <= 0) {
      throw ValidationException('Price must be positive');
    }

    // 2. Geschäftslogik
    // ...
  }
}
```

```
// 3. Repository aufrufen
return _repository.create(data);
}

Future<bool> reduceStock(int id, int quantity) async {
  // 1. Produkt laden
  final product = await _repository.findById(id);
  if (product == null) {
    throw NotFoundException('Product not found');
  }

  // 2. Validierung
  if (product.stock < quantity) {
    throw InsufficientStockException('Not enough stock');
  }

  // 3. Repository aufrufen
  return _repository.updateStock(id, -quantity);
}
```

#### 3.3.3.7 Cheat Sheet: Exceptions

```
class ValidationException implements Exception {
  final String message;
  ValidationException(this.message);
}

class NotFoundException implements Exception {
  final String message;
  NotFoundException(this.message);
}

class InsufficientStockException implements Exception {
  final String message;
  InsufficientStockException(this.message);
}
```

#### 3.3.3.8 Cheat Sheet: Unit Tests

```
import 'package:test/test.dart';

void main() {
  late MockProductRepository repository;
  late ProductService service;

  setUp(() {
    repository = MockProductRepository();
  });
}
```

```

    service = ProductService(repository);
});

test('createProduct - valid data', () async {
    final data = ProductCreate(name: 'Test', price: 10);

    final product = await service.createProduct(data);

    expect(product.name, equals('Test'));
    expect(product.id, isPositive);
});

test('createProduct - invalid price - throws', () async {
    final data = ProductCreate(name: 'Test', price: -10);

    expect(
        () => service.createProduct(data),
        throwsA(isA<ValidationException>()),
    );
});
}

```

### 3.3.3.9 Cheat Sheet: Dependency Injection

```

// Simple Container
class Container {
    static late Pool _pool;
    static late ProductRepository _productRepo;
    static late ProductService _productService;

    static Future<void> init() async {
        _pool = Pool.withEndpoints([...]);
        _productRepo = PostgresProductRepository(_pool);
        _productService = ProductService(_productRepo);
    }

    static ProductService get productService => _productService;

    static Future<void> dispose() async {
        await _pool.close();
    }
}

// Verwendung
await Container.init();
final service = Container.productService;
await Container.dispose();

```

### 3.3.3.10 Vorteile des Repository Patterns

Vorteil	Beschreibung
Testbarkeit	Mock-Repositories für Unit Tests
Austauschbarkeit	Einfacher Wechsel der Datenbank
Separation	Geschäftslogik getrennt von Datenzugriff
Wiederverwendbarkeit	Repositories in mehreren Services nutzbar
Wartbarkeit	Änderungen am Schema nur im Repository

### 3.3.3.11 Best Practices

1. **Interface zuerst** - Repository Interface definieren
2. **DTOs verwenden** - Trennung von Input/Output und Entity
3. **Validierung im Service** - Nicht im Repository
4. **Mock für Tests** - Kein Datenbankzugriff in Unit Tests
5. **Dependency Injection** - Repositories als Abhängigkeiten
6. **Transaktionen** - Unit of Work für mehrere Repositories

## 3.4 Einheit 7.4: Relationale Modellierung

### 3.4.0.1 Lernziele

Nach dieser Einheit kannst du: - Beziehungen zwischen Tabellen modellieren (1:1, 1:n, n:m) - Foreign Keys und Constraints korrekt einsetzen - Komplexe JOINS für verknüpfte Daten schreiben - Normalisierung verstehen und anwenden

### 3.4.0.2 Beziehungstypen

1:1 (One-to-One)

Ein Datensatz in Tabelle A gehört zu genau einem Datensatz in Tabelle B.

```
-- Beispiel: User und UserProfile
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL
);

CREATE TABLE user_profiles (
    id SERIAL PRIMARY KEY,
    user_id INTEGER UNIQUE REFERENCES users(id),
    bio TEXT,
    avatar_url VARCHAR(255)
);
```

1:n (One-to-Many)

Ein Datensatz in Tabelle A gehört zu vielen Datensätzen in Tabelle B.

```
-- Beispiel: Category und Products
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);
```

```
CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  category_id INTEGER REFERENCES categories(id)
);

-- Eine Kategorie hat viele Produkte
-- Ein Produkt gehört zu einer Kategorie
```

n:m (Many-to-Many)

Viele Datensätze in A gehören zu vielen Datensätzen in B.

```
-- Beispiel: Products und Tags
CREATE TABLE products (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL
);

CREATE TABLE tags (
  id SERIAL PRIMARY KEY,
  name VARCHAR(30) NOT NULL UNIQUE
);

-- Zwischentabelle (Junction Table)
CREATE TABLE product_tags (
  product_id INTEGER REFERENCES products(id) ON DELETE CASCADE,
  tag_id INTEGER REFERENCES tags(id) ON DELETE CASCADE,
  PRIMARY KEY (product_id, tag_id)
);
```

### 3.4.0.3 Foreign Key Constraints

ON DELETE Optionen

```
-- CASCADE: Verknüpfte Datensätze auch löschen
REFERENCES users(id) ON DELETE CASCADE

-- SET NULL: Foreign Key auf NULL setzen
REFERENCES category(id) ON DELETE SET NULL

-- RESTRICT: Löschen verhindern (Default)
REFERENCES category(id) ON DELETE RESTRICT

-- SET DEFAULT: Auf Default-Wert setzen
REFERENCES category(id) ON DELETE SET DEFAULT
```

ON UPDATE Optionen

```
-- CASCADE: ID-Änderung übernehmen
REFERENCES users(id) ON UPDATE CASCADE

-- RESTRICT: ID-Änderung verhindern
```

```
REFERENCES users(id) ON UPDATE RESTRICT
```

### 3.4.0.4 JOINS für verknüpfte Daten

INNER JOIN

```
-- Produkte mit Kategorienamen
SELECT p.id, p.name, c.name AS category
FROM products p
INNER JOIN categories c ON p.category_id = c.id;
```

LEFT JOIN

```
-- Alle Produkte, auch ohne Kategorie
SELECT p.id, p.name, c.name AS category
FROM products p
LEFT JOIN categories c ON p.category_id = c.id;
```

Multiple JOINS

```
-- Bestellungen mit Kunde und Produkten
SELECT
    o.id AS order_id,
    c.name AS customer,
    p.name AS product,
    oi.quantity,
    oi.price
FROM orders o
JOIN customers c ON o.customer_id = c.id
JOIN order_items oi ON oi.order_id = o.id
JOIN products p ON oi.product_id = p.id;
```

n:m JOIN

```
-- Produkte mit ihren Tags
SELECT p.name AS product, t.name AS tag
FROM products p
JOIN product_tags pt ON p.id = pt.product_id
JOIN tags t ON pt.tag_id = t.id
ORDER BY p.name, t.name;
```

### 3.4.0.5 Komplexes Schema: E-Commerce

```
-- Kunden
CREATE TABLE customers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Adressen (1:n mit Customer)
```

```
CREATE TABLE addresses (  
  id SERIAL PRIMARY KEY,  
  customer_id INTEGER REFERENCES customers(id) ON DELETE CASCADE,  
  type VARCHAR(20) NOT NULL, -- 'shipping', 'billing'  
  street VARCHAR(255) NOT NULL,  
  city VARCHAR(100) NOT NULL,  
  postal_code VARCHAR(20) NOT NULL,  
  country VARCHAR(50) NOT NULL  
);  
  
-- Kategorien  
CREATE TABLE categories (  
  id SERIAL PRIMARY KEY,  
  name VARCHAR(50) NOT NULL,  
  parent_id INTEGER REFERENCES categories(id) -- Selbstreferenz für Hierarchie  
);  
  
-- Produkte  
CREATE TABLE products (  
  id SERIAL PRIMARY KEY,  
  name VARCHAR(100) NOT NULL,  
  description TEXT,  
  price DECIMAL(10, 2) NOT NULL,  
  stock INTEGER NOT NULL DEFAULT 0,  
  category_id INTEGER REFERENCES categories(id) ON DELETE SET NULL,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);  
  
-- Bestellungen  
CREATE TABLE orders (  
  id SERIAL PRIMARY KEY,  
  customer_id INTEGER REFERENCES customers(id),  
  shipping_address_id INTEGER REFERENCES addresses(id),  
  billing_address_id INTEGER REFERENCES addresses(id),  
  status VARCHAR(20) DEFAULT 'pending',  
  total DECIMAL(10, 2) NOT NULL,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);  
  
-- Bestellpositionen (n:m zwischen Order und Product)  
CREATE TABLE order_items (  
  id SERIAL PRIMARY KEY,  
  order_id INTEGER REFERENCES orders(id) ON DELETE CASCADE,  
  product_id INTEGER REFERENCES products(id),  
  quantity INTEGER NOT NULL,  
  unit_price DECIMAL(10, 2) NOT NULL  
);
```

### 3.4.0.6 Dart Models für Relationen

One-to-Many

```

class Category {
    final int id;
    final String name;
    final List<Product> products; // Lazy loading oder eager

    Category({required this.id, required this.name, this.products = const []});
}

class Product {
    final int id;
    final String name;
    final int? categoryId;
    final Category? category; // Optionale Referenz

    Product({
        required this.id,
        required this.name,
        this.categoryId,
        this.category,
    });
}

```

Many-to-Many

```

class Product {
    final int id;
    final String name;
    final List<Tag> tags;

    Product({required this.id, required this.name, this.tags = const []});
}

class Tag {
    final int id;
    final String name;

    Tag({required this.id, required this.name});
}

```

### 3.4.0.7 Repository für Relationen

```

class ProductRepository {
    final Pool _pool;

    // Produkt mit Kategorie laden
    Future<Product?> findByIdWithCategory(int id) async {
        final result = await _pool.execute(
            Sql.named('''
                SELECT
                    p.*,
                    c.id AS cat_id,

```

```
        c.name AS cat_name
    FROM products p
    LEFT JOIN categories c ON p.category_id = c.id
    WHERE p.id = @id
    '''),
    parameters: {'id': id},
);

if (result.isEmpty) return null;

final row = result.first.toColumnMap();
Category? category;
if (row['cat_id'] != null) {
    category = Category(
        id: row['cat_id'] as int,
        name: row['cat_name'] as String,
    );
}

return Product(
    id: row['id'] as int,
    name: row['name'] as String,
    categoryId: row['category_id'] as int?,
    category: category,
);
}

// Produkt mit Tags laden
Future<Product?> findByIdWithTags(int id) async {
    final productResult = await _pool.execute(
        Sql.named('SELECT * FROM products WHERE id = @id'),
        parameters: {'id': id},
    );

    if (productResult.isEmpty) return null;

    final tagResult = await _pool.execute(
        Sql.named(''
            SELECT t.*
            FROM tags t
            JOIN product_tags pt ON t.id = pt.tag_id
            WHERE pt.product_id = @id
            '''),
        parameters: {'id': id},
    );

    final tags = tagResult.map((r) {
        final m = r.toColumnMap();
        return Tag(id: m['id'] as int, name: m['name'] as String);
    }).toList();
```

```

    final row = productResult.first.toColumnMap();
    return Product(
        id: row['id'] as int,
        name: row['name'] as String,
        tags: tags,
    );
}

// Tags zu Produkt hinzufügen
Future<void> addTags(int productId, List<int> tagIds) async {
    for (final tagId in tagIds) {
        await _pool.execute(
            Sql.named('''
                INSERT INTO product_tags (product_id, tag_id)
                VALUES (@productId, @tagId)
                ON CONFLICT DO NOTHING
            '''),
            parameters: {'productId': productId, 'tagId': tagId},
        );
    }
}
}

```

### 3.4.0.8 Normalisierung

#### 1. Normalform (1NF)

- Keine Wiederholungsgruppen
- Atomare Werte

```

-- Schlecht
CREATE TABLE orders (
    id SERIAL,
    products TEXT -- "Product1, Product2, Product3"
);

-- Gut (1NF)
CREATE TABLE order_items (
    order_id INTEGER,
    product_id INTEGER
);

```

#### 2. Normalform (2NF)

- 1NF + alle Nicht-Schlüssel-Attribute voll funktional abhängig vom Primärschlüssel

#### 3. Normalform (3NF)

- 2NF + keine transitiven Abhängigkeiten

```

-- Schlecht (Kunde-Stadt impliziert Land)
CREATE TABLE customers (
    id SERIAL,
    name VARCHAR(100),

```

```

    city VARCHAR(50),
    country VARCHAR(50) -- Abhängig von city
);

-- Gut (3NF)
CREATE TABLE cities (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50),
    country VARCHAR(50)
);

CREATE TABLE customers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    city_id INTEGER REFERENCES cities(id)
);

```

### 3.4.0.9 Zusammenfassung

Beziehung	Implementierung
1:1	Foreign Key mit UNIQUE
1:n	Foreign Key
n:m	Zwischentabelle mit zwei Foreign Keys

JOIN	Verwendung
INNER	Nur übereinstimmende
LEFT	Alle links + matches
RIGHT	Alle rechts + matches

### 3.4.0.10 Nächste Schritte

In der nächsten Einheit lernst du **Migrations**: Wie du Datenbankänderungen versioniert und sicher durchführst.

## 3.4.1 Übung

### 3.4.1.1 Ziel

Modelliere ein vollständiges E-Commerce-Datenbankschema mit allen Beziehungstypen.

### 3.4.1.2 Aufgabe 1: Schema entwerfen (20 min)

Entwirf ein Schema für einen Online-Shop mit folgenden Entitäten:

Entitäten

1. **customers** - Kunden
2. **addresses** - Adressen (mehrere pro Kunde)
3. **categories** - Produktkategorien (hierarchisch)
4. **products** - Produkte

5. **tags** - Produkt-Tags (n:m)
6. **orders** - Bestellungen
7. **order\_items** - Bestellpositionen

Beziehungen

- Customer 1:n Addresses
- Category 1:n Products
- Category 1:n Categories (Selbstreferenz für Hierarchie)
- Product n:m Tags
- Customer 1:n Orders
- Order n:m Products (über order\_items)

### 3.4.1.3 Aufgabe 2: SQL-Schema erstellen (15 min)

Erstelle die CREATE TABLE Statements.

customers

```
CREATE TABLE customers (  
    -- TODO: id, name, email, created_at  
);
```

addresses

```
CREATE TABLE addresses (  
    -- TODO: id, customer_id (FK), type, street, city, postal_code, country  
    -- ON DELETE CASCADE  
);
```

categories

```
CREATE TABLE categories (  
    -- TODO: id, name, parent_id (Selbstreferenz für Hierarchie)  
);
```

products

```
CREATE TABLE products (  
    -- TODO: id, name, description, price, stock, category_id, created_at  
);
```

tags & product\_tags

```
CREATE TABLE tags (  
    -- TODO: id, name (UNIQUE)  
);  
  
CREATE TABLE product_tags (  
    -- TODO: product_id, tag_id, composite primary key  
    -- ON DELETE CASCADE für beide  
);
```

orders & order\_items

```
CREATE TABLE orders (  
    -- TODO: id, customer_id, status, total, created_at  
);  
  
CREATE TABLE order_items (  
    -- TODO: id, order_id, product_id, quantity, unit_price  
);
```

#### 3.4.1.4 Aufgabe 3: Testdaten einfügen (10 min)

```
-- Kunden  
INSERT INTO customers (name, email) VALUES  
    ('Max Müller', 'max@example.com'),  
    ('Anna Schmidt', 'anna@example.com');  
  
-- Adressen  
INSERT INTO addresses (customer_id, type, street, city, postal_code, country)  
VALUES  
    (1, 'shipping', 'Hauptstr. 1', 'Berlin', '10115', 'Germany'),  
    (1, 'billing', 'Nebenstr. 2', 'Berlin', '10115', 'Germany'),  
    (2, 'shipping', 'Musterweg 3', 'München', '80331', 'Germany');  
  
-- Kategorien (mit Hierarchie)  
INSERT INTO categories (name, parent_id) VALUES  
    ('Electronics', NULL),  
    ('Clothing', NULL),  
    ('Laptops', 1),      -- Unterkategorie von Electronics  
    ('Smartphones', 1); -- Unterkategorie von Electronics  
  
-- Produkte  
-- TODO: Mindestens 5 Produkte in verschiedenen Kategorien  
  
-- Tags  
INSERT INTO tags (name) VALUES  
    ('sale'), ('new'), ('bestseller'), ('eco-friendly');  
  
-- Product-Tags verknüpfen  
-- TODO: Einige Produkte mit Tags verknüpfen  
  
-- Bestellungen  
-- TODO: 2 Bestellungen mit je 2-3 Positionen
```

#### 3.4.1.5 Aufgabe 4: JOINS schreiben (20 min)

##### 4.1 Produkte mit Kategorien

```
-- Alle Produkte mit ihrem Kategorienamen  
-- Produkte ohne Kategorie auch anzeigen  
SELECT  
    p.id,
```

```
p.name AS product,  
-- TODO: Kategorienname  
FROM products p  
-- TODO: JOIN
```

#### 4.2 Kategorien mit Unterkategorien

```
-- Kategorien mit ihren Parent-Kategorienamen  
SELECT  
    c.id,  
    c.name AS category,  
    -- TODO: parent.name AS parent_category  
FROM categories c  
-- TODO: Self-JOIN
```

#### 4.3 Produkte mit Tags

```
-- Alle Produkte mit ihren Tags (als comma-separated list)  
SELECT  
    p.name AS product,  
    STRING_AGG(t.name, ', ') AS tags  
FROM products p  
-- TODO: JOINS über product_tags  
GROUP BY p.id, p.name;
```

#### 4.4 Kunden mit Bestellübersicht

```
-- Kunden mit Anzahl Bestellungen und Gesamtumsatz  
SELECT  
    c.name AS customer,  
    COUNT(o.id) AS order_count,  
    COALESCE(SUM(o.total), 0) AS total_spent  
FROM customers c  
-- TODO: LEFT JOIN orders  
GROUP BY c.id, c.name;
```

#### 4.5 Bestelldetails

```
-- Vollständige Bestellübersicht  
SELECT  
    o.id AS order_id,  
    c.name AS customer,  
    o.status,  
    p.name AS product,  
    oi.quantity,  
    oi.unit_price,  
    oi.quantity * oi.unit_price AS line_total  
FROM orders o  
-- TODO: JOINS für customer, order_items, products  
ORDER BY o.id, p.name;
```

### 3.4.1.6 Aufgabe 5: Dart Models (15 min)

Customer mit Adressen

```
class Customer {  
  final int id;  
  final String name;  
  final String email;  
  final List<Address> addresses;  
  
  // TODO: Konstruktor, fromRow, toJson  
}  
  
class Address {  
  final int id;  
  final int customerId;  
  final String type;  
  final String street;  
  final String city;  
  final String postalCode;  
  final String country;  
  
  // TODO: Implementieren  
}
```

Product mit Category und Tags

```
class Product {  
  final int id;  
  final String name;  
  final double price;  
  final Category? category;  
  final List<Tag> tags;  
  
  // TODO: Implementieren  
}
```

### 3.4.1.7 Aufgabe 6: Repository für Relationen (20 min)

```
class ProductRepository {  
  final Pool _pool;  
  
  // Produkt mit Kategorie laden  
  Future<Product?> findByIdWithCategory(int id) async {  
    // TODO: JOIN query  
    // TODO: Category aus Ergebnis extrahieren  
  }  
  
  // Produkt mit Tags laden  
  Future<Product?> findByIdWithTags(int id) async {  
    // TODO: Produkt laden  
    // TODO: Tags separat laden  
  }  
}
```

```

    // TODO: Kombinieren
}

// Alle Produkte einer Kategorie (inkl. Unterkategorien)
Future<List<Product>> findByCategoryRecursive(int categoryId) async {
    // TODO: Recursive CTE oder mehrere Queries
}

// Tags setzen (alle ersetzen)
Future<void> setTags(int productId, List<int> tagIds) async {
    // TODO: Alte löschen
    // TODO: Neue einfügen
}
}

```

#### 3.4.1.8 Aufgabe 7: Order Repository (Bonus, 15 min)

```

class OrderRepository {
    // Bestellung mit allen Details laden
    Future<OrderDetails?> findByIdWithDetails(int id) async {
        // TODO: Order + Customer + Items + Products
    }

    // Bestellung erstellen (Transaktion)
    Future<Order> create({
        required int customerId,
        required List<OrderItemCreate> items,
    }) async {
        // TODO: Transaktion
        // 1. Order erstellen
        // 2. Items einfügen
        // 3. Stock reduzieren
        // 4. Total berechnen und updaten
    }
}

class OrderDetails {
    final Order order;
    final Customer customer;
    final List<OrderItemWithProduct> items;
}

```

#### 3.4.1.9 Abgabe-Checkliste

- ☐ Vollständiges SQL-Schema mit allen Tabellen
- ☐ Korrekte Foreign Keys mit ON DELETE
- ☐ Testdaten eingefügt
- ☐ JOIN: Produkte mit Kategorien
- ☐ JOIN: Kategorien mit Parent
- ☐ JOIN: Produkte mit Tags (STRING\_AGG)

- ☐ JOIN: Kundenübersicht mit Aggregation
- ☐ JOIN: Bestelldetails
- ☐ Dart Models für Customer, Address, Product, Tag
- ☐ ProductRepository mit findByIdWithCategory
- ☐ ProductRepository mit findByIdWithTags
- ☐ (Bonus) OrderRepository mit Transaktion

### 3.4.2 Lösung

#### 3.4.2.1 SQL-Schema

```
-- =====
-- Customers & Addresses
-- =====

CREATE TABLE customers (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE addresses (
    id SERIAL PRIMARY KEY,
    customer_id INTEGER NOT NULL REFERENCES customers(id) ON DELETE CASCADE,
    type VARCHAR(20) NOT NULL CHECK (type IN ('shipping', 'billing')),
    street VARCHAR(255) NOT NULL,
    city VARCHAR(100) NOT NULL,
    postal_code VARCHAR(20) NOT NULL,
    country VARCHAR(50) NOT NULL DEFAULT 'Germany'
);

CREATE INDEX idx_addresses_customer ON addresses(customer_id);

-- =====
-- Categories (hierarchisch)
-- =====

CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    parent_id INTEGER REFERENCES categories(id) ON DELETE SET NULL
);

CREATE INDEX idx_categories_parent ON categories(parent_id);

-- =====
-- Products
-- =====

CREATE TABLE products (
```

```

    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10, 2) NOT NULL CHECK (price > 0),
    stock INTEGER NOT NULL DEFAULT 0 CHECK (stock >= 0),
    category_id INTEGER REFERENCES categories(id) ON DELETE SET NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_products_category ON products(category_id);

-- =====
-- Tags (n:m)
-- =====

CREATE TABLE tags (
    id SERIAL PRIMARY KEY,
    name VARCHAR(30) NOT NULL UNIQUE
);

CREATE TABLE product_tags (
    product_id INTEGER REFERENCES products(id) ON DELETE CASCADE,
    tag_id INTEGER REFERENCES tags(id) ON DELETE CASCADE,
    PRIMARY KEY (product_id, tag_id)
);

-- =====
-- Orders
-- =====

CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    customer_id INTEGER NOT NULL REFERENCES customers(id),
    shipping_address_id INTEGER REFERENCES addresses(id),
    status VARCHAR(20) DEFAULT 'pending' CHECK (
        status IN ('pending', 'confirmed', 'shipped', 'delivered', 'cancelled')
    ),
    total DECIMAL(10, 2) NOT NULL DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_orders_customer ON orders(customer_id);

CREATE TABLE order_items (
    id SERIAL PRIMARY KEY,
    order_id INTEGER NOT NULL REFERENCES orders(id) ON DELETE CASCADE,
    product_id INTEGER NOT NULL REFERENCES products(id),
    quantity INTEGER NOT NULL CHECK (quantity > 0),
    unit_price DECIMAL(10, 2) NOT NULL
);

```

```
CREATE INDEX idx_order_items_order ON order_items(order_id);
```

### 3.4.2.2 Testdaten

```
-- Kunden
INSERT INTO customers (name, email) VALUES
  ('Max Müller', 'max@example.com'),
  ('Anna Schmidt', 'anna@example.com'),
  ('Tom Weber', 'tom@example.com');

-- Adressen
INSERT INTO addresses (customer_id, type, street, city, postal_code, country) ↵
↵ VALUES
  (1, 'shipping', 'Hauptstr. 1', 'Berlin', '10115', 'Germany'),
  (1, 'billing', 'Nebenstr. 2', 'Berlin', '10115', 'Germany'),
  (2, 'shipping', 'Musterweg 3', 'München', '80331', 'Germany'),
  (3, 'shipping', 'Teststr. 5', 'Hamburg', '20095', 'Germany');

-- Kategorien (hierarchisch)
INSERT INTO categories (name, parent_id) VALUES
  ('Electronics', NULL),      -- 1
  ('Clothing', NULL),        -- 2
  ('Books', NULL),           -- 3
  ('Laptops', 1),             -- 4, Unterkategorie von Electronics
  ('Smartphones', 1),         -- 5, Unterkategorie von Electronics
  ('T-Shirts', 2);            -- 6, Unterkategorie von Clothing

-- Produkte
INSERT INTO products (name, description, price, stock, category_id) VALUES
  ('MacBook Pro', '16" Laptop', 2499.00, 15, 4),
  ('ThinkPad X1', 'Business Laptop', 1499.00, 25, 4),
  ('iPhone 15', 'Latest iPhone', 1199.00, 50, 5),
  ('Galaxy S24', 'Samsung Flagship', 999.00, 40, 5),
  ('Basic T-Shirt', 'Cotton T-Shirt', 19.99, 200, 6),
  ('Clean Code', 'Programming Book', 39.99, 100, 3),
  ('USB-C Hub', '7-in-1 Hub', 49.99, 80, 1);

-- Tags
INSERT INTO tags (name) VALUES
  ('sale'),
  ('new'),
  ('bestseller'),
  ('eco-friendly'),
  ('premium');

-- Product-Tags
INSERT INTO product_tags (product_id, tag_id) VALUES
  (1, 2), (1, 5),      -- MacBook: new, premium
  (3, 2), (3, 3),      -- iPhone: new, bestseller
  (5, 1), (5, 4),      -- T-Shirt: sale, eco-friendly
```

```

(6, 3);                                -- Clean Code: bestseller

-- Bestellungen
INSERT INTO orders (customer_id, shipping_address_id, status, total) VALUES
  (1, 1, 'delivered', 2548.99),
  (1, 1, 'shipped', 1199.00),
  (2, 3, 'pending', 59.98);

-- Bestellpositionen
INSERT INTO order_items (order_id, product_id, quantity, unit_price) VALUES
  (1, 1, 1, 2499.00),
  (1, 7, 1, 49.99),
  (2, 3, 1, 1199.00),
  (3, 5, 2, 19.99),
  (3, 6, 1, 39.99);

```

### 3.4.2.3 JOIN-Abfragen

#### 4.1 Produkte mit Kategorien

```

SELECT
  p.id,
  p.name AS product,
  p.price,
  COALESCE(c.name, 'Keine Kategorie') AS category
FROM products p
LEFT JOIN categories c ON p.category_id = c.id
ORDER BY category, p.name;

```

#### 4.2 Kategorien mit Parent

```

SELECT
  c.id,
  c.name AS category,
  parent.name AS parent_category
FROM categories c
LEFT JOIN categories parent ON c.parent_id = parent.id
ORDER BY parent.name NULLS FIRST, c.name;

```

#### 4.3 Produkte mit Tags

```

SELECT
  p.id,
  p.name AS product,
  COALESCE(STRING_AGG(t.name, ', ' ORDER BY t.name), '') AS tags
FROM products p
LEFT JOIN product_tags pt ON p.id = pt.product_id
LEFT JOIN tags t ON pt.tag_id = t.id
GROUP BY p.id, p.name
ORDER BY p.name;

```

#### 4.4 Kundenübersicht

```
SELECT
  c.id,
  c.name AS customer,
  c.email,
  COUNT(o.id) AS order_count,
  COALESCE(SUM(o.total), 0) AS total_spent,
  MAX(o.created_at) AS last_order
FROM customers c
LEFT JOIN orders o ON c.id = o.customer_id
GROUP BY c.id, c.name, c.email
ORDER BY total_spent DESC;
```

#### 4.5 Bestelldetails

```
SELECT
  o.id AS order_id,
  o.created_at AS order_date,
  c.name AS customer,
  o.status,
  p.name AS product,
  oi.quantity,
  oi.unit_price,
  oi.quantity * oi.unit_price AS line_total
FROM orders o
JOIN customers c ON o.customer_id = c.id
JOIN order_items oi ON o.id = oi.order_id
JOIN products p ON oi.product_id = p.id
ORDER BY o.id, p.name;
```

Bonus: Kategorien mit Produktanzahl (rekursiv)

```
WITH RECURSIVE category_tree AS (
  -- Basis: Root-Kategorien
  SELECT id, name, parent_id, ARRAY[id] AS path
  FROM categories
  WHERE parent_id IS NULL

  UNION ALL

  -- Rekursion: Unterkategorien
  SELECT c.id, c.name, c.parent_id, ct.path || c.id
  FROM categories c
  JOIN category_tree ct ON c.parent_id = ct.id
)
SELECT
  ct.id,
  ct.name,
  ARRAY_LENGTH(ct.path, 1) - 1 AS depth,
  COUNT(p.id) AS product_count
FROM category_tree ct
LEFT JOIN products p ON p.category_id = ct.id
GROUP BY ct.id, ct.name, ct.path
```

```
ORDER BY ct.path;
```

#### 3.4.2.4 Dart Models

```
// Customer mit Adressen
class Customer {
  final int id;
  final String name;
  final String email;
  final DateTime createdAt;
  final List<Address> addresses;

  Customer({
    required this.id,
    required this.name,
    required this.email,
    required this.createdAt,
    this.addresses = const [],
  });

  factory Customer.fromRow(ResultRow row, [List<Address>? addresses]) {
    final map = row.toColumnMap();
    return Customer(
      id: map['id'] as int,
      name: map['name'] as String,
      email: map['email'] as String,
      createdAt: map['created_at'] as DateTime,
      addresses: addresses ?? [],
    );
  }
}

class Address {
  final int id;
  final int customerId;
  final String type;
  final String street;
  final String city;
  final String postalCode;
  final String country;

  Address({
    required this.id,
    required this.customerId,
    required this.type,
    required this.street,
    required this.city,
    required this.postalCode,
    required this.country,
  });
}
```

```
factory Address.fromRow(ResultRow row) {
    final map = row.toColumnMap();
    return Address(
        id: map['id'] as int,
        customerId: map['customer_id'] as int,
        type: map['type'] as String,
        street: map['street'] as String,
        city: map['city'] as String,
        postalCode: map['postal_code'] as String,
        country: map['country'] as String,
    );
}

// Product mit Category und Tags
class Product {
    final int id;
    final String name;
    final String? description;
    final double price;
    final int stock;
    final int? categoryId;
    final Category? category;
    final List<Tag> tags;

    Product({
        required this.id,
        required this.name,
        this.description,
        required this.price,
        required this.stock,
        this.categoryId,
        this.category,
        this.tags = const [],
    });
}

class Tag {
    final int id;
    final String name;

    Tag({required this.id, required this.name});
}
```

### 3.4.2.5 Repository

```
class ProductRepository {
    final Pool _pool;
```

```
ProductRepository(this._pool);

Future<Product?> findByIdWithCategory(int id) async {
    final result = await _pool.execute(
        Sql.named('''
            SELECT
                p.*,
                c.id AS cat_id,
                c.name AS cat_name
            FROM products p
            LEFT JOIN categories c ON p.category_id = c.id
            WHERE p.id = @id
        '''),
        parameters: {'id': id},
    );

    if (result.isEmpty) return null;

    final row = result.first.toColumnMap();
    Category? category;
    if (row['cat_id'] != null) {
        category = Category(
            id: row['cat_id'] as int,
            name: row['cat_name'] as String,
        );
    }

    return Product(
        id: row['id'] as int,
        name: row['name'] as String,
        description: row['description'] as String?,
        price: (row['price'] as num).toDouble(),
        stock: row['stock'] as int,
        categoryId: row['category_id'] as int?,
        category: category,
    );
}

Future<Product?> findByIdWithTags(int id) async {
    final productResult = await _pool.execute(
        Sql.named('SELECT * FROM products WHERE id = @id'),
        parameters: {'id': id},
    );

    if (productResult.isEmpty) return null;

    final tagResult = await _pool.execute(
        Sql.named('''
            SELECT t.id, t.name
            FROM tags t
            JOIN product_tags pt ON t.id = pt.tag_id
        ''')
    );
}
```

```

        WHERE pt.product_id = @id
        ORDER BY t.name
    '''),
    parameters: {'id': id},
);

final tags = tagResult.map((r) {
    final m = r.toColumnMap();
    return Tag(id: m['id'] as int, name: m['name'] as String);
}).toList();

final row = productResult.first.toColumnMap();
return Product(
    id: row['id'] as int,
    name: row['name'] as String,
    description: row['description'] as String?,
    price: (row['price'] as num).toDouble(),
    stock: row['stock'] as int,
    tags: tags,
);
}

Future<void> setTags(int productId, List<int> tagIds) async {
    await _pool.execute(
        Sql.named('DELETE FROM product_tags WHERE product_id = @id'),
        parameters: {'id': productId},
    );

    for (final tagId in tagIds) {
        await _pool.execute(
            Sql.named(''
                INSERT INTO product_tags (product_id, tag_id)
                VALUES (@productId, @tagId)
            '''),
            parameters: {'productId': productId, 'tagId': tagId},
        );
    }
}
}

```

### 3.4.3 Ressourcen

#### 3.4.3.1 Konzepte

- Database Normalization
- SQL Joins Visualizer
- PostgreSQL Constraints

### 3.4.3.2 Cheat Sheet: Beziehungen

```
-- 1:1 (One-to-One)
CREATE TABLE user_profiles (
    user_id INTEGER UNIQUE REFERENCES users(id)
);

-- 1:n (One-to-Many)
CREATE TABLE products (
    category_id INTEGER REFERENCES categories(id)
);

-- n:m (Many-to-Many)
CREATE TABLE product_tags (
    product_id INTEGER REFERENCES products(id),
    tag_id INTEGER REFERENCES tags(id),
    PRIMARY KEY (product_id, tag_id)
);

-- Selbstreferenz (Hierarchie)
CREATE TABLE categories (
    parent_id INTEGER REFERENCES categories(id)
);
```

### 3.4.3.3 Cheat Sheet: Foreign Key Options

```
-- ON DELETE
REFERENCES table(id) ON DELETE CASCADE    -- Löscht verknüpfte
REFERENCES table(id) ON DELETE SET NULL   -- Setzt NULL
REFERENCES table(id) ON DELETE RESTRICT    -- Verhindert Löschen
REFERENCES table(id) ON DELETE SET DEFAULT

-- ON UPDATE
REFERENCES table(id) ON UPDATE CASCADE    -- Übernimmt Änderung
REFERENCES table(id) ON UPDATE RESTRICT    -- Verhindert Änderung
```

### 3.4.3.4 Cheat Sheet: JOINS

```
-- INNER JOIN
SELECT * FROM a INNER JOIN b ON a.b_id = b.id;

-- LEFT JOIN
SELECT * FROM a LEFT JOIN b ON a.b_id = b.id;

-- RIGHT JOIN
SELECT * FROM a RIGHT JOIN b ON a.b_id = b.id;

-- FULL OUTER JOIN
SELECT * FROM a FULL OUTER JOIN b ON a.b_id = b.id;
```

```
-- Multiple JOINS
SELECT *
FROM orders o
JOIN customers c ON o.customer_id = c.id
JOIN order_items oi ON o.id = oi.order_id
JOIN products p ON oi.product_id = p.id;

-- Self JOIN
SELECT c.name, parent.name AS parent
FROM categories c
LEFT JOIN categories parent ON c.parent_id = parent.id;
```

### 3.4.3.5 Cheat Sheet: Aggregation mit JOIN

```
-- Gruppieren nach Parent
SELECT
    c.name AS category,
    COUNT(p.id) AS product_count,
    AVG(p.price) AS avg_price
FROM categories c
LEFT JOIN products p ON c.id = p.category_id
GROUP BY c.id, c.name;

-- STRING_AGG für Listen
SELECT
    p.name,
    STRING_AGG(t.name, ', ') AS tags
FROM products p
LEFT JOIN product_tags pt ON p.id = pt.product_id
LEFT JOIN tags t ON pt.tag_id = t.id
GROUP BY p.id, p.name;
```

### 3.4.3.6 Cheat Sheet: Recursive CTE

```
-- Hierarchie traversieren
WITH RECURSIVE category_tree AS (
    -- Basis
    SELECT id, name, parent_id, 0 AS depth
    FROM categories WHERE parent_id IS NULL

    UNION ALL

    -- Rekursion
    SELECT c.id, c.name, c.parent_id, ct.depth + 1
    FROM categories c
    JOIN category_tree ct ON c.parent_id = ct.id
)
SELECT * FROM category_tree;
```

### 3.4.3.7 Normalformen

Form	Regel
1NF	Atomare Werte, keine Wiederholungen
2NF	1NF + volle funktionale Abhängigkeit
3NF	2NF + keine transitiven Abhängigkeiten

### 3.4.3.8 Best Practices

1. Foreign Keys immer definieren
2. ON DELETE CASCADE nur bei echten Abhängigkeiten
3. Indizes auf Foreign Keys
4. Zwischentabellen für n:m
5. Normalisierung bis 3NF
6. Denormalisierung nur für Performance

## 3.5 Einheit 7.5: Migrations

### 3.5.0.1 Lernziele

Nach dieser Einheit kannst du: - Das Konzept von Migrations verstehen - Schema-Änderungen versioniert durchführen - Migrations in Dart implementieren - Rollbacks sicher handhaben

### 3.5.0.2 Was sind Migrations?

**Migrations** sind versionierte Änderungen am Datenbankschema, die nachvollziehbar und wiederholbar sind.

Ohne Migrations

Entwickler A: "Ich habe eine neue Spalte hinzugefügt"  
 Entwickler B: "Welche? Auf welcher Tabelle?"  
 Entwickler A: "products.rating, aber auf Prod ist sie schon drin"  
 Entwickler B: "Bei mir fehlt sie noch..."

Mit Migrations

```
+-- migrations/
|   +-- 001_create_users.sql
|   +-- 002_create_products.sql
|   +-- 003_add_rating_to_products.sql
|   +-- 004_create_orders.sql
```

### 3.5.0.3 Migrations-Tabelle

```
CREATE TABLE IF NOT EXISTS migrations (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL UNIQUE,
  applied_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Diese Tabelle trackt, welche Migrations bereits ausgeführt wurden.

### 3.5.0.4 Migration-Dateien

Struktur

```
migrations/
+-- 001_create_users.up.sql
+-- 001_create_users.down.sql
+-- 002_create_products.up.sql
+-- 002_create_products.down.sql
+-- ...
```

Up-Migration (Vorwärts)

```
-- 001_create_users.up.sql
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  name VARCHAR(100) NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Down-Migration (Rollback)

```
-- 001_create_users.down.sql
DROP TABLE IF EXISTS users;
```

### 3.5.0.5 Migration Runner in Dart

```
import 'dart:io';
import 'package:postgres/postgres.dart';

class MigrationRunner {
  final Pool _pool;
  final String _migrationsPath;

  MigrationRunner(this._pool, this._migrationsPath);

  Future<void> init() async {
    await _pool.execute('''
      CREATE TABLE IF NOT EXISTS migrations (
        id SERIAL PRIMARY KEY,
        name VARCHAR(255) NOT NULL UNIQUE,
        applied_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
      )
    ''');
  }

  Future<List<String>> getAppliedMigrations() async {
    final result = await _pool.execute(
      'SELECT name FROM migrations ORDER BY name',
    );
    return result.map((r) => r[0] as String).toList();
  }
}
```

```
Future<List<String>> getPendingMigrations() async {
  final applied = await getAppliedMigrations();
  final files = Directory(_migrationsPath)
    .listSync()
    .whereType<File>()
    .where((f) => f.path.endsWith('.up.sql'))
    .map((f) => f.uri.pathSegments.last.replaceAll('.up.sql', ''))
    .toList()
    ..sort();

  return files.where((f) => !applied.contains(f)).toList();
}

Future<void> migrate() async {
  await init();
  final pending = await getPendingMigrations();

  if (pending.isEmpty) {
    print('No pending migrations.');
```

```
    return;
  }

  for (final name in pending) {
    print('Applying: $name');
    final sql = await File('${_migrationsPath}/$name.up.sql').readAsString();

    await _pool.execute(sql);
    await _pool.execute(
      Sql.named('INSERT INTO migrations (name) VALUES (@name)'),
      parameters: {'name': name},
    );

    print('Applied: $name');
  }
}

Future<void> rollback([int steps = 1]) async {
  final applied = await getAppliedMigrations();
  if (applied.isEmpty) {
    print('No migrations to rollback.');
```

```
    return;
  }

  final toRollback = applied.reversed.take(steps).toList();

  for (final name in toRollback) {
    print('Rolling back: $name');
    final sql = await File('${_migrationsPath}/$name.down.sql').readAsString();

    await _pool.execute(sql);
```

```
    await _pool.execute(
        Sql.named('DELETE FROM migrations WHERE name = @name'),
        parameters: {'name': name},
    );

    print('Rolled back: $name');
}
}
```

### 3.5.0.6 Beispiel-Migrations

001\_\_create\_\_users

```
-- 001_create_users.up.sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    name VARCHAR(100) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- 001_create_users.down.sql
DROP TABLE IF EXISTS users CASCADE;
```

002\_\_create\_\_categories

```
-- 002_create_categories.up.sql
CREATE TABLE categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL UNIQUE,
    parent_id INTEGER REFERENCES categories(id)
);

-- 002_create_categories.down.sql
DROP TABLE IF EXISTS categories CASCADE;
```

003\_\_create\_\_products

```
-- 003_create_products.up.sql
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10, 2) NOT NULL CHECK (price > 0),
    stock INTEGER NOT NULL DEFAULT 0,
    category_id INTEGER REFERENCES categories(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_products_category ON products(category_id);
```

```
-- 003_create_products.down.sql
DROP TABLE IF EXISTS products CASCADE;
```

```
004_add_rating_to_products
```

```
-- 004_add_rating_to_products.up.sql
ALTER TABLE products
ADD COLUMN rating DECIMAL(2, 1) DEFAULT 0 CHECK (rating >= 0 AND rating <= 5);

-- 004_add_rating_to_products.down.sql
ALTER TABLE products
DROP COLUMN IF EXISTS rating;
```

### 3.5.0.7 CLI-Tool

```
// bin/migrate.dart
import 'dart:io';
import 'package:postgres/postgres.dart';

Future<void> main(List<String> args) async {
  final pool = Pool.withEndpoints([
    Endpoint(
      host: Platform.environment['DB_HOST'] ?? 'localhost',
      database: Platform.environment['DB_NAME'] ?? 'shop_db',
      username: Platform.environment['DB_USER'] ?? 'postgres',
      password: Platform.environment['DB_PASSWORD'] ?? 'secret',
    ),
  ]);

  final runner = MigrationRunner(pool, 'migrations');

  try {
    if (args.isEmpty || args[0] == 'up') {
      await runner.migrate();
    } else if (args[0] == 'down') {
      final steps = args.length > 1 ? int.parse(args[1]) : 1;
      await runner.rollback(steps);
    } else if (args[0] == 'status') {
      final applied = await runner.getAppliedMigrations();
      final pending = await runner.getPendingMigrations();

      print('Applied migrations:');
      for (final m in applied) print('  [x] $m');

      print('\nPending migrations:');
      for (final m in pending) print('  o $m');
    } else {
      print('Usage: dart run bin/migrate.dart [up|down|status]');
    }
  } finally {
    await pool.close();
  }
}
```

```
    await pool.close();
  }
}
```

Verwendung

```
# Migrations ausführen
dart run bin/migrate.dart up

# Status anzeigen
dart run bin/migrate.dart status

# Rollback (1 Migration)
dart run bin/migrate.dart down

# Rollback (3 Migrations)
dart run bin/migrate.dart down 3
```

### 3.5.0.8 Best Practices

1. Migrations sind unveränderlich

```
-- FALSCH: Bestehende Migration ändern
-- 001_create_users.up.sql (geändert)

-- RICHTIG: Neue Migration erstellen
-- 005_add_phone_to_users.up.sql
ALTER TABLE users ADD COLUMN phone VARCHAR(20);
```

2. Transaktionen verwenden

```
-- 006_add_columns.up.sql
BEGIN;

ALTER TABLE products ADD COLUMN weight DECIMAL(10, 2);
ALTER TABLE products ADD COLUMN dimensions VARCHAR(50);

COMMIT;
```

3. Daten-Migrations vorsichtig

```
-- 007_normalize_emails.up.sql
UPDATE users SET email = LOWER(email);

-- 007_normalize_emails.down.sql
-- Nicht rückgängig machbar!
-- Leere Datei oder Hinweis
```

4. Aussagekräftige Namen

```
[x] 001_create_users
[x] 002_add_email_index_to_users
[x] 003_create_orders_table
```

```
[ ] 001_update
[ ] 002_fix
[ ] 003_changes
```

### 3.5.0.9 Zusammenfassung

Konzept	Beschreibung
Migration	Versionierte Schema-Änderung
Up	Vorwärts-Migration
Down	Rollback-Migration
migrations-Tabelle	Tracking der angewandten Migrations

### 3.5.0.10 Nächste Schritte

In der nächsten Einheit lernst du **MongoDB**: Eine dokumentenorientierte NoSQL-Datenbank für flexible Datenstrukturen.

## 3.5.1 Übung

### 3.5.1.1 Ziel

Implementiere ein vollständiges Migrations-System für einen Online-Shop.

### 3.5.1.2 Aufgabe 1: Migrations-Ordner erstellen (5 min)

```
mkdir -p migrations
```

Struktur:

```
project/
+-- bin/
|   +-- migrate.dart
+-- lib/
|   +-- migration_runner.dart
+-- migrations/
    +-- 001_create_users.up.sql
    +-- 001_create_users.down.sql
    +-- ...
```

### 3.5.1.3 Aufgabe 2: Basis-Migrations erstellen (20 min)

001\_create\_users

```
-- migrations/001_create_users.up.sql
-- TODO: users Tabelle mit id, email, password_hash, name, created_at

-- migrations/001_create_users.down.sql
-- TODO: DROP TABLE
```

002\_create\_categories

```
-- TODO: categories mit id, name, parent_id (Hierarchie)
```

003\_create\_products

```
-- TODO: products mit id, name, description, price, stock, category_id,
↳ created_at
-- TODO: Index auf category_id
```

004\_create\_orders

```
-- TODO: orders mit id, user_id, status, total, created_at
-- TODO: order_items mit id, order_id, product_id, quantity, unit_price
```

### 3.5.1.4 Aufgabe 3: Migration Runner implementieren (25 min)

```
// lib/migration_runner.dart

class MigrationRunner {
  final Pool _pool;
  final String _migrationsPath;

  MigrationRunner(this._pool, this._migrationsPath);

  /// Erstellt migrations-Tabelle
  Future<void> init() async {
    // TODO
  }

  /// Gibt alle angewandten Migrations zurück
  Future<List<String>> getApplied() async {
    // TODO
  }

  /// Gibt alle ausstehenden Migrations zurück
  Future<List<String>> getPending() async {
    // TODO: Dateien lesen und mit applied vergleichen
  }

  /// Führt alle ausstehenden Migrations aus
  Future<void> migrate() async {
    // TODO
  }

  /// Rollback der letzten n Migrations
  Future<void> rollback([int steps = 1]) async {
    // TODO
  }

  /// Status ausgeben
  Future<void> status() async {
    // TODO: Applied und Pending anzeigen
  }
}
```

```
}  
}
```

### 3.5.1.5 Aufgabe 4: CLI-Tool erstellen (15 min)

```
// bin/migrate.dart  
  
Future<void> main(List<String> args) async {  
  // TODO: Pool erstellen  
  // TODO: MigrationRunner erstellen  
  
  try {  
    switch (args.firstOrNull) {  
      case 'up':  
      case null:  
        // TODO: migrate()  
        break;  
      case 'down':  
        // TODO: rollback mit optionalem steps-Argument  
        break;  
      case 'status':  
        // TODO: status()  
        break;  
      default:  
        print('Usage: dart run bin/migrate.dart [up|down|status]');  
    }  
  } finally {  
    // TODO: Pool schließen  
  }  
}
```

### 3.5.1.6 Aufgabe 5: Änderungs-Migrations (15 min)

Erstelle Migrations für Schema-Änderungen:

005\_add\_rating\_to\_products

```
-- TODO: rating Spalte hinzufügen (DECIMAL 2,1)
```

006\_add\_user\_role

```
-- TODO: role Spalte zu users (VARCHAR, Default 'customer')
```

007\_create\_reviews

```
-- TODO: reviews Tabelle (user_id, product_id, rating, comment, created_at)  
-- TODO: Unique constraint auf (user_id, product_id)
```

### 3.5.1.7 Aufgabe 6: Daten-Migration (Bonus, 10 min)

```
-- 008_populate_categories.up.sql
INSERT INTO categories (name, parent_id) VALUES
  ('Electronics', NULL),
  ('Clothing', NULL),
  ('Laptops', 1),
  ('Smartphones', 1)
ON CONFLICT DO NOTHING;

-- 008_populate_categories.down.sql
DELETE FROM categories WHERE name IN ('Electronics', 'Clothing', 'Laptops',
↵ 'Smartphones');
```

### 3.5.1.8 Testen

```
# Alle Migrations ausführen
dart run bin/migrate.dart up

# Status prüfen
dart run bin/migrate.dart status

# Rollback
dart run bin/migrate.dart down
dart run bin/migrate.dart down 3

# Datenbank prüfen
psql -d shop_db -c "\dt"
psql -d shop_db -c "SELECT * FROM migrations"
```

### 3.5.1.9 Abgabe-Checkliste

- ☐ migrations-Ordner mit mindestens 7 Migrations
- ☐ Jede Migration hat up.sql und down.sql
- ☐ MigrationRunner mit init, migrate, rollback
- ☐ CLI-Tool bin/migrate.dart
- ☐ Status-Befehl zeigt applied und pending
- ☐ Rollback funktioniert korrekt
- ☐ (Bonus) Daten-Migration

## 3.5.2 Lösung

### 3.5.2.1 Migration Runner

```
// lib/migration_runner.dart
import 'dart:io';
import 'package:postgres/postgres.dart';

class MigrationRunner {
  final Pool _pool;
  final String _migrationsPath;
```

```
MigrationRunner(this._pool, this._migrationsPath);

Future<void> init() async {
  await _pool.execute('''
    CREATE TABLE IF NOT EXISTS migrations (
      id SERIAL PRIMARY KEY,
      name VARCHAR(255) NOT NULL UNIQUE,
      applied_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    )
  ''');
}

Future<List<String>> getApplied() async {
  await init();
  final result = await _pool.execute(
    'SELECT name FROM migrations ORDER BY name',
  );
  return result.map((r) => r[0] as String).toList();
}

Future<List<String>> getPending() async {
  final applied = await getApplied();
  final dir = Directory(_migrationsPath);

  if (!await dir.exists()) {
    throw Exception('Migrations directory not found: $_migrationsPath');
  }

  final files = dir
    .listSync()
    .whereType<File>()
    .where((f) => f.path.endsWith('.up.sql'))
    .map((f) => f.uri.pathSegments.last.replaceAll('.up.sql', ''))
    .toList()
    ..sort();

  return files.where((f) => !applied.contains(f)).toList();
}

Future<void> migrate() async {
  final pending = await getPending();

  if (pending.isEmpty) {
    print('[x] No pending migrations');
    return;
  }

  print('Running ${pending.length} migration(s)...\n');

  for (final name in pending) {
    await _applyMigration(name);
  }
}
```

```
}

print('\n[x] All migrations applied');
}

Future<void> _applyMigration(String name) async {
  final file = File('${_migrationsPath}/${name}.up.sql');
  if (!await file.exists()) {
    throw Exception('Migration file not found: ${name}.up.sql');
  }

  final sql = await file.readAsString();

  print('  ↑ $name');

  try {
    await _pool.execute(sql);
    await _pool.execute(
      Sql.named('INSERT INTO migrations (name) VALUES (@name)'),
      parameters: {'name': name},
    );
  } catch (e) {
    print('  [ ] Failed: $e');
    rethrow;
  }
}

Future<void> rollback([int steps = 1]) async {
  final applied = await getApplied();

  if (applied.isEmpty) {
    print('[x] No migrations to rollback');
    return;
  }

  final toRollback = applied.reversed.take(steps).toList();

  print('Rolling back ${toRollback.length} migration(s)...\n');

  for (final name in toRollback) {
    await _rollbackMigration(name);
  }

  print('\n[x] Rollback complete');
}

Future<void> _rollbackMigration(String name) async {
  final file = File('${_migrationsPath}/${name}.down.sql');
  if (!await file.exists()) {
    throw Exception('Rollback file not found: ${name}.down.sql');
  }
}
```

```

    final sql = await file.readAsString();

    print('  ↓ $name');

    try {
      await _pool.execute(sql);
      await _pool.execute(
        Sql.named('DELETE FROM migrations WHERE name = @name'),
        parameters: {'name': name},
      );
    } catch (e) {
      print('  [ ] Failed: $e');
      rethrow;
    }
  }
}

Future<void> status() async {
  final applied = await getApplied();
  final pending = await getPending();

  print('Migration Status\n');

  print('Applied (${applied.length}):');
  if (applied.isEmpty) {
    print('  (none)');
  } else {
    for (final m in applied) {
      print('  [x] $m');
    }
  }

  print('\nPending (${pending.length}):');
  if (pending.isEmpty) {
    print('  (none)');
  } else {
    for (final m in pending) {
      print('  ○ $m');
    }
  }
}
}

```

### 3.5.2.2 CLI-Tool

```

// bin/migrate.dart
import 'dart:io';
import 'package:postgres/postgres.dart';
import '../lib/migration_runner.dart';

```

```

Future<void> main(List<String> args) async {
  final pool = Pool.withEndpoints(
    [
      Endpoint(
        host: Platform.environment['DB_HOST'] ?? 'localhost',
        port: int.parse(Platform.environment['DB_PORT'] ?? '5432'),
        database: Platform.environment['DB_NAME'] ?? 'shop_db',
        username: Platform.environment['DB_USER'] ?? 'postgres',
        password: Platform.environment['DB_PASSWORD'] ?? 'secret',
      ),
    ],
    settings: PoolSettings(sslMode: SslMode.disable),
  );

  final runner = MigrationRunner(pool, 'migrations');

  try {
    final command = args.isNotEmpty ? args[0] : 'up';

    switch (command) {
      case 'up':
        await runner.migrate();
        break;

      case 'down':
        final steps = args.length > 1 ? int.parse(args[1]) : 1;
        await runner.rollback(steps);
        break;

      case 'status':
        await runner.status();
        break;

      case 'reset':
        print('Resetting database...');
        final applied = await runner.getApplied();
        await runner.rollback(applied.length);
        await runner.migrate();
        break;

      default:
        print('')
    }
  } catch (e) {
    print(e);
  }
}

```

Usage: dart run bin/migrate.dart <command>

Commands:

up	Run all pending migrations (default)
down [n]	Rollback last n migrations (default: 1)
status	Show migration status
reset	Rollback all and re-migrate

```
} catch (e) {  
  print('Error: $e');  
  exit(1);  
} finally {  
  await pool.close();  
}  
}
```

### 3.5.2.3 Migration-Dateien

001\_create\_users

```
-- migrations/001_create_users.up.sql  
CREATE TABLE users (  
  id SERIAL PRIMARY KEY,  
  email VARCHAR(255) UNIQUE NOT NULL,  
  password_hash VARCHAR(255) NOT NULL,  
  name VARCHAR(100) NOT NULL,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);  
  
CREATE INDEX idx_users_email ON users(email);
```

```
-- migrations/001_create_users.down.sql  
DROP TABLE IF EXISTS users CASCADE;
```

002\_create\_categories

```
-- migrations/002_create_categories.up.sql  
CREATE TABLE categories (  
  id SERIAL PRIMARY KEY,  
  name VARCHAR(50) NOT NULL UNIQUE,  
  parent_id INTEGER REFERENCES categories(id) ON DELETE SET NULL  
);  
  
CREATE INDEX idx_categories_parent ON categories(parent_id);
```

```
-- migrations/002_create_categories.down.sql  
DROP TABLE IF EXISTS categories CASCADE;
```

003\_create\_products

```
-- migrations/003_create_products.up.sql  
CREATE TABLE products (  
  id SERIAL PRIMARY KEY,  
  name VARCHAR(100) NOT NULL,  
  description TEXT,  
  price DECIMAL(10, 2) NOT NULL CHECK (price > 0),  
  stock INTEGER NOT NULL DEFAULT 0 CHECK (stock >= 0),  
  category_id INTEGER REFERENCES categories(id) ON DELETE SET NULL,  
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

```
CREATE INDEX idx_products_category ON products(category_id);
CREATE INDEX idx_products_price ON products(price);
```

```
-- migrations/003_create_products.down.sql
DROP TABLE IF EXISTS products CASCADE;
```

004\_create\_orders

```
-- migrations/004_create_orders.up.sql
CREATE TABLE orders (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id),
  status VARCHAR(20) DEFAULT 'pending',
  total DECIMAL(10, 2) NOT NULL DEFAULT 0,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE order_items (
  id SERIAL PRIMARY KEY,
  order_id INTEGER NOT NULL REFERENCES orders(id) ON DELETE CASCADE,
  product_id INTEGER NOT NULL REFERENCES products(id),
  quantity INTEGER NOT NULL CHECK (quantity > 0),
  unit_price DECIMAL(10, 2) NOT NULL
);

CREATE INDEX idx_orders_user ON orders(user_id);
CREATE INDEX idx_order_items_order ON order_items(order_id);
```

```
-- migrations/004_create_orders.down.sql
DROP TABLE IF EXISTS order_items CASCADE;
DROP TABLE IF EXISTS orders CASCADE;
```

005\_add\_rating\_to\_products

```
-- migrations/005_add_rating_to_products.up.sql
ALTER TABLE products
ADD COLUMN rating DECIMAL(2, 1) DEFAULT 0
CHECK (rating >= 0 AND rating <= 5);
```

```
-- migrations/005_add_rating_to_products.down.sql
ALTER TABLE products DROP COLUMN IF EXISTS rating;
```

006\_add\_user\_role

```
-- migrations/006_add_user_role.up.sql
ALTER TABLE users
ADD COLUMN role VARCHAR(20) DEFAULT 'customer'
CHECK (role IN ('customer', 'admin', 'moderator'));
```

```
-- migrations/006_add_user_role.down.sql
ALTER TABLE users DROP COLUMN IF EXISTS role;
```

007\_create\_reviews

```
-- migrations/007_create_reviews.up.sql
CREATE TABLE reviews (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  product_id INTEGER NOT NULL REFERENCES products(id) ON DELETE CASCADE,
  rating INTEGER NOT NULL CHECK (rating >= 1 AND rating <= 5),
  comment TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE (user_id, product_id)
);

CREATE INDEX idx_reviews_product ON reviews(product_id);
```

```
-- migrations/007_create_reviews.down.sql
DROP TABLE IF EXISTS reviews CASCADE;
```

#### 3.5.2.4 Test-Ausgabe

```
$ dart run bin/migrate.dart status
Migration Status
```

```
Applied (0):
(none)
```

```
Pending (7):
  ○ 001_create_users
  ○ 002_create_categories
  ○ 003_create_products
  ○ 004_create_orders
  ○ 005_add_rating_to_products
  ○ 006_add_user_role
  ○ 007_create_reviews
```

```
$ dart run bin/migrate.dart up
Running 7 migration(s)...
```

```
↑ 001_create_users
↑ 002_create_categories
↑ 003_create_products
↑ 004_create_orders
↑ 005_add_rating_to_products
↑ 006_add_user_role
↑ 007_create_reviews
```

```
[x] All migrations applied
```

```
$ dart run bin/migrate.dart down 2
Rolling back 2 migration(s)...
```

```
↓ 007_create_reviews
↓ 006_add_user_role
```

```
[x] Rollback complete
```

### 3.5.3 Ressourcen

#### 3.5.3.1 Konzepte

- Database Migrations
- Flyway - Migration Tool Konzepte
- Liquibase - Alternative

#### 3.5.3.2 Cheat Sheet: Migration-Struktur

```
migrations/
+-- 001_create_users.up.sql
+-- 001_create_users.down.sql
+-- 002_create_products.up.sql
+-- 002_create_products.down.sql
+-- ...
```

#### 3.5.3.3 Cheat Sheet: Migrations-Tabelle

```
CREATE TABLE migrations (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL UNIQUE,
  applied_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

#### 3.5.3.4 Cheat Sheet: ALTER TABLE

```
-- Spalte hinzufügen
ALTER TABLE products ADD COLUMN rating DECIMAL(2,1);

-- Spalte ändern
ALTER TABLE products ALTER COLUMN price TYPE DECIMAL(12,2);

-- Spalte löschen
ALTER TABLE products DROP COLUMN rating;

-- Spalte umbenennen
ALTER TABLE products RENAME COLUMN price TO unit_price;

-- Tabelle umbenennen
ALTER TABLE products RENAME TO items;

-- Constraint hinzufügen
ALTER TABLE products ADD CONSTRAINT chk_price CHECK (price > 0);

-- Constraint entfernen
```

```
ALTER TABLE products DROP CONSTRAINT chk_price;

-- NOT NULL hinzufügen
ALTER TABLE products ALTER COLUMN name SET NOT NULL;

-- NOT NULL entfernen
ALTER TABLE products ALTER COLUMN name DROP NOT NULL;

-- Default setzen
ALTER TABLE products ALTER COLUMN stock SET DEFAULT 0;

-- Default entfernen
ALTER TABLE products ALTER COLUMN stock DROP DEFAULT;
```

### 3.5.3.5 Cheat Sheet: Index-Operationen

```
-- Index erstellen
CREATE INDEX idx_products_category ON products(category_id);

-- Unique Index
CREATE UNIQUE INDEX idx_users_email ON users(email);

-- Index löschen
DROP INDEX IF EXISTS idx_products_category;
```

### 3.5.3.6 Best Practices

1. **Niemals Migrations ändern** - Neue Migration erstellen
2. **Transaktionen verwenden** - BEGIN/COMMIT
3. **Aussagekräftige Namen** - 003\_add\_email\_to\_users
4. **Down-Migration testen** - Rollback sollte funktionieren
5. **Kleine Schritte** - Eine Änderung pro Migration
6. **Daten-Migrations vorsichtig** - Können irreversibel sein

### 3.5.3.7 Namenskonvention

NNN\_action\_target.up.sql  
NNN\_action\_target.down.sql

Beispiele:

001\_create\_users.up.sql  
002\_create\_products.up.sql  
003\_add\_rating\_to\_products.up.sql  
004\_rename\_price\_to\_unit\_price.up.sql  
005\_add\_index\_on\_email.up.sql

## 3.6 Einheit 7.6: NoSQL mit MongoDB

### 3.6.0.1 Lernziele

Nach dieser Einheit kannst du: - Die Grundlagen von MongoDB verstehen - Dokumente mit Dart erstellen und abfragen - CRUD-Operationen mit `mongo_dart` durchführen - Wann NoSQL vs. SQL sinnvoll ist

### 3.6.0.2 Was ist MongoDB?

**MongoDB** ist eine dokumentenorientierte NoSQL-Datenbank.

SQL vs. NoSQL

SQL (PostgreSQL)	NoSQL (MongoDB)
Tabellen	Collections
Zeilen	Documents
Spalten	Fields
Schema-gebunden	Schema-flexibel
JOINS	Eingebettete Dokumente
ACID	BASE (eventual consistency)

Wann MongoDB?

- Flexible, sich ändernde Datenstrukturen
- Hierarchische Daten (verschachtelte Objekte)
- Schnelle Iteration/Prototyping
- Horizontale Skalierung nötig

Wann SQL?

- Komplexe Beziehungen zwischen Daten
- Transaktionen über mehrere Tabellen
- Strikte Datenintegrität
- Komplexe Aggregationen/JOINS

### 3.6.0.3 MongoDB Setup

Docker

```
docker run --name mongodb \
  -e MONGO_INITDB_ROOT_USERNAME=root \
  -e MONGO_INITDB_ROOT_PASSWORD=secret \
  -p 27017:27017 \
  -d mongo:7
```

Dart Package

```
dependencies:
  mongo_dart: ^0.10.0
```

### 3.6.0.4 Verbindung herstellen

```
import 'package:mongo_dart/mongo_dart.dart';

Future<void> main() async {
  // Verbindung
  final db = Db('mongodb://root:secret@localhost:27017/shop');
  await db.open();

  print('Connected to MongoDB');

  // Collection (ähnlich Tabelle)
  final products = db.collection('products');

  // ... Operationen ...

  await db.close();
}
```

### 3.6.0.5 Dokumente einfügen

Einzelnes Dokument

```
final products = db.collection('products');

await products.insertOne({
  'name': 'Laptop Pro',
  'description': 'High-end laptop',
  'price': 1299.99,
  'stock': 25,
  'category': 'electronics',
  'tags': ['new', 'bestseller'],
  'specs': {
    'cpu': 'M3 Pro',
    'ram': '16GB',
    'storage': '512GB SSD',
  },
  'createdAt': DateTime.now(),
});
```

Mehrere Dokumente

```
await products.insertMany([
  {
    'name': 'Wireless Mouse',
    'price': 49.99,
    'stock': 100,
    'category': 'electronics',
  },
  {
    'name': 'USB-C Hub',
    'price': 79.99,
    'stock': 50,
```

```
    'category': 'electronics',  
  },  
]);
```

Mit automatischer ID

```
final result = await products.insertOne({  
  'name': 'New Product',  
  'price': 99.99,  
});  
  
print('Inserted ID: ${result.id}');
```

### 3.6.0.6 Dokumente abfragen

Alle Dokumente

```
final allProducts = await products.find().toList();  
for (final p in allProducts) {  
  print('${p['name']}: €${p['price']}');  
}
```

Mit Filter

```
// Exakter Match  
final electronics = await products  
  .find(where.eq('category', 'electronics'))  
  .toList();  
  
// Preisbereich  
final affordable = await products  
  .find(where.gte('price', 50).lte('price', 200))  
  .toList();  
  
// Mehrere Bedingungen (AND)  
final inStock = await products  
  .find(where.eq('category', 'electronics').gt('stock', 0))  
  .toList();  
  
// OR-Verknüpfung  
final selected = await products  
  .find(where.oneFrom('category', ['electronics', 'books']))  
  .toList();
```

Einzelnes Dokument

```
final product = await products.findOne(where.eq('name', 'Laptop Pro'));  
if (product != null) {  
  print('Found: ${product['name']}');  
}  
  
// Nach ID  
final byId = await products.findOne(where.id(ObjectId.parse('...')));
```

Sortierung und Limit

```
// Nach Preis sortiert, Top 10
final topProducts = await products
  .find(where.sortBy('price', descending: true).limit(10))
  .toList();

// Mit Skip (Pagination)
final page2 = await products
  .find(where.sortBy('name').skip(10).limit(10))
  .toList();
```

### 3.6.0.7 Dokumente aktualisieren

Einzelnes Dokument

```
await products.updateOne(
  where.eq('name', 'Laptop Pro'),
  modify.set('price', 1199.99),
);
```

Mehrere Felder

```
await products.updateOne(
  where.eq('name', 'Laptop Pro'),
  modify
    .set('price', 1199.99)
    .set('stock', 30)
    .set('updatedAt', DateTime.now()),
);
```

Inkrementieren

```
// Stock um 10 erhöhen
await products.updateOne(
  where.eq('name', 'Laptop Pro'),
  modify.inc('stock', 10),
);

// Stock um 5 verringern
await products.updateOne(
  where.eq('name', 'Laptop Pro'),
  modify.inc('stock', -5),
);
```

Array-Operationen

```
// Tag hinzufügen
await products.updateOne(
  where.eq('name', 'Laptop Pro'),
  modify.push('tags', 'sale'),
);
```

```
// Tag entfernen
await products.updateOne(
  where.eq('name', 'Laptop Pro'),
  modify.pull('tags', 'sale'),
);
```

Mehrere Dokumente

```
// Alle Produkte in Kategorie um 10% reduzieren
await products.updateMany(
  where.eq('category', 'electronics'),
  modify.mul('price', 0.9),
);
```

### 3.6.0.8 Dokumente löschen

```
// Einzelnes Dokument
await products.deleteOne(where.eq('name', 'Old Product'));

// Mehrere Dokumente
await products.deleteMany(where.eq('stock', 0));

// Alle Dokumente (Collection leeren)
await products.deleteMany({});
```

### 3.6.0.9 Eingebettete Dokumente

MongoDB unterstützt verschachtelte Objekte nativ.

```
// Dokument mit eingebettetem Objekt
await orders.insertOne({
  'customer': {
    'name': 'Max Müller',
    'email': 'max@example.com',
    'address': {
      'street': 'Hauptstr. 1',
      'city': 'Berlin',
      'zip': '10115',
    },
  },
  'items': [
    {'productId': 'p1', 'name': 'Laptop', 'quantity': 1, 'price': 1299.99},
    {'productId': 'p2', 'name': 'Mouse', 'quantity': 2, 'price': 49.99},
  ],
  'total': 1399.97,
  'status': 'pending',
  'createdAt': DateTime.now(),
});

// Abfragen auf eingebettete Felder
```

```
final berlinOrders = await orders
    .find(where.eq('customer.address.city', 'Berlin'))
    .toList();
```

#### 3.6.0.10 Indizes

```
// Einfacher Index
await products.createIndex(keys: {'name': 1});

// Unique Index
await products.createIndex(
    keys: {'email': 1},
    unique: true,
);

// Compound Index
await products.createIndex(keys: {'category': 1, 'price': -1});

// Text Index für Suche
await products.createIndex(keys: {'r$*': 'text'});
```

#### 3.6.0.11 Repository Pattern

```
class ProductRepository {
    final DbCollection _collection;

    ProductRepository(Db db) : _collection = db.collection('products');

    Future<List<Map<String, dynamic>>> findAll() async {
        return await _collection.find().toList();
    }

    Future<Map<String, dynamic>?> findById(String id) async {
        return await _collection.findOne(where.id(ObjectId.parse(id)));
    }

    Future<String> create(Map<String, dynamic> data) async {
        data['createdAt'] = DateTime.now();
        final result = await _collection.insertOne(data);
        return result.id.toHexString();
    }

    Future<bool> update(String id, Map<String, dynamic> data) async {
        data['updatedAt'] = DateTime.now();
        final result = await _collection.updateOne(
            where.id(ObjectId.parse(id)),
            modify.set('name', data['name']).set('price', data['price']),
        );
        return result.nModified > 0;
    }
}
```

```

}

Future<bool> delete(String id) async {
  final result = await _collection.deleteOne(
    where.id(ObjectId.parse(id)),
  );
  return result.nRemoved > 0;
}
}

```

### 3.6.0.12 Zusammenfassung

SQL	MongoDB
INSERT INTO	insertOne/Many
SELECT * FROM	find()
WHERE	where.eq()
UPDATE	updateOne/Many
DELETE	deleteOne/Many
JOIN	Eingebettete Dokumente

### 3.6.0.13 Nächste Schritte

In der nächsten Einheit lernst du **Queries & Aggregationen**: Komplexe Abfragen und Datenauswertungen in beiden Datenbanksystemen.

## 3.6.1 Übung

### 3.6.1.1 Ziel

Implementiere eine Produkt-API mit MongoDB als Backend.

### 3.6.1.2 Vorbereitung

MongoDB starten

```

docker run --name mongodb \
  -e MONGO_INITDB_ROOT_USERNAME=root \
  -e MONGO_INITDB_ROOT_PASSWORD=secret \
  -p 27017:27017 \
  -d mongo:7

```

Projekt Setup

```

# pubspec.yaml
dependencies:
  mongo_dart: ^0.10.0

```

### 3.6.1.3 Aufgabe 1: Verbindung & Collection (10 min)

```

// lib/database.dart
import 'package:mongo_dart/mongo_dart.dart';

```

```
class Database {
    static Db? _db;

    static Future<Db> connect() async {
        // TODO: Verbindung herstellen
        // URL: mongodb://root:secret@localhost:27017/shop
    }

    static DbCollection get products {
        // TODO: products Collection zurückgeben
    }

    static Future<void> close() async {
        // TODO: Verbindung schließen
    }
}
```

#### 3.6.1.4 Aufgabe 2: CRUD Operationen (25 min)

Create

```
Future<String> createProduct(Map<String, dynamic> data) async {
    // TODO: createdAt hinzufügen
    // TODO: insertOne
    // TODO: ID als String zurückgeben
}
```

Read

```
Future<List<Map<String, dynamic>>> getAllProducts() async {
    // TODO: find().toList()
}

Future<Map<String, dynamic>?> getProductById(String id) async {
    // TODO: findOne mit ObjectId
}

Future<List<Map<String, dynamic>>> getProductsByCategory(String category) async {
    // TODO: find mit where.eq
}

Future<List<Map<String, dynamic>>> searchProducts(String query) async {
    // TODO: Suche in name und description
    // Hint: where.match('name', '.*$query.*', caseInsensitive: true)
}
```

Update

```
Future<bool> updateProduct(String id, Map<String, dynamic> data) async {
    // TODO: updateOne
    // TODO: updatedAt setzen
}
```

```
}

Future<bool> adjustStock(String id, int delta) async {
  // TODO: modify.inc('stock', delta)
}
```

Delete

```
Future<bool> deleteProduct(String id) async {
  // TODO: deleteOne
}
```

### 3.6.1.5 Aufgabe 3: Eingebettete Dokumente (15 min)

Produkt mit Specs

```
await products.insertOne({
  'name': 'MacBook Pro',
  'price': 2499.99,
  'category': 'electronics',
  'specs': {
    'display': '16 inch',
    'cpu': 'M3 Pro',
    'ram': '18GB',
    'storage': '512GB SSD',
  },
  'variants': [
    {'color': 'Space Black', 'sku': 'MBP-16-BLK'},
    {'color': 'Silver', 'sku': 'MBP-16-SLV'},
  ],
  'reviews': [],
});
```

Abfragen

```
// Produkte mit mindestens 16GB RAM
Future<List<Map<String, dynamic>>> findByMinRam(int minGb) async {
  // TODO: where.gte('specs.ram', '${minGb}GB')
}

// Review hinzufügen
Future<void> addReview(String productId, Map<String, dynamic> review) async {
  // TODO: modify.push('reviews', review)
}
```

### 3.6.1.6 Aufgabe 4: Pagination & Sortierung (10 min)

```
Future<List<Map<String, dynamic>>> getProductsPaginated({
  int page = 1,
  int perPage = 10,
  String sortBy = 'createdAt',
```

```
    bool descending = true,
  }) async {
    // TODO: skip, limit, sortBy
  }

Future<int> countProducts({String? category}) async {
  // TODO: count() mit optionalem Filter
}
```

### 3.6.1.7 Aufgabe 5: Repository Pattern (20 min)

```
// lib/repositories/product_repository.dart

class Product {
  final String? id;
  final String name;
  final String? description;
  final double price;
  final int stock;
  final String category;
  final List<String> tags;
  final DateTime? createdAt;

  // TODO: Konstruktor
  // TODO: factory fromMap
  // TODO: Map<String, dynamic> toMap()
}

class ProductRepository {
  final DbCollection _collection;

  ProductRepository(Db db) : _collection = db.collection('products');

  Future<List<Product>> findAll() async {
    // TODO
  }

  Future<Product?> findById(String id) async {
    // TODO
  }

  Future<Product> create(Product product) async {
    // TODO
  }

  Future<Product?> update(String id, Product product) async {
    // TODO
  }

  Future<bool> delete(String id) async {
```

```
    // TODO
  }
}
```

### 3.6.1.8 Aufgabe 6: Indizes erstellen (5 min)

```
Future<void> createIndexes(DbCollection products) async {
  // Index auf name (für Suche)
  await products.createIndex(keys: {'name': 1});

  // Compound Index für Kategorie + Preis
  await products.createIndex(keys: {'category': 1, 'price': -1});

  // Index auf Tags (Array)
  await products.createIndex(keys: {'tags': 1});
}
```

### 3.6.1.9 Testen

```
Future<void> main() async {
  final db = await Database.connect();

  try {
    // Produkt erstellen
    final id = await createProduct({
      'name': 'Test Product',
      'price': 99.99,
      'stock': 50,
      'category': 'test',
      'tags': ['new'],
    });
    print('Created: $id');

    // Alle Produkte
    final all = await getAllProducts();
    print('Total: ${all.length}');

    // Nach ID
    final product = await getProductById(id);
    print('Found: ${product?['name']}');

    // Suchen
    final results = await searchProducts('Test');
    print('Search results: ${results.length}');

    // Stock erhöhen
    await adjustStock(id, 10);

    // Löschen
```

```
    await deleteProduct(id);

  } finally {
    await Database.close();
  }
}
```

#### 3.6.1.10 Abgabe-Checkliste

- ☐ Datenbankverbindung funktioniert
- ☐ createProduct mit ID-Rückgabe
- ☐ getAllProducts
- ☐ getProductById
- ☐ getProductsByCategory
- ☐ searchProducts
- ☐ updateProduct
- ☐ adjustStock mit inc
- ☐ deleteProduct
- ☐ Eingebettete Dokumente (specs, variants)
- ☐ Pagination mit skip/limit
- ☐ Product Model mit fromMap/toMap
- ☐ ProductRepository implementiert
- ☐ Indizes erstellt

### 3.6.2 Lösung

#### 3.6.2.1 Database Connection

```
// lib/database.dart
import 'package:mongo_dart/mongo_dart.dart';

class Database {
  static Db? _db;

  static Future<Db> connect() async {
    _db ??= Db('mongodb://root:secret@localhost:27017/shop');
    if (!_db!.isConnected) {
      await _db!.open();
    }
    return _db!;
  }

  static DbCollection get products {
    if (_db == null || !_db!.isConnected) {
      throw StateError('Database not connected');
    }
    return _db!.collection('products');
  }

  static Future<void> close() async {
    await _db?.close();
  }
}
```

```
    _db = null;
  }
}
```

### 3.6.2.2 CRUD Operations

```
// lib/product_operations.dart
import 'package:mongo_dart/mongo_dart.dart';
import 'database.dart';

// CREATE
Future<String> createProduct(Map<String, dynamic> data) async {
  data['createdAt'] = DateTime.now();
  data['tags'] ??= <String>[];

  final result = await Database.products.insertOne(data);
  return result.id.toHexString();
}

// READ - All
Future<List<Map<String, dynamic>>> getAllProducts() async {
  return await Database.products.find().toList();
}

// READ - By ID
Future<Map<String, dynamic>?> getProductById(String id) async {
  try {
    return await Database.products.findOne(
      where.id(ObjectId.parse(id)),
    );
  } catch (_) {
    return null;
  }
}

// READ - By Category
Future<List<Map<String, dynamic>>> getProductsByCategory(String category) async {
  return await Database.products
    .find(where.eq('category', category))
    .toList();
}

// READ - Search
Future<List<Map<String, dynamic>>> searchProducts(String query) async {
  return await Database.products
    .find(where.match('name', '.*$query.*', caseInsensitive: true))
    .toList();
}

// READ - By Price Range
```

```
Future<List<Map<String, dynamic>>> getProductsByPriceRange(
    double min,
    double max,
) async {
    return await Database.products
        .find(where.gte('price', min).lte('price', max))
        .toList();
}

// UPDATE
Future<bool> updateProduct(String id, Map<String, dynamic> data) async {
    try {
        var modifier = ModifierBuilder();

        if (data['name'] != null) modifier = modifier.set('name', data['name']);
        if (data['price'] != null) modifier = modifier.set('price', data['price']);
        if (data['stock'] != null) modifier = modifier.set('stock', data['stock']);
        if (data['description'] != null) {
            modifier = modifier.set('description', data['description']);
        }

        modifier = modifier.set('updatedAt', DateTime.now());

        final result = await Database.products.updateOne(
            where.id(ObjectId.parse(id)),
            modifier,
        );

        return result.nModified > 0;
    } catch (_) {
        return false;
    }
}

// UPDATE - Adjust Stock
Future<bool> adjustStock(String id, int delta) async {
    try {
        final result = await Database.products.updateOne(
            where.id(ObjectId.parse(id)),
            modify.inc('stock', delta).set('updatedAt', DateTime.now()),
        );
        return result.nModified > 0;
    } catch (_) {
        return false;
    }
}

// DELETE
Future<bool> deleteProduct(String id) async {
    try {
        final result = await Database.products.deleteOne(
```

```

        where.id(ObjectId.parse(id)),
    );
    return result.nRemoved > 0;
} catch (_) {
    return false;
}
}

```

### 3.6.2.3 Pagination & Sorting

```

Future<List<Map<String, dynamic>>> getProductsPaginated({
    int page = 1,
    int perPage = 10,
    String sortBy = 'createdAt',
    bool descending = true,
    String? category,
}) async {
    var query = where.sortBy(sortBy, descending: descending);

    if (category != null) {
        query = query.eq('category', category);
    }

    query = query.skip((page - 1) * perPage).limit(perPage);

    return await Database.products.find(query).toList();
}

Future<int> countProducts({String? category}) async {
    if (category != null) {
        return await Database.products.count(where.eq('category', category));
    }
    return await Database.products.count();
}

```

### 3.6.2.4 Product Model & Repository

```

// lib/models/product.dart
import 'package:mongo_dart/mongo_dart.dart';

class Product {
    final String? id;
    final String name;
    final String? description;
    final double price;
    final int stock;
    final String category;
    final List<String> tags;
    final Map<String, dynamic>? specs;
}

```

```

final DateTime? createdAt;
final DateTime? updatedAt;

Product({
  this.id,
  required this.name,
  this.description,
  required this.price,
  this.stock = 0,
  required this.category,
  this.tags = const [],
  this.specs,
  this.createdAt,
  this.updatedAt,
});

factory Product.fromMap(Map<String, dynamic> map) {
  return Product(
    id: map['_id']?.toHexString(),
    name: map['name'] as String,
    description: map['description'] as String?,
    price: (map['price'] as num).toDouble(),
    stock: map['stock'] as int? ?? 0,
    category: map['category'] as String,
    tags: List<String>.from(map['tags'] ?? []),
    specs: map['specs'] as Map<String, dynamic>?,
    createdAt: map['createdAt'] as DateTime?,
    updatedAt: map['updatedAt'] as DateTime?,
  );
}

Map<String, dynamic> toMap() {
  return {
    if (id != null) '_id': ObjectId.parse(id!),
    'name': name,
    'description': description,
    'price': price,
    'stock': stock,
    'category': category,
    'tags': tags,
    if (specs != null) 'specs': specs,
    'createdAt': createdAt ?? DateTime.now(),
    if (updatedAt != null) 'updatedAt': updatedAt,
  };
}

@override
String toString() => 'Product($id: $name, €$price)';
}

```

```
// lib/repositories/product_repository.dart
import 'package:mongo_dart/mongo_dart.dart';
import '../models/product.dart';

class ProductRepository {
  final DbCollection _collection;

  ProductRepository(Db db) : _collection = db.collection('products');

  Future<List<Product>> findAll() async {
    final docs = await _collection.find().toList();
    return docs.map(Product.fromMap).toList();
  }

  Future<Product?> findById(String id) async {
    try {
      final doc = await _collection.findOne(where.id(ObjectId.parse(id)));
      return doc != null ? Product.fromMap(doc) : null;
    } catch (_) {
      return null;
    }
  }

  Future<List<Product>> findByCategory(String category) async {
    final docs = await _collection
      .find(where.eq('category', category))
      .toList();
    return docs.map(Product.fromMap).toList();
  }

  Future<List<Product>> search(String query) async {
    final docs = await _collection
      .find(where.match('name', '.*$query.*', caseInsensitive: true))
      .toList();
    return docs.map(Product.fromMap).toList();
  }

  Future<Product> create(Product product) async {
    final map = product.toMap();
    map['createdAt'] = DateTime.now();
    map.remove('_id');

    final result = await _collection.insertOne(map);
    map['_id'] = result.id;

    return Product.fromMap(map);
  }

  Future<Product?> update(String id, Product product) async {
    try {
      final result = await _collection.findAndModify(
```

```
        query: where.id(ObjectId.parse(id)),
        update: modify
            .set('name', product.name)
            .set('description', product.description)
            .set('price', product.price)
            .set('stock', product.stock)
            .set('category', product.category)
            .set('tags', product.tags)
            .set('updatedAt', DateTime.now()),
        returnNew: true,
    );

    return result != null ? Product.fromMap(result) : null;
} catch (_) {
    return null;
}
}

Future<bool> delete(String id) async {
    try {
        final result = await _collection.deleteOne(
            where.id(ObjectId.parse(id)),
        );
        return result.nRemoved > 0;
    } catch (_) {
        return false;
    }
}

Future<bool> addTag(String id, String tag) async {
    try {
        final result = await _collection.updateOne(
            where.id(ObjectId.parse(id)),
            modify.addToSet('tags', tag),
        );
        return result.nModified > 0;
    } catch (_) {
        return false;
    }
}

Future<bool> removeTag(String id, String tag) async {
    try {
        final result = await _collection.updateOne(
            where.id(ObjectId.parse(id)),
            modify.pull('tags', tag),
        );
        return result.nModified > 0;
    } catch (_) {
        return false;
    }
}
```

```
}  
}
```

### 3.6.2.5 Indexes

```
Future<void> createIndexes(DbCollection products) async {  
    // Name Index (für Suche)  
    await products.createIndex(keys: {'name': 1});  
  
    // Category + Price (für gefilterte Sortierung)  
    await products.createIndex(keys: {'category': 1, 'price': -1});  
  
    // Tags (Multikey Index)  
    await products.createIndex(keys: {'tags': 1});  
  
    // Created At (für Sortierung)  
    await products.createIndex(keys: {'createdAt': -1});  
  
    print('Indexes created');  
}
```

### 3.6.2.6 Main

```
Future<void> main() async {  
    final db = await Database.connect();  
  
    try {  
        final repo = ProductRepository(db);  
  
        // Create  
        final product = await repo.create(Product(  
            name: 'Test Laptop',  
            price: 999.99,  
            stock: 25,  
            category: 'electronics',  
            tags: ['new'],  
        ));  
        print('Created: $product');  
  
        // Read  
        final found = await repo.findById(product.id!);  
        print('Found: $found');  
  
        // Update  
        final updated = await repo.update(  
            product.id!,  
            Product(  
                name: 'Updated Laptop',  
                price: 899.99,  
            ),  
        );  
    } catch (e) {  
        print(e);  
    }  
}
```

```
        stock: 30,
        category: 'electronics',
        tags: ['sale'],
    ),
);
print('Updated: $updated');

// Search
final results = await repo.search('Laptop');
print('Search results: ${results.length}');

// Delete
final deleted = await repo.delete(product.id!);
print('Deleted: $deleted');

} finally {
    await Database.close();
}
}
```

### 3.6.3 Ressourcen

#### 3.6.3.1 Offizielle Dokumentation

- [mongo\\_dart](#) (pub.dev)
- [MongoDB Documentation](#)
- [MongoDB Query Operators](#)

#### 3.6.3.2 Cheat Sheet: Verbindung

```
import 'package:mongo_dart/mongo_dart.dart';

// Verbinden
final db = Db('mongodb://user:password@localhost:27017/dbname');
await db.open();

// Collection
final collection = db.collection('products');

// Schließen
await db.close();
```

#### 3.6.3.3 Cheat Sheet: Insert

```
// Einzeln
await collection.insertOne({'name': 'Product', 'price': 99.99});

// Mehrere
await collection.insertMany([
    {'name': 'A', 'price': 10},
```

```
    {'name': 'B', 'price': 20},
  ]);

// Mit ID-Rückgabe
final result = await collection.insertOne({...});
final id = result.id.toHexString();
```

#### 3.6.3.4 Cheat Sheet: Find

```
// Alle
final all = await collection.find().toList();

// Mit Filter
final filtered = await collection.find(where.eq('category',
↪  'electronics')).toList();

// Einzeln
final one = await collection.findOne(where.eq('name', 'Product'));

// Nach ID
final byId = await collection.findOne(where.id(ObjectId.parse(id)));

// Vergleiche
where.gt('price', 100)    // >
where.gte('price', 100)   // >=
where.lt('price', 100)    // <
where.lte('price', 100)   // <=
where.ne('status', 'deleted') // !=

// Kombinationen
where.eq('category', 'electronics').gt('price', 100) // AND
where.oneFrom('category', ['a', 'b']) // IN

// Sortierung
where.sortBy('price', descending: true)

// Pagination
where.skip(10).limit(10)

// Regex
where.match('name', '.*laptop.*', caseInsensitive: true)
```

#### 3.6.3.5 Cheat Sheet: Update

```
// Einzeln
await collection.updateOne(
  where.eq('name', 'Product'),
  modify.set('price', 89.99),
);
```

```
// Mehrere Felder
modify.set('price', 89.99).set('stock', 100)

// Increment
modify.inc('stock', 10)    // +10
modify.inc('stock', -5)    // -5

// Array Operations
modify.push('tags', 'new')    // Add to array
modify.pull('tags', 'old')    // Remove from array
modify.addToSet('tags', 'sale') // Add if not exists

// Mehrere Dokumente
await collection.updateMany(
  where.eq('category', 'electronics'),
  modify.mul('price', 0.9), // -10%
);
```

#### 3.6.3.6 Cheat Sheet: Delete

```
// Einzeln
await collection.deleteOne(where.eq('name', 'Product'));

// Mehrere
await collection.deleteMany(where.eq('stock', 0));

// Alle
await collection.deleteMany({});
```

#### 3.6.3.7 Cheat Sheet: Aggregation

```
final pipeline = AggregationPipelineBuilder()
  .addStage(Match(where.eq('category', 'electronics')))
  .addStage(Group(
    id: r'$category',
    fields: {
      'avgPrice': {r'$avg': r'$price'},
      'count': {r'$sum': 1},
    },
  ))
  .build();

final result = await collection.aggregateToStream(pipeline).toList();
```

#### 3.6.3.8 Cheat Sheet: Indexes

```
// Einfach
await collection.createIndex(keys: {'name': 1});
```

```
// Unique
await collection.createIndex(keys: {'email': 1}, unique: true);

// Compound
await collection.createIndex(keys: {'category': 1, 'price': -1});

// 1 = ascending, -1 = descending
```

### 3.6.3.9 SQL vs MongoDB

SQL	MongoDB
SELECT * FROM t	find()
WHERE a = 1	where.eq('a', 1)
WHERE a > 1	where.gt('a', 1)
ORDER BY a	where.sortBy('a')
LIMIT 10	where.limit(10)
INSERT INTO	insertOne/Many
UPDATE	updateOne/Many
DELETE	deleteOne/Many

## 3.7 Einheit 7.7: Queries & Aggregationen

### 3.7.0.1 Lernziele

Nach dieser Einheit kannst du: - Komplexe SQL-Abfragen mit Subqueries schreiben - Window Functions für Analysen nutzen - Aggregation Pipelines in MongoDB erstellen - Performance-Optimierung für Queries

### 3.7.0.2 SQL: Subqueries

Scalar Subquery

```
-- Produkte teurer als Durchschnitt
SELECT name, price
FROM products
WHERE price > (SELECT AVG(price) FROM products);

-- Letzter Besteller
SELECT name,
       (SELECT MAX(created_at) FROM orders WHERE customer_id = c.id) AS
↪ last_order
FROM customers c;
```

IN Subquery

```
-- Kunden mit Bestellungen
SELECT name, email
FROM customers
WHERE id IN (SELECT DISTINCT customer_id FROM orders);
```

```
-- Produkte ohne Bestellungen
SELECT name
FROM products
WHERE id NOT IN (SELECT DISTINCT product_id FROM order_items);
```

EXISTS Subquery

```
-- Kunden mit mindestens einer Bestellung
SELECT name
FROM customers c
WHERE EXISTS (
    SELECT 1 FROM orders WHERE customer_id = c.id
);
```

FROM Subquery (Derived Table)

```
-- Top-Kategorien mit Statistiken
SELECT category_stats.*
FROM (
    SELECT
        c.name AS category,
        COUNT(p.id) AS product_count,
        AVG(p.price) AS avg_price,
        SUM(p.stock) AS total_stock
    FROM categories c
    LEFT JOIN products p ON c.id = p.category_id
    GROUP BY c.id, c.name
) AS category_stats
WHERE product_count > 5;
```

### 3.7.0.3 SQL: Common Table Expressions (CTE)

```
-- CTE für bessere Lesbarkeit
WITH category_sales AS (
    SELECT
        p.category_id,
        SUM(oi.quantity * oi.unit_price) AS total_sales
    FROM order_items oi
    JOIN products p ON oi.product_id = p.id
    GROUP BY p.category_id
),
category_info AS (
    SELECT id, name FROM categories
)
SELECT
    ci.name AS category,
    COALESCE(cs.total_sales, 0) AS sales
FROM category_info ci
LEFT JOIN category_sales cs ON ci.id = cs.category_id
ORDER BY sales DESC;
```

## Recursive CTE

```
-- Kategorie-Hierarchie
WITH RECURSIVE category_tree AS (
  -- Basis: Root-Kategorien
  SELECT id, name, parent_id, 0 AS depth, name AS path
  FROM categories
  WHERE parent_id IS NULL

  UNION ALL

  -- Rekursion
  SELECT c.id, c.name, c.parent_id, ct.depth + 1,
         ct.path || ' > ' || c.name
  FROM categories c
  JOIN category_tree ct ON c.parent_id = ct.id
)
SELECT * FROM category_tree ORDER BY path;
```

## 3.7.0.4 SQL: Window Functions

ROW\_NUMBER, RANK, DENSE\_RANK

```
-- Produkte mit Rang pro Kategorie
SELECT
  name,
  category_id,
  price,
  ROW_NUMBER() OVER (PARTITION BY category_id ORDER BY price DESC) AS row_num,
  RANK() OVER (PARTITION BY category_id ORDER BY price DESC) AS rank,
  DENSE_RANK() OVER (PARTITION BY category_id ORDER BY price DESC) AS
    ↪ dense_rank
FROM products;
```

## Running Total

```
-- Kumulativer Umsatz
SELECT
  DATE(created_at) AS date,
  SUM(total) AS daily_total,
  SUM(SUM(total)) OVER (ORDER BY DATE(created_at)) AS running_total
FROM orders
GROUP BY DATE(created_at)
ORDER BY date;
```

## LAG / LEAD

```
-- Vergleich mit vorherigem Tag
SELECT
  DATE(created_at) AS date,
  SUM(total) AS daily_sales,
  LAG(SUM(total)) OVER (ORDER BY DATE(created_at)) AS prev_day,
  SUM(total) - LAG(SUM(total)) OVER (ORDER BY DATE(created_at)) AS diff
```

```
FROM orders
GROUP BY DATE(created_at);
```

NTILE (Quartile)

```
-- Produkte in Preisquartile einteilen
SELECT
    name,
    price,
    NTILE(4) OVER (ORDER BY price) AS price_quartile
FROM products;
```

### 3.7.0.5 MongoDB: Aggregation Pipeline

Grundstruktur

```
final pipeline = AggregationPipelineBuilder()
    .addStage(Match(where.eq('category', 'electronics')))
    .addStage(Sort({'price': -1}))
    .addStage(Limit(10))
    .build();

final results = await collection.aggregateToStream(pipeline).toList();
```

\$match (Filter)

```
// Äquivalent zu WHERE
final pipeline = [
  {
    r'$match': {
      'category': 'electronics',
      'price': {r'$gte': 100},
    }
  }
];
```

\$group (Aggregation)

```
// Verkäufe pro Kategorie
final pipeline = [
  {
    r'$group': {
      '_id': r'$category',
      'totalSales': {r'$sum': r'$price'},
      'avgPrice': {r'$avg': r'$price'},
      'count': {r'$sum': 1},
    }
  }
];
```

\$project (Felder auswählen/transformieren)

```
final pipeline = [
  {
    r'$project': {
      'name': 1,
      'price': 1,
      'discountedPrice': {r'$multiply': [r'$price', 0.9]},
      '_id': 0,
    }
  }
];
```

\$lookup (JOIN)

```
// Produkte mit Kategorie-Details
final pipeline = [
  {
    r'$lookup': {
      'from': 'categories',
      'localField': 'categoryId',
      'foreignField': '_id',
      'as': 'category',
    }
  },
  {
    r'$unwind': r'$category'
  }
];
```

Komplettes Beispiel

```
Future<List<Map<String, dynamic>>> getCategorySales() async {
  final pipeline = [
    // Filter: Nur abgeschlossene Bestellungen
    {
      r'$match': {'status': 'completed'}
    },
    // Unwind: Array von Items auflösen
    {
      r'$unwind': r'$items'
    },
    // Lookup: Produkt-Details laden
    {
      r'$lookup': {
        'from': 'products',
        'localField': 'items.productId',
        'foreignField': '_id',
        'as': 'product',
      }
    },
    {
      r'$unwind': r'$product'
    },
  ],
```

```

// Group: Nach Kategorie gruppieren
{
  r'$group': {
    '_id': r'$product.category',
    'totalRevenue': {
      r'$sum': {
        r'$multiply': [r'$items.quantity', r'$items.price']
      }
    },
    'totalQuantity': {r'$sum': r'$items.quantity'},
    'orderCount': {r'$sum': 1},
  }
},
// Sort: Nach Umsatz absteigend
{
  r'$sort': {'totalRevenue': -1}
},
// Project: Felder umbenennen
{
  r'$project': {
    '_id': 0,
    'category': r'$_id',
    'totalRevenue': 1,
    'totalQuantity': 1,
    'orderCount': 1,
  }
}
];

return await orders.aggregateToStream(pipeline).toList();
}

```

### 3.7.0.6 Performance-Optimierung

SQL: EXPLAIN ANALYZE

```

EXPLAIN ANALYZE
SELECT p.*, c.name AS category
FROM products p
JOIN categories c ON p.category_id = c.id
WHERE p.price > 100;

```

Indizes für häufige Queries

```

-- Index für WHERE-Klauseln
CREATE INDEX idx_products_price ON products(price);

-- Index für JOINS
CREATE INDEX idx_products_category ON products(category_id);

-- Composite Index für Filter + Sort
CREATE INDEX idx_products_cat_price ON products(category_id, price DESC);

```

MongoDB: explain()

```
final explanation = await collection.aggregate(
  pipeline,
  explain: true,
).toList();
print(explanation);
```

### 3.7.0.7 Dart Integration

```
class AnalyticsRepository {
  final Pool _pool;

  AnalyticsRepository(this._pool);

  /// Top-Produkte pro Kategorie
  Future<List<Map<String, dynamic>>> getTopProductsByCategory({
    int topN = 5,
  }) async {
    final result = await _pool.execute(Sql.named('''
      WITH ranked_products AS (
        SELECT
          p.*,
          c.name AS category_name,
          ROW_NUMBER() OVER (
            PARTITION BY p.category_id
            ORDER BY p.price DESC
          ) AS rank
        FROM products p
        LEFT JOIN categories c ON p.category_id = c.id
      )
      SELECT *
      FROM ranked_products
      WHERE rank <= @topN
      ORDER BY category_name, rank
    '''), parameters: {'topN': topN});

    return result.map((r) => r.toColumnMap()).toList();
  }

  /// Umsatzentwicklung
  Future<List<Map<String, dynamic>>> getSalesTimeline({
    required DateTime from,
    required DateTime to,
  }) async {
    final result = await _pool.execute(Sql.named('''
      SELECT
        DATE(created_at) AS date,
        COUNT(*) AS order_count,
        SUM(total) AS daily_revenue,
```

```

        SUM(SUM(total)) OVER (ORDER BY DATE(created_at)) AS cumulative_revenue
    FROM orders
    WHERE created_at >= @from AND created_at <= @to
    GROUP BY DATE(created_at)
    ORDER BY date
    '''), parameters: {'from': from, 'to': to});

    return result.map((r) => r.toColumnMap()).toList();
}
}

```

### 3.7.0.8 Zusammenfassung

Konzept	SQL	MongoDB
Filter	WHERE	\$match
Gruppieren	GROUP BY	\$group
Sortieren	ORDER BY	\$sort
Limit	LIMIT	\$limit
JOIN	JOIN	\$lookup
Projektion	SELECT	\$project
Subquery	(SELECT...)	Pipeline Stage

### 3.7.0.9 Nächste Schritte

In der nächsten Einheit lernst du **Redis & Caching**: Wie du häufige Abfragen cachst und die Performance verbesserst.

## 3.7.1 Übung

### 3.7.1.1 Ziel

Schreibe komplexe Abfragen für Analysen und Reporting.

### 3.7.1.2 Vorbereitung

Verwende die Shop-Datenbank aus den vorherigen Einheiten mit: - customers, products, categories  
- orders, order\_items

### 3.7.1.3 Aufgabe 1: Subqueries (15 min)

1.1 Produkte über Durchschnittspreis

```

-- TODO: Produkte die teurer als der Durchschnitt sind
SELECT name, price
FROM products
WHERE price > (...)

```

1.2 Kunden ohne Bestellungen

```

-- TODO: Kunden die noch nie bestellt haben
SELECT name, email

```

```
FROM customers
WHERE id NOT IN (...)
```

### 1.3 Bestseller pro Kategorie

```
-- TODO: Das meistverkaufte Produkt pro Kategorie
-- Hint: Subquery mit GROUP BY
```

## 3.7.1.4 Aufgabe 2: CTEs (15 min)

### 2.1 Kategorie-Statistiken

```
WITH category_stats AS (
    -- TODO: Pro Kategorie: count, avg_price, total_stock
)
SELECT * FROM category_stats
WHERE count > 0
ORDER BY avg_price DESC;
```

### 2.2 Kunden-Übersicht

```
WITH customer_orders AS (
    -- TODO: Pro Kunde: order_count, total_spent, avg_order_value
)
SELECT
    c.name,
    c.email,
    COALESCE(co.order_count, 0) AS orders,
    COALESCE(co.total_spent, 0) AS spent
FROM customers c
LEFT JOIN customer_orders co ON c.id = co.customer_id
ORDER BY spent DESC;
```

## 3.7.1.5 Aufgabe 3: Recursive CTE (10 min)

```
-- Kategorie-Hierarchie mit Pfad und Tiefe
WITH RECURSIVE category_tree AS (
    -- Basis
    -- TODO

    UNION ALL

    -- Rekursion
    -- TODO
)
SELECT
    id,
    name,
    depth,
    path
FROM category_tree
```

```
ORDER BY path;
```

Erwartete Ausgabe:

id	name	depth	path
1	Electronics	0	Electronics
4	Laptops	1	Electronics > Laptops
5	Smartphones	1	Electronics > Smartphones
2	Clothing	0	Clothing
6	T-Shirts	1	Clothing > T-Shirts

### 3.7.1.6 Aufgabe 4: Window Functions (20 min)

#### 4.1 Ranking

```
-- Top 3 Produkte pro Kategorie nach Preis
SELECT
    name,
    category_id,
    price,
    -- TODO: RANK() OVER (...)
FROM products
WHERE rank <= 3;
```

#### 4.2 Running Total

```
-- Kumulativer Umsatz pro Tag
SELECT
    DATE(created_at) AS date,
    SUM(total) AS daily_revenue,
    -- TODO: SUM() OVER (ORDER BY ...) AS running_total
FROM orders
GROUP BY DATE(created_at);
```

#### 4.3 Vergleich mit Vortag

```
-- Tagesumsatz mit Differenz zum Vortag
SELECT
    DATE(created_at) AS date,
    SUM(total) AS revenue,
    -- TODO: LAG() für Vortag
    -- TODO: Differenz berechnen
FROM orders
GROUP BY DATE(created_at);
```

#### 4.4 Perzentile

```
-- Produkte in Preisquartile einteilen
SELECT
    name,
    price,
    -- TODO: NTILE(4)
CASE
```

```
        WHEN quartile = 1 THEN 'Budget'
        WHEN quartile = 4 THEN 'Premium'
        ELSE 'Standard'
    END AS segment
FROM (
    SELECT name, price, NTILE(4) OVER (ORDER BY price) AS quartile
    FROM products
) ranked;
```

### 3.7.1.7 Aufgabe 5: MongoDB Aggregation (20 min)

#### 5.1 Umsatz pro Kategorie

```
Future<List<Map<String, dynamic>>> getSalesByCategory() async {
    final pipeline = [
        // TODO: $unwind items
        // TODO: $group by category
        // TODO: $sort by revenue
    ];

    return await orders.aggregateToStream(pipeline).toList();
}
```

#### 5.2 Top-Kunden

```
Future<List<Map<String, dynamic>>> getTopCustomers({int limit = 10}) async {
    final pipeline = [
        // TODO: $group by customerId
        // TODO: Calculate totalSpent, orderCount
        // TODO: $sort by totalSpent desc
        // TODO: $limit
        // TODO: $lookup customer details
    ];

    return await orders.aggregateToStream(pipeline).toList();
}
```

#### 5.3 Produkt-Performance

```
Future<List<Map<String, dynamic>>> getProductPerformance() async {
    // Pro Produkt:
    // - totalSold (quantity)
    // - totalRevenue
    // - avgOrderSize
    // - orderCount

    final pipeline = [
        // TODO: Build pipeline
    ];

    return await orderItems.aggregateToStream(pipeline).toList();
}
```

### 3.7.1.8 Aufgabe 6: Analytics Repository (Bonus, 15 min)

```
class AnalyticsRepository {
    final Pool _pool;

    AnalyticsRepository(this._pool);

    /// Dashboard-Statistiken
    Future<DashboardStats> getDashboardStats() async {
        // TODO: In einer Query:
        // - totalRevenue
        // - orderCount
        // - avgOrderValue
        // - topCategory
        // - lowStockCount
    }

    /// Sales Trend (letzte 30 Tage)
    Future<List<DailySales>> getSalesTrend() async {
        // TODO: Window Function für Running Total
    }

    /// ABC-Analyse (Produkte nach Umsatzanteil)
    Future<List<ProductABC>> getABCAnalysis() async {
        // TODO:
        // A: Top 20% (80% Umsatz)
        // B: Nächste 30%
        // C: Restliche 50%
    }
}
```

### 3.7.1.9 Abgabe-Checkliste

- ☐ Subquery: Produkte über Durchschnitt
- ☐ Subquery: Kunden ohne Bestellungen
- ☐ CTE: Kategorie-Statistiken
- ☐ Recursive CTE: Kategorie-Hierarchie
- ☐ Window: Ranking pro Kategorie
- ☐ Window: Running Total
- ☐ Window: LAG für Vortagsvergleich
- ☐ MongoDB: Umsatz pro Kategorie
- ☐ MongoDB: Top-Kunden
- ☐ (Bonus) Analytics Repository

## 3.7.2 Lösung

### 3.7.2.1 SQL Lösungen

#### 1.1 Produkte über Durchschnittspreis

```
SELECT name, price
FROM products
```

```
WHERE price > (SELECT AVG(price) FROM products)
ORDER BY price DESC;
```

### 1.2 Kunden ohne Bestellungen

```
SELECT name, email
FROM customers
WHERE id NOT IN (
    SELECT DISTINCT customer_id FROM orders
);

-- Alternativ mit NOT EXISTS (oft effizienter)
SELECT name, email
FROM customers c
WHERE NOT EXISTS (
    SELECT 1 FROM orders WHERE customer_id = c.id
);
```

### 1.3 Bestseller pro Kategorie

```
WITH product_sales AS (
    SELECT
        p.id,
        p.name,
        p.category_id,
        SUM(oi.quantity) AS total_sold
    FROM products p
    JOIN order_items oi ON p.id = oi.product_id
    GROUP BY p.id, p.name, p.category_id
),
ranked AS (
    SELECT
        *,
        ROW_NUMBER() OVER (PARTITION BY category_id ORDER BY total_sold DESC) AS rank
    FROM product_sales
)
SELECT
    c.name AS category,
    r.name AS product,
    r.total_sold
FROM ranked r
JOIN categories c ON r.category_id = c.id
WHERE rank = 1;
```

### 2.1 Kategorie-Statistiken

```
WITH category_stats AS (
    SELECT
        c.id,
        c.name,
        COUNT(p.id) AS product_count,
```

```

        COALESCE(AVG(p.price), 0) AS avg_price,
        COALESCE(SUM(p.stock), 0) AS total_stock,
        COALESCE(MIN(p.price), 0) AS min_price,
        COALESCE(MAX(p.price), 0) AS max_price
    FROM categories c
    LEFT JOIN products p ON c.id = p.category_id
    GROUP BY c.id, c.name
)
SELECT *
FROM category_stats
WHERE product_count > 0
ORDER BY avg_price DESC;

```

## 2.2 Kunden-Übersicht

```

WITH customer_orders AS (
    SELECT
        customer_id,
        COUNT(*) AS order_count,
        SUM(total) AS total_spent,
        AVG(total) AS avg_order_value,
        MAX(created_at) AS last_order
    FROM orders
    GROUP BY customer_id
)
SELECT
    c.id,
    c.name,
    c.email,
    COALESCE(co.order_count, 0) AS orders,
    COALESCE(co.total_spent, 0) AS spent,
    COALESCE(co.avg_order_value, 0) AS avg_order,
    co.last_order
FROM customers c
LEFT JOIN customer_orders co ON c.id = co.customer_id
ORDER BY spent DESC;

```

## 3. Recursive CTE

```

WITH RECURSIVE category_tree AS (
    -- Basis: Root-Kategorien
    SELECT
        id,
        name,
        parent_id,
        0 AS depth,
        name::TEXT AS path
    FROM categories
    WHERE parent_id IS NULL

    UNION ALL

```

```

-- Rekursion: Unterkategorien
SELECT
    c.id,
    c.name,
    c.parent_id,
    ct.depth + 1,
    ct.path || ' > ' || c.name
FROM categories c
JOIN category_tree ct ON c.parent_id = ct.id
)
SELECT id, name, depth, path
FROM category_tree
ORDER BY path;

```

#### 4.1 Ranking

```

WITH ranked_products AS (
    SELECT
        p.id,
        p.name,
        c.name AS category,
        p.price,
        RANK() OVER (PARTITION BY p.category_id ORDER BY p.price DESC) AS rank
    FROM products p
    LEFT JOIN categories c ON p.category_id = c.id
)
SELECT category, name, price, rank
FROM ranked_products
WHERE rank <= 3
ORDER BY category, rank;

```

#### 4.2 Running Total

```

SELECT
    DATE(created_at) AS date,
    COUNT(*) AS order_count,
    SUM(total) AS daily_revenue,
    SUM(SUM(total)) OVER (ORDER BY DATE(created_at)) AS running_total
FROM orders
GROUP BY DATE(created_at)
ORDER BY date;

```

#### 4.3 Vergleich mit Vortag

```

WITH daily_stats AS (
    SELECT
        DATE(created_at) AS date,
        SUM(total) AS revenue
    FROM orders
    GROUP BY DATE(created_at)
)
SELECT

```

```

date,
revenue,
LAG(revenue) OVER (ORDER BY date) AS prev_day,
revenue - LAG(revenue) OVER (ORDER BY date) AS diff,
ROUND(
    (revenue - LAG(revenue) OVER (ORDER BY date)) /
    NULLIF(LAG(revenue) OVER (ORDER BY date), 0) * 100,
    2
) AS change_percent
FROM daily_stats
ORDER BY date;

```

#### 4.4 Preisquartile

```

SELECT
    name,
    price,
    quartile,
    CASE
        WHEN quartile = 1 THEN 'Budget'
        WHEN quartile = 2 THEN 'Standard'
        WHEN quartile = 3 THEN 'Premium'
        WHEN quartile = 4 THEN 'Luxury'
    END AS segment
FROM (
    SELECT
        name,
        price,
        NTILE(4) OVER (ORDER BY price) AS quartile
    FROM products
) ranked
ORDER BY price;

```

### 3.7.2.2 MongoDB Lösungen

#### 5.1 Umsatz pro Kategorie

```

Future<List<Map<String, dynamic>>> getSalesByCategory() async {
    final pipeline = [
        // Nur abgeschlossene Bestellungen
        {'$match': {'status': 'completed'}},

        // Items-Array auflösen
        {'$unwind': '$items'},

        // Nach Kategorie gruppieren
        {
            '$group': {
                '_id': '$items.category',
                'totalRevenue': {
                    '$sum': {'$multiply': ['$items.price', '$items.quantity']}
                },
            },
        },
    ],

```

```

        'totalQuantity': {r'$sum': r'$items.quantity'},
        'orderCount': {r'$sum': 1},
    }
},

// Sortieren
{r'$sort': {'totalRevenue': -1}},

// Felder umbenennen
{
    r'$project': {
        '_id': 0,
        'category': r'$_id',
        'revenue': r'$totalRevenue',
        'quantity': r'$totalQuantity',
        'orders': r'$orderCount',
    }
}
];

return await orders.aggregateToStream(pipeline).toList();
}

```

## 5.2 Top-Kunden

```

Future<List<Map<String, dynamic>>> getTopCustomers({int limit = 10}) async {
    final pipeline = [
        // Nach Kunde gruppieren
        {
            r'$group': {
                '_id': r'$customerId',
                'totalSpent': {r'$sum': r'$total'},
                'orderCount': {r'$sum': 1},
                'avgOrder': {r'$avg': r'$total'},
                'lastOrder': {r'$max': r'$createdAt'},
            }
        },

        // Sortieren
        {r'$sort': {'totalSpent': -1}},

        // Limit
        {r'$limit': limit},

        // Kunden-Details laden
        {
            r'$lookup': {
                'from': 'customers',
                'localField': '_id',
                'foreignField': '_id',
                'as': 'customer',
            }
        }
    ];

    return await orders.aggregateToStream(pipeline).toList();
}

```

```

    }
  },
  {r'$unwind': r'$customer'}},

  // Finales Format
  {
    r'$project': {
      '_id': 0,
      'customerId': r'$_id',
      'name': r'$customer.name',
      'email': r'$customer.email',
      'totalSpent': 1,
      'orderCount': 1,
      'avgOrder': {r'$round': [r'$avgOrder', 2]},
      'lastOrder': 1,
    }
  }
];

return await orders.aggregateToStream(pipeline).toList();
}

```

### 5.3 Produkt-Performance

```

Future<List<Map<String, dynamic>>> getProductPerformance() async {
  final pipeline = [
    // Items auflösen
    {r'$unwind': r'$items'},

    // Nach Produkt gruppieren
    {
      r'$group': {
        '_id': r'$items.productId',
        'totalSold': {r'$sum': r'$items.quantity'},
        'totalRevenue': {
          r'$sum': {r'$multiply': [r'$items.price', r'$items.quantity']}
        },
        'orderCount': {r'$sum': 1},
      }
    },

    // Durchschnitt berechnen
    {
      r'$addFields': {
        'avgOrderSize': {r'$divide': [r'$totalSold', r'$orderCount']},
        'avgOrderValue': {r'$divide': [r'$totalRevenue', r'$orderCount']},
      }
    },

    // Produkt-Details laden
    {

```

```

        r'$lookup': {
            'from': 'products',
            'localField': '_id',
            'foreignField': '_id',
            'as': 'product',
        }
    },
    {r'$unwind': r'$product'}},

    // Sortieren nach Umsatz
    {r'$sort': {'totalRevenue': -1}},

    // Finales Format
    {
        r'$project': {
            '_id': 0,
            'productId': r'$_id',
            'name': r'$product.name',
            'category': r'$product.category',
            'totalSold': 1,
            'totalRevenue': {r'$round': [r'$totalRevenue', 2]},
            'orderCount': 1,
            'avgOrderSize': {r'$round': [r'$avgOrderSize', 2]},
        }
    }
];

return await orderItems.aggregateToStream(pipeline).toList();
}

```

### 3.7.2.3 Analytics Repository

```

class AnalyticsRepository {
    final Pool _pool;

    AnalyticsRepository(this._pool);

    Future<Map<String, dynamic>> getDashboardStats() async {
        final result = await _pool.execute('''
        SELECT
            (SELECT COUNT(*) FROM orders) AS total_orders,
            (SELECT COALESCE(SUM(total), 0) FROM orders) AS total_revenue,
            (SELECT COALESCE(AVG(total), 0) FROM orders) AS avg_order_value,
            (SELECT COUNT(*) FROM customers) AS total_customers,
            (SELECT COUNT(*) FROM products WHERE stock < 10) AS low_stock_count,
            (
                SELECT c.name
                FROM categories c
                JOIN products p ON c.id = p.category_id
                JOIN order_items oi ON p.id = oi.product_id
            )
        ''');
    }
}

```

```

        GROUP BY c.id, c.name
        ORDER BY SUM(oi.quantity * oi.unit_price) DESC
        LIMIT 1
    ) AS top_category
    '');

    return result.first.toColumnMap();
}

Future<List<Map<String, dynamic>>> getSalesTrend({int days = 30}) async {
    final result = await _pool.execute(Sql.named('''
        WITH daily_sales AS (
            SELECT
                DATE(created_at) AS date,
                COUNT(*) AS orders,
                SUM(total) AS revenue
            FROM orders
            WHERE created_at >= NOW() - INTERVAL '@days days'
            GROUP BY DATE(created_at)
        )
        SELECT
            date,
            orders,
            revenue,
            SUM(revenue) OVER (ORDER BY date) AS cumulative_revenue,
            AVG(revenue) OVER (ORDER BY date ROWS BETWEEN 6 PRECEDING AND CURRENT
↵ ROW) AS moving_avg_7d
↵ FROM daily_sales
        ORDER BY date
    '''), parameters: {'days': days});

    return result.map((r) => r.toColumnMap()).toList();
}
}

```

### 3.7.3 Ressourcen

#### 3.7.3.1 SQL Dokumentation

- PostgreSQL Window Functions
- PostgreSQL CTEs
- MongoDB Aggregation

#### 3.7.3.2 Cheat Sheet: Subqueries

```

-- Scalar Subquery
SELECT name FROM products WHERE price > (SELECT AVG(price) FROM products);

-- IN Subquery
SELECT * FROM customers WHERE id IN (SELECT customer_id FROM orders);

```

```
-- EXISTS Subquery
SELECT * FROM customers c WHERE EXISTS (SELECT 1 FROM orders WHERE
  ↳ customer_id = c.id);

-- Correlated Subquery
SELECT name, (SELECT COUNT(*) FROM orders WHERE customer_id = c.id) AS
  ↳ order_count
FROM customers c;
```

### 3.7.3.3 Cheat Sheet: CTEs

```
-- Simple CTE
WITH stats AS (
  SELECT category_id, AVG(price) AS avg_price
  FROM products GROUP BY category_id
)
SELECT * FROM stats;

-- Multiple CTEs
WITH cte1 AS (...), cte2 AS (...)
SELECT * FROM cte1 JOIN cte2 ON ...;

-- Recursive CTE
WITH RECURSIVE tree AS (
  SELECT id, name, parent_id, 0 AS depth FROM categories WHERE parent_id
    ↳ IS NULL
  UNION ALL
  SELECT c.id, c.name, c.parent_id, t.depth + 1
  FROM categories c JOIN tree t ON c.parent_id = t.id
)
SELECT * FROM tree;
```

### 3.7.3.4 Cheat Sheet: Window Functions

```
-- Ranking
ROW_NUMBER() OVER (ORDER BY price)           -- 1,2,3,4,5
RANK() OVER (ORDER BY price)                  -- 1,2,2,4,5 (gaps)
DENSE_RANK() OVER (ORDER BY price)            -- 1,2,2,3,4 (no gaps)
NTILE(4) OVER (ORDER BY price)                -- Quartiles 1-4

-- Partition
RANK() OVER (PARTITION BY category_id ORDER BY price)

-- Running Aggregates
SUM(total) OVER (ORDER BY date)               -- Running total
AVG(total) OVER (ORDER BY date ROWS BETWEEN 6 PRECEDING AND CURRENT ROW) --
  ↳ 7-day average

-- Offset Functions
```

```
LAG(price) OVER (ORDER BY date)           -- Previous row
LEAD(price) OVER (ORDER BY date)          -- Next row
LAG(price, 7) OVER (ORDER BY date)        -- 7 rows back
FIRST_VALUE(price) OVER (ORDER BY date)   -- First in window
LAST_VALUE(price) OVER (ORDER BY date)    -- Last in window
```

### 3.7.3.5 Cheat Sheet: MongoDB Aggregation

```
// $match (Filter)
{r'$match': {'status': 'active'}}
```

```
// $group (Aggregation)
{r'$group': {
  '_id': r'$category',
  'total': {r'$sum': r'$price'},
  'count': {r'$sum': 1},
  'avg': {r'$avg': r'$price'},
}}
```

```
// $sort
{r'$sort': {'total': -1}}
```

```
// $limit / $skip
{r'$limit': 10}
{r'$skip': 20}
```

```
// $project (Select fields)
{r'$project': {'name': 1, 'price': 1, '_id': 0}}
```

```
// $lookup (JOIN)
{r'$lookup': {
  'from': 'categories',
  'localField': 'categoryId',
  'foreignField': '_id',
  'as': 'category',
}}
```

```
// $unwind (Array -> Documents)
{r'$unwind': r'$items'}
```

```
// $addFields (Computed fields)
{r'$addFields': {
  'total': {r'$multiply': [r'$price', r'$quantity']}
}}
```

### 3.7.3.6 Performance Tips

SQL

```
-- EXPLAIN für Query-Analyse
EXPLAIN ANALYZE SELECT ...;

-- Indizes für WHERE und JOIN
CREATE INDEX idx_name ON table(column);

-- Partial Index
CREATE INDEX idx_active ON users(email) WHERE active = true;

-- Covering Index
CREATE INDEX idx_cover ON products(category_id) INCLUDE (name, price);
```

MongoDB

```
// Index erstellen
await collection.createIndex(keys: {'category': 1, 'price': -1});

// explain()
final result = await collection.aggregate(pipeline, explain: true);
```

## 3.8 Einheit 7.8: Caching mit Redis

### 3.8.0.1 Lernziele

Nach dieser Einheit kannst du: - Redis-Grundlagen verstehen und anwenden - Caching-Strategien implementieren - Session-Management mit Redis umsetzen - Cache-Invalidierung richtig handhaben

### 3.8.0.2 Was ist Redis?

**Redis** (Remote Dictionary Server) ist ein In-Memory-Datenstore, der als Cache, Message Broker und Datenbank verwendet wird.

Eigenschaften

- **In-Memory:** Extrem schnell (< 1ms Latenz)
- **Persistent:** Optional auf Disk speichern
- **Datenstrukturen:** Strings, Lists, Sets, Hashes, Sorted Sets
- **Pub/Sub:** Messaging zwischen Services
- **TTL:** Automatisches Ablaufen von Keys

Wann Redis?

- Caching von Datenbankabfragen
- Session-Speicherung
- Rate Limiting
- Echtzeit-Leaderboards
- Pub/Sub Messaging

### 3.8.0.3 Redis Setup

Docker

```
docker run --name redis \
  -p 6379:6379 \
  -d redis:7
```

Dart Package

```
dependencies:
  redis: ^3.1.0
```

### 3.8.0.4 Verbindung herstellen

```
import 'package:redis/redis.dart';

Future<void> main() async {
  final conn = RedisConnection();
  final command = await conn.connect('localhost', 6379);

  // Einfacher Test
  await command.send_object(['SET', 'hello', 'world']);
  final result = await command.send_object(['GET', 'hello']);
  print(result); // 'world'

  await conn.close();
}
```

### 3.8.0.5 Basis-Operationen

Strings

```
// SET und GET
await command.send_object(['SET', 'user:1:name', 'Max']);
final name = await command.send_object(['GET', 'user:1:name']);

// Mit TTL (Sekunden)
await command.send_object(['SET', 'session:abc123', 'data', 'EX', '3600']);

// Inkrementieren
await command.send_object(['SET', 'counter', '0']);
await command.send_object(['INCR', 'counter']);
await command.send_object(['INCRBY', 'counter', '5']);

// Nur setzen wenn nicht existiert
await command.send_object(['SETNX', 'lock:resource', '1']);
```

Hashes (Objekte)

```
// Hash setzen
await command.send_object([
  'HSET', 'user:1',
  'name', 'Max',
  'email', 'max@example.com',
```

```
    'age', '30'
  });

// Einzelnes Feld lesen
final email = await command.send_object(['HGET', 'user:1', 'email']);

// Alle Felder lesen
final user = await command.send_object(['HGETALL', 'user:1']);
// ['name', 'Max', 'email', 'max@example.com', 'age', '30']

// Feld inkrementieren
await command.send_object(['HINCRBY', 'user:1', 'age', '1']);
```

#### Listen

```
// Am Ende hinzufügen
await command.send_object(['RPUSH', 'queue:tasks', 'task1', 'task2']);

// Am Anfang hinzufügen
await command.send_object(['LPUSH', 'queue:tasks', 'task0']);

// Vom Anfang entfernen
final task = await command.send_object(['LPOP', 'queue:tasks']);

// Bereich lesen
final tasks = await command.send_object(['LRANGE', 'queue:tasks', '0', '-1']);
```

#### Sets

```
// Hinzufügen
await command.send_object(['SADD', 'tags:product:1', 'new', 'sale', 'featured']);

// Prüfen ob enthalten
final isMember = await command.send_object(['SISMEMBER', 'tags:product:1',
  ↪ 'new']);

// Alle Elemente
final tags = await command.send_object(['SMEMBERS', 'tags:product:1']);

// Entfernen
await command.send_object(['SREM', 'tags:product:1', 'sale']);
```

#### Sorted Sets (Rankings)

```
// Hinzufügen mit Score
await command.send_object(['ZADD', 'leaderboard', '100', 'player1', '85',
  ↪ 'player2']);

// Top 10
final top10 = await command.send_object(['ZREVRANGE', 'leaderboard', '0',
  ↪ '9', 'WITHSCORES']);
```

```
// Rang eines Players
final rank = await command.send_object(['ZREVRANK', 'leaderboard', 'player1']);
```

### 3.8.0.6 Caching-Strategien

Cache-Aside (Lazy Loading)

```
class ProductCache {
    final Command _redis;
    final ProductRepository _repo;

    ProductCache(this._redis, this._repo);

    Future<Product?> getProduct(int id) async {
        final key = 'product:$id';

        // 1. Cache prüfen
        final cached = await _redis.send_object(['GET', key]);
        if (cached != null) {
            return Product.fromJson(jsonDecode(cached as String));
        }

        // 2. Aus DB laden
        final product = await _repo.findById(id);
        if (product == null) return null;

        // 3. In Cache speichern (1 Stunde TTL)
        await _redis.send_object([
            'SET', key, jsonEncode(product.toJson()),
            'EX', '3600'
        ]);

        return product;
    }
}
```

Write-Through

```
class ProductService {
    final Command _redis;
    final ProductRepository _repo;

    Future<Product> updateProduct(int id, ProductUpdate data) async {
        // 1. In DB speichern
        final product = await _repo.update(id, data);

        // 2. Cache aktualisieren
        await _redis.send_object([
            'SET', 'product:$id', jsonEncode(product.toJson()),
            'EX', '3600'
        ]);
    }
}
```

```

    return product;
}
}

```

Write-Behind (Async)

```

class ProductService {
    final Command _redis;
    final ProductRepository _repo;

    Future<void> updateProduct(int id, ProductUpdate data) async {
        // 1. Sofort in Cache schreiben
        await _redis.send_object([
            'SET', 'product:$id', jsonEncode(data.toJson()),
        ]);

        // 2. In Queue für DB-Write
        await _redis.send_object([
            'RPUSH', 'queue:db_writes',
            jsonEncode({'type': 'update_product', 'id': id, 'data': data.toJson()})
        ]);

        // Background Worker verarbeitet die Queue
    }
}

```

### 3.8.0.7 Cache-Invalidierung

Einzelnen Key löschen

```
await _redis.send_object(['DEL', 'product:$id']);
```

Pattern-basiert löschen

```

// Alle Produkt-Keys finden
final keys = await _redis.send_object(['KEYS', 'product:*']);

// Löschen
if (keys != null && (keys as List).isNotEmpty) {
    await _redis.send_object(['DEL', ...keys]);
}

```

Cache-Tags

```

class TaggedCache {
    final Command _redis;

    Future<void> set(String key, String value, List<String> tags) async {
        // Wert speichern
        await _redis.send_object(['SET', key, value, 'EX', '3600']);

        // Tags tracken
        for (final tag in tags) {

```

```

        await _redis.send_object(['SADD', 'tag:$tag', key]);
    }
}

Future<void> invalidateTag(String tag) async {
    // Alle Keys mit Tag finden
    final keys = await _redis.send_object(['SMEMBERS', 'tag:$tag']);

    if (keys != null && (keys as List).isNotEmpty) {
        await _redis.send_object(['DEL', ...keys]);
    }

    // Tag-Set löschen
    await _redis.send_object(['DEL', 'tag:$tag']);
}

// Verwendung
await cache.set('product:1', data, ['products', 'category:electronics']);
await cache.invalidateTag('category:electronics'); // Invalidiert alle

```

### 3.8.0.8 Session-Management

```

class SessionManager {
    final Command _redis;
    final Duration sessionDuration;

    SessionManager(this._redis, {this.sessionDuration = const Duration(hours: ↵
    ↵ 24)});

    Future<String> createSession(Map<String, dynamic> userData) async {
        final sessionId = Uuid().v4();
        final key = 'session:$sessionId';

        await _redis.send_object([
            'SET', key, jsonEncode(userData),
            'EX', sessionDuration.inSeconds.toString()
        ]);

        return sessionId;
    }

    Future<Map<String, dynamic>?> getSession(String sessionId) async {
        final key = 'session:$sessionId';
        final data = await _redis.send_object(['GET', key]);

        if (data == null) return null;

        // Session verlängern
        await _redis.send_object(['EXPIRE', key, ↵
    ↵ sessionDuration.inSeconds.toString()]);
    }
}

```

```

        return jsonDecode(data as String);
    }

    Future<void> destroySession(String sessionId) async {
        await _redis.send_object(['DEL', 'session:$sessionId']);
    }
}

```

### 3.8.0.9 Rate Limiting

```

class RateLimiter {
    final Command _redis;

    Future<bool> isAllowed(String clientId, {int maxRequests = 100, int
↪ windowSeconds = 60}) async {
        ↪ final key = 'ratelimit:$clientId';
        final now = DateTime.now().millisecondsSinceEpoch;
        final windowStart = now - (windowSeconds * 1000);

        // Alte Einträge entfernen
        await _redis.send_object(['ZREMRANGEBYSCORE', key, '0',
↪ windowStart.toString()]);
        ↪

        // Anzahl Requests im Fenster
        final count = await _redis.send_object(['ZCARD', key]);

        if ((count as int) >= maxRequests) {
            return false;
        }

        // Request hinzufügen
        await _redis.send_object(['ZADD', key, now.toString(), '$now']);
        await _redis.send_object(['EXPIRE', key, windowSeconds.toString()]);

        return true;
    }
}

```

### 3.8.0.10 Redis-Wrapper Klasse

```

class RedisClient {
    final RedisConnection _conn;
    Command? _command;

    RedisClient({String host = 'localhost', int port = 6379})
        : _conn = RedisConnection();

    Future<void> connect() async {

```

```

    _command = await _conn.connect('localhost', 6379);
}

Future<void> close() async {
    await _conn.close();
}

// String Operations
Future<void> set(String key, String value, {Duration? ttl}) async {
    if (ttl != null) {
        await _command!.send_object(['SET', key, value, 'EX',
↵ ttl.inSeconds.toString()]);
    } else {
        await _command!.send_object(['SET', key, value]);
    }
}

Future<String?> get(String key) async {
    final result = await _command!.send_object(['GET', key]);
    return result as String?;
}

Future<void> del(String key) async {
    await _command!.send_object(['DEL', key]);
}

// Hash Operations
Future<void> hset(String key, Map<String, String> fields) async {
    final args = ['HSET', key];
    fields.forEach((k, v) {
        args.add(k);
        args.add(v);
    });
    await _command!.send_object(args);
}

Future<Map<String, String>> hgetall(String key) async {
    final result = await _command!.send_object(['HGETALL', key]);
    if (result == null) return {};

    final list = result as List;
    final map = <String, String>{};
    for (var i = 0; i < list.length; i += 2) {
        map[list[i] as String] = list[i + 1] as String;
    }
    return map;
}

// JSON Helpers
Future<void> setJson(String key, Map<String, dynamic> value, {Duration?
↵ ttl}) async {

```

```

    await set(key, jsonEncode(value), ttl: ttl);
  }

Future<Map<String, dynamic>?> getJson(String key) async {
  final data = await get(key);
  if (data == null) return null;
  return jsonDecode(data);
}
}

```

### 3.8.0.11 Zusammenfassung

Operation	Redis Befehl
String setzen	SET key value
String lesen	GET key
Mit TTL	SET key value EX seconds
Löschen	DEL key
Hash setzen	HSET key field value
Hash lesen	HGETALL key
Liste pushen	R PUSH key value
Liste poppen	L POP key

Strategie	Verwendung
Cache-Aside	Lazy Loading, lesen-intensiv
Write-Through	Konsistenz wichtig
Write-Behind	Hohe Schreiblast

### 3.8.0.12 Fazit Block 7

Du hast in diesem Block gelernt:

1. **SQL-Grundlagen** - Relationale Datenbanken verstehen
2. **PostgreSQL mit Dart** - Queries ausführen
3. **Repository Pattern** - Saubere Architektur
4. **Relationale Modellierung** - Beziehungen und JOINS
5. **Migrations** - Versionierte Schema-Änderungen
6. **MongoDB** - Dokumentenorientierte Alternative
7. **Komplexe Queries** - Aggregationen und Analytics
8. **Redis & Caching** - Performance-Optimierung

Mit diesen Kenntnissen kannst du professionelle Backend-Systeme mit effizienter Datenhaltung entwickeln.

## 3.8.1 Übung

### 3.8.1.1 Ziel

Implementiere ein Caching-System für die Produkt-API.

### 3.8.1.2 Vorbereitung

Redis starten

```
docker run --name redis -p 6379:6379 -d redis:7
```

Dependencies

```
dependencies:  
  redis: ^3.1.0
```

### 3.8.1.3 Aufgabe 1: Redis-Verbindung (10 min)

```
// lib/cache/redis_client.dart  
  
class RedisClient {  
  final RedisConnection _conn = RedisConnection();  
  Command? _command;  
  
  Future<void> connect({String host = 'localhost', int port = 6379}) async {  
    // TODO: Verbindung herstellen  
  }  
  
  Future<void> close() async {  
    // TODO: Verbindung schließen  
  }  
  
  // TODO: set, get, del Methoden  
}
```

### 3.8.1.4 Aufgabe 2: Basis-Operationen (15 min)

Implementiere die grundlegenden Cache-Methoden:

```
// String Operations  
Future<void> set(String key, String value, {Duration? ttl});  
Future<String?> get(String key);  
Future<void> del(String key);  
  
// JSON Helpers  
Future<void> setJson(String key, Map<String, dynamic> value, {Duration? ttl});  
Future<Map<String, dynamic>?> getJson(String key);  
  
// Hilfsmethoden  
Future<bool> exists(String key);  
Future<void> expire(String key, Duration ttl);
```

### 3.8.1.5 Aufgabe 3: Product Cache (20 min)

```
// lib/cache/product_cache.dart  
  
class ProductCache {
```

```

final RedisClient _redis;
final ProductRepository _repo;
final Duration _ttl;

ProductCache(this._redis, this._repo, {Duration? ttl})
    : _ttl = ttl ?? Duration(hours: 1);

/// Produkt aus Cache oder DB laden
Future<Product?> getProduct(int id) async {
    // TODO: Cache-Aside Pattern
    // 1. Cache prüfen
    // 2. Bei Miss: aus DB laden und cachen
}

/// Liste von Produkten cachen
Future<List<Product>> getProducts({String? category}) async {
    // TODO: Cache-Key basierend auf Parametern
    // z.B. "products:all" oder "products:category:electronics"
}

/// Cache invalidieren
Future<void> invalidateProduct(int id) async {
    // TODO: Produkt-Key löschen
}

Future<void> invalidateAll() async {
    // TODO: Alle Produkt-Keys löschen
}

/// Produkt aktualisieren (Write-Through)
Future<Product> updateProduct(int id, ProductUpdate data) async {
    // TODO: DB updaten + Cache aktualisieren
}
}

```

#### 3.8.1.6 Aufgabe 4: Session-Manager (15 min)

```

// lib/cache/session_manager.dart

class Session {
    final String id;
    final int userId;
    final String email;
    final DateTime createdAt;
    final DateTime expiresAt;

    // TODO: Konstruktor, toJson, fromJson
}

class SessionManager {

```

```

final RedisClient _redis;
final Duration _sessionDuration;

SessionManager(this._redis, {Duration? duration})
    : _sessionDuration = duration ?? Duration(hours: 24);

/// Neue Session erstellen
Future<Session> createSession(int userId, String email) async {
    // TODO: Session-ID generieren (UUID)
    // TODO: In Redis speichern mit TTL
}

/// Session validieren und laden
Future<Session?> getSession(String sessionId) async {
    // TODO: Session laden
    // TODO: Bei Erfolg: TTL erneuern (Sliding Expiration)
}

/// Session beenden
Future<void> destroySession(String sessionId) async {
    // TODO: Aus Redis löschen
}

/// Alle Sessions eines Users beenden
Future<void> destroyUserSessions(int userId) async {
    // TODO: Pattern-basiert suchen und löschen
}
}

```

### 3.8.1.7 Aufgabe 5: Rate Limiter (15 min)

```

// lib/cache/rate_limiter.dart

class RateLimitResult {
    final bool allowed;
    final int remaining;
    final int resetInSeconds;

    RateLimitResult({
        required this.allowed,
        required this.remaining,
        required this.resetInSeconds,
    });
}

class RateLimiter {
    final RedisClient _redis;

    RateLimiter(this._redis);
}

```

```

/// Sliding Window Rate Limiting
Future<RateLimitResult> checkLimit({
  required String clientId,
  int maxRequests = 100,
  int windowSeconds = 60,
}) async {
  // TODO: Sorted Set für Sliding Window
  // 1. Alte Einträge entfernen (ZREMRANGEBYSCORE)
  // 2. Anzahl prüfen (ZCARD)
  // 3. Bei erlaubt: Request hinzufügen (ZADD)
  // 4. TTL setzen (EXPIRE)
}
}

```

### 3.8.1.8 Aufgabe 6: Cache Middleware (10 min)

```

// Middleware für shelf

Middleware cacheMiddleware(ProductCache cache) {
  return (Handler handler) {
    return (Request request) async {
      // Nur GET-Requests cachen
      if (request.method != 'GET') {
        return handler(request);
      }

      final path = request.url.path;

      // Pattern: /api/products/:id
      final productMatch = RegExp(r'^api/products/(\d+)$').firstMatch(path);
      if (productMatch != null) {
        final id = int.parse(productMatch.group(1)!);
        final cached = await cache.getProduct(id);

        if (cached != null) {
          return Response.ok(
            jsonEncode(cached.toJson()),
            headers: {
              'content-type': 'application/json',
              'x-cache': 'HIT',
            },
          );
        }
      }

      // Cache Miss - normale Handler-Kette
      final response = await handler(request);
      return response.change(headers: {'x-cache': 'MISS'});
    };
  };
}

```

```
}
```

### 3.8.1.9 Aufgabe 7: Cache-Tags (Bonus, 15 min)

```
// lib/cache/tagged_cache.dart

class TaggedCache {
  final RedisClient _redis;

  TaggedCache(this._redis);

  /// Mit Tags cachen
  Future<void> set(
    String key,
    String value,
    List<String> tags, {
    Duration? ttl,
  }) async {
    // TODO: Wert speichern
    // TODO: Key zu Tag-Sets hinzufügen
  }

  /// Tag invalidieren (alle zugehörigen Keys löschen)
  Future<int> invalidateTag(String tag) async {
    // TODO: Keys aus Tag-Set holen
    // TODO: Alle Keys löschen
    // TODO: Tag-Set löschen
    // Return: Anzahl gelöschter Keys
  }

  /// Mehrere Tags invalidieren
  Future<int> invalidateTags(List<String> tags) async {
    // TODO
  }
}

// Verwendung:
// await cache.set('product:1', data, ['products', 'category:electronics']);
// await cache.invalidateTag('category:electronics');
```

### 3.8.1.10 Testen

```
Future<void> main() async {
  final redis = RedisClient();
  await redis.connect();

  // Basis-Tests
  await redis.set('test', 'value', ttl: Duration(seconds: 60));
  print(await redis.get('test')); // 'value'
}
```

```

// Product Cache
final cache = ProductCache(redis, productRepo);
final product = await cache.getProduct(1);
print('Product: ${product?.name}');

// Session
final sessions = SessionManager(redis);
final session = await sessions.createSession(1, 'test@example.com');
print('Session ID: ${session.id}');

// Rate Limiting
final limiter = RateLimiter(redis);
final result = await limiter.checkLimit(clientId: '127.0.0.1');
print('Allowed: ${result.allowed}, Remaining: ${result.remaining}');

await redis.close();
}

```

#### 3.8.1.11 Abgabe-Checkliste

- ☐ RedisClient mit connect/close
- ☐ set, get, del Methoden
- ☐ setJson, getJson Helpers
- ☐ ProductCache mit Cache-Aside
- ☐ Cache-Invalidierung (einzeln + alle)
- ☐ SessionManager mit create/get/destroy
- ☐ Sliding Expiration für Sessions
- ☐ RateLimiter mit Sliding Window
- ☐ (Bonus) TaggedCache
- ☐ (Bonus) Cache Middleware

### 3.8.2 Lösung

#### 3.8.2.1 Redis Client

```

// lib/cache/redis_client.dart
import 'dart:convert';
import 'package:redis/redis.dart';

class RedisClient {
  final RedisConnection _conn = RedisConnection();
  Command? _command;

  Future<void> connect({String host = 'localhost', int port = 6379}) async {
    _command = await _conn.connect(host, port);
  }

  Future<void> close() async {
    await _conn.close();
  }
}

```

```

Command get command {
    if (_command == null) throw StateError('Not connected');
    return _command!;
}

// String Operations
Future<void> set(String key, String value, {Duration? ttl}) async {
    if (ttl != null) {
        await command.send_object(['SET', key, value, 'EX',
↪ ttl.inSeconds.toString()]);
    } else {
        await command.send_object(['SET', key, value]);
    }
}

Future<String?> get(String key) async {
    final result = await command.send_object(['GET', key]);
    return result as String?;
}

Future<void> del(String key) async {
    await command.send_object(['DEL', key]);
}

Future<void> delPattern(String pattern) async {
    final keys = await command.send_object(['KEYS', pattern]);
    if (keys != null && (keys as List).isNotEmpty) {
        await command.send_object(['DEL', ...keys]);
    }
}

Future<bool> exists(String key) async {
    final result = await command.send_object(['EXISTS', key]);
    return (result as int) > 0;
}

Future<void> expire(String key, Duration ttl) async {
    await command.send_object(['EXPIRE', key, ttl.inSeconds.toString()]);
}

// JSON Helpers
Future<void> setJson(String key, Map<String, dynamic> value, {Duration?
↪ ttl}) async {
    await set(key, jsonEncode(value), ttl: ttl);
}

Future<Map<String, dynamic>?> getJson(String key) async {
    final data = await get(key);
    if (data == null) return null;
    return jsonDecode(data) as Map<String, dynamic>;
}

```

```
}

// Hash Operations
Future<void> hset(String key, String field, String value) async {
    await command.send_object(['HSET', key, field, value]);
}

Future<void> hmset(String key, Map<String, String> fields) async {
    final args = ['HSET', key];
    fields.forEach((k, v) {
        args.add(k);
        args.add(v);
    });
    await command.send_object(args);
}

Future<String?> hget(String key, String field) async {
    final result = await command.send_object(['HGET', key, field]);
    return result as String?;
}

Future<Map<String, String>> hgetall(String key) async {
    final result = await command.send_object(['HGETALL', key]);
    if (result == null) return {};

    final list = result as List;
    final map = <String, String>{};
    for (var i = 0; i < list.length; i += 2) {
        map[list[i] as String] = list[i + 1] as String;
    }
    return map;
}

// Set Operations
Future<void> sadd(String key, String value) async {
    await command.send_object(['SADD', key, value]);
}

Future<List<String>> smembers(String key) async {
    final result = await command.send_object(['SMEMBERS', key]);
    if (result == null) return [];
    return (result as List).cast<String>();
}

// Sorted Set Operations
Future<void> zadd(String key, double score, String member) async {
    await command.send_object(['ZADD', key, score.toString(), member]);
}

Future<int> zcard(String key) async {
    final result = await command.send_object(['ZCARD', key]);
}
```

```

    return result as int;
  }

  Future<void> zremrangebyscore(String key, double min, double max) async {
    await command.send_object(['ZREMRANGEBYSCORE', key, min.toString(),
↪    max.toString()]);
  }
}

```

### 3.8.2.2 Product Cache

```

// lib/cache/product_cache.dart
import 'dart:convert';

class ProductCache {
  final RedisClient _redis;
  final ProductRepository _repo;
  final Duration _ttl;

  ProductCache(this._redis, this._repo, {Duration? ttl})
    : _ttl = ttl ?? const Duration(hours: 1);

  String _productKey(int id) => 'product:$id';
  String _listKey(String? category) =>
    category != null ? 'products:category:$category' : 'products:all';

  Future<Product?> getProduct(int id) async {
    final key = _productKey(id);

    // 1. Cache prüfen
    final cached = await _redis.getJson(key);
    if (cached != null) {
      return Product.fromJson(cached);
    }

    // 2. Aus DB laden
    final product = await _repo.findById(id);
    if (product == null) return null;

    // 3. In Cache speichern
    await _redis.setJson(key, product.toJson(), ttl: _ttl);

    return product;
  }

  Future<List<Product>> getProducts({String? category}) async {
    final key = _listKey(category);

    // Cache prüfen
    final cached = await _redis.get(key);

```

```

    if (cached != null) {
        final list = jsonDecode(cached) as List;
        return list.map((e) => Product.fromJson(e)).toList();
    }

    // Aus DB laden
    final products = category != null
        ? await _repo.findByCategory(category)
        : await _repo.findAll();

    // Cachen
    await _redis.set(
        key,
        jsonEncode(products.map((p) => p.toJson()).toList()),
        ttl: _ttl,
    );

    return products;
}

Future<void> invalidateProduct(int id) async {
    await _redis.del(_productKey(id));
    // Listen auch invalidieren
    await _redis.delPattern('products:*');
}

Future<void> invalidateAll() async {
    await _redis.delPattern('product:*');
    await _redis.delPattern('products:*');
}

Future<Product> updateProduct(int id, ProductUpdate data) async {
    // 1. DB updaten
    final product = await _repo.update(id, data);
    if (product == null) throw NotFoundException('Product not found');

    // 2. Cache aktualisieren
    await _redis.setJson(_productKey(id), product.toJson(), ttl: _ttl);

    // 3. Listen invalidieren
    await _redis.delPattern('products:*');

    return product;
}
}

```

### 3.8.2.3 Session Manager

```

// lib/cache/session_manager.dart
import 'dart:convert';

```

```
import 'package:uuid/uuid.dart';

class Session {
  final String id;
  final int userId;
  final String email;
  final DateTime createdAt;
  final DateTime expiresAt;

  Session({
    required this.id,
    required this.userId,
    required this.email,
    required this.createdAt,
    required this.expiresAt,
  });

  factory Session.fromJson(Map<String, dynamic> json) {
    return Session(
      id: json['id'] as String,
      userId: json['userId'] as int,
      email: json['email'] as String,
      createdAt: DateTime.parse(json['createdAt'] as String),
      expiresAt: DateTime.parse(json['expiresAt'] as String),
    );
  }

  Map<String, dynamic> toJson() => {
    'id': id,
    'userId': userId,
    'email': email,
    'createdAt': createdAt.toIso8601String(),
    'expiresAt': expiresAt.toIso8601String(),
  };

  bool get isExpired => DateTime.now().isAfter(expiresAt);
}

class SessionManager {
  final RedisClient _redis;
  final Duration _sessionDuration;

  SessionManager(this._redis, {Duration? duration})
    : _sessionDuration = duration ?? const Duration(hours: 24);

  String _sessionKey(String id) => 'session:$id';
  String _userSessionsKey(int userId) => 'user_sessions:$userId';

  Future<Session> createSession(int userId, String email) async {
    final sessionId = const Uuid().v4();
    final now = DateTime.now();
```

```

final session = Session(
    id: sessionId,
    userId: userId,
    email: email,
    createdAt: now,
    expiresAt: now.add(_sessionDuration),
);

// Session speichern
await _redis.setJson(_sessionKey(sessionId), session.toJson(), ttl:
↵ _sessionDuration);

// Session zu User-Sessions hinzufügen
await _redis.sadd(_userSessionsKey(userId), sessionId);

return session;
}

Future<Session?> getSession(String sessionId) async {
    final data = await _redis.getJson(_sessionKey(sessionId));
    if (data == null) return null;

    final session = Session.fromJson(data);

    // Sliding Expiration: TTL erneuern
    await _redis.expire(_sessionKey(sessionId), _sessionDuration);

    // expiresAt aktualisieren
    final updated = Session(
        id: session.id,
        userId: session.userId,
        email: session.email,
        createdAt: session.createdAt,
        expiresAt: DateTime.now().add(_sessionDuration),
    );
    await _redis.setJson(_sessionKey(sessionId), updated.toJson(), ttl:
↵ _sessionDuration);
↵

    return updated;
}

Future<void> destroySession(String sessionId) async {
    // Session-Daten laden für User-ID
    final data = await _redis.getJson(_sessionKey(sessionId));
    if (data != null) {
        final userId = data['userId'] as int;
        // Aus User-Sessions entfernen
        await _redis.command.send_object(['SREM', _userSessionsKey(userId),
↵ sessionId]);
↵
    }
}

```

```

    // Session löschen
    await _redis.del(_sessionKey(sessionId));
  }

Future<void> destroyUserSessions(int userId) async {
  final sessionIds = await _redis.smembers(_userSessionsKey(userId));

  for (final sessionId in sessionIds) {
    await _redis.del(_sessionKey(sessionId));
  }

  await _redis.del(_userSessionsKey(userId));
}
}

```

### 3.8.2.4 Rate Limiter

```

// lib/cache/rate_limiter.dart

class RateLimitResult {
  final bool allowed;
  final int remaining;
  final int resetInSeconds;

  RateLimitResult({
    required this.allowed,
    required this.remaining,
    required this.resetInSeconds,
  });
}

class RateLimiter {
  final RedisClient _redis;

  RateLimiter(this._redis);

  Future<RateLimitResult> checkLimit({
    required String clientId,
    int maxRequests = 100,
    int windowSeconds = 60,
  }) async {
    final key = 'ratelimit:$clientId';
    final now = DateTime.now().millisecondsSinceEpoch;
    final windowStart = now - (windowSeconds * 1000);

    // 1. Alte Einträge entfernen
    await _redis.zremrangebyscore(key, 0, windowStart.toDouble());

    // 2. Aktuelle Anzahl

```

```

final count = await _redis.zcard(key);

if (count >= maxRequests) {
  // Rate limit exceeded
  final oldestEntry = await _redis.command.send_object([
    'ZRANGE', key, '0', '0', 'WITHSCORES'
  ]);

  int resetIn = windowSeconds;
  if (oldestEntry != null && (oldestEntry as List).length >= 2) {
    final oldestTime = double.parse(oldestEntry[1] as String).toInt();
    resetIn = ((oldestTime + windowSeconds * 1000) - now) ~/ 1000;
    if (resetIn < 0) resetIn = 0;
  }

  return RateLimitResult(
    allowed: false,
    remaining: 0,
    resetInSeconds: resetIn,
  );
}

// 3. Request hinzufügen
await _redis.zadd(key, now.toDouble(), '$now');

// 4. TTL setzen
await _redis.expire(key, Duration(seconds: windowSeconds));

return RateLimitResult(
  allowed: true,
  remaining: maxRequests - count - 1,
  resetInSeconds: windowSeconds,
);
}
}

```

### 3.8.2.5 Tagged Cache

```

// lib/cache/tagged_cache.dart

class TaggedCache {
  final RedisClient _redis;

  TaggedCache(this._redis);

  String _tagKey(String tag) => 'tag:$tag';

  Future<void> set(
    String key,
    String value,

```

```
List<String> tags, {
    Duration? ttl,
}) async {
    // Wert speichern
    await _redis.set(key, value, ttl: ttl);

    // Key zu Tag-Sets hinzufügen
    for (final tag in tags) {
        await _redis.sadd(_tagKey(tag), key);
    }
}

Future<String?> get(String key) async {
    return _redis.get(key);
}

Future<int> invalidateTag(String tag) async {
    final tagKey = _tagKey(tag);

    // Keys aus Tag-Set holen
    final keys = await _redis.smembers(tagKey);

    if (keys.isEmpty) return 0;

    // Alle Keys löschen
    for (final key in keys) {
        await _redis.del(key);
    }

    // Tag-Set löschen
    await _redis.del(tagKey);

    return keys.length;
}

Future<int> invalidateTags(List<String> tags) async {
    int total = 0;
    for (final tag in tags) {
        total += await invalidateTag(tag);
    }
    return total;
}
}
```

#### 3.8.2.6 Main

```
Future<void> main() async {
    final redis = RedisClient();
    await redis.connect();
}
```

```

print('=== Redis Connected ===\n');

// Basis-Test
await redis.set('test', 'hello', ttl: Duration(seconds: 60));
print('GET test: ${await redis.get("test")}');

// Session Test
final sessions = SessionManager(redis);
final session = await sessions.createSession(1, 'test@example.com');
print('\nSession created: ${session.id}');

final loaded = await sessions.getSession(session.id);
print('Session loaded: ${loaded?.email}');

// Rate Limiter Test
final limiter = RateLimiter(redis);
for (var i = 0; i < 5; i++) {
    final result = await limiter.checkLimit(
        clientId: 'test-client',
        maxRequests: 3,
        windowSeconds: 60,
    );
    print('\nRequest ${i + 1}: allowed=${result.allowed},
↪ remaining=${result.remaining}');
}

// Tagged Cache Test
final taggedCache = TaggedCache(redis);
await taggedCache.set('product:1', '{"name":"Laptop"}', ['products',
↪ 'electronics']);
await taggedCache.set('product:2', '{"name":"Mouse"}', ['products',
↪ 'electronics']);
await taggedCache.set('product:3', '{"name":"Shirt"}', ['products',
↪ 'clothing']);

print('\nInvalidating electronics tag...');
final deleted = await taggedCache.invalidateTag('electronics');
print('Deleted $deleted keys');

await redis.close();
print('\n=== Redis Closed ===');
}

```

### 3.8.3 Ressourcen

#### 3.8.3.1 Offizielle Dokumentation

- redis Package (pub.dev)
- Redis Documentation
- Redis Commands

### 3.8.3.2 Cheat Sheet: Verbindung

```
import 'package:redis/redis.dart';

final conn = RedisConnection();
final command = await conn.connect('localhost', 6379);

// Mit Passwort
await command.send_object(['AUTH', 'password']);

// Schließen
await conn.close();
```

### 3.8.3.3 Cheat Sheet: String Commands

```
// SET
await cmd.send_object(['SET', 'key', 'value']);

// SET mit TTL (Sekunden)
await cmd.send_object(['SET', 'key', 'value', 'EX', '3600']);

// SET mit TTL (Millisekunden)
await cmd.send_object(['SET', 'key', 'value', 'PX', '60000']);

// SET nur wenn nicht existiert
await cmd.send_object(['SETNX', 'key', 'value']);

// GET
final value = await cmd.send_object(['GET', 'key']);

// DEL
await cmd.send_object(['DEL', 'key']);

// INCR / DECR
await cmd.send_object(['INCR', 'counter']);
await cmd.send_object(['INCRBY', 'counter', '5']);

// EXPIRE
await cmd.send_object(['EXPIRE', 'key', '3600']);

// TTL (verbleibende Zeit)
final ttl = await cmd.send_object(['TTL', 'key']);
```

### 3.8.3.4 Cheat Sheet: Hash Commands

```
// HSET
await cmd.send_object(['HSET', 'user:1', 'name', 'Max', 'email',
  ↪ 'max@example.com']);

// HGET
```

```
final name = await cmd.send_object(['HGET', 'user:1', 'name']);

// HGETALL
final user = await cmd.send_object(['HGETALL', 'user:1']);
// ['name', 'Max', 'email', 'max@example.com']

// HINCRBY
await cmd.send_object(['HINCRBY', 'user:1', 'visits', '1']);
```

### 3.8.3.5 Cheat Sheet: List Commands

```
// RPU SH (Ende)
await cmd.send_object(['RPU SH', 'queue', 'item1', 'item2']);

// LPU SH (Anfang)
await cmd.send_object(['LPU SH', 'queue', 'item0']);

// LPOP (vom Anfang)
final item = await cmd.send_object(['LPOP', 'queue']);

// RPOP (vom Ende)
final item = await cmd.send_object(['RPOP', 'queue']);

// LRANGE (Bereich)
final items = await cmd.send_object(['LRANGE', 'queue', '0', '-1']);
```

### 3.8.3.6 Cheat Sheet: Set Commands

```
// SADD
await cmd.send_object(['SADD', 'tags', 'new', 'sale']);

// S MEMBERS
final tags = await cmd.send_object(['S MEMBERS', 'tags']);

// SISMEMBER
final exists = await cmd.send_object(['SISMEMBER', 'tags', 'new']);

// SREM
await cmd.send_object(['SREM', 'tags', 'sale']);
```

### 3.8.3.7 Cheat Sheet: Sorted Set Commands

```
// ZADD
await cmd.send_object(['ZADD', 'leaderboard', '100', 'player1']);

// ZRANGE (aufsteigend)
final bottom = await cmd.send_object(['ZRANGE', 'leaderboard', '0', '9']);

// ZREVRANGE (absteigend)
```

```
final top = await cmd.send_object(['ZREVRANGE', 'leaderboard', '0', '9',  
  ↪ 'WITHSCORES']);  
  
// ZRANK / ZREVRANK  
final rank = await cmd.send_object(['ZREVRANK', 'leaderboard', 'player1']);  
  
// ZINCRBY  
await cmd.send_object(['ZINCRBY', 'leaderboard', '10', 'player1']);
```

### 3.8.3.8 Caching Strategien

Strategie	Beschreibung	Use Case
Cache-Aside	Bei Miss aus DB laden	Lesen-intensiv
Write-Through	Bei Write sofort cachen	Konsistenz wichtig
Write-Behind	Async in DB schreiben	Hohe Schreiblast
Read-Through	Cache lädt selbst	Transparentes Caching

### 3.8.3.9 Best Practices

1. **Kurze TTLs** - Lieber oft refreshen als stale data
2. **Konsistente Keys** - `type:id:field` Konvention
3. **JSON für komplexe Objekte** - Einfach zu serialisieren
4. **Pipelines für Bulk-Ops** - Mehrere Commands in einem Roundtrip
5. **Memory-Limits setzen** - Redis wächst unbegrenzt
6. **Invalidierung bei Writes** - Write-Through oder explizit

## Chapter 4

# Block 8: Authentifizierung & Sicherheit

Passwort-Hashing, JWT, OAuth, API-Sicherheit und Auth-Testing.

## 4.1 Einheit 8.1: Passwort Hashing

### 4.1.0.1 Lernziele

Nach dieser Einheit kannst du: - Passwörter sicher hashen mit bcrypt - Salt und Hashing-Algorithmen verstehen - Einen User-Registrierungsflow implementieren - Passwort-Validierung durchführen

### 4.1.0.2 Warum Passwort-Hashing?

**Niemals Passwörter im Klartext speichern!**

Wenn deine Datenbank kompromittiert wird: - Klartext: Sofort alle Accounts übernommen - Gehashte Passwörter: Angreifer muss jeden Hash knacken

Was ist Hashing?

Passwort -> Hash-Funktion -> Hash (nicht umkehrbar)  
"geheim123" -> bcrypt -> "\$2a\$12\$LQv3c1yqBW..."

**Eigenschaften einer guten Hash-Funktion:** - **Einweg:** Hash -> Passwort nicht möglich - **Deterministisch:** Gleiches Passwort = Gleicher Hash - **Kollisionsresistent:** Unterschiedliche Passwörter Gleicher Hash - **Langsam:** Brute-Force erschweren

### 4.1.0.3 Hashing-Algorithmen

Nicht geeignet (zu schnell!)

```
// NIEMALS verwenden für Passwörter!  
import 'dart:convert';  
import 'crypto/crypto.dart';  
  
final hash = md5.convert(utf8.encode('password')).toString();  
final hash2 = sha256.convert(utf8.encode('password')).toString();
```

MD5 und SHA sind für Passwörter **ungeeignet** - zu schnell, milliarden Hashes pro Sekunde möglich.

Geeignete Algorithmen

Algorithmus	Beschreibung	Empfehlung
<b>bcrypt</b>	Bewährt, überall verfügbar	Standard-Wahl
<b>Argon2</b>	Modernster Algorithmus	Beste Sicherheit
<b>scrypt</b>	Memory-hard	Gute Alternative

#### 4.1.0.4 bcrypt in Dart

Installation

```
dependencies:
  bcrypt: ^1.1.3
```

Passwort hashen

```
import 'package:bcrypt/bcrypt.dart';

class PasswordService {
  // Cost Factor: Erhöht Rechenzeit exponentiell
  // 10 = ~100ms, 12 = ~300ms, 14 = ~1s
  static const int _costFactor = 12;

  /// Passwort hashen
  String hashPassword(String password) {
    // Salt wird automatisch generiert und im Hash gespeichert
    return BCrypt.hashpw(password, BCrypt.gensalt(logRounds: _costFactor));
  }

  /// Passwort verifizieren
  bool verifyPassword(String password, String hashedPassword) {
    return BCrypt.checkpw(password, hashedPassword);
  }
}
```

bcrypt Hash-Format

```
$2a$12$LQv3c1yqBWHxkd0LHAKCOYz6TtxMQJqhN8/X4.N1tJ9vPmUHPK3a
| | |                                     |
| | |                                     +-- Hash (31 Zeichen)
| | +-- Salt (22 Zeichen, Base64)
| +-- Cost Factor (12 Runden = 2^12 Iterationen)
+-- Algorithmus-Version (2a = bcrypt)
```

#### 4.1.0.5 Was ist Salt?

**Salt** ist ein zufälliger Wert, der dem Passwort vor dem Hashing hinzugefügt wird.

Ohne Salt (Rainbow Table Attack)

```
"password" -> hash("password") -> "5f4dcc3b5aa765d61d8327deb882cf99"
```

Angreifer kann vorberechnete Tabellen (Rainbow Tables) verwenden.

Mit Salt

```
salt = "x7Km9pQ2"
```

"password" + salt -> hash -> "\$2a\$12\$x7Km9pQ2..."

- Gleiche Passwörter haben unterschiedliche Hashes
- Rainbow Tables werden nutzlos
- bcrypt speichert Salt automatisch im Hash

#### 4.1.0.6 User-Model

```
// lib/models/user.dart

class User {
  final int? id;
  final String email;
  final String passwordHash;
  final String? name;
  final DateTime createdAt;
  final DateTime? updatedAt;
  final bool isActive;
  final String role;

  User({
    this.id,
    required this.email,
    required this.passwordHash,
    this.name,
    DateTime? createdAt,
    this.updatedAt,
    this.isActive = true,
    this.role = 'user',
  }) : createdAt = createdAt ?? DateTime.now();

  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      id: json['id'] as int?,
      email: json['email'] as String,
      passwordHash: json['password_hash'] as String,
      name: json['name'] as String?,
      createdAt: DateTime.parse(json['created_at'] as String),
      updatedAt: json['updated_at'] != null
        ? DateTime.parse(json['updated_at'] as String)
        : null,
      isActive: json['is_active'] as bool? ?? true,
      role: json['role'] as String? ?? 'user',
    );
  }

  Map<String, dynamic> toJson() => {
    if (id != null) 'id': id,
    'email': email,
    'password_hash': passwordHash,
    'name': name,
    'created_at': createdAt.toIso8601String(),
  }
}
```

```
    'updated_at': updatedAt?.toIso8601String(),
    'is_active': isActive,
    'role': role,
  };

  /// Für API-Responses (ohne sensible Daten)
  Map<String, dynamic> toPublicJson() => {
    'id': id,
    'email': email,
    'name': name,
    'created_at': createdAt.toIso8601String(),
    'role': role,
  };
}
```

#### 4.1.0.7 Registrierungs-DTOs

```
// lib/models/auth_dto.dart

class RegisterRequest {
  final String email;
  final String password;
  final String? name;

  RegisterRequest({
    required this.email,
    required this.password,
    this.name,
  });

  factory RegisterRequest.fromJson(Map<String, dynamic> json) {
    return RegisterRequest(
      email: json['email'] as String,
      password: json['password'] as String,
      name: json['name'] as String?,
    );
  }

  List<String> validate() {
    final errors = <String>[];

    // Email validieren
    if (!_isValidEmail(email)) {
      errors.add('Invalid email format');
    }

    // Passwort validieren
    errors.addAll(validatePassword(password));

    return errors;
  }
}
```

```

}

bool _isValidEmail(String email) {
  return RegExp(r'^[\w-\.]@([\w-]+\.)+[\w-]{2,4}$').hasMatch(email);
}

static List<String> validatePassword(String password) {
  final errors = <String>[];

  if (password.length < 8) {
    errors.add('Password must be at least 8 characters');
  }
  if (!password.contains(RegExp(r'[A-Z]'))) {
    errors.add('Password must contain an uppercase letter');
  }
  if (!password.contains(RegExp(r'[a-z]'))) {
    errors.add('Password must contain a lowercase letter');
  }
  if (!password.contains(RegExp(r'[0-9]'))) {
    errors.add('Password must contain a number');
  }
  if (!password.contains(RegExp(r'[!@#$$%^&*(),.?":{}|<>]'))) {
    errors.add('Password must contain a special character');
  }

  return errors;
}
}

class LoginRequest {
  final String email;
  final String password;

  LoginRequest({
    required this.email,
    required this.password,
  });

  factory LoginRequest.fromJson(Map<String, dynamic> json) {
    return LoginRequest(
      email: json['email'] as String,
      password: json['password'] as String,
    );
  }
}

```

#### 4.1.0.8 User Repository

```

// lib/repositories/user_repository.dart
import 'package:postgres/postgres.dart';

```

```
class UserRepository {
    final Connection _db;

    UserRepository(this._db);

    Future<User?> findByEmail(String email) async {
        final result = await _db.execute(
            Sql.named('SELECT * FROM users WHERE email = @email'),
            parameters: {'email': email},
        );

        if (result.isEmpty) return null;
        return User.fromJson(result.first.toColumnMap());
    }

    Future<User?> findById(int id) async {
        final result = await _db.execute(
            Sql.named('SELECT * FROM users WHERE id = @id'),
            parameters: {'id': id},
        );

        if (result.isEmpty) return null;
        return User.fromJson(result.first.toColumnMap());
    }

    Future<User> create(User user) async {
        final result = await _db.execute(
            Sql.named(''
                INSERT INTO users (email, password_hash, name, role, is_active,
↵ created_at)
↵ VALUES (@email, @passwordHash, @name, @role, @isActive, @createdAt)
                RETURNING *
            '''),
            parameters: {
                'email': user.email,
                'passwordHash': user.passwordHash,
                'name': user.name,
                'role': user.role,
                'isActive': user.isActive,
                'createdAt': user.createdAt,
            },
        );

        return User.fromJson(result.first.toColumnMap());
    }

    Future<bool> emailExists(String email) async {
        final result = await _db.execute(
            Sql.named('SELECT 1 FROM users WHERE email = @email'),
            parameters: {'email': email},
        );
    }
}
```

```
);
return result.isNotEmpty;
}

Future<void> updatePassword(int userId, String newPasswordHash) async {
  await _db.execute(
    Sql.named('''
      UPDATE users
      SET password_hash = @passwordHash, updated_at = NOW()
      WHERE id = @id
    '''),
    parameters: {
      'id': userId,
      'passwordHash': newPasswordHash,
    },
  );
}
```

#### 4.1.0.9 Auth Service

```
// lib/services/auth_service.dart
import 'package:bcrypt/bcrypt.dart';

class AuthException implements Exception {
  final String message;
  final int statusCode;

  AuthException(this.message, {this.statusCode = 400});
}

class AuthService {
  final UserRepository _userRepo;
  static const int _bcryptCost = 12;

  AuthService(this._userRepo);

  /// Benutzer registrieren
  Future<User> register(RegisterRequest request) async {
    // 1. Validierung
    final errors = request.validate();
    if (errors.isNotEmpty) {
      throw AuthException(errors.join(', '));
    }

    // 2. Email-Duplikat prüfen
    if (await _userRepo.emailExists(request.email)) {
      throw AuthException('Email already registered', statusCode: 409);
    }
  }
}
```

```
// 3. Passwort hashen
final passwordHash = BCrypt.hashpw(
    request.password,
    BCrypt.gensalt(logRounds: _bcryptCost),
);

// 4. User erstellen
final user = User(
    email: request.email.toLowerCase().trim(),
    passwordHash: passwordHash,
    name: request.name?.trim(),
);

return await _userRepo.create(user);
}

/// Benutzer authentifizieren
Future<User> authenticate(LoginRequest request) async {
    // 1. User laden
    final user = await _userRepo.findByEmail(request.email.toLowerCase());

    if (user == null) {
        // Timing-Attack verhindern: Immer hashen, auch wenn User nicht existiert
        BCrypt.hashpw('dummy', BCrypt.gensalt(logRounds: _bcryptCost));
        throw AuthException('Invalid credentials', statusCode: 401);
    }

    // 2. Account aktiv?
    if (!user.isActive) {
        throw AuthException('Account is deactivated', statusCode: 403);
    }

    // 3. Passwort prüfen
    if (!BCrypt.checkpw(request.password, user.passwordHash)) {
        throw AuthException('Invalid credentials', statusCode: 401);
    }

    return user;
}

/// Passwort ändern
Future<void> changePassword({
    required int userId,
    required String currentPassword,
    required String newPassword,
}) async {
    // 1. User laden
    final user = await _userRepo.findById(userId);
    if (user == null) {
        throw AuthException('User not found', statusCode: 404);
    }
}
```

```

// 2. Aktuelles Passwort prüfen
if (!BCrypt.checkpw(currentPassword, user.passwordHash)) {
  throw AuthException('Current password is incorrect', statusCode: 401);
}

// 3. Neues Passwort validieren
final errors = RegisterRequest.validatePassword(newPassword);
if (errors.isNotEmpty) {
  throw AuthException(errors.join(', '));
}

// 4. Neues Passwort hashen und speichern
final newHash = BCrypt.hashpw(
  newPassword,
  BCrypt.gensalt(logRounds: _bcryptCost),
);

await _userRepo.updatePassword(userId, newHash);
}
}

```

#### 4.1.0.10 Auth Handler

```

// lib/handlers/auth_handler.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';

class AuthHandler {
  final AuthService _authService;

  AuthHandler(this._authService);

  Router get router {
    final router = Router();

    router.post('/register', _register);
    router.post('/login', _login);

    return router;
  }

  Future<Response> _register(Request request) async {
    try {
      final body = await request.readAsString();
      final data = jsonDecode(body) as Map<String, dynamic>;
      final registerRequest = RegisterRequest.fromJson(data);

      final user = await _authService.register(registerRequest);

```

```
        return Response(
            201,
            body: jsonEncode({
                'message': 'User registered successfully',
                'user': user.toPublicJson(),
            }),
            headers: {'content-type': 'application/json'},
        );
    } on AuthException catch (e) {
        return Response(
            e.statusCode,
            body: jsonEncode({'error': e.message}),
            headers: {'content-type': 'application/json'},
        );
    } catch (e) {
        return Response.internalServerError(
            body: jsonEncode({'error': 'Registration failed'}),
            headers: {'content-type': 'application/json'},
        );
    }
}

Future<Response> _login(Request request) async {
    try {
        final body = await request.readAsString();
        final data = jsonDecode(body) as Map<String, dynamic>;
        final loginRequest = LoginRequest.fromJson(data);

        final user = await _authService.authenticate(loginRequest);

        // TODO: JWT-Token generieren (nächste Einheit)
        return Response.ok(
            jsonEncode({
                'message': 'Login successful',
                'user': user.toPublicJson(),
            }),
            headers: {'content-type': 'application/json'},
        );
    } on AuthException catch (e) {
        return Response(
            e.statusCode,
            body: jsonEncode({'error': e.message}),
            headers: {'content-type': 'application/json'},
        );
    } catch (e) {
        return Response.internalServerError(
            body: jsonEncode({'error': 'Login failed'}),
            headers: {'content-type': 'application/json'},
        );
    }
}
```

```
}
}
```

#### 4.1.0.11 Datenbank-Schema

```
-- migrations/001_create_users.sql

CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  name VARCHAR(255),
  role VARCHAR(50) NOT NULL DEFAULT 'user',
  is_active BOOLEAN NOT NULL DEFAULT true,
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMP
);

-- Index für Login-Queries
CREATE INDEX idx_users_email ON users(email);

-- Index für aktive User
CREATE INDEX idx_users_active ON users(is_active) WHERE is_active = true;
```

#### 4.1.0.12 Sicherheits-Best-Practices

##### 1. Timing Attacks verhindern

```
// SCHLECHT: Verrät, ob Email existiert
if (user == null) {
  throw AuthException('Email not found'); // Schnelle Antwort
}
if (!verifyPassword()) {
  throw AuthException('Wrong password'); // Langsame Antwort
}

// GUT: Konstante Zeit für beide Fälle
if (user == null) {
  // Trotzdem hashen, um gleiche Antwortzeit zu haben
  BCrypt.hashpw('dummy', BCrypt.gensalt(logRounds: 12));
  throw AuthException('Invalid credentials');
}
if (!verifyPassword()) {
  throw AuthException('Invalid credentials');
}
```

##### 2. Generische Fehlermeldungen

```
// SCHLECHT
throw AuthException('Email not found');
throw AuthException('Wrong password');
```

```
// GUT
throw AuthException('Invalid credentials');
```

### 3. Rate Limiting

```
// Brute-Force-Schutz (mit Redis)
class LoginRateLimiter {
    final RedisClient _redis;
    final int maxAttempts = 5;
    final Duration window = Duration(minutes: 15);

    Future<bool> isBlocked(String email) async {
        final key = 'login_attempts:$email';
        final attempts = await _redis.get(key);
        return (int.tryParse(attempts ?? '0') ?? 0) >= maxAttempts;
    }

    Future<void> recordFailedAttempt(String email) async {
        final key = 'login_attempts:$email';
        await _redis.command.send_object(['INCR', key]);
        await _redis.expire(key, window);
    }

    Future<void> clearAttempts(String email) async {
        await _redis.del('login_attempts:$email');
    }
}
```

### 4. Account Lockout

```
// Nach 5 fehlgeschlagenen Versuchen Account sperren
if (await _rateLimiter.isBlocked(request.email)) {
    throw AuthException(
        'Account temporarily locked. Try again later.',
        statusCode: 429,
    );
}

// Bei Fehlschlag
await _rateLimiter.recordFailedAttempt(request.email);

// Bei Erfolg
await _rateLimiter.clearAttempts(request.email);
```

#### 4.1.0.13 Zusammenfassung

Konzept	Beschreibung
<b>Hashing</b>	Einweg-Transformation, nicht umkehrbar
<b>Salt</b>	Zufälliger Wert, verhindert Rainbow Tables
<b>bcrypt</b>	Standard für Passwort-Hashing, inkl. Salt

Konzept	Beschreibung
<b>Cost Factor</b>	Erhöht Rechenzeit exponentiell
<b>Timing Attack</b>	Konstante Antwortzeit wichtig
<b>Rate Limiting</b>	Brute-Force-Schutz

**Wichtigste Regeln:** 1. Niemals Klartext-Passwörter speichern 2. bcrypt oder Argon2 verwenden 3. Generische Fehlermeldungen 4. Rate Limiting implementieren 5. Passwort-Policies durchsetzen

### 4.1.1 Übung

#### 4.1.1.1 Ziel

Implementiere ein sicheres User-Management-System mit Passwort-Hashing.

#### 4.1.1.2 Vorbereitung

Projekt erstellen

```
dart create -t server-shelf auth_api
cd auth_api
```

Dependencies

```
dependencies:
  shelf: ^1.4.0
  shelf_router: ^1.1.0
  bcrypt: ^1.1.3
  postgres: ^3.0.0

dev_dependencies:
  test: ^1.24.0
```

Datenbank

```
docker run --name postgres -e POSTGRES_PASSWORD=postgres -p 5432:5432 -d postgres:15
```

```
CREATE DATABASE auth_db;
```

```
\c auth_db
```

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  name VARCHAR(255),
  role VARCHAR(50) NOT NULL DEFAULT 'user',
  is_active BOOLEAN NOT NULL DEFAULT true,
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMP
);
```

#### 4.1.1.3 Aufgabe 1: Password Service (15 min)

Implementiere den PasswordService mit bcrypt:

```
// lib/services/password_service.dart

class PasswordService {
  final int _costFactor;

  PasswordService({int costFactor = 12}) : _costFactor = costFactor;

  /// Hash ein Passwort mit bcrypt
  String hash(String password) {
    // TODO: bcrypt verwenden
  }

  /// Verifiziere ein Passwort gegen einen Hash
  bool verify(String password, String hash) {
    // TODO: bcrypt.checkpw verwenden
  }

  /// Prüfe ob ein Hash neu gehasht werden sollte
  /// (z.B. wenn Cost Factor erhöht wurde)
  bool needsRehash(String hash) {
    // TODO: Cost Factor aus Hash extrahieren und vergleichen
  }
}
```

Test:

```
void main() {
  final service = PasswordService(costFactor: 10);

  final hash = service.hash('MySecurePassword123!');
  print('Hash: $hash');
  print('Verify correct: ${service.verify('MySecurePassword123!', hash)}');
  print('Verify wrong: ${service.verify('WrongPassword', hash)}');
}
```

#### 4.1.1.4 Aufgabe 2: Passwort-Validierung (15 min)

Implementiere einen PasswordValidator:

```
// lib/validators/password_validator.dart

class PasswordValidationResult {
  final bool isValid;
  final List<String> errors;
  final int strength; // 0-4

  PasswordValidationResult({
    required this.isValid,
    required this.errors,
```

```
        required this.strength,
    });
}

class PasswordValidator {
    final int minLength;
    final bool requireUppercase;
    final bool requireLowercase;
    final bool requireDigit;
    final bool requireSpecialChar;

    PasswordValidator({
        this.minLength = 8,
        this.requireUppercase = true,
        this.requireLowercase = true,
        this.requireDigit = true,
        this.requireSpecialChar = true,
    });

    PasswordValidationResult validate(String password) {
        final errors = <String>[];
        int strength = 0;

        // TODO: Implementiere Validierungsregeln
        // - Mindestlänge prüfen
        // - Großbuchstaben prüfen
        // - Kleinbuchstaben prüfen
        // - Ziffern prüfen
        // - Sonderzeichen prüfen
        // - Stärke berechnen (0-4)

        return PasswordValidationResult(
            isValid: errors.isEmpty,
            errors: errors,
            strength: strength,
        );
    }

    /// Prüft ob Passwort in Common-Password-Liste
    bool isCommonPassword(String password) {
        final common = [
            'password', '123456', '12345678', 'qwerty', 'abc123',
            'password1', 'admin', 'letmein', 'welcome', 'monkey',
        ];
        return common.contains(password.toLowerCase());
    }
}
```

#### 4.1.1.5 Aufgabe 3: User Model & Repository (20 min)

User Model

```
// lib/models/user.dart

class User {
  final int? id;
  final String email;
  final String passwordHash;
  final String? name;
  final String role;
  final bool isActive;
  final DateTime createdAt;
  final DateTime? updatedAt;

  User({
    this.id,
    required this.email,
    required this.passwordHash,
    this.name,
    this.role = 'user',
    this.isActive = true,
    DateTime? createdAt,
    this.updatedAt,
  }) : createdAt = createdAt ?? DateTime.now();

  // TODO: fromJson, toJson, toPublicJson implementieren
}
```

#### User Repository

```
// lib/repositories/user_repository.dart

abstract class UserRepository {
  Future<User?> findByEmail(String email);
  Future<User?> findById(int id);
  Future<User> create(User user);
  Future<bool> emailExists(String email);
  Future<void> updatePassword(int userId, String newPasswordHash);
  Future<void> deactivate(int userId);
}

class PostgresUserRepository implements UserRepository {
  final Connection _db;

  PostgresUserRepository(this._db);

  // TODO: Alle Methoden implementieren
}
```

#### 4.1.1.6 Aufgabe 4: Auth Service (20 min)

```
// lib/services/auth_service.dart
```

```
class AuthResult {
    final bool success;
    final User? user;
    final String? error;
    final int statusCode;

    AuthResult.success(this.user)
        : success = true,
          error = null,
          statusCode = 200;

    AuthResult.failure(this.error, {this.statusCode = 400})
        : success = false,
          user = null;
}

class AuthService {
    final UserRepository _userRepo;
    final PasswordService _passwordService;
    final PasswordValidator _passwordValidator;

    AuthService(this._userRepo, this._passwordService, this._passwordValidator);

    /// Registriere einen neuen Benutzer
    Future<AuthResult> register({
        required String email,
        required String password,
        String? name,
    }) async {
        // TODO:
        // 1. Email validieren (Format)
        // 2. Email normalisieren (lowercase, trim)
        // 3. Prüfen ob Email schon existiert
        // 4. Passwort validieren
        // 5. Passwort hashen
        // 6. User erstellen
    }

    /// Authentifiziere einen Benutzer
    Future<AuthResult> authenticate({
        required String email,
        required String password,
    }) async {
        // TODO:
        // 1. User laden
        // 2. Timing-Attack verhindern (auch bei nicht existierendem User hashen)
        // 3. Prüfen ob Account aktiv
        // 4. Passwort verifizieren
        // 5. Optional: Rehash wenn nötig
    }
}
```

```
/// Passwort ändern
Future<AuthResult> changePassword({
  required int userId,
  required String currentPassword,
  required String newPassword,
}) async {
  // TODO:
  // 1. User laden
  // 2. Aktuelles Passwort prüfen
  // 3. Neues Passwort validieren
  // 4. Neues Passwort hashen
  // 5. Speichern
}
}
```

#### 4.1.1.7 Aufgabe 5: Auth Handler (15 min)

```
// lib/handlers/auth_handler.dart

class AuthHandler {
  final AuthService _authService;

  AuthHandler(this._authService);

  Router get router {
    final router = Router();

    router.post('/register', _register);
    router.post('/login', _login);

    return router;
  }

  Future<Response> _register(Request request) async {
    // TODO:
    // 1. Body parsen
    // 2. AuthService.register aufrufen
    // 3. Response erstellen (201 bei Erfolg)
  }

  Future<Response> _login(Request request) async {
    // TODO:
    // 1. Body parsen
    // 2. AuthService.authenticate aufrufen
    // 3. Response erstellen
  }
}
```

**4.1.1.8 Aufgabe 6: Rate Limiter (Bonus, 15 min)**

```
// lib/services/rate_limiter.dart

class RateLimitResult {
  final bool allowed;
  final int remaining;
  final int resetInSeconds;

  RateLimitResult({
    required this.allowed,
    required this.remaining,
    required this.resetInSeconds,
  });
}

class LoginRateLimiter {
  final Map<String, List<DateTime>> _attempts = {};
  final int maxAttempts;
  final Duration window;

  LoginRateLimiter({
    this.maxAttempts = 5,
    this.window = const Duration(minutes: 15),
  });

  /// Prüfe ob Login erlaubt ist
  RateLimitResult check(String identfier) {
    // TODO:
    // 1. Alte Einträge entfernen
    // 2. Aktuelle Anzahl zählen
    // 3. RateLimitResult zurückgeben
  }

  /// Registriere einen fehlgeschlagenen Versuch
  void recordFailedAttempt(String identfier) {
    // TODO: Timestamp hinzufügen
  }

  /// Lösche alle Versuche nach erfolgreichem Login
  void clearAttempts(String identfier) {
    // TODO: Einträge löschen
  }
}
```

**4.1.1.9 Aufgabe 7: Main Server (10 min)**

```
// bin/server.dart

Future<void> main() async {
  // Database Connection
```

```

final db = await Connection.open(
  Endpoint(
    host: 'localhost',
    database: 'auth_db',
    username: 'postgres',
    password: 'postgres',
  ),
  settings: ConnectionSettings(sslMode: SslMode.disable),
);

// Services
final passwordService = PasswordService();
final passwordValidator = PasswordValidator();
final userRepo = PostgresUserRepository(db);
final authService = AuthService(userRepo, passwordService, passwordValidator);

// Handler
final authHandler = AuthHandler(authService);

// Router
final router = Router();
router.mount('/api/auth', authHandler.router);

// Server
final handler = const Pipeline()
  .addMiddleware(logRequests())
  .addHandler(router);

final server = await serve(handler, 'localhost', 8080);
print('Server running on http://${server.address.host}:${server.port}');
}

```

#### 4.1.1.10 Testen

##### Registrierung

```

# Erfolgreiche Registrierung
curl -X POST http://localhost:8080/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"email": "test@example.com", "password": "SecurePass123!", "name":
    ↪ "Test User"}'

# Schwaches Passwort
curl -X POST http://localhost:8080/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"email": "test2@example.com", "password": "weak"}'

# Duplicate Email
curl -X POST http://localhost:8080/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"email": "test@example.com", "password": "AnotherPass123!"}'

```

Login

```
# Erfolgreicher Login
curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email": "test@example.com", "password": "SecurePass123!"}'

# Falsches Passwort
curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email": "test@example.com", "password": "WrongPassword"}'
```

#### 4.1.1.11 Abgabe-Checkliste

- ☐ PasswordService mit hash/verify
- ☐ PasswordValidator mit Stärke-Berechnung
- ☐ User Model mit toPublicJson (ohne passwordHash)
- ☐ UserRepository mit allen Methoden
- ☐ AuthService mit register/authenticate/changePassword
- ☐ AuthHandler mit /register und /login Endpoints
- ☐ Timing-Attack-Schutz implementiert
- ☐ Generische Fehlermeldungen
- ☐ (Bonus) Rate Limiter

### 4.1.2 Lösung

#### 4.1.2.1 Password Service

```
// lib/services/password_service.dart
import 'package:bcrypt/bcrypt.dart';

class PasswordService {
  final int _costFactor;

  PasswordService({int costFactor = 12}) : _costFactor = costFactor;

  /// Hash ein Passwort mit bcrypt
  String hash(String password) {
    final salt = BCrypt.gensalt(logRounds: _costFactor);
    return BCrypt.hashpw(password, salt);
  }

  /// Verifiziere ein Passwort gegen einen Hash
  bool verify(String password, String hash) {
    try {
      return BCrypt.checkpw(password, hash);
    } catch (e) {
      return false;
    }
  }
}

/// Prüfe ob ein Hash neu gehasht werden sollte
```

```

bool needsRehash(String hash) {
  // bcrypt Hash Format: $2a$XX$...
  // XX = Cost Factor
  final match = RegExp(r'^\$2[aby]?\$(\d+)\$').firstMatch(hash);
  if (match == null) return true;

  final hashCost = int.tryParse(match.group(1) ?? '0') ?? 0;
  return hashCost < _costFactor;
}

/// Generiere einen zufälligen Salt (für Tests)
String generateSalt() {
  return BCrypt.gensalt(logRounds: _costFactor);
}
}

```

#### 4.1.2.2 Password Validator

```

// lib/validators/password_validator.dart

class PasswordValidationResult {
  final bool isValid;
  final List<String> errors;
  final int strength; // 0-4

  PasswordValidationResult({
    required this.isValid,
    required this.errors,
    required this.strength,
  });

  String get strengthLabel {
    switch (strength) {
      case 0:
        return 'Very Weak';
      case 1:
        return 'Weak';
      case 2:
        return 'Fair';
      case 3:
        return 'Strong';
      case 4:
        return 'Very Strong';
      default:
        return 'Unknown';
    }
  }
}

class PasswordValidator {

```

```
final int minLength;
final bool requireUppercase;
final bool requireLowercase;
final bool requireDigit;
final bool requireSpecialChar;

static const List<String> _commonPasswords = [
  'password', '123456', '12345678', 'qwerty', 'abc123',
  'password1', 'admin', 'letmein', 'welcome', 'monkey',
  'dragon', 'master', 'login', 'princess', 'sunshine',
  'passw0rd', 'shadow', 'trustno1', 'iloveyou', 'football',
];

PasswordValidator({
  this.minLength = 8,
  this.requireUppercase = true,
  this.requireLowercase = true,
  this.requireDigit = true,
  this.requireSpecialChar = true,
});

PasswordValidationResult validate(String password) {
  final errors = <String>[];
  int strength = 0;

  // Mindestlänge
  if (password.length < minLength) {
    errors.add('Password must be at least $minLength characters');
  } else {
    strength++;
  }

  // Bonus für längere Passwörter
  if (password.length >= 12) strength++;

  // Großbuchstaben
  final hasUppercase = password.contains(RegExp(r'[A-Z]'));
  if (requireUppercase && !hasUppercase) {
    errors.add('Password must contain at least one uppercase letter');
  }
  if (hasUppercase) strength++;

  // Kleinbuchstaben
  final hasLowercase = password.contains(RegExp(r'[a-z]'));
  if (requireLowercase && !hasLowercase) {
    errors.add('Password must contain at least one lowercase letter');
  }

  // Ziffern
  final hasDigit = password.contains(RegExp(r'[0-9]'));
  if (requireDigit && !hasDigit) {
```

```

        errors.add('Password must contain at least one digit');
    }
    if (hasDigit) strength++;

    // Sonderzeichen
    final hasSpecial =
↪ password.contains(RegExp(r'[!@#$%^&*(),.?":{}|<>_\-+=\[\]\|\;~/\`']'));
    if (requireSpecialChar && !hasSpecial) {
        errors.add('Password must contain at least one special character');
    }
    if (hasSpecial) strength++;

    // Common Password Check
    if (isCommonPassword(password)) {
        errors.add('Password is too common');
        strength = 0;
    }

    // Strength auf 0-4 begrenzen
    strength = strength.clamp(0, 4);

    return PasswordValidationResult(
        isValid: errors.isEmpty,
        errors: errors,
        strength: strength,
    );
}

bool isCommonPassword(String password) {
    return _commonPasswords.contains(password.toLowerCase());
}
}

```

#### 4.1.2.3 User Model

```

// lib/models/user.dart

class User {
    final int? id;
    final String email;
    final String passwordHash;
    final String? name;
    final String role;
    final bool isActive;
    final DateTime createdAt;
    final DateTime? updatedAt;

    User({
        this.id,
        required this.email,

```

```

    required this.passwordHash,
    this.name,
    this.role = 'user',
    this.isActive = true,
    DateTime? createdAt,
    this.updatedAt,
  }) : createdAt = createdAt ?? DateTime.now();

factory User.fromJson(Map<String, dynamic> json) {
  return User(
    id: json['id'] as int?,
    email: json['email'] as String,
    passwordHash: json['password_hash'] as String,
    name: json['name'] as String?,
    role: json['role'] as String? ?? 'user',
    isActive: json['is_active'] as bool? ?? true,
    createdAt: json['created_at'] is DateTime
      ? json['created_at'] as DateTime
      : DateTime.parse(json['created_at'] as String),
    updatedAt: json['updated_at'] != null
      ? (json['updated_at'] is DateTime
        ? json['updated_at'] as DateTime
        : DateTime.parse(json['updated_at'] as String))
      : null,
  );
}

Map<String, dynamic> toJson() => {
  if (id != null) 'id': id,
  'email': email,
  'password_hash': passwordHash,
  'name': name,
  'role': role,
  'is_active': isActive,
  'created_at': createdAt.toIso8601String(),
  if (updatedAt != null) 'updated_at': updatedAt!.toIso8601String(),
};

/// Für API-Responses (ohne sensible Daten)
Map<String, dynamic> toPublicJson() => {
  'id': id,
  'email': email,
  'name': name,
  'role': role,
  'created_at': createdAt.toIso8601String(),
};

User copyWith({
  int? id,
  String? email,
  String? passwordHash,

```

```
String? name,  
String? role,  
bool? isActive,  
DateTime? createdAt,  
DateTime? updatedAt,  
) {  
  return User(  
    id: id ?? this.id,  
    email: email ?? this.email,  
    passwordHash: passwordHash ?? this.passwordHash,  
    name: name ?? this.name,  
    role: role ?? this.role,  
    isActive: isActive ?? this.isActive,  
    createdAt: createdAt ?? this.createdAt,  
    updatedAt: updatedAt ?? this.updatedAt,  
  );  
}  
}
```

#### 4.1.2.4 User Repository

```
// lib/repositories/user_repository.dart  
import 'package:postgres/postgres.dart';  
import '../models/user.dart';  
  
abstract class UserRepository {  
  Future<User?> findByEmail(String email);  
  Future<User?> findById(int id);  
  Future<User> create(User user);  
  Future<bool> emailExists(String email);  
  Future<void> updatePassword(int userId, String newPasswordHash);  
  Future<void> deactivate(int userId);  
}  
  
class PostgresUserRepository implements UserRepository {  
  final Connection _db;  
  
  PostgresUserRepository(this._db);  
  
  @override  
  Future<User?> findByEmail(String email) async {  
    final result = await _db.execute(  
      Sql.named('SELECT * FROM users WHERE LOWER(email) = LOWER(@email)'),  
      parameters: {'email': email},  
    );  
  
    if (result.isEmpty) return null;  
    return User.fromJson(result.first.toColumnMap());  
  }  
}
```

```

@Override
Future<User?> findById(int id) async {
    final result = await _db.execute(
        Sql.named('SELECT * FROM users WHERE id = @id'),
        parameters: {'id': id},
    );

    if (result.isEmpty) return null;
    return User.fromJson(result.first.toColumnMap());
}

@Override
Future<User> create(User user) async {
    final result = await _db.execute(
        Sql.named('''
            INSERT INTO users (email, password_hash, name, role, is_active,
↪ created_at)
↪ VALUES (@email, @passwordHash, @name, @role, @isActive, @createdAt)
            RETURNING *
        '''),
        parameters: {
            'email': user.email,
            'passwordHash': user.passwordHash,
            'name': user.name,
            'role': user.role,
            'isActive': user.isActive,
            'createdAt': user.createdAt,
        },
    );

    return User.fromJson(result.first.toColumnMap());
}

@Override
Future<bool> emailExists(String email) async {
    final result = await _db.execute(
        Sql.named('SELECT 1 FROM users WHERE LOWER(email) = LOWER(@email)'),
        parameters: {'email': email},
    );
    return result.isNotEmpty;
}

@Override
Future<void> updatePassword(int userId, String newPasswordHash) async {
    await _db.execute(
        Sql.named('''
            UPDATE users
            SET password_hash = @passwordHash, updated_at = NOW()
            WHERE id = @id
        '''),
        parameters: {

```

```

        'id': userId,
        'passwordHash': newPasswordHash,
      },
    );
  }

  @override
  Future<void> deactivate(int userId) async {
    await _db.execute(
      Sql.named('''
        UPDATE users
        SET is_active = false, updated_at = NOW()
        WHERE id = @id
      '''),
      parameters: {'id': userId},
    );
  }
}

```

#### 4.1.2.5 Auth Service

```

// lib/services/auth_service.dart
import '../models/user.dart';
import '../repositories/user_repository.dart';
import 'password_service.dart';
import '../validators/password_validator.dart';

class AuthResult {
  final bool success;
  final User? user;
  final String? error;
  final int statusCode;

  AuthResult.success(this.user)
    : success = true,
      error = null,
      statusCode = 200;

  AuthResult.failure(this.error, {this.statusCode = 400})
    : success = false,
      user = null;
}

class AuthService {
  final UserRepository _userRepo;
  final PasswordService _passwordService;
  final PasswordValidator _passwordValidator;

  AuthService(this._userRepo, this._passwordService, this._passwordValidator);
}

```

```
/// Registriere einen neuen Benutzer
Future<AuthResult> register({
  required String email,
  required String password,
  String? name,
}) async {
  // 1. Email validieren
  if (!_isValidEmail(email)) {
    return AuthResult.failure('Invalid email format');
  }

  // 2. Email normalisieren
  final normalizedEmail = email.toLowerCase().trim();

  // 3. Prüfen ob Email existiert
  if (await _userRepo.emailExists(normalizedEmail)) {
    return AuthResult.failure('Email already registered', statusCode: 409);
  }

  // 4. Passwort validieren
  final passwordResult = _passwordValidator.validate(password);
  if (!passwordResult.isValid) {
    return AuthResult.failure(passwordResult.errors.join(', '));
  }

  // 5. Passwort hashen
  final passwordHash = _passwordService.hash(password);

  // 6. User erstellen
  final user = User(
    email: normalizedEmail,
    passwordHash: passwordHash,
    name: name?.trim(),
  );

  try {
    final createdUser = await _userRepo.create(user);
    return AuthResult.success(createdUser);
  } catch (e) {
    return AuthResult.failure('Registration failed', statusCode: 500);
  }
}

/// Authentifiziere einen Benutzer
Future<AuthResult> authenticate({
  required String email,
  required String password,
}) async {
  // 1. User laden
  final user = await _userRepo.findByEmail(email.toLowerCase().trim());
```

```
if (user == null) {
    // 2. Timing-Attack verhindern
    _passwordService.hash('dummy_password_to_prevent_timing_attack');
    return AuthResult.failure('Invalid credentials', statusCode: 401);
}

// 3. Account aktiv?
if (!user.isActive) {
    return AuthResult.failure('Account is deactivated', statusCode: 403);
}

// 4. Passwort verifizieren
if (!_passwordService.verify(password, user.passwordHash)) {
    return AuthResult.failure('Invalid credentials', statusCode: 401);
}

// 5. Optional: Rehash wenn nötig
if (_passwordService.needsRehash(user.passwordHash)) {
    final newHash = _passwordService.hash(password);
    await _userRepo.updatePassword(user.id!, newHash);
}

return AuthResult.success(user);
}

/// Passwort ändern
Future<AuthResult> changePassword({
    required int userId,
    required String currentPassword,
    required String newPassword,
}) async {
    // 1. User laden
    final user = await _userRepo.findById(userId);
    if (user == null) {
        return AuthResult.failure('User not found', statusCode: 404);
    }

    // 2. Aktuelles Passwort prüfen
    if (!_passwordService.verify(currentPassword, user.passwordHash)) {
        return AuthResult.failure('Current password is incorrect', statusCode: ↵
        ↵ 401);
    }

    // 3. Neues Passwort validieren
    final passwordResult = _passwordValidator.validate(newPassword);
    if (!passwordResult.isValid) {
        return AuthResult.failure(passwordResult.errors.join(', '));
    }

    // 4. Prüfen dass neues Passwort nicht gleich altem ist
    if (_passwordService.verify(newPassword, user.passwordHash)) {
```

```

        return AuthResult.failure('New password must be different from current
↪ password');
    }

    // 5. Neues Passwort hashen und speichern
    final newHash = _passwordService.hash(newPassword);
    await _userRepo.updatePassword(userId, newHash);

    return AuthResult.success(user);
}

bool _isValidEmail(String email) {
    return RegExp(r'^[\w-\.]@([\w-]+\w-)+[\w-]{2,4}$').hasMatch(email);
}
}

```

#### 4.1.2.6 Auth Handler

```

// lib/handlers/auth_handler.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';
import '../services/auth_service.dart';

class AuthHandler {
    final AuthService _authService;

    AuthHandler(this._authService);

    Router get router {
        final router = Router();

        router.post('/register', _register);
        router.post('/login', _login);

        return router;
    }

    Future<Response> _register(Request request) async {
        try {
            final body = await request.readAsString();
            final data = jsonDecode(body) as Map<String, dynamic>;

            final email = data['email'] as String?;
            final password = data['password'] as String?;
            final name = data['name'] as String?;

            if (email == null || password == null) {
                return _jsonResponse(
                    {'error': 'Email and password are required'},

```

```
        statusCode: 400,
    );
}

final result = await _authService.register(
    email: email,
    password: password,
    name: name,
);

if (result.success) {
    return _jsonResponse(
        {
            'message': 'User registered successfully',
            'user': result.user!.toPublicJson(),
        },
        statusCode: 201,
    );
} else {
    return _jsonResponse(
        {'error': result.error},
        statusCode: result.statusCode,
    );
}
} on FormatException {
    return _jsonResponse({'error': 'Invalid JSON'}, statusCode: 400);
} catch (e) {
    return _jsonResponse(
        {'error': 'Registration failed'},
        statusCode: 500,
    );
}
}

Future<Response> _login(Request request) async {
    try {
        final body = await request.readAsString();
        final data = jsonDecode(body) as Map<String, dynamic>;

        final email = data['email'] as String?;
        final password = data['password'] as String?;

        if (email == null || password == null) {
            return _jsonResponse(
                {'error': 'Email and password are required'},
                statusCode: 400,
            );
        }

        final result = await _authService.authenticate(
            email: email,
```

```

        password: password,
      );

      if (result.success) {
        return _jsonResponse({
          'message': 'Login successful',
          'user': result.user!.toJson(),
          // Token wird in nächster Einheit hinzugefügt
        });
      } else {
        return _jsonResponse(
          {'error': result.error},
          statusCode: result.statusCode,
        );
      }
    } on FormatException {
      return _jsonResponse({'error': 'Invalid JSON'}, statusCode: 400);
    } catch (e) {
      return _jsonResponse({'error': 'Login failed'}, statusCode: 500);
    }
  }

  Response _jsonResponse(Map<String, dynamic> data, {int statusCode = 200}) {
    return Response(
      statusCode,
      body: jsonEncode(data),
      headers: {'content-type': 'application/json'},
    );
  }
}

```

#### 4.1.2.7 Rate Limiter

```

// lib/services/rate_limiter.dart

class RateLimitResult {
  final bool allowed;
  final int remaining;
  final int resetInSeconds;

  RateLimitResult({
    required this.allowed,
    required this.remaining,
    required this.resetInSeconds,
  });
}

class LoginRateLimiter {
  final Map<String, List<DateTime>> _attempts = {};
  final int maxAttempts;
}

```

```
final Duration window;

LoginRateLimiter({
    this.maxAttempts = 5,
    this.window = const Duration(minutes: 15),
});

/// Prüfe ob Login erlaubt ist
RateLimitResult check(String identifier) {
    _cleanupOldAttempts(identifier);

    final attempts = _attempts[identifier] ?? [];
    final remaining = maxAttempts - attempts.length;

    if (attempts.length >= maxAttempts) {
        final oldestAttempt = attempts.first;
        final resetTime = oldestAttempt.add(window);
        final resetInSeconds = resetTime.difference(DateTime.now()).inSeconds;

        return RateLimitResult(
            allowed: false,
            remaining: 0,
            resetInSeconds: resetInSeconds > 0 ? resetInSeconds : 0,
        );
    }

    return RateLimitResult(
        allowed: true,
        remaining: remaining,
        resetInSeconds: window.inSeconds,
    );
}

/// Registriere einen fehlgeschlagenen Versuch
void recordFailedAttempt(String identifier) {
    _attempts.putIfAbsent(identifier, () => []);
    _attempts[identifier]!.add(DateTime.now());
}

/// Lösche alle Versuche nach erfolgreichem Login
void clearAttempts(String identifier) {
    _attempts.remove(identifier);
}

void _cleanupOldAttempts(String identifier) {
    final attempts = _attempts[identifier];
    if (attempts == null) return;

    final cutoff = DateTime.now().subtract(window);
    attempts.removeWhere((attempt) => attempt.isBefore(cutoff));
}
```

```
    if (attempts.isEmpty) {  
      _attempts.remove(identifier);  
    }  
  }  
}
```

#### 4.1.2.8 Main Server

```
// bin/server.dart  
import 'dart:io';  
import 'package:shelf/shelf.dart';  
import 'package:shelf/shelf_io.dart' as shelf_io;  
import 'package:shelf_router/shelf_router.dart';  
import 'package:postgres/postgres.dart';  
  
import '../lib/services/password_service.dart';  
import '../lib/validators/password_validator.dart';  
import '../lib/repositories/user_repository.dart';  
import '../lib/services/auth_service.dart';  
import '../lib/handlers/auth_handler.dart';  
  
Future<void> main() async {  
  // Database Connection  
  final db = await Connection.open(  
    Endpoint(  
      host: Platform.environment['DB_HOST'] ?? 'localhost',  
      database: Platform.environment['DB_NAME'] ?? 'auth_db',  
      username: Platform.environment['DB_USER'] ?? 'postgres',  
      password: Platform.environment['DB_PASSWORD'] ?? 'postgres',  
    ),  
    settings: ConnectionSettings(sslMode: SslMode.disable),  
  );  
  
  print('Connected to database');  
  
  // Services  
  final passwordService = PasswordService(costFactor: 12);  
  final passwordValidator = PasswordValidator(  
    minLength: 8,  
    requireUppercase: true,  
    requireLowercase: true,  
    requireDigit: true,  
    requireSpecialChar: true,  
  );  
  
  final userRepo = PostgresUserRepository(db);  
  final authService = AuthService(userRepo, passwordService, passwordValidator);  
  
  // Handler  
  final authHandler = AuthHandler(authService);
```

```

// Router
final router = Router();
router.mount('/api/auth', authHandler.router);

// Health Check
router.get('/health', (Request request) {
  return Response.ok('OK');
});

// Pipeline
final handler = const Pipeline()
  .addMiddleware(logRequests())
  .addMiddleware(_corsMiddleware())
  .addHandler(router);

// Server starten
final port = int.parse(Platform.environment['PORT'] ?? '8080');
final server = await shelf_io.serve(handler, 'localhost', port);
print('Server running on http://${server.address.host}:${server.port}');
}

Middleware _corsMiddleware() {
  return (Handler handler) {
    return (Request request) async {
      if (request.method == 'OPTIONS') {
        return Response.ok('', headers: _corsHeaders);
      }

      final response = await handler(request);
      return response.change(headers: _corsHeaders);
    };
  };
}

const _corsHeaders = {
  'Access-Control-Allow-Origin': '*',
  'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE, OPTIONS',
  'Access-Control-Allow-Headers': 'Origin, Content-Type, Authorization',
};

```

#### 4.1.2.9 Unit Tests

```

// test/password_service_test.dart
import 'package:test/test.dart';
import '../lib/services/password_service.dart';

void main() {
  group('PasswordService', () {
    late PasswordService service;

```

```

setUp(() {
  service = PasswordService(costFactor: 4); // Niedriger für schnelle Tests
});

test('hash() generates valid bcrypt hash', () {
  final hash = service.hash('password123');

  expect(hash, startsWith(r'$2'));
  expect(hash.length, greaterThan(50));
});

test('verify() returns true for correct password', () {
  final hash = service.hash('MySecretPassword!');

  expect(service.verify('MySecretPassword!', hash), isTrue);
});

test('verify() returns false for wrong password', () {
  final hash = service.hash('MySecretPassword!');

  expect(service.verify('WrongPassword', hash), isFalse);
});

test('same password generates different hashes', () {
  final hash1 = service.hash('password');
  final hash2 = service.hash('password');

  expect(hash1, isNot(equals(hash2)));
});

test('needsRehash() returns true for lower cost factor', () {
  final lowCostService = PasswordService(costFactor: 4);
  final hash = lowCostService.hash('password');

  final highCostService = PasswordService(costFactor: 10);
  expect(highCostService.needsRehash(hash), isTrue);
});
});
}

```

```

// test/password_validator_test.dart
import 'package:test/test.dart';
import '../lib/validators/password_validator.dart';

void main() {
  group('PasswordValidator', () {
    late PasswordValidator validator;

    setUp(() {
      validator = PasswordValidator();
    });
  });
}

```

```
test('valid password passes', () {
  final result = validator.validate('SecurePass123!');

  expect(result.isValid, isTrue);
  expect(result.errors, isEmpty);
  expect(result.strength, greaterThanOrEqualTo(3));
});

test('short password fails', () {
  final result = validator.validate('Abc1!');

  expect(result.isValid, isFalse);
  expect(result.errors, contains(contains('8 characters')));
});

test('password without uppercase fails', () {
  final result = validator.validate('lowercase123!');

  expect(result.isValid, isFalse);
  expect(result.errors, contains(contains('uppercase')));
});

test('common password fails', () {
  final result = validator.validate('password');

  expect(result.isValid, isFalse);
  expect(result.errors, contains(contains('common')));
});

test('strength increases with complexity', () {
  final weak = validator.validate('abcdefgh');
  final strong = validator.validate('SecurePass123!@#');

  expect(strong.strength, greaterThan(weak.strength));
});
});
}
```

### 4.1.3 Ressourcen

#### 4.1.3.1 Offizielle Dokumentation

- bcrypt Package (pub.dev)
- OWASP Password Storage Cheat Sheet
- NIST Password Guidelines

#### 4.1.3.2 Cheat Sheet: bcrypt

```
import 'package:bcrypt/bcrypt.dart';
```

```
// Salt generieren
final salt = BCrypt.gensalt(logRounds: 12);
// Ergebnis: "$2a$12$LQv3c1yqBWVHxkd0LHAKC0"

// Passwort hashen
final hash = BCrypt.hashpw('password', salt);
// Ergebnis: "$2a$12$LQv3c1yqBWVHxkd0LHAKC0Yz6TtxMQJqhN8/X4.N1tJ9vPmUHPK3a"

// Passwort verifizieren
final isValid = BCrypt.checkpw('password', hash);
// Ergebnis: true
```

#### 4.1.3.3 Cheat Sheet: Hash-Format

```
$2a$12$LQv3c1yqBWVHxkd0LHAKC0Yz6TtxMQJqhN8/X4.N1tJ9vPmUHPK3a
| | |                                     |
| | |                                     +-- Hash (31 Zeichen, Base64)
| | +-- Salt (22 Zeichen, Base64)
| +-- Cost Factor (Exponentiell: 2^12 = 4096 Iterationen)
+-- Algorithmus ($2a = bcrypt)
```

#### 4.1.3.4 Cheat Sheet: Cost Factor

Cost	Iterationen	Zeit (ca.)	Empfehlung
10	1.024	~100ms	Minimum
11	2.048	~200ms	Okay
12	4.096	~300ms	<b>Empfohlen</b>
13	8.192	~600ms	Gut
14	16.384	~1s	Hoch

**Faustregel:** 100-300ms ist ein guter Kompromiss.

#### 4.1.3.5 Cheat Sheet: Password Validation

```
class PasswordRules {
  // Mindestlänge
  static bool hasMinLength(String p, int min) => p.length >= min;

  // Großbuchstaben
  static bool hasUppercase(String p) => p.contains(RegExp(r'[A-Z]'));

  // Kleinbuchstaben
  static bool hasLowercase(String p) => p.contains(RegExp(r'[a-z]'));

  // Ziffern
  static bool hasDigit(String p) => p.contains(RegExp(r'[0-9]'));

  // Sonderzeichen
  static bool hasSpecial(String p) =>
```

```

        p.contains(RegExp(r'[!@#$$%^&*(),.?:{}|<>]')));

// Keine Sequenzen (123, abc)
static bool noSequences(String p) =>
    !p.contains(RegExp(r'(012|123|234|345|456|567|678|789|890)')) &&
    !p.contains(RegExp(r'(abc|bcd|cde|def|efg|fgh|ghi)', caseSensitive:
↪ false));

// Keine Wiederholungen (aaa, 111)
static bool noRepeats(String p) => !p.contains(RegExp(r'(\.){2,}'));
}

```

#### 4.1.3.6 Cheat Sheet: Timing Attack Prevention

```

// SCHLECHT - Timing Attack möglich
Future<User?> authenticate(String email, String password) async {
    final user = await userRepo.findByEmail(email);
    if (user == null) {
        return null; // Schnelle Antwort
    }
    if (!bcrypt.verify(password, user.hash)) {
        return null; // Langsame Antwort
    }
    return user;
}

// GUT - Konstante Zeit
Future<User?> authenticate(String email, String password) async {
    final user = await userRepo.findByEmail(email);

    // Immer hashen, auch wenn User nicht existiert
    final hashToCheck = user?.hash ?? _dummyHash;
    final isValid = bcrypt.verify(password, hashToCheck);

    if (user == null || !isValid) {
        return null;
    }
    return user;
}

```

#### 4.1.3.7 Cheat Sheet: Email Validation

```

// Einfache Regex
final emailRegex = RegExp(r'^[\w-\.]@([\w-]+\.)+[\w-]{2,4}$');

// Umfassendere Regex (RFC 5322)
final emailRegexFull = RegExp(
    r'^[a-zA-Z0-9.!#$%&*+/?^_`{|}~-]+@[a-zA-Z0-9]'
    r'(?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?'

```

```

    r'(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$'
);

// Normalisierung
String normalizeEmail(String email) {
    return email.toLowerCase().trim();
}

```

#### 4.1.3.8 Cheat Sheet: Common Passwords

```

const commonPasswords = [
    // Top 20 weltweit
    '123456', 'password', '12345678', 'qwerty', '123456789',
    '12345', '1234', '111111', '1234567', 'dragon',
    '123123', 'baseball', 'iloveyou', 'trustno1', 'sunshine',
    'princess', 'welcome', 'shadow', 'superman', 'michael',

    // Deutsche Klassiker
    'hallo', 'schatz', 'passwort', 'sommer', 'berlin',
];

bool isCommon(String password) {
    return commonPasswords.contains(password.toLowerCase());
}

```

#### 4.1.3.9 Best Practices

##### DO

1. **bcrypt oder Argon2 verwenden** - Nie MD5/SHA für Passwörter
2. **Cost Factor mindestens 12** - Anpassen an Hardware
3. **Generische Fehlermeldungen** - “Invalid credentials”
4. **Email normalisieren** - Lowercase, trim
5. **Rate Limiting** - Brute-Force verhindern
6. **Passwort-Policies** - Mindestlänge, Komplexität
7. **Timing Attack verhindern** - Konstante Antwortzeit

##### DON'T

1. **Passwörter im Klartext speichern** - Nie!
2. **MD5/SHA1 für Passwörter** - Zu schnell
3. **Eigenen Hash-Algorithmus** - Nutze bewährte Bibliotheken
4. **Salt wiederverwenden** - Immer neu generieren
5. **Spezifische Fehlermeldungen** - “Email nicht gefunden”
6. **Maximum-Länge für Passwörter** - Keine künstlichen Limits

#### 4.1.3.10 Algorithmen-Vergleich

Algorithmus	Geschwindigkeit	Memory	Empfehlung
<b>bcrypt</b>	Langsam	Niedrig	Standard
<b>Argon2id</b>	Langsam	Hoch	Beste Wahl

Algorithmus	Geschwindigkeit	Memory	Empfehlung
<b>script</b>	Langsam	Hoch	Alternative
PBKDF2	Langsam	Niedrig	Veraltet
SHA-256	Schnell	Niedrig	Nicht für Passwörter
MD5	Sehr schnell	Niedrig	Nie verwenden

#### 4.1.3.11 SQL Schema

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) NOT NULL UNIQUE,
  password_hash VARCHAR(255) NOT NULL,
  name VARCHAR(255),
  role VARCHAR(50) NOT NULL DEFAULT 'user',
  is_active BOOLEAN NOT NULL DEFAULT true,
  failed_login_attempts INT NOT NULL DEFAULT 0,
  locked_until TIMESTAMP,
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMP
);

-- Index für schnelle Email-Suche
CREATE INDEX idx_users_email ON users(email);

-- Partial Index für aktive User
CREATE INDEX idx_users_active ON users(is_active) WHERE is_active = true;
```

## 4.2 Einheit 8.2: JWT Authentication

### 4.2.0.1 Lernziele

Nach dieser Einheit kannst du: - JWT-Struktur und -Funktionsweise verstehen - Access und Refresh Tokens implementieren - Tokens generieren und validieren - Token-basierte Authentifizierung aufbauen

### 4.2.0.2 Was ist JWT?

**JWT** (JSON Web Token) ist ein offener Standard (RFC 7519) für die sichere Übertragung von Informationen zwischen Parteien als JSON-Objekt.

Eigenschaften

- **Kompakt:** URL-sicher, kann in HTTP-Header übertragen werden
- **Selbstbeschreibend:** Enthält alle nötigen Informationen
- **Signiert:** Integrität und Authentizität gewährleistet
- **Optional verschlüsselt:** JWE für sensitive Daten

Wann JWT?

Use Case	JWT geeignet?
Stateless API Auth	Ideal

Use Case	JWT geeignet?
Single Sign-On (SSO)	Ideal
Microservices	Ideal
Session-basierte Apps	Sessions oft einfacher
Sensitive Daten im Token	Nicht empfohlen

### 4.2.0.3 JWT-Struktur

Ein JWT besteht aus drei Teilen, getrennt durch Punkte:

xxxxx.yyyyy.zzzzz

Header.Payload.Signature

#### 1. Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- **alg**: Signatur-Algorithmus (HS256, RS256, ES256)
- **typ**: Token-Typ (immer "JWT")

#### 2. Payload (Claims)

```
{
  "sub": "1234567890",
  "name": "Max Mustermann",
  "email": "max@example.com",
  "role": "admin",
  "iat": 1516239022,
  "exp": 1516242622
}
```

#### Registered Claims (Standard):

Claim	Beschreibung
sub	Subject (User-ID)
iss	Issuer (Aussteller)
aud	Audience (Empfänger)
exp	Expiration Time
iat	Issued At
nbf	Not Before
jti	JWT ID (einmalig)

**Private Claims (Custom):** - role, email, permissions, etc.

#### 3. Signature

```
HMACSHA256(
  base64UrlEncode(header) + "." + base64UrlEncode(payload),
  secret
)
```

#### 4.2.0.4 JWT in Dart

##### Installation

```
dependencies:  
  dart_jsonwebtoken: ^2.12.0
```

##### Token generieren

```
import 'package:dart_jsonwebtoken/dart_jsonwebtoken.dart';  
  
class JwtService {  
  final String _secret;  
  final Duration _accessTokenDuration;  
  final Duration _refreshTokenDuration;  
  
  JwtService({  
    required String secret,  
    Duration? accessTokenDuration,  
    Duration? refreshTokenDuration,  
  }) : _secret = secret,  
       _accessTokenDuration = accessTokenDuration ?? const Duration(minutes: 15),  
       _refreshTokenDuration = refreshTokenDuration ?? const Duration(days: 7);  
  
  /// Access Token generieren  
  String generateAccessToken(User user) {  
    final jwt = JWT(  
      {  
        'sub': user.id.toString(),  
        'email': user.email,  
        'name': user.name,  
        'role': user.role,  
        'type': 'access',  
      },  
      issuer: 'my-api',  
      subject: user.id.toString(),  
    );  
  
    return jwt.sign(  
      SecretKey(_secret),  
      expiresIn: _accessTokenDuration,  
    );  
  }  
  
  /// Refresh Token generieren  
  String generateRefreshToken(User user) {  
    final jwt = JWT(  
      {  
        'sub': user.id.toString(),  
        'type': 'refresh',  
      },  
      issuer: 'my-api',
```

```

        subject: user.id.toString(),
    );

    return jwt.sign(
        SecretKey(_secret),
        expiresIn: _refreshTokenDuration,
    );
}

/// Token-Paar generieren
TokenPair generateTokenPair(User user) {
    return TokenPair(
        accessToken: generateAccessToken(user),
        refreshToken: generateRefreshToken(user),
        expiresIn: _accessTokenDuration.inSeconds,
    );
}

class TokenPair {
    final String accessToken;
    final String refreshToken;
    final int expiresIn;

    TokenPair({
        required this.accessToken,
        required this.refreshToken,
        required this.expiresIn,
    });

    Map<String, dynamic> toJson() => {
        'access_token': accessToken,
        'refresh_token': refreshToken,
        'expires_in': expiresIn,
        'token_type': 'Bearer',
    };
}

```

Token validieren

```

class JwtService {
    // ... vorheriger Code ...

    /// Token verifizieren und Payload extrahieren
    JwtPayload? verifyToken(String token) {
        try {
            final jwt = JWT.verify(token, SecretKey(_secret));

            return JwtPayload(
                sub: jwt.payload['sub'] as String,
                email: jwt.payload['email'] as String?,
            );
        } catch (e) {
            return null;
        }
    }
}

```

```
        name: jwt.payload['name'] as String?,
        role: jwt.payload['role'] as String?,
        type: jwt.payload['type'] as String,
        exp: jwt.payload['exp'] as int?,
        iat: jwt.payload['iat'] as int?,
    );
} on JWTExpiredException {
    throw TokenExpiredException();
} on JWTException catch (e) {
    throw InvalidTokenException(e.message);
}
}

/// Prüfen ob Token abgelaufen ist
bool isExpired(String token) {
    try {
        JWT.verify(token, SecretKey(_secret));
        return false;
    } on JWTExpiredException {
        return true;
    } catch (e) {
        return true;
    }
}

/// Payload ohne Verifizierung lesen (unsicher!)
Map<String, dynamic>? decodePayload(String token) {
    try {
        final jwt = JWT.decode(token);
        return jwt.payload as Map<String, dynamic>;
    } catch (e) {
        return null;
    }
}

class JwtPayload {
    final String sub;
    final String? email;
    final String? name;
    final String? role;
    final String type;
    final int? exp;
    final int? iat;

    JwtPayload({
        required this.sub,
        this.email,
        this.name,
        this.role,
        required this.type,
```

```
    this.exp,  
    this.iat,  
  });  
  
  int get userId => int.parse(sub);  
  
  bool get isAccessToken => type == 'access';  
  bool get isRefreshToken => type == 'refresh';  
}
```

#### Exceptions

```
// lib/exceptions/auth_exceptions.dart  
  
class TokenExpiredException implements Exception {  
  final String message = 'Token has expired';  
}  
  
class InvalidTokenException implements Exception {  
  final String message;  
  InvalidTokenException([this.message = 'Invalid token']);  
}  
  
class RefreshTokenRevokedException implements Exception {  
  final String message = 'Refresh token has been revoked';  
}
```

#### 4.2.0.5 Access vs Refresh Tokens

##### Access Token

- **Kurze Lebensdauer:** 15-60 Minuten
- **Verwendung:** In jedem API-Request
- **Enthält:** User-Daten für Autorisierung
- **Speicherung:** Im Memory (Frontend)

##### Refresh Token

- **Lange Lebensdauer:** Tage/Wochen
- **Verwendung:** Nur zum Erneuern des Access Tokens
- **Enthält:** Nur User-ID
- **Speicherung:** HttpOnly Cookie oder Secure Storage

##### Flow

1. Login -> Access Token + Refresh Token
2. API Request mit Access Token
3. Access Token abgelaufen -> 401
4. Refresh Request mit Refresh Token -> Neuer Access Token
5. Logout -> Refresh Token invalidieren

#### 4.2.0.6 Token Refresh

```
// lib/services/auth_service.dart

class AuthService {
  final UserRepository _userRepo;
  final JwtService _jwtService;
  final RefreshTokenRepository _refreshTokenRepo;

  // ... constructor ...

  /// Login mit Token-Generierung
  Future<AuthResult> login(LoginRequest request) async {
    final user = await _authenticate(request);
    if (user == null) {
      return AuthResult.failure('Invalid credentials', statusCode: 401);
    }

    final tokenPair = _jwtService.generateTokenPair(user);

    // Refresh Token in DB speichern
    await _refreshTokenRepo.create(RefreshTokenRecord(
      userId: user.id!,
      token: tokenPair.refreshToken,
      expiresAt: DateTime.now().add(const Duration(days: 7)),
    ));

    return AuthResult.success(user, tokens: tokenPair);
  }

  /// Access Token erneuern
  Future<TokenPair> refreshAccessToken(String refreshToken) async {
    // 1. Token verifizieren
    final payload = _jwtService.verifyToken(refreshToken);
    if (payload == null || !payload.isRefreshToken) {
      throw InvalidTokenException('Invalid refresh token');
    }

    // 2. Prüfen ob Token in DB existiert (nicht revoked)
    final record = await _refreshTokenRepo.findByToken(refreshToken);
    if (record == null || record.isRevoked) {
      throw RefreshTokenRevokedException();
    }

    // 3. User laden
    final user = await _userRepo.findById(payload.userId);
    if (user == null || !user.isActive) {
      throw InvalidTokenException('User not found or inactive');
    }

    // 4. Neues Token-Paar generieren (Token Rotation)
```

```
final newTokenPair = _jwtService.generateTokenPair(user);

// 5. Alten Refresh Token revoke
await _refreshTokenRepo.revoke(refreshToken);

// 6. Neuen Refresh Token speichern
await _refreshTokenRepo.create(RefreshTokenRecord(
  userId: user.id!,
  token: newTokenPair.refreshToken,
  expiresAt: DateTime.now().add(const Duration(days: 7)),
));

return newTokenPair;
}

/// Logout - Refresh Token invalidieren
Future<void> logout(String refreshToken) async {
  await _refreshTokenRepo.revoke(refreshToken);
}

/// Alle Sessions eines Users beenden
Future<void> logoutAllSessions(int userId) async {
  await _refreshTokenRepo.revokeAllForUser(userId);
}
}
```

#### 4.2.0.7 Refresh Token Repository

```
// lib/repositories/refresh_token_repository.dart

class RefreshTokenRecord {
  final int? id;
  final int userId;
  final String token;
  final DateTime createdAt;
  final DateTime expiresAt;
  final bool isRevoked;

  RefreshTokenRecord({
    this.id,
    required this.userId,
    required this.token,
    DateTime? createdAt,
    required this.expiresAt,
    this.isRevoked = false,
  }) : createdAt = createdAt ?? DateTime.now();
}

class RefreshTokenRepository {
  final Connection _db;
```

```
RefreshTokenRepository(this._db);

Future<void> create(RefreshTokenRecord record) async {
  await _db.execute(
    Sql.named('''
      INSERT INTO refresh_tokens (user_id, token, expires_at)
      VALUES (@userId, @token, @expiresAt)
    '''),
    parameters: {
      'userId': record.userId,
      'token': record.token,
      'expiresAt': record.expiresAt,
    },
  );
}

Future<RefreshTokenRecord?> findByToken(String token) async {
  final result = await _db.execute(
    Sql.named('''
      SELECT * FROM refresh_tokens
      WHERE token = @token AND is_revoked = false
    '''),
    parameters: {'token': token},
  );

  if (result.isEmpty) return null;
  return _mapToRecord(result.first.toColumnMap());
}

Future<void> revoke(String token) async {
  await _db.execute(
    Sql.named('''
      UPDATE refresh_tokens
      SET is_revoked = true
      WHERE token = @token
    '''),
    parameters: {'token': token},
  );
}

Future<void> revokeAllForUser(int userId) async {
  await _db.execute(
    Sql.named('''
      UPDATE refresh_tokens
      SET is_revoked = true
      WHERE user_id = @userId
    '''),
    parameters: {'userId': userId},
  );
}
```

```

Future<void> deleteExpired() async {
  await _db.execute(
    Sql.named('''
      DELETE FROM refresh_tokens
      WHERE expires_at < NOW() OR is_revoked = true
    '''),
  );
}

RefreshTokenRecord _mapToRecord(Map<String, dynamic> row) {
  return RefreshTokenRecord(
    id: row['id'] as int,
    userId: row['user_id'] as int,
    token: row['token'] as String,
    createdAt: row['created_at'] as DateTime,
    expiresAt: row['expires_at'] as DateTime,
    isRevoked: row['is_revoked'] as bool,
  );
}

```

#### 4.2.0.8 Auth Handler (erweitert)

```

// lib/handlers/auth_handler.dart

class AuthHandler {
  final AuthService _authService;

  AuthHandler(this._authService);

  Router get router {
    final router = Router();

    router.post('/register', _register);
    router.post('/login', _login);
    router.post('/refresh', _refresh);
    router.post('/logout', _logout);

    return router;
  }

  Future<Response> _login(Request request) async {
    try {
      final body = await request.readAsString();
      final data = jsonDecode(body) as Map<String, dynamic>;

      final result = await _authService.login(LoginRequest(
        email: data['email'] as String,
        password: data['password'] as String,

```

```
));

if (result.success) {
    return _jsonResponse({
        'message': 'Login successful',
        'user': result.user!.toPublicJson(),
        ...result.tokens!.toJson(),
    });
}

return _jsonResponse(
    {'error': result.error},
    statusCode: result.statusCode,
);
} catch (e) {
    return _jsonResponse({'error': 'Login failed'}, statusCode: 500);
}
}

Future<Response> _refresh(Request request) async {
    try {
        final body = await request.readAsString();
        final data = jsonDecode(body) as Map<String, dynamic>;
        final refreshToken = data['refresh_token'] as String?;

        if (refreshToken == null) {
            return _jsonResponse(
                {'error': 'Refresh token required'},
                statusCode: 400,
            );
        }

        final newTokens = await _authService.refreshAccessToken(refreshToken);

        return _jsonResponse(newTokens.toJson());
    } on TokenExpiredException {
        return _jsonResponse(
            {'error': 'Refresh token expired'},
            statusCode: 401,
        );
    } on RefreshTokenRevokedException {
        return _jsonResponse(
            {'error': 'Refresh token revoked'},
            statusCode: 401,
        );
    } catch (e) {
        return _jsonResponse({'error': 'Token refresh failed'}, statusCode: 500);
    }
}

Future<Response> _logout(Request request) async {
```

```

    try {
        final body = await request.readAsString();
        final data = jsonDecode(body) as Map<String, dynamic>;
        final refreshToken = data['refresh_token'] as String?;

        if (refreshToken != null) {
            await _authService.logout(refreshToken);
        }

        return _jsonResponse({'message': 'Logged out successfully'});
    } catch (e) {
        return _jsonResponse({'message': 'Logged out'});
    }
}

// ... _register, _jsonResponse ...
}

```

#### 4.2.0.9 Datenbank-Schema

```

-- migrations/002_create_refresh_tokens.sql

CREATE TABLE refresh_tokens (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    token TEXT NOT NULL UNIQUE,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    expires_at TIMESTAMP NOT NULL,
    is_revoked BOOLEAN NOT NULL DEFAULT false
);

-- Index für Token-Lookup
CREATE INDEX idx_refresh_tokens_token ON refresh_tokens(token);

-- Index für User-Cleanup
CREATE INDEX idx_refresh_tokens_user ON refresh_tokens(user_id);

-- Index für Expired-Cleanup
CREATE INDEX idx_refresh_tokens_expires ON refresh_tokens(expires_at);

```

#### 4.2.0.10 Best Practices

##### Token-Sicherheit

1. **Kurze Access Token Lifetime** - 15-60 Minuten
2. **Refresh Token Rotation** - Bei jedem Refresh neues Token
3. **Sichere Secrets** - Mindestens 256 Bit, aus sicherer Quelle
4. **HTTPS** - Tokens nur über verschlüsselte Verbindung
5. **Nicht in localStorage** - XSS-Gefahr

##### Token-Invalidierung

```
// Alle Tokens eines Users invalidieren (z.B. bei Passwortänderung)
Future<void> invalidateAllTokens(int userId) async {
  await _refreshTokenRepo.revokeAllForUser(userId);
}

// Token-Blacklist für Access Tokens (optional)
class TokenBlacklist {
  final RedisClient _redis;

  Future<void> blacklist(String token, Duration ttl) async {
    await _redis.set('blacklist:$token', '1', ttl: ttl);
  }

  Future<bool> isBlacklisted(String token) async {
    return await _redis.exists('blacklist:$token');
  }
}
```

#### Secret Management

```
// NIEMALS hardcoded!
// FALSCH:
final secret = 'my-super-secret-key';

// RICHTIG:
final secret = Platform.environment['JWT_SECRET']
  ?? (throw Exception('JWT_SECRET not set'));

// Secret-Generierung (einmalig):
// dart -e "import 'dart:math'; print(List.generate(32, (_) =>
  ↪ Random.secure().nextInt(256).toRadixString(16).padLeft(2, '0')).join());"
```

#### 4.2.0.11 Zusammenfassung

Konzept	Beschreibung
<b>JWT</b>	Signiertes JSON-Token
<b>Header</b>	Algorithmus und Typ
<b>Payload</b>	Claims (User-Daten)
<b>Signature</b>	Integritätsprüfung
<b>Access Token</b>	Kurzlebig, für API-Requests
<b>Refresh Token</b>	Langlebig, für Token-Erneuerung
<b>Token Rotation</b>	Neues Refresh Token bei jedem Refresh

### 4.2.1 Übung

#### 4.2.1.1 Ziel

Erweitere das Auth-System aus Einheit 8.1 um JWT-basierte Authentifizierung mit Access und Refresh Tokens.

#### 4.2.1.2 Vorbereitung

Dependencies hinzufügen

```
dependencies:
  dart_jsonwebtoken: ^2.12.0
  # ... bestehende Dependencies
```

Datenbank erweitern

```
CREATE TABLE refresh_tokens (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  token TEXT NOT NULL UNIQUE,
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  expires_at TIMESTAMP NOT NULL,
  is_revoked BOOLEAN NOT NULL DEFAULT false
);

CREATE INDEX idx_refresh_tokens_token ON refresh_tokens(token);
CREATE INDEX idx_refresh_tokens_user ON refresh_tokens(user_id);
```

#### 4.2.1.3 Aufgabe 1: JWT Payload Model (10 min)

```
// lib/models/jwt_payload.dart

class JwtPayload {
  final String sub;           // User ID
  final String? email;
  final String? name;
  final String? role;
  final String type;         // 'access' oder 'refresh'
  final DateTime? expiresAt;
  final DateTime? issuedAt;

  JwtPayload({
    required this.sub,
    this.email,
    this.name,
    this.role,
    required this.type,
    this.expiresAt,
    this.issuedAt,
  });

  // TODO: Getter für userId (int parse)
  // TODO: Getter isAccessToken / isRefreshToken
  // TODO: Getter isExpired

  factory JwtPayload.fromJwtPayload(Map<String, dynamic> payload) {
    // TODO: Payload aus JWT extrahieren
    // Hinweis: exp und iat sind Unix-Timestamps (Sekunden)
```

```
}  
}
```

#### 4.2.1.4 Aufgabe 2: Token Pair Model (5 min)

```
// lib/models/token_pair.dart  
  
class TokenPair {  
  final String accessToken;  
  final String refreshToken;  
  final int expiresIn; // Sekunden bis Access Token abläuft  
  
  TokenPair({  
    required this.accessToken,  
    required this.refreshToken,  
    required this.expiresIn,  
  });  
  
  Map<String, dynamic> toJson() {  
    // TODO: JSON-Response für Client  
    // {  
    //   "access_token": "...",  
    //   "refresh_token": "...",  
    //   "expires_in": 900,  
    //   "token_type": "Bearer"  
    // }  
  }  
}
```

#### 4.2.1.5 Aufgabe 3: JWT Service (25 min)

```
// lib/services/jwt_service.dart  
import 'package:dart_jsonwebtoken/dart_jsonwebtoken.dart';  
  
class JwtService {  
  final String _secret;  
  final String _issuer;  
  final Duration _accessTokenDuration;  
  final Duration _refreshTokenDuration;  
  
  JwtService({  
    required String secret,  
    String issuer = 'my-api',  
    Duration? accessTokenDuration,  
    Duration? refreshTokenDuration,  
  }) : _secret = secret,  
      _issuer = issuer,  
      _accessTokenDuration = accessTokenDuration ?? const Duration(minutes: ↵  
↵ 15),
```

```
    _refreshTokenDuration = refreshTokenDuration ?? const Duration(days: 7);

/// Generiere Access Token für User
String generateAccessToken(User user) {
  // TODO:
  // 1. JWT mit Payload erstellen (sub, email, name, role, type: 'access')
  // 2. Mit SecretKey signieren
  // 3. expiresIn setzen
}

/// Generiere Refresh Token für User
String generateRefreshToken(User user) {
  // TODO:
  // 1. JWT mit minimalem Payload (sub, type: 'refresh')
  // 2. Längere Lebensdauer
}

/// Generiere Token-Paar
TokenPair generateTokenPair(User user) {
  // TODO: Beide Tokens generieren und als TokenPair zurückgeben
}

/// Verifiziere und dekodiere Token
JwtPayload? verifyToken(String token) {
  // TODO:
  // 1. JWT.verify mit SecretKey
  // 2. JwtPayload aus payload erstellen
  // 3. Bei Fehler: entsprechende Exception werfen
}

/// Dekodiere Token ohne Verifizierung (nur für Debugging!)
Map<String, dynamic>? decodeWithoutVerification(String token) {
  // TODO: JWT.decode (unsicher, nur für Debugging)
}
}
```

#### 4.2.1.6 Aufgabe 4: Refresh Token Repository (15 min)

```
// lib/repositories/refresh_token_repository.dart

class RefreshTokenRecord {
  final int? id;
  final int userId;
  final String token;
  final DateTime createdAt;
  final DateTime expiresAt;
  final bool isRevoked;

  RefreshTokenRecord({
    this.id,
```

```

        required this.userId,
        required this.token,
        DateTime? createdAt,
        required this.expiresAt,
        this.isRevoked = false,
    }) : createdAt = createdAt ?? DateTime.now();
}

class RefreshTokenRepository {
    final Connection _db;

    RefreshTokenRepository(this._db);

    /// Neuen Refresh Token speichern
    Future<void> create(RefreshTokenRecord record) async {
        // TODO: INSERT INTO refresh_tokens
    }

    /// Token suchen (nur wenn nicht revoked)
    Future<RefreshTokenRecord?> findByToken(String token) async {
        // TODO: SELECT WHERE token = @token AND is_revoked = false
    }

    /// Token als revoked markieren
    Future<void> revoke(String token) async {
        // TODO: UPDATE SET is_revoked = true
    }

    /// Alle Tokens eines Users revoke
    Future<void> revokeAllForUser(int userId) async {
        // TODO: UPDATE WHERE user_id = @userId
    }

    /// Abgelaufene Tokens löschen (Cleanup-Job)
    Future<int> deleteExpired() async {
        // TODO: DELETE WHERE expires_at < NOW()
        // Return: Anzahl gelöschter Einträge
    }
}

```

#### 4.2.1.7 Aufgabe 5: Auth Service erweitern (20 min)

Erweitere deinen AuthService aus Einheit 8.1:

```

// lib/services/auth_service.dart

class AuthService {
    final UserRepository _userRepo;
    final PasswordService _passwordService;
    final JwtService _jwtService;
    final RefreshTokenRepository _refreshTokenRepo;
}

```

```
// ... Constructor ...

/// Login mit Token-Generierung
Future<AuthResult> login(LoginRequest request) async {
    // TODO:
    // 1. User authentifizieren (wie in 8.1)
    // 2. Token-Paar generieren
    // 3. Refresh Token in DB speichern
    // 4. AuthResult mit Tokens zurückgeben
}

/// Access Token erneuern
Future<TokenPair> refreshAccessToken(String refreshToken) async {
    // TODO:
    // 1. Refresh Token verifizieren
    // 2. Prüfen ob type == 'refresh'
    // 3. Prüfen ob in DB existiert und nicht revoked
    // 4. User laden
    // 5. Alten Token revoke (Token Rotation!)
    // 6. Neues Token-Paar generieren
    // 7. Neuen Refresh Token speichern
    // 8. TokenPair zurückgeben
}

/// Logout - Refresh Token invalidieren
Future<void> logout(String refreshToken) async {
    // TODO: Token revoke
}

/// Alle Sessions beenden
Future<void> logoutAllSessions(int userId) async {
    // TODO: Alle Tokens des Users revoke
}
}

// AuthResult erweitern
class AuthResult {
    final bool success;
    final User? user;
    final TokenPair? tokens; // NEU
    final String? error;
    final int statusCode;

    // ... Constructors anpassen ...
}
```

#### 4.2.1.8 Aufgabe 6: Auth Handler erweitern (15 min)

```
// lib/handlers/auth_handler.dart

class AuthHandler {
  Router get router {
    final router = Router();

    router.post('/register', _register);
    router.post('/login', _login);
    router.post('/refresh', _refresh); // NEU
    router.post('/logout', _logout);  // NEU

    return router;
  }

  Future<Response> _login(Request request) async {
    // TODO: Login Response mit Tokens erweitern
    // {
    //   "message": "Login successful",
    //   "user": { ... },
    //   "access_token": "...",
    //   "refresh_token": "...",
    //   "expires_in": 900,
    //   "token_type": "Bearer"
    // }
  }

  Future<Response> _refresh(Request request) async {
    // TODO:
    // 1. refresh_token aus Body lesen
    // 2. authService.refreshAccessToken aufrufen
    // 3. Neues Token-Paar zurückgeben
    // 4. Fehlerbehandlung (expired, revoked, invalid)
  }

  Future<Response> _logout(Request request) async {
    // TODO:
    // 1. refresh_token aus Body lesen
    // 2. authService.logout aufrufen
    // 3. Erfolg zurückgeben
  }
}
```

#### 4.2.1.9 Aufgabe 7: Exceptions (5 min)

```
// lib/exceptions/auth_exceptions.dart

class TokenExpiredException implements Exception {
  final String message = 'Token has expired';
}
```

```
class InvalidTokenException implements Exception {
  final String message;
  InvalidTokenException([this.message = 'Invalid token']);
}

class RefreshTokenRevokedException implements Exception {
  final String message = 'Refresh token has been revoked';
}

class TokenTypeMismatchException implements Exception {
  final String expected;
  final String actual;

  TokenTypeMismatchException(this.expected, this.actual);

  String get message => 'Expected $expected token, got $actual';
}
```

#### 4.2.1.10 Aufgabe 8: Konfiguration (5 min)

```
// lib/config/jwt_config.dart

class JwtConfig {
  final String secret;
  final String issuer;
  final Duration accessTokenDuration;
  final Duration refreshTokenDuration;

  JwtConfig({
    required this.secret,
    this.issuer = 'my-api',
    this.accessTokenDuration = const Duration(minutes: 15),
    this.refreshTokenDuration = const Duration(days: 7),
  });

  factory JwtConfig.fromEnvironment() {
    final secret = Platform.environment['JWT_SECRET'];
    if (secret == null || secret.isEmpty) {
      throw Exception('JWT_SECRET environment variable not set');
    }

    return JwtConfig(
      secret: secret,
      issuer: Platform.environment['JWT_ISSUER'] ?? 'my-api',
      accessTokenDuration: Duration(
        minutes: int.parse(Platform.environment['JWT_ACCESS_DURATION'] ?? '15'),
      ),
      refreshTokenDuration: Duration(
        days: int.parse(Platform.environment['JWT_REFRESH_DURATION'] ?? '7'),
      ),
    ),
  }
```

```

    );
  }
}

```

#### 4.2.1.11 Testen

Login mit Token-Response

```

curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email": "test@example.com", "password": "SecurePass123!"}'

# Response:
# {
#   "message": "Login successful",
#   "user": { "id": 1, "email": "test@example.com", ... },
#   "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
#   "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
#   "expires_in": 900,
#   "token_type": "Bearer"
# }

```

Token Refresh

```

curl -X POST http://localhost:8080/api/auth/refresh \
  -H "Content-Type: application/json" \
  -d '{"refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."}'

# Response:
# {
#   "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
#   "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
#   "expires_in": 900,
#   "token_type": "Bearer"
# }

```

Logout

```

curl -X POST http://localhost:8080/api/auth/logout \
  -H "Content-Type: application/json" \
  -d '{"refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."}'

# Nach Logout: Refresh sollte fehlschlagen
curl -X POST http://localhost:8080/api/auth/refresh \
  -H "Content-Type: application/json" \
  -d '{"refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."}'

# Response: 401 { "error": "Refresh token revoked" }

```

#### 4.2.1.12 Abgabe-Checkliste

- ☐ JwtPayload Model mit Gettern

- ☐ TokenPair Model mit toJson
- ☐ JwtService mit generateAccessToken/generateRefreshToken
- ☐ JwtService mit verifyToken
- ☐ RefreshTokenRepository mit CRUD
- ☐ AuthService.login mit Token-Generierung
- ☐ AuthService.refreshAccessToken mit Token Rotation
- ☐ AuthService.logout mit Token Revocation
- ☐ Auth Handler /refresh Endpoint
- ☐ Auth Handler /logout Endpoint
- ☐ Exception-Klassen für Token-Fehler
- ☐ JWT\_SECRET aus Environment Variable

## 4.2.2 Lösung

### 4.2.2.1 JWT Payload Model

```
// lib/models/jwt_payload.dart

class JwtPayload {
  final String sub;
  final String? email;
  final String? name;
  final String? role;
  final String type;
  final DateTime? expiresAt;
  final DateTime? issuedAt;

  JwtPayload({
    required this.sub,
    this.email,
    this.name,
    this.role,
    required this.type,
    this.expiresAt,
    this.issuedAt,
  });

  int get userId => int.parse(sub);

  bool get isAccessToken => type == 'access';
  bool get isRefreshToken => type == 'refresh';

  bool get isExpired {
    if (expiresAt == null) return false;
    return DateTime.now().isAfter(expiresAt!);
  }

  factory JwtPayload.fromJwtPayload(Map<String, dynamic> payload) {
    return JwtPayload(
      sub: payload['sub'] as String,
      email: payload['email'] as String?,
```

```

        name: payload['name'] as String?,
        role: payload['role'] as String?,
        type: payload['type'] as String? ?? 'access',
        expiresAt: payload['exp'] != null
            ? DateTime.fromMillisecondsSinceEpoch((payload['exp'] as int) * 1000)
            : null,
        issuedAt: payload['iat'] != null
            ? DateTime.fromMillisecondsSinceEpoch((payload['iat'] as int) * 1000)
            : null,
    );
}

Map<String, dynamic> toJson() => {
    'sub': sub,
    if (email != null) 'email': email,
    if (name != null) 'name': name,
    if (role != null) 'role': role,
    'type': type,
};
}

```

#### 4.2.2.2 Token Pair Model

```

// lib/models/token_pair.dart

class TokenPair {
    final String accessToken;
    final String refreshToken;
    final int expiresIn;

    TokenPair({
        required this.accessToken,
        required this.refreshToken,
        required this.expiresIn,
    });

    Map<String, dynamic> toJson() => {
        'access_token': accessToken,
        'refresh_token': refreshToken,
        'expires_in': expiresIn,
        'token_type': 'Bearer',
    };

    factory TokenPair.fromJson(Map<String, dynamic> json) {
        return TokenPair(
            accessToken: json['access_token'] as String,
            refreshToken: json['refresh_token'] as String,
            expiresIn: json['expires_in'] as int,
        );
    }
}

```

```
}
```

#### 4.2.2.3 Exceptions

```
// lib/exceptions/auth_exceptions.dart

class TokenExpiredException implements Exception {
  final String message;
  TokenExpiredException([this.message = 'Token has expired']);

  @override
  String toString() => message;
}

class InvalidTokenException implements Exception {
  final String message;
  InvalidTokenException([this.message = 'Invalid token']);

  @override
  String toString() => message;
}

class RefreshTokenRevokedException implements Exception {
  final String message;
  RefreshTokenRevokedException([this.message = 'Refresh token has been
↪   ↪   revoked']);

  @override
  String toString() => message;
}

class TokenTypeMismatchException implements Exception {
  final String expected;
  final String actual;

  TokenTypeMismatchException(this.expected, this.actual);

  String get message => 'Expected $expected token, got $actual';

  @override
  String toString() => message;
}
```

#### 4.2.2.4 JWT Service

```
// lib/services/jwt_service.dart
import 'package:dart_jsonwebtoken/dart_jsonwebtoken.dart';
import '../models/user.dart';
import '../models/jwt_payload.dart';
```

```
import '../models/token_pair.dart';
import '../exceptions/auth_exceptions.dart';

class JwtService {
  final String _secret;
  final String _issuer;
  final Duration _accessTokenDuration;
  final Duration _refreshTokenDuration;

  JwtService({
    required String secret,
    String issuer = 'my-api',
    Duration? accessTokenDuration,
    Duration? refreshTokenDuration,
  }) : _secret = secret,
       _issuer = issuer,
       _accessTokenDuration = accessTokenDuration ?? const Duration(minutes: 15),
       _refreshTokenDuration = refreshTokenDuration ?? const Duration(days: 7);

  Duration get accessTokenDuration => _accessTokenDuration;
  Duration get refreshTokenDuration => _refreshTokenDuration;

  /// Generiere Access Token für User
  String generateAccessToken(User user) {
    final jwt = JWT(
      {
        'sub': user.id.toString(),
        'email': user.email,
        'name': user.name,
        'role': user.role,
        'type': 'access',
      },
      issuer: _issuer,
      subject: user.id.toString(),
    );

    return jwt.sign(
      SecretKey(_secret),
      expiresIn: _accessTokenDuration,
    );
  }

  /// Generiere Refresh Token für User
  String generateRefreshToken(User user) {
    final jwt = JWT(
      {
        'sub': user.id.toString(),
        'type': 'refresh',
      },
      issuer: _issuer,
```

```
        subject: user.id.toString(),
    );

    return jwt.sign(
        SecretKey(_secret),
        expiresIn: _refreshTokenDuration,
    );
}

/// Generiere Token-Paar
TokenPair generateTokenPair(User user) {
    return TokenPair(
        accessToken: generateAccessToken(user),
        refreshToken: generateRefreshToken(user),
        expiresIn: _accessTokenDuration.inSeconds,
    );
}

/// Verifiziere und dekodiere Token
JwtPayload verifyToken(String token) {
    try {
        final jwt = JWT.verify(token, SecretKey(_secret));
        final payload = jwt.payload as Map<String, dynamic>;
        return JwtPayload.fromJwtPayload(payload);
    } on JWTExpiredException {
        throw TokenExpiredException();
    } on JWTInvalidException catch (e) {
        throw InvalidTokenException('Invalid token: ${e.message}');
    } on JWTException catch (e) {
        throw InvalidTokenException('Token error: ${e.message}');
    }
}

/// Prüfen ob Token gültig ist (ohne Exception)
bool isValid(String token) {
    try {
        verifyToken(token);
        return true;
    } catch (e) {
        return false;
    }
}

/// Dekodiere Token ohne Verifizierung (nur für Debugging!)
Map<String, dynamic>? decodeWithoutVerification(String token) {
    try {
        final jwt = JWT.decode(token);
        return jwt.payload as Map<String, dynamic>;
    } catch (e) {
        return null;
    }
}
```

```

}

/// Extrahiere Token aus Authorization Header
String? extractTokenFromHeader(String? authHeader) {
  if (authHeader == null) return null;
  if (!authHeader.startsWith('Bearer ')) return null;
  return authHeader.substring(7);
}
}

```

#### 4.2.2.5 Refresh Token Repository

```

// lib/repositories/refresh_token_repository.dart
import 'package:postgres/postgres.dart';

class RefreshTokenRecord {
  final int? id;
  final int userId;
  final String token;
  final DateTime createdAt;
  final DateTime expiresAt;
  final bool isRevoked;

  RefreshTokenRecord({
    this.id,
    required this.userId,
    required this.token,
    DateTime? createdAt,
    required this.expiresAt,
    this.isRevoked = false,
  }) : createdAt = createdAt ?? DateTime.now();

  factory RefreshTokenRecord.fromRow(Map<String, dynamic> row) {
    return RefreshTokenRecord(
      id: row['id'] as int,
      userId: row['user_id'] as int,
      token: row['token'] as String,
      createdAt: row['created_at'] as DateTime,
      expiresAt: row['expires_at'] as DateTime,
      isRevoked: row['is_revoked'] as bool,
    );
  }
}

class RefreshTokenRepository {
  final Connection _db;

  RefreshTokenRepository(this._db);

  Future<void> create(RefreshTokenRecord record) async {

```

```
await _db.execute(
    Sql.named('''
        INSERT INTO refresh_tokens (user_id, token, expires_at, is_revoked)
        VALUES (@userId, @token, @expiresAt, @isRevoked)
    '''),
    parameters: {
        'userId': record.userId,
        'token': record.token,
        'expiresAt': record.expiresAt,
        'isRevoked': record.isRevoked,
    },
);
}

Future<RefreshTokenRecord?> findByToken(String token) async {
    final result = await _db.execute(
        Sql.named('''
            SELECT * FROM refresh_tokens
            WHERE token = @token AND is_revoked = false AND expires_at > NOW()
        '''),
        parameters: {'token': token},
    );

    if (result.isEmpty) return null;
    return RefreshTokenRecord.fromRow(result.first.toColumnMap());
}

Future<void> revoke(String token) async {
    await _db.execute(
        Sql.named('''
            UPDATE refresh_tokens
            SET is_revoked = true
            WHERE token = @token
        '''),
        parameters: {'token': token},
    );
}

Future<void> revokeAllForUser(int userId) async {
    await _db.execute(
        Sql.named('''
            UPDATE refresh_tokens
            SET is_revoked = true
            WHERE user_id = @userId AND is_revoked = false
        '''),
        parameters: {'userId': userId},
    );
}

Future<int> deleteExpired() async {
    final result = await _db.execute(
```

```

        Sql.named('''
            DELETE FROM refresh_tokens
            WHERE expires_at < NOW() OR is_revoked = true
            RETURNING id
        '''),
    );
    return result.length;
}

Future<List<RefreshTokenRecord>> findActiveByUser(int userId) async {
    final result = await _db.execute(
        Sql.named('''
            SELECT * FROM refresh_tokens
            WHERE user_id = @userId AND is_revoked = false AND expires_at > NOW()
            ORDER BY created_at DESC
        '''),
        parameters: {'userId': userId},
    );

    return result.map((row) =>
        ↪ RefreshTokenRecord.fromRow(row.toColumnMap())).toList();
    ↪
}

```

#### 4.2.2.6 Auth Service (erweitert)

```

// lib/services/auth_service.dart
import '../models/user.dart';
import '../models/token_pair.dart';
import '../repositories/user_repository.dart';
import '../repositories/refresh_token_repository.dart';
import 'password_service.dart';
import 'jwt_service.dart';
import '../exceptions/auth_exceptions.dart';

class AuthResult {
    final bool success;
    final User? user;
    final TokenPair? tokens;
    final String? error;
    final int statusCode;

    AuthResult.success(this.user, {this.tokens})
        : success = true,
          error = null,
          statusCode = 200;

    AuthResult.failure(this.error, {this.statusCode = 400})
        : success = false,
          user = null,

```

```
        tokens = null;
    }

class AuthService {
    final UserRepository _userRepo;
    final PasswordService _passwordService;
    final JwtService _jwtService;
    final RefreshTokenRepository _refreshTokenRepo;

    AuthService(
        this._userRepo,
        this._passwordService,
        this._jwtService,
        this._refreshTokenRepo,
    );

    /// Login mit Token-Generierung
    Future<AuthResult> login({
        required String email,
        required String password,
    }) async {
        // 1. User finden
        final user = await _userRepo.findByEmail(email.toLowerCase().trim());

        if (user == null) {
            // Timing-Attack verhindern
            _passwordService.hash('dummy_password');
            return AuthResult.failure('Invalid credentials', statusCode: 401);
        }

        // 2. Account aktiv?
        if (!user.isActive) {
            return AuthResult.failure('Account is deactivated', statusCode: 403);
        }

        // 3. Passwort prüfen
        if (!_passwordService.verify(password, user.passwordHash)) {
            return AuthResult.failure('Invalid credentials', statusCode: 401);
        }

        // 4. Token-Paar generieren
        final tokenPair = _jwtService.generateTokenPair(user);

        // 5. Refresh Token speichern
        await _refreshTokenRepo.create(RefreshTokenRecord(
            userId: user.id!,
            token: tokenPair.refreshToken,
            expiresAt: DateTime.now().add(_jwtService.refreshTokenDuration),
        ));

        return AuthResult.success(user, tokens: tokenPair);
    }
}
```

```
}

/// Access Token erneuern
Future<TokenPair> refreshAccessToken(String refreshToken) async {
  // 1. Token verifizieren
  JwtPayload payload;
  try {
    payload = _jwtService.verifyToken(refreshToken);
  } on TokenExpiredException {
    throw TokenExpiredException('Refresh token has expired');
  }

  // 2. Prüfen ob es ein Refresh Token ist
  if (!payload.isRefreshToken) {
    throw TokenTypeMismatchException('refresh', payload.type);
  }

  // 3. In DB prüfen
  final record = await _refreshTokenRepo.findByToken(refreshToken);
  if (record == null) {
    throw RefreshTokenRevokedException();
  }

  // 4. User laden
  final user = await _userRepo.findById(payload.userId);
  if (user == null) {
    throw InvalidTokenException('User not found');
  }

  if (!user.isActive) {
    throw InvalidTokenException('User account is deactivated');
  }

  // 5. Alten Token revoked (Token Rotation)
  await _refreshTokenRepo.revoke(refreshToken);

  // 6. Neues Token-Paar generieren
  final newTokenPair = _jwtService.generateTokenPair(user);

  // 7. Neuen Refresh Token speichern
  await _refreshTokenRepo.create(RefreshTokenRecord(
    userId: user.id!,
    token: newTokenPair.refreshToken,
    expiresAt: DateTime.now().add(_jwtService.refreshTokenDuration),
  ));

  return newTokenPair;
}

/// Logout - Refresh Token invalidieren
Future<void> logout(String refreshToken) async {
```

```

    await _refreshTokenRepo.revoke(refreshToken);
  }

  /// Alle Sessions beenden
  Future<void> logoutAllSessions(int userId) async {
    await _refreshTokenRepo.revokeAllForUser(userId);
  }

  /// Token aus Header verifizieren und User zurückgeben
  Future<User?> getUserFromToken(String? authHeader) async {
    final token = _jwtService.extractTokenFromHeader(authHeader);
    if (token == null) return null;

    try {
      final payload = _jwtService.verifyToken(token);
      if (!payload.isAccessToken) return null;

      return await _userRepo.findById(payload.userId);
    } catch (e) {
      return null;
    }
  }
}

```

#### 4.2.2.7 Auth Handler (erweitert)

```

// lib/handlers/auth_handler.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';
import '../services/auth_service.dart';
import '../exceptions/auth_exceptions.dart';

class AuthHandler {
  final AuthService _authService;

  AuthHandler(this._authService);

  Router get router {
    final router = Router();

    router.post('/register', _register);
    router.post('/login', _login);
    router.post('/refresh', _refresh);
    router.post('/logout', _logout);

    return router;
  }

  Future<Response> _register(Request request) async {

```

```
try {
    final body = await request.readAsString();
    final data = jsonDecode(body) as Map<String, dynamic>;

    final email = data['email'] as String?;
    final password = data['password'] as String?;
    final name = data['name'] as String?;

    if (email == null || password == null) {
        return _jsonResponse(
            {'error': 'Email and password are required'},
            statusCode: 400,
        );
    }

    // Hier würde register aufgerufen (aus Einheit 8.1)
    // final result = await _authService.register(...);

    return _jsonResponse(
        {'message': 'Registration not implemented in this example'},
        statusCode: 501,
    );
} catch (e) {
    return _jsonResponse({'error': 'Registration failed'}, statusCode: 500);
}
}

Future<Response> _login(Request request) async {
    try {
        final body = await request.readAsString();
        final data = jsonDecode(body) as Map<String, dynamic>;

        final email = data['email'] as String?;
        final password = data['password'] as String?;

        if (email == null || password == null) {
            return _jsonResponse(
                {'error': 'Email and password are required'},
                statusCode: 400,
            );
        }

        final result = await _authService.login(
            email: email,
            password: password,
        );

        if (result.success) {
            return _jsonResponse({
                'message': 'Login successful',
                'user': result.user!.toPublicJson(),
            });
        }
    }
}
```

```

        ...result.tokens!.toJson(),
    });
}

return _jsonResponse(
    {'error': result.error},
    statusCode: result.statusCode,
);
} on FormatException {
    return _jsonResponse({'error': 'Invalid JSON'}, statusCode: 400);
} catch (e) {
    return _jsonResponse({'error': 'Login failed'}, statusCode: 500);
}
}

Future<Response> _refresh(Request request) async {
    try {
        final body = await request.readAsString();
        final data = jsonDecode(body) as Map<String, dynamic>;
        final refreshToken = data['refresh_token'] as String?;

        if (refreshToken == null) {
            return _jsonResponse(
                {'error': 'Refresh token is required'},
                statusCode: 400,
            );
        }

        final newTokens = await _authService.refreshAccessToken(refreshToken);

        return _jsonResponse(newTokens.toJson());
    } on TokenExpiredException catch (e) {
        return _jsonResponse({'error': e.message}, statusCode: 401);
    } on RefreshTokenRevokedException catch (e) {
        return _jsonResponse({'error': e.message}, statusCode: 401);
    } on TokenTypeMismatchException catch (e) {
        return _jsonResponse({'error': e.message}, statusCode: 400);
    } on InvalidTokenException catch (e) {
        return _jsonResponse({'error': e.message}, statusCode: 401);
    } on FormatException {
        return _jsonResponse({'error': 'Invalid JSON'}, statusCode: 400);
    } catch (e) {
        return _jsonResponse({'error': 'Token refresh failed'}, statusCode: 500);
    }
}

Future<Response> _logout(Request request) async {
    try {
        final body = await request.readAsString();
        final data = jsonDecode(body) as Map<String, dynamic>;
        final refreshToken = data['refresh_token'] as String?;

```

```
    if (refreshToken != null) {
      await _authService.logout(refreshToken);
    }

    return _jsonResponse({'message': 'Logged out successfully'});
  } catch (e) {
    // Logout sollte nie fehlschlagen aus User-Perspektive
    return _jsonResponse({'message': 'Logged out'});
  }
}

Response _jsonResponse(Map<String, dynamic> data, {int statusCode = 200}) {
  return Response(
    statusCode,
    body: jsonEncode(data),
    headers: {'content-type': 'application/json'},
  );
}
```

#### 4.2.2.8 JWT Config

```
// lib/config/jwt_config.dart
import 'dart:io';

class JwtConfig {
  final String secret;
  final String issuer;
  final Duration accessTokenDuration;
  final Duration refreshTokenDuration;

  JwtConfig({
    required this.secret,
    this.issuer = 'my-api',
    this.accessTokenDuration = const Duration(minutes: 15),
    this.refreshTokenDuration = const Duration(days: 7),
  });

  factory JwtConfig.fromEnvironment() {
    final secret = Platform.environment['JWT_SECRET'];
    if (secret == null || secret.isEmpty) {
      throw Exception('JWT_SECRET environment variable is not set');
    }

    if (secret.length < 32) {
      throw Exception('JWT_SECRET must be at least 32 characters');
    }

    return JwtConfig(
```

```
secret: secret,
issuer: Platform.environment['JWT_ISSUER'] ?? 'my-api',
accessTokenDuration: Duration(
  minutes: int.parse(
    Platform.environment['JWT_ACCESS_DURATION_MINUTES'] ?? '15',
  ),
),
refreshTokenDuration: Duration(
  days: int.parse(
    Platform.environment['JWT_REFRESH_DURATION_DAYS'] ?? '7',
  ),
),
);
}
```

#### 4.2.2.9 Unit Tests

```
// test/jwt_service_test.dart
import 'package:test/test.dart';
import '../lib/services/jwt_service.dart';
import '../lib/models/user.dart';

void main() {
  group('JwtService', () {
    late JwtService jwtService;
    late User testUser;

    setUp(() {
      jwtService = JwtService(
        secret: 'test-secret-key-that-is-at-least-32-characters-long',
        issuer: 'test-api',
        accessTokenDuration: const Duration(minutes: 15),
        refreshTokenDuration: const Duration(days: 7),
      );

      testUser = User(
        id: 1,
        email: 'test@example.com',
        passwordHash: 'hash',
        name: 'Test User',
        role: 'user',
      );
    });

    test('generateAccessToken creates valid token', () {
      final token = jwtService.generateAccessToken(testUser);

      expect(token, isEmpty);
      expect(token.split('.').length, equals(3)); // JWT has 3 parts
    });
  });
}
```

```
});

test('generateRefreshToken creates valid token', () {
    final token = jwtService.generateRefreshToken(testUser);

    expect(token, isEmpty);
});

test('verifyToken returns correct payload for access token', () {
    final token = jwtService.generateAccessToken(testUser);
    final payload = jwtService.verifyToken(token);

    expect(payload.userId, equals(1));
    expect(payload.email, equals('test@example.com'));
    expect(payload.isAccessToken, isTrue);
    expect(payload.isRefreshToken, isFalse);
});

test('verifyToken returns correct payload for refresh token', () {
    final token = jwtService.generateRefreshToken(testUser);
    final payload = jwtService.verifyToken(token);

    expect(payload.userId, equals(1));
    expect(payload.isRefreshToken, isTrue);
    expect(payload.isAccessToken, isFalse);
});

test('verifyToken throws for invalid token', () {
    expect(
        () => jwtService.verifyToken('invalid.token.here'),
        throwsA(isA<InvalidTokenException>()),
    );
});

test('generateTokenPair returns both tokens', () {
    final tokenPair = jwtService.generateTokenPair(testUser);

    expect(tokenPair.accessToken, isEmpty);
    expect(tokenPair.refreshToken, isEmpty);
    expect(tokenPair.expiresIn, equals(900)); // 15 min in seconds
});

test('extractTokenFromHeader parses Bearer token', () {
    final token = jwtService.extractTokenFromHeader('Bearer abc123');
    expect(token, equals('abc123'));
});

test('extractTokenFromHeader returns null for invalid header', () {
    expect(jwtService.extractTokenFromHeader(null), isNull);
    expect(jwtService.extractTokenFromHeader(''), isNull);
    expect(jwtService.extractTokenFromHeader('Basic abc'), isNull);
});
```

```
});
});
}
```

### 4.2.3 Ressourcen

#### 4.2.3.1 Offizielle Dokumentation

- [dart\\_jsonwebtoken Package](#)
- [JWT.io - JWT Debugger und Dokumentation](#)
- [RFC 7519 - JSON Web Token](#)
- [OWASP JWT Cheat Sheet](#)

#### 4.2.3.2 Cheat Sheet: JWT Struktur

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4iLCJpYXQiOi0jE1M
|
+----- Header (Base64) -----+----- Payload (Base64) -----+
+---- Signature ----+
```

**Header (dekodiert):**

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

**Payload (dekodiert):**

```
{
  "sub": "1234567890",
  "name": "John",
  "iat": 1516239022
}
```

#### 4.2.3.3 Cheat Sheet: Registered Claims

Claim	Name	Beschreibung
sub	Subject	User-ID oder eindeutiger Identifier
iss	Issuer	Wer hat das Token ausgestellt
aud	Audience	Für wen ist das Token bestimmt
exp	Expiration	Ablaufzeit (Unix Timestamp)
iat	Issued At	Ausstellungszeit (Unix Timestamp)
nbf	Not Before	Gültig ab (Unix Timestamp)
jti	JWT ID	Eindeutige Token-ID

#### 4.2.3.4 Cheat Sheet: dart\_jsonwebtoken

```
import 'package:dart_jsonwebtoken/dart_jsonwebtoken.dart';

// Token erstellen
```

```

final jwt = JWT(
    {
        'sub': '123',
        'name': 'Max',
        'role': 'admin',
    },
    issuer: 'my-api',
);

// Mit Secret signieren (HS256)
final token = jwt.sign(
    SecretKey('super-secret-key'),
    expiresIn: Duration(hours: 1),
);

// Token verifizieren
try {
    final verified = JWT.verify(token, SecretKey('super-secret-key'));
    print(verified.payload); // {'sub': '123', 'name': 'Max', ...}
} on JWTExpiredException {
    print('Token abgelaufen');
} on JWTException catch (e) {
    print('Ungültiges Token: ${e.message}');
}

// Token ohne Verifizierung dekodieren (UNSICHER!)
final decoded = JWT.decode(token);
print(decoded.payload);

```

#### 4.2.3.5 Cheat Sheet: Algorithmen

Algorithmus	Typ	Beschreibung
<b>HS256</b>	Symmetrisch	HMAC mit SHA-256
<b>HS384</b>	Symmetrisch	HMAC mit SHA-384
<b>HS512</b>	Symmetrisch	HMAC mit SHA-512
<b>RS256</b>	Asymmetrisch	RSA mit SHA-256
<b>RS384</b>	Asymmetrisch	RSA mit SHA-384
<b>RS512</b>	Asymmetrisch	RSA mit SHA-512
<b>ES256</b>	Asymmetrisch	ECDSA mit P-256

**Empfehlung:** - HS256 für einfache APIs (ein Secret) - RS256 für Microservices (Public Key zur Verifizierung)

#### 4.2.3.6 Cheat Sheet: Token Typen

```

// Access Token (kurzlebig)
final accessToken = JWT({
    'sub': userId,
    'email': email,

```

```

    'role': role,
    'type': 'access',
  }).sign(secret, expiresIn: Duration(minutes: 15));

// Refresh Token (langlebig)
final refreshToken = JWT({
  'sub': userId,
  'type': 'refresh',
}).sign(secret, expiresIn: Duration(days: 7));

// ID Token (für OpenID Connect)
final idToken = JWT({
  'sub': userId,
  'email': email,
  'name': name,
  'type': 'id',
}).sign(secret, expiresIn: Duration(hours: 1));

```

#### 4.2.3.7 Cheat Sheet: Token Refresh Flow

Client	Server
+-- Login ----->	
<-- Access + Refresh --+	
+-- API Request ----->	(mit Access Token)
<-- Response -----+	
+-- API Request ----->	(Access Token abgelaufen)
<-- 401 Unauthorized --+	
+-- Refresh ----->	(mit Refresh Token)
<-- New Access Token --+	
+-- API Request ----->	(mit neuem Access Token)
<-- Response -----+	

#### 4.2.3.8 Cheat Sheet: Authorization Header

```

// Token aus Header extrahieren
String? extractToken(Request request) {
  final auth = request.headers['authorization'];
  if (auth == null) return null;
  if (!auth.startsWith('Bearer ')) return null;
  return auth.substring(7);
}

// Request mit Token senden (Client)
final response = await http.get(

```

```
Uri.parse('https://api.example.com/data'),
headers: {
  'Authorization': 'Bearer $accessToken',
},
);
```

#### 4.2.3.9 Best Practices

##### DO

1. **Kurze Access Token Lebensdauer** - 15-60 Minuten
2. **Token Rotation** - Neuer Refresh Token bei jedem Refresh
3. **Sichere Secrets** - Mindestens 256 Bit, zufällig generiert
4. **HTTPS verwenden** - Tokens nie über HTTP
5. **Claims minimal halten** - Nur nötige Daten
6. **Refresh Tokens in DB** - Zum Widerrufen

##### DON'T

1. **Sensitive Daten in Payload** - Passwörter, Secrets
2. **Token in URL** - Als Query Parameter
3. **Token in localStorage** - XSS-Anfällig
4. **Zu lange Lebensdauer** - Risiko bei Kompromittierung
5. **“none” Algorithmus akzeptieren** - Sicherheitslücke

#### 4.2.3.10 SQL Schema

```
CREATE TABLE refresh_tokens (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  token TEXT NOT NULL UNIQUE,
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  expires_at TIMESTAMP NOT NULL,
  is_revoked BOOLEAN NOT NULL DEFAULT false,
  -- Optional: Device/Client Info
  user_agent TEXT,
  ip_address INET
);

CREATE INDEX idx_refresh_tokens_token ON refresh_tokens(token);
CREATE INDEX idx_refresh_tokens_user ON refresh_tokens(user_id);
CREATE INDEX idx_refresh_tokens_expires ON refresh_tokens(expires_at) WHERE ↵
  ↵ is_revoked = false;
```

#### 4.2.3.11 Token Blacklist (für Access Tokens)

```
// Mit Redis
class TokenBlacklist {
  final RedisClient _redis;

  Future<void> blacklist(String token, Duration ttl) async {
```

```

    // Token hashen (Speicher sparen)
    final hash = sha256.convert(utf8.encode(token)).toString();
    await _redis.set('blacklist:$hash', '1', ttl: ttl);
  }

  Future<bool> isBlacklisted(String token) async {
    final hash = sha256.convert(utf8.encode(token)).toString();
    return await _redis.exists('blacklist:$hash');
  }
}

```

#### 4.2.3.12 Secret Generierung

```

# 256-Bit Secret generieren (Linux/Mac)
openssl rand -hex 32

# Mit Dart
dart -e "import 'dart:math'; print(List.generate(32, (_) =>
  ↪ Random.secure().nextInt(256).toRadixString(16).padLeft(2, '0')).join());"
↪

# Ergebnis z.B.: a1b2c3d4e5f6789012345678901234567890abcdef1234567890abcdef123456

```

## 4.3 Einheit 8.3: Auth Middleware

### 4.3.0.1 Lernziele

Nach dieser Einheit kannst du: - Authentication Middleware implementieren - Routen mit Berechtigungen schützen - Role-Based Access Control (RBAC) umsetzen - Guards für verschiedene Anforderungen erstellen

### 4.3.0.2 Middleware-Konzept

Middleware sitzt zwischen dem HTTP-Request und deinem Handler:

```

Request -> Middleware 1 -> Middleware 2 -> Handler -> Response
           ↓               ↓
        (Logging)      (Auth-Check)

```

Shelf Middleware

```

import 'package:shelf/shelf.dart';

Middleware myMiddleware() {
  return (Handler innerHandler) {
    return (Request request) async {
      // VOR dem Handler
      print('Request: ${request.method} ${request.url}');

      // Handler aufrufen
      final response = await innerHandler(request);
    };
  };
}

```

```
    // NACH dem Handler
    print('Response: ${response.statusCode}');

    return response;
  };
};
}
```

#### 4.3.0.3 Authentication Middleware

```
// lib/middleware/auth_middleware.dart
import 'package:shelf/shelf.dart';
import '../services/jwt_service.dart';
import '../models/jwt_payload.dart';

/// Middleware für JWT-Authentifizierung
Middleware authMiddleware(JwtService jwtService) {
  return (Handler innerHandler) {
    return (Request request) async {
      // 1. Token aus Header extrahieren
      final authHeader = request.headers['authorization'];
      final token = jwtService.extractTokenFromHeader(authHeader);

      if (token == null) {
        return Response(401,
          body: '{"error": "No authorization token provided"}',
          headers: {'content-type': 'application/json'},
        );
      }

      // 2. Token verifizieren
      try {
        final payload = jwtService.verifyToken(token);

        // Nur Access Tokens akzeptieren
        if (!payload.isAccessToken) {
          return Response(401,
            body: '{"error": "Invalid token type"}',
            headers: {'content-type': 'application/json'},
          );
        }

        // 3. Payload in Request-Context speichern
        final updatedRequest = request.change(
          context: {
            ...request.context,
            'auth': payload,
            'userId': payload.userId,
          },
        );
      }
    };
  };
}
```

```

        return innerHandler(updatedRequest);
    } on TokenExpiredException {
        return Response(401,
            body: '{"error": "Token has expired"}',
            headers: {'content-type': 'application/json'},
        );
    } on InvalidTokenException {
        return Response(401,
            body: '{"error": "Invalid token"}',
            headers: {'content-type': 'application/json'},
        );
    }
};
};
}

/// Hilfsfunktion: Auth-Payload aus Request lesen
JwtPayload? getAuthPayload(Request request) {
    return request.context['auth'] as JwtPayload?;
}

/// Hilfsfunktion: User-ID aus Request lesen
int? getUserId(Request request) {
    return request.context['userId'] as int?;
}

```

#### 4.3.0.4 Optionale Authentifizierung

```

// lib/middleware/optional_auth_middleware.dart

/// Middleware die Auth-Info hinzufügt, aber nicht erzwingt
Middleware optionalAuthMiddleware(JwtService jwtService) {
    return (Handler innerHandler) {
        return (Request request) async {
            final authHeader = request.headers['authorization'];
            final token = jwtService.extractTokenFromHeader(authHeader);

            if (token != null) {
                try {
                    final payload = jwtService.verifyToken(token);
                    if (payload.isAccessToken) {
                        final updatedRequest = request.change(
                            context: {
                                ...request.context,
                                'auth': payload,
                                'userId': payload.userId,
                            },
                        );
                        return innerHandler(updatedRequest);
                    }
                } catch {}
            }
        };
    };
}

```

```
    }  
  } catch (e) {  
    // Token ungültig - ignorieren  
  }  
}  
  
// Ohne Auth weiter  
return innerHandler(request);  
};  
};  
}
```

#### 4.3.0.5 Role-Based Access Control (RBAC)

Rollen-System

```
// lib/models/role.dart  
  
enum Role {  
  guest,  
  user,  
  moderator,  
  admin,  
  superadmin;  
  
  /// Hierarchie-Level (höher = mehr Rechte)  
  int get level {  
    switch (this) {  
      case Role.guest:  
        return 0;  
      case Role.user:  
        return 10;  
      case Role.moderator:  
        return 50;  
      case Role.admin:  
        return 90;  
      case Role.superadmin:  
        return 100;  
    }  
  }  
}  
  
/// Prüfen ob diese Rolle mindestens so hoch ist wie required  
bool hasAtLeast(Role required) => level >= required.level;  
  
static Role fromString(String? value) {  
  return Role.values.firstWhere(  
    (r) => r.name == value,  
    orElse: () => Role.guest,  
  );  
}  
}
```

## Rollen-Middleware

```
// lib/middleware/role_middleware.dart

/// Middleware die eine Mindest-Rolle erfordert
Middleware requireRole(Role requiredRole) {
  return (Handler innerHandler) {
    return (Request request) async {
      final payload = getAuthPayload(request);

      if (payload == null) {
        return Response(401,
          body: '{"error": "Authentication required"}',
          headers: {'content-type': 'application/json'},
        );
      }

      final userRole = Role.fromString(payload.role);

      if (!userRole.hasAtLeast(requiredRole)) {
        return Response(403,
          body: '{"error": "Insufficient permissions. Required:
↪  ${requiredRole.name}"}',
          headers: {'content-type': 'application/json'},
        );
      }

      return innerHandler(request);
    };
  };
}

/// Convenience-Middleware für häufige Rollen
Middleware requireAdmin() => requireRole(Role.admin);
Middleware requireModerator() => requireRole(Role.moderator);
Middleware requireUser() => requireRole(Role.user);
```

## 4.3.0.6 Permission-Based Access Control

```
// lib/models/permission.dart

class Permission {
  static const String readUsers = 'users:read';
  static const String writeUsers = 'users:write';
  static const String deleteUsers = 'users:delete';

  static const String readProducts = 'products:read';
  static const String writeProducts = 'products:write';
  static const String deleteProducts = 'products:delete';

  static const String readOrders = 'orders:read';
```

```
static const String writeOrders = 'orders:write';
static const String deleteOrders = 'orders:delete';

static const String manageRoles = 'roles:manage';

/// Permissions pro Rolle
static const Map<Role, Set<String>> rolePermissions = {
  Role.guest: {},
  Role.user: {
    readProducts,
    readOrders,
    writeOrders,
  },
  Role.moderator: {
    readUsers,
    readProducts,
    writeProducts,
    readOrders,
    writeOrders,
  },
  Role.admin: {
    readUsers,
    writeUsers,
    readProducts,
    writeProducts,
    deleteProducts,
    readOrders,
    writeOrders,
    deleteOrders,
  },
  Role.superadmin: {
    readUsers,
    writeUsers,
    deleteUsers,
    readProducts,
    writeProducts,
    deleteProducts,
    readOrders,
    writeOrders,
    deleteOrders,
    manageRoles,
  },
};

static bool hasPermission(Role role, String permission) {
  // Superadmin hat alle Rechte
  if (role == Role.superadmin) return true;

  return rolePermissions[role]?.contains(permission) ?? false;
}
```

```

/// Middleware die eine bestimmte Permission erfordert
Middleware requirePermission(String permission) {
  return (Handler innerHandler) {
    return (Request request) async {
      final payload = getAuthPayload(request);

      if (payload == null) {
        return Response(401,
          body: '{"error": "Authentication required"}',
          headers: {'content-type': 'application/json'},
        );
      }

      final userRole = Role.fromString(payload.role);

      if (!Permission.hasPermission(userRole, permission)) {
        return Response(403,
          body: '{"error": "Missing permission: $permission"}',
          headers: {'content-type': 'application/json'},
        );
      }

      return innerHandler(request);
    };
  };
}

```

#### 4.3.0.7 Guards

Guards sind spezialisierte Middleware für komplexere Prüfungen.

Owner Guard

```

// lib/guards/owner_guard.dart

/// Guard der prüft ob der User der Owner einer Ressource ist
Middleware ownerGuard({
  required String paramName,
  required Future<int?> Function(int resourceId) getOwnerId,
  bool allowAdmin = true,
}) {
  return (Handler innerHandler) {
    return (Request request) async {
      final payload = getAuthPayload(request);
      if (payload == null) {
        return Response(401,
          body: '{"error": "Authentication required"}',
          headers: {'content-type': 'application/json'},
        );
      }
    };
  };
}

```

```

// Ressourcen-ID aus URL-Parametern
final resourceIdStr = request.params[paramName];
if (resourceIdStr == null) {
    return innerHandler(request);
}

final resourceId = int.tryParse(resourceIdStr);
if (resourceId == null) {
    return Response(400,
        body: '{"error": "Invalid resource ID"}',
        headers: {'content-type': 'application/json'},
    );
}

// Admin-Bypass
if (allowAdmin) {
    final userRole = Role.fromString(payload.role);
    if (userRole.hasAtLeast(Role.admin)) {
        return innerHandler(request);
    }
}

// Owner prüfen
final ownerId = await getOwnerId(resourceId);
if (ownerId == null || ownerId != payload.userId) {
    return Response(403,
        body: '{"error": "You do not own this resource"}',
        headers: {'content-type': 'application/json'},
    );
}

return innerHandler(request);
};
}

```

Verwendung

```

// Router mit Owner Guard
final router = Router();

router.get('/orders/<id>', (Request request) async {
    // Handler nur erreicht wenn User der Owner ist oder Admin
    final orderId = int.parse(request.params['id']!);
    final order = await orderRepo.findById(orderId);
    return Response.ok(jsonEncode(order?.toJson()));
});

// Mit Guard
final protectedRouter = const Pipeline()
    .addMiddleware(authMiddleware(jwtService))

```

```

    .addMiddleware(ownerGuard(
      paramName: 'id',
      getOwnerId: (id) async {
        final order = await orderRepo.findById(id);
        return order?.userId;
      },
    ))
    .addHandler(router);

```

#### 4.3.0.8 Geschützte Routen

Router-basiert

```

// lib/routes/routes.dart

class Routes {
  final JwtService _jwtService;
  final AuthHandler _authHandler;
  final UserHandler _userHandler;
  final ProductHandler _productHandler;

  Routes(this._jwtService, this._authHandler, this._userHandler,
    ↪ this._productHandler);

  Handler get handler {
    final router = Router();

    // Öffentliche Routen
    router.mount('/api/auth', _authHandler.router);
    router.get('/api/products', _productHandler.list);
    router.get('/api/products/<id>', _productHandler.get);

    // Geschützte Routen (User)
    router.mount('/api/users', _protectedUserRoutes);

    // Admin-Routen
    router.mount('/api/admin', _adminRoutes);

    return router;
  }

  Handler get _protectedUserRoutes {
    final router = Router();

    router.get('/me', _userHandler.getMe);
    router.put('/me', _userHandler.updateMe);
    router.delete('/me', _userHandler.deleteMe);

    return const Pipeline()
      .addMiddleware(authMiddleware(_jwtService))
      .addHandler(router);
  }

```

```

}

Handler get _adminRoutes {
  final router = Router();

  router.get('/users', _userHandler.listAll);
  router.get('/users/<id>', _userHandler.getById);
  router.put('/users/<id>', _userHandler.updateById);
  router.delete('/users/<id>', _userHandler.deleteById);

  return const Pipeline()
    .addMiddleware(authMiddleware(_jwtService))
    .addMiddleware(requireRole(Role.admin))
    .addHandler(router);
}
}

```

Handler-basiert

```

// lib/handlers/product_handler.dart

class ProductHandler {
  final ProductService _productService;
  final JwtService _jwtService;

  ProductHandler(this._productService, this._jwtService);

  Router get router {
    final router = Router();

    // Öffentlich
    router.get('/', list);
    router.get('/:<id>', get);

    // Geschützt
    router.post('/', _withAuth(_withPermission(Permission.writeProducts,
↪ create)));
    ↪ router.put('/:<id>', _withAuth(_withPermission(Permission.writeProducts,
↪ update)));
    ↪ router.delete('/:<id>',
↪ _withAuth(_withPermission(Permission.deleteProducts, delete)));

    return router;
  }

  // Handler-Level Middleware
  Handler _withAuth(Handler handler) {
    return const Pipeline()
      .addMiddleware(authMiddleware(_jwtService))
      .addHandler(handler);
  }
}

```

```

Handler _withPermission(String permission, Handler handler) {
  return const Pipeline()
    .addMiddleware(requirePermission(permission))
    .addHandler(handler);
}

// Handler-Methoden
Future<Response> list(Request request) async {
  final products = await _productService.findAll();
  return Response.ok(jsonEncode(products.map((p) => p.toJson()).toList()));
}

Future<Response> create(Request request) async {
  // Nur erreicht mit gültiger Auth und Permission
  final userId = getUserId(request)!;
  // ...
}
}

```

#### 4.3.0.9 Request Context

```

// lib/middleware/context_middleware.dart

/// Fügt Kontext-Informationen zum Request hinzu
Middleware contextMiddleware({
  required UserRepository userRepo,
}) {
  return (Handler innerHandler) {
    return (Request request) async {
      final context = <String, Object>{
        'requestId': _generateRequestId(),
        'timestamp': DateTime.now(),
      };

      // Falls authentifiziert: User laden
      final payload = getAuthPayload(request);
      if (payload != null) {
        final user = await userRepo.findById(payload.userId);
        if (user != null) {
          context['user'] = user;
        }
      }

      final updatedRequest = request.change(
        context: {...request.context, ...context},
      );

      return innerHandler(updatedRequest);
    };
  };
}

```

```
};  
}  
  
String _generateRequestId() {  
    return DateTime.now().millisecondsSinceEpoch.toRadixString(36) +  
        Random().nextInt(10000).toRadixString(36);  
}  
  
/// User aus Context lesen  
User? getUser(Request request) {  
    return request.context['user'] as User?;  
}
```

#### 4.3.0.10 Middleware-Pipeline

```
// bin/server.dart  
  
void main() async {  
    // Services initialisieren...  
  
    // Routen  
    final publicRouter = Router()  
        ..get('/health', (_) => Response.ok('OK'))  
        ..mount('/api/auth', authHandler.router)  
        ..get('/api/products', productHandler.list);  
  
    final protectedRouter = Router()  
        ..mount('/api/users', userHandler.router)  
        ..mount('/api/orders', orderHandler.router);  
  
    final adminRouter = Router()  
        ..mount('/api/admin/users', adminUserHandler.router)  
        ..mount('/api/admin/settings', settingsHandler.router);  
  
    // Cascade: Versucht jeden Router der Reihe nach  
    final cascade = Cascade()  
        .add(publicRouter)  
        .add(const Pipeline()  
            .addMiddleware(authMiddleware(jwtService))  
            .addHandler(protectedRouter))  
        .add(const Pipeline()  
            .addMiddleware(authMiddleware(jwtService))  
            .addMiddleware(requireRole(Role.admin))  
            .addHandler(adminRouter));  
  
    // Globale Middleware  
    final handler = const Pipeline()  
        .addMiddleware(logRequests())  
        .addMiddleware(corsMiddleware())  
        .addHandler(cascade.handler);
```

```
await serve(handler, 'localhost', 8080);
}
```

#### 4.3.0.11 Zusammenfassung

Konzept	Beschreibung
<b>Auth Middleware</b>	Prüft JWT und fügt Payload zum Context
<b>RBAC</b>	Zugriff basierend auf Rollen
<b>Permissions</b>	Feingranulare Berechtigungen
<b>Guards</b>	Komplexe Zugriffsprüfungen
<b>Context</b>	Request-spezifische Daten durchreichen
<b>Pipeline</b>	Middleware-Ketten für Routen

### 4.3.1 Übung

#### 4.3.1.1 Ziel

Implementiere ein vollständiges Autorisierungssystem mit Middleware, Rollen und Guards.

#### 4.3.1.2 Aufgabe 1: Auth Middleware (15 min)

```
// lib/middleware/auth_middleware.dart

/// Authentication Middleware
/// - Extrahiert JWT aus Authorization Header
/// - Verifiziert Token
/// - Fügt Payload zum Request-Context hinzu
Middleware authMiddleware(JwtService jwtService) {
  return (Handler innerHandler) {
    return (Request request) async {
      // TODO:
      // 1. Authorization Header lesen
      // 2. Token extrahieren (Bearer-Schema)
      // 3. Token verifizieren mit jwtService
      // 4. Prüfen ob es ein Access Token ist
      // 5. Payload in request.context speichern
      // 6. Bei Fehler: 401 Response zurückgeben
    };
  };
}

/// Hilfsfunktion: Auth-Payload aus Request lesen
JwtPayload? getAuthPayload(Request request) {
  // TODO: Aus request.context lesen
}

/// Hilfsfunktion: User-ID aus Request lesen
int? getUserId(Request request) {
```

```
// TODO: Aus Payload extrahieren  
}
```

#### 4.3.1.3 Aufgabe 2: Optional Auth Middleware (10 min)

```
// lib/middleware/optional_auth_middleware.dart  
  
/// Fügt Auth-Info hinzu wenn vorhanden, aber erzwingt keine Auth  
Middleware optionalAuthMiddleware(JwtService jwtService) {  
  return (Handler innerHandler) {  
    return (Request request) async {  
      // TODO:  
      // 1. Token extrahieren (wenn vorhanden)  
      // 2. Token verifizieren (Fehler ignorieren)  
      // 3. Bei gültigem Token: Payload zum Context  
      // 4. Immer zum Handler weiter  
    };  
  };  
}
```

#### 4.3.1.4 Aufgabe 3: Role-Based Access Control (20 min)

Role Enum

```
// lib/models/role.dart  
  
enum Role {  
  guest,  
  user,  
  moderator,  
  admin,  
  superadmin;  
  
  /// Hierarchie-Level  
  int get level {  
    // TODO: Level für jede Rolle (0-100)  
  }  
  
  /// Hat diese Rolle mindestens required?  
  bool hasAtLeast(Role required) {  
    // TODO: Level-Vergleich  
  }  
  
  /// String zu Role konvertieren  
  static Role fromString(String? value) {  
    // TODO: Fallback auf guest  
  }  
}
```

Role Middleware

```
// lib/middleware/role_middleware.dart

/// Middleware die eine Mindest-Rolle erfordert
Middleware requireRole(Role requiredRole) {
  return (Handler innerHandler) {
    return (Request request) async {
      // TODO:
      // 1. Auth-Payload holen
      // 2. Rolle aus Payload extrahieren
      // 3. Rolle prüfen mit hasAtLeast
      // 4. 401 wenn nicht authentifiziert
      // 5. 403 wenn unzureichende Rolle
    };
  };
}

// Convenience-Funktionen
Middleware requireAdmin() => requireRole(Role.admin);
Middleware requireModerator() => requireRole(Role.moderator);
Middleware requireUser() => requireRole(Role.user);
```

#### 4.3.1.5 Aufgabe 4: Permission-Based Access Control (15 min)

```
// lib/models/permission.dart

class Permission {
  // Definiere Permissions als Konstanten
  static const String readUsers = 'users:read';
  static const String writeUsers = 'users:write';
  static const String deleteUsers = 'users:delete';

  static const String readProducts = 'products:read';
  static const String writeProducts = 'products:write';
  static const String deleteProducts = 'products:delete';

  // TODO: Weitere Permissions nach Bedarf

  /// Permissions pro Rolle definieren
  static const Map<Role, Set<String>> rolePermissions = {
    // TODO: Permissions für jede Rolle
    // guest: {}
    // user: {readProducts}
    // moderator: {readProducts, writeProducts}
    // admin: alle Products + Users
    // superadmin: alles
  };

  /// Prüfen ob Rolle eine Permission hat
  static bool hasPermission(Role role, String permission) {
    // TODO: Implementieren (Superadmin hat alle Rechte)
```

```

    }
}

/// Permission Middleware
Middleware requirePermission(String permission) {
  return (Handler innerHandler) {
    return (Request request) async {
      // TODO:
      // 1. Auth-Payload holen
      // 2. Permission prüfen mit hasPermission
      // 3. 403 wenn nicht erlaubt
    };
  };
}

```

#### 4.3.1.6 Aufgabe 5: Owner Guard (15 min)

```

// lib/guards/owner_guard.dart

/// Guard der prüft ob User der Owner einer Ressource ist
///
/// Beispiel: User kann nur eigene Orders sehen/bearbeiten
Middleware ownerGuard({
  required String paramName,
  required Future<int?> Function(int resourceId) getOwnerId,
  bool allowAdmin = true,
}) {
  return (Handler innerHandler) {
    return (Request request) async {
      // TODO:
      // 1. Auth-Payload prüfen
      // 2. Resource-ID aus URL-Params extrahieren
      // 3. Admin-Bypass wenn allowAdmin=true
      // 4. Owner-ID laden mit getOwnerId
      // 5. Mit aktuellem User vergleichen
      // 6. 403 wenn nicht Owner
    };
  };
}

```

#### 4.3.1.7 Aufgabe 6: Protected Router (20 min)

```

// lib/routes/api_router.dart

class ApiRouter {
  final JwtService _jwtService;
  final AuthHandler _authHandler;
  final UserHandler _userHandler;
  final ProductHandler _productHandler;
}

```

```
final OrderHandler _orderHandler;

ApiRouter({
  required JwtService jwtService,
  required AuthHandler authHandler,
  required UserHandler userHandler,
  required ProductHandler productHandler,
  required OrderHandler orderHandler,
}) : _jwtService = jwtService,
    _authHandler = authHandler,
    _userHandler = userHandler,
    _productHandler = productHandler,
    _orderHandler = orderHandler;

Handler get handler {
  final router = Router();

  // Öffentliche Routen
  router.mount('/auth', _publicAuthRoutes);
  router.mount('/products', _publicProductRoutes);

  // Geschützte User-Routen
  router.mount('/users', _protectedUserRoutes);

  // Geschützte Order-Routen
  router.mount('/orders', _protectedOrderRoutes);

  // Admin-Routen
  router.mount('/admin', _adminRoutes);

  return router;
}

Handler get _publicAuthRoutes {
  // TODO: Auth-Routen ohne Middleware
}

Handler get _publicProductRoutes {
  final router = Router();
  // TODO: GET / und GET /<id> ohne Auth
  return router;
}

Handler get _protectedUserRoutes {
  // TODO:
  // GET /me - eigenes Profil
  // PUT /me - Profil bearbeiten
  // DELETE /me - Account löschen
  // Mit authMiddleware
}
```

```
Handler get _protectedOrderRoutes {
  // TODO:
  // GET / - eigene Orders
  // POST / - neue Order
  // GET /<id> - nur eigene Order (ownerGuard)
  // Mit authMiddleware + ownerGuard
}

Handler get _adminRoutes {
  // TODO:
  // CRUD für alle User/Products/Orders
  // Mit authMiddleware + requireAdmin
}
}
```

#### 4.3.1.8 Aufgabe 7: Handler mit Context (10 min)

```
// lib/handlers/user_handler.dart

class UserHandler {
  final UserService _userService;

  UserHandler(this._userService);

  /// GET /users/me - Eigenes Profil
  Future<Response> getMe(Request request) async {
    // TODO:
    // 1. userId aus Context holen
    // 2. User laden
    // 3. Als JSON zurückgeben
  }

  /// PUT /users/me - Profil bearbeiten
  Future<Response> updateMe(Request request) async {
    // TODO:
    // 1. userId aus Context
    // 2. Body parsen
    // 3. Update durchführen
    // 4. Aktualisiertes Profil zurückgeben
  }

  /// DELETE /users/me - Account löschen
  Future<Response> deleteMe(Request request) async {
    // TODO:
    // 1. userId aus Context
    // 2. Account deaktivieren/löschen
    // 3. 204 No Content zurückgeben
  }
}
```

**4.3.1.9 Aufgabe 8: Integration Test (Bonus, 15 min)**

```
// test/middleware/auth_middleware_test.dart

void main() {
  late JwtService jwtService;
  late Handler testHandler;

  setUp(() {
    jwtService = JwtService(secret: 'test-secret-key-at-least-32-chars');

    final innerHandler = (Request request) {
      final userId = getUserId(request);
      return Response.ok('User: $userId');
    };

    testHandler = const Pipeline()
      .addMiddleware(authMiddleware(jwtService))
      .addHandler(innerHandler);
  });

  test('rejects request without token', () async {
    final request = Request('GET', Uri.parse('http://localhost/test'));
    final response = await testHandler(request);

    expect(response.statusCode, equals(401));
  });

  test('rejects request with invalid token', () async {
    final request = Request(
      'GET',
      Uri.parse('http://localhost/test'),
      headers: {'authorization': 'Bearer invalid-token'},
    );
    final response = await testHandler(request);

    expect(response.statusCode, equals(401));
  });

  test('accepts request with valid token', () async {
    // TODO:
    // 1. Gültigen Token generieren
    // 2. Request mit Token senden
    // 3. 200 OK erwarten
    // 4. Body prüfen
  });

  test('adds user info to context', () async {
    // TODO:
    // 1. Token für User mit ID 123 generieren
    // 2. Request senden
  });
}
```

```
// 3. Prüfen ob Handler die User-ID aus Context bekommt
});
}
```

#### 4.3.1.10 Testen

Ohne Token

```
curl http://localhost:8080/api/users/me
# 401 {"error": "No authorization token provided"}
```

Mit gültigem Token

```
# Zuerst einloggen
TOKEN=$(curl -s -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email": "user@example.com", "password": "Password123!"}' \
  | jq -r '.access_token')

# Geschützte Route aufrufen
curl http://localhost:8080/api/users/me \
  -H "Authorization: Bearer $TOKEN"
# 200 {"id": 1, "email": "user@example.com", ...}
```

Admin-Route als User

```
curl http://localhost:8080/api/admin/users \
  -H "Authorization: Bearer $TOKEN"
# 403 {"error": "Insufficient permissions. Required: admin"}
```

Fremde Ressource

```
curl http://localhost:8080/api/orders/999 \
  -H "Authorization: Bearer $TOKEN"
# 403 {"error": "You do not own this resource"}
```

#### 4.3.1.11 Abgabe-Checkliste

- ☐ authMiddleware mit Token-Extraktion und -Verifizierung
- ☐ optionalAuthMiddleware
- ☐ Hilfsfunktionen getAuthPayload und getUserId
- ☐ Role Enum mit Level-System
- ☐ requireRole Middleware
- ☐ Permission-Klasse mit Rollen-Mapping
- ☐ requirePermission Middleware
- ☐ ownerGuard mit Admin-Bypass
- ☐ ApiRouter mit öffentlichen und geschützten Routen
- ☐ Handler die userId aus Context nutzen
- ☐ (Bonus) Integration Tests

## 4.3.2 Lösung

### 4.3.2.1 Auth Middleware

```
// lib/middleware/auth_middleware.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';
import '../services/jwt_service.dart';
import '../models/jwt_payload.dart';
import '../exceptions/auth_exceptions.dart';

/// Authentication Middleware
Middleware authMiddleware(JwtService jwtService) {
  return (Handler innerHandler) {
    return (Request request) async {
      // 1. Authorization Header lesen
      final authHeader = request.headers['authorization'];
      final token = jwtService.extractTokenFromHeader(authHeader);

      if (token == null) {
        return _errorResponse(401, 'No authorization token provided');
      }

      // 2. Token verifizieren
      try {
        final payload = jwtService.verifyToken(token);

        // 3. Nur Access Tokens akzeptieren
        if (!payload.isAccessToken) {
          return _errorResponse(401, 'Invalid token type');
        }

        // 4. Payload in Context speichern
        final updatedRequest = request.change(
          context: {
            ...request.context,
            'auth': payload,
            'userId': payload.userId,
            'userRole': payload.role,
          },
        );

        return innerHandler(updatedRequest);
      } on TokenExpiredException {
        return _errorResponse(401, 'Token has expired');
      } on InvalidTokenException catch (e) {
        return _errorResponse(401, e.message);
      } catch (e) {
        return _errorResponse(401, 'Invalid token');
      }
    };
  };
};
```

```
}

/// Optional Authentication Middleware
Middleware optionalAuthMiddleware(JwtService jwtService) {
    return (Handler innerHandler) {
        return (Request request) async {
            final authHeader = request.headers['authorization'];
            final token = jwtService.extractTokenFromHeader(authHeader);

            if (token != null) {
                try {
                    final payload = jwtService.verifyToken(token);
                    if (payload.isAccessToken) {
                        final updatedRequest = request.change(
                            context: {
                                ...request.context,
                                'auth': payload,
                                'userId': payload.userId,
                                'userRole': payload.role,
                            },
                        );
                        return innerHandler(updatedRequest);
                    }
                } catch (e) {
                    // Token ungültig - ignorieren
                }
            }

            return innerHandler(request);
        };
    };
}

// Hilfsfunktionen
JwtPayload? getAuthPayload(Request request) {
    return request.context['auth'] as JwtPayload?;
}

int? getUserId(Request request) {
    return request.context['userId'] as int?;
}

String? getUserRole(Request request) {
    return request.context['userRole'] as String?;
}

Response _errorResponse(int statusCode, String message) {
    return Response(
        statusCode,
        body: jsonEncode({'error': message}),
        headers: {'content-type': 'application/json'},
    );
}
```

```
);  
}
```

#### 4.3.2.2 Role System

```
// lib/models/role.dart  
  
enum Role {  
  guest,  
  user,  
  moderator,  
  admin,  
  superadmin;  
  
  int get level {  
    switch (this) {  
      case Role.guest:  
        return 0;  
      case Role.user:  
        return 10;  
      case Role.moderator:  
        return 50;  
      case Role.admin:  
        return 90;  
      case Role.superadmin:  
        return 100;  
    }  
  }  
}  
  
bool hasAtLeast(Role required) => level >= required.level;  
  
bool canManage(Role other) => level > other.level;  
  
static Role fromString(String? value) {  
  if (value == null) return Role.guest;  
  return Role.values.firstWhere(  
    (r) => r.name.toLowerCase() == value.toLowerCase(),  
    orElse: () => Role.guest,  
  );  
}  
  
String get displayName {  
  switch (this) {  
    case Role.guest:  
    return 'Guest';  
    case Role.user:  
    return 'User';  
    case Role.moderator:  
    return 'Moderator';  
    case Role.admin:
```

```
        return 'Administrator';
      case Role.superadmin:
        return 'Super Administrator';
    }
  }
}
```

#### 4.3.2.3 Role Middleware

```
// lib/middleware/role_middleware.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';
import '../models/role.dart';
import 'auth_middleware.dart';

/// Middleware die eine Mindest-Rolle erfordert
Middleware requireRole(Role requiredRole) {
  return (Handler innerHandler) {
    return (Request request) async {
      final payload = getAuthPayload(request);

      if (payload == null) {
        return Response(
          401,
          body: jsonEncode({'error': 'Authentication required'}),
          headers: {'content-type': 'application/json'},
        );
      }

      final userRole = Role.fromString(payload.role);

      if (!userRole.hasAtLeast(requiredRole)) {
        return Response(
          403,
          body: jsonEncode({
            'error': 'Insufficient permissions',
            'required': requiredRole.name,
            'current': userRole.name,
          }),
          headers: {'content-type': 'application/json'},
        );
      }

      return innerHandler(request);
    };
  };
}

// Convenience-Funktionen
Middleware requireAdmin() => requireRole(Role.admin);
```

```
Middleware requireModerator() => requireRole(Role.moderator);
Middleware requireUser() => requireRole(Role.user);
Middleware requireSuperadmin() => requireRole(Role.superadmin);
```

#### 4.3.2.4 Permission System

```
// lib/models/permission.dart
import 'role.dart';

class Permission {
  // User Permissions
  static const String readUsers = 'users:read';
  static const String writeUsers = 'users:write';
  static const String deleteUsers = 'users:delete';

  // Product Permissions
  static const String readProducts = 'products:read';
  static const String writeProducts = 'products:write';
  static const String deleteProducts = 'products:delete';

  // Order Permissions
  static const String readOrders = 'orders:read';
  static const String writeOrders = 'orders:write';
  static const String deleteOrders = 'orders:delete';

  // Admin Permissions
  static const String manageRoles = 'roles:manage';
  static const String viewLogs = 'logs:view';
  static const String manageSettings = 'settings:manage';

  /// Permissions pro Rolle
  static const Map<Role, Set<String>> rolePermissions = {
    Role.guest: {
      readProducts,
    },
    Role.user: {
      readProducts,
      readOrders,
      writeOrders,
    },
    Role.moderator: {
      readUsers,
      readProducts,
      writeProducts,
      readOrders,
      writeOrders,
      viewLogs,
    },
    Role.admin: {
      readUsers,
```

```

        writeUsers,
        readProducts,
        writeProducts,
        deleteProducts,
        readOrders,
        writeOrders,
        deleteOrders,
        viewLogs,
    },
    Role.superadmin: {
        // Hat alle Permissions (wird in hasPermission gehandelt)
    },
};

/// Alle verfügbaren Permissions
static Set<String> get allPermissions => {
    readUsers, writeUsers, deleteUsers,
    readProducts, writeProducts, deleteProducts,
    readOrders, writeOrders, deleteOrders,
    manageRoles, viewLogs, manageSettings,
};

/// Prüfen ob Rolle eine Permission hat
static bool hasPermission(Role role, String permission) {
    // Superadmin hat alle Rechte
    if (role == Role.superadmin) return true;

    return rolePermissions[role]?.contains(permission) ?? false;
}

/// Mehrere Permissions prüfen (alle müssen erfüllt sein)
static bool hasAllPermissions(Role role, List<String> permissions) {
    return permissions.every((p) => hasPermission(role, p));
}

/// Mindestens eine Permission haben
static bool hasAnyPermission(Role role, List<String> permissions) {
    return permissions.any((p) => hasPermission(role, p));
}
}

```

#### 4.3.2.5 Permission Middleware

```

// lib/middleware/permission_middleware.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';
import '../models/role.dart';
import '../models/permission.dart';
import 'auth_middleware.dart';

```

```
/// Middleware die eine bestimmte Permission erfordert
Middleware requirePermission(String permission) {
  return (Handler innerHandler) {
    return (Request request) async {
      final payload = getAuthPayload(request);

      if (payload == null) {
        return Response(
          401,
          body: jsonEncode({'error': 'Authentication required'}),
          headers: {'content-type': 'application/json'},
        );
      }

      final userRole = Role.fromString(payload.role);

      if (!Permission.hasPermission(userRole, permission)) {
        return Response(
          403,
          body: jsonEncode({
            'error': 'Missing permission',
            'required': permission,
          }),
          headers: {'content-type': 'application/json'},
        );
      }

      return innerHandler(request);
    };
  };
}

/// Mehrere Permissions erforderlich (alle)
Middleware requireAllPermissions(List<String> permissions) {
  return (Handler innerHandler) {
    return (Request request) async {
      final payload = getAuthPayload(request);

      if (payload == null) {
        return Response(
          401,
          body: jsonEncode({'error': 'Authentication required'}),
          headers: {'content-type': 'application/json'},
        );
      }

      final userRole = Role.fromString(payload.role);

      if (!Permission.hasAllPermissions(userRole, permissions)) {
        final missing = permissions
          .where((p) => !Permission.hasPermission(userRole, p))
      }
    };
  };
}
```

```

        .toList();
    return Response(
        403,
        body: jsonEncode({
            'error': 'Missing permissions',
            'required': missing,
        }),
        headers: {'content-type': 'application/json'},
    );
}

return innerHandler(request);
};
};
}

/// Mindestens eine Permission erforderlich
Middleware requireAnyPermission(List<String> permissions) {
    return (Handler innerHandler) {
        return (Request request) async {
            final payload = getAuthPayload(request);

            if (payload == null) {
                return Response(
                    401,
                    body: jsonEncode({'error': 'Authentication required'}),
                    headers: {'content-type': 'application/json'},
                );
            }

            final userRole = Role.fromString(payload.role);

            if (!Permission.hasAnyPermission(userRole, permissions)) {
                return Response(
                    403,
                    body: jsonEncode({
                        'error': 'Requires at least one of these permissions',
                        'required': permissions,
                    }),
                    headers: {'content-type': 'application/json'},
                );
            }

            return innerHandler(request);
        };
    };
}

```

#### 4.3.2.6 Owner Guard

```
// lib/guards/owner_guard.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';
import '../models/role.dart';
import '../middleware/auth_middleware.dart';

/// Guard der prüft ob User der Owner einer Ressource ist
Middleware ownerGuard({
  required String paramName,
  required Future<int?> Function(int resourceId) getOwnerId,
  bool allowAdmin = true,
  Role? adminRole,
}) {
  final requiredAdminRole = adminRole ?? Role.admin;

  return (Handler innerHandler) {
    return (Request request) async {
      final payload = getAuthPayload(request);

      if (payload == null) {
        return Response(
          401,
          body: jsonEncode({'error': 'Authentication required'}),
          headers: {'content-type': 'application/json'},
        );
      }

      // Resource-ID aus URL-Params
      final resourceIdStr = request.params[paramName];
      if (resourceIdStr == null) {
        // Kein Parameter - weiter zum Handler
        return innerHandler(request);
      }

      final resourceId = int.tryParse(resourceIdStr);
      if (resourceId == null) {
        return Response(
          400,
          body: jsonEncode({'error': 'Invalid resource ID'}),
          headers: {'content-type': 'application/json'},
        );
      }

      // Admin-Bypass
      if (allowAdmin) {
        final userRole = Role.fromString(payload.role);
        if (userRole.hasAtLeast(requiredAdminRole)) {
          return innerHandler(request);
        }
      }
    };
  };
}
```

```
}

// Owner prüfen
final ownerId = await getOwnerId(resourceId);

if (ownerId == null) {
    return Response(
        404,
        body: jsonEncode({'error': 'Resource not found'}),
        headers: {'content-type': 'application/json'},
    );
}

if (ownerId != payload.userId) {
    return Response(
        403,
        body: jsonEncode({'error': 'You do not own this resource'}),
        headers: {'content-type': 'application/json'},
    );
}

return innerHandler(request);
};
}

/// Selbst-oder-Admin Guard
/// Erlaubt Zugriff wenn User sich selbst oder Admin
Middleware selfOrAdminGuard({String paramName = 'id'}) {
    return (Handler innerHandler) {
        return (Request request) async {
            final payload = getAuthPayload(request);

            if (payload == null) {
                return Response(
                    401,
                    body: jsonEncode({'error': 'Authentication required'}),
                    headers: {'content-type': 'application/json'},
                );
            }

            final targetIdStr = request.params[paramName];
            if (targetIdStr == null) {
                return innerHandler(request);
            }

            final targetId = int.tryParse(targetIdStr);
            if (targetId == null) {
                return Response(
                    400,
                    body: jsonEncode({'error': 'Invalid ID'}),
                );
            }
        };
    };
}
```

```

        headers: {'content-type': 'application/json'},
    );
}

// Selbst?
if (targetId == payload.userId) {
    return innerHandler(request);
}

// Admin?
final userRole = Role.fromString(payload.role);
if (userRole.hasAtLeast(Role.admin)) {
    return innerHandler(request);
}

return Response(
    403,
    body: jsonEncode({'error': 'Access denied'}),
    headers: {'content-type': 'application/json'},
);
};
};
}

```

#### 4.3.2.7 API Router

```

// lib/routes/api_router.dart
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';
import '../services/jwt_service.dart';
import '../handlers/auth_handler.dart';
import '../handlers/user_handler.dart';
import '../handlers/product_handler.dart';
import '../handlers/order_handler.dart';
import '../middleware/auth_middleware.dart';
import '../middleware/role_middleware.dart';
import '../middleware/permission_middleware.dart';
import '../guards/owner_guard.dart';
import '../models/permission.dart';
import '../repositories/order_repository.dart';

class ApiRouter {
    final JwtService _jwtService;
    final AuthHandler _authHandler;
    final UserHandler _userHandler;
    final ProductHandler _productHandler;
    final OrderHandler _orderHandler;
    final OrderRepository _orderRepo;

    ApiRouter({

```

```
required JwtService jwtService,
required AuthHandler authHandler,
required UserHandler userHandler,
required ProductHandler productHandler,
required OrderHandler orderHandler,
required OrderRepository orderRepo,
}) : _jwtService = jwtService,
    _authHandler = authHandler,
    _userHandler = userHandler,
    _productHandler = productHandler,
    _orderHandler = orderHandler,
    _orderRepo = orderRepo;

Handler get handler {
  final router = Router();

  // Öffentliche Routen
  router.mount('/auth', _publicAuthRoutes);
  router.mount('/products', _publicProductRoutes);

  // Geschützte User-Routen
  router.mount('/users', _protectedUserRoutes);

  // Geschützte Order-Routen
  router.mount('/orders', _protectedOrderRoutes);

  // Admin-Routen
  router.mount('/admin', _adminRoutes);

  return router;
}

Handler get _publicAuthRoutes {
  return _authHandler.router;
}

Handler get _publicProductRoutes {
  final router = Router();
  router.get('/', _productHandler.list);
  router.get('/:id', _productHandler.get);
  return router;
}

Handler get _protectedUserRoutes {
  final router = Router();

  router.get('/me', _userHandler.getMe);
  router.put('/me', _userHandler.updateMe);
  router.delete('/me', _userHandler.deleteMe);

  return const Pipeline()
```

```
        .addMiddleware(authMiddleware(_jwtService))
        .addHandler(router);
    }

    Handler get _protectedOrderRoutes {
        final router = Router();

        // Liste eigener Orders
        router.get('/', _orderHandler.listMine);

        // Neue Order erstellen
        router.post('/', _orderHandler.create);

        // Einzelne Order (nur eigene oder Admin)
        final orderDetailRouter = Router();
        orderDetailRouter.get('/:<id>', _orderHandler.get);
        orderDetailRouter.put('/:<id>', _orderHandler.update);
        orderDetailRouter.delete('/:<id>', _orderHandler.cancel);

        router.mount(
            '/',
            const Pipeline()
                .addMiddleware(ownerGuard(
                    paramName: 'id',
                    getOwnerId: (id) async {
                        final order = await _orderRepo.findById(id);
                        return order?.userId;
                    },
                ))
                .addHandler(orderDetailRouter),
        );

        return const Pipeline()
            .addMiddleware(authMiddleware(_jwtService))
            .addHandler(router);
    }

    Handler get _adminRoutes {
        final router = Router();

        // User Management
        router.get('/users', _userHandler.listAll);
        router.get('/users/<id>', _userHandler.getById);
        router.put('/users/<id>', _userHandler.updateById);
        router.delete('/users/<id>', _userHandler.deleteById);

        // Product Management
        router.post('/products', _productHandler.create);
        router.put('/products/<id>', _productHandler.update);
        router.delete('/products/<id>', _productHandler.delete);
    }
}
```

```

// Order Management
router.get('/orders', _orderHandler.listAll);
router.get('/orders/<id>', _orderHandler.getAny);
router.put('/orders/<id>', _orderHandler.updateAny);

return const Pipeline()
  .addMiddleware(authMiddleware(_jwtService))
  .addMiddleware(requireAdmin())
  .addHandler(router);
}
}

```

#### 4.3.2.8 User Handler

```

// lib/handlers/user_handler.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';
import '../services/user_service.dart';
import '../middleware/auth_middleware.dart';

class UserHandler {
  final UserService _userService;

  UserHandler(this._userService);

  /// GET /users/me - Eigenes Profil
  Future<Response> getMe(Request request) async {
    final userId = getUserId(request);

    if (userId == null) {
      return Response(401,
        body: jsonEncode({'error': 'Not authenticated'}),
        headers: {'content-type': 'application/json'},
      );
    }

    final user = await _userService.findById(userId);

    if (user == null) {
      return Response(404,
        body: jsonEncode({'error': 'User not found'}),
        headers: {'content-type': 'application/json'},
      );
    }

    return Response.ok(
      jsonEncode(user.toPublicJson()),
      headers: {'content-type': 'application/json'},
    );
  }
}

```

```
/// PUT /users/me - Profil bearbeiten
Future<Response> updateMe(Request request) async {
  final userId = getUserId(request);

  if (userId == null) {
    return Response(401,
      body: jsonEncode({'error': 'Not authenticated'}),
      headers: {'content-type': 'application/json'},
    );
  }

  try {
    final body = await request.readAsString();
    final data = jsonDecode(body) as Map<String, dynamic>;

    final user = await _userService.update(userId, data);

    return Response.ok(
      jsonEncode(user.toPublicJson()),
      headers: {'content-type': 'application/json'},
    );
  } catch (e) {
    return Response(400,
      body: jsonEncode({'error': e.toString()}),
      headers: {'content-type': 'application/json'},
    );
  }
}

/// DELETE /users/me - Account löschen
Future<Response> deleteMe(Request request) async {
  final userId = getUserId(request);

  if (userId == null) {
    return Response(401,
      body: jsonEncode({'error': 'Not authenticated'}),
      headers: {'content-type': 'application/json'},
    );
  }

  await _userService.deactivate(userId);

  return Response(204);
}

/// GET /admin/users - Alle User (Admin)
Future<Response> listAll(Request request) async {
  final users = await _userService.findAll();

  return Response.ok(
```

```
        jsonEncode(users.map((u) => u.toPublicJson()).toList()),
        headers: {'content-type': 'application/json'},
    );
}

/// GET /admin/users/:id - User by ID (Admin)
Future<Response> getById(Request request) async {
    final id = int.tryParse(request.params['id'] ?? '');

    if (id == null) {
        return Response(400,
            body: jsonEncode({'error': 'Invalid ID'}),
            headers: {'content-type': 'application/json'},
        );
    }

    final user = await _userService.findById(id);

    if (user == null) {
        return Response(404,
            body: jsonEncode({'error': 'User not found'}),
            headers: {'content-type': 'application/json'},
        );
    }

    return Response.ok(
        jsonEncode(user.toPublicJson()),
        headers: {'content-type': 'application/json'},
    );
}

Future<Response> updateById(Request request) async {
    final id = int.tryParse(request.params['id'] ?? '');
    if (id == null) {
        return Response(400,
            body: jsonEncode({'error': 'Invalid ID'}),
            headers: {'content-type': 'application/json'},
        );
    }

    final body = await request.readAsString();
    final data = jsonDecode(body) as Map<String, dynamic>;

    final user = await _userService.update(id, data);

    return Response.ok(
        jsonEncode(user.toPublicJson()),
        headers: {'content-type': 'application/json'},
    );
}
```

```
Future<Response> deleteById(Request request) async {
  final id = int.tryParse(request.params['id'] ?? '');
  if (id == null) {
    return Response(400,
      body: jsonEncode({'error': 'Invalid ID'}),
      headers: {'content-type': 'application/json'},
    );
  }

  await _userService.deactivate(id);

  return Response(204);
}
```

#### 4.3.2.9 Tests

```
// test/middleware/auth_middleware_test.dart
import 'package:test/test.dart';
import 'package:shelf/shelf.dart';
import '../lib/services/jwt_service.dart';
import '../lib/middleware/auth_middleware.dart';
import '../lib/models/user.dart';

void main() {
  late JwtService jwtService;

  setUp(() {
    jwtService = JwtService(
      secret: 'test-secret-key-that-is-long-enough-for-testing',
    );
  });

  group('authMiddleware', () {
    test('rejects request without token', () async {
      final handler = const Pipeline()
        .addMiddleware(authMiddleware(jwtService))
        .addHandler((r) => Response.ok('OK'));

      final request = Request('GET', Uri.parse('http://localhost/test'));
      final response = await handler(request);

      expect(response.statusCode, equals(401));
    });

    test('rejects invalid token', () async {
      final handler = const Pipeline()
        .addMiddleware(authMiddleware(jwtService))
        .addHandler((r) => Response.ok('OK'));
    });
  });
}
```

```
    final request = Request(
        'GET',
        Uri.parse('http://localhost/test'),
        headers: {'authorization': 'Bearer invalid'},
    );
    final response = await handler(request);

    expect(response.statusCode, equals(401));
});

test('accepts valid token and adds context', () async {
    final user = User(
        id: 123,
        email: 'test@example.com',
        passwordHash: 'hash',
        role: 'user',
    );

    final token = jwtService.generateAccessToken(user);

    int? receivedUserId;
    final handler = const Pipeline()
        .addMiddleware(authMiddleware(jwtService))
        .addHandler((r) {
            receivedUserId = getUserId(r);
            return Response.ok('OK');
        });

    final request = Request(
        'GET',
        Uri.parse('http://localhost/test'),
        headers: {'authorization': 'Bearer $token'},
    );
    final response = await handler(request);

    expect(response.statusCode, equals(200));
    expect(receivedUserId, equals(123));
});
}
```

### 4.3.3 Ressourcen

#### 4.3.3.1 Offizielle Dokumentation

- Shelf Middleware
- shelf\_router Package
- OWASP Authorization Cheat Sheet

#### 4.3.3.2 Cheat Sheet: Shelf Middleware

```
// Middleware Grundstruktur
Middleware myMiddleware() {
    return (Handler innerHandler) {
        return (Request request) async {
            // VOR dem Handler
            // ...

            // Handler aufrufen
            final response = await innerHandler(request);

            // NACH dem Handler
            // ...

            return response;
        };
    };
}

// Middleware verketten
final handler = const Pipeline()
    .addMiddleware(logRequests())
    .addMiddleware(authMiddleware(jwtService))
    .addMiddleware(requireRole(Role.admin))
    .addHandler(router);
```

#### 4.3.3.3 Cheat Sheet: Request Context

```
// Wert zum Context hinzufügen
final updatedRequest = request.change(
    context: {
        ...request.context,
        'userId': 123,
        'role': 'admin',
    },
);

// Wert aus Context lesen
final userId = request.context['userId'] as int?;

// Typ-sichere Helper-Funktion
T? getContext<T>(Request request, String key) {
    return request.context[key] as T?;
}
```

#### 4.3.3.4 Cheat Sheet: HTTP Status Codes

Code	Name	Verwendung
200	OK	Erfolgreiche Anfrage
201	Created	Ressource erstellt
204	No Content	Erfolg ohne Body
400	Bad Request	Ungültige Anfrage
<b>401</b>	<b>Unauthorized</b>	<b>Nicht authentifiziert</b>
<b>403</b>	<b>Forbidden</b>	<b>Keine Berechtigung</b>
404	Not Found	Ressource nicht gefunden
409	Conflict	Konflikt (z.B. Duplikat)
429	Too Many Requests	Rate Limit
500	Internal Server Error	Server-Fehler

#### 4.3.3.5 Cheat Sheet: Role-Based Access Control

```
// Hierarchisches Rollen-System
enum Role {
    guest(0),
    user(10),
    moderator(50),
    admin(90),
    superadmin(100);

    final int level;
    const Role(this.level);

    bool hasAtLeast(Role other) => level >= other.level;
}

// Middleware
Middleware requireRole(Role required) {
    return (Handler handler) => (Request request) async {
        final userRole = getUserRole(request);
        if (!userRole.hasAtLeast(required)) {
            return Response(403, body: 'Forbidden');
        }
        return handler(request);
    };
}
```

#### 4.3.3.6 Cheat Sheet: Permission-Based Access Control

```
// Permissions als Konstanten
class Permissions {
    static const read = 'resource:read';
    static const write = 'resource:write';
    static const delete = 'resource:delete';
    static const admin = 'resource:admin';
}
```

```
// Rollen -> Permissions Mapping
final rolePermissions = {
  Role.user: {Permissions.read},
  Role.moderator: {Permissions.read, Permissions.write},
  Role.admin: {Permissions.read, Permissions.write, Permissions.delete},
};

// Middleware
Middleware requirePermission(String permission) {
  return (Handler handler) => (Request request) async {
    final role = getUserRole(request);
    if (!hasPermission(role, permission)) {
      return Response(403, body: 'Missing permission: $permission');
    }
    return handler(request);
  };
}
```

#### 4.3.3.7 Cheat Sheet: Guard Patterns

```
// Owner Guard
Middleware ownerGuard(Future<int?> Function(int) getOwnerId) {
  return (Handler handler) => (Request request) async {
    final resourceId = getResourceId(request);
    final userId = getUserId(request);
    final ownerId = await getOwnerId(resourceId);

    if (ownerId != userId && !isAdmin(request)) {
      return Response(403, body: 'Not owner');
    }
    return handler(request);
  };
}

// Self-or-Admin Guard
Middleware selfOrAdmin({String param = 'id'}) {
  return (Handler handler) => (Request request) async {
    final targetId = int.parse(request.params[param!]);
    final userId = getUserId(request);

    if (targetId != userId && !isAdmin(request)) {
      return Response(403, body: 'Access denied');
    }
    return handler(request);
  };
}
```

#### 4.3.3.8 Cheat Sheet: Route Protection

```
// Mit Pipeline (Gruppen)
final publicRouter = Router()
  ..get('/products', listProducts)
  ..get('/products/<id>', getProduct);

final protectedRouter = const Pipeline()
  .addMiddleware(authMiddleware(jwt))
  .addHandler(Router()
    ..get('/orders', listOrders)
    ..post('/orders', createOrder));

final adminRouter = const Pipeline()
  .addMiddleware(authMiddleware(jwt))
  .addMiddleware(requireAdmin())
  .addHandler(Router()
    ..get('/users', listUsers)
    ..delete('/users/<id>', deleteUser));

// Mit Cascade
final cascade = Cascade()
  .add(publicRouter)
  .add(protectedRouter)
  .add(adminRouter);
```

#### 4.3.3.9 Best Practices

##### DO

1. **401 vs 403 unterscheiden**
  - 401: Nicht authentifiziert (kein/ungültiger Token)
  - 403: Authentifiziert, aber keine Berechtigung
2. **Principle of Least Privilege**
  - Nur nötige Berechtigungen vergeben
  - Standardmäßig alles verbieten
3. **Defense in Depth**
  - Mehrere Sicherheitsebenen
  - Middleware + Handler-Prüfungen
4. **Generische Fehlermeldungen**
  - Keine Details über Berechtigungssystem
  - “Access denied” statt “Admin required”

##### DON'T

1. **Autorisierung nur im Frontend**
  - Backend muss immer prüfen
2. **Berechtigungen in JWT speichern (langlebig)**
  - Bei Änderungen bleiben alte Tokens gültig
3. **Hardcoded Rollen-Checks**
  - Nutze flexibles System mit Permissions
4. **Ungeprüfte URL-Parameter**
  - Immer validieren und autorisieren

#### 4.3.3.10 Häufige Patterns

```
// Optional Auth (öffentlich, aber mit User-Info wenn eingeloggt)
router.get('/products', (Request request) async {
  final userId = getUserId(request); // Kann null sein
  final products = await getProducts(userId: userId);
  return Response.ok(jsonEncode(products));
});

// Eigene Ressourcen filtern
router.get('/orders', (Request request) async {
  final userId = getUserId(request)!;
  final orders = await orderRepo.findById(userId);
  return Response.ok(jsonEncode(orders));
});

// Admin sieht alles, User nur eigenes
router.get('/orders', (Request request) async {
  final userId = getUserId(request)!;
  final role = getUserRole(request);

  final orders = role.hasAtLeast(Role.admin)
    ? await orderRepo.findAll()
    : await orderRepo.findById(userId);

  return Response.ok(jsonEncode(orders));
});
```

## 4.4 Einheit 8.4: OAuth 2.0 & Social Login

### 4.4.0.1 Lernziele

Nach dieser Einheit kannst du: - OAuth 2.0 Flows verstehen und implementieren - Google und GitHub Login integrieren - Token-Exchange durchführen - OAuth-Provider in deine API einbinden

### 4.4.0.2 Was ist OAuth 2.0?

**OAuth 2.0** ist ein Autorisierungsframework, das Drittanwendungen begrenzten Zugriff auf Benutzerressourcen ermöglicht, ohne Passwörter zu teilen.

Begriffe

Begriff	Beschreibung
<b>Resource Owner</b>	Der Benutzer
<b>Client</b>	Deine Anwendung
<b>Authorization Server</b>	Google, GitHub, etc.
<b>Resource Server</b>	API mit geschützten Daten
<b>Access Token</b>	Erlaubt Zugriff auf Ressourcen
<b>Refresh Token</b>	Zum Erneuern des Access Tokens

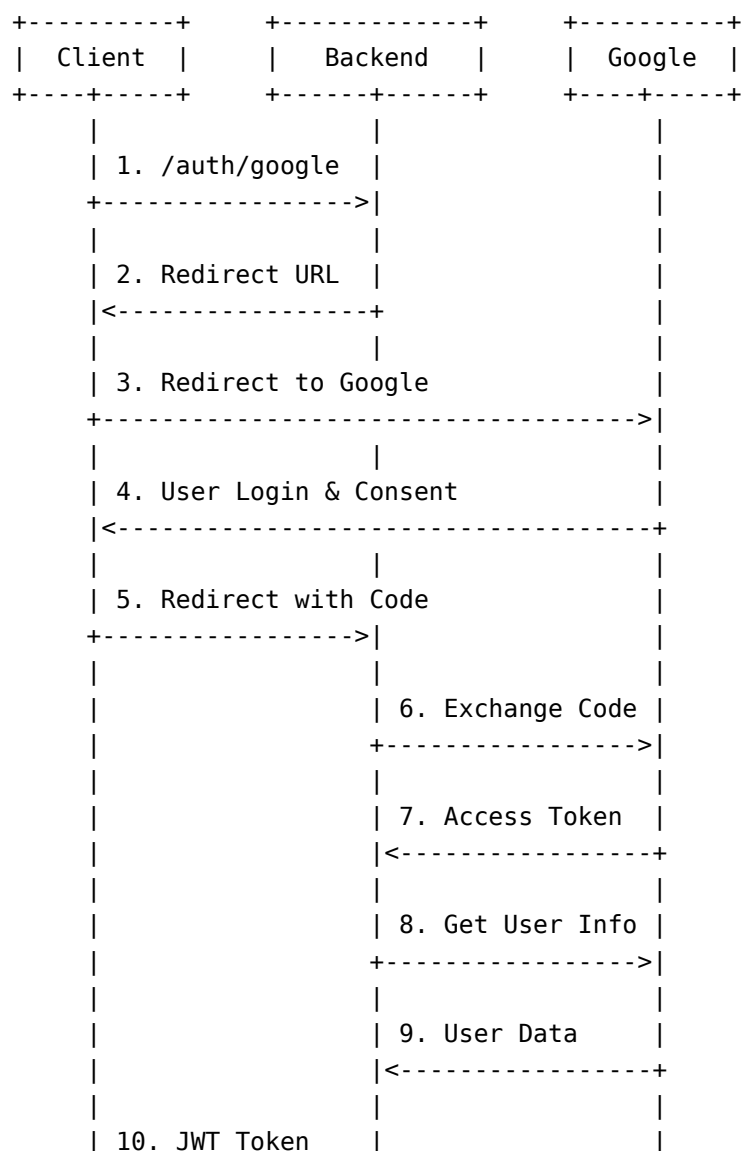
## OAuth vs. Eigene Auth

Eigene Auth	OAuth/Social Login
Volle Kontrolle	Delegiert an Provider
Passwort-Verwaltung nötig	Keine Passwörter
Mehr Entwicklungsaufwand	Schnelle Integration
Nur eigene Accounts	Millionen existierende Accounts

## 4.4.0.3 OAuth 2.0 Flows

Authorization Code Flow (empfohlen für Server)

1. User klickt "Login mit Google"
2. Redirect zu Google Authorization URL
3. User gibt Zustimmung bei Google
4. Google redirected zurück mit Authorization Code
5. Backend tauscht Code gegen Access Token
6. Backend lädt User-Info von Google
7. Backend erstellt/aktualisiert User und gibt eigenen JWT aus



|<-----+ |

#### 4.4.0.4 Google OAuth Setup

##### 1. Google Cloud Console

1. Gehe zu Google Cloud Console
2. Neues Projekt erstellen
3. APIs & Services -> OAuth consent screen
4. APIs & Services -> Credentials -> OAuth 2.0 Client IDs
5. Client ID und Client Secret notieren

##### 2. Konfiguration

```
// lib/config/oauth_config.dart

class OAuthConfig {
  final String googleClientId;
  final String googleClientSecret;
  final String googleRedirectUri;

  final String githubClientId;
  final String githubClientSecret;
  final String githubRedirectUri;

  OAuthConfig({
    required this.googleClientId,
    required this.googleClientSecret,
    required this.googleRedirectUri,
    required this.githubClientId,
    required this.githubClientSecret,
    required this.githubRedirectUri,
  });

  factory OAuthConfig.fromEnvironment() {
    return OAuthConfig(
      googleClientId: Platform.environment['GOOGLE_CLIENT_ID'] ?? '',
      googleClientSecret: Platform.environment['GOOGLE_CLIENT_SECRET'] ?? '',
      googleRedirectUri: Platform.environment['GOOGLE_REDIRECT_URI'] ??
        'http://localhost:8080/api/auth/google/callback',
      githubClientId: Platform.environment['GITHUB_CLIENT_ID'] ?? '',
      githubClientSecret: Platform.environment['GITHUB_CLIENT_SECRET'] ?? '',
      githubRedirectUri: Platform.environment['GITHUB_REDIRECT_URI'] ??
        'http://localhost:8080/api/auth/github/callback',
    );
  }
}
```

#### 4.4.0.5 Google OAuth Provider

```
// lib/services/oauth/google_oauth_provider.dart
import 'dart:convert';
```

```
import 'package:http/http.dart' as http;

class GoogleOAuthProvider {
  final String clientId;
  final String clientSecret;
  final String redirectUri;

  static const authorizationEndpoint =
    ↪ 'https://accounts.google.com/o/oauth2/v2/auth';
  static const tokenEndpoint = 'https://oauth2.googleapis.com/token';
  static const userInfoEndpoint =
    ↪ 'https://www.googleapis.com/oauth2/v2/userinfo';

  GoogleOAuthProvider({
    required this.clientId,
    required this.clientSecret,
    required this.redirectUri,
  });

  /// Generiere Authorization URL
  String getAuthorizationUrl({String? state}) {
    final params = {
      'client_id': clientId,
      'redirect_uri': redirectUri,
      'response_type': 'code',
      'scope': 'openid email profile',
      'access_type': 'offline', // Für Refresh Token
      'prompt': 'consent',
      if (state != null) 'state': state,
    };

    return Uri.parse(authorizationEndpoint)
      .replace(queryParameters: params)
      .toString();
  }

  /// Tausche Authorization Code gegen Access Token
  Future<OAuthTokens> exchangeCode(String code) async {
    final response = await http.post(
      Uri.parse(tokenEndpoint),
      headers: {'Content-Type': 'application/x-www-form-urlencoded'},
      body: {
        'client_id': clientId,
        'client_secret': clientSecret,
        'code': code,
        'grant_type': 'authorization_code',
        'redirect_uri': redirectUri,
      },
    );

    if (response.statusCode != 200) {
```

```

        throw OAuthException('Token exchange failed: ${response.body}');
    }

    final data = jsonDecode(response.body) as Map<String, dynamic>;

    return OAuthTokens(
      accessToken: data['access_token'] as String,
      refreshToken: data['refresh_token'] as String?,
      expiresIn: data['expires_in'] as int?,
      idToken: data['id_token'] as String?,
    );
  }

  /// Lade User-Info mit Access Token
  Future<OAuthUserInfo> getUserInfo(String accessToken) async {
    final response = await http.get(
      Uri.parse(userInfoEndpoint),
      headers: {'Authorization': 'Bearer $accessToken'},
    );

    if (response.statusCode != 200) {
      throw OAuthException('Failed to get user info: ${response.body}');
    }

    final data = jsonDecode(response.body) as Map<String, dynamic>;

    return OAuthUserInfo(
      id: data['id'] as String,
      email: data['email'] as String,
      name: data['name'] as String?,
      picture: data['picture'] as String?,
      provider: 'google',
    );
  }
}

```

#### 4.4.0.6 GitHub OAuth Provider

```

// lib/services/oauth/github_oauth_provider.dart
import 'dart:convert';
import 'package:http/http.dart' as http;

class GitHubOAuthProvider {
  final String clientId;
  final String clientSecret;
  final String redirectUri;

  static const authorizationEndpoint =
    ↪ 'https://github.com/login/oauth/authorize';
  ↪ static const tokenEndpoint = 'https://github.com/login/oauth/access_token';

```

```
static const userInfoEndpoint = 'https://api.github.com/user';
static const emailEndpoint = 'https://api.github.com/user/emails';

GitHubOAuthProvider({
  required this.clientId,
  required this.clientSecret,
  required this.redirectUri,
});

String getAuthorizationUrl({String? state}) {
  final params = {
    'client_id': clientId,
    'redirect_uri': redirectUri,
    'scope': 'read:user user:email',
    if (state != null) 'state': state,
  };

  return Uri.parse(authorizationEndpoint)
    .replace(queryParameters: params)
    .toString();
}

Future<OAuthTokens> exchangeCode(String code) async {
  final response = await http.post(
    Uri.parse(tokenEndpoint),
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
      'Accept': 'application/json',
    },
    body: {
      'client_id': clientId,
      'client_secret': clientSecret,
      'code': code,
      'redirect_uri': redirectUri,
    },
  );

  if (response.statusCode != 200) {
    throw OAuthException('Token exchange failed: ${response.body}');
  }

  final data = jsonDecode(response.body) as Map<String, dynamic>;

  if (data.containsKey('error')) {
    throw OAuthException('OAuth error: ${data['error_description']}');
  }

  return OAuthTokens(
    accessToken: data['access_token'] as String,
    refreshToken: null, // GitHub doesn't provide refresh tokens
    expiresIn: null,
```

```

    );
}

Future<OAuthUserInfo> getUserInfo(String accessToken) async {
    // User-Info laden
    final userResponse = await http.get(
        Uri.parse(userInfoEndpoint),
        headers: {
            'Authorization': 'Bearer $accessToken',
            'Accept': 'application/vnd.github.v3+json',
        },
    );

    if (userResponse.statusCode != 200) {
        throw OAuthException('Failed to get user info');
    }

    final userData = jsonDecode(userResponse.body) as Map<String, dynamic>;

    // Email kann in User-Info null sein - dann separat laden
    String? email = userData['email'] as String?;

    if (email == null) {
        final emailResponse = await http.get(
            Uri.parse(emailEndpoint),
            headers: {
                'Authorization': 'Bearer $accessToken',
                'Accept': 'application/vnd.github.v3+json',
            },
        );

        if (emailResponse.statusCode == 200) {
            final emails = jsonDecode(emailResponse.body) as List;
            final primaryEmail = emails.firstWhere(
                (e) => e['primary'] == true,
                orElse: () => emails.isNotEmpty ? emails.first : null,
            );
            email = primaryEmail?['email'] as String?;
        }
    }

    return OAuthUserInfo(
        id: userData['id'].toString(),
        email: email ?? '',
        name: userData['name'] as String? ?? userData['login'] as String?,
        picture: userData['avatar_url'] as String?,
        provider: 'github',
    );
}
}

```

#### 4.4.0.7 OAuth Models

```
// lib/models/oauth_models.dart

class OAuthTokens {
  final String accessToken;
  final String? refreshToken;
  final int? expiresIn;
  final String? idToken;

  OAuthTokens({
    required this.accessToken,
    this.refreshToken,
    this.expiresIn,
    this.idToken,
  });
}

class OAuthUserInfo {
  final String id;
  final String email;
  final String? name;
  final String? picture;
  final String provider;

  OAuthUserInfo({
    required this.id,
    required this.email,
    this.name,
    this.picture,
    required this.provider,
  });
}

class OAuthException implements Exception {
  final String message;
  OAuthException(this.message);

  @override
  String toString() => message;
}
```

#### 4.4.0.8 OAuth Service

```
// lib/services/oauth_service.dart

class OAuthService {
  final GoogleOAuthProvider _google;
  final GitHubOAuthProvider _github;
  final UserRepository _userRepo;
  final OAuthAccountRepository _oauthRepo;
```

```
final JwtService _jwtService;

OAuthService({
  required GoogleOAuthProvider google,
  required GitHubOAuthProvider github,
  required UserRepository userRepo,
  required OAuthAccountRepository oauthRepo,
  required JwtService jwtService,
}) : _google = google,
    _github = github,
    _userRepo = userRepo,
    _oauthRepo = oauthRepo,
    _jwtService = jwtService;

/// Generiere State-Token für CSRF-Schutz
String generateState() {
  final random = Random.secure();
  final values = List<int>.generate(32, (_) => random.nextInt(256));
  return base64Url.encode(values);
}

/// Authorization URL für Provider
String getAuthorizationUrl(String provider, String state) {
  switch (provider) {
    case 'google':
      return _google.getAuthorizationUrl(state: state);
    case 'github':
      return _github.getAuthorizationUrl(state: state);
    default:
      throw OAuthException('Unknown provider: $provider');
  }
}

/// OAuth Callback verarbeiten
Future<TokenPair> handleCallback({
  required String provider,
  required String code,
}) async {
  // 1. Code gegen Token tauschen
  OAuthTokens tokens;
  OAuthUserInfo userInfo;

  switch (provider) {
    case 'google':
      tokens = await _google.exchangeCode(code);
      userInfo = await _google.getUserInfo(tokens.accessToken);
      break;
    case 'github':
      tokens = await _github.exchangeCode(code);
      userInfo = await _github.getUserInfo(tokens.accessToken);
      break;
  }
}
```

```
        default:
            throw OAuthException('Unknown provider: $provider');
    }

    // 2. User finden oder erstellen
    final user = await _findOrCreateUser(userInfo);

    // 3. OAuth-Account verknüpfen
    await _linkOAuthAccount(user.id!, userInfo, tokens);

    // 4. Eigenen JWT generieren
    return _jwtService.generateTokenPair(user);
}

Future<User> _findOrCreateUser(OAuthUserInfo info) async {
    // Prüfen ob OAuth-Account existiert
    final oauthAccount = await _oauthRepo.findByProviderAndId(
        info.provider,
        info.id,
    );

    if (oauthAccount != null) {
        // User existiert
        final user = await _userRepo.findById(oauthAccount.userId);
        if (user != null) return user;
    }

    // Prüfen ob Email schon existiert
    var user = await _userRepo.findByEmail(info.email);

    if (user != null) {
        return user;
    }

    // Neuen User erstellen
    user = User(
        email: info.email,
        passwordHash: '', // Kein Passwort bei OAuth
        name: info.name,
        isActive: true,
    );

    return await _userRepo.create(user);
}

Future<void> _linkOAuthAccount(
    int userId,
    OAuthUserInfo info,
    OAuthTokens tokens,
) async {
    final existing = await _oauthRepo.findByProviderAndId(
```

```

        info.provider,
        info.id,
    );

    if (existing != null) {
        // Update tokens
        await _oauthRepo.updateTokens(
            existing.id!,
            tokens.accessToken,
            tokens.refreshToken,
        );
    } else {
        // Neue Verknüpfung
        await _oauthRepo.create(OAuthAccount(
            userId: userId,
            provider: info.provider,
            providerUserId: info.id,
            accessToken: tokens.accessToken,
            refreshToken: tokens.refreshToken,
        ));
    }
}
}
}

```

#### 4.4.0.9 OAuth Handler

```

// lib/handlers/oauth_handler.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';

class OAuthHandler {
    final OAuthService _oauthService;
    final Map<String, String> _stateStore = {}; // In Produktion: Redis

    OAuthHandler(this._oauthService);

    Router get router {
        final router = Router();

        // Initiiere OAuth Flow
        router.get('/<provider>', _initiateOAuth);

        // Callback von Provider
        router.get('/<provider>/callback', _handleCallback);

        return router;
    }

    Future<Response> _initiateOAuth(Request request) async {

```

```
final provider = request.params['provider']!;

// State generieren und speichern
final state = _oauthService.generateState();
_stateStore[state] = DateTime.now().toIso8601String();

// Redirect URL generieren
final authUrl = _oauthService.getAuthorizationUrl(provider, state);

// Redirect zum Provider
return Response.found(authUrl);
}

Future<Response> _handleCallback(Request request) async {
  final provider = request.params['provider']!;
  final code = request.url.queryParameters['code'];
  final state = request.url.queryParameters['state'];
  final error = request.url.queryParameters['error'];

  // Error vom Provider
  if (error != null) {
    return _errorRedirect('OAuth error: $error');
  }

  // Code fehlt
  if (code == null) {
    return _errorRedirect('No authorization code received');
  }

  // State validieren (CSRF-Schutz)
  if (state == null || !_stateStore.containsKey(state)) {
    return _errorRedirect('Invalid state parameter');
  }
  _stateStore.remove(state);

  try {
    // OAuth Flow abschließen
    final tokens = await _oauthService.handleCallback(
      provider: provider,
      code: code,
    );

    // Redirect mit Tokens (oder zu Frontend mit Tokens)
    // Option 1: JSON Response
    return Response.ok(
      jsonEncode({
        'message': 'Login successful',
        ...tokens.toJson(),
      }),
      headers: {'content-type': 'application/json'},
    );
  }
```

```

        // Option 2: Redirect zum Frontend mit Token als Fragment
        // return Response.found(
        //   'https://myapp.com/auth/callback#access_token=${tokens.accessToken}',
        // );
      } on OAuthException catch (e) {
        return _errorRedirect(e.message);
      } catch (e) {
        return _errorRedirect('OAuth failed');
      }
    }
  }

  Response _errorRedirect(String message) {
    return Response(
      400,
      body: jsonEncode({'error': message}),
      headers: {'content-type': 'application/json'},
    );
  }
}

```

#### 4.4.0.10 OAuth Account Repository

```

// lib/repositories/oauth_account_repository.dart

class OAuthAccount {
  final int? id;
  final int userId;
  final String provider;
  final String providerUserId;
  final String accessToken;
  final String? refreshToken;
  final DateTime createdAt;

  OAuthAccount({
    this.id,
    required this.userId,
    required this.provider,
    required this.providerUserId,
    required this.accessToken,
    this.refreshToken,
    DateTime? createdAt,
  }) : createdAt = createdAt ?? DateTime.now();
}

class OAuthAccountRepository {
  final Connection _db;

  OAuthAccountRepository(this._db);
}

```

```

Future<OAuthAccount?> findByProviderAndId(String provider, String id) async {
    final result = await _db.execute(
        Sql.named('''
            SELECT * FROM oauth_accounts
            WHERE provider = @provider AND provider_user_id = @id
        '''),
        parameters: {'provider': provider, 'id': id},
    );

    if (result.isEmpty) return null;
    return _mapToAccount(result.first.toColumnMap());
}

Future<OAuthAccount> create(OAuthAccount account) async {
    final result = await _db.execute(
        Sql.named('''
            INSERT INTO oauth_accounts
            (user_id, provider, provider_user_id, access_token, refresh_token)
            VALUES
            (@userId, @provider, @providerId, @accessToken, @refreshToken)
            RETURNING *
        '''),
        parameters: {
            'userId': account.userId,
            'provider': account.provider,
            'providerId': account.providerUserId,
            'accessToken': account.accessToken,
            'refreshToken': account.refreshToken,
        },
    );

    return _mapToAccount(result.first.toColumnMap());
}

Future<void> updateTokens(int id, String accessToken, String? refreshToken) ↵
↵ async {
    await _db.execute(
        Sql.named('''
            UPDATE oauth_accounts
            SET access_token = @accessToken, refresh_token = @refreshToken
            WHERE id = @id
        '''),
        parameters: {
            'id': id,
            'accessToken': accessToken,
            'refreshToken': refreshToken,
        },
    );
}

OAuthAccount _mapToAccount(Map<String, dynamic> row) {

```

```
return OAuthAccount(
    id: row['id'] as int,
    userId: row['user_id'] as int,
    provider: row['provider'] as String,
    providerUserId: row['provider_user_id'] as String,
    accessToken: row['access_token'] as String,
    refreshToken: row['refresh_token'] as String?,
    createdAt: row['created_at'] as DateTime,
);
}
```

#### 4.4.0.11 Datenbank-Schema

```
-- migrations/003_create_oauth_accounts.sql

CREATE TABLE oauth_accounts (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  provider VARCHAR(50) NOT NULL, -- 'google', 'github', etc.
  provider_user_id VARCHAR(255) NOT NULL,
  access_token TEXT NOT NULL,
  refresh_token TEXT,
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMP,

  UNIQUE(provider, provider_user_id)
);

CREATE INDEX idx_oauth_accounts_user ON oauth_accounts(user_id);
CREATE INDEX idx_oauth_accounts_provider ON oauth_accounts(provider,
↪ provider_user_id);
```

#### 4.4.0.12 Zusammenfassung

Konzept	Beschreibung
<b>OAuth 2.0</b>	Autorisierungsframework für Drittanbieter-Login
<b>Authorization Code Flow</b>	Sicherer Flow für Server-Anwendungen
<b>State Parameter</b>	CSRF-Schutz bei OAuth
<b>Token Exchange</b>	Code gegen Access Token tauschen
<b>Provider</b>	Google, GitHub, Facebook, etc.

### 4.4.1 Übung

#### 4.4.1.1 Ziel

Implementiere Google und GitHub Login für deine API.

#### 4.4.1.2 Vorbereitung

Dependencies

```
dependencies:
  http: ^1.1.0
  # ... bestehende Dependencies
```

Google OAuth Setup

1. Gehe zu Google Cloud Console
2. Erstelle ein Projekt
3. Aktiviere "Google+ API"
4. Erstelle OAuth 2.0 Client ID (Web application)
5. Füge `http://localhost:8080/api/auth/google/callback` als Redirect URI hinzu
6. Notiere Client ID und Client Secret

GitHub OAuth Setup

1. Gehe zu GitHub -> Settings -> Developer settings -> OAuth Apps
2. Erstelle eine neue OAuth App
3. Callback URL: `http://localhost:8080/api/auth/github/callback`
4. Notiere Client ID und Client Secret

Environment Variables

```
export GOOGLE_CLIENT_ID="your-google-client-id"
export GOOGLE_CLIENT_SECRET="your-google-client-secret"
export GITHUB_CLIENT_ID="your-github-client-id"
export GITHUB_CLIENT_SECRET="your-github-client-secret"
```

Datenbank

```
CREATE TABLE oauth_accounts (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  provider VARCHAR(50) NOT NULL,
  provider_user_id VARCHAR(255) NOT NULL,
  access_token TEXT NOT NULL,
  refresh_token TEXT,
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  UNIQUE(provider, provider_user_id)
);
```

#### 4.4.1.3 Aufgabe 1: OAuth Models (10 min)

```
// lib/models/oauth_models.dart

class OAuthTokens {
  final String accessToken;
  final String? refreshToken;
  final int? expiresIn;
  final String? idToken;

  OAuthTokens({
```

```
        required this.accessToken,
        this.refreshToken,
        this.expiresIn,
        this.idToken,
    });

    // TODO: Factory für JSON-Parsing
}

class OAuthUserInfo {
    final String id;
    final String email;
    final String? name;
    final String? picture;
    final String provider;

    OAuthUserInfo({
        required this.id,
        required this.email,
        this.name,
        this.picture,
        required this.provider,
    });
}

class OAuthException implements Exception {
    final String message;
    OAuthException(this.message);
}
```

#### 4.4.1.4 Aufgabe 2: OAuth Provider Interface (10 min)

```
// lib/services/oauth/oauth_provider.dart

abstract class OAuthProvider {
    /// Name des Providers ('google', 'github')
    String get name;

    /// Generiere Authorization URL
    String getAuthorizationUrl({String? state});

    /// Tausche Code gegen Tokens
    Future<OAuthTokens> exchangeCode(String code);

    /// Lade User-Info
    Future<OAuthUserInfo> getUserInfo(String accessToken);
}
```

**4.4.1.5 Aufgabe 3: Google OAuth Provider (20 min)**

```
// lib/services/oauth/google_oauth_provider.dart

class GoogleOAuthProvider implements OAuthProvider {
  final String clientId;
  final String clientSecret;
  final String redirectUri;

  // Endpoints
  static const authEndpoint = 'https://accounts.google.com/o/oauth2/v2/auth';
  static const tokenEndpoint = 'https://oauth2.googleapis.com/token';
  static const userInfoEndpoint =
    ↪ 'https://www.googleapis.com/oauth2/v2/userinfo';

  GoogleOAuthProvider({
    required this.clientId,
    required this.clientSecret,
    required this.redirectUri,
  });

  @override
  String get name => 'google';

  @override
  String getAuthorizationUrl({String? state}) {
    // TODO:
    // Query-Parameter: client_id, redirect_uri, response_type=code,
    // scope=openid email profile, access_type=offline, state
  }

  @override
  Future<OAuthTokens> exchangeCode(String code) async {
    // TODO:
    // POST zu tokenEndpoint mit:
    // client_id, client_secret, code, grant_type=authorization_code,
    ↪ redirect_uri
    // Response parsen und OAuthTokens zurückgeben
  }

  @override
  Future<OAuthUserInfo> getUserInfo(String accessToken) async {
    // TODO:
    // GET zu userInfoEndpoint mit Authorization: Bearer header
    // Response parsen und OAuthUserInfo zurückgeben
  }
}
```

**4.4.1.6 Aufgabe 4: GitHub OAuth Provider (20 min)**

```
// lib/services/oauth/github_oauth_provider.dart

class GitHubOAuthProvider implements OAuthProvider {
  final String clientId;
  final String clientSecret;
  final String redirectUri;

  static const authEndpoint = 'https://github.com/login/oauth/authorize';
  static const tokenEndpoint = 'https://github.com/login/oauth/access_token';
  static const userInfoEndpoint = 'https://api.github.com/user';
  static const emailEndpoint = 'https://api.github.com/user/emails';

  GitHubOAuthProvider({
    required this.clientId,
    required this.clientSecret,
    required this.redirectUri,
  });

  @override
  String get name => 'github';

  @override
  String getAuthorizationUrl({String? state}) {
    // TODO: scope=read:user user:email
  }

  @override
  Future<OAuthTokens> exchangeCode(String code) async {
    // TODO: Header Accept: application/json setzen!
  }

  @override
  Future<OAuthUserInfo> getUserInfo(String accessToken) async {
    // TODO:
    // 1. User-Info laden
    // 2. Falls email null: separaten Email-Endpoint aufrufen
    // 3. Primary Email finden
  }
}
```

**4.4.1.7 Aufgabe 5: OAuth Account Repository (15 min)**

```
// lib/repositories/oauth_account_repository.dart

class OAuthAccount {
  final int? id;
  final int userId;
  final String provider;
  final String providerUserId;
}
```

```

final String accessToken;
final String? refreshToken;
final DateTime createdAt;

OAuthAccount({
    this.id,
    required this.userId,
    required this.provider,
    required this.providerUserId,
    required this.accessToken,
    this.refreshToken,
    DateTime? createdAt,
}) : createdAt = createdAt ?? DateTime.now();
}

class OAuthAccountRepository {
    final Connection _db;

    OAuthAccountRepository(this._db);

    /// Finde Account nach Provider und Provider-User-ID
    Future<OAuthAccount?> findByProviderAndId(String provider, String id) async {
        // TODO
    }

    /// Erstelle neuen OAuth-Account
    Future<OAuthAccount> create(OAuthAccount account) async {
        // TODO
    }

    /// Update Tokens
    Future<void> updateTokens(int id, String accessToken, String? refreshToken) ↵
    ↵ async {
        // TODO
    }

    /// Finde alle Accounts eines Users
    Future<List<OAuthAccount>> findByUserId(int userId) async {
        // TODO
    }

    /// Lösche Account
    Future<void> delete(int id) async {
        // TODO
    }
}

```

#### 4.4.1.8 Aufgabe 6: OAuth Service (25 min)

```
// lib/services/oauth_service.dart

class OAuthService {
  final Map<String, OAuthProvider> _providers = {};
  final UserRepository _userRepo;
  final OAuthAccountRepository _oauthRepo;
  final JwtService _jwtService;

  OAuthService({
    required UserRepository userRepo,
    required OAuthAccountRepository oauthRepo,
    required JwtService jwtService,
  }) : _userRepo = userRepo,
       _oauthRepo = oauthRepo,
       _jwtService = jwtService;

  void registerProvider(OAuthProvider provider) {
    _providers[provider.name] = provider;
  }

  OAuthProvider getProvider(String name) {
    final provider = _providers[name];
    if (provider == null) {
      throw OAuthException('Unknown provider: $name');
    }
    return provider;
  }

  /// Generiere State für CSRF-Schutz
  String generateState() {
    // TODO: Zufälligen Base64-String generieren
  }

  /// Authorization URL für Provider
  String getAuthorizationUrl(String provider, String state) {
    return getProvider(provider).getAuthorizationUrl(state: state);
  }

  /// OAuth Callback verarbeiten
  Future<TokenPair> handleCallback({
    required String provider,
    required String code,
  }) async {
    // TODO:
    // 1. Provider holen
    // 2. Code gegen Token tauschen
    // 3. User-Info laden
    // 4. User finden oder erstellen
    // 5. OAuth-Account verknüpfen/updaten
    // 6. JWT generieren
  }
}
```

```
/// User finden oder erstellen
Future<User> _findOrCreateUser(OAuthUserInfo info) async {
  // TODO:
  // 1. Prüfen ob OAuth-Account existiert -> User laden
  // 2. Prüfen ob Email existiert -> User zurückgeben
  // 3. Neuen User erstellen
}

/// OAuth-Account mit User verknüpfen
Future<void> _linkOAuthAccount(
  int userId,
  OAuthUserInfo info,
  OAuthTokens tokens,
) async {
  // TODO:
  // Existierenden Account updaten oder neuen erstellen
}
}
```

#### 4.4.1.9 Aufgabe 7: OAuth Handler (15 min)

```
// lib/handlers/oauth_handler.dart

class OAuthHandler {
  final OAuthService _oauthService;

  // State-Store (in Produktion: Redis mit TTL)
  final Map<String, DateTime> _stateStore = {};
  final Duration _stateTimeout = const Duration(minutes: 10);

  OAuthHandler(this._oauthService);

  Router get router {
    final router = Router();

    // GET /auth/oauth/:provider - Initiiere OAuth
    router.get('/<provider>', _initiateOAuth);

    // GET /auth/oauth/:provider/callback - Callback
    router.get('/<provider>/callback', _handleCallback);

    return router;
  }

  /// Initiiere OAuth-Flow
  Future<Response> _initiateOAuth(Request request) async {
    // TODO:
    // 1. Provider aus URL-Params
    // 2. State generieren und speichern
  }
}
```

```

    // 3. Authorization URL generieren
    // 4. Redirect Response zurückgeben
  }

  /// Verarbeite Callback
  Future<Response> _handleCallback(Request request) async {
    // TODO:
    // 1. Provider, code, state, error aus Query-Params
    // 2. Error prüfen
    // 3. State validieren
    // 4. OAuth-Flow mit Service durchführen
    // 5. Token-Response oder Error zurückgeben
  }

  /// State validieren und entfernen
  bool _validateState(String state) {
    final timestamp = _stateStore.remove(state);
    if (timestamp == null) return false;
    return DateTime.now().difference(timestamp) < _stateTimeout;
  }
}

```

#### 4.4.1.10 Aufgabe 8: Integration (10 min)

```

// bin/server.dart (Erweiterung)

// OAuth-Provider registrieren
final oauthService = OAuthService(
  userRepo: userRepo,
  oauthRepo: oauthAccountRepo,
  jwtService: jwtService,
);

oauthService.registerProvider(GoogleOAuthProvider(
  clientId: config.googleClientId,
  clientSecret: config.googleClientSecret,
  redirectUri: config.googleRedirectUri,
));

oauthService.registerProvider(GitHubOAuthProvider(
  clientId: config.githubClientId,
  clientSecret: config.githubClientSecret,
  redirectUri: config.githubRedirectUri,
));

// Handler
final oauthHandler = OAuthHandler(oauthService);

// Router erweitern
router.mount('/api/auth/oauth', oauthHandler.router);

```

#### 4.4.1.11 Testen

Google Login starten

```
# Browser öffnet sich automatisch
open "http://localhost:8080/api/auth/oauth/google"

# Oder manuell
curl -v http://localhost:8080/api/auth/oauth/google
# Folge dem Location-Header zu Google
```

GitHub Login starten

```
open "http://localhost:8080/api/auth/oauth/github"
```

Nach erfolgreichem Login

```
{
  "message": "Login successful",
  "access_token": "eyJhbGciOi...",
  "refresh_token": "eyJhbGciOi...",
  "expires_in": 900,
  "token_type": "Bearer"
}
```

#### 4.4.1.12 Bonus: Account Linking (Optional)

Erlaube authentifizierten Usern, weitere OAuth-Accounts zu verknüpfen:

```
// POST /api/users/me/oauth/:provider/link
Future<Response> linkOAuthAccount(Request request) async {
  final userId = getUserId(request)!;
  final provider = request.params['provider']!;

  // State generieren und zu Provider redirecten
  // Bei Callback: Account mit bestehendem User verknüpfen
}

// DELETE /api/users/me/oauth/:provider
Future<Response> unlinkOAuthAccount(Request request) async {
  final userId = getUserId(request)!;
  final provider = request.params['provider']!;

  await _oauthRepo.deleteByUserAndProvider(userId, provider);
  return Response(204);
}
```

#### 4.4.1.13 Abgabe-Checkliste

- ☐ OAuthTokens und OAuthUserInfo Models
- ☐ GoogleOAuthProvider mit allen Methoden
- ☐ GitHubOAuthProvider mit Email-Fallback
- ☐ OAuthAccountRepository
- ☐ OAuthService mit handleCallback

- ☐ State-Generierung für CSRF-Schutz
- ☐ User erstellen oder finden bei OAuth-Login
- ☐ OAuth-Account mit User verknüpfen
- ☐ OAuthHandler mit /oauth/:provider Endpoints
- ☐ State-Validierung mit Timeout
- ☐ JWT-Response nach erfolgreichem OAuth

## 4.4.2 Lösung

### 4.4.2.1 OAuth Models

```
// lib/models/oauth_models.dart

class OAuthTokens {
  final String accessToken;
  final String? refreshToken;
  final int? expiresIn;
  final String? idToken;

  OAuthTokens({
    required this.accessToken,
    this.refreshToken,
    this.expiresIn,
    this.idToken,
  });

  factory OAuthTokens.fromJson(Map<String, dynamic> json) {
    return OAuthTokens(
      accessToken: json['access_token'] as String,
      refreshToken: json['refresh_token'] as String?,
      expiresIn: json['expires_in'] as int?,
      idToken: json['id_token'] as String?,
    );
  }
}

class OAuthUserInfo {
  final String id;
  final String email;
  final String? name;
  final String? picture;
  final String provider;

  OAuthUserInfo({
    required this.id,
    required this.email,
    this.name,
    this.picture,
    required this.provider,
  });
}
```

```

@override
String toString() =>
    'OAuthUserInfo(id: $id, email: $email, name: $name, provider: $provider)';
}

class OAuthException implements Exception {
    final String message;
    final int? statusCode;

    OAuthException(this.message, {this.statusCode});

    @override
    String toString() => 'OAuthException: $message';
}

```

#### 4.4.2.2 OAuth Provider Interface

```

// lib/services/oauth/oauth_provider.dart

abstract class OAuthProvider {
    String get name;
    String getAuthorizationUrl({String? state});
    Future<OAuthTokens> exchangeCode(String code);
    Future<OAuthUserInfo> getUserInfo(String accessToken);
}

```

#### 4.4.2.3 Google OAuth Provider

```

// lib/services/oauth/google_oauth_provider.dart
import 'dart:convert';
import 'package:http/http.dart' as http;
import '../models/oauth_models.dart';
import 'oauth_provider.dart';

class GoogleOAuthProvider implements OAuthProvider {
    final String clientId;
    final String clientSecret;
    final String redirectUri;

    static const authEndpoint = 'https://accounts.google.com/o/oauth2/v2/auth';
    static const tokenEndpoint = 'https://oauth2.googleapis.com/token';
    static const userInfoEndpoint =
↵ 'https://www.googleapis.com/oauth2/v2/userinfo';

    GoogleOAuthProvider({
        required this.clientId,
        required this.clientSecret,
        required this.redirectUri,
    });
}

```

```
@override
String get name => 'google';

@override
String getAuthorizationUrl({String? state}) {
  final params = <String, String>{
    'client_id': clientId,
    'redirect_uri': redirectUri,
    'response_type': 'code',
    'scope': 'openid email profile',
    'access_type': 'offline',
    'prompt': 'consent',
  };

  if (state != null) {
    params['state'] = state;
  }

  return Uri.parse(authEndpoint)
    .replace(queryParameters: params)
    .toString();
}

@override
Future<OAuthTokens> exchangeCode(String code) async {
  final response = await http.post(
    Uri.parse(tokenEndpoint),
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
    },
    body: {
      'client_id': clientId,
      'client_secret': clientSecret,
      'code': code,
      'grant_type': 'authorization_code',
      'redirect_uri': redirectUri,
    },
  );

  if (response.statusCode != 200) {
    final error = _parseError(response.body);
    throw OAuthException(
      'Google token exchange failed: $error',
      statusCode: response.statusCode,
    );
  }

  final data = jsonDecode(response.body) as Map<String, dynamic>;
  return OAuthTokens.fromJson(data);
}
```

```

@override
Future<OAuthUserInfo> getUserInfo(String accessToken) async {
  final response = await http.get(
    Uri.parse(userInfoEndpoint),
    headers: {
      'Authorization': 'Bearer $accessToken',
    },
  );

  if (response.statusCode != 200) {
    throw OAuthException(
      'Failed to get Google user info',
      statusCode: response.statusCode,
    );
  }

  final data = jsonDecode(response.body) as Map<String, dynamic>;

  return OAuthUserInfo(
    id: data['id'] as String,
    email: data['email'] as String,
    name: data['name'] as String?,
    picture: data['picture'] as String?,
    provider: name,
  );
}

String _parseError(String body) {
  try {
    final data = jsonDecode(body) as Map<String, dynamic>;
    return data['error_description'] as String? ??
      data['error'] as String? ??
      body;
  } catch (e) {
    return body;
  }
}
}

```

#### 4.4.2.4 GitHub OAuth Provider

```

// lib/services/oauth/github_oauth_provider.dart
import 'dart:convert';
import 'package:http/http.dart' as http;
import '../models/oauth_models.dart';
import 'oauth_provider.dart';

class GitHubOAuthProvider implements OAuthProvider {
  final String clientId;

```

```
final String clientSecret;
final String redirectUri;

static const authEndpoint = 'https://github.com/login/oauth/authorize';
static const tokenEndpoint = 'https://github.com/login/oauth/access_token';
static const userInfoEndpoint = 'https://api.github.com/user';
static const emailEndpoint = 'https://api.github.com/user/emails';

GitHubOAuthProvider({
  required this.clientId,
  required this.clientSecret,
  required this.redirectUri,
});

@override
String get name => 'github';

@override
String getAuthorizationUrl({String? state}) {
  final params = <String, String>{
    'client_id': clientId,
    'redirect_uri': redirectUri,
    'scope': 'read:user user:email',
  };

  if (state != null) {
    params['state'] = state;
  }

  return Uri.parse(authEndpoint)
    .replace(queryParameters: params)
    .toString();
}

@override
Future<OAuthTokens> exchangeCode(String code) async {
  final response = await http.post(
    Uri.parse(tokenEndpoint),
    headers: {
      'Content-Type': 'application/x-www-form-urlencoded',
      'Accept': 'application/json', // Wichtig für JSON-Response
    },
    body: {
      'client_id': clientId,
      'client_secret': clientSecret,
      'code': code,
      'redirect_uri': redirectUri,
    },
  );

  if (response.statusCode != 200) {
```

```
        throw OAuthException(
            'GitHub token exchange failed',
            statusCode: response.statusCode,
        );
    }

    final data = jsonDecode(response.body) as Map<String, dynamic>;

    if (data.containsKey('error')) {
        throw OAuthException(
            data['error_description'] as String? ?? data['error'] as String,
        );
    }

    return OAuthTokens(
        accessToken: data['access_token'] as String,
        refreshToken: null, // GitHub provides no refresh tokens
        expiresIn: null,
    );
}

@override
Future<OAuthUserInfo> getUserInfo(String accessToken) async {
    // 1. User-Info laden
    final userResponse = await http.get(
        Uri.parse(userInfoEndpoint),
        headers: {
            'Authorization': 'Bearer $accessToken',
            'Accept': 'application/vnd.github.v3+json',
        },
    );

    if (userResponse.statusCode != 200) {
        throw OAuthException(
            'Failed to get GitHub user info',
            statusCode: userResponse.statusCode,
        );
    }

    final userData = jsonDecode(userResponse.body) as Map<String, dynamic>;

    // 2. Email (kann in User-Daten null sein)
    String? email = userData['email'] as String?;

    if (email == null || email.isEmpty) {
        email = await _getPrimaryEmail(accessToken);
    }

    if (email == null || email.isEmpty) {
        throw OAuthException('Could not retrieve email from GitHub');
    }
}
```

```
return OAuthUserInfo(
    id: userData['id'].toString(),
    email: email,
    name: userData['name'] as String? ?? userData['login'] as String?,
    picture: userData['avatar_url'] as String?,
    provider: name,
);
}

Future<String?> _getPrimaryEmail(String accessToken) async {
    final response = await http.get(
        Uri.parse(emailEndpoint),
        headers: {
            'Authorization': 'Bearer $accessToken',
            'Accept': 'application/vnd.github.v3+json',
        },
    );

    if (response.statusCode != 200) {
        return null;
    }

    final emails = jsonDecode(response.body) as List<dynamic>;

    // Primäre Email suchen
    for (final email in emails) {
        if (email['primary'] == true && email['verified'] == true) {
            return email['email'] as String;
        }
    }

    // Fallback: Erste verifizierte Email
    for (final email in emails) {
        if (email['verified'] == true) {
            return email['email'] as String;
        }
    }

    // Fallback: Erste Email
    if (emails.isNotEmpty) {
        return emails.first['email'] as String;
    }

    return null;
}
}
```

## 4.4.2.5 OAuth Account Repository

```
// lib/repositories/oauth_account_repository.dart
import 'package:postgres/postgres.dart';

class OAuthAccount {
  final int? id;
  final int userId;
  final String provider;
  final String providerUserId;
  final String accessToken;
  final String? refreshToken;
  final DateTime createdAt;
  final DateTime? updatedAt;

  OAuthAccount({
    this.id,
    required this.userId,
    required this.provider,
    required this.providerUserId,
    required this.accessToken,
    this.refreshToken,
    DateTime? createdAt,
    this.updatedAt,
  }) : createdAt = createdAt ?? DateTime.now();

  factory OAuthAccount.fromRow(Map<String, dynamic> row) {
    return OAuthAccount(
      id: row['id'] as int,
      userId: row['user_id'] as int,
      provider: row['provider'] as String,
      providerUserId: row['provider_user_id'] as String,
      accessToken: row['access_token'] as String,
      refreshToken: row['refresh_token'] as String?,
      createdAt: row['created_at'] as DateTime,
      updatedAt: row['updated_at'] as DateTime?,
    );
  }
}

class OAuthAccountRepository {
  final Connection _db;

  OAuthAccountRepository(this._db);

  Future<OAuthAccount?> findByProviderAndId(String provider, String
↵ providerUserId) async {
  ↵ final result = await _db.execute(
    Sql.named(''
      SELECT * FROM oauth_accounts
      WHERE provider = @provider AND provider_user_id = @providerUserId
    ''
  );
}
```

```

        '''),
        parameters: {
            'provider': provider,
            'providerUserId': providerUserId,
        },
    );

    if (result.isEmpty) return null;
    return OAuthAccount.fromRow(result.first.toColumnMap());
}

Future<OAuthAccount> create(OAuthAccount account) async {
    final result = await _db.execute(
        Sql.named('''
            INSERT INTO oauth_accounts
              (user_id, provider, provider_user_id, access_token, refresh_token)
            VALUES
              (@userId, @provider, @providerUserId, @accessToken, @refreshToken)
            RETURNING *
        '''),
        parameters: {
            'userId': account.userId,
            'provider': account.provider,
            'providerUserId': account.providerUserId,
            'accessToken': account.accessToken,
            'refreshToken': account.refreshToken,
        },
    );

    return OAuthAccount.fromRow(result.first.toColumnMap());
}

Future<void> updateTokens(int id, String accessToken, String? refreshToken) ↵
↵ async {
    await _db.execute(
        Sql.named('''
            UPDATE oauth_accounts
            SET access_token = @accessToken,
              refresh_token = @refreshToken,
              updated_at = NOW()
            WHERE id = @id
        '''),
        parameters: {
            'id': id,
            'accessToken': accessToken,
            'refreshToken': refreshToken,
        },
    );
}

Future<List<OAuthAccount>> findByUserId(int userId) async {

```

```

    final result = await _db.execute(
      Sql.named('SELECT * FROM oauth_accounts WHERE user_id = @userId'),
      parameters: {'userId': userId},
    );

    return result.map((row) => OAuthAccount.fromRow(row.toColumnMap())).toList();
  }

  Future<void> delete(int id) async {
    await _db.execute(
      Sql.named('DELETE FROM oauth_accounts WHERE id = @id'),
      parameters: {'id': id},
    );
  }

  Future<void> deleteByUserAndProvider(int userId, String provider) async {
    await _db.execute(
      Sql.named('''
        DELETE FROM oauth_accounts
        WHERE user_id = @userId AND provider = @provider
      '''),
      parameters: {'userId': userId, 'provider': provider},
    );
  }
}

```

#### 4.4.2.6 OAuth Service

```

// lib/services/oauth_service.dart
import 'dart:convert';
import 'dart:math';
import '../models/oauth_models.dart';
import '../models/user.dart';
import '../models/token_pair.dart';
import '../repositories/user_repository.dart';
import '../repositories/oauth_account_repository.dart';
import 'jwt_service.dart';
import 'oauth/oauth_provider.dart';

class OAuthService {
  final Map<String, OAuthProvider> _providers = {};
  final UserRepository _userRepo;
  final OAuthAccountRepository _oauthRepo;
  final JwtService _jwtService;

  OAuthService({
    required UserRepository userRepo,
    required OAuthAccountRepository oauthRepo,
    required JwtService jwtService,
  }) : _userRepo = userRepo,

```

```
        _oauthRepo = oauthRepo,
        _jwtService = jwtService;

void registerProvider(OAuthProvider provider) {
    _providers[provider.name] = provider;
}

OAuthProvider getProvider(String name) {
    final provider = _providers[name];
    if (provider == null) {
        throw OAuthException('Unknown OAuth provider: $name');
    }
    return provider;
}

List<String> get availableProviders => _providers.keys.toList();

String generateState() {
    final random = Random.secure();
    final values = List<int>.generate(32, (_) => random.nextInt(256));
    return base64Url.encode(values);
}

String getAuthorizationUrl(String provider, String state) {
    return getProvider(provider).getAuthorizationUrl(state: state);
}

Future<TokenPair> handleCallback({
    required String provider,
    required String code,
}) async {
    final oauthProvider = getProvider(provider);

    // 1. Code gegen Token tauschen
    final tokens = await oauthProvider.exchangeCode(code);

    // 2. User-Info laden
    final userInfo = await oauthProvider.getUserInfo(tokens.accessToken);

    // 3. User finden oder erstellen
    final user = await _findOrCreateUser(userInfo);

    // 4. OAuth-Account verknüpfen
    await _linkOAuthAccount(user.id!, userInfo, tokens);

    // 5. JWT generieren
    return _jwtService.generateTokenPair(user);
}

Future<User> _findOrCreateUser(OAuthUserInfo info) async {
    // 1. Prüfen ob OAuth-Account existiert
```

```
final existingOAuth = await _oauthRepo.findByProviderAndId(
    info.provider,
    info.id,
);

if (existingOAuth != null) {
    final user = await _userRepo.findById(existingOAuth.userId);
    if (user != null) return user;
}

// 2. Prüfen ob Email schon existiert
final existingUser = await _userRepo.findByEmail(info.email.toLowerCase());
if (existingUser != null) {
    return existingUser;
}

// 3. Neuen User erstellen
final newUser = User(
    email: info.email.toLowerCase(),
    passwordHash: '', // Kein Passwort bei OAuth
    name: info.name,
    isActive: true,
    role: 'user',
);

return await _userRepo.create(newUser);
}

Future<void> _linkOAuthAccount(
    int userId,
    OAuthUserInfo info,
    OAuthTokens tokens,
) async {
    final existing = await _oauthRepo.findByProviderAndId(
        info.provider,
        info.id,
    );

    if (existing != null) {
        // Update tokens
        await _oauthRepo.updateTokens(
            existing.id!,
            tokens.accessToken,
            tokens.refreshToken,
        );
    } else {
        // Neue Verknüpfung erstellen
        await _oauthRepo.create(OAuthAccount(
            userId: userId,
            provider: info.provider,
            providerUserId: info.id,
```

```

        accessToken: tokens.accessToken,
        refreshToken: tokens.refreshToken,
    ));
}
}

/// Verknüpfe OAuth-Account mit bestehendem User
Future<void> linkAccountToUser({
    required int userId,
    required String provider,
    required String code,
}) async {
    final oauthProvider = getProvider(provider);

    final tokens = await oauthProvider.exchangeCode(code);
    final userInfo = await oauthProvider.getUserInfo(tokens.accessToken);

    // Prüfen ob Account schon verknüpft ist
    final existing = await _oauthRepo.findByProviderAndId(
        provider,
        userInfo.id,
    );

    if (existing != null && existing.userId != userId) {
        throw OAuthException('This account is already linked to another user');
    }

    await _linkOAuthAccount(userId, userInfo, tokens);
}

/// Entferne OAuth-Account-Verknüpfung
Future<void> unlinkAccount(int userId, String provider) async {
    await _oauthRepo.deleteByUserAndProvider(userId, provider);
}

/// Liste verknüpfter OAuth-Accounts eines Users
Future<List<OAuthAccount>> getLinkedAccounts(int userId) async {
    return await _oauthRepo.findById(userId);
}
}

```

#### 4.4.2.7 OAuth Handler

```

// lib/handlers/oauth_handler.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';
import '../services/oauth_service.dart';
import '../models/oauth_models.dart';

```

```
class OAuthHandler {
    final OAuthService _oauthService;

    // State-Store mit Timestamp für Timeout
    final Map<String, DateTime> _stateStore = {};
    final Duration _stateTimeout = const Duration(minutes: 10);

    OAuthHandler(this._oauthService);

    Router get router {
        final router = Router();

        // Verfügbare Provider auflisten
        router.get('/providers', _listProviders);

        // OAuth-Flow initiieren
        router.get('/<provider>', _initiateOAuth);

        // Callback verarbeiten
        router.get('/<provider>/callback', _handleCallback);

        return router;
    }

    Future<Response> _listProviders(Request request) async {
        return Response.ok(
            jsonEncode({
                'providers': _oauthService.availableProviders,
            }),
            headers: {'content-type': 'application/json'},
        );
    }

    Future<Response> _initiateOAuth(Request request) async {
        try {
            final provider = request.params['provider']!;

            // State generieren
            final state = _oauthService.generateState();
            _stateStore[state] = DateTime.now();

            // Alte States aufräumen
            _cleanupExpiredStates();

            // Authorization URL generieren
            final authUrl = _oauthService.getAuthorizationUrl(provider, state);

            // Redirect zum Provider
            return Response.found(authUrl);
        } on OAuthException catch (e) {
            return _errorResponse(400, e.message);
        }
    }
}
```

```
}
}

Future<Response> _handleCallback(Request request) async {
  final provider = request.params['provider']!;
  final queryParams = request.url.queryParameters;

  final code = queryParams['code'];
  final state = queryParams['state'];
  final error = queryParams['error'];
  final errorDescription = queryParams['error_description'];

  // 1. Error vom Provider
  if (error != null) {
    return _errorResponse(400, errorDescription ?? error);
  }

  // 2. Code fehlt
  if (code == null) {
    return _errorResponse(400, 'No authorization code received');
  }

  // 3. State validieren
  if (state == null || !_validateAndRemoveState(state)) {
    return _errorResponse(400, 'Invalid or expired state parameter');
  }

  try {
    // 4. OAuth-Flow abschließen
    final tokens = await _oauthService.handleCallback(
      provider: provider,
      code: code,
    );

    // 5. Erfolg-Response
    return Response.ok(
      jsonEncode({
        'message': 'Login successful',
        ...tokens.toJson(),
      }),
      headers: {'content-type': 'application/json'},
    );
  } on OAuthException catch (e) {
    return _errorResponse(e.statusCode ?? 400, e.message);
  } catch (e) {
    return _errorResponse(500, 'OAuth authentication failed');
  }
}

bool _validateAndRemoveState(String state) {
  final timestamp = _stateStore.remove(state);
```

```

    if (timestamp == null) return false;
    return DateTime.now().difference(timestamp) < _stateTimeout;
}

void _cleanupExpiredStates() {
    final now = DateTime.now();
    _stateStore.removeWhere((_, timestamp) =>
        now.difference(timestamp) > _stateTimeout);
}

Response _errorResponse(int statusCode, String message) {
    return Response(
        statusCode,
        body: jsonEncode({'error': message}),
        headers: {'content-type': 'application/json'},
    );
}
}

```

#### 4.4.2.8 Integration

```

// bin/server.dart (Erweiterung)

void main() async {
    // ... bestehende Initialisierung ...

    // OAuth Config
    final oauthConfig = OAuthConfig.fromEnvironment();

    // OAuth Service
    final oauthService = OAuthService(
        userRepo: userRepo,
        oauthRepo: oauthAccountRepo,
        jwtService: jwtService,
    );

    // Provider registrieren
    if (oauthConfig.googleClientId.isNotEmpty) {
        oauthService.registerProvider(GoogleOAuthProvider(
            clientId: oauthConfig.googleClientId,
            clientSecret: oauthConfig.googleClientSecret,
            redirectUri: oauthConfig.googleRedirectUri,
        ));
    }

    if (oauthConfig.githubClientId.isNotEmpty) {
        oauthService.registerProvider(GitHubOAuthProvider(
            clientId: oauthConfig.githubClientId,
            clientSecret: oauthConfig.githubClientSecret,
            redirectUri: oauthConfig.githubRedirectUri,
        ));
    }
}

```

```

    ));
}

// Handler
final oauthHandler = OAuthHandler(oauthService);

// Router
final router = Router();
router.mount('/api/auth/oauth', oauthHandler.router);
// ... weitere Routen ...
}

```

### 4.4.3 Ressourcen

#### 4.4.3.1 Offizielle Dokumentation

- OAuth 2.0 RFC 6749
- Google OAuth Documentation
- GitHub OAuth Documentation
- OWASP OAuth Cheat Sheet

#### 4.4.3.2 Cheat Sheet: OAuth 2.0 Flows

Flow	Verwendung	Sicherheit
<b>Authorization Code</b>	Server-Apps	Hoch
Authorization Code + PKCE	Mobile/SPA	Hoch
Implicit	Legacy SPAs	Niedrig (veraltet)
Client Credentials	Server-to-Server	Hoch
Resource Owner Password	Legacy	Niedrig

#### 4.4.3.3 Cheat Sheet: Authorization Code Flow

1. Client -> Auth Server: GET /authorize  
?client\_id=...  
&redirect\_uri=...  
&response\_type=code  
&scope=...  
&state=...
2. User authentifiziert sich
3. Auth Server -> Client: Redirect  
?code=...  
&state=...
4. Client -> Auth Server: POST /token  
client\_id=...  
&client\_secret=...  
&code=...  
&grant\_type=authorization\_code

5. Auth Server -> Client: Access Token

#### 4.4.3.4 Cheat Sheet: Google OAuth

```
// Endpoints
const authEndpoint = 'https://accounts.google.com/o/oauth2/v2/auth';
const tokenEndpoint = 'https://oauth2.googleapis.com/token';
const userInfoEndpoint = 'https://www.googleapis.com/oauth2/v2/userinfo';

// Authorization URL
final authUrl = Uri.parse(authEndpoint).replace(queryParameters: {
  'client_id': clientId,
  'redirect_uri': redirectUri,
  'response_type': 'code',
  'scope': 'openid email profile',
  'access_type': 'offline', // Für Refresh Token
  'prompt': 'consent',      // Immer Consent-Screen zeigen
  'state': state,
});

// Token Exchange
final response = await http.post(Uri.parse(tokenEndpoint), body: {
  'client_id': clientId,
  'client_secret': clientSecret,
  'code': code,
  'grant_type': 'authorization_code',
  'redirect_uri': redirectUri,
});

// User Info
final response = await http.get(
  Uri.parse(userInfoEndpoint),
  headers: {'Authorization': 'Bearer $accessToken'},
);
```

#### 4.4.3.5 Cheat Sheet: GitHub OAuth

```
// Endpoints
const authEndpoint = 'https://github.com/login/oauth/authorize';
const tokenEndpoint = 'https://github.com/login/oauth/access_token';
const userInfoEndpoint = 'https://api.github.com/user';
const emailEndpoint = 'https://api.github.com/user/emails';

// Wichtig: Accept: application/json für Token-Request!
final response = await http.post(Uri.parse(tokenEndpoint),
  headers: {'Accept': 'application/json'},
  body: {
    'client_id': clientId,
    'client_secret': clientSecret,
```

```

        'code': code,
    },
);

// User Info (GitHub-spezifischer Accept-Header)
final response = await http.get(
    Uri.parse(userInfoEndpoint),
    headers: {
        'Authorization': 'Bearer $accessToken',
        'Accept': 'application/vnd.github.v3+json',
    },
);

```

#### 4.4.3.6 Cheat Sheet: State Parameter (CSRF-Schutz)

```

// State generieren
String generateState() {
    final random = Random.secure();
    final values = List<int>.generate(32, (_) => random.nextInt(256));
    return base64Url.encode(values);
}

// State speichern (mit Timeout)
final stateStore = <String, DateTime>{};
stateStore[state] = DateTime.now();

// State validieren
bool validateState(String state) {
    final timestamp = stateStore.remove(state);
    if (timestamp == null) return false;
    return DateTime.now().difference(timestamp) < Duration(minutes: 10);
}

```

#### 4.4.3.7 Cheat Sheet: Scopes

Google

Scope	Beschreibung
openid	OpenID Connect
email	Email-Adresse
profile	Basis-Profil
<a href="https://www.googleapis.com/auth/calendar">https://www.googleapis.com/auth/calendar</a>	Kalender
<a href="https://www.googleapis.com/auth/drive">https://www.googleapis.com/auth/drive</a>	Drive

GitHub

Scope	Beschreibung
read:user	Profil lesen

Scope	Beschreibung
user:email	Email lesen
repo	Repositories
gist	Gists

#### 4.4.3.8 Best Practices

##### DO

1. **State Parameter** - Immer für CSRF-Schutz
2. **HTTPS** - OAuth nur über HTTPS
3. **Redirect URI validieren** - Exakte Match
4. **Secrets serverseitig** - Nie im Client
5. **Tokens sicher speichern** - Verschlüsselt in DB
6. **Minimale Scopes** - Nur nötige anfordern

##### DON'T

1. **Client Secret im Frontend** - Nie!
2. **State wiederverwenden** - Immer neu generieren
3. **Implicit Flow** - Veraltet und unsicher
4. **Unvalidierte Redirects** - Open Redirect Vulnerability
5. **Tokens im URL-Parameter** - Log-Leaks

#### 4.4.3.9 SQL Schema

```
CREATE TABLE oauth_accounts (
  id SERIAL PRIMARY KEY,
  user_id INTEGER NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  provider VARCHAR(50) NOT NULL,
  provider_user_id VARCHAR(255) NOT NULL,
  access_token TEXT NOT NULL,
  refresh_token TEXT,
  token_expires_at TIMESTAMP,
  created_at TIMESTAMP NOT NULL DEFAULT NOW(),
  updated_at TIMESTAMP,
  UNIQUE(provider, provider_user_id)
);

CREATE INDEX idx_oauth_user ON oauth_accounts(user_id);
CREATE INDEX idx_oauth_provider ON oauth_accounts(provider, provider_user_id);
```

#### 4.4.3.10 Provider Setup URLs

Provider	Console URL
Google	<a href="https://console.cloud.google.com/apis/credentials">https://console.cloud.google.com/apis/credentials</a>
GitHub	<a href="https://github.com/settings/developers">https://github.com/settings/developers</a>
Microsoft	<a href="https://portal.azure.com/#blade/Microsoft_AAD_RegisteredApps">https://portal.azure.com/#blade/Microsoft_AAD_RegisteredApps</a>
Facebook	<a href="https://developers.facebook.com/apps">https://developers.facebook.com/apps</a>

Provider	Console URL
Apple	<a href="https://developer.apple.com/account/resources/identifiers">https://developer.apple.com/account/resources/identifiers</a>

## 4.5 Einheit 8.5: API Sicherheit

### 4.5.0.1 Lernziele

Nach dieser Einheit kannst du: - CORS richtig konfigurieren - Rate Limiting implementieren - Input Sanitization durchführen - SQL Injection verhindern - HTTPS und Security Headers einsetzen

### 4.5.0.2 OWASP Top 10

Die häufigsten Sicherheitsrisiken für Web-APIs:

Rang	Risiko	Beschreibung
1	Broken Access Control	Unzureichende Autorisierung
2	Cryptographic Failures	Schwache Verschlüsselung
3	Injection	SQL/NoSQL/Command Injection
4	Insecure Design	Architektur-Schwächen
5	Security Misconfiguration	Falsche Konfiguration
6	Vulnerable Components	Unsichere Dependencies
7	Authentication Failures	Schwache Authentifizierung
8	Data Integrity Failures	Manipulierte Daten
9	Logging Failures	Unzureichendes Logging
10	SSRF	Server-Side Request Forgery

### 4.5.0.3 CORS (Cross-Origin Resource Sharing)

Problem

Browser blockieren Requests von anderen Domains (Same-Origin Policy).

Lösung

Server erlaubt explizit bestimmte Origins:

```
// lib/middleware/cors_middleware.dart

Middleware corsMiddleware({
  List<String>? allowedOrigins,
  List<String>? allowedMethods,
  List<String>? allowedHeaders,
  bool allowCredentials = false,
  Duration? maxAge,
}) {
  final origins = allowedOrigins ?? ['*'];
  final methods = allowedMethods ?? ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'];
  final headers = allowedHeaders ?? ['Content-Type', 'Authorization'];

  return (Handler innerHandler) {
```

```
return (Request request) async {
  final origin = request.headers['origin'];

  // CORS-Header erstellen
  final corsHeaders = <String, String>{};

  // Origin prüfen
  if (origin != null) {
    if (origins.contains('*')) {
      corsHeaders['Access-Control-Allow-Origin'] = '*';
    } else if (origins.contains(origin)) {
      corsHeaders['Access-Control-Allow-Origin'] = origin;
      corsHeaders['Vary'] = 'Origin';
    }
  }

  corsHeaders['Access-Control-Allow-Methods'] = methods.join(' ', '');
  corsHeaders['Access-Control-Allow-Headers'] = headers.join(' ', '');

  if (allowCredentials) {
    corsHeaders['Access-Control-Allow-Credentials'] = 'true';
  }

  if (maxAge != null) {
    corsHeaders['Access-Control-Max-Age'] = maxAge.inSeconds.toString();
  }

  // Preflight Request (OPTIONS)
  if (request.method == 'OPTIONS') {
    return Response.ok('', headers: corsHeaders);
  }

  // Normaler Request
  final response = await innerHandler(request);
  return response.change(headers: corsHeaders);
};
};
}
```

Konfiguration für Produktion

```
// Entwicklung
final cors = corsMiddleware(
  allowedOrigins: ['*'],
);

// Produktion
final cors = corsMiddleware(
  allowedOrigins: [
    'https://myapp.com',
    'https://admin.myapp.com',
```

```
],
allowedMethods: ['GET', 'POST', 'PUT', 'DELETE'],
allowedHeaders: ['Content-Type', 'Authorization'],
allowCredentials: true,
maxAge: Duration(hours: 1),
);
```

#### 4.5.0.4 Rate Limiting

Sliding Window mit Redis

```
// lib/middleware/rate_limit_middleware.dart
import '../services/rate_limiter.dart';

Middleware rateLimitMiddleware({
  required RateLimiter rateLimiter,
  int maxRequests = 100,
  Duration window = const Duration(minutes: 1),
  String Function(Request)? keyExtractor,
}) {
  return (Handler innerHandler) {
    return (Request request) async {
      // Client-Identifizieren
      final clientId = keyExtractor?.call(request) ?? _getClientId(request);

      // Rate Limit prüfen
      final result = await rateLimiter.check(
        clientId: clientId,
        maxRequests: maxRequests,
        window: window,
      );

      // Response-Header setzen
      final headers = {
        'X-RateLimit-Limit': maxRequests.toString(),
        'X-RateLimit-Remaining': result.remaining.toString(),
        'X-RateLimit-Reset': result.resetAt.toIso8601String(),
      };

      if (!result.allowed) {
        return Response(
          429,
          body: jsonEncode({
            'error': 'Too many requests',
            'retry_after': result.retryAfterSeconds,
          }),
          headers: {
            ...headers,
            'content-type': 'application/json',
            'Retry-After': result.retryAfterSeconds.toString(),
          },
        );
      }
    };
  };
}
```

```

        );
    }

    final response = await innerHandler(request);
    return response.change(headers: headers);
};
};
}

String _getClientId(Request request) {
  // 1. Authentifizierter User
  final userId = request.context['userId'] as int?;
  if (userId != null) return 'user:$userId';

  // 2. IP-Adresse
  final forwarded = request.headers['x-forwarded-for'];
  if (forwarded != null) return 'ip:${forwarded.split(',').first.trim()}';

  final realIp = request.headers['x-real-ip'];
  if (realIp != null) return 'ip:$realIp';

  return 'ip:unknown';
}

```

#### Rate Limiter Service

```

// lib/services/rate_limiter.dart

class RateLimitResult {
  final bool allowed;
  final int remaining;
  final DateTime resetAt;
  final int retryAfterSeconds;

  RateLimitResult({
    required this.allowed,
    required this.remaining,
    required this.resetAt,
    this.retryAfterSeconds = 0,
  });
}

class RateLimiter {
  final RedisClient _redis;

  RateLimiter(this._redis);

  Future<RateLimitResult> check({
    required String clientId,
    required int maxRequests,
    required Duration window,

```

```
} async {
    final key = 'ratelimit:$clientId';
    final now = DateTime.now();
    final windowStart = now.subtract(window);

    // Sliding Window: Alte Einträge entfernen
    await _redis.zremrangebyscore(
        key,
        0,
        windowStart.millisecondsSinceEpoch.toDouble(),
    );

    // Aktuelle Anzahl
    final count = await _redis.zcard(key);

    final resetAt = now.add(window);

    if (count >= maxRequests) {
        // Rate Limit erreicht
        final oldestEntry = await _redis.command.send_object([
            'ZRANGE', key, '0', '0', 'WITHSCORES'
        ]);

        int retryAfter = window.inSeconds;
        if (oldestEntry != null && (oldestEntry as List).length >= 2) {
            final oldestTime = double.parse(oldestEntry[1] as String).toInt();
            final oldestDate = DateTime.fromMillisecondsSinceEpoch(oldestTime);
            retryAfter = oldestDate.add(window).difference(now).inSeconds;
            if (retryAfter < 0) retryAfter = 0;
        }

        return RateLimitResult(
            allowed: false,
            remaining: 0,
            resetAt: resetAt,
            retryAfterSeconds: retryAfter,
        );
    }

    // Request hinzufügen
    await _redis.zadd(
        key,
        now.millisecondsSinceEpoch.toDouble(),
        now.millisecondsSinceEpoch.toString(),
    );

    // TTL setzen
    await _redis.expire(key, window);

    return RateLimitResult(
        allowed: true,
```

```
        remaining: maxRequests - count - 1,  
        resetAt: resetAt,  
    );  
}  
}
```

#### 4.5.0.5 Input Validation & Sanitization

##### Validation Middleware

```
// lib/middleware/validation_middleware.dart  
  
class ValidationError {  
    final String field;  
    final String message;  
  
    ValidationError(this.field, this.message);  
  
    Map<String, dynamic> toJson() => {'field': field, 'message': message};  
}  
  
class Validator {  
    final List<ValidationError> _errors = [];  
  
    List<ValidationError> get errors => List.unmodifiable(_errors);  
    bool get hasErrors => _errors.isNotEmpty;  
    bool get isValid => _errors.isEmpty;  
  
    void addError(String field, String message) {  
        _errors.add(ValidationError(field, message));  
    }  
  
    // String Validierung  
    String? validateString(  
        String field,  
        String? value, {  
        bool required = true,  
        int? minLength,  
        int? maxLength,  
        RegExp? pattern,  
        String? patternMessage,  
    }) {  
        if (value == null || value.isEmpty) {  
            if (required) addError(field, '$field is required');  
            return null;  
        }  
  
        final trimmed = value.trim();  
  
        if (minLength != null && trimmed.length < minLength) {  
            addError(field, '$field must be at least $minLength characters');  
        }  
    }  
}
```

```
}

if (maxLength != null && trimmed.length > maxLength) {
    addError(field, '$field must be at most $maxLength characters');
}

if (pattern != null && !pattern.hasMatch(trimmed)) {
    addError(field, patternMessage ?? '$field has invalid format');
}

return trimmed;
}

// Email Validierung
String? validateEmail(String field, String? value, {bool required = true}) {
    return validateString(
        field,
        value,
        required: required,
        pattern: RegExp(r'^[\w-\.\.]+\@([\w-]+\.\.)+[\w-]{2,4}$'),
        patternMessage: 'Invalid email format',
    );
}

// Integer Validierung
int? validateInt(
    String field,
    dynamic value, {
    bool required = true,
    int? min,
    int? max,
}) {
    if (value == null) {
        if (required) addError(field, '$field is required');
        return null;
    }

    final intValue = value is int ? value : int.tryParse(value.toString());

    if (intValue == null) {
        addError(field, '$field must be a valid integer');
        return null;
    }

    if (min != null && intValue < min) {
        addError(field, '$field must be at least $min');
    }

    if (max != null && intValue > max) {
        addError(field, '$field must be at most $max');
    }
}
```

```
    return intValue;
  }
}
```

#### HTML/XSS Sanitization

```
// lib/utils/sanitizer.dart

class Sanitizer {
  /// HTML-Tags entfernen
  static String stripHtml(String input) {
    return input.replaceAll(RegExp(r'<[^>]*>'), '');
  }

  /// HTML-Entities escapen
  static String escapeHtml(String input) {
    return input
      .replaceAll('&', '&amp;')
      .replaceAll('<', '&lt;')
      .replaceAll('>', '&gt;')
      .replaceAll('"', '&quot;')
      .replaceAll("'", '&#x27;');
  }

  /// SQL-kritische Zeichen (für Logging, nicht als SQL-Schutz!)
  static String sanitizeForLog(String input) {
    return input
      .replaceAll('\n', '\\n')
      .replaceAll('\r', '\\r')
      .replaceAll('\t', '\\t');
  }

  /// URL-Parameter sanitizen
  static String sanitizeUrlParam(String input) {
    return Uri.encodeComponent(input);
  }

  /// Dateinamen sanitizen
  static String sanitizeFilename(String filename) {
    return filename
      .replaceAll(RegExp(r'^\w\-\.\.'], '_')
      .replaceAll(RegExp(r'\.{2,}'), '.')
      .replaceAll(RegExp(r'^\.+|\..+$'), '');
  }
}
```

#### 4.5.0.6 SQL Injection Prevention

Prepared Statements (immer verwenden!)

```
// SICHER: Prepared Statements
final result = await db.execute(
  Sql.named('SELECT * FROM users WHERE email = @email'),
  parameters: {'email': userInput},
);

// UNSICHER: String Concatenation - NIEMALS!
final result = await db.execute(
  "SELECT * FROM users WHERE email = '$userInput'", // GEFÄHRLICH!
);
```

Repository mit sicheren Queries

```
class UserRepository {
  final Connection _db;

  // SICHER
  Future<User?> findByEmail(String email) async {
    final result = await _db.execute(
      Sql.named('SELECT * FROM users WHERE email = @email'),
      parameters: {'email': email.toLowerCase().trim()},
    );
    if (result.isEmpty) return null;
    return User.fromJson(result.first.toColumnMap());
  }

  // SICHER: Dynamische Sortierung mit Whitelist
  Future<List<User>> findAll({
    String orderBy = 'created_at',
    bool descending = true,
  }) async {
    // Whitelist für erlaubte Spalten
    const allowedColumns = {'id', 'email', 'name', 'created_at'};

    if (!allowedColumns.contains(orderBy)) {
      orderBy = 'created_at';
    }

    final direction = descending ? 'DESC' : 'ASC';

    // Spaltenname ist gewhitelistet, daher sicher
    final result = await _db.execute(
      'SELECT * FROM users ORDER BY $orderBy $direction',
    );

    return result.map((r) => User.fromJson(r.toColumnMap())).toList();
  }

  // SICHER: LIKE-Suche mit Escaping
  Future<List<User>> search(String query) async {
    // Sonderzeichen in LIKE escapen
```

```
final escapedQuery = query
  .replaceAll('%', '\\%')
  .replaceAll('_', '\\_');

final result = await _db.execute(
  Sql.named('''
    SELECT * FROM users
    WHERE name ILIKE @pattern OR email ILIKE @pattern
  '''),
  parameters: {'pattern': '%$escapedQuery%'},
);

return result.map((r) => User.fromJson(r.toColumnMap())).toList();
}
}
```

#### 4.5.0.7 Security Headers

```
// lib/middleware/security_headers_middleware.dart

Middleware securityHeadersMiddleware() {
  return (Handler innerHandler) {
    return (Request request) async {
      final response = await innerHandler(request);

      return response.change(headers: {
        // Verhindert MIME-Type Sniffing
        'X-Content-Type-Options': 'nosniff',

        // Clickjacking-Schutz
        'X-Frame-Options': 'DENY',

        // XSS-Filter (Legacy-Browser)
        'X-XSS-Protection': '1; mode=block',

        // Referrer einschränken
        'Referrer-Policy': 'strict-origin-when-cross-origin',

        // Content Security Policy
        'Content-Security-Policy': "default-src 'self'",

        // Strict Transport Security (nur mit HTTPS)
        // 'Strict-Transport-Security': 'max-age=31536000; includeSubDomains',

        // Keine Cache für API-Responses
        'Cache-Control': 'no-store',
        'Pragma': 'no-cache',
      });
    };
  };
};
```

```
}
```

#### 4.5.0.8 HTTPS

TLS mit Shelf

```
// bin/server.dart
import 'dart:io';

void main() async {
  final securityContext = SecurityContext()
    ..useCertificateChain('cert.pem')
    ..usePrivateKey('key.pem');

  final server = await HttpServer.bindSecure(
    InternetAddress.anyIPv4,
    443,
    securityContext,
  );

  serveRequests(server, handler);
}
```

In Produktion: Reverse Proxy

Besser: nginx/Caddy vor der Dart-App für TLS-Terminierung.

```
# nginx.conf
server {
  listen 443 ssl http2;
  server_name api.example.com;

  ssl_certificate /etc/letsencrypt/live/api.example.com/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/api.example.com/privkey.pem;

  location / {
    proxy_pass http://localhost:8080;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
  }
}
```

#### 4.5.0.9 Error Handling (ohne Leak)

```
// lib/middleware/error_handler_middleware.dart

Middleware errorHandlerMiddleware({bool isDevelopment = false}) {
  return (Handler innerHandler) {
    return (Request request) async {
```

```

    try {
      return await innerHandler(request);
    } catch (e, stack) {
      // Immer loggen
      print('Error: $e\n$stack');

      // Produktion: Keine Details leaken
      if (!isDevelopment) {
        return Response.internalServerError(
          body: jsonEncode({'error': 'Internal server error'}),
          headers: {'content-type': 'application/json'},
        );
      }

      // Development: Details anzeigen
      return Response.internalServerError(
        body: jsonEncode({
          'error': e.toString(),
          'stack': stack.toString(),
        }),
        headers: {'content-type': 'application/json'},
      );
    }
  };
};
}

```

#### 4.5.0.10 Zusammenfassung

Maßnahme	Schutz gegen
<b>CORS</b>	Unauthorized Cross-Origin Requests
<b>Rate Limiting</b>	DoS, Brute Force
<b>Input Validation</b>	Invalid Data, Injection
<b>Prepared Statements</b>	SQL Injection
<b>Sanitization</b>	XSS, Injection
<b>Security Headers</b>	Clickjacking, MIME Sniffing
<b>HTTPS</b>	Man-in-the-Middle
<b>Error Handling</b>	Information Disclosure

### 4.5.1 Übung

#### 4.5.1.1 Ziel

Implementiere umfassende Sicherheitsmaßnahmen für deine API.

#### 4.5.1.2 Aufgabe 1: CORS Middleware (15 min)

```

// lib/middleware/cors_middleware.dart

class CorsConfig {

```

```

final List<String> allowedOrigins;
final List<String> allowedMethods;
final List<String> allowedHeaders;
final bool allowCredentials;
final Duration? maxAge;

const CorsConfig({
  this.allowedOrigins = const ['*'],
  this.allowedMethods = const ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
  this.allowedHeaders = const ['Content-Type', 'Authorization'],
  this.allowCredentials = false,
  this.maxAge,
});

// Vordefinierte Configs
static const development = CorsConfig();

static const production = CorsConfig(
  allowedOrigins: [], // Muss konfiguriert werden
  allowCredentials: true,
  maxAge: Duration(hours: 1),
);
}

Middleware corsMiddleware(CorsConfig config) {
  return (Handler innerHandler) {
    return (Request request) async {
      // TODO:
      // 1. Origin aus Request-Header lesen
      // 2. Prüfen ob Origin erlaubt ist
      // 3. CORS-Header erstellen:
      //    - Access-Control-Allow-Origin
      //    - Access-Control-Allow-Methods
      //    - Access-Control-Allow-Headers
      //    - Access-Control-Allow-Credentials (wenn true)
      //    - Access-Control-Max-Age (wenn gesetzt)
      //    - Vary: Origin (wenn nicht '*')
      // 4. Bei OPTIONS (Preflight): 200 OK mit Headers
      // 5. Sonst: Response mit CORS-Headers erweitern
    };
  };
}

```

#### 4.5.1.3 Aufgabe 2: Rate Limiter (20 min)

```

// lib/services/rate_limiter.dart

class RateLimitResult {
  final bool allowed;
  final int remaining;
}

```

```
final int limit;
final DateTime resetAt;

RateLimitResult({
  required this.allowed,
  required this.remaining,
  required this.limit,
  required this.resetAt,
});

int get retryAfterSeconds =>
  resetAt.difference(DateTime.now()).inSeconds.clamp(0, 999999);
}

abstract class RateLimiter {
  Future<RateLimitResult> check({
    required String clientId,
    required int maxRequests,
    required Duration window,
  });
}

/// In-Memory Rate Limiter (für Entwicklung/kleine Apps)
class InMemoryRateLimiter implements RateLimiter {
  final Map<String, List<DateTime>> _requests = {};

  @override
  Future<RateLimitResult> check({
    required String clientId,
    required int maxRequests,
    required Duration window,
  }) async {
    // TODO:
    // 1. Alte Requests außerhalb des Fensters entfernen
    // 2. Aktuelle Anzahl prüfen
    // 3. Bei Überschreitung: allowed=false
    // 4. Sonst: Request hinzufügen, allowed=true
  }
}

/// Redis Rate Limiter (für Produktion)
class RedisRateLimiter implements RateLimiter {
  final RedisClient _redis;

  RedisRateLimiter(this._redis);

  @override
  Future<RateLimitResult> check({
    required String clientId,
    required int maxRequests,
    required Duration window,
```

```

    }) async {
      // TODO: Sliding Window mit Redis Sorted Sets
    }
  }
}

```

#### Rate Limit Middleware

```

// lib/middleware/rate_limit_middleware.dart

Middleware rateLimitMiddleware({
  required RateLimiter rateLimiter,
  int maxRequests = 100,
  Duration window = const Duration(minutes: 1),
  String Function(Request)? keyExtractor,
}) {
  return (Handler innerHandler) {
    return (Request request) async {
      // TODO:
      // 1. Client-ID extrahieren (keyExtractor oder default)
      // 2. Rate Limit prüfen
      // 3. Header setzen:
      //    - X-RateLimit-Limit
      //    - X-RateLimit-Remaining
      //    - X-RateLimit-Reset
      // 4. Bei Überschreitung: 429 Too Many Requests
      //    - Retry-After Header
      // 5. Sonst: Handler aufrufen
    };
  };
}

/// Standard Key Extractor (User-ID oder IP)
String defaultKeyExtractor(Request request) {
  // TODO:
  // 1. Prüfen ob authentifiziert (userId in context)
  // 2. X-Forwarded-For Header prüfen
  // 3. X-Real-IP Header prüfen
  // 4. Fallback: 'unknown'
}

```

#### 4.5.1.4 Aufgabe 3: Input Validator (20 min)

```

// lib/validation/validator.dart

class ValidationResult {
  final bool isValid;
  final Map<String, List<String>> errors;

  ValidationResult(this.errors) : isValid = errors.isEmpty;

  Map<String, dynamic> toJson() => {

```

```
        'valid': isValid,
        'errors': errors,
    };
}

class Validator {
    final Map<String, List<String>> _errors = {};

    ValidationResult get result => ValidationResult(Map.from(_errors));

    void addError(String field, String message) {
        _errors.putIfAbsent(field, () => []).add(message);
    }

    /// Validiere String-Feld
    String? string(
        String field,
        dynamic value, {
        bool required = true,
        int? minLength,
        int? maxLength,
        RegExp? pattern,
        String? patternMessage,
    }) {
        // TODO: Implementieren
    }

    /// Validiere Email
    String? email(String field, dynamic value, {bool required = true}) {
        // TODO: Implementieren (mit email Regex)
    }

    /// Validiere Integer
    int? integer(
        String field,
        dynamic value, {
        bool required = true,
        int? min,
        int? max,
    }) {
        // TODO: Implementieren
    }

    /// Validiere Liste
    List<T>? list<T>(
        String field,
        dynamic value, {
        bool required = true,
        int? minLength,
        int? maxLength,
    }) {
```

```
// TODO: Implementieren
}

/// Validiere Enum
T? enumValue<T extends Enum>(
  String field,
  dynamic value,
  List<T> values, {
  bool required = true,
}) {
  // TODO: Implementieren
}
}
```

#### Validation Middleware

```
// lib/middleware/validation_middleware.dart

Middleware validateBody<T>({
  required T Function(Map<String, dynamic>) fromJson,
  required void Function(Validator, T) validate,
}) {
  return (Handler innerHandler) {
    return (Request request) async {
      // TODO:
      // 1. Body parsen
      // 2. DTO erstellen mit fromJson
      // 3. Validator erstellen und validate aufrufen
      // 4. Bei Fehlern: 400 Bad Request mit errors
      // 5. Sonst: DTO in context speichern, Handler aufrufen
    };
  };
}
```

#### 4.5.1.5 Aufgabe 4: Sanitizer (15 min)

```
// lib/utils/sanitizer.dart

class Sanitizer {
  /// Entferne HTML-Tags
  static String stripHtml(String input) {
    // TODO: Regex für HTML-Tags
  }

  /// Escape HTML-Entities
  static String escapeHtml(String input) {
    // TODO: &, <, >, ", ' ersetzen
  }

  /// Sanitize für SQL LIKE-Queries
  static String escapeLike(String input) {

```

```

    // TODO: %, _ escapen
}

/// Sanitize Dateiname
static String sanitizeFilename(String filename) {
    // TODO:
    // - Nur alphanumerisch, -, _, .
    // - Keine doppelten Punkte
    // - Keine führenden/trailing Punkte
}

/// Sanitize für Logging (keine Newlines etc.)
static String sanitizeForLog(String input, {int maxLength = 200}) {
    // TODO: \n, \r, \t ersetzen, Länge begrenzen
}

/// Trim und Normalize Whitespace
static String normalizeWhitespace(String input) {
    return input.trim().replaceAll(RegExp(r'\s+'), ' ');
}
}

```

#### 4.5.1.6 Aufgabe 5: Security Headers Middleware (10 min)

```

// lib/middleware/security_headers_middleware.dart

class SecurityHeadersConfig {
    final bool noSniff;
    final String frameOptions;
    final bool xssProtection;
    final String referrerPolicy;
    final String? contentSecurityPolicy;
    final bool noCache;

    const SecurityHeadersConfig({
        this.noSniff = true,
        this.frameOptions = 'DENY',
        this.xssProtection = true,
        this.referrerPolicy = 'strict-origin-when-cross-origin',
        this.contentSecurityPolicy,
        this.noCache = true,
    });

    static const api = SecurityHeadersConfig(
        contentSecurityPolicy: "default-src 'none'",
    );
}

Middleware securityHeadersMiddleware([
    SecurityHeadersConfig config = const SecurityHeadersConfig(),

```

```
1) {  
  return (Handler innerHandler) {  
    return (Request request) async {  
      // TODO:  
      // 1. Handler aufrufen  
      // 2. Security-Header zur Response hinzufügen:  
      //    - X-Content-Type-Options: nosniff  
      //    - X-Frame-Options  
      //    - X-XSS-Protection  
      //    - Referrer-Policy  
      //    - Content-Security-Policy  
      //    - Cache-Control, Pragma (wenn noCache)  
    };  
  };  
}
```

#### 4.5.1.7 Aufgabe 6: Error Handler (ohne Leak) (10 min)

```
// lib/middleware/error_handler_middleware.dart  
  
Middleware errorHandlerMiddleware({  
  required bool isDevelopment,  
  void Function(Object error, StackTrace stack)? onError,  
}) {  
  return (Handler innerHandler) {  
    return (Request request) async {  
      try {  
        return await innerHandler(request);  
      } catch (e, stack) {  
        // TODO:  
        // 1. Error loggen (onError callback)  
        // 2. In Development: Details zurückgeben  
        // 3. In Production: Generische Fehlermeldung  
        // 4. Passenden Status Code verwenden:  
        //    - ValidationException -> 400  
        //    - AuthException -> 401/403  
        //    - NotFoundException -> 404  
        //    - Sonstige -> 500  
      }  
    };  
  };  
}
```

#### 4.5.1.8 Aufgabe 7: Sichere Repository-Methoden (15 min)

```
// lib/repositories/secure_product_repository.dart  
  
class ProductRepository {  
  final Connection _db;
```

```
ProductRepository(this._db);

/// Suche mit sicherer LIKE-Query
Future<List<Product>> search(String query) async {
  // TODO:
  // 1. Query sanitizen mit Sanitizer.escapeLike
  // 2. Prepared Statement verwenden
  // 3. ILIKE für case-insensitive Suche
}

/// Sortierung mit Whitelist
Future<List<Product>> findAll({
  String sortBy = 'created_at',
  bool descending = true,
  int limit = 20,
  int offset = 0,
}) async {
  // TODO:
  // 1. sortBy gegen Whitelist prüfen
  // 2. limit/offset validieren (max 100)
  // 3. Sichere Query bauen
}

/// Bulk-Insert mit Prepared Statements
Future<void> createMany(List<Product> products) async {
  // TODO: Transaction mit mehreren INSERT-Statements
}
}
```

#### 4.5.1.9 Aufgabe 8: Integration (10 min)

```
// bin/server.dart

void main() async {
  final isDevelopment = Platform.environment['ENV'] != 'production';

  // Rate Limiter
  final rateLimiter = isDevelopment
    ? InMemoryRateLimiter()
    : RedisRateLimiter(redisClient);

  // CORS Config
  final corsConfig = isDevelopment
    ? CorsConfig.development
    : CorsConfig(
      allowedOrigins: ['https://myapp.com'],
      allowCredentials: true,
    );
}
```

```
// Pipeline aufbauen
final handler = const Pipeline()
  .addMiddleware(errorHandlerMiddleware(isDevelopment: isDevelopment))
  .addMiddleware(securityHeadersMiddleware())
  .addMiddleware(corsMiddleware(corsConfig))
  .addMiddleware(rateLimitMiddleware(
    rateLimiter: rateLimiter,
    maxRequests: 100,
    window: Duration(minutes: 1),
  ))
  .addMiddleware(logRequests())
  .addHandler(router);

await serve(handler, 'localhost', 8080);
}
```

#### 4.5.1.10 Testen

##### Rate Limiting

```
# Viele Requests schnell senden
for i in {1..110}; do
  curl -s -o /dev/null -w "%{http_code}\n" http://localhost:8080/api/products
done

# Sollte nach 100 Requests "429" zurückgeben
```

##### CORS

```
# Preflight Request
curl -X OPTIONS http://localhost:8080/api/products \
  -H "Origin: https://example.com" \
  -H "Access-Control-Request-Method: POST" \
  -v

# Prüfe Access-Control-* Header in Response
```

##### Security Headers

```
curl -v http://localhost:8080/api/products 2>&1 | grep -i
↵  "x-content-type|x-frame|x-xss"
```

#### 4.5.1.11 Abgabe-Checkliste

- ☐ CORS Middleware mit konfigurierbaren Origins
- ☐ Rate Limiter (InMemory und/oder Redis)
- ☐ Rate Limit Middleware mit Headers
- ☐ Validator für String, Email, Integer, List
- ☐ Sanitizer für HTML, LIKE, Filename, Log
- ☐ Security Headers Middleware
- ☐ Error Handler ohne Information Disclosure
- ☐ Sichere Repository-Methoden mit Prepared Statements

□ Integration aller Middleware in Pipeline

## 4.5.2 Lösung

### 4.5.2.1 CORS Middleware

```
// lib/middleware/cors_middleware.dart
import 'package:shelf/shelf.dart';

class CorsConfig {
  final List<String> allowedOrigins;
  final List<String> allowedMethods;
  final List<String> allowedHeaders;
  final bool allowCredentials;
  final Duration? maxAge;

  const CorsConfig({
    this.allowedOrigins = const ['*'],
    this.allowedMethods = const ['GET', 'POST', 'PUT', 'DELETE', 'PATCH', ↵
    ↵ 'OPTIONS'],
    this.allowedHeaders = const ['Content-Type', 'Authorization', 'Accept'],
    this.allowCredentials = false,
    this.maxAge,
  });

  static const development = CorsConfig();

  static CorsConfig production(List<String> origins) => CorsConfig(
    allowedOrigins: origins,
    allowCredentials: true,
    maxAge: const Duration(hours: 1),
  );
}

Middleware corsMiddleware(CorsConfig config) {
  return (Handler innerHandler) {
    return (Request request) async {
      final origin = request.headers['origin'];
      final corsHeaders = <String, String>{};

      // Origin prüfen und Header setzen
      if (origin != null) {
        if (config.allowedOrigins.contains('*')) {
          corsHeaders['Access-Control-Allow-Origin'] = '*';
        } else if (config.allowedOrigins.contains(origin)) {
          corsHeaders['Access-Control-Allow-Origin'] = origin;
          corsHeaders['Vary'] = 'Origin';
        } else {
          // Origin nicht erlaubt - keine CORS-Header
          if (request.method == 'OPTIONS') {
            return Response.forbidden('Origin not allowed');
          }
        }
      }
    };
  };
}
```

```
    }
    return await innerHandler(request);
  }
}

// Weitere CORS-Header
corsHeaders['Access-Control-Allow-Methods'] =
  config.allowedMethods.join(', ');
corsHeaders['Access-Control-Allow-Headers'] =
  config.allowedHeaders.join(', ');

if (config.allowCredentials) {
  corsHeaders['Access-Control-Allow-Credentials'] = 'true';
}

if (config.maxAge != null) {
  corsHeaders['Access-Control-Max-Age'] =
    config.maxAge!.inSeconds.toString();
}

// Preflight Request
if (request.method == 'OPTIONS') {
  return Response.ok('', headers: corsHeaders);
}

// Normaler Request mit CORS-Headers
final response = await innerHandler(request);
return response.change(headers: corsHeaders);
};
};
}
```

#### 4.5.2.2 Rate Limiter

```
// lib/services/rate_limiter.dart
import 'dart:async';

class RateLimitResult {
  final bool allowed;
  final int remaining;
  final int limit;
  final DateTime resetAt;

  RateLimitResult({
    required this.allowed,
    required this.remaining,
    required this.limit,
    required this.resetAt,
  });
}
```

```
int get retryAfterSeconds {
    final seconds = resetAt.difference(DateTime.now()).inSeconds;
    return seconds > 0 ? seconds : 0;
}

abstract class RateLimiter {
    Future<RateLimitResult> check({
        required String clientId,
        required int maxRequests,
        required Duration window,
    });
}

/// In-Memory Rate Limiter
class InMemoryRateLimiter implements RateLimiter {
    final Map<String, List<DateTime>> _requests = {};
    Timer? _cleanupTimer;

    InMemoryRateLimiter() {
        // Periodisches Cleanup
        _cleanupTimer = Timer.periodic(
            const Duration(minutes: 5),
            (_) => _cleanup(),
        );
    }

    void dispose() {
        _cleanupTimer?.cancel();
    }

    @override
    Future<RateLimitResult> check({
        required String clientId,
        required int maxRequests,
        required Duration window,
    }) async {
        final now = DateTime.now();
        final windowStart = now.subtract(window);

        // Requests für diesen Client
        final requests = _requests.putIfAbsent(clientId, () => []);

        // Alte Requests entfernen
        requests.removeWhere((time) => time.isBefore(windowStart));

        final resetAt = now.add(window);

        if (requests.length >= maxRequests) {
            // Rate Limit erreicht
            return RateLimitResult(
```

```

        allowed: false,
        remaining: 0,
        limit: maxRequests,
        resetAt: requests.isEmpty
            ? requests.first.add(window)
            : resetAt,
    );
}

// Request hinzufügen
requests.add(now);

return RateLimitResult(
    allowed: true,
    remaining: maxRequests - requests.length,
    limit: maxRequests,
    resetAt: resetAt,
);
}

void _cleanup() {
    final now = DateTime.now();
    final maxAge = const Duration(hours: 1);

    _requests.removeWhere((_, requests) {
        requests.removeWhere((time) =>
            now.difference(time) > maxAge);
        return requests.isEmpty;
    });
}

}

/// Redis Rate Limiter
class RedisRateLimiter implements RateLimiter {
    final dynamic _redis; // RedisClient

    RedisRateLimiter(this._redis);

    @override
    Future<RateLimitResult> check({
        required String clientId,
        required int maxRequests,
        required Duration window,
    }) async {
        final key = 'ratelimit:$clientId';
        final now = DateTime.now();
        final nowMs = now.millisecondsSinceEpoch;
        final windowStartMs = nowMs - window.inMilliseconds;

        // Sliding Window mit Sorted Set
        // 1. Alte Einträge entfernen

```

```
await _redis.zremrangebyscore(key, 0, windowStartMs.toDouble());

// 2. Aktuelle Anzahl
final count = await _redis.zcard(key) as int;

final resetAt = now.add(window);

if (count >= maxRequests) {
  return RateLimitResult(
    allowed: false,
    remaining: 0,
    limit: maxRequests,
    resetAt: resetAt,
  );
}

// 3. Request hinzufügen
await _redis.zadd(key, nowMs.toDouble(), nowMs.toString());

// 4. TTL setzen
await _redis.expire(key, window);

return RateLimitResult(
  allowed: true,
  remaining: maxRequests - count - 1,
  limit: maxRequests,
  resetAt: resetAt,
);
}
}
```

#### 4.5.2.3 Rate Limit Middleware

```
// lib/middleware/rate_limit_middleware.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';
import '../services/rate_limiter.dart';

Middleware rateLimitMiddleware({
  required RateLimiter rateLimiter,
  int maxRequests = 100,
  Duration window = const Duration(minutes: 1),
  String Function(Request)? keyExtractor,
}) {
  final getKey = keyExtractor ?? defaultKeyExtractor;

  return (Handler innerHandler) {
    return (Request request) async {
      final clientId = getKey(request);
```

```

        final result = await rateLimiter.check(
            clientId: clientId,
            maxRequests: maxRequests,
            window: window,
        );

        // Rate Limit Headers
        final headers = {
            'X-RateLimit-Limit': result.limit.toString(),
            'X-RateLimit-Remaining': result.remaining.toString(),
            'X-RateLimit-Reset': (result.resetAt.millisecondsSinceEpoch ~/
↪ 1000).toString(),
        };

        if (!result.allowed) {
            return Response(
                429,
                body: jsonEncode({
                    'error': 'Too many requests',
                    'retry_after': result.retryAfterSeconds,
                }),
                headers: {
                    ...headers,
                    'content-type': 'application/json',
                    'Retry-After': result.retryAfterSeconds.toString(),
                },
            );
        }

        final response = await innerHandler(request);
        return response.change(headers: headers);
    };
}

String defaultKeyExtractor(Request request) {
    // 1. Authentifizierter User
    final userId = request.context['userId'];
    if (userId != null) return 'user:$userId';

    // 2. X-Forwarded-For (erster Eintrag)
    final forwarded = request.headers['x-forwarded-for'];
    if (forwarded != null) {
        final ip = forwarded.split(',').first.trim();
        return 'ip:$ip';
    }

    // 3. X-Real-IP
    final realIp = request.headers['x-real-ip'];
    if (realIp != null) return 'ip:$realIp';
}

```

```
// 4. Fallback
return 'ip:unknown';
}
```

#### 4.5.2.4 Validator

```
// lib/validation/validator.dart

class ValidationResult {
  final bool isValid;
  final Map<String, List<String>> errors;

  ValidationResult(this.errors) : isValid = errors.isEmpty;

  Map<String, dynamic> toJson() => {
    'valid': isValid,
    if (!isValid) 'errors': errors,
  };
}

class Validator {
  final Map<String, List<String>> _errors = {};

  ValidationResult get result => ValidationResult(Map.from(_errors));
  bool get isValid => _errors.isEmpty;

  void addError(String field, String message) {
    _errors.putIfAbsent(field, () => []).add(message);
  }

  void clear() => _errors.clear();

  /// Validiere String-Feld
  String? string(
    String field,
    dynamic value, {
    bool required = true,
    int? minLength,
    int? maxLength,
    RegExp? pattern,
    String? patternMessage,
  }) {
    if (value == null || (value is String && value.trim().isEmpty)) {
      if (required) {
        addError(field, '$field is required');
      }
      return null;
    }

    if (value is! String) {
```

```
        addError(field, '$field must be a string');
        return null;
    }

    final trimmed = value.trim();

    if (minLength != null && trimmed.length < minLength) {
        addError(field, '$field must be at least $minLength characters');
    }

    if (maxLength != null && trimmed.length > maxLength) {
        addError(field, '$field must be at most $maxLength characters');
    }

    if (pattern != null && !pattern.hasMatch(trimmed)) {
        addError(field, patternMessage ?? '$field has invalid format');
    }

    return trimmed;
}

/// Validiere Email
String? email(String field, dynamic value, {bool required = true}) {
    return string(
        field,
        value,
        required: required,
        pattern: RegExp(r'^[\w\-\.\.]+\@([\w\-\.\.]+\w\-\.){2,4}$'),
        patternMessage: 'Invalid email format',
    );
}

/// Validiere Integer
int? integer(
    String field,
    dynamic value, {
    bool required = true,
    int? min,
    int? max,
}) {
    if (value == null) {
        if (required) {
            addError(field, '$field is required');
        }
        return null;
    }

    int? intValue;
    if (value is int) {
        intValue = value;
    } else if (value is String) {
```

```
        intValue = int.tryParse(value);
    }

    if (intValue == null) {
        addError(field, '$field must be a valid integer');
        return null;
    }

    if (min != null && intValue < min) {
        addError(field, '$field must be at least $min');
    }

    if (max != null && intValue > max) {
        addError(field, '$field must be at most $max');
    }

    return intValue;
}

/// Validiere Double
double? number(
    String field,
    dynamic value, {
    bool required = true,
    double? min,
    double? max,
}) {
    if (value == null) {
        if (required) {
            addError(field, '$field is required');
        }
        return null;
    }

    double? doubleValue;
    if (value is num) {
        doubleValue = value.toDouble();
    } else if (value is String) {
        doubleValue = double.tryParse(value);
    }

    if (doubleValue == null) {
        addError(field, '$field must be a valid number');
        return null;
    }

    if (min != null && doubleValue < min) {
        addError(field, '$field must be at least $min');
    }

    if (max != null && doubleValue > max) {
```

```
        addError(field, '$field must be at most $max');
    }

    return doubleValue;
}

/// Validiere Boolean
bool? boolean(String field, dynamic value, {bool required = true}) {
    if (value == null) {
        if (required) {
            addError(field, '$field is required');
        }
        return null;
    }

    if (value is bool) return value;

    if (value is String) {
        if (value.toLowerCase() == 'true') return true;
        if (value.toLowerCase() == 'false') return false;
    }

    addError(field, '$field must be a boolean');
    return null;
}

/// Validiere Liste
List<T>? list<T>(  
    String field,  
    dynamic value, {  
        bool required = true,  
        int? minLength,  
        int? maxLength,  
    }) {
    if (value == null) {
        if (required) {
            addError(field, '$field is required');
        }
        return null;
    }

    if (value is! List) {
        addError(field, '$field must be a list');
        return null;
    }

    if (minLength != null && value.length < minLength) {
        addError(field, '$field must have at least $minLength items');
    }

    if (maxLength != null && value.length > maxLength) {
```

```

        addError(field, '$field must have at most $maxLength items');
    }

    try {
        return value.cast<T>();
    } catch (e) {
        addError(field, '$field contains invalid items');
        return null;
    }
}

/// Validiere Enum
T? enumValue<T extends Enum>(
    String field,
    dynamic value,
    List<T> values, {
    bool required = true,
}) {
    if (value == null) {
        if (required) {
            addError(field, '$field is required');
        }
        return null;
    }

    final stringValue = value.toString();

    try {
        return values.firstWhere(
            (e) => e.name == stringValue || e.toString() == stringValue,
        );
    } catch (e) {
        addError(
            field,
            '$field must be one of: ${values.map((e) => e.name).join(", ")}' ,
        );
        return null;
    }
}
}

```

#### 4.5.2.5 Sanitizer

```

// lib/utils/sanitizer.dart

class Sanitizer {
    /// Entferne HTML-Tags
    static String stripHtml(String input) {
        return input.replaceAll(RegExp(r'<[^>]*>'), '');
    }
}

```

```
/// Escape HTML-Entities
static String escapeHtml(String input) {
    return input
        .replaceAll('&', '&amp;')
        .replaceAll('<', '&lt;')
        .replaceAll('>', '&gt;')
        .replaceAll('"', '&quot;')
        .replaceAll("'", '&#x27;');
}

/// Escape für SQL LIKE-Pattern
static String escapeLike(String input) {
    return input
        .replaceAll('\\', '\\\\')
        .replaceAll('%', '\\%')
        .replaceAll('_', '\\_');
}

/// Sanitize Dateiname
static String sanitizeFilename(String filename) {
    // Nur sichere Zeichen
    var safe = filename.replaceAll(RegExp(r'[^\w\-\.\.]'), '_');

    // Keine doppelten Punkte (Path Traversal)
    safe = safe.replaceAll(RegExp(r'\\.\\{2,}'), '.');

    // Keine führenden/trailing Punkte
    safe = safe.replaceAll(RegExp(r'^\\.|\\.$'), '');

    // Leerer Name?
    if (safe.isEmpty) safe = 'file';

    return safe;
}

/// Sanitize für Logging
static String sanitizeForLog(String input, {int maxLength = 200}) {
    var sanitized = input
        .replaceAll('\\n', '\\n')
        .replaceAll('\\r', '\\r')
        .replaceAll('\\t', '\\t');

    if (sanitized.length > maxLength) {
        sanitized = '${sanitized.substring(0, maxLength)}...';
    }

    return sanitized;
}

/// Normalize Whitespace
```

```

static String normalizeWhitespace(String input) {
  return input.trim().replaceAll(RegExp(r'\s+'), ' ');
}

/// URL-Parameter encoden
static String encodeUrlParam(String input) {
  return Uri.encodeComponent(input);
}

/// JSON-String escapen
static String escapeJson(String input) {
  return input
    .replaceAll('\\', '\\\\')
    .replaceAll('"', '\\"')
    .replaceAll('\n', '\\n')
    .replaceAll('\r', '\\r')
    .replaceAll('\t', '\\t');
}
}

```

#### 4.5.2.6 Security Headers Middleware

```

// lib/middleware/security_headers_middleware.dart
import 'package:shelf/shelf.dart';

class SecurityHeadersConfig {
  final bool noSniff;
  final String frameOptions;
  final bool xssProtection;
  final String referrerPolicy;
  final String? contentSecurityPolicy;
  final bool noCache;

  const SecurityHeadersConfig({
    this.noSniff = true,
    this.frameOptions = 'DENY',
    this.xssProtection = true,
    this.referrerPolicy = 'strict-origin-when-cross-origin',
    this.contentSecurityPolicy,
    this.noCache = true,
  });

  static const api = SecurityHeadersConfig(
    contentSecurityPolicy: "default-src 'none'",
  );
}

Middleware securityHeadersMiddleware([
  SecurityHeadersConfig config = const SecurityHeadersConfig(),
]) {

```

```

return (Handler innerHandler) {
  return (Request request) async {
    final response = await innerHandler(request);

    final headers = <String, String>{};

    if (config.noSniff) {
      headers['X-Content-Type-Options'] = 'nosniff';
    }

    headers['X-Frame-Options'] = config.frameOptions;

    if (config.xssProtection) {
      headers['X-XSS-Protection'] = '1; mode=block';
    }

    headers['Referrer-Policy'] = config.referrerPolicy;

    if (config.contentSecurityPolicy != null) {
      headers['Content-Security-Policy'] = config.contentSecurityPolicy!;
    }

    if (config.noCache) {
      headers['Cache-Control'] = 'no-store, no-cache, must-revalidate';
      headers['Pragma'] = 'no-cache';
    }

    return response.change(headers: headers);
  };
};
}

```

#### 4.5.2.7 Error Handler Middleware

```

// lib/middleware/error_handler_middleware.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';

class ApiException implements Exception {
  final String message;
  final int statusCode;

  ApiException(this.message, {this.statusCode = 400});
}

class ValidationException extends ApiException {
  final Map<String, List<String>> errors;

  ValidationException(this.errors)
    : super('Validation failed', statusCode: 400);
}

```

```

}

class NotFoundException extends ApiException {
  NotFoundException([String message = 'Not found'])
    : super(message, statusCode: 404);
}

Middleware errorHandlerMiddleware({
  required bool isDevelopment,
  void Function(Object error, StackTrace stack)? onError,
}) {
  return (Handler innerHandler) {
    return (Request request) async {
      try {
        return await innerHandler(request);
      } on ValidationException catch (e) {
        return _jsonResponse(400, {
          'error': e.message,
          'errors': e.errors,
        });
      } on ApiException catch (e) {
        return _jsonResponse(e.statusCode, {'error': e.message});
      } catch (e, stack) {
        // Loggen
        onError?.call(e, stack);
        print('Error: $e\n$stack');

        // Response
        if (isDevelopment) {
          return _jsonResponse(500, {
            'error': e.toString(),
            'stack': stack.toString().split('\n').take(10).toList(),
          });
        }

        return _jsonResponse(500, {'error': 'Internal server error'});
      }
    };
  };
}

Response _jsonResponse(int statusCode, Map<String, dynamic> body) {
  return Response(
    statusCode,
    body: jsonEncode(body),
    headers: {'content-type': 'application/json'},
  );
}

```

#### 4.5.2.8 Sichere Repository Methoden

```
// lib/repositories/secure_product_repository.dart
import 'package:postgres/postgres.dart';
import '../utils/sanitizer.dart';

class ProductRepository {
  final Connection _db;

  static const _allowedSortColumns = {'id', 'name', 'price', 'created_at'};
  static const _maxLimit = 100;

  ProductRepository(this._db);

  Future<List<Product>> search(String query) async {
    // Query sanitizen
    final escapedQuery = Sanitizer.escapeLike(query.trim());

    if (escapedQuery.isEmpty) {
      return [];
    }

    final result = await _db.execute(
      Sql.named('''
        SELECT * FROM products
        WHERE name ILIKE @pattern OR description ILIKE @pattern
        LIMIT 50
      '''),
      parameters: {'pattern': '%$escapedQuery%'},
    );

    return result.map((r) => Product.fromRow(r.toColumnMap())).toList();
  }

  Future<List<Product>> findAll({
    String sortBy = 'created_at',
    bool descending = true,
    int limit = 20,
    int offset = 0,
  }) async {
    // Whitelist für Sortierung
    if (!_allowedSortColumns.contains(sortBy)) {
      sortBy = 'created_at';
    }

    // Limits validieren
    limit = limit.clamp(1, _maxLimit);
    offset = offset.clamp(0, 10000);

    final direction = descending ? 'DESC' : 'ASC';
```

```
// Spalte ist gewhitelistet - sicher
final result = await _db.execute(
  Sql.named('''
    SELECT * FROM products
    ORDER BY $sortBy $direction
    LIMIT @limit OFFSET @offset
  '''),
  parameters: {'limit': limit, 'offset': offset},
);

return result.map((r) => Product.fromRow(r.toColumnMap())).toList();
}
}
```

### 4.5.3 Ressourcen

#### 4.5.3.1 Offizielle Dokumentation

- OWASP Top 10
- OWASP API Security Top 10
- MDN CORS
- Security Headers

#### 4.5.3.2 Cheat Sheet: CORS Headers

```
// Preflight Response (OPTIONS)
headers = {
  'Access-Control-Allow-Origin': 'https://example.com',
  'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE',
  'Access-Control-Allow-Headers': 'Content-Type, Authorization',
  'Access-Control-Allow-Credentials': 'true',
  'Access-Control-Max-Age': '3600', // 1 Stunde cachен
};

// Wichtig: Bei Credentials niemals '*' als Origin!
// FALSCH:
'Access-Control-Allow-Origin': '*',
'Access-Control-Allow-Credentials': 'true',

// RICHTIG:
'Access-Control-Allow-Origin': 'https://myapp.com',
'Access-Control-Allow-Credentials': 'true',
'Vary': 'Origin', // Wichtig für Caching
```

#### 4.5.3.3 Cheat Sheet: Rate Limiting

```
// Response Headers
'X-RateLimit-Limit': '100',           // Max Requests
'X-RateLimit-Remaining': '95',       // Verbleibend
'X-RateLimit-Reset': '1672531200',   // Unix Timestamp
```

```
// Bei 429 Too Many Requests
'Retry-After': '60', // Sekunden bis Retry

// Algorithmen:
// - Fixed Window: Einfach, aber Burst am Window-Ende
// - Sliding Window: Glatter, aber mehr Speicher
// - Token Bucket: Erlaubt Bursts bis zu Limit
// - Leaky Bucket: Konstante Rate
```

#### 4.5.3.4 Cheat Sheet: Security Headers

```
// Basis-Set für APIs
headers = {
  // Verhindert MIME-Type Sniffing
  'X-Content-Type-Options': 'nosniff',

  // Verhindert Clickjacking
  'X-Frame-Options': 'DENY',

  // XSS-Filter (Legacy)
  'X-XSS-Protection': '1; mode=block',

  // Referrer einschränken
  'Referrer-Policy': 'strict-origin-when-cross-origin',

  // Content Security Policy
  'Content-Security-Policy': "default-src 'none'",

  // Kein Caching für API-Responses
  'Cache-Control': 'no-store',
  'Pragma': 'no-cache',
};

// Für HTTPS (nach TLS-Setup)
'Strict-Transport-Security': 'max-age=31536000; includeSubDomains',
```

#### 4.5.3.5 Cheat Sheet: Input Validation

```
// String
final name = validator.string('name', data['name'],
  required: true,
  minLength: 2,
  maxLength: 100,
  pattern: RegExp(r'^[\w\s\.-]+$'),
);

// Email
final email = validator.email('email', data['email']);
```

```
// Integer mit Range
final age = validator.integer('age', data['age'],
    min: 0,
    max: 150,
);

// Enum
final status = validator.enumValue('status', data['status'],
    Status.values,
);

// Liste
final tags = validator.list<String>('tags', data['tags'],
    minLength: 1,
    maxLength: 10,
);
```

#### 4.5.3.6 Cheat Sheet: SQL Injection Prevention

```
// □ SICHER: Prepared Statements
await db.execute(
    Sql.named('SELECT * FROM users WHERE email = @email'),
    parameters: {'email': userInput},
);

// □ SICHER: LIKE mit Escaping
final escaped = input
    .replaceAll('\\', '\\\\')
    .replaceAll('%', '\\%')
    .replaceAll('_', '\\_');
await db.execute(
    Sql.named('SELECT * FROM products WHERE name ILIKE @pattern'),
    parameters: {'pattern': '%$escaped%'},
);

// □ SICHER: Whitelist für dynamische Spalten
const allowed = {'id', 'name', 'created_at'};
final column = allowed.contains(input) ? input : 'created_at';
await db.execute('SELECT * FROM users ORDER BY $column');

// □ UNSICHER: String Concatenation
await db.execute("SELECT * FROM users WHERE email = '$userInput'");
```

#### 4.5.3.7 Cheat Sheet: XSS Prevention

```
// HTML escapen (für HTML-Output)
String escapeHtml(String input) {
    return input
```

```

        .replaceAll('&', '&amp;')
        .replaceAll('<', '&lt;')
        .replaceAll('>', '&gt;')
        .replaceAll('"', '&quot;')
        .replaceAll("'", '&#x27;');
    }

    // HTML-Tags entfernen
    String stripHtml(String input) {
        return input.replaceAll(RegExp(r'<[^>]*>'), '');
    }

    // Content-Type richtig setzen
    Response.ok(
        jsonEncode(data),
        headers: {'content-type': 'application/json'}, // Nicht text/html!
    );

```

#### 4.5.3.8 Cheat Sheet: HTTP Status Codes für Errors

Code	Name	Verwendung
400	Bad Request	Ungültige Anfrage/Validierung
401	Unauthorized	Nicht authentifiziert
403	Forbidden	Keine Berechtigung
404	Not Found	Ressource nicht gefunden
409	Conflict	Konflikt (Duplikat)
422	Unprocessable	Semantisch ungültig
429	Too Many Requests	Rate Limit
500	Internal Error	Server-Fehler

#### 4.5.3.9 Best Practices

##### DO

1. **Prepared Statements** - Immer für SQL
2. **Input validieren** - Serverseitig, nie nur Client
3. **Output escapen** - Kontextabhängig (HTML, JSON, SQL)
4. **Rate Limiting** - Pro User und global
5. **Security Headers** - Für alle Responses
6. **HTTPS** - In Produktion Pflicht
7. **Generische Fehler** - Keine internen Details leaken
8. **Logging** - Aber sensitive Daten maskieren

##### DON'T

1. **String Concatenation für SQL** - Nie!
2. **Sensitive Daten in URLs** - Tokens, Passwörter
3. **Detaillierte Fehlermeldungen** - In Produktion
4. **\*\*CORS \*mit Credentials\*** - Sicherheitslücke
5. **Secrets im Code** - Environment Variables nutzen
6. **User-Input vertrauen** - Nie!

## 7. Security by Obscurity - Kein Ersatz für echte Sicherheit

### 4.5.3.10 Sicherheits-Checkliste

#### Vor dem Deployment

- [ ] Alle Environment Variables gesetzt
- [ ] Secrets sicher gespeichert (nicht im Repo)
- [ ] HTTPS konfiguriert
- [ ] CORS auf erlaubte Origins beschränkt
- [ ] Rate Limiting aktiviert
- [ ] Security Headers gesetzt
- [ ] Alle SQL-Queries nutzen Prepared Statements
- [ ] Input-Validierung für alle Endpoints
- [ ] Fehler-Responses ohne interne Details
- [ ] Logging ohne sensitive Daten
- [ ] Dependencies auf Vulnerabilities geprüft

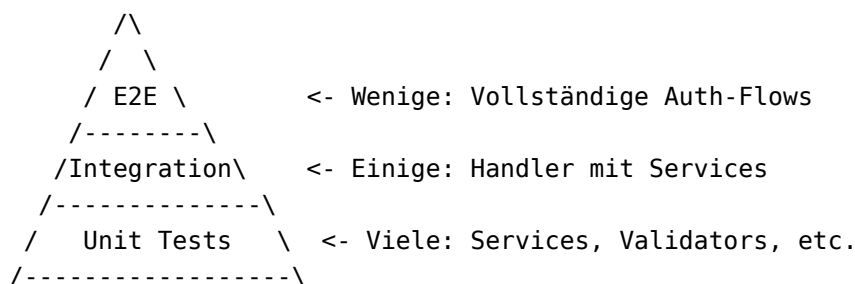
## 4.6 Einheit 8.6: Auth Testing

### 4.6.0.1 Lernziele

Nach dieser Einheit kannst du: - Unit Tests für Auth-Komponenten schreiben - Integration Tests für Auth-Flows erstellen - Mocks für Auth-Services implementieren - Test-Fixtures für User und Tokens nutzen

### 4.6.0.2 Test-Strategie für Auth

Test-Pyramide



Was testen?

Komponente	Test-Fokus
<b>PasswordService</b>	Hashing, Verifizierung
<b>JwtService</b>	Token-Generierung, Validierung
<b>AuthService</b>	Login, Register, Refresh
<b>Auth Middleware</b>	Token-Extraktion, Context
<b>Guards</b>	Rollen, Permissions, Owner
<b>Auth Handler</b>	HTTP-Requests, Responses

#### 4.6.0.3 Unit Tests: Password Service

```
// test/services/password_service_test.dart
import 'package:test/test.dart';
import 'package:my_api/services/password_service.dart';

void main() {
  group('PasswordService', () {
    late PasswordService service;

    setUp(() {
      // Niedriger Cost für schnelle Tests
      service = PasswordService(costFactor: 4);
    });

    group('hash', () {
      test('generates valid bcrypt hash', () {
        final hash = service.hash('password123');

        expect(hash, startsWith(r'$2'));
        expect(hash.length, greaterThan(50));
      });

      test('generates different hashes for same password', () {
        final hash1 = service.hash('password');
        final hash2 = service.hash('password');

        expect(hash1, isNot(equals(hash2)));
      });
    });

    group('verify', () {
      test('returns true for correct password', () {
        final hash = service.hash('secret123');

        expect(service.verify('secret123', hash), isTrue);
      });

      test('returns false for wrong password', () {
        final hash = service.hash('secret123');

        expect(service.verify('wrong', hash), isFalse);
      });

      test('returns false for invalid hash format', () {
        expect(service.verify('password', 'invalid'), isFalse);
      });
    });

    group('needsRehash', () {
      test('returns true when cost factor increased', () {

```

```
    final lowCostService = PasswordService(costFactor: 4);
    final hash = lowCostService.hash('password');

    final highCostService = PasswordService(costFactor: 10);
    expect(highCostService.needsRehash(hash), isTrue);
  });

  test('returns false when cost factor same or higher', () {
    final hash = service.hash('password');
    expect(service.needsRehash(hash), isFalse);
  });
});
});
}
```

#### 4.6.0.4 Unit Tests: JWT Service

```
// test/services/jwt_service_test.dart
import 'package:test/test.dart';
import 'package:my_api/services/jwt_service.dart';
import 'package:my_api/models/user.dart';

void main() {
  group('JwtService', () {
    late JwtService jwtService;
    late User testUser;

    setUp(() {
      jwtService = JwtService(
        secret: 'test-secret-key-that-is-at-least-32-characters',
        accessTokenDuration: const Duration(minutes: 15),
        refreshTokenDuration: const Duration(days: 7),
      );

      testUser = User(
        id: 1,
        email: 'test@example.com',
        passwordHash: 'hash',
        name: 'Test User',
        role: 'user',
      );
    });

    group('generateAccessToken', () {
      test('creates valid JWT', () {
        final token = jwtService.generateAccessToken(testUser);

        expect(token, isEmpty);
        expect(token.split('.').length, equals(3));
      });
    });
  });
}
```

```
test('contains correct claims', () {
  final token = jwtService.generateAccessToken(testUser);
  final payload = jwtService.verifyToken(token);

  expect(payload.userId, equals(1));
  expect(payload.email, equals('test@example.com'));
  expect(payload.role, equals('user'));
  expect(payload.isAccessToken, isTrue);
});

group('generateRefreshToken', () {
  test('creates token with refresh type', () {
    final token = jwtService.generateRefreshToken(testUser);
    final payload = jwtService.verifyToken(token);

    expect(payload.isRefreshToken, isTrue);
    expect(payload.userId, equals(1));
    expect(payload.email, isNull); // Minimal payload
  });
});

group('verifyToken', () {
  test('validates correct token', () {
    final token = jwtService.generateAccessToken(testUser);

    expect(() => jwtService.verifyToken(token), returnsNormally);
  });

  test('throws for invalid token', () {
    expect(
      () => jwtService.verifyToken('invalid.token.here'),
      throwsA(isA<InvalidTokenException>()),
    );
  });

  test('throws for expired token', () async {
    final shortLivedService = JwtService(
      secret: 'test-secret-key-that-is-at-least-32-characters',
      accessTokenDuration: const Duration(milliseconds: 1),
    );

    final token = shortLivedService.generateAccessToken(testUser);
    await Future.delayed(const Duration(milliseconds: 10));

    expect(
      () => shortLivedService.verifyToken(token),
      throwsA(isA<TokenExpiredException>()),
    );
  });
});
```

```

test('throws for wrong secret', () {
  final token = jwtService.generateAccessToken(testUser);

  final otherService = JwtService(
    secret: 'different-secret-key-that-is-also-32-chars',
  );

  expect(
    () => otherService.verifyToken(token),
    throwsA(isA<InvalidTokenException>()),
  );
});

group('extractTokenFromHeader', () {
  test('extracts Bearer token', () {
    final token = jwtService.extractTokenFromHeader('Bearer abc123');
    expect(token, equals('abc123'));
  });

  test('returns null for missing header', () {
    expect(jwtService.extractTokenFromHeader(null), isNull);
  });

  test('returns null for wrong scheme', () {
    expect(jwtService.extractTokenFromHeader('Basic abc'), isNull);
  });
});
});
}

```

#### 4.6.0.5 Unit Tests: Auth Service

```

// test/services/auth_service_test.dart
import 'package:test/test.dart';
import 'package:mocktail/mocktail.dart';

// Mocks
class MockUserRepository extends Mock implements UserRepository {}
class MockPasswordService extends Mock implements PasswordService {}
class MockJwtService extends Mock implements JwtService {}
class MockRefreshTokenRepository extends Mock implements
  ↪ RefreshTokenRepository {}

void main() {
  group('AuthService', () {
    late AuthService authService;
    late MockUserRepository userRepo;
    late MockPasswordService passwordService;

```

↪

```
late MockJwtService jwtService;
late MockRefreshTokenRepository refreshTokenRepo;

setUp(() {
  userRepo = MockUserRepository();
  passwordService = MockPasswordService();
  jwtService = MockJwtService();
  refreshTokenRepo = MockRefreshTokenRepository();

  authService = AuthService(
    userRepo,
    passwordService,
    jwtService,
    refreshTokenRepo,
  );
});

group('login', () {
  final testUser = User(
    id: 1,
    email: 'test@example.com',
    passwordHash: 'hashed',
    isActive: true,
  );

  test('returns tokens for valid credentials', () async {
    when(() => userRepo.findByEmail(any()))
      .thenAnswer((_) async => testUser);
    when(() => passwordService.verify(any(), any()))
      .thenReturn(true);
    when(() => jwtService.generateTokenPair(any()))
      .thenReturn(TokenPair(
        accessToken: 'access',
        refreshToken: 'refresh',
        expiresIn: 900,
      ));
    when(() => jwtService.refreshTokenDuration)
      .thenReturn(const Duration(days: 7));
    when(() => refreshTokenRepo.create(any()))
      .thenAnswer((_) async {});

    final result = await authService.login(
      email: 'test@example.com',
      password: 'password',
    );

    expect(result.success, isTrue);
    expect(result.tokens, isNotNull);
    expect(result.user?.id, equals(1));
  });
});
```

```
test('returns error for non-existent user', () async {
  when(() => userRepo.findByEmail(any()))
    .thenAnswer((_) async => null);
  when(() => passwordService.hash(any()))
    .thenReturn('dummy'); // Timing attack prevention

  final result = await authService.login(
    email: 'nonexistent@example.com',
    password: 'password',
  );

  expect(result.success, isFalse);
  expect(result.statusCode, equals(401));
  expect(result.error, contains('Invalid credentials'));
});

test('returns error for wrong password', () async {
  when(() => userRepo.findByEmail(any()))
    .thenAnswer((_) async => testUser);
  when(() => passwordService.verify(any(), any()))
    .thenReturn(false);

  final result = await authService.login(
    email: 'test@example.com',
    password: 'wrong',
  );

  expect(result.success, isFalse);
  expect(result.statusCode, equals(401));
});

test('returns error for inactive user', () async {
  final inactiveUser = User(
    id: 1,
    email: 'test@example.com',
    passwordHash: 'hashed',
    isActive: false,
  );

  when(() => userRepo.findByEmail(any()))
    .thenAnswer((_) async => inactiveUser);

  final result = await authService.login(
    email: 'test@example.com',
    password: 'password',
  );

  expect(result.success, isFalse);
  expect(result.statusCode, equals(403));
});
});
```

```

group('refreshAccessToken', () {
  test('returns new tokens for valid refresh token', () async {
    final payload = JwtPayload(
      sub: '1',
      type: 'refresh',
    );

    when(() => jwtService.verifyToken(any()))
      .thenReturn(payload);
    when(() => refreshTokenRepo.findByToken(any()))
      .thenAnswer((_) async => RefreshTokenRecord(
        userId: 1,
        token: 'old-refresh',
        expiresAt: DateTime.now().add(const Duration(days: 1)),
      ));
    when(() => userRepo.findById(any()))
      .thenAnswer((_) async => User(
        id: 1,
        email: 'test@example.com',
        passwordHash: 'hash',
        isActive: true,
      ));
    when(() => refreshTokenRepo.revoke(any()))
      .thenAnswer((_) async {});
    when(() => jwtService.generateTokenPair(any()))
      .thenReturn(TokenPair(
        accessToken: 'new-access',
        refreshToken: 'new-refresh',
        expiresIn: 900,
      ));
    when(() => jwtService.refreshTokenDuration)
      .thenReturn(const Duration(days: 7));
    when(() => refreshTokenRepo.create(any()))
      .thenAnswer((_) async {});

    final tokens = await authService.refreshAccessToken('old-refresh');

    expect(tokens.accessToken, equals('new-access'));
    verify(() => refreshTokenRepo.revoke('old-refresh')).called(1);
  });

  test('throws for revoked token', () async {
    final payload = JwtPayload(sub: '1', type: 'refresh');

    when(() => jwtService.verifyToken(any()))
      .thenReturn(payload);
    when(() => refreshTokenRepo.findByToken(any()))
      .thenAnswer((_) async => null);

    expect(

```

```

        () => authService.refreshAccessToken('revoked-token'),
        throwsA(isA<RefreshTokenRevokedException>()),
    );
  });
});
});
}

```

#### 4.6.0.6 Integration Tests: Auth Handler

```

// test/handlers/auth_handler_test.dart
import 'dart:convert';
import 'package:test/test.dart';
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';

void main() {
  group('AuthHandler Integration', () {
    late Handler handler;
    late TestDatabase testDb;

    setUpAll(() async {
      testDb = await TestDatabase.create();
      await testDb.migrate();

      final userRepo = PostgresUserRepository(testDb.connection);
      final passwordService = PasswordService(costFactor: 4);
      final jwtService = JwtService(secret: 'test-secret-32-chars-long-key');
      final refreshTokenRepo = RefreshTokenRepository(testDb.connection);
      final authService = AuthService(
        userRepo,
        passwordService,
        jwtService,
        refreshTokenRepo,
      );

      final authHandler = AuthHandler(authService);
      handler = authHandler.router;
    });

    tearDownAll(() async {
      await testDb.close();
    });

    setUp(() async {
      await testDb.truncate(['users', 'refresh_tokens']);
    });

    group('POST /register', () {
      test('creates user with valid data', () async {

```

```
    final request = _postRequest('/register', {
      'email': 'new@example.com',
      'password': 'SecurePass123!',
      'name': 'New User',
    });

    final response = await handler(request);
    final body = await _parseBody(response);

    expect(response.statusCode, equals(201));
    expect(body['user']['email'], equals('new@example.com'));
    expect(body['user']['name'], equals('New User'));
    expect(body['user'], isNot(contains('passwordHash')));
  });

  test('rejects weak password', () async {
    final request = _postRequest('/register', {
      'email': 'test@example.com',
      'password': 'weak',
    });

    final response = await handler(request);

    expect(response.statusCode, equals(400));
  });

  test('rejects duplicate email', () async {
    // Ersten User erstellen
    await handler(_postRequest('/register', {
      'email': 'existing@example.com',
      'password': 'SecurePass123!',
    }));

    // Duplikat versuchen
    final response = await handler(_postRequest('/register', {
      'email': 'existing@example.com',
      'password': 'AnotherPass123!',
    }));

    expect(response.statusCode, equals(409));
  });
});

group('POST /login', () {
  setUp(() async {
    // Test-User erstellen
    await handler(_postRequest('/register', {
      'email': 'user@example.com',
      'password': 'Password123!',
    }));
  });
});
```

```
test('returns tokens for valid credentials', () async {
  final request = _postRequest('/login', {
    'email': 'user@example.com',
    'password': 'Password123!',
  });

  final response = await handler(request);
  final body = await _parseBody(response);

  expect(response.statusCode, equals(200));
  expect(body['access_token'], isEmpty);
  expect(body['refresh_token'], isEmpty);
  expect(body['expires_in'], isA<int>());
  expect(body['token_type'], equals('Bearer'));
});

test('rejects invalid password', () async {
  final request = _postRequest('/login', {
    'email': 'user@example.com',
    'password': 'WrongPassword',
  });

  final response = await handler(request);

  expect(response.statusCode, equals(401));
});

group('POST /refresh', () {
  late String refreshToken;

  setUp(() async {
    await handler(_postRequest('/register', {
      'email': 'refresh@example.com',
      'password': 'Password123!',
    }));

    final loginResponse = await handler(_postRequest('/login', {
      'email': 'refresh@example.com',
      'password': 'Password123!',
    }));

    final body = await _parseBody(loginResponse);
    refreshToken = body['refresh_token'];
  });

  test('returns new tokens for valid refresh token', () async {
    final request = _postRequest('/refresh', {
      'refresh_token': refreshToken,
    });
  });
});
```

```

    final response = await handler(request);
    final body = await _parseBody(response);

    expect(response.statusCode, equals(200));
    expect(body['access_token'], isEmpty);
    expect(body['refresh_token'], isEmpty);
    expect(body['refresh_token'], isNot(equals(refreshToken))); // Rotation
  });

  test('rejects reused refresh token', () async {
    // Ersten Refresh
    await handler(_postRequest('/refresh', {
      'refresh_token': refreshToken,
    }));

    // Zweiter Versuch mit gleichem Token
    final response = await handler(_postRequest('/refresh', {
      'refresh_token': refreshToken,
    }));

    expect(response.statusCode, equals(401));
  });
});
}

// Helper Funktionen
Request _postRequest(String path, Map<String, dynamic> body) {
  return Request(
    'POST',
    Uri.parse('http://localhost$path'),
    body: jsonEncode(body),
    headers: {'content-type': 'application/json'},
  );
}

Future<Map<String, dynamic>> _parseBody(Response response) async {
  final body = await response.readAsString();
  return jsonDecode(body) as Map<String, dynamic>;
}

```

#### 4.6.0.7 Test-Fixtures

```

// test/fixtures/auth_fixtures.dart

class AuthFixtures {
  static User createUser({
    int? id,
    String? email,

```

```
String? passwordHash,
String? name,
String role = 'user',
bool isActive = true,
}) {
  return User(
    id: id ?? 1,
    email: email ?? 'test@example.com',
    passwordHash: passwordHash ?? 'hashed_password',
    name: name ?? 'Test User',
    role: role,
    isActive: isActive,
  );
}

static User createAdmin({int? id, String? email}) {
  return createUser(
    id: id ?? 99,
    email: email ?? 'admin@example.com',
    role: 'admin',
  );
}

static TokenPair createTokenPair({
  String? accessToken,
  String? refreshToken,
  int expiresIn = 900,
}) {
  return TokenPair(
    accessToken: accessToken ?? 'test-access-token',
    refreshToken: refreshToken ?? 'test-refresh-token',
    expiresIn: expiresIn,
  );
}

static JwtPayload createPayload({
  int userId = 1,
  String? email,
  String? role,
  String type = 'access',
}) {
  return JwtPayload(
    sub: userId.toString(),
    email: email ?? 'test@example.com',
    role: role ?? 'user',
    type: type,
  );
}

static Request createAuthenticatedRequest(
  String method,
```

```
String path, {
  String? accessToken,
  Map<String, dynamic>? body,
}) {
  return Request(
    method,
    Uri.parse('http://localhost$path'),
    headers: {
      'authorization': 'Bearer ${accessToken ?? "test-token"}',
      if (body != null) 'content-type': 'application/json',
    },
    body: body != null ? jsonEncode(body) : null,
  );
}
```

#### 4.6.0.8 Mock Auth Middleware

```
// test/helpers/mock_auth_middleware.dart

/// Middleware für Tests die Auth simuliert
Middleware mockAuthMiddleware({
  int? userId,
  String? email,
  String? role,
  bool isAuthenticated = true,
}) {
  return (Handler innerHandler) {
    return (Request request) async {
      if (!isAuthenticated) {
        return innerHandler(request);
      }

      final payload = JwtPayload(
        sub: (userId ?? 1).toString(),
        email: email ?? 'test@example.com',
        role: role ?? 'user',
        type: 'access',
      );

      final updatedRequest = request.change(
        context: {
          ...request.context,
          'auth': payload,
          'userId': payload.userId,
          'userRole': payload.role,
        },
      );

      return innerHandler(updatedRequest);
    }
  }
}
```

```

    };
  };
}

// Verwendung in Tests:
test('protected endpoint works with auth', () async {
  final handler = const Pipeline()
    .addMiddleware(mockAuthMiddleware(userId: 1, role: 'admin'))
    .addHandler(protectedHandler);

  final response = await handler(Request('GET', Uri.parse('/protected')));
  expect(response.statusCode, equals(200));
});

```

#### 4.6.0.9 Test Database

```

// test/helpers/test_database.dart
import 'package:postgres/postgres.dart';

class TestDatabase {
  final Connection connection;

  TestDatabase._(this.connection);

  static Future<TestDatabase> create() async {
    final connection = await Connection.open(
      Endpoint(
        host: 'localhost',
        database: 'test_db',
        username: 'postgres',
        password: 'postgres',
      ),
      settings: ConnectionSettings(sslMode: SslMode.disable),
    );

    return TestDatabase._(connection);
  }

  Future<void> migrate() async {
    await connection.execute('''
      CREATE TABLE IF NOT EXISTS users (
        id SERIAL PRIMARY KEY,
        email VARCHAR(255) NOT NULL UNIQUE,
        password_hash VARCHAR(255) NOT NULL,
        name VARCHAR(255),
        role VARCHAR(50) DEFAULT 'user',
        is_active BOOLEAN DEFAULT true,
        created_at TIMESTAMP DEFAULT NOW()
      );
    ''');
  }
}

```

```

CREATE TABLE IF NOT EXISTS refresh_tokens (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
  token TEXT NOT NULL UNIQUE,
  expires_at TIMESTAMP NOT NULL,
  is_revoked BOOLEAN DEFAULT false,
  created_at TIMESTAMP DEFAULT NOW()
);
''');
}

Future<void> truncate(List<String> tables) async {
  for (final table in tables.reversed) {
    await connection.execute('TRUNCATE TABLE $table CASCADE');
  }
}

Future<void> close() async {
  await connection.close();
}
}

```

#### 4.6.0.10 Zusammenfassung

Test-Typ	Fokus	Tools
<b>Unit Tests</b>	Einzelne Services	mocktail, test
<b>Integration</b>	Handler + Services	TestDatabase
<b>E2E</b>	Vollständige Flows	HTTP Client
<b>Fixtures</b>	Test-Daten	Helper Classes
<b>Mocks</b>	Dependencies	mocktail

### 4.6.1 Übung

#### 4.6.1.1 Ziel

Erstelle eine umfassende Test-Suite für die Auth-Komponenten.

#### 4.6.1.2 Vorbereitung

Dependencies

```

dev_dependencies:
  test: ^1.24.0
  mocktail: ^1.0.0

```

Test-Datenbank

```

# Test-Datenbank erstellen
docker exec -it postgres psql -U postgres -c "CREATE DATABASE test_db;"

```

#### 4.6.1.3 Aufgabe 1: Password Service Tests (15 min)

```
// test/services/password_service_test.dart

void main() {
  group('PasswordService', () {
    late PasswordService service;

    setUp(() {
      service = PasswordService(costFactor: 4); // Niedrig für schnelle Tests
    });

    group('hash', () {
      test('generates valid bcrypt hash', () {
        // TODO: Hash generieren und Format prüfen
      });

      test('generates different hashes for same password (salt)', () {
        // TODO: Zwei Hashes vergleichen
      });
    });

    group('verify', () {
      test('returns true for correct password', () {
        // TODO
      });

      test('returns false for wrong password', () {
        // TODO
      });

      test('handles invalid hash gracefully', () {
        // TODO: verify mit ungültigem Hash
      });
    });

    group('needsRehash', () {
      test('returns true when cost factor increased', () {
        // TODO: Mit verschiedenen Cost Factors testen
      });
    });
  });
}
```

#### 4.6.1.4 Aufgabe 2: JWT Service Tests (20 min)

```
// test/services/jwt_service_test.dart

void main() {
  group('JwtService', () {
    late JwtService jwtService;
```

```
late User testUser;

setUp() {
    jwtService = JwtService(
        secret: 'test-secret-key-that-is-at-least-32-characters',
        accessTokenDuration: Duration(minutes: 15),
        refreshTokenDuration: Duration(days: 7),
    );

    testUser = User(
        id: 1,
        email: 'test@example.com',
        passwordHash: 'hash',
        name: 'Test',
        role: 'user',
    );
});

group('generateAccessToken', () {
    test('creates valid JWT format', () {
        // TODO: Token generieren, Format prüfen (3 Teile)
    });

    test('includes correct claims', () {
        // TODO: Token generieren, verifizieren, Claims prüfen
    });

    test('sets correct token type', () {
        // TODO: isAccessToken == true
    });
});

group('generateRefreshToken', () {
    test('creates token with minimal claims', () {
        // TODO: Nur sub und type, kein email/name
    });

    test('sets refresh token type', () {
        // TODO: isRefreshToken == true
    });
});

group('verifyToken', () {
    test('validates correct token', () {
        // TODO
    });

    test('throws InvalidTokenException for malformed token', () {
        // TODO
    });
});
```

```

    test('throws TokenExpiredException for expired token', () {
      // TODO: Token mit kurzer Duration generieren, warten
    });

    test('throws for token signed with different secret', () {
      // TODO
    });
  });

  group('generateTokenPair', () {
    test('returns both tokens', () {
      // TODO
    });

    test('sets correct expiresIn', () {
      // TODO: expiresIn entspricht accessTokenDuration
    });
  });

  group('extractTokenFromHeader', () {
    test('extracts Bearer token', () {
      // TODO
    });

    test('returns null for null header', () {
      // TODO
    });

    test('returns null for wrong scheme', () {
      // TODO: 'Basic xyz'
    });

    test('returns null for malformed header', () {
      // TODO: 'Bearer' ohne Token
    });
  });
});
}

```

#### 4.6.1.5 Aufgabe 3: Auth Service Tests mit Mocks (25 min)

```

// test/services/auth_service_test.dart
import 'package:mocktail/mocktail.dart';

class MockUserRepository extends Mock implements UserRepository {}
class MockPasswordService extends Mock implements PasswordService {}
class MockJwtService extends Mock implements JwtService {}
class MockRefreshTokenRepository extends Mock implements
  ↪ RefreshTokenRepository {}

```

```
void main() {
    group('AuthService', () {
        late AuthService authService;
        late MockUserRepository userRepo;
        late MockPasswordService passwordService;
        late MockJwtService jwtService;
        late MockRefreshTokenRepository refreshTokenRepo;

        setUp(() {
            userRepo = MockUserRepository();
            passwordService = MockPasswordService();
            jwtService = MockJwtService();
            refreshTokenRepo = MockRefreshTokenRepository();

            authService = AuthService(
                userRepo,
                passwordService,
                jwtService,
                refreshTokenRepo,
            );
        });

        group('login', () {
            test('returns tokens for valid credentials', () async {
                // TODO:
                // 1. Mocks konfigurieren
                // 2. login aufrufen
                // 3. Ergebnis prüfen
                // 4. Verify dass Methoden aufgerufen wurden
            });

            test('returns error for non-existent user', () async {
                // TODO: userRepo.findByEmail returns null
            });

            test('returns error for wrong password', () async {
                // TODO: passwordService.verify returns false
            });

            test('returns error for inactive user', () async {
                // TODO: User mit isActive=false
            });

            test('prevents timing attack on non-existent user', () async {
                // TODO: Verify dass passwordService.hash aufgerufen wird
                // auch wenn User nicht existiert
            });
        });

        group('refreshAccessToken', () {
            test('returns new tokens for valid refresh token', () async {
```

```
    // TODO
  });

  test('revokes old token (rotation)', () async {
    // TODO: verify refreshTokenRepo.revoke called
  });

  test('throws for access token', () async {
    // TODO: Token mit type='access' sollte fehlschlagen
  });

  test('throws for revoked token', () async {
    // TODO: refreshTokenRepo.findByToken returns null
  });

  test('throws for inactive user', () async {
    // TODO
  });
});

group('logout', () {
  test('revokes refresh token', () async {
    // TODO: verify refreshTokenRepo.revoke called
  });
});
}
```

#### 4.6.1.6 Aufgabe 4: Auth Middleware Tests (20 min)

```
// test/middleware/auth_middleware_test.dart

void main() {
  group('authMiddleware', () {
    late JwtService jwtService;
    late Handler testHandler;
    late int? capturedUserId;

    setUp(() {
      jwtService = JwtService(
        secret: 'test-secret-key-that-is-long-enough-32',
      );

      capturedUserId = null;

      final innerHandler = (Request request) {
        capturedUserId = getUserId(request);
        return Response.ok('OK');
      };
    });
  });
}
```

```
    testHandler = const Pipeline()
      .addMiddleware(authMiddleware(jwtService))
      .addHandler(innerHandler);
  });

  test('rejects request without Authorization header', () async {
    // TODO
  });

  test('rejects request with invalid token', () async {
    // TODO
  });

  test('rejects request with expired token', () async {
    // TODO: Token mit abgelaufener Duration
  });

  test('rejects refresh token', () async {
    // TODO: Refresh Token sollte 401 geben
  });

  test('accepts valid access token', () async {
    // TODO
  });

  test('adds userId to context', () async {
    // TODO: capturedUserId prüfen
  });

  test('adds auth payload to context', () async {
    // TODO: getAuthPayload(request) prüfen
  });
});
}
```

#### 4.6.1.7 Aufgabe 5: Role Middleware Tests (15 min)

```
// test/middleware/role_middleware_test.dart

void main() {
  group('requireRole', () {
    late JwtService jwtService;

    setUp(() {
      jwtService = JwtService(secret: 'test-secret-32-chars-long-key');
    });

    Handler createHandler(Role requiredRole) {
      return const Pipeline()
        .addMiddleware(authMiddleware(jwtService))
```

```

        .addMiddleware(requireRole(requiredRole))
        .addHandler((r) => Response.ok('OK'));
    }

    test('allows user with sufficient role', () async {
        // TODO: Admin-Token für admin-Route
    });

    test('rejects user with insufficient role', () async {
        // TODO: User-Token für admin-Route -> 403
    });

    test('returns 401 without authentication', () async {
        // TODO: Kein Token -> 401
    });

    test('respects role hierarchy', () async {
        // TODO: Superadmin kann Admin-Route nutzen
    });
});

group('requirePermission', () {
    test('allows user with required permission', () async {
        // TODO
    });

    test('rejects user without permission', () async {
        // TODO
    });

    test('superadmin has all permissions', () async {
        // TODO
    });
});
}

```

#### 4.6.1.8 Aufgabe 6: Integration Tests (20 min)

```

// test/integration/auth_flow_test.dart

void main() {
    group('Auth Flow Integration', () {
        late TestDatabase testDb;
        late Handler handler;

        setUpAll(() async {
            testDb = await TestDatabase.create();
            await testDb.migrate();
            // Services und Handler aufsetzen...
        });
    });
}

```

```
tearDownAll(() async {
  await testDb.close();
});

setUp(() async {
  await testDb.truncate(['users', 'refresh_tokens']);
});

test('complete registration flow', () async {
  // TODO:
  // 1. POST /register
  // 2. Prüfen: 201, User-Daten in Response
  // 3. Prüfen: User in DB
});

test('complete login flow', () async {
  // TODO:
  // 1. User registrieren
  // 2. POST /login
  // 3. Prüfen: Tokens in Response
});

test('complete token refresh flow', () async {
  // TODO:
  // 1. Registrieren + Login
  // 2. POST /refresh mit refresh_token
  // 3. Prüfen: Neue Tokens
  // 4. Prüfen: Alter Token invalidiert
});

test('protected endpoint with valid token', () async {
  // TODO:
  // 1. Login
  // 2. GET /protected mit access_token
  // 3. Prüfen: 200 OK
});

test('protected endpoint without token', () async {
  // TODO: 401
});

test('protected endpoint with expired token', () async {
  // TODO: 401
});
});
```

**4.6.1.9 Aufgabe 7: Test Fixtures erstellen (10 min)**

```
// test/fixtures/auth_fixtures.dart

class AuthFixtures {
  /// Test-User erstellen
  static User createUser({
    int? id,
    String? email,
    String role = 'user',
    bool isActive = true,
  }) {
    // TODO
  }

  /// Admin-User erstellen
  static User createAdmin({int? id}) {
    // TODO
  }

  /// Token-Paar erstellen
  static TokenPair createTokenPair() {
    // TODO
  }

  /// JWT Payload erstellen
  static JwtPayload createPayload({
    int userId = 1,
    String type = 'access',
    String role = 'user',
  }) {
    // TODO
  }

  /// Authenticated Request erstellen
  static Request createAuthRequest(
    String method,
    String path, {
    String? token,
    Map<String, dynamic>? body,
  }) {
    // TODO
  }
}
```

**4.6.1.10 Aufgabe 8: Mock Auth Middleware (10 min)**

```
// test/helpers/mock_auth.dart

/// Middleware die Auth simuliert ohne echten JWT
Middleware mockAuthMiddleware({
```

```
int userId = 1,
String email = 'test@example.com',
String role = 'user',
}) {
  // TODO:
  // Payload erstellen und in Context setzen
  // Ohne echte Token-Validierung
}

/// Request mit Auth-Context erstellen
Request withAuth(
  Request request, {
    int userId = 1,
    String role = 'user',
  }) {
  // TODO:
  // request.change mit Auth-Context
}
```

#### 4.6.1.11 Testen

```
# Alle Tests
dart test

# Nur Auth-Tests
dart test test/services/auth_service_test.dart

# Mit Coverage
dart test --coverage=coverage
genhtml coverage/lcov.info -o coverage/html
open coverage/html/index.html
```

#### 4.6.1.12 Abgabe-Checkliste

- ☐ PasswordService Tests (hash, verify, needsRehash)
- ☐ JwtService Tests (generate, verify, extract)
- ☐ AuthService Tests mit Mocks (login, refresh, logout)
- ☐ Auth Middleware Tests (Token-Validierung, Context)
- ☐ Role Middleware Tests (Hierarchie, Permissions)
- ☐ Integration Tests (vollständige Flows)
- ☐ Test Fixtures für User, Tokens
- ☐ Mock Auth Middleware für Handler-Tests
- ☐ Alle Tests grün

### 4.6.2 Lösung

#### 4.6.2.1 Password Service Tests

```
// test/services/password_service_test.dart
import 'package:test/test.dart';
```

```
import 'package:my_api/services/password_service.dart';

void main() {
  group('PasswordService', () {
    late PasswordService service;

    setUp(() {
      service = PasswordService(costFactor: 4);
    });

    group('hash', () {
      test('generates valid bcrypt hash', () {
        final hash = service.hash('password123');

        expect(hash, startsWith(r'$2'));
        expect(hash.length, greaterThan(50));
        expect(hash.contains(r'$'), isTrue);
      });

      test('generates different hashes for same password (salt)', () {
        final hash1 = service.hash('samePassword');
        final hash2 = service.hash('samePassword');

        expect(hash1, isNot(equals(hash2)));

        // Aber beide sollten verifizierbar sein
        expect(service.verify('samePassword', hash1), isTrue);
        expect(service.verify('samePassword', hash2), isTrue);
      });

      test('hash contains cost factor', () {
        final serviceHighCost = PasswordService(costFactor: 10);
        final hash = serviceHighCost.hash('password');

        // Format: $2a$10$...
        expect(hash, contains(r'$10$'));
      });
    });

    group('verify', () {
      test('returns true for correct password', () {
        final hash = service.hash('correctPassword');
        expect(service.verify('correctPassword', hash), isTrue);
      });

      test('returns false for wrong password', () {
        final hash = service.hash('correctPassword');
        expect(service.verify('wrongPassword', hash), isFalse);
      });

      test('is case sensitive', () {
```

```
    final hash = service.hash('Password');
    expect(service.verify('password', hash), isFalse);
    expect(service.verify('PASSWORD', hash), isFalse);
  });

  test('handles invalid hash gracefully', () {
    expect(service.verify('password', 'invalid'), isFalse);
    expect(service.verify('password', ''), isFalse);
    expect(service.verify('password', 'short'), isFalse);
  });

  test('handles empty password', () {
    final hash = service.hash('');
    expect(service.verify('', hash), isTrue);
    expect(service.verify('something', hash), isFalse);
  });
});

group('needsRehash', () {
  test('returns true when cost factor increased', () {
    final lowCostService = PasswordService(costFactor: 4);
    final hash = lowCostService.hash('password');

    final highCostService = PasswordService(costFactor: 10);
    expect(highCostService.needsRehash(hash), isTrue);
  });

  test('returns false when cost factor same', () {
    final hash = service.hash('password');
    expect(service.needsRehash(hash), isFalse);
  });

  test('returns false when cost factor lower', () {
    final highCostService = PasswordService(costFactor: 10);
    final hash = highCostService.hash('password');

    final lowCostService = PasswordService(costFactor: 4);
    expect(lowCostService.needsRehash(hash), isFalse);
  });

  test('returns true for invalid hash', () {
    expect(service.needsRehash('invalid'), isTrue);
  });
});
});
```

#### 4.6.2.2 JWT Service Tests

```
// test/services/jwt_service_test.dart
import 'package:test/test.dart';
import 'package:my_api/services/jwt_service.dart';
import 'package:my_api/models/user.dart';
import 'package:my_api/exceptions/auth_exceptions.dart';

void main() {
  group('JwtService', () {
    late JwtService jwtService;
    late User testUser;

    setUp(() {
      jwtService = JwtService(
        secret: 'test-secret-key-that-is-at-least-32-characters',
        accessTokenDuration: const Duration(minutes: 15),
        refreshTokenDuration: const Duration(days: 7),
      );

      testUser = User(
        id: 42,
        email: 'test@example.com',
        passwordHash: 'hash',
        name: 'Test User',
        role: 'moderator',
      );
    });

    group('generateAccessToken', () {
      test('creates valid JWT format', () {
        final token = jwtService.generateAccessToken(testUser);
        final parts = token.split('.');

        expect(parts.length, equals(3));
        expect(parts[0], isEmpty); // Header
        expect(parts[1], isEmpty); // Payload
        expect(parts[2], isEmpty); // Signature
      });

      test('includes correct claims', () {
        final token = jwtService.generateAccessToken(testUser);
        final payload = jwtService.verifyToken(token);

        expect(payload.userId, equals(42));
        expect(payload.email, equals('test@example.com'));
        expect(payload.name, equals('Test User'));
        expect(payload.role, equals('moderator'));
      });

      test('sets correct token type', () {
        final token = jwtService.generateAccessToken(testUser);
        final payload = jwtService.verifyToken(token);
```

```
        expect(payload.isAccessToken, isTrue);
        expect(payload.isRefreshToken, isFalse);
    });
});

group('generateRefreshToken', () {
    test('creates token with minimal claims', () {
        final token = jwtService.generateRefreshToken(testUser);
        final payload = jwtService.verifyToken(token);

        expect(payload.userId, equals(42));
        expect(payload.email, isNull);
        expect(payload.name, isNull);
    });

    test('sets refresh token type', () {
        final token = jwtService.generateRefreshToken(testUser);
        final payload = jwtService.verifyToken(token);

        expect(payload.isRefreshToken, isTrue);
        expect(payload.isAccessToken, isFalse);
    });
});

group('verifyToken', () {
    test('validates correct token', () {
        final token = jwtService.generateAccessToken(testUser);

        expect(() => jwtService.verifyToken(token), returnsNormally);
    });

    test('throws InvalidTokenException for malformed token', () {
        expect(
            () => jwtService.verifyToken('not.a.valid.token'),
            throwsA(isA<InvalidTokenException>()),
        );

        expect(
            () => jwtService.verifyToken(''),
            throwsA(isA<InvalidTokenException>()),
        );

        expect(
            () => jwtService.verifyToken('abc'),
            throwsA(isA<InvalidTokenException>()),
        );
    });

    test('throws TokenExpiredException for expired token', () async {
        final shortLivedService = JwtService(
```

```
        secret: 'test-secret-key-that-is-at-least-32-characters',
        accessTokenDuration: const Duration(milliseconds: 1),
    );

    final token = shortLivedService.generateAccessToken(testUser);

    // Warten bis Token abläuft
    await Future.delayed(const Duration(milliseconds: 50));

    expect(
      () => shortLivedService.verifyToken(token),
      throwsA(isA<TokenExpiredException>()),
    );
  });

  test('throws for token signed with different secret', () {
    final token = jwtService.generateAccessToken(testUser);

    final otherService = JwtService(
      secret: 'different-secret-that-is-also-32-characters',
    );

    expect(
      () => otherService.verifyToken(token),
      throwsA(isA<InvalidTokenException>()),
    );
  });
});

group('generateTokenPair', () {
  test('returns both tokens', () {
    final pair = jwtService.generateTokenPair(testUser);

    expect(pair.accessToken, isNotEmpty);
    expect(pair.refreshToken, isNotEmpty);
    expect(pair.accessToken, isNot(equals(pair.refreshToken)));
  });

  test('sets correct expiresIn', () {
    final pair = jwtService.generateTokenPair(testUser);

    expect(pair.expiresIn, equals(15 * 60)); // 15 Minuten in Sekunden
  });

  test('access token is verifiable', () {
    final pair = jwtService.generateTokenPair(testUser);
    final payload = jwtService.verifyToken(pair.accessToken);

    expect(payload.isAccessToken, isTrue);
  });
});
```

```

    test('refresh token is verifiable', () {
      final pair = jwtService.generateTokenPair(testUser);
      final payload = jwtService.verifyToken(pair.refreshToken);

      expect(payload.isRefreshToken, isTrue);
    });
  });

  group('extractTokenFromHeader', () {
    test('extracts Bearer token', () {
      final token = jwtService.extractTokenFromHeader('Bearer abc123xyz');
      expect(token, equals('abc123xyz'));
    });

    test('handles real JWT', () {
      final realToken = jwtService.generateAccessToken(testUser);
      final extracted = jwtService.extractTokenFromHeader('Bearer $realToken');
      expect(extracted, equals(realToken));
    });

    test('returns null for null header', () {
      expect(jwtService.extractTokenFromHeader(null), isNull);
    });

    test('returns null for empty header', () {
      expect(jwtService.extractTokenFromHeader(''), isNull);
    });

    test('returns null for wrong scheme', () {
      expect(jwtService.extractTokenFromHeader('Basic abc'), isNull);
      expect(jwtService.extractTokenFromHeader('Token abc'), isNull);
    });

    test('returns null for malformed header', () {
      expect(jwtService.extractTokenFromHeader('Bearer'), isNull);
      expect(jwtService.extractTokenFromHeader('Bearer '), isNull);
    });

    test('is case sensitive for scheme', () {
      expect(jwtService.extractTokenFromHeader('bearer abc'), isNull);
      expect(jwtService.extractTokenFromHeader('BEARER abc'), isNull);
    });
  });
}

```

#### 4.6.2.3 Auth Service Tests mit Mocks

```

// test/services/auth_service_test.dart
import 'package:test/test.dart';

```

```
import 'package:mocktail/mocktail.dart';

class MockUserRepository extends Mock implements UserRepository {}
class MockPasswordService extends Mock implements PasswordService {}
class MockJwtService extends Mock implements JwtService {}
class MockRefreshTokenRepository extends Mock implements ↵
  ↵ RefreshTokenRepository {}

class FakeUser extends Fake implements User {}
class FakeRefreshTokenRecord extends Fake implements RefreshTokenRecord {}

void main() {
  setUpAll(() {
    registerFallbackValue(FakeUser());
    registerFallbackValue(FakeRefreshTokenRecord());
  });

  group('AuthService', () {
    late AuthService authService;
    late MockUserRepository userRepo;
    late MockPasswordService passwordService;
    late MockJwtService jwtService;
    late MockRefreshTokenRepository refreshTokenRepo;

    final testUser = User(
      id: 1,
      email: 'test@example.com',
      passwordHash: 'hashed_password',
      name: 'Test',
      role: 'user',
      isActive: true,
    );

    final testTokenPair = TokenPair(
      accessToken: 'access_token',
      refreshToken: 'refresh_token',
      expiresIn: 900,
    );

    setUp(() {
      userRepo = MockUserRepository();
      passwordService = MockPasswordService();
      jwtService = MockJwtService();
      refreshTokenRepo = MockRefreshTokenRepository();

      authService = AuthService(
        userRepo,
        passwordService,
        jwtService,
        refreshTokenRepo,
      );
    });
  });
}
```

```

// Default Mocks
when(() => jwtService.refreshTokenDuration)
  .thenReturn(const Duration(days: 7));
});

group('login', () {
  test('returns tokens for valid credentials', () async {
    when(() => userRepo.findByEmail(any()))
      .thenAnswer((_) async => testUser);
    when(() => passwordService.verify(any(), any()))
      .thenReturn(true);
    when(() => jwtService.generateTokenPair(any()))
      .thenReturn(testTokenPair);
    when(() => refreshTokenRepo.create(any()))
      .thenAnswer((_) async {});

    final result = await authService.login(
      email: 'test@example.com',
      password: 'password123',
    );

    expect(result.success, isTrue);
    expect(result.tokens?.accessToken, equals('access_token'));
    expect(result.user?.id, equals(1));

    verify(() => userRepo.findByEmail('test@example.com')).called(1);
    verify(() => passwordService.verify('password123',
↵ 'hashed_password')).called(1);
    verify(() => refreshTokenRepo.create(any())).called(1);
  });

  test('returns error for non-existent user', () async {
    when(() => userRepo.findByEmail(any()))
      .thenAnswer((_) async => null);
    when(() => passwordService.hash(any()))
      .thenReturn('dummy_hash');

    final result = await authService.login(
      email: 'nonexistent@example.com',
      password: 'password',
    );

    expect(result.success, isFalse);
    expect(result.statusCode, equals(401));
    expect(result.error, contains('Invalid credentials'));
  });

  test('prevents timing attack on non-existent user', () async {
    when(() => userRepo.findByEmail(any()))
      .thenAnswer((_) async => null);

```

```
when(() => passwordService.hash(any()))
    .thenReturn('dummy');

await authService.login(
    email: 'nonexistent@example.com',
    password: 'password',
);

// Verify hash wird aufgerufen auch bei nicht-existent User
verify(() => passwordService.hash(any())).called(1);
});

test('returns error for wrong password', () async {
    when(() => userRepo.findByEmail(any()))
        .thenAnswer((_) async => testUser);
    when(() => passwordService.verify(any(), any()))
        .thenReturn(false);

    final result = await authService.login(
        email: 'test@example.com',
        password: 'wrong',
    );

    expect(result.success, isFalse);
    expect(result.statusCode, equals(401));
});

test('returns error for inactive user', () async {
    final inactiveUser = User(
        id: 1,
        email: 'test@example.com',
        passwordHash: 'hash',
        isActive: false,
    );

    when(() => userRepo.findByEmail(any()))
        .thenAnswer((_) async => inactiveUser);

    final result = await authService.login(
        email: 'test@example.com',
        password: 'password',
    );

    expect(result.success, isFalse);
    expect(result.statusCode, equals(403));
});

group('refreshAccessToken', () {
    test('returns new tokens for valid refresh token', () async {
        final payload = JwtPayload(sub: '1', type: 'refresh');
```

```
when(() => jwtService.verifyToken(any()))
    .thenReturn(payload);
when(() => refreshTokenRepo.findByToken(any()))
    .thenAnswer((_) async => RefreshTokenRecord(
        userId: 1,
        token: 'old_refresh',
        expiresAt: DateTime.now().add(const Duration(days: 1)),
    ));
when(() => userRepo.findById(any()))
    .thenAnswer((_) async => testUser);
when(() => refreshTokenRepo.revoke(any()))
    .thenAnswer((_) async {});
when(() => jwtService.generateTokenPair(any()))
    .thenReturn(testTokenPair);
when(() => refreshTokenRepo.create(any()))
    .thenAnswer((_) async {});

final tokens = await authService.refreshAccessToken('old_refresh');

expect(tokens.accessToken, equals('access_token'));
verify(() => refreshTokenRepo.revoke('old_refresh')).called(1);
});

test('throws for access token', () async {
    final payload = JwtPayload(sub: '1', type: 'access');

    when(() => jwtService.verifyToken(any()))
        .thenReturn(payload);

    expect(
        () => authService.refreshAccessToken('access_token'),
        throwsA(isA<TokenTypeMismatchException>()),
    );
});

test('throws for revoked token', () async {
    final payload = JwtPayload(sub: '1', type: 'refresh');

    when(() => jwtService.verifyToken(any()))
        .thenReturn(payload);
    when(() => refreshTokenRepo.findByToken(any()))
        .thenAnswer((_) async => null);

    expect(
        () => authService.refreshAccessToken('revoked'),
        throwsA(isA<RefreshTokenRevokedException>()),
    );
});
});
```

```

group('logout', () {
  test('revokes refresh token', () async {
    when(() => refreshTokenRepo.revoke(any()))
      .thenAnswer((_) async {});

    await authService.logout('refresh_token');

    verify(() => refreshTokenRepo.revoke('refresh_token')).called(1);
  });
});

group('logoutAllSessions', () {
  test('revokes all user tokens', () async {
    when(() => refreshTokenRepo.revokeAllForUser(any()))
      .thenAnswer((_) async {});

    await authService.logoutAllSessions(1);

    verify(() => refreshTokenRepo.revokeAllForUser(1)).called(1);
  });
});
});
}

```

#### 4.6.2.4 Auth Middleware Tests

```

// test/middleware/auth_middleware_test.dart
import 'package:test/test.dart';
import 'package:shelf/shelf.dart';

void main() {
  group('authMiddleware', () {
    late JwtService jwtService;
    late User testUser;
    late int? capturedUserId;
    late JwtPayload? capturedPayload;

    setUp(() {
      jwtService = JwtService(
        secret: 'test-secret-key-that-is-long-enough-32',
      );

      testUser = User(
        id: 123,
        email: 'test@example.com',
        passwordHash: 'hash',
        role: 'admin',
      );

      capturedUserId = null;
    });
  });
}

```

```
    capturedPayload = null;
  });

  Handler createHandler() {
    return const Pipeline()
      .addMiddleware(authMiddleware(jwtService))
      .addHandler((Request request) {
        capturedUserId = getUserId(request);
        capturedPayload = getAuthPayload(request);
        return Response.ok('OK');
      });
  }

  test('rejects request without Authorization header', () async {
    final handler = createHandler();
    final request = Request('GET', Uri.parse('http://localhost/test'));

    final response = await handler(request);

    expect(response.statusCode, equals(401));
  });

  test('rejects request with invalid token', () async {
    final handler = createHandler();
    final request = Request(
      'GET',
      Uri.parse('http://localhost/test'),
      headers: {'authorization': 'Bearer invalid-token'},
    );

    final response = await handler(request);

    expect(response.statusCode, equals(401));
  });

  test('rejects refresh token', () async {
    final refreshToken = jwtService.generateRefreshToken(testUser);
    final handler = createHandler();
    final request = Request(
      'GET',
      Uri.parse('http://localhost/test'),
      headers: {'authorization': 'Bearer $refreshToken'},
    );

    final response = await handler(request);

    expect(response.statusCode, equals(401));
  });

  test('accepts valid access token', () async {
    final accessToken = jwtService.generateAccessToken(testUser);
```

```

    final handler = createHandler();
    final request = Request(
      'GET',
      Uri.parse('http://localhost/test'),
      headers: {'authorization': 'Bearer $accessToken'},
    );

    final response = await handler(request);

    expect(response.statusCode, equals(200));
  });

  test('adds userId to context', () async {
    final accessToken = jwtService.generateAccessToken(testUser);
    final handler = createHandler();
    final request = Request(
      'GET',
      Uri.parse('http://localhost/test'),
      headers: {'authorization': 'Bearer $accessToken'},
    );

    await handler(request);

    expect(capturedUserId, equals(123));
  });

  test('adds auth payload to context', () async {
    final accessToken = jwtService.generateAccessToken(testUser);
    final handler = createHandler();
    final request = Request(
      'GET',
      Uri.parse('http://localhost/test'),
      headers: {'authorization': 'Bearer $accessToken'},
    );

    await handler(request);

    expect(capturedPayload, isNotNull);
    expect(capturedPayload!.userId, equals(123));
    expect(capturedPayload!.email, equals('test@example.com'));
    expect(capturedPayload!.role, equals('admin'));
  });
});
}

```

#### 4.6.2.5 Test Fixtures

```

// test/fixtures/auth_fixtures.dart
import 'dart:convert';
import 'package:shelf/shelf.dart';

```

```
class AuthFixtures {
    static User createUser({
        int? id,
        String? email,
        String? passwordHash,
        String? name,
        String role = 'user',
        bool isActive = true,
    }) {
        return User(
            id: id ?? 1,
            email: email ?? 'test${id ?? 1}@example.com',
            passwordHash: passwordHash ?? 'hashed_password',
            name: name ?? 'Test User ${id ?? 1}',
            role: role,
            isActive: isActive,
        );
    }

    static User createAdmin({int? id, String? email}) {
        return createUser(
            id: id ?? 99,
            email: email ?? 'admin@example.com',
            role: 'admin',
        );
    }

    static User createSuperadmin({int? id}) {
        return createUser(
            id: id ?? 100,
            email: 'superadmin@example.com',
            role: 'superadmin',
        );
    }

    static TokenPair createTokenPair({
        String? accessToken,
        String? refreshToken,
        int expiresIn = 900,
    }) {
        return TokenPair(
            accessToken: accessToken ??
↪ 'test-access-token-${DateTime.now().millisecondsSinceEpoch}',
            refreshToken: refreshToken ??
↪ 'test-refresh-token-${DateTime.now().millisecondsSinceEpoch}',
            expiresIn: expiresIn,
        );
    }

    static JwtPayload createPayload({
```

```

    int userId = 1,
    String? email,
    String? name,
    String role = 'user',
    String type = 'access',
  }) {
    return JwtPayload(
      sub: userId.toString(),
      email: email ?? 'test@example.com',
      name: name,
      role: role,
      type: type,
    );
  }

  static Request createAuthRequest(
    String method,
    String path, {
    String? token,
    Map<String, dynamic>? body,
    Map<String, String>? headers,
  }) {
    return Request(
      method,
      Uri.parse('http://localhost$path'),
      headers: {
        if (token != null) 'authorization': 'Bearer $token',
        if (body != null) 'content-type': 'application/json',
        ...?headers,
      },
      body: body != null ? jsonEncode(body) : null,
    );
  }

  static Request createPostRequest(String path, Map<String, dynamic> body) {
    return Request(
      'POST',
      Uri.parse('http://localhost$path'),
      headers: {'content-type': 'application/json'},
      body: jsonEncode(body),
    );
  }
}

```

#### 4.6.2.6 Mock Auth Middleware

```

// test/helpers/mock_auth.dart
import 'package:shelf/shelf.dart';

/// Middleware die Auth simuliert ohne echten JWT

```

```
Middleware mockAuthMiddleware({
  int userId = 1,
  String email = 'test@example.com',
  String? name,
  String role = 'user',
}) {
  return (Handler innerHandler) {
    return (Request request) async {
      final payload = JwtPayload(
        sub: userId.toString(),
        email: email,
        name: name,
        role: role,
        type: 'access',
      );

      final updatedRequest = request.change(
        context: {
          ...request.context,
          'auth': payload,
          'userId': userId,
          'userRole': role,
        },
      );

      return innerHandler(updatedRequest);
    };
  };
}
```

/// Request mit Auth-Context erweitern

```
Request withAuth(
  Request request, {
    int userId = 1,
    String email = 'test@example.com',
    String role = 'user',
  }) {
  final payload = JwtPayload(
    sub: userId.toString(),
    email: email,
    role: role,
    type: 'access',
  );

  return request.change(
    context: {
      ...request.context,
      'auth': payload,
      'userId': userId,
      'userRole': role,
    },
  );
}
```

```
);  
}  
  
/// Middleware die immer 401 zurückgibt  
Middleware denyAuthMiddleware() {  
  return (Handler innerHandler) {  
    return (Request request) async {  
      return Response(401, body: '{"error": "Unauthorized"}');  
    };  
  };  
}
```

### 4.6.3 Ressourcen

#### 4.6.3.1 Offizielle Dokumentation

- Dart test Package
- mocktail Package
- shelf Testing

#### 4.6.3.2 Cheat Sheet: Test Struktur

```
void main() {  
  group('ComponentName', () {  
    late ComponentType component;  
  
    // Einmalig vor allen Tests  
    setUpAll(() async {  
      // DB-Connection, etc.  
    });  
  
    // Vor jedem Test  
    setUp(() {  
      component = ComponentType();  
    });  
  
    // Nach jedem Test  
    tearDown(() {  
      // Cleanup  
    });  
  
    // Einmalig nach allen Tests  
    tearDownAll(() async {  
      // DB schließen, etc.  
    });  
  
    group('methodName', () {  
      test('does something specific', () {  
        // Arrange  
        // Act  
        // Assert  
      });  
    });  
  });  
}
```

```
    });  
  });  
});  
}
```

#### 4.6.3.3 Cheat Sheet: Matchers

```
// Gleichheit  
expect(value, equals(expected));  
expect(value, isNot(equals(other)));  
  
// Typen  
expect(value, isA<String>());  
expect(value, isNotNull);  
expect(value, isNull);  
  
// Strings  
expect(str, startsWith('prefix'));  
expect(str, endsWith('suffix'));  
expect(str, contains('substring'));  
  
// Listen  
expect(list, isEmpty);  
expect(list, isNotEmpty);  
expect(list, hasLength(3));  
expect(list, contains(item));  
  
// Zahlen  
expect(num, greaterThan(5));  
expect(num, lessThan(10));  
expect(num, inInclusiveRange(5, 10));  
  
// Exceptions  
expect(() => fn(), throwsA(isA<MyException>()));  
expect(() => fn(), throwsException);  
expect(() => fn(), returnsNormally);  
  
// Async  
expect(future, completion(equals(value)));  
expect(future, throwsA(isA<Exception>()));
```

#### 4.6.3.4 Cheat Sheet: Mocktail

```
// Mock erstellen  
class MockService extends Mock implements Service {}  
  
// Fallback registrieren (für any())  
setUpAll(() {  
  registerFallbackValue(FakeUser());  
});
```

```
});

// Verhalten definieren
when(() => mock.method(any())).thenReturn(value);
when(() => mock.asyncMethod(any())).thenAnswer((_) async => value);
when(() => mock.method(any())).thenThrow(Exception());

// Aufrufe verifizieren
verify(() => mock.method(any())).called(1);
verify(() => mock.method('specific')).called(1);
verifyNever(() => mock.method(any()));

// Argument Capture
final captured = verify(() => mock.method(captureAny())).captured;

// Reset
reset(mock);
```

#### 4.6.3.5 Cheat Sheet: Shelf Request Testing

```
// Request erstellen
final request = Request(
  'POST',
  Uri.parse('http://localhost/api/resource'),
  headers: {
    'content-type': 'application/json',
    'authorization': 'Bearer token',
  },
  body: jsonEncode({'key': 'value'}),
);

// Response prüfen
final response = await handler(request);
expect(response.statusCode, equals(200));

final body = await response.readAsString();
final json = jsonDecode(body);
expect(json['key'], equals('value'));

// Headers prüfen
expect(response.headers['content-type'], contains('application/json'));
```

#### 4.6.3.6 Cheat Sheet: Test Database

```
class TestDatabase {
  late Connection connection;

  Future<void> setUp() async {
    connection = await Connection.open(
```

```

    Endpoint(
        host: 'localhost',
        database: 'test_db',
        username: 'test',
        password: 'test',
    ),
);
}

Future<void> migrate() async {
    await connection.execute('''
        CREATE TABLE IF NOT EXISTS ...
    ''');
}

Future<void> truncate(List<String> tables) async {
    for (final table in tables.reversed) {
        await connection.execute('TRUNCATE $table CASCADE');
    }
}

Future<void> tearDown() async {
    await connection.close();
}
}

```

#### 4.6.3.7 Cheat Sheet: Auth Test Helpers

```

// Token für Tests generieren
String createTestToken(JwtService jwt, {int userId = 1, String role = 'user'}) {
    final user = User(id: userId, email: 'test@test.com', passwordHash: '',
    ↪ role: role);
    return jwt.generateAccessToken(user);
}

// Authenticated Request
Request authRequest(String method, String path, String token, {Map<String,
    ↪ dynamic>? body}) {
    return Request(
        method,
        Uri.parse('http://localhost$path'),
        headers: {
            'authorization': 'Bearer $token',
            if (body != null) 'content-type': 'application/json',
        },
        body: body != null ? jsonEncode(body) : null,
    );
}

// Response Body parsen

```

```
Future<Map<String, dynamic>> parseJson(Response response) async {
  return jsonDecode(await response.readAsString());
}
```

#### 4.6.3.8 Best Practices

##### DO

1. **Isolierte Tests** - Jeder Test unabhängig
2. **Schnelle Unit Tests** - Niedrige Cost Factors
3. **Mocks für externe Dependencies**
4. **Fixtures für Testdaten**
5. **Descriptive Test Names**
6. **Test-Datenbank für Integration Tests**
7. **Cleanup nach Tests**

##### DON'T

1. **Tests abhängig voneinander**
2. **Produktions-Datenbank in Tests**
3. **Hardcoded Delays** (außer für Expiry-Tests)
4. **Secrets in Test-Code**
5. **Zu viele Integration Tests**

#### 4.6.3.9 Test-Pyramide für Auth

```

      /\
    /E2E\      Login -> API Call -> Logout
  /-----\
 /Integr.\    Handler + Service + DB
/-----\
/ Unit Tests \ PasswordService, JwtService, etc.
/-----\
```

#### 4.6.3.10 Coverage

```
# Tests mit Coverage
dart test --coverage=coverage

# HTML-Report generieren
dart pub global activate coverage
format_coverage --lcov --in=coverage --out=coverage/lcov.info
genhtml coverage/lcov.info -o coverage/html

# Report öffnen
open coverage/html/index.html
```

# Chapter 5

## Block 9: Produktion & Abschlussprojekt

WebSockets, Background Jobs, Logging, Deployment und das Backend-Abschlussprojekt.

### 5.1 Einheit 9.1: WebSockets & Real-time

#### 5.1.0.1 Lernziele

- WebSocket-Protokoll verstehen
- Bidirektionale Kommunikation implementieren
- shelf\_web\_socket Package verwenden
- Real-time Features entwickeln

#### 5.1.0.2 WebSocket-Grundlagen

Was sind WebSockets?

WebSockets ermöglichen bidirektionale, persistente Verbindungen zwischen Client und Server:

HTTP (Request-Response):

Client --Request--> Server

Client <--Response-- Server

(Verbindung geschlossen)

WebSocket (Bidirektional):

Client ===== Server

<--Messages-->

(Verbindung bleibt offen)

Vorteile gegenüber HTTP

Aspekt	HTTP Polling	WebSocket
Latenz	Hoch (Request-Overhead)	Niedrig (persistente Verbindung)
Overhead	Groß (Header bei jedem Request)	Klein (nur bei Handshake)
Server -> Client	Nur als Response	Jederzeit
Skalierung	Viele Connections	Weniger Connections

Wann WebSockets?

**Gute Anwendungsfälle:** - Chat-Anwendungen - Live-Notifications - Collaborative Editing - Real-time Dashboards - Online Games - Live Tracking

**Besser HTTP:** - Standard CRUD - Seltene Updates - Request-Response Pattern

### 5.1.0.3 WebSocket-Protokoll

Handshake

WebSocket beginnt als HTTP-Upgrade:

```
# Client Request
GET /ws HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13

# Server Response
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+x0o=
```

Message Types

```
// Text Messages
socket.add('Hello, World!');

// Binary Messages
socket.add(Uint8List.fromList([0x01, 0x02, 0x03]));

// Close Frame
socket.close(1000, 'Normal closure');
```

Close Codes

Code	Bedeutung
1000	Normal Closure
1001	Going Away
1002	Protocol Error
1003	Unsupported Data
1006	Abnormal Closure
1011	Internal Error

### 5.1.0.4 shelf\_web\_socket Package

Setup

```
# pubspec.yaml
dependencies:
  shelf: ^1.4.0
  shelf_web_socket: ^2.0.0
  web_socket_channel: ^3.0.0
```

Einfacher WebSocket Handler

```
import 'dart:async';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as io;
import 'package:shelf_web_socket/shelf_web_socket.dart';
import 'package:web_socket_channel/web_socket_channel.dart';

void main() async {
  final handler = webSocketHandler((WebSocketChannel webSocket) {
    // Verbindung hergestellt
    print('Client connected');

    // Willkommensnachricht senden
    webSocket.sink.add('Welcome!');

    // Auf Nachrichten hören
    webSocket.stream.listen(
      (message) {
        print('Received: $message');
        // Echo zurücksenden
        webSocket.sink.add('Echo: $message');
      },
      onDone: () {
        print('Client disconnected');
      },
      onError: (error) {
        print('Error: $error');
      },
    );
  });

  final server = await io.serve(handler, 'localhost', 8080);
  print('WebSocket server running on ws://localhost:${server.port}');
}
```

Integration mit Router

```
import 'package:shelf_router/shelf_router.dart';

Router createRouter() {
  final router = Router();

  // HTTP Endpoints
  router.get('/api/status', (Request request) {
    return Response.ok({'status': 'online'});
  });

  // WebSocket Endpoint
  router.get('/ws', webSocketHandler((WebSocketChannel webSocket) {
    handleWebSocket(webSocket);
  }));
}
```

```

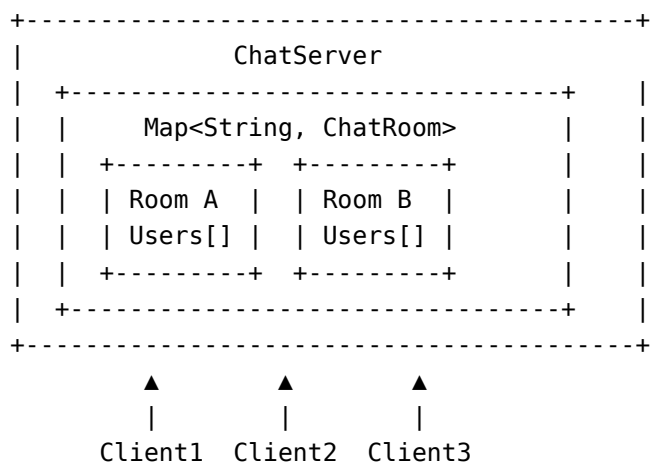
    return router;
}

void handleWebSocket(WebSocketChannel webSocket) {
  webSocket.stream.listen((message) {
    webSocket.sink.add('Received: $message');
  });
}

```

### 5.1.0.5 Chat-Server Beispiel

Architektur



ChatUser Klasse

```

import 'dart:convert';
import 'package:web_socket_channel/web_socket_channel.dart';

class ChatUser {
  final String id;
  final String name;
  final WebSocketChannel socket;
  String? currentRoom;

  ChatUser({
    required this.id,
    required this.name,
    required this.socket,
  });

  void send(Map<String, dynamic> message) {
    socket.sink.add(jsonEncode(message));
  }

  void sendText(String type, String text) {
    send({'type': type, 'message': text});
  }
}

```

```
void sendError(String error) {  
    send({'type': 'error', 'message': error});  
}  
}
```

ChatRoom Klasse

```
class ChatRoom {  
    final String name;  
    final Set<ChatUser> users = {};  
    final List<Map<String, dynamic>> messageHistory = [];  
    final int maxHistorySize;  
  
    ChatRoom(this.name, {this.maxHistorySize = 100});  
  
    void join(ChatUser user) {  
        users.add(user);  
        user.currentRoom = name;  
  
        // Benachrichtige andere User  
        broadcast({  
            'type': 'user_joined',  
            'user': user.name,  
            'room': name,  
            'userCount': users.length,  
        }, exclude: user);  
  
        // Sende History an neuen User  
        user.send({  
            'type': 'room_joined',  
            'room': name,  
            'history': messageHistory.take(50).toList(),  
            'users': users.map((u) => u.name).toList(),  
        });  
    }  
  
    void leave(ChatUser user) {  
        users.remove(user);  
        user.currentRoom = null;  
  
        broadcast({  
            'type': 'user_left',  
            'user': user.name,  
            'room': name,  
            'userCount': users.length,  
        });  
    }  
  
    void broadcast(Map<String, dynamic> message, {ChatUser? exclude}) {  
        final encoded = jsonEncode(message);  
        for (final user in users) {
```

```

        if (user != exclude) {
            user.socket.sink.add(encoded);
        }
    }
}

void sendMessage(ChatUser sender, String text) {
    final message = {
        'type': 'message',
        'room': name,
        'user': sender.name,
        'text': text,
        'timestamp': DateTime.now().toIso8601String(),
    };

    // In History speichern
    messageHistory.add(message);
    if (messageHistory.length > maxHistorySize) {
        messageHistory.removeAt(0);
    }

    // An alle senden (inkl. Sender für Bestätigung)
    broadcast(message);
}
}

```

#### ChatServer

```

import 'dart:convert';
import 'package:uuid/uuid.dart';
import 'package:web_socket_channel/web_socket_channel.dart';

class ChatServer {
    final Map<String, ChatRoom> _rooms = {};
    final Map<String, ChatUser> _users = {};
    final _uuid = Uuid();

    ChatServer() {
        // Default-Räume erstellen
        _rooms['general'] = ChatRoom('general');
        _rooms['random'] = ChatRoom('random');
    }

    void handleConnection(WebSocketChannel socket) {
        final userId = _uuid.v4();
        ChatUser? user;

        socket.stream.listen(
            (data) {
                try {
                    final message = jsonDecode(data as String) as Map<String, dynamic>;

```

```

        final type = message['type'] as String?;

        switch (type) {
            case 'auth':
                user = _handleAuth(userId, message, socket);
                break;
            case 'join':
                if (user != null) _handleJoin(user!, message);
                break;
            case 'leave':
                if (user != null) _handleLeave(user!);
                break;
            case 'message':
                if (user != null) _handleMessage(user!, message);
                break;
            case 'list_rooms':
                _handleListRooms(socket);
                break;
            default:
                socket.sink.add(jsonEncode({
                    'type': 'error',
                    'message': 'Unknown message type: $type',
                }));
        }
    } catch (e) {
        socket.sink.add(jsonEncode({
            'type': 'error',
            'message': 'Invalid message format',
        }));
    }
},
onDone: () {
    _handleDisconnect(user);
},
onError: (error) {
    print('WebSocket error: $error');
    _handleDisconnect(user);
},
);
}

```

```

ChatUser _handleAuth(String id, Map<String, dynamic> message,
↪ WebSocketChannel socket) {
    final name = message['name'] as String? ?? 'Anonymous';

    final user = ChatUser(
        id: id,
        name: name,
        socket: socket,
    );
}

```

↪

```
_users[id] = user;

user.send({
    'type': 'auth_success',
    'userId': id,
    'name': name,
});

return user;
}

void _handleJoin(ChatUser user, Map<String, dynamic> message) {
    final roomName = message['room'] as String?;

    if (roomName == null) {
        user.sendError('Room name required');
        return;
    }

    // Aktuellen Raum verlassen
    if (user.currentRoom != null) {
        _rooms[user.currentRoom]?.leave(user);
    }

    // Raum erstellen falls nicht vorhanden
    _rooms.putIfAbsent(roomName, () => ChatRoom(roomName));

    // Beitreten
    _rooms[roomName]!.join(user);
}

void _handleLeave(ChatUser user) {
    if (user.currentRoom != null) {
        _rooms[user.currentRoom]?.leave(user);
    }
}

void _handleMessage(ChatUser user, Map<String, dynamic> message) {
    final text = message['text'] as String?;

    if (text == null || text.isEmpty) {
        user.sendError('Message text required');
        return;
    }

    if (user.currentRoom == null) {
        user.sendError('Join a room first');
        return;
    }

    _rooms[user.currentRoom]!.sendMessage(user, text);
}
```

```

}

void _handleListRooms(WebSocketChannel socket) {
  socket.sink.add(jsonEncode({
    'type': 'room_list',
    'rooms': _rooms.entries.map((e) => {
      'name': e.key,
      'userCount': e.value.users.length,
    }).toList(),
  }));
}

void _handleDisconnect(ChatUser? user) {
  if (user == null) return;

  // Aus Raum entfernen
  if (user.currentRoom != null) {
    _rooms[user.currentRoom]?.leave(user);
  }

  // Aus User-Liste entfernen
  _users.remove(user.id);

  print('User ${user.name} disconnected');
}
}

```

Server starten

```

import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as io;
import 'package:shelf_router/shelf_router.dart';
import 'package:shelf_web_socket/shelf_web_socket.dart';

void main() async {
  final chatServer = ChatServer();

  final router = Router();

  // REST API für Infos
  router.get('/api/rooms', (Request request) async {
    return Response.ok(
      jsonEncode({
        'rooms': chatServer._rooms.keys.toList(),
      }),
      headers: {'content-type': 'application/json'},
    );
  });

  // WebSocket Endpoint
  router.get('/ws', websocketHandler(chatServer.handleConnection));
}

```

```

// CORS für Browser-Clients
final handler = const Pipeline()
  .addMiddleware(logRequests())
  .addMiddleware(_corsMiddleware())
  .addHandler(router.call);

final server = await io.serve(handler, 'localhost', 8080);
print('Chat server running on http://localhost:${server.port}');
print('WebSocket: ws://localhost:${server.port}/ws');
}

Middleware _corsMiddleware() {
  return (Handler innerHandler) {
    return (Request request) async {
      if (request.method == 'OPTIONS') {
        return Response.ok('', headers: _corsHeaders);
      }
      final response = await innerHandler(request);
      return response.change(headers: _corsHeaders);
    };
  };
}

const _corsHeaders = {
  'Access-Control-Allow-Origin': '*',
  'Access-Control-Allow-Methods': 'GET, POST, OPTIONS',
  'Access-Control-Allow-Headers': 'Content-Type',
};

```

#### 5.1.0.6 Message-Protokoll Design

JSON-basiertes Protokoll

```

// Basis Message-Struktur
abstract class WsMessage {
  String get type;
  Map<String, dynamic> toJson();
}

// Konkrete Messages
class AuthMessage extends WsMessage {
  final String name;
  final String? token;

  AuthMessage({required this.name, this.token});

  @override
  String get type => 'auth';

  @override

```

```

Map<String, dynamic> toJson() => {
    'type': type,
    'name': name,
    if (token != null) 'token': token,
};

factory AuthMessage.fromJson(Map<String, dynamic> json) {
    return AuthMessage(
        name: json['name'] as String,
        token: json['token'] as String?,
    );
}

class ChatMessage extends WsMessage {
    final String text;
    final String? room;

    ChatMessage({required this.text, this.room});

    @override
    String get type => 'message';

    @override
    Map<String, dynamic> toJson() => {
        'type': type,
        'text': text,
        if (room != null) 'room': room,
    };
}

// Message Factory
WsMessage parseMessage(String data) {
    final json = jsonDecode(data) as Map<String, dynamic>;
    final type = json['type'] as String;

    switch (type) {
        case 'auth':
            return AuthMessage.fromJson(json);
        case 'message':
            return ChatMessage.fromJson(json);
        default:
            throw FormatException('Unknown message type: $type');
    }
}

```

Typed Message Handler

```

typedef MessageHandler<T extends WsMessage> = void Function(ChatUser user, T  ↵
    ↵ message);

```

```

class MessageRouter {
  final Map<String, Function> _handlers = {};

  void on<T extends WsMessage>(String type, MessageHandler<T> handler) {
    _handlers[type] = handler;
  }

  void handle(ChatUser user, Map<String, dynamic> json) {
    final type = json['type'] as String?;
    final handler = _handlers[type];

    if (handler == null) {
      user.sendError('Unknown message type: $type');
      return;
    }

    try {
      final message = parseMessage(jsonEncode(json));
      handler(user, message);
    } catch (e) {
      user.sendError('Invalid message: $e');
    }
  }
}

// Verwendung
final router = MessageRouter()
  ..on<AuthMessage>('auth', (user, msg) {
    // Handle auth
  })
  ..on<ChatMessage>('message', (user, msg) {
    // Handle message
  });

```

### 5.1.0.7 Heartbeat & Connection Management

Ping/Pong für Keep-Alive

```

import 'dart:async';

class ConnectionManager {
  final Duration pingInterval;
  final Duration timeout;
  final Map<String, Timer> _pingTimers = {};
  final Map<String, DateTime> _lastPong = {};

  ConnectionManager({
    this.pingInterval = const Duration(seconds: 30),
    this.timeout = const Duration(seconds: 60),
  });

```

```

void startHeartbeat(ChatUser user) {
    _lastPong[user.id] = DateTime.now();

    _pingTimers[user.id] = Timer.periodic(pingInterval, (timer) {
        // Prüfen ob Timeout
        final lastPong = _lastPong[user.id];
        if (lastPong != null &&
            DateTime.now().difference(lastPong) > timeout) {
            print('User ${user.name} timed out');
            user.socket.sink.close(1000, 'Ping timeout');
            stopHeartbeat(user.id);
            return;
        }

        // Ping senden
        user.send({'type': 'ping', 'timestamp':
↵ DateTime.now().millisecondsSinceEpoch});
    });
}

void handlePong(String oderId) {
    _lastPong[userId] = DateTime.now();
}

void stopHeartbeat(String userId) {
    _pingTimers[userId]?.cancel();
    _pingTimers.remove(userId);
    _lastPong.remove(userId);
}
}

```

Reconnection auf Client-Seite

```

// Flutter/Dart Client
class WebSocketClient {
    WebSocketChannel? _channel;
    Timer? _reconnectTimer;
    final String url;
    final Duration reconnectDelay;
    bool _isConnecting = false;

    final StreamController<Map<String, dynamic>> _messageController =
        StreamController.broadcast();

    Stream<Map<String, dynamic>> get messages => _messageController.stream;

    WebSocketClient(this.url, {this.reconnectDelay = const Duration(seconds: 5)});

    Future<void> connect() async {
        if (_isConnecting) return;
        _isConnecting = true;
    }
}

```

```

try {
    _channel = WebSocketChannel.connect(Uri.parse(url));

    _channel!.stream.listen(
        (data) {
            final message = jsonDecode(data as String);

            if (message['type'] == 'ping') {
                // Pong antworten
                send({'type': 'pong', 'timestamp': message['timestamp']});
            } else {
                _messageController.add(message);
            }
        },
        onDone: () {
            print('Connection closed, reconnecting...');
            _scheduleReconnect();
        },
        onError: (error) {
            print('Connection error: $error');
            _scheduleReconnect();
        },
    );

    _isConnecting = false;
} catch (e) {
    _isConnecting = false;
    _scheduleReconnect();
}

void _scheduleReconnect() {
    _reconnectTimer?.cancel();
    _reconnectTimer = Timer(reconnectDelay, connect);
}

void send(Map<String, dynamic> message) {
    _channel?.sink.add(jsonEncode(message));
}

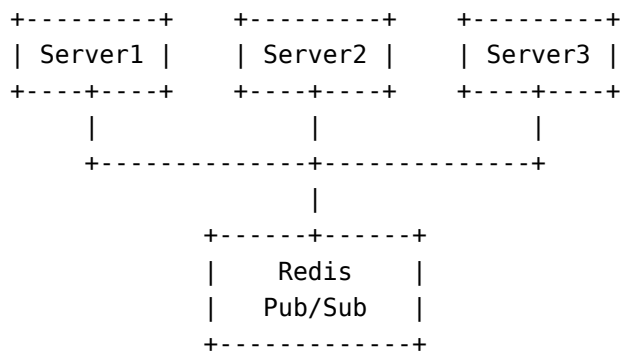
void close() {
    _reconnectTimer?.cancel();
    _channel?.sink.close();
    _messageController.close();
}
}

```

#### 5.1.0.8 Skalierung & Best Practices

Mehrere Server-Instanzen

Für Skalierung mit mehreren Server-Instanzen braucht man einen Message Broker:



Redis Pub/Sub Integration

```

import 'package:redis/redis.dart';

class ScalableChat {
  final RedisConnection _redisPub;
  final RedisConnection _redisSub;
  final ChatServer _localServer;

  ScalableChat(this._redisPub, this._redisSub, this._localServer);

  Future<void> start() async {
    // Subscribe to chat channel
    final pubsub = PubSub(await _redisSub.connect());

    pubsub.subscribe(['chat:messages']).listen((message) {
      final data = jsonDecode(message.message as String);
      _localServer.broadcastToRoom(data['room'], data);
    });
  }

  void publishMessage(String room, Map<String, dynamic> message) {
    // Publizieren für alle Server
    _redisPub.execute(['PUBLISH', 'chat:messages', jsonEncode({
      'room': room,
      ...message,
    })]);

    // Lokal auch senden (falls User auf diesem Server)
    _localServer.broadcastToRoom(room, message);
  }
}
  
```

Best Practices

```

// 1. Nachrichten-Validierung
void validateMessage(Map<String, dynamic> message) {
  if (message['type'] == null) {
    throw FormatException('Message type required');
  }
}
  
```

```
if (message['type'] == 'message') {
    final text = message['text'] as String?;
    if (text == null || text.isEmpty || text.length > 10000) {
        throw FormatException('Invalid message text');
    }
}
}

// 2. Rate Limiting
class WebSocketRateLimiter {
    final int maxMessagesPerSecond;
    final Map<String, List<DateTime>> _messageLog = {};

    WebSocketRateLimiter({this.maxMessagesPerSecond = 10});

    bool allowMessage(String oderId) {
        final now = DateTime.now();
        final log = _messageLog.putIfAbsent(oderId, () => []);

        // Alte Einträge entfernen
        log.removeWhere((time) => now.difference(time).inSeconds > 1);

        if (log.length >= maxMessagesPerSecond) {
            return false;
        }

        log.add(now);
        return true;
    }
}

// 3. Graceful Shutdown
Future<void> gracefulShutdown(HttpServer server, ChatServer chat) async {
    print('Shutting down...');

    // Allen Clients Bescheid geben
    for (final user in chat.users) {
        user.send({'type': 'server_shutdown'});
        user.socket.sink.close(1001, 'Server shutting down');
    }

    // Kurz warten
    await Future.delayed(Duration(seconds: 1));

    // Server stoppen
    await server.close();
}
```

### 5.1.0.9 Zusammenfassung

- **WebSockets** ermöglichen bidirektionale Real-time-Kommunikation
- **shelf\_web\_socket** integriert WebSockets nahtlos in Shelf
- **Message-Protokoll** mit JSON für strukturierte Kommunikation
- **Rooms** für Gruppierung von Clients
- **Heartbeat** für Connection-Überwachung
- **Redis Pub/Sub** für Multi-Server-Skalierung

Nächste Schritte

In der nächsten Einheit behandeln wir Background Jobs & Scheduling für asynchrone Aufgaben.

### 5.1.1 Übung

#### 5.1.1.1 Ziel

Entwickle einen Real-time Notification Service mit WebSockets.

#### 5.1.1.2 Vorbereitung

Dependencies

```
# pubspec.yaml
dependencies:
  shelf: ^1.4.0
  shelf_router: ^1.1.0
  shelf_web_socket: ^2.0.0
  web_socket_channel: ^3.0.0
  uuid: ^4.0.0
```

#### 5.1.1.3 Aufgabe 1: Einfacher Echo-Server (15 min)

Erstelle einen WebSocket-Server, der empfangene Nachrichten zurücksendet.

```
// bin/echo_server.dart

import 'package:shelf/shelf_io.dart' as io;
import 'package:shelf_web_socket/shelf_web_socket.dart';
import 'package:web_socket_channel/web_socket_channel.dart';

void main() async {
  // TODO: WebSocket Handler erstellen
  // - Bei Verbindung "Connected!" senden
  // - Empfangene Nachrichten mit "Echo: " Prefix zurücksenden
  // - Bei Disconnect loggen

  // TODO: Server auf Port 8080 starten
}
```

Test mit websocat:

```
# Installation: cargo install websocat
websocat ws://localhost:8080
> Hello
```

```
< Connected!  
< Echo: Hello
```

#### 5.1.1.4 Aufgabe 2: Connection Manager (20 min)

Implementiere einen Connection Manager für mehrere Clients.

```
// lib/connection_manager.dart  
  
import 'package:web_socket_channel/web_socket_channel.dart';  
  
class Client {  
  final String id;  
  final WebSocketChannel socket;  
  final DateTime connectedAt;  
  String? name;  
  
  // TODO: Implementieren  
}  
  
class ConnectionManager {  
  final Map<String, Client> _clients = {};  
  
  /// Neue Verbindung registrieren  
  Client addConnection(WebSocketChannel socket) {  
    // TODO: Client erstellen und speichern  
  }  
  
  /// Verbindung entfernen  
  void removeConnection(String clientId) {  
    // TODO  
  }  
  
  /// Nachricht an alle senden  
  void broadcast(Map<String, dynamic> message, {String? excludeId}) {  
    // TODO: An alle außer excludeId senden  
  }  
  
  /// Nachricht an bestimmten Client senden  
  void sendTo(String clientId, Map<String, dynamic> message) {  
    // TODO  
  }  
  
  /// Anzahl verbundener Clients  
  int get clientCount => _clients.length;  
  
  /// Alle Client-IDs  
  List<String> get clientIds => _clients.keys.toList();  
}
```

**5.1.1.5 Aufgabe 3: Notification Service (25 min)**

Baue einen Notification Service mit Channels.

```
// lib/notification_service.dart

class NotificationChannel {
  final String name;
  final Set<String> subscribers = {};

  NotificationChannel(this.name);

  void subscribe(String clientId) {
    // TODO
  }

  void unsubscribe(String clientId) {
    // TODO
  }

  bool hasSubscriber(String clientId) {
    // TODO
  }
}

class NotificationService {
  final ConnectionManager _connections;
  final Map<String, NotificationChannel> _channels = {};

  NotificationService(this._connections);

  /// Client für Channel anmelden
  void subscribe(String clientId, String channelName) {
    // TODO: Channel erstellen falls nicht vorhanden
    // TODO: Client zum Channel hinzufügen
    // TODO: Bestätigung an Client senden
  }

  /// Client von Channel abmelden
  void unsubscribe(String clientId, String channelName) {
    // TODO
  }

  /// Notification an Channel senden
  void notify(String channelName, Map<String, dynamic> data) {
    // TODO: Nachricht an alle Subscriber des Channels senden
  }

  /// Client komplett entfernen (bei Disconnect)
  void removeClient(String clientId) {
    // TODO: Aus allen Channels entfernen
  }
}
```

```
/// Channels eines Clients
List<String> getSubscriptions(String clientId) {
  // TODO
}
}
```

#### 5.1.1.6 Aufgabe 4: Message Protocol (20 min)

Definiere ein strukturiertes Message-Protokoll.

```
// lib/messages.dart

/// Basis für alle Messages
abstract class WsMessage {
  String get type;
  Map<String, dynamic> toJson();
}

/// Client -> Server: Subscribe
class SubscribeMessage extends WsMessage {
  final String channel;

  SubscribeMessage(this.channel);

  @override
  String get type => 'subscribe';

  @override
  Map<String, dynamic> toJson() {
    // TODO
  }

  factory SubscribeMessage.fromJson(Map<String, dynamic> json) {
    // TODO
  }
}

/// Client -> Server: Unsubscribe
class UnsubscribeMessage extends WsMessage {
  // TODO: Analog zu SubscribeMessage
}

/// Server -> Client: Notification
class NotificationMessage extends WsMessage {
  final String channel;
  final String title;
  final String body;
  final DateTime timestamp;
  final Map<String, dynamic>? data;
}
```

```

    // TODO: Implementieren
}

/// Server -> Client: Error
class ErrorMessage extends WsMessage {
    final String error;
    final String? code;

    // TODO: Implementieren
}

/// Server -> Client: Subscription confirmed
class SubscribedMessage extends WsMessage {
    final String channel;

    // TODO: Implementieren
}

/// Message Parser
WsMessage? parseMessage(String data) {
    // TODO: JSON parsen und richtige Message-Klasse zurückgeben
}

```

#### 5.1.1.7 Aufgabe 5: WebSocket Handler (25 min)

Verbinde alles in einem Handler.

```

// lib/ws_handler.dart

import 'dart:convert';
import 'package:web_socket_channel/web_socket_channel.dart';

class NotificationHandler {
    final ConnectionManager connections;
    final NotificationService notifications;

    NotificationHandler(this.connections, this.notifications);

    void handleConnection(WebSocketChannel socket) {
        // TODO: Client registrieren
        final client = connections.addConnection(socket);

        // TODO: Willkommensnachricht senden

        // TODO: Auf Nachrichten hören
        socket.stream.listen(
            (data) {
                _handleMessage(client, data as String);
            },
            onDone: () {
                // TODO: Aufräumen bei Disconnect
            }
        );
    }
}

```

```

    },
    onError: (error) {
      // TODO: Error handling
    },
  );
}

void _handleMessage(Client client, String data) {
  try {
    final message = parseMessage(data);

    // TODO: Switch über message.type
    // - 'subscribe': Channel abonnieren
    // - 'unsubscribe': Channel verlassen
    // - 'set_name': Client-Namen setzen

  } catch (e) {
    // TODO: Error zurücksenden
  }
}
}

```

#### 5.1.1.8 Aufgabe 6: REST API für Notifications (20 min)

Erstelle REST-Endpoints zum Senden von Notifications.

```

// lib/api_handler.dart

import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';

class NotificationApiHandler {
  final NotificationService notifications;

  NotificationApiHandler(this.notifications);

  Router get router {
    final router = Router();

    // POST /api/notify/:channel
    // Body: {"title": "...", "body": "...", "data": {...}}
    router.post('/api/notify/<channel>', _sendNotification);

    // GET /api/channels
    // Returns: {"channels": ["channel1", "channel2"], "subscribers": {...}}
    router.get('/api/channels', _listChannels);

    // GET /api/stats
    // Returns: {"connections": 5, "channels": 3}
    router.get('/api/stats', _getStats);
  }
}

```

```
    return router;
  }

Future<Response> _sendNotification(Request request, String channel) async {
  // TODO: Body parsen
  // TODO: Notification senden
  // TODO: Response mit Anzahl erreichter Clients
}

Future<Response> _listChannels(Request request) async {
  // TODO
}

Future<Response> _getStats(Request request) async {
  // TODO
}
}
```

#### 5.1.1.9 Aufgabe 7: Server Assembly (15 min)

Führe alles zusammen.

```
// bin/server.dart

import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as io;
import 'package:shelf_router/shelf_router.dart';
import 'package:shelf_web_socket/shelf_web_socket.dart';

void main() async {
  // TODO: Services erstellen
  final connections = ConnectionManager();
  final notifications = NotificationService(connections);
  final wsHandler = NotificationHandler(connections, notifications);
  final apiHandler = NotificationApiHandler(notifications);

  // TODO: Router mit WebSocket und REST
  final router = Router();

  // WebSocket auf /ws
  router.get('/ws', websocketHandler(wsHandler.handleConnection));

  // REST API
  router.mount('/api', apiHandler.router.call);

  // TODO: Pipeline mit CORS und Logging

  // TODO: Server starten

  print('Notification Server running on http://localhost:8080');
  print('WebSocket: ws://localhost:8080/ws');
```

```
    print('Send notification: POST http://localhost:8080/api/notify/:channel');
  }
```

#### 5.1.1.10 Aufgabe 8: Client Simulator (15 min)

Schreibe einen Test-Client.

```
// bin/client.dart

import 'dart:async';
import 'dart:convert';
import 'dart:io';
import 'package:web_socket_channel/web_socket_channel.dart';

void main() async {
  final channel = WebSocketChannel.connect(Uri.parse('ws://localhost:8080/ws'));

  // TODO: Auf Server-Nachrichten hören und ausgeben
  channel.stream.listen((data) {
    final message = jsonDecode(data as String);
    print('Received: $message');
  });

  // TODO: Stdin lesen und Kommandos senden
  // Kommandos:
  // - sub <channel>: Channel abonnieren
  // - unsub <channel>: Channel verlassen
  // - name <name>: Namen setzen
  // - quit: Beenden

  print('Commands: sub <channel>, unsub <channel>, name <name>, quit');

  await for (final line in
    ↪ stdin.transform(utf8.decoder).transform(LineSplitter())) {
    ↪ final parts = line.split(' ');
    ↪ final cmd = parts[0];

    switch (cmd) {
      case 'sub':
        // TODO: Subscribe Message senden
        break;
      case 'unsub':
        // TODO: Unsubscribe Message senden
        break;
      case 'name':
        // TODO: Name setzen
        break;
      case 'quit':
        channel.sink.close();
        exit(0);
    }
  }
}
```

```
}  
}
```

#### 5.1.1.11 Bonus: Heartbeat (Optional, 15 min)

Implementiere Heartbeat für Connection-Health.

```
// lib/heartbeat.dart  
  
class HeartbeatManager {  
  final Duration pingInterval;  
  final Duration timeout;  
  final ConnectionManager connections;  
  final Map<String, DateTime> _lastPong = {};  
  Timer? _timer;  
  
  HeartbeatManager({  
    required this.connections,  
    this.pingInterval = const Duration(seconds: 30),  
    this.timeout = const Duration(seconds: 60),  
  });  
  
  void start() {  
    // TODO: Timer starten  
    // TODO: Regelmäßig Pings senden  
    // TODO: Timeouts prüfen  
  }  
  
  void handlePong(String clientId) {  
    // TODO: Last pong Zeit aktualisieren  
  }  
  
  void stop() {  
    // TODO: Timer stoppen  
  }  
}
```

#### 5.1.1.12 Testen

1. Server starten

```
dart run bin/server.dart
```

2. Client starten (mehrere Terminals)

```
dart run bin/client.dart  
# > sub news  
# > name Alice
```

3. Notification senden

```
curl -X POST http://localhost:8080/api/notify/news \
```

```
-H "Content-Type: application/json" \  
-d '{"title": "Breaking News", "body": "Something happened!"}'
```

#### 4. Stats prüfen

```
curl http://localhost:8080/api/stats
```

##### 5.1.1.13 Abgabe-Checkliste

- ☐ Echo-Server funktioniert
- ☐ ConnectionManager verwaltet Clients
- ☐ NotificationService mit Channels
- ☐ Strukturiertes Message-Protokoll
- ☐ WebSocket Handler verarbeitet Messages
- ☐ REST API zum Senden von Notifications
- ☐ Server mit WebSocket und REST kombiniert
- ☐ Client Simulator funktioniert
- ☐ Bonus: Heartbeat implementiert

### 5.1.2 Lösung

#### 5.1.2.1 Aufgabe 1: Echo-Server

```
// bin/echo_server.dart  
  
import 'package:shelf/shelf_io.dart' as io;  
import 'package:shelf_web_socket/shelf_web_socket.dart';  
import 'package:web_socket_channel/web_socket_channel.dart';  
  
void main() async {  
  final handler = webSocketHandler((WebSocketChannel socket) {  
    print('Client connected');  
  
    // Willkommensnachricht  
    socket.sink.add('Connected!');  
  
    // Auf Nachrichten hören  
    socket.stream.listen(  
      (message) {  
        print('Received: $message');  
        socket.sink.add('Echo: $message');  
      },  
      onDone: () {  
        print('Client disconnected');  
      },  
      onError: (error) {  
        print('Error: $error');  
      },  
    );  
  });  
}
```

```
final server = await io.serve(handler, 'localhost', 8080);
print('Echo server running on ws://localhost:${server.port}');
}
```

### 5.1.2.2 Aufgabe 2: Connection Manager

```
// lib/connection_manager.dart

import 'dart:convert';
import 'package:uuid/uuid.dart';
import 'package:web_socket_channel/web_socket_channel.dart';

class Client {
  final String id;
  final WebSocketChannel socket;
  final DateTime connectedAt;
  String? name;

  Client({
    required this.id,
    required this.socket,
    required this.connectedAt,
    this.name,
  });

  void send(Map<String, dynamic> message) {
    socket.sink.add(jsonEncode(message));
  }

  void sendRaw(String data) {
    socket.sink.add(data);
  }

  @override
  String toString() => 'Client(id: $id, name: $name)';
}

class ConnectionManager {
  final Map<String, Client> _clients = {};
  final _uuid = Uuid();

  /// Neue Verbindung registrieren
  Client addConnection(WebSocketChannel socket) {
    final id = _uuid.v4();
    final client = Client(
      id: id,
      socket: socket,
      connectedAt: DateTime.now(),
    );
  }
}
```

```

    _clients[id] = client;
    print('Client connected: $id (total: ${_clients.length})');

    return client;
}

/// Verbindung entfernen
void removeConnection(String clientId) {
    final client = _clients.remove(clientId);
    if (client != null) {
        print('Client disconnected: $clientId (total: ${_clients.length})');
    }
}

/// Client abrufen
Client? getClient(String clientId) => _clients[clientId];

/// Nachricht an alle senden
void broadcast(Map<String, dynamic> message, {String? excludeId}) {
    final encoded = jsonEncode(message);
    for (final entry in _clients.entries) {
        if (entry.key != excludeId) {
            entry.value.sendRaw(encoded);
        }
    }
}

/// Nachricht an bestimmten Client senden
void sendTo(String clientId, Map<String, dynamic> message) {
    _clients[clientId]?.send(message);
}

/// Anzahl verbundener Clients
int get clientCount => _clients.length;

/// Alle Client-IDs
List<String> get clientIds => _clients.keys.toList();

/// Alle Clients
Iterable<Client> get clients => _clients.values;
}

```

### 5.1.2.3 Aufgabe 3: Notification Service

```

// lib/notification_service.dart

import 'connection_manager.dart';

class NotificationChannel {
    final String name;

```

```
final Set<String> subscribers = {};
final DateTime createdAt;

NotificationChannel(this.name) : createdAt = DateTime.now();

void subscribe(String clientId) {
    subscribers.add(clientId);
}

void unsubscribe(String clientId) {
    subscribers.remove(clientId);
}

bool hasSubscriber(String clientId) {
    return subscribers.contains(clientId);
}

int get subscriberCount => subscribers.length;

bool get isEmpty => subscribers.isEmpty;
}

class NotificationService {
    final ConnectionManager _connections;
    final Map<String, NotificationChannel> _channels = {};

    NotificationService(this._connections);

    /// Client für Channel anmelden
    void subscribe(String clientId, String channelName) {
        // Channel erstellen falls nicht vorhanden
        final channel = _channels.putIfAbsent(
            channelName,
            () => NotificationChannel(channelName),
        );

        // Client zum Channel hinzufügen
        channel.subscribe(clientId);

        // Bestätigung an Client senden
        _connections.sendTo(clientId, {
            'type': 'subscribed',
            'channel': channelName,
            'subscriberCount': channel.subscriberCount,
        });

        print('Client $clientId subscribed to $channelName');
    }

    /// Client von Channel abmelden
    void unsubscribe(String clientId, String channelName) {
```

```
final channel = _channels[channelName];
if (channel == null) return;

channel.unsubscribe(clientId);

// Bestätigung senden
_connections.sendTo(clientId, {
    'type': 'unsubscribed',
    'channel': channelName,
});

// Leere Channels aufräumen
if (channel.isEmpty) {
    _channels.remove(channelName);
}

print('Client $clientId unsubscribed from $channelName');
}

/// Notification an Channel senden
int notify(String channelName, Map<String, dynamic> data) {
    final channel = _channels[channelName];
    if (channel == null) return 0;

    final message = {
        'type': 'notification',
        'channel': channelName,
        'timestamp': DateTime.now().toIso8601String(),
        ...data,
    };

    int delivered = 0;
    for (final clientId in channel.subscribers) {
        final client = _connections.getClient(clientId);
        if (client != null) {
            client.send(message);
            delivered++;
        }
    }

    print('Notification sent to $delivered clients on $channelName');
    return delivered;
}

/// Client komplett entfernen (bei Disconnect)
void removeClient(String clientId) {
    final emptyChannels = <String>[];

    for (final entry in _channels.entries) {
        entry.value.unsubscribe(clientId);
        if (entry.value.isEmpty) {
```

```

        emptyChannels.add(entry.key);
    }
}

// Leere Channels entfernen
for (final name in emptyChannels) {
    _channels.remove(name);
}

/// Channels eines Clients
List<String> getSubscriptions(String clientId) {
    return _channels.entries
        .where((e) => e.value.hasSubscriber(clientId))
        .map((e) => e.key)
        .toList();
}

/// Alle Channels
Map<String, int> getChannelStats() {
    return Map.fromEntries(
        _channels.entries.map((e) => MapEntry(e.key, e.value.subscriberCount)),
    );
}

/// Anzahl Channels
int get channelCount => _channels.length;
}

```

#### 5.1.2.4 Aufgabe 4: Message Protocol

```

// lib/messages.dart

import 'dart:convert';

/// Basis für alle Messages
abstract class WsMessage {
    String get type;
    Map<String, dynamic> toJson();

    String encode() => jsonEncode(toJson());
}

/// Client -> Server: Subscribe
class SubscribeMessage extends WsMessage {
    final String channel;

    SubscribeMessage(this.channel);

    @override

```

```
String get type => 'subscribe';

@override
Map<String, dynamic> toJson() => {
    'type': type,
    'channel': channel,
};

factory SubscribeMessage.fromJson(Map<String, dynamic> json) {
    return SubscribeMessage(json['channel'] as String);
}

/// Client -> Server: Unsubscribe
class UnsubscribeMessage extends WsMessage {
    final String channel;

    UnsubscribeMessage(this.channel);

    @override
    String get type => 'unsubscribe';

    @override
    Map<String, dynamic> toJson() => {
        'type': type,
        'channel': channel,
    };

    factory UnsubscribeMessage.fromJson(Map<String, dynamic> json) {
        return UnsubscribeMessage(json['channel'] as String);
    }
}

/// Client -> Server: Set Name
class SetNameMessage extends WsMessage {
    final String name;

    SetNameMessage(this.name);

    @override
    String get type => 'set_name';

    @override
    Map<String, dynamic> toJson() => {
        'type': type,
        'name': name,
    };

    factory SetNameMessage.fromJson(Map<String, dynamic> json) {
        return SetNameMessage(json['name'] as String);
    }
}
```

```
}

/// Server -> Client: Notification
class NotificationMessage extends WsMessage {
  final String channel;
  final String title;
  final String body;
  final DateTime timestamp;
  final Map<String, dynamic>? data;

  NotificationMessage({
    required this.channel,
    required this.title,
    required this.body,
    DateTime? timestamp,
    this.data,
  }) : timestamp = timestamp ?? DateTime.now();

  @override
  String get type => 'notification';

  @override
  Map<String, dynamic> toJson() => {
    'type': type,
    'channel': channel,
    'title': title,
    'body': body,
    'timestamp': timestamp.toIso8601String(),
    if (data != null) 'data': data,
  };

  factory NotificationMessage.fromJson(Map<String, dynamic> json) {
    return NotificationMessage(
      channel: json['channel'] as String,
      title: json['title'] as String,
      body: json['body'] as String,
      timestamp: DateTime.parse(json['timestamp'] as String),
      data: json['data'] as Map<String, dynamic>?,
    );
  }
}

/// Server -> Client: Error
class ErrorMessage extends WsMessage {
  final String error;
  final String? code;

  ErrorMessage(this.error, {this.code});

  @override
  String get type => 'error';
```

```
@override
Map<String, dynamic> toJson() => {
    'type': type,
    'error': error,
    if (code != null) 'code': code,
};

factory ErrorMessage.fromJson(Map<String, dynamic> json) {
    return ErrorMessage(
        json['error'] as String,
        code: json['code'] as String?,
    );
}
}

/// Server -> Client: Subscription confirmed
class SubscribedMessage extends WsMessage {
    final String channel;
    final int subscriberCount;

    SubscribedMessage(this.channel, {this.subscriberCount = 0});

    @override
    String get type => 'subscribed';

    @override
    Map<String, dynamic> toJson() => {
        'type': type,
        'channel': channel,
        'subscriberCount': subscriberCount,
    };

    factory SubscribedMessage.fromJson(Map<String, dynamic> json) {
        return SubscribedMessage(
            json['channel'] as String,
            subscriberCount: json['subscriberCount'] as int? ?? 0,
        );
    }
}

/// Server -> Client: Welcome
class WelcomeMessage extends WsMessage {
    final String clientId;

    WelcomeMessage(this.clientId);

    @override
    String get type => 'welcome';

    @override
```

```
Map<String, dynamic> toJson() => {
  'type': type,
  'clientId': clientId,
};
}

/// Message Parser
WsMessage? parseMessage(String data) {
  try {
    final json = jsonDecode(data) as Map<String, dynamic>;
    final type = json['type'] as String?;

    switch (type) {
      case 'subscribe':
        return SubscribeMessage.fromJson(json);
      case 'unsubscribe':
        return UnsubscribeMessage.fromJson(json);
      case 'set_name':
        return SetNameMessage.fromJson(json);
      case 'notification':
        return NotificationMessage.fromJson(json);
      case 'error':
        return ErrorMessage.fromJson(json);
      case 'subscribed':
        return SubscribedMessage.fromJson(json);
      default:
        return null;
    }
  } catch (e) {
    return null;
  }
}
```

#### 5.1.2.5 Aufgabe 5: WebSocket Handler

```
// lib/ws_handler.dart

import 'dart:convert';
import 'package:web_socket_channel/web_socket_channel.dart';

import 'connection_manager.dart';
import 'notification_service.dart';
import 'messages.dart';

class NotificationHandler {
  final ConnectionManager connections;
  final NotificationService notifications;

  NotificationHandler(this.connections, this.notifications);
```

```
void handleConnection(WebSocketChannel socket) {
    // Client registrieren
    final client = connections.addConnection(socket);

    // Willkommensnachricht senden
    client.send({
        'type': 'welcome',
        'clientId': client.id,
        'message': 'Connected to notification server',
    });

    // Auf Nachrichten hören
    socket.stream.listen(
        (data) {
            _handleMessage(client, data as String);
        },
        onDone: () {
            _handleDisconnect(client);
        },
        onError: (error) {
            print('WebSocket error for ${client.id}: $error');
            _handleDisconnect(client);
        },
    );
}

void _handleMessage(Client client, String data) {
    try {
        final json = jsonDecode(data) as Map<String, dynamic>;
        final type = json['type'] as String?;

        switch (type) {
            case 'subscribe':
                final channel = json['channel'] as String?;
                if (channel != null && channel.isNotEmpty) {
                    notifications.subscribe(client.id, channel);
                } else {
                    _sendError(client, 'Channel name required');
                }
                break;

            case 'unsubscribe':
                final channel = json['channel'] as String?;
                if (channel != null) {
                    notifications.unsubscribe(client.id, channel);
                }
                break;

            case 'set_name':
                final name = json['name'] as String?;
                if (name != null && name.isNotEmpty) {
```

```
        client.name = name;
        client.send({
          'type': 'name_set',
          'name': name,
        });
      } else {
        _sendError(client, 'Name required');
      }
      break;

    case 'list_subscriptions':
      final subs = notifications.getSubscriptions(client.id);
      client.send({
        'type': 'subscriptions',
        'channels': subs,
      });
      break;

    case 'ping':
      client.send({
        'type': 'pong',
        'timestamp': DateTime.now().millisecondsSinceEpoch,
      });
      break;

    default:
      _sendError(client, 'Unknown message type: $type');
  }
} catch (e) {
  _sendError(client, 'Invalid message format: $e');
}
}

void _handleDisconnect(Client client) {
  notifications.removeClient(client.id);
  connections.removeConnection(client.id);
}

void _sendError(Client client, String message) {
  client.send({
    'type': 'error',
    'error': message,
  });
}
}
```

#### 5.1.2.6 Aufgabe 6: REST API

```
// lib/api_handler.dart
```

```
import 'dart:convert';
import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';

import 'connection_manager.dart';
import 'notification_service.dart';

class NotificationApiHandler {
  final ConnectionManager connections;
  final NotificationService notifications;

  NotificationApiHandler(this.connections, this.notifications);

  Router get router {
    final router = Router();

    router.post('/notify/<channel>', _sendNotification);
    router.get('/channels', _listChannels);
    router.get('/stats', _getStats);
    router.get('/clients', _listClients);

    return router;
  }

  Future<Response> _sendNotification(Request request, String channel) async {
    try {
      final body = await request.readAsString();
      final json = jsonDecode(body) as Map<String, dynamic>;

      final title = json['title'] as String? ?? 'Notification';
      final messageBody = json['body'] as String? ?? '';
      final data = json['data'] as Map<String, dynamic>?;

      final delivered = notifications.notify(channel, {
        'title': title,
        'body': messageBody,
        if (data != null) 'data': data,
      });

      return Response.ok(
        jsonEncode({
          'success': true,
          'channel': channel,
          'delivered': delivered,
        }),
        headers: {'content-type': 'application/json'},
      );
    } catch (e) {
      return Response.badRequest(
        body: jsonEncode({
          'error': 'Invalid request body',
        })
      );
    }
  }
}
```

```
        'details': e.toString(),
      }),
      headers: {'content-type': 'application/json'},
    );
  }
}

Future<Response> _listChannels(Request request) async {
  final stats = notifications.getChannelStats();

  return Response.ok(
    jsonEncode({
      'channels': stats.entries
        .map((e) => {
          'name': e.key,
          'subscribers': e.value,
        })
        .toList(),
      'total': stats.length,
    }),
    headers: {'content-type': 'application/json'},
  );
}

Future<Response> _getStats(Request request) async {
  return Response.ok(
    jsonEncode({
      'connections': connections.clientCount,
      'channels': notifications.channelCount,
      'timestamp': DateTime.now().toIso8601String(),
    }),
    headers: {'content-type': 'application/json'},
  );
}

Future<Response> _listClients(Request request) async {
  final clients = connections.clients.map((c) => {
    'id': c.id,
    'name': c.name,
    'connectedAt': c.connectedAt.toIso8601String(),
    'subscriptions': notifications.getSubscriptions(c.id),
  });

  return Response.ok(
    jsonEncode({
      'clients': clients.toList(),
      'total': connections.clientCount,
    }),
    headers: {'content-type': 'application/json'},
  );
}
```

```
}
```

### 5.1.2.7 Aufgabe 7: Server Assembly

```
// bin/server.dart

import 'dart:io';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as io;
import 'package:shelf_router/shelf_router.dart';
import 'package:shelf_web_socket/shelf_web_socket.dart';

import '../lib/connection_manager.dart';
import '../lib/notification_service.dart';
import '../lib/ws_handler.dart';
import '../lib/api_handler.dart';

void main() async {
  // Services erstellen
  final connections = ConnectionManager();
  final notifications = NotificationService(connections);
  final wsHandler = NotificationHandler(connections, notifications);
  final apiHandler = NotificationApiHandler(connections, notifications);

  // Router mit WebSocket und REST
  final router = Router();

  // WebSocket auf /ws
  router.get('/ws', webSocketHandler(wsHandler.handleConnection));

  // REST API
  router.mount('/api/', apiHandler.router.call);

  // Health Check
  router.get('/health', (Request request) {
    return Response.ok('OK');
  });

  // Pipeline mit CORS und Logging
  final handler = const Pipeline()
    .addMiddleware(logRequests())
    .addMiddleware(_corsMiddleware())
    .addHandler(router.call);

  // Server starten
  final port = int.parse(Platform.environment['PORT'] ?? '8080');
  final server = await io.serve(handler, InternetAddress.anyIPv4, port);

  print('=====');
  print('Notification Server running on port ${server.port}');
```

```

print('=====');
print('WebSocket: ws://localhost:${server.port}/ws');
print('REST API: http://localhost:${server.port}/api');
print('');
print('Endpoints:');
print('  POST /api/notify/:channel - Send notification');
print('  GET  /api/channels        - List channels');
print('  GET  /api/stats            - Server statistics');
print('  GET  /api/clients          - List clients');
print('  GET  /health                - Health check');
print('=====');

// Graceful Shutdown
ProcessSignal.sigint.watch().listen((_) async {
  print('\nShutting down...');

  // Alle Clients benachrichtigen
  connections.broadcast({
    'type': 'server_shutdown',
    'message': 'Server is shutting down',
  });

  await Future.delayed(Duration(milliseconds: 500));
  await server.close();
  exit(0);
});
}

Middleware _corsMiddleware() {
  const corsHeaders = {
    'Access-Control-Allow-Origin': '*',
    'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE, OPTIONS',
    'Access-Control-Allow-Headers': 'Content-Type, Authorization',
  };

  return (Handler innerHandler) {
    return (Request request) async {
      if (request.method == 'OPTIONS') {
        return Response.ok('', headers: corsHeaders);
      }

      final response = await innerHandler(request);
      return response.change(headers: corsHeaders);
    };
  };
}

```

### 5.1.2.8 Aufgabe 8: Client Simulator

```
// bin/client.dart

import 'dart:async';
import 'dart:convert';
import 'dart:io';
import 'package:web_socket_channel/web_socket_channel.dart';

void main() async {
  final url = Platform.environment['WS_URL'] ?? 'ws://localhost:8080/ws';

  print('Connecting to $url...');

  final channel = WebSocketChannel.connect(Uri.parse(url));

  // Auf Server-Nachrichten hören
  channel.stream.listen(
    (data) {
      final message = jsonDecode(data as String);
      _printMessage(message);
    },
    onDone: () {
      print('Connection closed');
      exit(0);
    },
    onError: (error) {
      print('Error: $error');
      exit(1);
    },
  );
};

print('Connected! Commands:');
print('  sub <channel>    - Subscribe to channel');
print('  unsub <channel> - Unsubscribe from channel');
print('  name <name>       - Set your name');
print('  list              - List subscriptions');
print('  ping             - Send ping');
print('  quit             - Exit');
print('');

// Stdin lesen
await for (final line in
  ↪ stdin.transform(utf8.decoder).transform(LineSplitter())) {
  final parts = line.trim().split(' ');
  if (parts.isEmpty) continue;

  final cmd = parts[0].toLowerCase();
  final arg = parts.length > 1 ? parts.sublist(1).join(' ') : null;

  switch (cmd) {
    case 'sub':
      if (arg != null && arg.isNotEmpty) {
```

```
        _send(channel, {'type': 'subscribe', 'channel': arg});
        print('-> Subscribing to "$arg"...');
    } else {
        print('Usage: sub <channel>');
    }
    break;

case 'unsub':
    if (arg != null && arg.isNotEmpty) {
        _send(channel, {'type': 'unsubscribe', 'channel': arg});
        print('-> Unsubscribing from "$arg"...');
    } else {
        print('Usage: unsub <channel>');
    }
    break;

case 'name':
    if (arg != null && arg.isNotEmpty) {
        _send(channel, {'type': 'set_name', 'name': arg});
        print('-> Setting name to "$arg"...');
    } else {
        print('Usage: name <your-name>');
    }
    break;

case 'list':
    _send(channel, {'type': 'list_subscriptions'});
    print('-> Requesting subscriptions...');
    break;

case 'ping':
    _send(channel, {'type': 'ping'});
    print('-> Ping sent');
    break;

case 'quit':
case 'exit':
    print('Goodbye!');
    channel.sink.close();
    exit(0);

case 'help':
    print('Commands: sub, unsub, name, list, ping, quit');
    break;

default:
    print('Unknown command: $cmd (type "help" for commands)');
}
}
}
```

```

void _send(WebSocketChannel channel, Map<String, dynamic> message) {
    channel.sink.add(jsonEncode(message));
}

void _printMessage(Map<String, dynamic> message) {
    final type = message['type'] as String?;

    switch (type) {
        case 'welcome':
            print('[x] Connected as ${message['clientId']}');
            break;

        case 'subscribed':
            print('[x] Subscribed to "${message['channel']}"
↪ (${message['subscriberCount']} subscribers)');
            break;

        case 'unsubscribed':
            print('[x] Unsubscribed from "${message['channel']}"');
            break;

        case 'name_set':
            print('[x] Name set to "${message['name']}"');
            break;

        case 'notification':
            print('');
            print('=====');
            print('|| □ NOTIFICATION [${message['channel']}]');
            print('|| Title: ${message['title']}');
            print('|| Body: ${message['body']}');
            if (message['data'] != null) {
                print('|| Data: ${message['data']}');
            }
            print('=====');
            print('');
            break;

        case 'subscriptions':
            final channels = message['channels'] as List;
            print('[x] Your subscriptions: ${channels.isEmpty ? '(none)' :
↪ channels.join(', ')}');
            break;

        case 'pong':
            print('[x] Pong received');
            break;

        case 'error':
            print('[ ] Error: ${message['error']}');
            break;
    }
}

```

```

    case 'server_shutdown':
      print('⚠ Server is shutting down: ${message['message']}');
      break;

    default:
      print('<- $message');
  }
}

```

#### 5.1.2.9 Bonus: Heartbeat

```

// lib/heartbeat.dart

import 'dart:async';
import 'connection_manager.dart';

class HeartbeatManager {
  final Duration pingInterval;
  final Duration timeout;
  final ConnectionManager connections;
  final void Function(String clientId)? onTimeout;

  final Map<String, DateTime> _lastPong = {};
  Timer? _timer;

  HeartbeatManager({
    required this.connections,
    this.pingInterval = const Duration(seconds: 30),
    this.timeout = const Duration(seconds: 60),
    this.onTimeout,
  });

  void start() {
    _timer = Timer.periodic(pingInterval, (_) {
      _checkConnections();
    });
    print('Heartbeat started (interval: ${pingInterval.inSeconds}s, timeout: ↪
↪ ${timeout.inSeconds}s)');
  }

  void _checkConnections() {
    final now = DateTime.now();
    final timedOut = <String>[];

    for (final client in connections.clients) {
      final lastPong = _lastPong[client.id];

      // Neuer Client ohne Pong
      if (lastPong == null) {

```

```

        _lastPong[client.id] = client.connectedAt;
    }
    // Timeout prüfen
    else if (now.difference(lastPong) > timeout) {
        timedOut.add(client.id);
        continue;
    }

    // Ping senden
    client.send({
        'type': 'ping',
        'timestamp': now.millisecondsSinceEpoch,
    });
}

// Timeouts verarbeiten
for (final clientId in timedOut) {
    print('Client $clientId timed out');
    _lastPong.remove(clientId);

    final client = connections.getClient(clientId);
    if (client != null) {
        client.socket.sink.close(1000, 'Ping timeout');
    }

    onTimeout?.call(clientId);
}

void handlePong(String clientId) {
    _lastPong[clientId] = DateTime.now();
}

void removeClient(String clientId) {
    _lastPong.remove(clientId);
}

void stop() {
    _timer?.cancel();
    _timer = null;
    _lastPong.clear();
    print('Heartbeat stopped');
}
}

```

#### Integration in Handler

```

// In ws_handler.dart erweitern

class NotificationHandler {
    final ConnectionManager connections;

```

```

final NotificationService notifications;
final HeartbeatManager heartbeat;

NotificationHandler(this.connections, this.notifications)
  : heartbeat = HeartbeatManager(
      connections: connections,
      onTimeout: (clientId) {
        notifications.removeClient(clientId);
        connections.removeConnection(clientId);
      },
    ) {
  heartbeat.start();
}

void _handleMessage(Client client, String data) {
  // ... existing code ...

  switch (type) {
    case 'pong':
      heartbeat.handlePong(client.id);
      break;

    // ... other cases ...
  }
}

void _handleDisconnect(Client client) {
  heartbeat.removeClient(client.id);
  notifications.removeClient(client.id);
  connections.removeConnection(client.id);
}
}

```

#### 5.1.2.10 Projektstruktur

```

notification_server/
+-- bin/
|   +-- server.dart      # Hauptserver
|   +-- client.dart      # Test-Client
+-- lib/
|   +-- connection_manager.dart
|   +-- notification_service.dart
|   +-- messages.dart
|   +-- ws_handler.dart
|   +-- api_handler.dart
|   +-- heartbeat.dart
+-- pubspec.yaml
+-- README.md

```

#### 5.1.2.11 Test-Szenario

```
# Terminal 1: Server starten
dart run bin/server.dart

# Terminal 2: Client 1
dart run bin/client.dart
> name Alice
> sub news
> sub sports

# Terminal 3: Client 2
dart run bin/client.dart
> name Bob
> sub news

# Terminal 4: Notification senden
curl -X POST http://localhost:8080/api/notify/news \
  -H "Content-Type: application/json" \
  -d '{"title": "Breaking", "body": "Big news!"}'

# Stats abrufen
curl http://localhost:8080/api/stats
curl http://localhost:8080/api/channels
curl http://localhost:8080/api/clients
```

### 5.1.3 Ressourcen

#### 5.1.3.1 Offizielle Dokumentation

- shelf\_web\_socket Package
- web\_socket\_channel Package
- WebSocket Protocol RFC 6455
- MDN WebSocket API

#### 5.1.3.2 Cheat Sheet: WebSocket Handler

```
import 'package:shelf_web_socket/shelf_web_socket.dart';
import 'package:web_socket_channel/web_socket_channel.dart';

// Einfacher Handler
final handler = WebSocketHandler((WebSocketChannel socket) {
  // Nachricht senden
  socket.sink.add('Hello');
  socket.sink.add(jsonEncode({'type': 'welcome'}));

  // Auf Nachrichten hören
  socket.stream.listen(
    (message) => print('Received: $message'),
    onDone: () => print('Disconnected'),
    onError: (e) => print('Error: $e'),
  );
});
```

```
);  
});  
  
// Mit Router  
router.get('/ws', websocketHandler(handleConnection));
```

### 5.1.3.3 Cheat Sheet: Message Protocol

```
// Standard Message Format  
{  
  "type": "message_type",  
  "payload": { ... },  
  "timestamp": "2024-01-01T12:00:00Z"  
}  
  
// Client -> Server Messages  
{"type": "auth", "token": "..."}  
{"type": "subscribe", "channel": "news"}  
{"type": "unsubscribe", "channel": "news"}  
{"type": "message", "text": "Hello"}  
{"type": "ping"}  
  
// Server -> Client Messages  
{"type": "welcome", "clientId": "..."}  
{"type": "subscribed", "channel": "news"}  
{"type": "notification", "channel": "news", "data": {...}}  
{"type": "error", "message": "..."}  
{"type": "pong"}
```

### 5.1.3.4 Cheat Sheet: Connection Management

```
class ConnectionManager {  
  final Map<String, WebSocketChannel> _clients = {};  
  
  // Client hinzufügen  
  void add(String id, WebSocketChannel socket) {  
    _clients[id] = socket;  
  }  
  
  // Client entfernen  
  void remove(String id) {  
    _clients.remove(id);  
  }  
  
  // An alle senden  
  void broadcast(String message) {  
    for (final socket in _clients.values) {  
      socket.sink.add(message);  
    }  
  }  
}
```

```
}

// An bestimmten Client
void sendTo(String id, String message) {
    _clients[id]?.sink.add(message);
}

// An alle außer einem
void broadcastExcept(String excludeId, String message) {
    for (final entry in _clients.entries) {
        if (entry.key != excludeId) {
            entry.value.sink.add(message);
        }
    }
}
}
```

#### 5.1.3.5 Cheat Sheet: Channel/Room Pattern

```
class Room {
    final String name;
    final Set<String> members = {};

    void join(String clientId) => members.add(clientId);
    void leave(String clientId) => members.remove(clientId);
    bool hasMember(String id) => members.contains(id);
}

class RoomManager {
    final Map<String, Room> _rooms = {};
    final ConnectionManager _connections;

    void join(String clientId, String roomName) {
        final room = _rooms.putIfAbsent(roomName, () => Room(roomName));
        room.join(clientId);
    }

    void broadcast(String roomName, String message) {
        final room = _rooms[roomName];
        if (room == null) return;

        for (final memberId in room.members) {
            _connections.sendTo(memberId, message);
        }
    }
}
```

### 5.1.3.6 Cheat Sheet: Heartbeat

```
// Server-seitig
Timer.periodic(Duration(seconds: 30), (_) {
  for (final client in clients) {
    client.send({'type': 'ping'});
  }
});

// Timeout prüfen
if (DateTime.now().difference(lastPong) > Duration(seconds: 60)) {
  client.socket.sink.close(1000, 'Timeout');
}

// Client-seitig
socket.stream.listen((data) {
  final msg = jsonDecode(data);
  if (msg['type'] == 'ping') {
    socket.sink.add(jsonEncode({'type': 'pong'}));
  }
});
```

### 5.1.3.7 Cheat Sheet: Reconnection (Client)

```
class ReconnectingWebSocket {
  WebSocketChannel? _channel;
  final String url;
  int _retryCount = 0;
  final int maxRetries = 5;
  final Duration baseDelay = Duration(seconds: 1);

  Future<void> connect() async {
    try {
      _channel = WebSocketChannel.connect(Uri.parse(url));
      _retryCount = 0;

      _channel!.stream.listen(
        onMessage,
        onDone: _reconnect,
        onError: (_) => _reconnect(),
      );
    } catch (e) {
      _reconnect();
    }
  }

  void _reconnect() {
    if (_retryCount >= maxRetries) {
      print('Max retries reached');
      return;
    }
  }
}
```

```

    // Exponential Backoff
    final delay = baseDelay * (1 << _retryCount);
    _retryCount++;

    print('Reconnecting in ${delay.inSeconds}s...');
    Future.delayed(delay, connect);
  }
}

```

#### 5.1.3.8 Cheat Sheet: Close Codes

```

// Standard Close Codes
const CLOSE_NORMAL = 1000;      // Normale Beendigung
const CLOSE_GOING_AWAY = 1001;  // Server/Client geht offline
const CLOSE_PROTOCOL_ERROR = 1002; // Protokollfehler
const CLOSE_UNSUPPORTED = 1003; // Nicht unterstützter Datentyp
const CLOSE_NO_STATUS = 1005;   // Kein Status (reserviert)
const CLOSE_ABNORMAL = 1006;    // Abnormaler Abbruch
const CLOSE_INVALID_DATA = 1007; // Ungültige Daten
const CLOSE_POLICY = 1008;      // Policy-Verletzung
const CLOSE_TOO_LARGE = 1009;   // Nachricht zu groß
const CLOSE_EXTENSION = 1010;   // Extension fehlt
const CLOSE_INTERNAL = 1011;    // Interner Fehler

// Verbindung schließen
socket.sink.close(1000, 'Normal closure');

```

#### 5.1.3.9 Cheat Sheet: Flutter WebSocket Client

```

import 'package:web_socket_channel/web_socket_channel.dart';

class WebSocketService {
  WebSocketChannel? _channel;
  final _messageController = StreamController<Map<String, dynamic>>.broadcast();

  Stream<Map<String, dynamic>> get messages => _messageController.stream;

  void connect(String url) {
    _channel = WebSocketChannel.connect(Uri.parse(url));

    _channel!.stream.listen(
      (data) {
        final message = jsonDecode(data);
        _messageController.add(message);
      },
      onError: (e) => _messageController.addError(e),
    );
  }
}

```

```

void send(Map<String, dynamic> message) {
  _channel?.sink.add(jsonEncode(message));
}

void dispose() {
  _channel?.sink.close();
  _messageController.close();
}
}

// In Widget verwenden
class ChatWidget extends StatefulWidget {
  @override
  _ChatWidgetState createState() => _ChatWidgetState();
}

class _ChatWidgetState extends State<ChatWidget> {
  final _ws = WebSocketService();

  @override
  void initState() {
    super.initState();
    _ws.connect('ws://localhost:8080/ws');
  }

  @override
  Widget build(BuildContext context) {
    return StreamBuilder<Map<String, dynamic>>(
      stream: _ws.messages,
      builder: (context, snapshot) {
        if (snapshot.hasData) {
          return Text('Message: ${snapshot.data}');
        }
        return CircularProgressIndicator();
      },
    );
  }

  @override
  void dispose() {
    _ws.dispose();
    super.dispose();
  }
}

```

#### 5.1.3.10 Best Practices

DO

1. **Strukturiertes Message-Format** - JSON mit type-Feld

2. **Heartbeat implementieren** - Connection-Health prüfen
3. **Reconnection-Logik** - Mit Exponential Backoff
4. **Graceful Shutdown** - Clients benachrichtigen
5. **Rate Limiting** - Spam verhindern
6. **Message Validation** - Input prüfen
7. **Error Handling** - Fehler abfangen und loggen

#### DON'T

1. **Große Payloads** - WebSocket für kleine Nachrichten
2. **Synchrone Operationen** - Blockiert Event Loop
3. **Sensitive Daten unverschlüsselt** - WSS verwenden
4. **Unbegrenzter History** - Memory Leaks
5. **Fehlende Timeouts** - Zombie Connections

#### 5.1.3.11 Skalierung mit Redis

```
// Pub/Sub für Multi-Server
import 'package:redis/redis.dart';

class ScalableNotifications {
  late RedisConnection _pub;
  late RedisConnection _sub;

  Future<void> init() async {
    _pub = await RedisConnection.connect('localhost', 6379);
    _sub = await RedisConnection.connect('localhost', 6379);

    // Subscribe
    final pubsub = PubSub(_sub);
    pubsub.subscribe(['notifications']).listen((msg) {
      // An lokale Clients weiterleiten
      localBroadcast(msg.message);
    });
  }

  void publish(String channel, Map<String, dynamic> message) {
    _pub.execute(['PUBLISH', channel, jsonEncode(message)]);
  }
}
```

#### 5.1.3.12 Testing WebSockets

```
import 'package:test/test.dart';
import 'package:web_socket_channel/web_socket_channel.dart';

void main() {
  test('WebSocket echo', () async {
    final channel = WebSocketChannel.connect(
      Uri.parse('ws://localhost:8080/ws'),
    );
```

```
// Nachricht senden
channel.sink.add('Hello');

// Antwort erwarten
final response = await channel.stream.first;
expect(response, contains('Hello'));

await channel.sink.close();
});
}
```

### 5.1.3.13 Tools

- **websocat** - WebSocket CLI Client

```
cargo install websocat
websocat ws://localhost:8080/ws
```

- **wscat** - Node.js WebSocket Client

```
npm install -g wscat
wscat -c ws://localhost:8080/ws
```

- **Postman** - WebSocket Support in neueren Versionen
- **Browser DevTools** - Network Tab -> WS Filter

## 5.2 Einheit 9.2: Background Jobs & Scheduling

### 5.2.0.1 Lernziele

- Asynchrone Hintergrundaufgaben verstehen
- Job Queues implementieren
- Scheduled Tasks mit Cron-ähnlicher Syntax
- Worker-Pattern für skalierbare Verarbeitung

### 5.2.0.2 Warum Background Jobs?

Probleme mit synchroner Verarbeitung

```
// ☐ Schlecht: Blockiert den Request
router.post('/api/orders', (Request request) async {
  final order = await parseOrder(request);
  await saveOrder(order);

  // Diese Operationen blockieren die Response:
  await sendConfirmationEmail(order); // 2-5 Sekunden
  await generateInvoicePdf(order);    // 3-10 Sekunden
  await notifyWarehouse(order);       // 1-3 Sekunden
  await updateAnalytics(order);        // 1-2 Sekunden

  return Response.ok('Order created'); // Nutzer wartet 7-20 Sekunden!
```

```
});

// ☐ Besser: Jobs in Queue
router.post('/api/orders', (Request request) async {
  final order = await parseOrder(request);
  await saveOrder(order);

  // Jobs zur späteren Verarbeitung einreihen
  await jobQueue.enqueue(SendEmailJob(order.id));
  await jobQueue.enqueue(GenerateInvoiceJob(order.id));
  await jobQueue.enqueue(NotifyWarehouseJob(order.id));

  return Response.ok('Order created'); // Sofortige Response!
});
```

### Anwendungsfälle

Kategorie	Beispiele
<b>Email</b>	Bestätigungen, Newsletter, Reports
<b>Verarbeitung</b>	PDF-Generierung, Bildkonvertierung
<b>Integration</b>	Webhook-Aufrufe, API-Synchronisation
<b>Cleanup</b>	Alte Daten löschen, Temp-Files
<b>Scheduled</b>	Tägliche Backups, Reports, Erinnerungen

#### 5.2.0.3 Einfache Job Queue

##### Job-Klasse

```
// lib/jobs/job.dart

enum JobStatus { pending, running, completed, failed }

abstract class Job {
  final String id;
  final DateTime createdAt;
  JobStatus status;
  int attempts;
  final int maxAttempts;
  DateTime? startedAt;
  DateTime? completedAt;
  String? error;

  Job({
    String? id,
    this.maxAttempts = 3,
  }) : id = id ?? Uuid().v4(),
      createdAt = DateTime.now(),
      status = JobStatus.pending,
      attempts = 0;

  /// Job-Typ für Serialisierung
```

```
String get type;

/// Job ausführen
Future<void> execute();

/// Zu JSON konvertieren
Map<String, dynamic> toJson();

/// Sollte bei Fehler wiederholt werden?
bool get shouldRetry => attempts < maxAttempts;

/// Delay vor Retry (exponential backoff)
Duration get retryDelay => Duration(seconds: 1 << attempts);
}
```

#### Konkrete Jobs

```
// lib/jobs/email_job.dart

class SendEmailJob extends Job {
  final String to;
  final String subject;
  final String body;

  SendEmailJob({
    required this.to,
    required this.subject,
    required this.body,
    super.maxAttempts = 3,
  });

  @override
  String get type => 'send_email';

  @override
  Future<void> execute() async {
    print('Sending email to $to: $subject');
    // Email-Service aufrufen
    await EmailService.send(
      to: to,
      subject: subject,
      body: body,
    );
  }

  @override
  Map<String, dynamic> toJson() => {
    'type': type,
    'id': id,
    'to': to,
    'subject': subject,
  }
}
```

```
        'body': body,
        'attempts': attempts,
        'maxAttempts': maxAttempts,
    };

    factory SendEmailJob.fromJson(Map<String, dynamic> json) {
        final job = SendEmailJob(
            to: json['to'],
            subject: json['subject'],
            body: json['body'],
            maxAttempts: json['maxAttempts'] ?? 3,
        );
        job.attempts = json['attempts'] ?? 0;
        return job;
    }
}

// lib/jobs/pdf_job.dart

class GeneratePdfJob extends Job {
    final int orderId;
    final String outputPath;

    GeneratePdfJob({
        required this.orderId,
        required this.outputPath,
        super.maxAttempts = 2,
    });

    @override
    String get type => 'generate_pdf';

    @override
    Future<void> execute() async {
        print('Generating PDF for order $orderId');

        final order = await OrderRepository.findById(orderId);
        if (order == null) {
            throw Exception('Order not found: $orderId');
        }

        await PdfService.generateInvoice(order, outputPath);
    }

    @override
    Map<String, dynamic> toJson() => {
        'type': type,
        'id': id,
        'orderId': orderId,
        'outputPath': outputPath,
    };
}
```

```
}
```

### In-Memory Queue

```
// lib/queue/memory_queue.dart

import 'dart:async';
import 'dart:collection';

class MemoryJobQueue {
  final Queue<Job> _pending = Queue();
  final List<Job> _completed = [];
  final List<Job> _failed = [];

  bool _isProcessing = false;
  final _controller = StreamController<Job>.broadcast();

  Stream<Job> get onJobCompleted => _controller.stream;

  /// Job zur Queue hinzufügen
  Future<void> enqueue(Job job) async {
    _pending.add(job);
    print('Job enqueued: ${job.type} (${job.id})');

    // Verarbeitung starten falls nicht aktiv
    if (!_isProcessing) {
      _processQueue();
    }
  }

  /// Queue verarbeiten
  Future<void> _processQueue() async {
    if (_isProcessing) return;
    _isProcessing = true;

    while (_pending.isNotEmpty) {
      final job = _pending.removeFirst();
      await _executeJob(job);
    }

    _isProcessing = false;
  }

  /// Einzelnen Job ausführen
  Future<void> _executeJob(Job job) async {
    job.status = JobStatus.running;
    job.startedAt = DateTime.now();
    job.attempts++;

    try {
      await job.execute();
    }
  }
}
```

```

        job.status = JobStatus.completed;
        job.completedAt = DateTime.now();
        _completed.add(job);
        _controller.add(job);

        print('Job completed: ${job.type} (${job.id})');
    } catch (e) {
        job.error = e.toString();
        print('Job failed: ${job.type} (${job.id}) - $e');

        if (job.shouldRetry) {
            // Zurück in Queue mit Delay
            print('Retrying in ${job.retryDelay.inSeconds}s...');
            Future.delayed(job.retryDelay, () {
                job.status = JobStatus.pending;
                _pending.add(job);
                _processQueue();
            });
        } else {
            job.status = JobStatus.failed;
            _failed.add(job);
        }
    }
}

/// Statistiken
Map<String, int> get stats => {
    'pending': _pending.length,
    'completed': _completed.length,
    'failed': _failed.length,
};
}

```

#### 5.2.0.4 Persistente Queue mit Redis

Redis-basierte Queue

```

// lib/queue/redis_queue.dart

import 'package:redis/redis.dart';

class RedisJobQueue {
    final RedisConnection _redis;
    final String _queueName;
    final JobRegistry _registry;

    RedisJobQueue(this._redis, this._queueName, this._registry);

    /// Job einreihen
    Future<void> enqueue(Job job) async {

```

```
final json = jsonEncode(job.toJson());
await _redis.execute(['RPUSH', _queueName, json]);
print('Job enqueued: ${job.type} (${job.id})');
}

/// Job mit Verzögerung einreihen
Future<void> enqueueDelayed(Job job, Duration delay) async {
  final executeAt = DateTime.now().add(delay).millisecondsSinceEpoch;
  final json = jsonEncode(job.toJson());
  await _redis.execute([
    'ZADD',
    '$_queueName:delayed',
    executeAt.toString(),
    json,
  ]);
}

/// Nächsten Job holen (blockierend)
Future<Job?> dequeue({Duration timeout = const Duration(seconds: 5)}) async {
  final result = await _redis.execute([
    'BLOPP',
    _queueName,
    timeout.inSeconds.toString(),
  ]);

  if (result == null || result.isEmpty) return null;

  final json = jsonDecode(result[1] as String);
  return _registry.createJob(json);
}

/// Verzögerte Jobs prüfen
Future<void> processDelayed() async {
  final now = DateTime.now().millisecondsSinceEpoch;

  // Jobs holen die fällig sind
  final result = await _redis.execute([
    'ZRANGEBYSCORE',
    '$_queueName:delayed',
    '0',
    now.toString(),
  ]);

  if (result == null || result.isEmpty) return;

  for (final json in result) {
    // In Hauptqueue verschieben
    await _redis.execute(['RPUSH', _queueName, json]);
    await _redis.execute(['ZREM', '$_queueName:delayed', json]);
  }
}
```

```

    /// Queue-Länge
    Future<int> get length async {
      final result = await _redis.execute(['LLEN', _queueName]);
      return result as int;
    }
  }
}

```

Job Registry für Deserialisierung

```

// lib/queue/job_registry.dart

typedef JobFactory = Job Function(Map<String, dynamic> json);

class JobRegistry {
  final Map<String, JobFactory> _factories = {};

  void register(String type, JobFactory factory) {
    _factories[type] = factory;
  }

  Job createJob(Map<String, dynamic> json) {
    final type = json['type'] as String;
    final factory = _factories[type];

    if (factory == null) {
      throw Exception('Unknown job type: $type');
    }

    return factory(json);
  }
}

// Verwendung
final registry = JobRegistry()
  ..register('send_email', SendEmailJob.fromJson)
  ..register('generate_pdf', GeneratePdfJob.fromJson)
  ..register('notify_webhook', WebhookJob.fromJson);

```

### 5.2.0.5 Worker Pattern

Worker-Klasse

```

// lib/worker/worker.dart

class Worker {
  final String id;
  final RedisJobQueue queue;
  final int concurrency;

  bool _isRunning = false;
  final List<Future<void>> _activeTasks = [];

```

```
Worker({
  required this.queue,
  this.concurrency = 1,
  String? id,
}) : id = id ?? 'worker-${Uuid().v4().substring(0, 8)}';

/// Worker starten
Future<void> start() async {
  if (_isRunning) return;
  _isRunning = true;

  print('Worker $id started (concurrency: $concurrency)');

  while (_isRunning) {
    // Auf freien Slot warten
    while (_activeTasks.length >= concurrency) {
      await Future.any(_activeTasks);
      _activeTasks.removeWhere((f) => f == Future.value());
    }

    // Verzögerte Jobs prüfen
    await queue.processDelayed();

    // Job holen
    final job = await queue.dequeue();
    if (job == null) continue;

    // Job asynchron verarbeiten
    final task = _processJob(job);
    _activeTasks.add(task);
    task.then((_) => _activeTasks.remove(task));
  }
}

Future<void> _processJob(Job job) async {
  print('[${id}] Processing: ${job.type} (${job.id})');
  job.status = JobStatus.running;
  job.attempts++;

  try {
    await job.execute();
    job.status = JobStatus.completed;
    print('[${id}] Completed: ${job.type} (${job.id})');
  } catch (e, stack) {
    print('[${id}] Failed: ${job.type} (${job.id}) - $e');

    if (job.shouldRetry) {
      print('[${id}] Scheduling retry in ${job.retryDelay.inSeconds}s');
      await queue.enqueueDelayed(job, job.retryDelay);
    } else {
```

```

        // In Dead Letter Queue
        await _moveToDeadLetter(job, e.toString());
    }
}

Future<void> _moveToDeadLetter(Job job, String error) async {
    // Job mit Fehlerinfo speichern
    final deadJob = {
        ...job.toJson(),
        'error': error,
        'failedAt': DateTime.now().toIso8601String(),
    };
    // In Redis speichern für manuelle Überprüfung
    print('[${id}] Moved to dead letter: ${job.id}');
}

/// Worker stoppen
Future<void> stop() async {
    _isRunning = false;
    // Aktive Tasks abwarten
    if (_activeTasks.isNotEmpty) {
        print('[${id}] Waiting for ${_activeTasks.length} tasks to complete...');
        await Future.wait(_activeTasks);
    }
    print('[${id}] Stopped');
}
}

```

### Worker-Pool

```

// lib/worker/worker_pool.dart

class WorkerPool {
    final List<Worker> _workers = [];
    final RedisJobQueue queue;
    final int workerCount;
    final int concurrencyPerWorker;

    WorkerPool({
        required this.queue,
        this.workerCount = 4,
        this.concurrencyPerWorker = 2,
    });

    /// Pool starten
    Future<void> start() async {
        for (var i = 0; i < workerCount; i++) {
            final worker = Worker(
                queue: queue,
                concurrency: concurrencyPerWorker,
            );
            _workers.add(worker);
        }
    }
}

```

```

        id: 'worker-$i',
      );
      _workers.add(worker);
      worker.start(); // Non-blocking
    }

    print('Worker pool started: $workerCount workers');
  }

  /// Pool stoppen
  Future<void> stop() async {
    print('Stopping worker pool...');
    await Future.wait(_workers.map((w) => w.stop()));
    _workers.clear();
    print('Worker pool stopped');
  }

  /// Statistiken
  Map<String, dynamic> get stats => {
    'workers': _workers.length,
    'totalConcurrency': workerCount * concurrencyPerWorker,
  };
}

```

### 5.2.0.6 Scheduled Tasks (Cron)

Cron Parser

```

// lib/scheduler/cron.dart

class CronExpression {
  final String expression;
  final List<int>? minutes;
  final List<int>? hours;
  final List<int>? daysOfMonth;
  final List<int>? months;
  final List<int>? daysOfWeek;

  CronExpression._(
    this.expression, {
    this.minutes,
    this.hours,
    this.daysOfMonth,
    this.months,
    this.daysOfWeek,
  });

  /// Cron-Ausdruck parsen
  /// Format: minute hour dayOfMonth month dayOfWeek
  /// Beispiele:
  /// "0 * * * *" - Jede Stunde

```

```

///  "0 0 * * *"      - Täglich um Mitternacht
///  "0 9 * * 1-5"    - Werktags um 9 Uhr
///  "*/15 * * * *"   - Alle 15 Minuten
factory CronExpression.parse(String expression) {
  final parts = expression.split(' ');
  if (parts.length != 5) {
    throw FormatException('Invalid cron expression: $expression');
  }

  return CronExpression._(
    expression,
    minutes: _parseField(parts[0], 0, 59),
    hours: _parseField(parts[1], 0, 23),
    daysOfMonth: _parseField(parts[2], 1, 31),
    months: _parseField(parts[3], 1, 12),
    daysOfWeek: _parseField(parts[4], 0, 6),
  );
}

static List<int>? _parseField(String field, int min, int max) {
  if (field == '*') return null; // Alle Werte

  final values = <int>{};

  for (final part in field.split(',')) {
    if (part.contains('/')) {
      // Step: */5 oder 0-30/5
      final stepParts = part.split('/');
      final step = int.parse(stepParts[1]);
      final range = stepParts[0] == '*'
        ? [min, max]
        : stepParts[0].split('-').map(int.parse).toList();

      for (var i = range[0]; i <= (range.length > 1 ? range[1] : max); i += ↵
↵ step) {
        values.add(i);
      }
    } else if (part.contains('-')) {
      // Range: 1-5
      final rangeParts = part.split('-').map(int.parse).toList();
      for (var i = rangeParts[0]; i <= rangeParts[1]; i++) {
        values.add(i);
      }
    } else {
      // Einzelwert
      values.add(int.parse(part));
    }
  }

  return values.toList()..sort();
}

```

```

/// Prüft ob DateTime zum Ausdruck passt
bool matches(DateTime dt) {
  if (minutes != null && !minutes!.contains(dt.minute)) return false;
  if (hours != null && !hours!.contains(dt.hour)) return false;
  if (daysOfMonth != null && !daysOfMonth!.contains(dt.day)) return false;
  if (months != null && !months!.contains(dt.month)) return false;
  if (daysOfWeek != null && !daysOfWeek!.contains(dt.weekday % 7)) return  ↪
  ↪ false;
  return true;
}

/// Nächste Ausführungszeit berechnen
DateTime nextRun([DateTime? from]) {
  var dt = from ?? DateTime.now();
  dt = DateTime(dt.year, dt.month, dt.day, dt.hour, dt.minute + 1);

  // Maximal 2 Jahre suchen
  final maxDate = dt.add(Duration(days: 730));

  while (dt.isBefore(maxDate)) {
    if (matches(dt)) return dt;
    dt = dt.add(Duration(minutes: 1));
  }

  throw StateError('No matching date found');
}
}

```

### Scheduled Task

```

// lib/scheduler/scheduled_task.dart

typedef TaskCallback = Future<void> Function();

class ScheduledTask {
  final String name;
  final CronExpression schedule;
  final TaskCallback callback;
  DateTime? lastRun;
  DateTime? nextRun;
  bool isRunning = false;

  ScheduledTask({
    required this.name,
    required String cron,
    required this.callback,
  }) : schedule = CronExpression.parse(cron) {
    nextRun = schedule.nextRun();
  }
}

```

```

Future<void> execute() async {
  if (isRunning) {
    print('Task $name already running, skipping');
    return;
  }

  isRunning = true;
  lastRun = DateTime.now();

  try {
    print('Executing scheduled task: $name');
    await callback();
    print('Scheduled task completed: $name');
  } catch (e) {
    print('Scheduled task failed: $name - $e');
  } finally {
    isRunning = false;
    nextRun = schedule.nextRun();
  }
}
}

```

### Scheduler

```

// lib/scheduler/scheduler.dart

class Scheduler {
  final List<ScheduledTask> _tasks = [];
  Timer? _timer;
  bool _isRunning = false;

  /// Task hinzufügen
  void schedule(String name, String cron, TaskCallback callback) {
    _tasks.add(ScheduledTask(
      name: name,
      cron: cron,
      callback: callback,
    ));
    print('Scheduled: $name ($cron)');
  }

  /// Scheduler starten
  void start() {
    if (_isRunning) return;
    _isRunning = true;

    // Jede Minute prüfen
    _timer = Timer.periodic(Duration(minutes: 1), (_) => _tick());
    print('Scheduler started with ${_tasks.length} tasks');

    // Einmal initial
  }
}

```

```

    _tick();
}

void _tick() {
    final now = DateTime.now();

    for (final task in _tasks) {
        if (task.nextRun != null &&
            now.isAfter(task.nextRun!) &&
            !task.isRunning) {
            task.execute();
        }
    }
}

/// Scheduler stoppen
void stop() {
    _timer?.cancel();
    _isRunning = false;
    print('Scheduler stopped');
}

/// Nächste Ausführungen anzeigen
void printSchedule() {
    print('\nScheduled Tasks:');
    for (final task in _tasks) {
        print('  ${task.name}: next run at ${task.nextRun}');
    }
}
}

```

### Verwendung

```

void main() {
    final scheduler = Scheduler();

    // Täglich um Mitternacht
    scheduler.schedule('cleanup', '0 0 * * *', () async {
        await cleanupOldFiles();
    });

    // Alle 15 Minuten
    scheduler.schedule('health_check', '* /15 * * * *', () async {
        await checkExternalServices();
    });

    // Werktags um 9 Uhr
    scheduler.schedule('daily_report', '0 9 * * 1-5', () async {
        await generateDailyReport();
    });
}

```

```
// Jeden Sonntag um 2 Uhr (Backup)
scheduler.schedule('weekly_backup', '0 2 * * 0', () async {
  await performBackup();
});

scheduler.printSchedule();
scheduler.start();
}
```

### 5.2.0.7 Kombinierte Architektur

Vollständiges Setup

```
// bin/worker.dart

import 'dart:io';

void main() async {
  // Redis-Verbindung
  final redis = await RedisConnection.connect('localhost', 6379);

  // Job Registry
  final registry = JobRegistry()
    ..register('send_email', SendEmailJob.fromJson)
    ..register('generate_pdf', GeneratePdfJob.fromJson)
    ..register('sync_data', SyncDataJob.fromJson);

  // Job Queue
  final queue = RedisJobQueue(redis, 'jobs', registry);

  // Worker Pool
  final workerPool = WorkerPool(
    queue: queue,
    workerCount: int.parse(Platform.environment['WORKERS'] ?? '4'),
    concurrencyPerWorker: 2,
  );

  // Scheduler
  final scheduler = Scheduler()
    ..schedule('process_delayed', '* * * * *', () async {
      await queue.processDelayed();
    })
    ..schedule('cleanup_completed', '0 * * * *', () async {
      // Abgeschlossene Jobs aufräumen
    });

  // Starten
  await workerPool.start();
  scheduler.start();

  // Graceful Shutdown
}
```

```
ProcessSignal.sigint.watch().listen((_) async {
  print('\nShutting down...');
  scheduler.stop();
  await workerPool.stop();
  await redis.close();
  exit(0);
});

print('Worker service running. Press Ctrl+C to stop.');
```

API zum Einreihen von Jobs

```
// In der Haupt-API

router.post('/api/orders', (Request request) async {
  final order = await createOrder(request);

  // Jobs einreihen
  await jobQueue.enqueue(SendEmailJob(
    to: order.customerEmail,
    subject: 'Order Confirmation',
    body: 'Your order ${order.id} has been received.',
  ));

  await jobQueue.enqueue(GeneratePdfJob(
    orderId: order.id,
    outputPath: '/invoices/${order.id}.pdf',
  ));

  // Mit Verzögerung
  await jobQueue.enqueueDelayed(
    FollowUpEmailJob(orderId: order.id),
    Duration(days: 7),
  );

  return Response.ok(jsonEncode(order.toJson()));
});
```

#### 5.2.0.8 Zusammenfassung

- **Background Jobs** entkoppeln langsame Operationen vom Request
- **Job Queue** speichert Jobs zur späteren Verarbeitung
- **Worker** verarbeiten Jobs parallel
- **Retry-Logik** mit Exponential Backoff für Fehlertoleranz
- **Scheduler** für zeitgesteuerte wiederkehrende Aufgaben
- **Redis** für persistente, skalierbare Queues

Nächste Schritte

In der nächsten Einheit behandeln wir Logging & Monitoring.

## 5.2.1 Übung

### 5.2.1.1 Ziel

Implementiere ein Job-System mit Queue, Worker und Scheduler.

### 5.2.1.2 Vorbereitung

Dependencies

```
# pubspec.yaml
dependencies:
  uuid: ^4.0.0
  redis: ^3.0.0 # Optional für Redis-Queue
```

Redis (Optional)

```
docker run -d -p 6379:6379 redis:7
```

### 5.2.1.3 Aufgabe 1: Job-Basisklasse (15 min)

Erstelle die abstrakte Job-Klasse.

```
// lib/jobs/job.dart

import 'package:uuid/uuid.dart';

enum JobStatus { pending, running, completed, failed }
enum JobPriority { low, normal, high, critical }

abstract class Job {
  final String id;
  final DateTime createdAt;
  final JobPriority priority;
  JobStatus status;
  int attempts;
  final int maxAttempts;
  DateTime? startedAt;
  DateTime? completedAt;
  String? error;

  Job({
    String? id,
    this.maxAttempts = 3,
    this.priority = JobPriority.normal,
  }) : id = id ?? Uuid().v4(),
      createdAt = DateTime.now(),
      status = JobStatus.pending,
      attempts = 0;

  /// Job-Typ für Serialisierung
  String get type;

  /// Job ausführen - implementieren in Subklassen
```

```
Future<void> execute();

/// Zu JSON konvertieren
Map<String, dynamic> toJson();

/// Sollte bei Fehler wiederholt werden?
bool get shouldRetry {
  // TODO: Implementieren
}

/// Delay vor Retry (exponential backoff)
Duration get retryDelay {
  // TODO: 2^attempts Sekunden
}

/// Job-Dauer berechnen
Duration? get duration {
  // TODO: Differenz zwischen completedAt und startedAt
}
}
```

#### 5.2.1.4 Aufgabe 2: Konkrete Jobs (20 min)

Implementiere verschiedene Job-Typen.

```
// lib/jobs/email_job.dart

class SendEmailJob extends Job {
  final String to;
  final String subject;
  final String body;
  final List<String>? cc;

  SendEmailJob({
    required this.to,
    required this.subject,
    required this.body,
    this.cc,
    super.maxAttempts = 3,
    super.priority = JobPriority.normal,
  });

  @override
  String get type => 'send_email';

  @override
  Future<void> execute() async {
    // TODO: Simuliere Email-Versand (500ms delay)
    // TODO: 10% Fehlerrate simulieren für Retry-Tests
  }
}
```

```
@override
Map<String, dynamic> toJson() {
  // TODO: Alle Felder serialisieren
}

factory SendEmailJob.fromJson(Map<String, dynamic> json) {
  // TODO: Aus JSON erstellen
}

// lib/jobs/webhook_job.dart

class WebhookJob extends Job {
  final String url;
  final Map<String, dynamic> payload;
  final Map<String, String>? headers;

  WebhookJob({
    required this.url,
    required this.payload,
    this.headers,
    super.maxAttempts = 5,
    super.priority = JobPriority.high,
  });

  @override
  String get type => 'webhook';

  @override
  Future<void> execute() async {
    // TODO: HTTP POST an URL mit payload
    // TODO: Bei nicht-2xx Status Exception werfen
  }

  @override
  Map<String, dynamic> toJson() {
    // TODO
  }

  factory WebhookJob.fromJson(Map<String, dynamic> json) {
    // TODO
  }
}

// lib/jobs/cleanup_job.dart

class CleanupJob extends Job {
  final String directory;
  final int maxAgeDays;

  CleanupJob({
```

```

    required this.directory,
    this.maxAgeDays = 30,
    super.maxAttempts = 1,
    super.priority = JobPriority.low,
  });

  @override
  String get type => 'cleanup';

  @override
  Future<void> execute() async {
    // TODO: Dateien älter als maxAgeDays löschen
    // TODO: Anzahl gelöschter Dateien loggen
  }

  @override
  Map<String, dynamic> toJson() {
    // TODO
  }
}

```

### 5.2.1.5 Aufgabe 3: Job Registry (10 min)

Implementiere die Factory für Job-Deserialisierung.

```

// lib/queue/job_registry.dart

typedef JobFactory = Job Function(Map<String, dynamic> json);

class JobRegistry {
  final Map<String, JobFactory> _factories = {};

  /// Job-Typ registrieren
  void register(String type, JobFactory factory) {
    // TODO
  }

  /// Job aus JSON erstellen
  Job createJob(Map<String, dynamic> json) {
    // TODO: type aus JSON lesen
    // TODO: passende Factory finden
    // TODO: Job erstellen oder Exception werfen
  }

  /// Alle registrierten Typen
  List<String> get registeredTypes {
    // TODO
  }
}

```

### 5.2.1.6 Aufgabe 4: In-Memory Queue (25 min)

Implementiere eine einfache Queue mit Priority-Support.

```
// lib/queue/memory_queue.dart

import 'dart:async';
import 'dart:collection';

class MemoryJobQueue {
  // Priority Queues (high priority first)
  final Map<JobPriority, Queue<Job>> _queues = {
    for (final p in JobPriority.values) p: Queue<Job>(),
  };

  final List<Job> _completed = [];
  final List<Job> _failed = [];
  final _completedController = StreamController<Job>.broadcast();

  Stream<Job> get onJobCompleted => _completedController.stream;

  /// Job einreihen
  void enqueue(Job job) {
    // TODO: In richtige Priority-Queue einfügen
    // TODO: Log ausgeben
  }

  /// Job mit Verzögerung einreihen
  void enqueueDelayed(Job job, Duration delay) {
    // TODO: Timer erstellen der Job später einreicht
  }

  /// Nächsten Job holen (höchste Priority zuerst)
  Job? dequeue() {
    // TODO: Von critical nach low durchgehen
    // TODO: Ersten verfügbaren Job zurückgeben
  }

  /// Job als abgeschlossen markieren
  void markCompleted(Job job) {
    // TODO: Status setzen
    // TODO: In completed-Liste
    // TODO: Event auslösen
  }

  /// Job als fehlgeschlagen markieren
  void markFailed(Job job, String error) {
    // TODO: Error setzen
    // TODO: In failed-Liste oder retry
  }

  /// Statistiken
```

```
Map<String, dynamic> get stats {  
  // TODO: Pending pro Priority, completed, failed  
}  
  
/// Queue leeren  
void clear() {  
  // TODO  
}  
}
```

### 5.2.1.7 Aufgabe 5: Worker (25 min)

Implementiere den Worker der Jobs verarbeitet.

```
// lib/worker/worker.dart  
  
class Worker {  
  final String id;  
  final MemoryJobQueue queue;  
  final int concurrency;  
  
  bool _isRunning = false;  
  int _activeJobs = 0;  
  int _processedJobs = 0;  
  
  final _statusController = StreamController<WorkerStatus>.broadcast();  
  Stream<WorkerStatus> get status => _statusController.stream;  
  
  Worker({  
    required this.queue,  
    this.concurrency = 1,  
    String? id,  
  }) : id = id ?? 'worker-${DateTime.now().millisecondsSinceEpoch}';  
  
  /// Worker starten  
  Future<void> start() async {  
    // TODO: _isRunning setzen  
    // TODO: Loop der Jobs verarbeitet  
    // TODO: Concurrency beachten  
    // TODO: Bei leerem Queue kurz warten  
  }  
  
  /// Einzelnen Job verarbeiten  
  Future<void> _processJob(Job job) async {  
    // TODO: Job-Status auf running  
    // TODO: attempts erhöhen  
    // TODO: startedAt setzen  
    // TODO: execute() aufrufen  
    // TODO: Bei Erfolg: markCompleted  
    // TODO: Bei Fehler: Retry oder markFailed  
  }  
}
```

```

    /// Worker stoppen
    Future<void> stop() async {
      // TODO: _isRunning = false
      // TODO: Auf aktive Jobs warten
    }

    /// Status-Info
    WorkerStatus getStatus() {
      // TODO
    }
  }

class WorkerStatus {
  final String workerId;
  final bool isRunning;
  final int activeJobs;
  final int processedJobs;
  final DateTime timestamp;

  WorkerStatus({
    required this.workerId,
    required this.isRunning,
    required this.activeJobs,
    required this.processedJobs,
  }) : timestamp = DateTime.now();

  Map<String, dynamic> toJson() => {
    'workerId': workerId,
    'isRunning': isRunning,
    'activeJobs': activeJobs,
    'processedJobs': processedJobs,
    'timestamp': timestamp.toIso8601String(),
  };
}

```

### 5.2.1.8 Aufgabe 6: Cron Parser (20 min)

Implementiere einen einfachen Cron-Parser.

```

// lib/scheduler/cron.dart

class CronExpression {
  final String expression;
  final List<int>? minutes;    // 0-59
  final List<int>? hours;     // 0-23
  final List<int>? daysOfWeek; // 0-6 (0 = Sonntag)

  CronExpression._({
    required this.expression,
    this.minutes,

```

```

    this.hours,
    this.daysOfWeek,
  });

  /// Vereinfachter Parser für:
  /// - "*" * "*" (minute hour dayOfWeek)
  /// - "*" = jeder Wert
  /// - "5" = genau dieser Wert
  /// - "1,3,5" = Liste
  /// - "1-5" = Range
  /// - "*/15" = Alle 15
  factory CronExpression.parse(String expression) {
    final parts = expression.split(' ');
    if (parts.length != 3) {
      throw FormatException('Expected 3 parts: minute hour dayOfWeek');
    }

    return CronExpression._(
      expression: expression,
      minutes: _parseField(parts[0], 0, 59),
      hours: _parseField(parts[1], 0, 23),
      daysOfWeek: _parseField(parts[2], 0, 6),
    );
  }

  static List<int>? _parseField(String field, int min, int max) {
    // TODO: "*" -> null (alle)
    // TODO: "5" -> [5]
    // TODO: "1,3,5" -> [1, 3, 5]
    // TODO: "1-5" -> [1, 2, 3, 4, 5]
    // TODO: "*/15" -> [0, 15, 30, 45] (bei min=0, max=59)
  }

  /// Prüft ob DateTime zum Ausdruck passt
  bool matches(DateTime dt) {
    // TODO: Alle Felder prüfen
  }

  /// Nächste Ausführungszeit
  DateTime nextRun([DateTime? from]) {
    // TODO: Von jetzt (oder from) aus suchen
    // TODO: Minutenweise prüfen bis Match
  }
}

```

### 5.2.1.9 Aufgabe 7: Scheduler (20 min)

Implementiere den Task-Scheduler.

```
// lib/scheduler/scheduler.dart
```

```
typedef TaskCallback = Future<void> Function();

class ScheduledTask {
    final String name;
    final CronExpression schedule;
    final TaskCallback callback;
    DateTime? lastRun;
    DateTime? nextRun;
    bool isRunning;
    int runCount;
    int errorCount;

    ScheduledTask({
        required this.name,
        required String cron,
        required this.callback,
    }) : schedule = CronExpression.parse(cron),
        isRunning = false,
        runCount = 0,
        errorCount = 0 {
        nextRun = schedule.nextRun();
    }

    Future<void> execute() async {
        // TODO: Guard gegen doppelte Ausführung
        // TODO: Status aktualisieren
        // TODO: Callback ausführen
        // TODO: Error handling
        // TODO: nextRun berechnen
    }
}

class Scheduler {
    final List<ScheduledTask> _tasks = [];
    Timer? _timer;
    bool _isRunning = false;

    /// Task hinzufügen
    void schedule(String name, String cron, TaskCallback callback) {
        // TODO: ScheduledTask erstellen und speichern
    }

    /// Scheduler starten
    void start() {
        // TODO: Timer alle 60 Sekunden
        // TODO: Bei jedem Tick prüfen welche Tasks fällig sind
    }

    void _tick() {
        // TODO: Aktuelle Zeit
        // TODO: Für jeden Task prüfen ob nextRun erreicht
    }
}
```

```
// TODO: Falls ja und nicht running: execute()
}

/// Scheduler stoppen
void stop() {
  // TODO
}

/// Alle Tasks auflisten
List<Map<String, dynamic>> getTaskInfo() {
  // TODO: Name, lastRun, nextRun, runCount, errorCount
}
}
```

#### 5.2.1.10 Aufgabe 8: Integration & API (20 min)

Erstelle einen Server mit Job-Queue und Scheduler.

```
// bin/server.dart

import 'dart:io';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as io;
import 'package:shelf_router/shelf_router.dart';

void main() async {
  // TODO: Queue und Registry erstellen
  final registry = JobRegistry()
    ..register('send_email', SendEmailJob.fromJson)
    ..register('webhook', WebhookJob.fromJson);

  final queue = MemoryJobQueue();

  // TODO: Worker starten
  final worker = Worker(queue: queue, concurrency: 3);
  worker.start();

  // TODO: Scheduler erstellen
  final scheduler = Scheduler();

  scheduler.schedule('health_log', '* * *', () async {
    print('Health check: ${DateTime.now()}');
  });

  scheduler.schedule('queue_stats', '* / 5 * *', () async {
    print('Queue stats: ${queue.stats}');
  });

  scheduler.start();

  // API Router
```

```

final router = Router();

// POST /jobs - Job einreihen
router.post('/jobs', (Request request) async {
  // TODO: Body parsen
  // TODO: Job über Registry erstellen
  // TODO: In Queue einreihen
  // TODO: Job-ID zurückgeben
});

// GET /jobs/stats - Queue-Statistiken
router.get('/jobs/stats', (Request request) {
  // TODO: queue.stats und worker.getStatus() zurückgeben
});

// GET /scheduler - Scheduled Tasks
router.get('/scheduler', (Request request) {
  // TODO: scheduler.getTaskInfo() zurückgeben
});

// POST /scheduler/:name/run - Task manuell ausführen
router.post('/scheduler/<name>/run', (Request request, String name) async {
  // TODO: Task finden und ausführen
});

final handler = const Pipeline()
  .addMiddleware(logRequests())
  .addHandler(router.call);

final server = await io.serve(handler, 'localhost', 8080);
print('Job server running on http://localhost:${server.port}');

// Graceful shutdown
ProcessSignal.sigint.watch().listen((_) async {
  print('\nShutting down...');
  scheduler.stop();
  await worker.stop();
  await server.close();
  exit(0);
});
}

```

#### 5.2.1.11 Testen

Jobs einreihen

```

# Email Job
curl -X POST http://localhost:8080/jobs \
  -H "Content-Type: application/json" \
  -d '{
    "type": "send_email",

```

```
"to": "test@example.com",
"subject": "Test",
"body": "Hello World"
}'
```

*# Webhook Job mit hoher Priorität*

```
curl -X POST http://localhost:8080/jobs \
-H "Content-Type: application/json" \
-d '{
  "type": "webhook",
  "url": "https://httpbin.org/post",
  "payload": {"event": "test"},
  "priority": "high"
}'
```

Stats abrufen

```
curl http://localhost:8080/jobs/stats
curl http://localhost:8080/scheduler
```

Task manuell ausführen

```
curl -X POST http://localhost:8080/scheduler/health_log/run
```

#### 5.2.1.12 Bonus: Redis Queue (Optional)

```
// lib/queue/redis_queue.dart

class RedisJobQueue {
  final RedisConnection redis;
  final String queueName;
  final JobRegistry registry;

  RedisJobQueue(this.redis, this.queueName, this.registry);

  Future<void> enqueue(Job job) async {
    // TODO: RPush mit JSON
  }

  Future<Job?> dequeue() async {
    // TODO: BLPop mit Timeout
  }

  Future<void> enqueueDelayed(Job job, Duration delay) async {
    // TODO: ZADD in sorted set mit Timestamp
  }

  Future<void> processDelayed() async {
    // TODO: ZRANGEBYSCORE für fällige Jobs
    // TODO: In Hauptqueue verschieben
  }
}
```

```
}
```

### 5.2.1.13 Abgabe-Checkliste

- ☐ Job-Basisklasse mit Retry-Logik
- ☐ Mindestens 2 konkrete Job-Typen
- ☐ JobRegistry für Deserialisierung
- ☐ MemoryJobQueue mit Priority-Support
- ☐ Worker mit Concurrency
- ☐ CronExpression Parser
- ☐ Scheduler mit Tasks
- ☐ REST API für Jobs und Scheduler
- ☐ Graceful Shutdown
- ☐ Bonus: Redis Queue

## 5.2.2 Lösung

### 5.2.2.1 Aufgabe 1: Job-Basisklasse

```
// lib/jobs/job.dart

import 'package:uuid/uuid.dart';

enum JobStatus { pending, running, completed, failed }
enum JobPriority { low, normal, high, critical }

abstract class Job {
  final String id;
  final DateTime createdAt;
  final JobPriority priority;
  JobStatus status;
  int attempts;
  final int maxAttempts;
  DateTime? startedAt;
  DateTime? completedAt;
  String? error;

  Job({
    String? id,
    this.maxAttempts = 3,
    this.priority = JobPriority.normal,
  }) : id = id ?? const Uuid().v4(),
       createdAt = DateTime.now(),
       status = JobStatus.pending,
       attempts = 0;

  /// Job-Typ für Serialisierung
  String get type;

  /// Job ausführen
  Future<void> execute();
```

```

/// Zu JSON konvertieren
Map<String, dynamic> toJson();

/// Sollte bei Fehler wiederholt werden?
bool get shouldRetry => attempts < maxAttempts;

/// Delay vor Retry (exponential backoff)
Duration get retryDelay => Duration(seconds: 1 << attempts);

/// Job-Dauer berechnen
Duration? get duration {
  if (startedAt == null || completedAt == null) return null;
  return completedAt!.difference(startedAt!);
}

/// Basis-JSON mit gemeinsamen Feldern
Map<String, dynamic> baseToJson() => {
  'id': id,
  'type': type,
  'priority': priority.name,
  'status': status.name,
  'attempts': attempts,
  'maxAttempts': maxAttempts,
  'createdAt': createdAt.toIso8601String(),
  if (startedAt != null) 'startedAt': startedAt!.toIso8601String(),
  if (completedAt != null) 'completedAt': completedAt!.toIso8601String(),
  if (error != null) 'error': error,
};
}

```

### 5.2.2.2 Aufgabe 2: Konkrete Jobs

```

// lib/jobs/email_job.dart

import 'dart:math';
import 'job.dart';

class SendEmailJob extends Job {
  final String to;
  final String subject;
  final String body;
  final List<String>? cc;

  SendEmailJob({
    required this.to,
    required this.subject,
    required this.body,
    this.cc,
    super.maxAttempts = 3,
  }) : super();
}

```

```

    super.priority = JobPriority.normal,
  });

  @override
  String get type => 'send_email';

  @override
  Future<void> execute() async {
    print(' Sending email to $to: "$subject"');

    // Simuliere Verarbeitung
    await Future.delayed(Duration(milliseconds: 300 + Random().nextInt(400)));

    // 10% Fehlerrate für Tests
    if (Random().nextDouble() < 0.1) {
      throw Exception('SMTP connection failed');
    }

    print(' Email sent successfully');
  }

  @override
  Map<String, dynamic> toJson() => {
    ...baseToJson(),
    'to': to,
    'subject': subject,
    'body': body,
    if (cc != null) 'cc': cc,
  };

  factory SendEmailJob.fromJson(Map<String, dynamic> json) {
    final job = SendEmailJob(
      to: json['to'] as String,
      subject: json['subject'] as String,
      body: json['body'] as String,
      cc: (json['cc'] as List?)?.cast<String>(),
      maxAttempts: json['maxAttempts'] as int? ?? 3,
      priority: JobPriority.values.byName(json['priority'] as String? ??
↵ 'normal'),
    );
    job.attempts = json['attempts'] as int? ?? 0;
    return job;
  }
}

// lib/jobs/webhook_job.dart

import 'dart:convert';
import 'dart:math';
import 'job.dart';

```

```
class WebhookJob extends Job {
  final String url;
  final Map<String, dynamic> payload;
  final Map<String, String>? headers;

  WebhookJob({
    required this.url,
    required this.payload,
    this.headers,
    super.maxAttempts = 5,
    super.priority = JobPriority.high,
  });

  @override
  String get type => 'webhook';

  @override
  Future<void> execute() async {
    print(' Calling webhook: $url');

    // Simuliere HTTP Call
    await Future.delayed(Duration(milliseconds: 200 + Random().nextInt(300)));

    // 15% Fehlerrate
    if (Random().nextDouble() < 0.15) {
      throw Exception('Webhook returned 503');
    }

    print(' Webhook called successfully');
  }

  @override
  Map<String, dynamic> toJson() => {
    ...baseToJson(),
    'url': url,
    'payload': payload,
    if (headers != null) 'headers': headers,
  };

  factory WebhookJob.fromJson(Map<String, dynamic> json) {
    final job = WebhookJob(
      url: json['url'] as String,
      payload: Map<String, dynamic>.from(json['payload'] as Map),
      headers: (json['headers'] as Map?)?.cast<String, String>(),
      maxAttempts: json['maxAttempts'] as int? ?? 5,
      priority: JobPriority.values.byName(json['priority'] as String? ?? 'high'),
    );
    job.attempts = json['attempts'] as int? ?? 0;
    return job;
  }
}
```

```
// lib/jobs/cleanup_job.dart

import 'dart:io';
import 'job.dart';

class CleanupJob extends Job {
  final String directory;
  final int maxAgeDays;

  CleanupJob({
    required this.directory,
    this.maxAgeDays = 30,
    super.maxAttempts = 1,
    super.priority = JobPriority.low,
  });

  @override
  String get type => 'cleanup';

  @override
  Future<void> execute() async {
    print('  Cleaning up $directory (files older than $maxAgeDays days)');

    final dir = Directory(directory);
    if (!await dir.exists()) {
      print('  Directory does not exist, skipping');
      return;
    }

    final cutoff = DateTime.now().subtract(Duration(days: maxAgeDays));
    int deletedCount = 0;

    await for (final entity in dir.list()) {
      if (entity is File) {
        final stat = await entity.stat();
        if (stat.modified.isBefore(cutoff)) {
          await entity.delete();
          deletedCount++;
        }
      }
    }

    print('  Deleted $deletedCount files');
  }

  @override
  Map<String, dynamic> toJson() => {
    ...baseToJson(),
    'directory': directory,
    'maxAgeDays': maxAgeDays,
  }
}
```

```

    };

    factory CleanupJob.fromJson(Map<String, dynamic> json) {
      return CleanupJob(
        directory: json['directory'] as String,
        maxAgeDays: json['maxAgeDays'] as int? ?? 30,
      );
    }
  }
}

```

### 5.2.2.3 Aufgabe 3: Job Registry

```

// lib/queue/job_registry.dart

import '../jobs/job.dart';

typedef JobFactory = Job Function(Map<String, dynamic> json);

class JobRegistry {
  final Map<String, JobFactory> _factories = {};

  /// Job-Typ registrieren
  void register(String type, JobFactory factory) {
    _factories[type] = factory;
    print('Registered job type: $type');
  }

  /// Job aus JSON erstellen
  Job createJob(Map<String, dynamic> json) {
    final type = json['type'] as String?;
    if (type == null) {
      throw ArgumentError('Job JSON must contain "type" field');
    }

    final factory = _factories[type];
    if (factory == null) {
      throw ArgumentError('Unknown job type: $type');
    }

    return factory(json);
  }

  /// Alle registrierten Typen
  List<String> get registeredTypes => _factories.keys.toList();

  /// Prüfen ob Typ registriert ist
  bool hasType(String type) => _factories.containsKey(type);
}

```

#### 5.2.2.4 Aufgabe 4: In-Memory Queue

```
// lib/queue/memory_queue.dart

import 'dart:async';
import 'dart:collection';
import '../jobs/job.dart';

class MemoryJobQueue {
  // Priority Queues
  final Map<JobPriority, Queue<Job>> _queues = {
    JobPriority.critical: Queue<Job>(),
    JobPriority.high: Queue<Job>(),
    JobPriority.normal: Queue<Job>(),
    JobPriority.low: Queue<Job>(),
  };

  final List<Job> _completed = [];
  final List<Job> _failed = [];
  final List<Timer> _delayedTimers = [];

  final _completedController = StreamController<Job>.broadcast();
  Stream<Job> get onJobCompleted => _completedController.stream;

  /// Job einreihen
  void enqueue(Job job) {
    _queues[job.priority]!.add(job);
    print('Enqueued: ${job.type} (${job.id}) [${job.priority.name}]');
  }

  /// Job mit Verzögerung einreihen
  void enqueueDelayed(Job job, Duration delay) {
    print('Scheduling ${job.type} in ${delay.inSeconds}s');
    final timer = Timer(delay, () => enqueue(job));
    _delayedTimers.add(timer);
  }

  /// Nächsten Job holen (höchste Priority zuerst)
  Job? dequeue() {
    // Von höchster zu niedrigster Priorität
    for (final priority in [
      JobPriority.critical,
      JobPriority.high,
      JobPriority.normal,
      JobPriority.low,
    ]) {
      final queue = _queues[priority]!;
      if (queue.isNotEmpty) {
        return queue.removeFirst();
      }
    }
  }
}
```

```

    return null;
}

/// Job als abgeschlossen markieren
void markCompleted(Job job) {
    job.status = JobStatus.completed;
    job.completedAt = DateTime.now();
    _completed.add(job);
    _completedController.add(job);
}

/// Job als fehlgeschlagen markieren
void markFailed(Job job, String error) {
    job.error = error;

    if (job.shouldRetry) {
        print('Retry ${job.attempts}/${job.maxAttempts} in
↪  ${job.retryDelay.inSeconds}s');
        job.status = JobStatus.pending;
        enqueueDelayed(job, job.retryDelay);
    } else {
        job.status = JobStatus.failed;
        _failed.add(job);
        print('Job failed permanently: ${job.type} (${job.id})');
    }
}

/// Anzahl wartender Jobs
int get pendingCount {
    return _queues.values.fold(0, (sum, q) => sum + q.length);
}

/// Ist leer?
bool get isEmpty => pendingCount == 0;

/// Statistiken
Map<String, dynamic> get stats => {
    'pending': {
        'total': pendingCount,
        'critical': _queues[JobPriority.critical]!.length,
        'high': _queues[JobPriority.high]!.length,
        'normal': _queues[JobPriority.normal]!.length,
        'low': _queues[JobPriority.low]!.length,
    },
    'completed': _completed.length,
    'failed': _failed.length,
};

/// Abgeschlossene Jobs
List<Job> get completedJobs => List.unmodifiable(_completed);

```

```
/// Fehlgeschlagene Jobs
List<Job> get failedJobs => List.unmodifiable(_failed);

/// Queue leeren
void clear() {
  for (final queue in _queues.values) {
    queue.clear();
  }
  _completed.clear();
  _failed.clear();
  for (final timer in _delayedTimers) {
    timer.cancel();
  }
  _delayedTimers.clear();
}

void dispose() {
  clear();
  _completedController.close();
}
}
```

#### 5.2.2.5 Aufgabe 5: Worker

```
// lib/worker/worker.dart

import 'dart:async';
import '../jobs/job.dart';
import '../queue/memory_queue.dart';

class WorkerStatus {
  final String workerId;
  final bool isRunning;
  final int activeJobs;
  final int processedJobs;
  final DateTime timestamp;

  WorkerStatus({
    required this.workerId,
    required this.isRunning,
    required this.activeJobs,
    required this.processedJobs,
  }) : timestamp = DateTime.now();

  Map<String, dynamic> toJson() => {
    'workerId': workerId,
    'isRunning': isRunning,
    'activeJobs': activeJobs,
    'processedJobs': processedJobs,
    'timestamp': timestamp.toIso8601String(),
  }
}
```

```
};
}

class Worker {
  final String id;
  final MemoryJobQueue queue;
  final int concurrency;

  bool _isRunning = false;
  int _activeJobs = 0;
  int _processedJobs = 0;
  final List<Future<void>> _runningTasks = [];

  final _statusController = StreamController<WorkerStatus>.broadcast();
  Stream<WorkerStatus> get status => _statusController.stream;

  Worker({
    required this.queue,
    this.concurrency = 1,
    String? id,
  }) : id = id ?? 'worker-${DateTime.now().millisecondsSinceEpoch}';

  bool get isRunning => _isRunning;

  /// Worker starten
  Future<void> start() async {
    if (_isRunning) return;
    _isRunning = true;

    print('Worker $id started (concurrency: $concurrency)');
    _emitStatus();

    while (_isRunning) {
      // Warte wenn Queue leer
      if (queue.isEmpty) {
        await Future.delayed(Duration(milliseconds: 100));
        continue;
      }

      // Warte wenn max concurrency erreicht
      if (_activeJobs >= concurrency) {
        await Future.delayed(Duration(milliseconds: 50));
        continue;
      }

      // Job holen und verarbeiten
      final job = queue.dequeue();
      if (job != null) {
        _activeJobs++;
        _emitStatus();
      }
    }
  }
}
```

```

        final task = _processJob(job).then((_) {
            _activeJobs--;
            _processedJobs++;
            _emitStatus();
        });

        _runningTasks.add(task);
        task.then((_) => _runningTasks.remove(task));
    }
}

/// Einzelnen Job verarbeiten
Future<void> _processJob(Job job) async {
    print('[${id}] Processing: ${job.type} (${job.id}) - Attempt ${job.attempts} ↵
↵ + 1}');

    job.status = JobStatus.running;
    job.attempts++;
    job.startedAt = DateTime.now();

    try {
        await job.execute();
        queue.markCompleted(job);
        print('[${id}] Completed: ${job.type} (${job.id}) in ↵
↵ ${job.duration?.inMilliseconds}ms');
    } catch (e) {
        print('[${id}] Failed: ${job.type} (${job.id}) - $e');
        queue.markFailed(job, e.toString());
    }
}

/// Worker stoppen
Future<void> stop() async {
    print('[${id}] Stopping...');
    _isRunning = false;

    // Auf aktive Tasks warten
    if (_runningTasks.isNotEmpty) {
        print('[${id}] Waiting for ${_runningTasks.length} active jobs...');
        await Future.wait(_runningTasks);
    }

    print('[${id}] Stopped (processed: $_processedJobs jobs)');
    _statusController.close();
}

void _emitStatus() {
    if (!_statusController.isClosed) {
        _statusController.add(getStatus());
    }
}

```

```
}

WorkerStatus getStatus() => WorkerStatus(
  workerId: id,
  isRunning: _isRunning,
  activeJobs: _activeJobs,
  processedJobs: _processedJobs,
);
}
```

#### 5.2.2.6 Aufgabe 6: Cron Parser

```
// lib/scheduler/cron.dart

class CronExpression {
  final String expression;
  final List<int>? minutes;
  final List<int>? hours;
  final List<int>? daysOfWeek;

  CronExpression._({
    required this.expression,
    this.minutes,
    this.hours,
    this.daysOfWeek,
  });

  factory CronExpression.parse(String expression) {
    final parts = expression.trim().split(RegExp(r'\s+'));
    if (parts.length != 3) {
      throw FormatException(
        'Expected 3 parts (minute hour dayOfWeek), got ${parts.length}',
      );
    }

    return CronExpression._(
      expression: expression,
      minutes: _parseField(parts[0], 0, 59),
      hours: _parseField(parts[1], 0, 23),
      daysOfWeek: _parseField(parts[2], 0, 6),
    );
  }

  static List<int>? _parseField(String field, int min, int max) {
    // Wildcard - alle Werte
    if (field == '*') return null;

    final values = <int>{};

    for (final part in field.split(',')) {
```

```

    if (part.contains('/')) {
        // Step: */5 oder 0-30/5
        final stepParts = part.split('/');
        final step = int.parse(stepParts[1]);

        int start = min;
        int end = max;

        if (stepParts[0] != '*') {
            if (stepParts[0].contains('-')) {
                final range = stepParts[0].split('-');
                start = int.parse(range[0]);
                end = int.parse(range[1]);
            } else {
                start = int.parse(stepParts[0]);
            }
        }

        for (var i = start; i <= end; i += step) {
            values.add(i);
        }
    } else if (part.contains('-')) {
        // Range: 1-5
        final range = part.split('-');
        final start = int.parse(range[0]);
        final end = int.parse(range[1]);

        for (var i = start; i <= end; i++) {
            values.add(i);
        }
    } else {
        // Einzelwert
        values.add(int.parse(part));
    }
}

return values.toList()..sort();
}

/// Prüft ob DateTime zum Ausdruck passt
bool matches(DateTime dt) {
    if (minutes != null && !minutes!.contains(dt.minute)) return false;
    if (hours != null && !hours!.contains(dt.hour)) return false;
    if (daysOfWeek != null && !daysOfWeek!.contains(dt.weekday % 7)) return
↪ false;
    return true;
}

/// Nächste Ausführungszeit
DateTime nextRun([DateTime? from]) {
    var dt = from ?? DateTime.now();

```

```

    // Nächste Minute
    dt = DateTime(dt.year, dt.month, dt.day, dt.hour, dt.minute + 1);

    // Maximal 7 Tage suchen
    final maxDate = dt.add(Duration(days: 7));

    while (dt.isBefore(maxDate)) {
      if (matches(dt)) return dt;
      dt = dt.add(Duration(minutes: 1));
    }

    throw StateError('No matching time found within 7 days');
  }

  @override
  String toString() => 'CronExpression($expression)';
}

```

### 5.2.2.7 Aufgabe 7: Scheduler

```

// lib/scheduler/scheduler.dart

import 'dart:async';
import 'cron.dart';

typedef TaskCallback = Future<void> Function();

class ScheduledTask {
  final String name;
  final CronExpression schedule;
  final TaskCallback callback;
  DateTime? lastRun;
  DateTime? nextRun;
  bool isRunning;
  int runCount;
  int errorCount;
  String? lastError;

  ScheduledTask({
    required this.name,
    required String cron,
    required this.callback,
  }) : schedule = CronExpression.parse(cron),
      isRunning = false,
      runCount = 0,
      errorCount = 0 {
    _updateNextRun();
  }

  void _updateNextRun() {

```

```
    try {
        nextRun = schedule.nextRun();
    } catch (e) {
        nextRun = null;
    }
}

Future<void> execute() async {
    if (isRunning) {
        print('Task "$name" already running, skipping');
        return;
    }

    isRunning = true;
    lastRun = DateTime.now();
    lastError = null;

    try {
        print('Executing task: $name');
        await callback();
        runCount++;
        print('Task completed: $name');
    } catch (e) {
        errorCount++;
        lastError = e.toString();
        print('Task failed: $name - $e');
    } finally {
        isRunning = false;
        _updateNextRun();
    }
}

Map<String, dynamic> toJson() => {
    'name': name,
    'schedule': schedule.expression,
    'isRunning': isRunning,
    'runCount': runCount,
    'errorCount': errorCount,
    'lastRun': lastRun?.toIso8601String(),
    'nextRun': nextRun?.toIso8601String(),
    if (lastError != null) 'lastError': lastError,
};
}

class Scheduler {
    final List<ScheduledTask> _tasks = [];
    Timer? _timer;
    bool _isRunning = false;

    /// Task hinzufügen
    void schedule(String name, String cron, TaskCallback callback) {
```

```
    final task = ScheduledTask(name: name, cron: cron, callback: callback);
    _tasks.add(task);
    print('Scheduled: "$name" (${task.schedule.expression}) - next:
↵  ${task.nextRun}');
}

/// Scheduler starten
void start() {
    if (_isRunning) return;
    _isRunning = true;

    // Alle 30 Sekunden prüfen
    _timer = Timer.periodic(Duration(seconds: 30), (_) => _tick());
    print('Scheduler started with ${_tasks.length} tasks');

    // Initial tick
    _tick();
}

void _tick() {
    final now = DateTime.now();

    for (final task in _tasks) {
        if (task.nextRun != null &&
            now.isAfter(task.nextRun!) &&
            !task.isRunning) {
            // Asynchron ausführen
            task.execute();
        }
    }
}

/// Task manuell ausführen
Future<bool> runTask(String name) async {
    final task = _tasks.where((t) => t.name == name).firstOrNull;
    if (task == null) return false;

    await task.execute();
    return true;
}

/// Scheduler stoppen
void stop() {
    _timer?.cancel();
    _timer = null;
    _isRunning = false;
    print('Scheduler stopped');
}

/// Task finden
ScheduledTask? getTask(String name) {
```

```
    return _tasks.where((t) => t.name == name).firstOrNull;
  }

  /// Alle Tasks auflisten
  List<Map<String, dynamic>> getTaskInfo() {
    return _tasks.map((t) => t.toJson()).toList();
  }

  bool get isRunning => _isRunning;
  int get taskCount => _tasks.length;
}
```

### 5.2.2.8 Aufgabe 8: Server Integration

```
// bin/server.dart

import 'dart:convert';
import 'dart:io';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as io;
import 'package:shelf_router/shelf_router.dart';

import '../lib/jobs/job.dart';
import '../lib/jobs/email_job.dart';
import '../lib/jobs/webhook_job.dart';
import '../lib/jobs/cleanup_job.dart';
import '../lib/queue/job_registry.dart';
import '../lib/queue/memory_queue.dart';
import '../lib/worker/worker.dart';
import '../lib/scheduler/scheduler.dart';

void main() async {
  // Registry
  final registry = JobRegistry()
    ..register('send_email', SendEmailJob.fromJson)
    ..register('webhook', WebhookJob.fromJson)
    ..register('cleanup', CleanupJob.fromJson);

  // Queue
  final queue = MemoryJobQueue();

  // Worker
  final worker = Worker(queue: queue, concurrency: 3);
  worker.start();

  // Scheduler
  final scheduler = Scheduler();

  scheduler.schedule('health_log', '* * *', () async {
    print('[Health] System OK at ${DateTime.now()}');
  });
}
```

```
});

scheduler.schedule('queue_stats', '*/*5 * *', () async {
    print('[Stats] Queue: ${queue.stats}');
});

scheduler.start();

// Router
final router = Router();

// POST /jobs - Job einreihen
router.post('/jobs', (Request request) async {
    try {
        final body = await request.readAsString();
        final json = jsonDecode(body) as Map<String, dynamic>;

        if (!registry.hasType(json['type'] as String? ?? '')) {
            return Response.badRequest(
                body: jsonEncode({
                    'error': 'Unknown job type',
                    'registeredTypes': registry.registeredTypes,
                }),
                headers: {'content-type': 'application/json'},
            );
        }

        final job = registry.createJob(json);
        queue.enqueue(job);

        return Response.ok(
            jsonEncode({
                'success': true,
                'jobId': job.id,
                'type': job.type,
                'priority': job.priority.name,
            }),
            headers: {'content-type': 'application/json'},
        );
    } catch (e) {
        return Response.badRequest(
            body: jsonEncode({'error': e.toString()}),
            headers: {'content-type': 'application/json'},
        );
    }
});

// POST /jobs/batch - Mehrere Jobs einreihen
router.post('/jobs/batch', (Request request) async {
    try {
        final body = await request.readAsString();
```

```
final jobs = (jsonDecode(body) as List).cast<Map<String, dynamic>>();

final results = <Map<String, dynamic>>[];

for (final json in jobs) {
  try {
    final job = registry.createJob(json);
    queue.enqueue(job);
    results.add({'jobId': job.id, 'status': 'enqueued'});
  } catch (e) {
    results.add({'error': e.toString(), 'status': 'failed'});
  }
}

return Response.ok(
  jsonEncode({'jobs': results}),
  headers: {'content-type': 'application/json'},
);
} catch (e) {
  return Response.badRequest(
    body: jsonEncode({'error': e.toString()}),
    headers: {'content-type': 'application/json'},
  );
}
});

// GET /jobs/stats - Statistiken
router.get('/jobs/stats', (Request request) {
  return Response.ok(
    jsonEncode({
      'queue': queue.stats,
      'worker': worker.getStatus().toJson(),
    }),
    headers: {'content-type': 'application/json'},
  );
});

// GET /jobs/completed - Abgeschlossene Jobs
router.get('/jobs/completed', (Request request) {
  final limit = int.tryParse(request.url.queryParameters['limit'] ?? '10')
  ?? 10;
  final jobs = queue.completedJobs.reversed.take(limit).map((j) => j.toJson());

  return Response.ok(
    jsonEncode({'jobs': jobs.toList()}),
    headers: {'content-type': 'application/json'},
  );
});

// GET /jobs/failed - Fehlgeschlagene Jobs
router.get('/jobs/failed', (Request request) {
```

```
final jobs = queue.failedJobs.map((j) => j.toJson());

return Response.ok(
  jsonEncode({'jobs': jobs.toList()}),
  headers: {'content-type': 'application/json'},
);
});

// GET /scheduler - Tasks
router.get('/scheduler', (Request request) {
  return Response.ok(
    jsonEncode({
      'isRunning': scheduler.isRunning,
      'taskCount': scheduler.taskCount,
      'tasks': scheduler.getTaskInfo(),
    }),
    headers: {'content-type': 'application/json'},
  );
});

// POST /scheduler/:name/run - Manuell ausführen
router.post('/scheduler/<name>/run', (Request request, String name) async {
  final success = await scheduler.runTask(name);

  if (!success) {
    return Response.notFound(
      jsonEncode({'error': 'Task not found: $name'}),
      headers: {'content-type': 'application/json'},
    );
  }

  return Response.ok(
    jsonEncode({'success': true, 'task': name}),
    headers: {'content-type': 'application/json'},
  );
});

// GET /types - Registrierte Job-Typen
router.get('/types', (Request request) {
  return Response.ok(
    jsonEncode({'types': registry.registeredTypes}),
    headers: {'content-type': 'application/json'},
  );
});

// Handler mit Middleware
final handler = const Pipeline()
  .addMiddleware(logRequests())
  .addMiddleware(_corsMiddleware())
  .addHandler(router.call);
```

```

// Server starten
final port = int.parse(Platform.environment['PORT'] ?? '8080');
final server = await io.serve(handler, InternetAddress.anyIPv4, port);

print('');
print('
    Job Server Running on port $port
');
print('');
print('Endpoints:');
print('  POST /jobs          - Enqueue a job');
print('  POST /jobs/batch     - Enqueue multiple jobs');
print('  GET  /jobs/stats     - Queue and worker stats');
print('  GET  /jobs/completed - List completed jobs');
print('  GET  /jobs/failed    - List failed jobs');
print('  GET  /scheduler      - List scheduled tasks');
print('  POST /scheduler/:name/run - Run task manually');
print('  GET  /types          - List registered job types');
print('');

// Graceful shutdown
ProcessSignal.sigint.watch().listen((_) async {
  print('\nShutting down...');
  scheduler.stop();
  await worker.stop();
  queue.dispose();
  await server.close();
  exit(0);
});
}

Middleware _corsMiddleware() {
  const headers = {
    'Access-Control-Allow-Origin': '*',
    'Access-Control-Allow-Methods': 'GET, POST, OPTIONS',
    'Access-Control-Allow-Headers': 'Content-Type',
  };

  return (Handler inner) {
    return (Request request) async {
      if (request.method == 'OPTIONS') {
        return Response.ok('', headers: headers);
      }
      final response = await inner(request);
      return response.change(headers: headers);
    };
  };
}

```

### 5.2.2.9 Projektstruktur

```
job_server/  
+-- bin/  
|   +-- server.dart  
+-- lib/  
|   +-- jobs/  
|   |   +-- job.dart  
|   |   +-- email_job.dart  
|   |   +-- webhook_job.dart  
|   |   +-- cleanup_job.dart  
|   +-- queue/  
|   |   +-- job_registry.dart  
|   |   +-- memory_queue.dart  
|   +-- worker/  
|   |   +-- worker.dart  
|   +-- scheduler/  
|       +-- cron.dart  
|       +-- scheduler.dart  
+-- pubspec.yaml  
+-- README.md
```

#### 5.2.2.10 Test-Befehle

```
# Server starten  
dart run bin/server.dart  
  
# Email Job einreihen  
curl -X POST http://localhost:8080/jobs \  
  -H "Content-Type: application/json" \  
  -d '{"type": "send_email", "to": "test@example.com", "subject": "Hello",  
    ↪   "body": "World"}'  
  
# Mehrere Jobs (Batch)  
curl -X POST http://localhost:8080/jobs/batch \  
  -H "Content-Type: application/json" \  
  -d '[  
    {"type": "send_email", "to": "a@test.com", "subject": "Test 1", "body":  
    ↪   "Body 1"},  
    {"type": "send_email", "to": "b@test.com", "subject": "Test 2", "body":  
    ↪   "Body 2"},  
    {"type": "webhook", "url": "https://httpbin.org/post", "payload":  
    ↪   {"test": true}}  
  ]'  
  
# Stats abrufen  
curl http://localhost:8080/jobs/stats | jq  
  
# Scheduler-Tasks  
curl http://localhost:8080/scheduler | jq  
  
# Task manuell ausführen
```

```
curl -X POST http://localhost:8080/scheduler/health_log/run

# Job-Typen anzeigen
curl http://localhost:8080/types
```

### 5.2.3 Ressourcen

#### 5.2.3.1 Offizielle Dokumentation

- Dart Isolates
- Redis Pub/Sub
- Cron Expression Format

#### 5.2.3.2 Cheat Sheet: Job Pattern

```
// Basis Job
abstract class Job {
  String get type;
  Future<void> execute();
  Map<String, dynamic> toJson();

  // Retry-Logik
  int attempts = 0;
  int maxAttempts = 3;
  bool get shouldRetry => attempts < maxAttempts;
  Duration get retryDelay => Duration(seconds: 1 << attempts);
}

// Konkreter Job
class EmailJob extends Job {
  final String to, subject, body;

  EmailJob({required this.to, required this.subject, required this.body});

  @override
  String get type => 'email';

  @override
  Future<void> execute() async {
    await sendEmail(to, subject, body);
  }

  @override
  Map<String, dynamic> toJson() => {
    'type': type, 'to': to, 'subject': subject, 'body': body
  };

  factory EmailJob.fromJson(Map<String, dynamic> json) => EmailJob(
    to: json['to'], subject: json['subject'], body: json['body']
  );
}
```

### 5.2.3.3 Cheat Sheet: Queue Operations

```
class JobQueue {
    final Queue<Job> _queue = Queue();

    // Einreihen
    void enqueue(Job job) => _queue.add(job);

    // Mit Delay
    void enqueueDelayed(Job job, Duration delay) {
        Timer(delay, () => enqueue(job));
    }

    // Abrufen
    Job? dequeue() => _queue.isNotEmpty ? _queue.removeFirst() : null;

    // Priorisiert
    void enqueuePriority(Job job) => _queue.addFirst(job);

    // Batch
    void enqueueBatch(List<Job> jobs) => _queue.addAll(jobs);
}
```

### 5.2.3.4 Cheat Sheet: Worker

```
class Worker {
    final JobQueue queue;
    final int concurrency;
    bool _running = false;
    int _active = 0;

    Worker(this.queue, {this.concurrency = 1});

    Future<void> start() async {
        _running = true;
        while (_running) {
            if (_active >= concurrency || queue.isEmpty) {
                await Future.delayed(Duration(milliseconds: 100));
                continue;
            }

            final job = queue.dequeue();
            if (job != null) {
                _active++;
                _processJob(job).then((_) => _active--);
            }
        }
    }

    Future<void> _processJob(Job job) async {
        job.attempts++;
    }
}
```

```

    try {
        await job.execute();
    } catch (e) {
        if (job.shouldRetry) {
            queue.enqueueDelayed(job, job.retryDelay);
        }
    }
}

void stop() => _running = false;
}

```

### 5.2.3.5 Cheat Sheet: Cron Expressions

```

+----- Minute (0-59)
| +----- Stunde (0-23)
| | +----- Tag des Monats (1-31)
| | | +----- Monat (1-12)
| | | | +----- Wochentag (0-6, 0=Sonntag)
| | | | |
* * * * *

```

Expression	Bedeutung
* * * * *	Jede Minute
0 * * * *	Jede Stunde
0 0 * * *	Täglich Mitternacht
0 9 * * 1-5	Werktags 9 Uhr
*/15 * * * *	Alle 15 Minuten
0 0 1 * *	Monatlich am 1.
0 2 * * 0	Sonntags 2 Uhr

### 5.2.3.6 Cheat Sheet: Scheduler

```

class Scheduler {
    final List<(String, CronExpression, Function)> _tasks = [];
    Timer? _timer;

    void schedule(String name, String cron, Function callback) {
        _tasks.add((name, CronExpression.parse(cron), callback));
    }

    void start() {
        _timer = Timer.periodic(Duration(minutes: 1), (_) {
            final now = DateTime.now();
            for (final (name, cron, callback) in _tasks) {
                if (cron.matches(now)) {
                    callback();
                }
            }
        });
    }
}

```

```
}

void stop() => _timer?.cancel();
}
```

### 5.2.3.7 Cheat Sheet: Redis Queue

```
class RedisQueue {
    final RedisConnection redis;
    final String name;

    // Einreihen (FIFO)
    Future<void> enqueue(Job job) async {
        await redis.execute(['RPUSH', name, jsonEncode(job.toJson())]);
    }

    // Abrufen (blockierend)
    Future<String?> dequeue({int timeout = 5}) async {
        final result = await redis.execute(['BLPOP', name, timeout.toString()]);
        return result?[1] as String?;
    }

    // Verzögert (Sorted Set)
    Future<void> enqueueAt(Job job, DateTime when) async {
        await redis.execute([
            'ZADD', '$name:delayed',
            when.millisecondsSinceEpoch.toString(),
            jsonEncode(job.toJson())
        ]);
    }

    // Fällige Jobs verschieben
    Future<void> processDelayed() async {
        final now = DateTime.now().millisecondsSinceEpoch;
        final jobs = await redis.execute([
            'ZRANGEBYSCORE', '$name:delayed', '0', now.toString()
        ]);
        for (final job in jobs ?? []) {
            await redis.execute(['RPUSH', name, job]);
            await redis.execute(['ZREM', '$name:delayed', job]);
        }
    }
}
```

### 5.2.3.8 Cheat Sheet: Priority Queue

```
class PriorityQueue {
    final Map<int, Queue<Job>> _queues = {};
```

```

void enqueue(Job job, {int priority = 0}) {
    _queues.putIfAbsent(priority, () => Queue()).add(job);
}

Job? dequeue() {
    // Höchste Priorität zuerst
    final priorities = _queues.keys.toList()..sort((a, b) => b.compareTo(a));
    for (final p in priorities) {
        final queue = _queues[p]!;
        if (queue.isNotEmpty) {
            return queue.removeFirst();
        }
    }
    return null;
}
}

```

### 5.2.3.9 Cheat Sheet: Dead Letter Queue

```

class DeadLetterHandler {
    final List<(Job, String, DateTime)> _deadLetters = [];

    void add(Job job, String error) {
        _deadLetters.add((job, error, DateTime.now()));
    }

    List<Map<String, dynamic>> getAll() {
        return _deadLetters.map((entry) => {
            'job': entry.$1.toJson(),
            'error': entry.$2,
            'failedAt': entry.$3.toIso8601String(),
        }).toList();
    }

    // Retry einzelner Job
    Future<void> retry(String jobId, JobQueue queue) async {
        final index = _deadLetters.indexWhere((e) => e.$1.id == jobId);
        if (index >= 0) {
            final job = _deadLetters.removeAt(index).$1;
            job.attempts = 0;
            queue.enqueue(job);
        }
    }
}

```

### 5.2.3.10 Best Practices

DO

1. **Idempotente Jobs** - Mehrfache Ausführung = gleiches Ergebnis
2. **Exponential Backoff** - Bei Retries warten

3. **Timeouts** - Jobs nicht ewig laufen lassen
4. **Dead Letter Queue** - Fehlgeschlagene Jobs aufbewahren
5. **Logging** - Start, Ende, Fehler loggen
6. **Graceful Shutdown** - Aktive Jobs beenden lassen
7. **Health Checks** - Queue-Größe überwachen

#### DON'T

1. **Synchrone Queue-Operationen** im Request
2. **Zu viele Retries** - Max 3-5
3. **Große Payloads** - Job-Daten klein halten
4. **Zustand in Job** - Immer aus DB laden
5. **Silent Failures** - Fehler immer loggen

#### 5.2.3.11 Monitoring

```
class QueueMetrics {
  int enqueued = 0;
  int processed = 0;
  int failed = 0;
  int pending = 0;

  Map<String, dynamic> toJson() => {
    'enqueued': enqueued,
    'processed': processed,
    'failed': failed,
    'pending': pending,
    'successRate': processed > 0
      ? ((processed - failed) / processed * 100).toStringAsFixed(1)
      : '0.0',
  };
}
```

#### 5.2.3.12 Isolates für CPU-intensive Jobs

```
import 'dart:isolate';

class IsolateWorker {
  Future<R> run<R>(Future<R> Function() work) async {
    return await Isolate.run(work);
  }
}

// Verwendung
final result = await IsolateWorker().run(() async {
  // CPU-intensive Arbeit
  return heavyComputation();
});
```

#### 5.2.3.13 Tools

- **Redis** - Persistente Queue

- **BullMQ** (Node.js Equivalent) - Referenz für Features
- **Crontab Guru** - Cron Expression Builder
- **Grafana** - Queue Monitoring

## 5.3 Einheit 9.3: Logging & Monitoring

### 5.3.0.1 Lernziele

- Strukturiertes Logging implementieren
- Log-Level richtig einsetzen
- Health Checks erstellen
- Metriken sammeln und exponieren
- Alerting-Grundlagen verstehen

### 5.3.0.2 Warum Logging & Monitoring?

Die Realität in Produktion

"Es funktioniert auf meinem Rechner" -> Produktion

Was passiert wirklich?

- Wie viele Requests/Sekunde?
- Welche Fehler treten auf?
- Wie lange dauern Requests?
- Ist die Datenbank erreichbar?
- Läuft der Speicher voll?

Die drei Säulen der Observability

Säule	Zweck	Beispiel
<b>Logs</b>	Was ist passiert?	"User 123 hat sich eingeloggt"
<b>Metriken</b>	Wie performt das System?	"95% der Requests < 200ms"
<b>Traces</b>	Wie fließen Requests?	Request -> Auth -> DB -> Response

### 5.3.0.3 Strukturiertes Logging

Das Problem mit unstrukturierten Logs

```
// ☐ Schwer zu parsen und filtern
print('User logged in: john@example.com at 2024-01-15 10:30:00');
print('Error: Connection failed');
print('Request to /api/users took 150ms');

// ☐ Strukturiert (JSON)
{
  "timestamp": "2024-01-15T10:30:00Z",
  "level": "info",
  "message": "User logged in",
  "user_email": "john@example.com",
  "request_id": "abc-123"
}
```

Logger-Klasse

```
// lib/logging/logger.dart

import 'dart:convert';
import 'dart:io';

enum LogLevel {
  debug(0, 'DEBUG'),
  info(1, 'INFO'),
  warning(2, 'WARN'),
  error(3, 'ERROR'),
  fatal(4, 'FATAL');

  final int value;
  final String label;
  const LogLevel(this.value, this.label);

  bool operator >=(LogLevel other) => value >= other.value;
}

class Logger {
  final String name;
  final LogLevel minLevel;
  final bool jsonOutput;
  final IOSink _sink;

  // Globaler Request Context
  static String? currentRequestId;
  static String? currentUserId;

  Logger({
    required this.name,
    this.minLevel = LogLevel.info,
    this.jsonOutput = true,
    IOSink? sink,
  }) : _sink = sink ?? stdout;

  void debug(String message, [Map<String, dynamic>? data]) {
    _log(LogLevel.debug, message, data);
  }

  void info(String message, [Map<String, dynamic>? data]) {
    _log(LogLevel.info, message, data);
  }

  void warning(String message, [Map<String, dynamic>? data]) {
    _log(LogLevel.warning, message, data);
  }

  void error(String message, [Object? error, StackTrace? stackTrace]) {
    _log(LogLevel.error, message, {
      if (error != null) 'error': error.toString(),
    });
  }
}
```

```
        if (stackTrace != null) 'stackTrace': stackTrace.toString(),
    });
}

void fatal(String message, [Object? error, StackTrace? stackTrace]) {
    _log(LogLevel.fatal, message, {
        if (error != null) 'error': error.toString(),
        if (stackTrace != null) 'stackTrace': stackTrace.toString(),
    });
}

void _log(LogLevel level, String message, Map<String, dynamic>? data) {
    if (level.value < minLevel.value) return;

    final entry = LogEntry(
        timestamp: DateTime.now(),
        level: level,
        logger: name,
        message: message,
        data: data,
        requestId: currentRequestId,
        userId: currentUserId,
    );

    if (jsonOutput) {
        _sink.writeln(jsonEncode(entry.toJson()));
    } else {
        _sink.writeln(entry.toText());
    }
}

class LogEntry {
    final DateTime timestamp;
    final LogLevel level;
    final String logger;
    final String message;
    final Map<String, dynamic>? data;
    final String? requestId;
    final String? userId;

    LogEntry({
        required this.timestamp,
        required this.level,
        required this.logger,
        required this.message,
        this.data,
        this.requestId,
        this.userId,
    });
}
```

```

Map<String, dynamic> toJson() => {
  'timestamp': timestamp.toUtc().toIso8601String(),
  'level': level.label,
  'logger': logger,
  'message': message,
  if (requestId != null) 'request_id': requestId,
  if (userId != null) 'user_id': userId,
  if (data != null) ...data!,
};

String toText() {
  final time = timestamp.toIso8601String().substring(11, 23);
  final ctx = [
    if (requestId != null) 'req=$requestId',
    if (userId != null) 'user=$userId',
  ].join(' ');

  return '[$time] ${level.label.padRight(5)} [$logger] $message
↪  ${ctx.isNotEmpty ? '($ctx)' : ''}';
}

```

#### Request-Logging Middleware

```

// lib/logging/request_logger.dart

import 'package:shelf/shelf.dart';
import 'package:uuid/uuid.dart';

Middleware requestLoggingMiddleware(Logger logger) {
  return (Handler innerHandler) {
    return (Request request) async {
      final requestId = const Uuid().v4().substring(0, 8);
      final stopwatch = Stopwatch()..start();

      // Request Context setzen
      Logger.currentRequestId = requestId;

      logger.info('Request started', {
        'method': request.method,
        'path': request.url.path,
        'query': request.url.queryParameters,
      });

      try {
        final response = await innerHandler(request);
        stopwatch.stop();

        logger.info('Request completed', {
          'method': request.method,
          'path': request.url.path,

```

```
        'status': response.statusCode,
        'duration_ms': stopwatch.elapsedMilliseconds,
    });

    // Request-ID im Response Header
    return response.change(headers: {
        'X-Request-ID': requestId,
    });
} catch (e, stack) {
    stopwatch.stop();

    logger.error('Request failed', e, stack);

    rethrow;
} finally {
    Logger.currentRequestId = null;
}
};
};
}
```

#### 5.3.0.4 Log-Level Best Practices

Wann welches Level?

```
// DEBUG: Detaillierte Infos für Entwicklung
logger.debug('SQL Query executed', {
    'query': 'SELECT * FROM users WHERE id = ?',
    'params': [123],
    'duration_ms': 5,
});

// INFO: Normale Geschäftsereignisse
logger.info('User registered', {
    'user_id': 456,
    'email': 'new@user.com',
});

// WARNING: Unerwartetes, aber behandeltes Verhalten
logger.warning('Rate limit approaching', {
    'user_id': 789,
    'requests': 95,
    'limit': 100,
});

// ERROR: Fehler die behandelt wurden
logger.error('Payment failed', PaymentException('Card declined'));

// FATAL: Kritische Fehler, System instabil
logger.fatal('Database connection lost', dbError);
```

Level nach Umgebung

```
class AppConfig {
  final LogLevel logLevel;

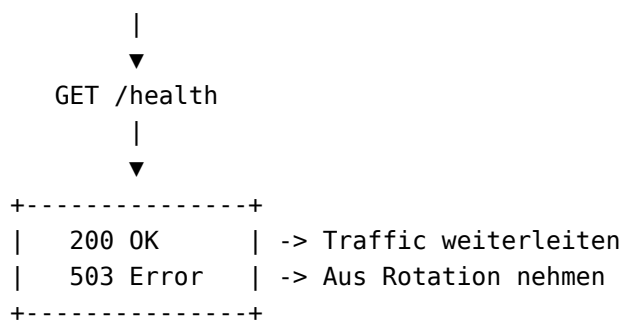
  factory AppConfig.fromEnvironment() {
    final env = Platform.environment['ENV'] ?? 'development';

    return AppConfig(
      logLevel: switch (env) {
        'production' => LogLevel.info,
        'staging' => LogLevel.debug,
        _ => LogLevel.debug,
      },
    );
  }
}
```

### 5.3.0.5 Health Checks

Warum Health Checks?

Load Balancer / Kubernetes / Docker



Health Check Service

```
// lib/health/health_check.dart

enum HealthStatus { healthy, degraded, unhealthy }

class HealthCheckResult {
  final String name;
  final HealthStatus status;
  final Duration duration;
  final String? message;
  final Map<String, dynamic>? details;

  HealthCheckResult({
    required this.name,
    required this.status,
    required this.duration,
    this.message,
    this.details,
  });
}
```

```
Map<String, dynamic> toJson() => {
    'name': name,
    'status': status.name,
    'duration_ms': duration.inMilliseconds,
    if (message != null) 'message': message,
    if (details != null) 'details': details,
};
}

abstract class HealthCheck {
    String get name;
    Future<HealthCheckResult> check();
}

// Datenbank Health Check
class DatabaseHealthCheck implements HealthCheck {
    final Connection db;

    DatabaseHealthCheck(this.db);

    @override
    String get name => 'database';

    @override
    Future<HealthCheckResult> check() async {
        final stopwatch = Stopwatch()..start();

        try {
            await db.execute('SELECT 1');
            stopwatch.stop();

            return HealthCheckResult(
                name: name,
                status: HealthStatus.healthy,
                duration: stopwatch.elapsed,
                message: 'Connection OK',
            );
        } catch (e) {
            stopwatch.stop();
            return HealthCheckResult(
                name: name,
                status: HealthStatus.unhealthy,
                duration: stopwatch.elapsed,
                message: e.toString(),
            );
        }
    }
}

// Redis Health Check
```

```
class RedisHealthCheck implements HealthCheck {
    final RedisConnection redis;

    RedisHealthCheck(this.redis);

    @override
    String get name => 'redis';

    @override
    Future<HealthCheckResult> check() async {
        final stopwatch = Stopwatch()..start();

        try {
            final result = await redis.execute(['PING']);
            stopwatch.stop();

            return HealthCheckResult(
                name: name,
                status: result == 'PONG' ? HealthStatus.healthy : HealthStatus.degraded,
                duration: stopwatch.elapsed,
            );
        } catch (e) {
            stopwatch.stop();
            return HealthCheckResult(
                name: name,
                status: HealthStatus.unhealthy,
                duration: stopwatch.elapsed,
                message: e.toString(),
            );
        }
    }
}

// Disk Space Health Check
class DiskSpaceHealthCheck implements HealthCheck {
    final int warningThresholdPercent;
    final int criticalThresholdPercent;

    DiskSpaceHealthCheck({
        this.warningThresholdPercent = 80,
        this.criticalThresholdPercent = 95,
    });

    @override
    String get name => 'disk_space';

    @override
    Future<HealthCheckResult> check() async {

        // Simuliert - in echtem Code: df -h parsen oder dart:ffi
```

```

    final usedPercent = 60; // Beispiel

    stopwatch.stop();

    final status = usedPercent >= criticalThresholdPercent
        ? HealthStatus.unhealthy
        : usedPercent >= warningThresholdPercent
        ? HealthStatus.degraded
        : HealthStatus.healthy;

    return HealthCheckResult(
        name: name,
        status: status,
        duration: stopwatch.elapsed,
        details: {'used_percent': usedPercent},
    );
}
}

```

#### Health Check Aggregator

```

// lib/health/health_service.dart

class HealthService {
    final List<HealthCheck> _checks = [];
    final Logger _logger;

    HealthService(this._logger);

    void register(HealthCheck check) {
        _checks.add(check);
        _logger.info('Health check registered', {'name': check.name});
    }

    Future<OverallHealth> checkAll() async {
        final results = await Future.wait(
            _checks.map((check) => _runCheck(check)),
        );

        final overallStatus = results.any((r) => r.status == HealthStatus.unhealthy)
            ? HealthStatus.unhealthy
            : results.any((r) => r.status == HealthStatus.degraded)
            ? HealthStatus.degraded
            : HealthStatus.healthy;

        return OverallHealth(
            status: overallStatus,
            checks: results,
            timestamp: DateTime.now(),
        );
    }
}

```

```

Future<HealthCheckResult> _runCheck(HealthCheck check) async {
  try {
    return await check.check().timeout(Duration(seconds: 5));
  } catch (e) {
    return HealthCheckResult(
      name: check.name,
      status: HealthStatus.unhealthy,
      duration: Duration(seconds: 5),
      message: 'Health check timed out: $e',
    );
  }
}

class OverallHealth {
  final HealthStatus status;
  final List<HealthCheckResult> checks;
  final DateTime timestamp;

  OverallHealth({
    required this.status,
    required this.checks,
    required this.timestamp,
  });

  Map<String, dynamic> toJson() => {
    'status': status.name,
    'timestamp': timestamp.toIso8601String(),
    'checks': checks.map((c) => c.toJson()).toList(),
  };

  int get httpStatusCode => switch (status) {
    HealthStatus.healthy => 200,
    HealthStatus.degraded => 200,
    HealthStatus.unhealthy => 503,
  };
}

```

### Health Endpoints

```

// Health Endpoints
router.get('/health', (Request request) async {
  final health = await healthService.checkAll();
  return Response(
    health.httpStatusCode,
    body: jsonEncode(health.toJson()),
    headers: {'content-type': 'application/json'},
  );
});

```

```
// Liveness: Prozess läuft?
router.get('/health/live', (Request request) {
  return Response.ok(jsonEncode({'status': 'alive'}));
});

// Readiness: Bereit für Traffic?
router.get('/health/ready', (Request request) async {
  final health = await healthService.checkAll();
  if (health.status == HealthStatus.unhealthy) {
    return Response(503, body: jsonEncode({'status': 'not ready'}));
  }
  return Response.ok(jsonEncode({'status': 'ready'}));
});
```

### 5.3.0.6 Metriken

Metrik-Typen

```
// lib/metrics/metrics.dart

/// Counter: Nur aufwärts (Requests, Errors)
class Counter {
  final String name;
  final Map<String, String> labels;
  int _value = 0;

  Counter(this.name, [this.labels = const {}]);

  void increment([int amount = 1]) => _value += amount;
  int get value => _value;
}

/// Gauge: Auf/Ab (Connections, Queue Size)
class Gauge {
  final String name;
  final Map<String, String> labels;
  double _value = 0;

  Gauge(this.name, [this.labels = const {}]);

  void set(double value) => _value = value;
  void increment([double amount = 1]) => _value += amount;
  void decrement([double amount = 1]) => _value -= amount;
  double get value => _value;
}

/// Histogram: Verteilung (Response Times)
class Histogram {
  final String name;
  final List<double> buckets;
  final Map<double, int> _bucketCounts = {};
```

```

double _sum = 0;
int _count = 0;

Histogram(this.name, {this.buckets = const [0.01, 0.05, 0.1, 0.5, 1, 5]}) {
  for (final b in buckets) {
    _bucketCounts[b] = 0;
  }
}

void observe(double value) {
  _count++;
  _sum += value;

  for (final bucket in buckets) {
    if (value <= bucket) {
      _bucketCounts[bucket] = (_bucketCounts[bucket] ?? 0) + 1;
    }
  }
}

double get mean => _count > 0 ? _sum / _count : 0;
int get count => _count;
double get sum => _sum;
Map<double, int> get bucketCounts => Map.unmodifiable(_bucketCounts);
}

```

#### Metrics Registry

```

// lib/metrics/registry.dart

class MetricsRegistry {
  final Map<String, Counter> _counters = {};
  final Map<String, Gauge> _gauges = {};
  final Map<String, Histogram> _histograms = {};

  Counter counter(String name, [Map<String, String> labels = const {}]) {
    final key = _makeKey(name, labels);
    return _counters.putIfAbsent(key, () => Counter(name, labels));
  }

  Gauge gauge(String name, [Map<String, String> labels = const {}]) {
    final key = _makeKey(name, labels);
    return _gauges.putIfAbsent(key, () => Gauge(name, labels));
  }

  Histogram histogram(String name, {List<double>? buckets}) {
    return _histograms.putIfAbsent(
      name,
      () => Histogram(name, buckets: buckets ?? [0.01, 0.05, 0.1, 0.5, 1, 5]),
    );
  }
}

```

```

String _makeKey(String name, Map<String, String> labels) {
    if (labels.isEmpty()) return name;
    final labelStr = labels.entries.map((e) =>
↳   '${e.key}="${e.value}"').join(', ');
    return '$name${labelStr}';
}

/// Prometheus-Format exportieren
String exportPrometheus() {
    final buffer = StringBuffer();

    for (final counter in _counters.values) {
        buffer.writeln('# TYPE ${counter.name} counter');
        buffer.writeln('${counter.name} ${counter.value}');
    }

    for (final gauge in _gauges.values) {
        buffer.writeln('# TYPE ${gauge.name} gauge');
        buffer.writeln('${gauge.name} ${gauge.value}');
    }

    for (final hist in _histograms.values) {
        buffer.writeln('# TYPE ${hist.name} histogram');
        for (final entry in hist.bucketCounts.entries) {
            buffer.writeln('${hist.name}_bucket{le="${entry.key}"} ${entry.value}');
        }
        buffer.writeln('${hist.name}_sum ${hist.sum}');
        buffer.writeln('${hist.name}_count ${hist.count}');
    }

    return buffer.toString();
}

/// JSON-Format exportieren
Map<String, dynamic> exportJson() => {
    'counters': {
        for (final c in _counters.values) c.name: c.value,
    },
    'gauges': {
        for (final g in _gauges.values) g.name: g.value,
    },
    'histograms': {
        for (final h in _histograms.values)
            h.name: {
                'count': h.count,
                'sum': h.sum,
                'mean': h.mean,
            },
    },
};

```

```
}
```

#### Metrics Middleware

```
// lib/metrics/middleware.dart

Middleware metricsMiddleware(MetricsRegistry metrics) {
  final requestCounter = metrics.counter('http_requests_total');
  final requestDuration = metrics.histogram('http_request_duration_seconds');
  final activeRequests = metrics.gauge('http_requests_active');

  return (Handler innerHandler) {
    return (Request request) async {
      activeRequests.increment();
      requestCounter.increment();

      final stopwatch = Stopwatch()..start();

      try {
        final response = await innerHandler(request);
        stopwatch.stop();

        requestDuration.observe(stopwatch.elapsedMilliseconds / 1000);

        // Zähler pro Status-Code
        metrics.counter('http_responses_total', {
          'status': (response.statusCode ~/ 100 * 100).toString(),
        }).increment();

        return response;
      } finally {
        activeRequests.decrement();
      }
    };
  };
}
```

#### Metrics Endpoint

```
// Prometheus-Format
router.get('/metrics', (Request request) {
  return Response.ok(
    metrics.exportPrometheus(),
    headers: {'content-type': 'text/plain'},
  );
});

// JSON-Format
router.get('/metrics/json', (Request request) {
  return Response.ok(
    jsonEncode(metrics.exportJson()),
    headers: {'content-type': 'application/json'},
  );
});
```

```
);  
});
```

### 5.3.0.7 Alerting

Alert-Regeln definieren

```
// lib/alerting/alert.dart  
  
enum AlertSeverity { info, warning, critical }  
  
class AlertRule {  
  final String name;  
  final AlertSeverity severity;  
  final Duration checkInterval;  
  final Future<bool> Function() condition;  
  final String message;  
  
  AlertRule({  
    required this.name,  
    required this.severity,  
    required this.condition,  
    required this.message,  
    this.checkInterval = const Duration(minutes: 1),  
  });  
}  
  
class AlertManager {  
  final List<AlertRule> _rules = [];  
  final List<AlertNotifier> _notifiers = [];  
  final Logger _logger;  
  final Map<String, DateTime> _lastFired = {};  
  
  AlertManager(this._logger);  
  
  void addRule(AlertRule rule) => _rules.add(rule);  
  void addNotifier(AlertNotifier notifier) => _notifiers.add(notifier);  
  
  Future<void> check() async {  
    for (final rule in _rules) {  
      try {  
        final triggered = await rule.condition();  
  
        if (triggered) {  
          await _fireAlert(rule);  
        } else {  
          _lastFired.remove(rule.name);  
        }  
      } catch (e) {  
        _logger.error('Alert check failed: ${rule.name}', e);  
      }  
    }  
  }  
}
```

```

    }
}

Future<void> _fireAlert(AlertRule rule) async {
    // Cooldown: Nicht öfter als alle 5 Minuten feuern
    final lastFired = _lastFired[rule.name];
    if (lastFired != null &&
        DateTime.now().difference(lastFired) < Duration(minutes: 5)) {
        return;
    }

    _lastFired[rule.name] = DateTime.now();

    final alert = Alert(
        name: rule.name,
        severity: rule.severity,
        message: rule.message,
        timestamp: DateTime.now(),
    );

    _logger.warning('Alert fired: ${rule.name}', {
        'severity': rule.severity.name,
        'message': rule.message,
    });

    for (final notifier in _notifiers) {
        try {
            await notifier.notify(alert);
        } catch (e) {
            _logger.error('Failed to send alert notification', e);
        }
    }
}

class Alert {
    final String name;
    final AlertSeverity severity;
    final String message;
    final DateTime timestamp;

    Alert({
        required this.name,
        required this.severity,
        required this.message,
        required this.timestamp,
    });
}

abstract class AlertNotifier {
    Future<void> notify(Alert alert);
}

```

```

}

// Slack Notifier
class SlackNotifier implements AlertNotifier {
    final String webhookUrl;

    SlackNotifier(this.webhookUrl);

    @override
    Future<void> notify(Alert alert) async {
        final emoji = switch (alert.severity) {
            AlertSeverity.info => ':information_source:',
            AlertSeverity.warning => ':warning:',
            AlertSeverity.critical => ':rotating_light:',
        };

        // HTTP POST an Slack Webhook
        // await http.post(webhookUrl, body: {...});
    }
}

// Email Notifier
class EmailNotifier implements AlertNotifier {
    final String smtpHost;
    final List<String> recipients;

    EmailNotifier(this.smtpHost, this.recipients);

    @override
    Future<void> notify(Alert alert) async {
        // Email senden
    }
}

```

Alert-Regeln konfigurieren

```

void setupAlerts(AlertManager alerts, MetricsRegistry metrics, HealthService ↵
↵ health) {
    // Hohe Fehlerrate
    alerts.addRule(AlertRule(
        name: 'high_error_rate',
        severity: AlertSeverity.critical,
        message: 'Error rate exceeded 5%',
        condition: () async {
            final errors = metrics.counter('http_responses_total', {'status': ↵
↵ '500'}).value;
            final total = metrics.counter('http_requests_total').value;
            return total > 100 && errors / total > 0.05;
        },
    ));
}

```

```
// Langsame Responses
alerts.addRule(AlertRule(
  name: 'slow_responses',
  severity: AlertSeverity.warning,
  message: 'Average response time > 1s',
  condition: () async {
    final hist = metrics.histogram('http_request_duration_seconds');
    return hist.mean > 1.0;
  },
));

// Unhealthy Services
alerts.addRule(AlertRule(
  name: 'service_unhealthy',
  severity: AlertSeverity.critical,
  message: 'One or more services are unhealthy',
  condition: () async {
    final health = await health.checkAll();
    return health.status == HealthStatus.unhealthy;
  },
));
}
```

#### 5.3.0.8 Zusammenfassung

- **Strukturiertes Logging** mit JSON für maschinelle Verarbeitung
- **Log-Level** sinnvoll einsetzen (Debug -> Fatal)
- **Health Checks** für Load Balancer und Orchestrierung
- **Metriken** für Performance-Überwachung
- **Alerting** für proaktive Fehlererkennung

Nächste Schritte

In der nächsten Einheit behandeln wir Deployment & Docker.

### 5.3.1 Übung

#### 5.3.1.1 Ziel

Implementiere ein Logging- und Monitoring-System für einen Produktions-Server.

#### 5.3.1.2 Aufgabe 1: Logger-Klasse (20 min)

Erstelle einen strukturierten Logger.

```
// lib/logging/logger.dart

import 'dart:convert';
import 'dart:io';

enum LogLevel {
  debug(0, 'DEBUG'),
  info(1, 'INFO'),
```

```
warning(2, 'WARN'),
error(3, 'ERROR'),
fatal(4, 'FATAL');

final int value;
final String label;
const LogLevel(this.value, this.label);
}

class Logger {
  final String name;
  final LogLevel minLevel;
  final bool jsonOutput;

  // Context für Request-Tracking
  static String? currentRequestId;
  static String? currentUserId;

  Logger({
    required this.name,
    this.minLevel = LogLevel.info,
    this.jsonOutput = true,
  });

  void debug(String message, [Map<String, dynamic>? data]) {
    // TODO
  }

  void info(String message, [Map<String, dynamic>? data]) {
    // TODO
  }

  void warning(String message, [Map<String, dynamic>? data]) {
    // TODO
  }

  void error(String message, [Object? error, StackTrace? stack]) {
    // TODO: Error und Stack in data einfügen
  }

  void fatal(String message, [Object? error, StackTrace? stack]) {
    // TODO
  }

  void _log(LogLevel level, String message, Map<String, dynamic>? data) {
    // TODO: Level-Filter
    // TODO: LogEntry erstellen
    // TODO: JSON oder Text ausgeben
  }
}
```

```
class LogEntry {
  final DateTime timestamp;
  final LogLevel level;
  final String logger;
  final String message;
  final Map<String, dynamic>? data;
  final String? requestId;
  final String? userId;

  LogEntry({
    required this.timestamp,
    required this.level,
    required this.logger,
    required this.message,
    this.data,
    this.requestId,
    this.userId,
  });

  Map<String, dynamic> toJson() {
    // TODO: Alle Felder als JSON
  }

  String toText() {
    // TODO: Human-readable Format
    // [10:30:45.123] INFO [auth] User logged in (req=abc123)
  }
}
```

### 5.3.1.3 Aufgabe 2: Request Logging Middleware (15 min)

Erstelle eine Middleware die alle Requests loggt.

```
// lib/logging/request_logger.dart

import 'package:shelf/shelf.dart';
import 'package:uuid/uuid.dart';

Middleware requestLoggingMiddleware(Logger logger) {
  return (Handler innerHandler) {
    return (Request request) async {
      // TODO: Request-ID generieren
      // TODO: Logger.currentRequestId setzen
      // TODO: Stopwatch starten
      // TODO: Request-Start loggen

      try {
        final response = await innerHandler(request);

        // TODO: Request-Ende loggen mit Status und Duration
        // TODO: X-Request-ID Header hinzufügen
      }
    };
  };
}
```

```
        return response;
    } catch (e, stack) {
        // TODO: Error loggen
        rethrow;
    } finally {
        // TODO: Context zurücksetzen
    }
};
};
}
```

#### 5.3.1.4 Aufgabe 3: Health Check System (25 min)

Implementiere Health Checks für verschiedene Komponenten.

```
// lib/health/health_check.dart

enum HealthStatus { healthy, degraded, unhealthy }

class HealthCheckResult {
    final String name;
    final HealthStatus status;
    final Duration duration;
    final String? message;
    final Map<String, dynamic>? details;

    HealthCheckResult({
        required this.name,
        required this.status,
        required this.duration,
        this.message,
        this.details,
    });

    Map<String, dynamic> toJson() {
        // TODO
    }
}

abstract class HealthCheck {
    String get name;
    Future<HealthCheckResult> check();
}

// Ping Check - prüft ob ein Service erreichbar ist
class PingHealthCheck implements HealthCheck {
    final String serviceName;
    final Future<void> Function() pingFn;

    PingHealthCheck(this.serviceName, this.pingFn);
}
```

```
@Override
String get name => serviceName;

@Override
Future<HealthCheckResult> check() async {
    // TODO: Stopwatch starten
    // TODO: pingFn aufrufen
    // TODO: Bei Erfolg: healthy
    // TODO: Bei Fehler: unhealthy mit Message
}
}

// Memory Check - prüft Speicherverbrauch
class MemoryHealthCheck implements HealthCheck {
    final int warningMb;
    final int criticalMb;

    MemoryHealthCheck({
        this.warningMb = 500,
        this.criticalMb = 800,
    });

    @Override
    String get name => 'memory';

    @Override
    Future<HealthCheckResult> check() async {
        // TODO: ProcessInfo.currentRss abfragen
        // TODO: Mit Thresholds vergleichen
        // TODO: Status und Details zurückgeben
    }
}

// Uptime Check
class UptimeHealthCheck implements HealthCheck {
    final DateTime startTime;

    UptimeHealthCheck() : startTime = DateTime.now();

    @Override
    String get name => 'uptime';

    @Override
    Future<HealthCheckResult> check() async {
        // TODO: Uptime berechnen und als healthy zurückgeben
    }
}
```

**5.3.1.5 Aufgabe 4: Health Service (15 min)**

Aggregiere alle Health Checks.

```
// lib/health/health_service.dart

class HealthService {
  final List<HealthCheck> _checks = [];
  final Logger _logger;

  HealthService(this._logger);

  void register(HealthCheck check) {
    // TODO
  }

  Future<OverallHealth> checkAll() async {
    // TODO: Alle Checks parallel ausführen
    // TODO: Mit Timeout (5 Sekunden)
    // TODO: Gesamtstatus berechnen (unhealthy wenn einer unhealthy)
  }

  Future<HealthCheckResult?> checkOne(String name) async {
    // TODO: Einzelnen Check ausführen
  }
}

class OverallHealth {
  final HealthStatus status;
  final List<HealthCheckResult> checks;
  final DateTime timestamp;

  OverallHealth({
    required this.status,
    required this.checks,
    required this.timestamp,
  });

  Map<String, dynamic> toJson() {
    // TODO
  }

  int get httpStatusCode => status == HealthStatus.unhealthy ? 503 : 200;
}
```

**5.3.1.6 Aufgabe 5: Metriken (25 min)**

Implementiere Counter, Gauge und Histogram.

```
// lib/metrics/metrics.dart

/// Counter: Zählt nur aufwärts
class Counter {
```

```
final String name;
final String? help;
int _value = 0;

Counter(this.name, {this.help});

void increment([int amount = 1]) {
  // TODO
}

int get value => _value;

void reset() {
  // TODO
}
}

/// Gauge: Kann auf- und abwärts
class Gauge {
  final String name;
  final String? help;
  double _value = 0;

  Gauge(this.name, {this.help});

  void set(double value) {
    // TODO
  }

  void increment([double amount = 1]) {
    // TODO
  }

  void decrement([double amount = 1]) {
    // TODO
  }

  double get value => _value;
}

/// Histogram: Verteilung von Werten
class Histogram {
  final String name;
  final String? help;
  final List<double> buckets;
  final Map<double, int> _bucketCounts = {};
  double _sum = 0;
  int _count = 0;

  Histogram(
    this.name, {
```

```

    this.help,
    this.buckets = const [0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2.5, ↵
↵ 5, 10],
  }) {
    // TODO: _bucketCounts initialisieren
  }

  void observe(double value) {
    // TODO: _count und _sum aktualisieren
    // TODO: Alle Buckets <= value inkrementieren
  }

  int get count => _count;
  double get sum => _sum;
  double get mean => _count > 0 ? _sum / _count : 0;

  /// Perzentil berechnen (approximiert)
  double percentile(double p) {
    // TODO: Aus Bucket-Daten approximieren
  }
}

```

#### 5.3.1.7 Aufgabe 6: Metrics Registry (15 min)

Zentrale Verwaltung aller Metriken.

```

// lib/metrics/registry.dart

class MetricsRegistry {
  final Map<String, Counter> _counters = {};
  final Map<String, Gauge> _gauges = {};
  final Map<String, Histogram> _histograms = {};

  Counter counter(String name, {String? help}) {
    // TODO: Existierenden zurückgeben oder neuen erstellen
  }

  Gauge gauge(String name, {String? help}) {
    // TODO
  }

  Histogram histogram(String name, {String? help, List<double>? buckets}) {
    // TODO
  }

  /// Export als JSON
  Map<String, dynamic> toJson() {
    // TODO: Alle Metriken als JSON
  }

  /// Export im Prometheus-Format

```

```
String toPrometheus() {  
  // TODO: Prometheus Text-Format  
  // # HELP counter_name Description  
  // # TYPE counter_name counter  
  // counter_name 42  
}  
}
```

#### 5.3.1.8 Aufgabe 7: Metrics Middleware (15 min)

Sammle HTTP-Metriken automatisch.

```
// lib/metrics/http_metrics.dart  
  
import 'package:shelf/shelf.dart';  
  
class HttpMetrics {  
  final Counter requestsTotal;  
  final Counter errorsTotal;  
  final Histogram requestDuration;  
  final Gauge activeRequests;  
  
  HttpMetrics(MetricsRegistry registry)  
    : requestsTotal = registry.counter('http_requests_total',  
        help: 'Total HTTP requests'),  
      errorsTotal = registry.counter('http_errors_total',  
        help: 'Total HTTP errors (5xx)'),  
      requestDuration = registry.histogram('http_request_duration_seconds',  
        help: 'HTTP request duration'),  
      activeRequests = registry.gauge('http_requests_active',  
        help: 'Currently active requests');  
}  
  
Middleware httpMetricsMiddleware(HttpMetrics metrics) {  
  return (Handler innerHandler) {  
    return (Request request) async {  
      // TODO: activeRequests.increment()  
      // TODO: requestsTotal.increment()  
      // TODO: Stopwatch starten  
  
      try {  
        final response = await innerHandler(request);  
  
        // TODO: Duration messen und observieren  
        // TODO: Bei 5xx: errorsTotal.increment()  
  
        return response;  
      } catch (e) {  
        // TODO: errorsTotal.increment()  
        rethrow;  
      } finally {  

```

```
        // TODO: activeRequests.decrement()
    }
};
};
}
```

#### 5.3.1.9 Aufgabe 8: Server Assembly (20 min)

Baue alles zusammen.

```
// bin/server.dart

import 'dart:convert';
import 'dart:io';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as io;
import 'package:shelf_router/shelf_router.dart';

void main() async {
  // Logger
  final logger = Logger(
    name: 'server',
    minLevel: LogLevel.debug,
    jsonOutput: false, // Für Entwicklung
  );

  // Metrics
  final metrics = MetricsRegistry();
  final httpMetrics = HttpMetrics(metrics);

  // Health Checks
  final health = HealthService(logger);
  health.register(UptimeHealthCheck());
  health.register(MemoryHealthCheck());

  // Router
  final router = Router();

  // Business Endpoints
  router.get('/api/hello', (Request request) {
    logger.info('Hello endpoint called');
    return Response.ok(jsonEncode({'message': 'Hello World'}));
  });

  router.get('/api/error', (Request request) {
    logger.warning('Error endpoint called');
    throw Exception('Test error');
  });

  // Health Endpoints
  router.get('/health', (Request request) async {
```

```
// TODO: Alle Checks ausführen
// TODO: JSON Response mit richtigem Status Code
});

router.get('/health/live', (Request request) {
  // TODO: Einfaches 200 OK
});

router.get('/health/ready', (Request request) async {
  // TODO: Prüfen ob alle Checks healthy
});

// Metrics Endpoint
router.get('/metrics', (Request request) {
  // TODO: Prometheus-Format zurückgeben
});

router.get('/metrics/json', (Request request) {
  // TODO: JSON-Format zurückgeben
});

// Pipeline
final handler = const Pipeline()
  .addMiddleware(requestLoggingMiddleware(logger))
  .addMiddleware(httpMetricsMiddleware(httpMetrics))
  .addMiddleware(_errorHandler(logger))
  .addHandler(router.call);

// Server starten
final port = int.parse(Platform.environment['PORT'] ?? '8080');
final server = await io.serve(handler, InternetAddress.anyIPv4, port);

logger.info('Server started', {'port': port});

// Graceful Shutdown
ProcessSignal.sigint.watch().listen((_) async {
  logger.info('Shutting down...');
  await server.close();
  exit(0);
});
}

Middleware _errorHandler(Logger logger) {
  return (Handler innerHandler) {
    return (Request request) async {
      try {
        return await innerHandler(request);
      } catch (e, stack) {
        logger.error('Unhandled error', e, stack);
        return Response.internalServerError(
          body: jsonEncode({'error': 'Internal Server Error'}),

```

```
        headers: {'content-type': 'application/json'},
      );
    }
  };
};
}
```

#### 5.3.1.10 Testen

Server starten

```
dart run bin/server.dart
```

Endpoints testen

```
# Hello
curl http://localhost:8080/api/hello

# Error erzeugen
curl http://localhost:8080/api/error

# Health Check
curl http://localhost:8080/health

# Liveness
curl http://localhost:8080/health/live

# Readiness
curl http://localhost:8080/health/ready

# Metriken (Prometheus)
curl http://localhost:8080/metrics

# Metriken (JSON)
curl http://localhost:8080/metrics/json
```

Last erzeugen

```
# 100 Requests
for i in {1..100}; do curl -s http://localhost:8080/api/hello > /dev/null; done

# Metriken prüfen
curl http://localhost:8080/metrics/json | jq
```

#### 5.3.1.11 Bonus: Simple Dashboard (Optional)

```
// Einfaches HTML Dashboard
router.get('/dashboard', (Request request) {
  final html = '''
  <!DOCTYPE html>
  <html>
```

```

<head>
  <title>Server Dashboard</title>
  <script>
    async function refresh() {
      const metrics = await fetch('/metrics/json').then(r => r.json());
      const health = await fetch('/health').then(r => r.json());
      document.getElementById('metrics').textContent =
↪ JSON.stringify(metrics, null, 2);
      document.getElementById('health').textContent =
↪ JSON.stringify(health, null, 2);
    }
    setInterval(refresh, 5000);
    refresh();
  </script>
</head>
<body>
  <h1>Server Dashboard</h1>
  <h2>Health</h2>
  <pre id="health">Loading...</pre>
  <h2>Metrics</h2>
  <pre id="metrics">Loading...</pre>
</body>
</html>
''';
return Response.ok(html, headers: {'content-type': 'text/html'});
});

```

#### 5.3.1.12 Abgabe-Checkliste

- ☐ Logger mit allen Log-Levels
- ☐ LogEntry mit JSON und Text-Format
- ☐ Request Logging Middleware mit Request-ID
- ☐ Health Check Interface und Implementierungen
- ☐ HealthService mit Aggregation
- ☐ Counter, Gauge, Histogram Metriken
- ☐ MetricsRegistry mit Export
- ☐ HTTP Metrics Middleware
- ☐ Health Endpoints (/health, /health/live, /health/ready)
- ☐ Metrics Endpoints (/metrics, /metrics/json)
- ☐ Error Handler Middleware
- ☐ Graceful Shutdown

### 5.3.2 Lösung

#### 5.3.2.1 Aufgabe 1: Logger-Klasse

```

// lib/logging/logger.dart

import 'dart:convert';
import 'dart:io';

```

```
enum LogLevel {
    debug(0, 'DEBUG'),
    info(1, 'INFO'),
    warning(2, 'WARN'),
    error(3, 'ERROR'),
    fatal(4, 'FATAL');

    final int value;
    final String label;
    const LogLevel(this.value, this.label);
}

class Logger {
    final String name;
    final LogLevel minLevel;
    final bool jsonOutput;
    final IOSink _sink;

    static String? currentRequestId;
    static String? currentUserId;

    Logger({
        required this.name,
        this.minLevel = LogLevel.info,
        this.jsonOutput = true,
        IOSink? sink,
    }) : _sink = sink ?? stdout;

    void debug(String message, [Map<String, dynamic>? data]) {
        _log(LogLevel.debug, message, data);
    }

    void info(String message, [Map<String, dynamic>? data]) {
        _log(LogLevel.info, message, data);
    }

    void warning(String message, [Map<String, dynamic>? data]) {
        _log(LogLevel.warning, message, data);
    }

    void error(String message, [Object? error, StackTrace? stack]) {
        _log(LogLevel.error, message, {
            if (error != null) 'error': error.toString(),
            if (stack != null) 'stack_trace':
↵ stack.toString().split('\n').take(10).join('\n'),
        });
    }

    void fatal(String message, [Object? error, StackTrace? stack]) {
        _log(LogLevel.fatal, message, {
            if (error != null) 'error': error.toString(),
```

```
        if (stack != null) 'stack_trace': stack.toString(),
    });
}

void _log(LogLevel level, String message, Map<String, dynamic>? data) {
    if (level.value < minLevel.value) return;

    final entry = LogEntry(
        timestamp: DateTime.now(),
        level: level,
        logger: name,
        message: message,
        data: data,
        requestId: currentRequestId,
        userId: currentUserId,
    );

    if (jsonOutput) {
        _sink.writeln(jsonEncode(entry.toJson()));
    } else {
        _sink.writeln(entry.toText());
    }
}

/// Child Logger mit Prefix
Logger child(String childName) {
    return Logger(
        name: '$name.$childName',
        minLevel: minLevel,
        jsonOutput: jsonOutput,
        sink: _sink,
    );
}

class LogEntry {
    final DateTime timestamp;
    final LogLevel level;
    final String logger;
    final String message;
    final Map<String, dynamic>? data;
    final String? requestId;
    final String? userId;

    LogEntry({
        required this.timestamp,
        required this.level,
        required this.logger,
        required this.message,
        this.data,
        this.requestId,
```

```

    this.userId,
  });

  Map<String, dynamic> toJson() {
    final json = <String, dynamic>{
      'timestamp': timestamp.toUtc().toIso8601String(),
      'level': level.label,
      'logger': logger,
      'message': message,
    };

    if (requestId != null) json['request_id'] = requestId;
    if (userId != null) json['user_id'] = userId;
    if (data != null && data!.isNotEmpty) {
      json.addAll(data!);
    }

    return json;
  }

  String toText() {
    final time = timestamp.toIso8601String().substring(11, 23);
    final levelStr = level.label.padRight(5);
    final loggerStr = '[$logger]'.padRight(15);

    final contextParts = <String>[];
    if (requestId != null) contextParts.add('req=$requestId');
    if (userId != null) contextParts.add('user=$userId');
    final context = contextParts.isNotEmpty ? ' (${contextParts.join(' ')})'
↵ ↵
    : '';

    final dataStr = data != null && data!.isNotEmpty ? ' $data' : '';

    return '[$time] $levelStr $loggerStr $message$context$dataStr';
  }
}

```

### 5.3.2.2 Aufgabe 2: Request Logging Middleware

```

// lib/logging/request_logger.dart

import 'package:shelf/shelf.dart';
import 'package:uuid/uuid.dart';
import 'logger.dart';

Middleware requestLoggingMiddleware(Logger logger) {
  final uuid = Uuid();

  return (Handler innerHandler) {
    return (Request request) async {

```

```
final requestId = uuid.v4().substring(0, 8);
final stopwatch = Stopwatch()..start();

// Context setzen
Logger.currentRequestId = requestId;

// User ID aus Auth extrahieren (falls vorhanden)
final authHeader = request.headers['authorization'];
if (authHeader != null && authHeader.startsWith('Bearer ')) {
  // In echtem Code: Token parsen und User ID extrahieren
}

logger.info('Request started', {
  'method': request.method,
  'path': request.url.path,
  'query': request.url.query.isNotEmpty ? request.url.queryParameters : ↵
↵ null,
  'remote_ip': request.headers['x-forwarded-for'] ?? 'unknown',
  'user_agent': request.headers['user-agent'],
});

try {
  final response = await innerHandler(request);
  stopwatch.stop();

  final logLevel = response.statusCode >= 500
    ? LogLevel.error
    : response.statusCode >= 400
      ? LogLevel.warning
      : LogLevel.info;

  logger._log(logLevel, 'Request completed', {
    'method': request.method,
    'path': request.url.path,
    'status': response.statusCode,
    'duration_ms': stopwatch.elapsedMilliseconds,
  });

  return response.change(headers: {
    'X-Request-ID': requestId,
  });
} catch (e, stack) {
  stopwatch.stop();

  logger.error('Request failed', e, stack);

  rethrow;
} finally {
  Logger.currentRequestId = null;
  Logger.currentUserId = null;
}
```

```
    };  
  };  
}
```

### 5.3.2.3 Aufgabe 3: Health Check System

```
// lib/health/health_check.dart  
  
import 'dart:io';  
  
enum HealthStatus { healthy, degraded, unhealthy }  
  
class HealthCheckResult {  
  final String name;  
  final HealthStatus status;  
  final Duration duration;  
  final String? message;  
  final Map<String, dynamic>? details;  
  
  HealthCheckResult({  
    required this.name,  
    required this.status,  
    required this.duration,  
    this.message,  
    this.details,  
  });  
  
  Map<String, dynamic> toJson() => {  
    'name': name,  
    'status': status.name,  
    'duration_ms': duration.inMilliseconds,  
    if (message != null) 'message': message,  
    if (details != null) 'details': details,  
  };  
}  
  
abstract class HealthCheck {  
  String get name;  
  Future<HealthCheckResult> check();  
}  
  
/// Ping Check - prüft ob ein Service erreichbar ist  
class PingHealthCheck implements HealthCheck {  
  final String serviceName;  
  final Future<void> Function() pingFn;  
  
  PingHealthCheck(this.serviceName, this.pingFn);  
  
  @override  
  String get name => serviceName;
```

```
@override
Future<HealthCheckResult> check() async {
  final stopwatch = Stopwatch()..start();

  try {
    await pingFn();
    stopwatch.stop();

    return HealthCheckResult(
      name: name,
      status: HealthStatus.healthy,
      duration: stopwatch.elapsed,
      message: 'OK',
    );
  } catch (e) {
    stopwatch.stop();

    return HealthCheckResult(
      name: name,
      status: HealthStatus.unhealthy,
      duration: stopwatch.elapsed,
      message: e.toString(),
    );
  }
}

/// Memory Check - prüft Speicherverbrauch
class MemoryHealthCheck implements HealthCheck {
  final int warningMb;
  final int criticalMb;

  MemoryHealthCheck({
    this.warningMb = 500,
    this.criticalMb = 800,
  });

  @override
  String get name => 'memory';

  @override
  Future<HealthCheckResult> check() async {
    final stopwatch = Stopwatch()..start();

    final rss = ProcessInfo.currentRss;
    final usedMb = rss ~/ (1024 * 1024);

    stopwatch.stop();

    final status = usedMb >= criticalMb
```

```

        ? HealthStatus.unhealthy
        : usedMb >= warningMb
            ? HealthStatus.degraded
            : HealthStatus.healthy;

    return HealthCheckResult(
        name: name,
        status: status,
        duration: stopwatch.elapsed,
        message: '$usedMb MB used',
        details: {
            'used_mb': usedMb,
            'warning_threshold_mb': warningMb,
            'critical_threshold_mb': criticalMb,
        },
    );
}
}

/// Uptime Check
class UptimeHealthCheck implements HealthCheck {
    final DateTime startTime;

    UptimeHealthCheck() : startTime = DateTime.now();

    @override
    String get name => 'uptime';

    @override
    Future<HealthCheckResult> check() async {
        final uptime = DateTime.now().difference(startTime);

        return HealthCheckResult(
            name: name,
            status: HealthStatus.healthy,
            duration: Duration.zero,
            message: _formatDuration(uptime),
            details: {
                'started_at': startTime.toIso8601String(),
                'uptime_seconds': uptime.inSeconds,
            },
        );
    }

    String _formatDuration(Duration d) {
        if (d.inDays > 0) return '${d.inDays}d ${d.inHours % 24}h';
        if (d.inHours > 0) return '${d.inHours}h ${d.inMinutes % 60}m';
        if (d.inMinutes > 0) return '${d.inMinutes}m ${d.inSeconds % 60}s';
        return '${d.inSeconds}s';
    }
}

```

```
/// HTTP Endpoint Check
class HttpHealthCheck implements HealthCheck {
    final String serviceName;
    final String url;
    final Duration timeout;

    HttpHealthCheck(this.serviceName, this.url, {this.timeout = const
↵ Duration(seconds: 5)});

    @override
    String get name => serviceName;

    @override
    Future<HealthCheckResult> check() async {
        final stopwatch = Stopwatch()..start();

        try {
            final client = HttpClient();
            client.connectionTimeout = timeout;

            final request = await client.getUrl(Uri.parse(url));
            final response = await request.close().timeout(timeout);

            stopwatch.stop();
            client.close();

            final status = response.statusCode < 400
                ? HealthStatus.healthy
                : response.statusCode < 500
                    ? HealthStatus.degraded
                    : HealthStatus.unhealthy;

            return HealthCheckResult(
                name: name,
                status: status,
                duration: stopwatch.elapsed,
                message: 'Status ${response.statusCode}',
                details: {'status_code': response.statusCode},
            );
        } catch (e) {
            stopwatch.stop();

            return HealthCheckResult(
                name: name,
                status: HealthStatus.unhealthy,
                duration: stopwatch.elapsed,
                message: e.toString(),
            );
        }
    }
}
```

```
}
```

#### 5.3.2.4 Aufgabe 4: Health Service

```
// lib/health/health_service.dart

import 'dart:async';
import 'health_check.dart';
import '../logging/logger.dart';

class HealthService {
  final List<HealthCheck> _checks = [];
  final Logger _logger;
  final Duration timeout;

  HealthService(this._logger, {this.timeout = const Duration(seconds: 5)});

  void register(HealthCheck check) {
    _checks.add(check);
    _logger.debug('Health check registered', {'name': check.name});
  }

  Future<OverallHealth> checkAll() async {
    final results = await Future.wait(
      _checks.map((check) => _runWithTimeout(check)),
    );

    final status = _calculateOverallStatus(results);

    return OverallHealth(
      status: status,
      checks: results,
      timestamp: DateTime.now(),
    );
  }

  Future<HealthCheckResult> _runWithTimeout(HealthCheck check) async {
    try {
      return await check.check().timeout(timeout);
    } on TimeoutException {
      return HealthCheckResult(
        name: check.name,
        status: HealthStatus.unhealthy,
        duration: timeout,
        message: 'Health check timed out',
      );
    } catch (e) {
      return HealthCheckResult(
        name: check.name,
        status: HealthStatus.unhealthy,
```

```

        duration: Duration.zero,
        message: 'Health check error: $e',
    );
}
}

HealthStatus _calculateOverallStatus(List<HealthCheckResult> results) {
    if (results.any((r) => r.status == HealthStatus.unhealthy)) {
        return HealthStatus.unhealthy;
    }
    if (results.any((r) => r.status == HealthStatus.degraded)) {
        return HealthStatus.degraded;
    }
    return HealthStatus.healthy;
}

Future<HealthCheckResult?> checkOne(String name) async {
    final check = _checks.where((c) => c.name == name).firstOrNull;
    if (check == null) return null;
    return await _runWithTimeout(check);
}

List<String> get checkNames => _checks.map((c) => c.name).toList();
}

class OverallHealth {
    final HealthStatus status;
    final List<HealthCheckResult> checks;
    final DateTime timestamp;

    OverallHealth({
        required this.status,
        required this.checks,
        required this.timestamp,
    });

    Map<String, dynamic> toJson() => {
        'status': status.name,
        'timestamp': timestamp.toIso8601String(),
        'checks': checks.map((c) => c.toJson()).toList(),
    };

    int get httpStatusCode => status == HealthStatus.unhealthy ? 503 : 200;
}

```

### 5.3.2.5 Aufgabe 5: Metriken

```

// lib/metrics/metrics.dart

/// Counter: Zählt nur aufwärts

```

```
class Counter {
    final String name;
    final String? help;
    int _value = 0;

    Counter(this.name, {this.help});

    void increment([int amount = 1]) {
        if (amount < 0) throw ArgumentError('Counter can only increment');
        _value += amount;
    }

    int get value => _value;

    void reset() {
        _value = 0;
    }

    Map<String, dynamic> toJson() => {
        'name': name,
        'type': 'counter',
        'value': _value,
        if (help != null) 'help': help,
    };
}

/// Gauge: Kann auf- und abwärts
class Gauge {
    final String name;
    final String? help;
    double _value = 0;

    Gauge(this.name, {this.help});

    void set(double value) {
        _value = value;
    }

    void increment([double amount = 1]) {
        _value += amount;
    }

    void decrement([double amount = 1]) {
        _value -= amount;
    }

    double get value => _value;

    Map<String, dynamic> toJson() => {
        'name': name,
        'type': 'gauge',
```

```

        'value': _value,
        if (help != null) 'help': help,
    };
}

/// Histogram: Verteilung von Werten
class Histogram {
    final String name;
    final String? help;
    final List<double> buckets;
    final Map<double, int> _bucketCounts = {};
    double _sum = 0;
    int _count = 0;
    double _min = double.infinity;
    double _max = double.negativeInfinity;

    Histogram(
        this.name, {
        this.help,
        this.buckets = const [0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2.5,
↵ 5, 10],
    }) {
        for (final b in buckets) {
            _bucketCounts[b] = 0;
        }
        _bucketCounts[double.infinity] = 0; // +Inf bucket
    }

    void observe(double value) {
        _count++;
        _sum += value;
        if (value < _min) _min = value;
        if (value > _max) _max = value;

        for (final bucket in buckets) {
            if (value <= bucket) {
                _bucketCounts[bucket] = (_bucketCounts[bucket] ?? 0) + 1;
            }
        }
        _bucketCounts[double.infinity] = _count; // +Inf always contains all
    }

    int get count => _count;
    double get sum => _sum;
    double get mean => _count > 0 ? _sum / _count : 0;
    double get min => _count > 0 ? _min : 0;
    double get max => _count > 0 ? _max : 0;

    /// Perzentil approximieren
    double percentile(double p) {
        if (_count == 0) return 0;

```

```

    if (p < 0 || p > 1) throw ArgumentError('Percentile must be between 0 and 1');
    ↪

    final target = (p * _count).ceil();
    int cumulative = 0;

    for (final bucket in buckets) {
        cumulative = _bucketCounts[bucket] ?? 0;
        if (cumulative >= target) {
            return bucket;
        }
    }

    return buckets.last;
}

Map<String, dynamic> toJson() => {
    'name': name,
    'type': 'histogram',
    'count': _count,
    'sum': _sum,
    'mean': mean,
    'min': _count > 0 ? _min : null,
    'max': _count > 0 ? _max : null,
    'p50': percentile(0.5),
    'p90': percentile(0.9),
    'p99': percentile(0.99),
    if (help != null) 'help': help,
};

String toPrometheus() {
    final buffer = StringBuffer();
    if (help != null) {
        buffer.writeln('# HELP $name $help');
    }
    buffer.writeln('# TYPE $name histogram');

    for (final bucket in buckets) {
        buffer.writeln('${name}_bucket{le="$bucket"} ${_bucketCounts[bucket]}');
    }
    buffer.writeln('${name}_bucket{le="+Inf"} $_count');
    buffer.writeln('${name}_sum $sum');
    buffer.writeln('${name}_count $count');

    return buffer.toString();
}
}

```

**5.3.2.6 Aufgabe 6: Metrics Registry**

```
// lib/metrics/registry.dart

import 'metrics.dart';

class MetricsRegistry {
  final Map<String, Counter> _counters = {};
  final Map<String, Gauge> _gauges = {};
  final Map<String, Histogram> _histograms = {};

  Counter counter(String name, {String? help}) {
    return _counters.putIfAbsent(name, () => Counter(name, help: help));
  }

  Gauge gauge(String name, {String? help}) {
    return _gauges.putIfAbsent(name, () => Gauge(name, help: help));
  }

  Histogram histogram(String name, {String? help, List<double>? buckets}) {
    return _histograms.putIfAbsent(
      name,
      () => Histogram(name, help: help, buckets: buckets ?? const [0.005,
↵ 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2.5, 5, 10]),
    );
  }

  Map<String, dynamic> toJson() => {
    'counters': {
      for (final c in _counters.values) c.name: c.value,
    },
    'gauges': {
      for (final g in _gauges.values) g.name: g.value,
    },
    'histograms': {
      for (final h in _histograms.values) h.name: h.toJson(),
    },
  };

  String toPrometheus() {
    final buffer = StringBuffer();

    // Counters
    for (final counter in _counters.values) {
      if (counter.help != null) {
        buffer.writeln('# HELP ${counter.name} ${counter.help}');
      }
      buffer.writeln('# TYPE ${counter.name} counter');
      buffer.writeln('${counter.name} ${counter.value}');
      buffer.writeln();
    }
  }
}
```

```

// Gauges
for (final gauge in _gauges.values) {
  if (gauge.help != null) {
    buffer.writeln('# HELP ${gauge.name} ${gauge.help}');
  }
  buffer.writeln('# TYPE ${gauge.name} gauge');
  buffer.writeln('${gauge.name} ${gauge.value}');
  buffer.writeln();
}

// Histograms
for (final histogram in _histograms.values) {
  buffer.write(histogram.toPrometheus());
  buffer.writeln();
}

return buffer.toString();
}

void reset() {
  for (final c in _counters.values) {
    c.reset();
  }
  _gauges.clear();
  _histograms.clear();
}
}

```

### 5.3.2.7 Aufgabe 7: HTTP Metrics Middleware

```

// lib/metrics/http_metrics.dart

import 'package:shelf/shelf.dart';
import 'metrics.dart';
import 'registry.dart';

class HttpMetrics {
  final Counter requestsTotal;
  final Counter errorsTotal;
  final Histogram requestDuration;
  final Gauge activeRequests;

  HttpMetrics(MetricsRegistry registry)
    : requestsTotal = registry.counter(
        'http_requests_total',
        help: 'Total number of HTTP requests',
      ),
      errorsTotal = registry.counter(
        'http_errors_total',

```

```

        help: 'Total number of HTTP errors (5xx)',
      ),
      requestDuration = registry.histogram(
        'http_request_duration_seconds',
        help: 'HTTP request duration in seconds',
        buckets: [0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5, 1, 2.5, 5, 10],
      ),
      activeRequests = registry.gauge(
        'http_requests_active',
        help: 'Number of currently active HTTP requests',
      );
}

Middleware httpMetricsMiddleware(HttpMetrics metrics) {
  return (Handler innerHandler) {
    return (Request request) async {
      metrics.activeRequests.increment();
      metrics.requestsTotal.increment();

      final stopwatch = Stopwatch()..start();

      try {
        final response = await innerHandler(request);
        stopwatch.stop();

        // Duration in Sekunden
        metrics.requestDuration.observe(stopwatch.elapsedMilliseconds / 1000);

        // 5xx Errors zählen
        if (response.statusCode >= 500) {
          metrics.errorsTotal.increment();
        }

        return response;
      } catch (e) {
        stopwatch.stop();
        metrics.requestDuration.observe(stopwatch.elapsedMilliseconds / 1000);
        metrics.errorsTotal.increment();
        rethrow;
      } finally {
        metrics.activeRequests.decrement();
      }
    };
  };
}

```

### 5.3.2.8 Aufgabe 8: Server Assembly

```
// bin/server.dart
```

```
import 'dart:convert';
import 'dart:io';
import 'package:shelf/shelf.dart';
import 'package:shelf/shelf_io.dart' as io;
import 'package:shelf_router/shelf_router.dart';

import '../lib/logging/logger.dart';
import '../lib/logging/request_logger.dart';
import '../lib/health/health_check.dart';
import '../lib/health/health_service.dart';
import '../lib/metrics/metrics.dart';
import '../lib/metrics/registry.dart';
import '../lib/metrics/http_metrics.dart';

void main() async {
  // Logger
  final logger = Logger(
    name: 'server',
    minLevel: LogLevel.values.byName(
      Platform.environment['LOG_LEVEL'] ?? 'debug',
    ),
    jsonOutput: Platform.environment['LOG_FORMAT'] == 'json',
  );

  // Metrics
  final metrics = MetricsRegistry();
  final httpMetrics = HttpMetrics(metrics);

  // Custom Business Metrics
  final usersCreated = metrics.counter('users_created_total', help: 'Total ↵
↵ users created');
  final cacheHits = metrics.counter('cache_hits_total');
  final cacheMisses = metrics.counter('cache_misses_total');

  // Health Checks
  final health = HealthService(logger);
  health.register(UptimeHealthCheck());
  health.register(MemoryHealthCheck(warningMb: 200, criticalMb: 400));

  // Router
  final router = Router();

  // Business Endpoints
  router.get('/api/hello', (Request request) {
    logger.info('Hello endpoint called');
    return Response.ok(
      jsonEncode({'message': 'Hello World', 'timestamp': ↵
↵ DateTime.now().toIso8601String()}),
      headers: {'content-type': 'application/json'},
    );
  });
}
```

```
router.post('/api/users', (Request request) async {
  logger.info('Creating user');
  usersCreated.increment();

  // Simuliere User-Erstellung
  await Future.delayed(Duration(milliseconds: 100));

  return Response.ok(
    jsonEncode({'id': 1, 'created': true}),
    headers: {'content-type': 'application/json'},
  );
});

router.get('/api/slow', (Request request) async {
  logger.debug('Slow endpoint called');
  await Future.delayed(Duration(seconds: 2));
  return Response.ok(jsonEncode({'message': 'Finally done'}));
});

router.get('/api/error', (Request request) {
  logger.warning('Error endpoint called');
  throw Exception('Intentional test error');
});

// Health Endpoints
router.get('/health', (Request request) async {
  final result = await health.checkAll();
  return Response(
    result.statusCode,
    body: jsonEncode(result.toJson()),
    headers: {'content-type': 'application/json'},
  );
});

router.get('/health/live', (Request request) {
  return Response.ok(
    jsonEncode({'status': 'alive', 'timestamp':
↵ DateTime.now().toIso8601String()}),
    headers: {'content-type': 'application/json'},
  );
});

router.get('/health/ready', (Request request) async {
  final result = await health.checkAll();
  if (result.status == HealthStatus.unhealthy) {
    return Response(
      503,
      body: jsonEncode({'status': 'not_ready', 'checks': result.toJson()}),
      headers: {'content-type': 'application/json'},
    );
  }
}
```

```
}
return Response.ok(
  jsonEncode({'status': 'ready'}),
  headers: {'content-type': 'application/json'},
);
});

router.get('/health/<check>', (Request request, String check) async {
  final result = await health.checkOne(check);
  if (result == null) {
    return Response.notFound(jsonEncode({'error': 'Check not found'}));
  }
  return Response(
    result.status == HealthStatus.unhealthy ? 503 : 200,
    body: jsonEncode(result.toJson()),
    headers: {'content-type': 'application/json'},
  );
});

// Metrics Endpoints
router.get('/metrics', (Request request) {
  return Response.ok(
    metrics.toPrometheus(),
    headers: {'content-type': 'text/plain; charset=utf-8'},
  );
});

router.get('/metrics/json', (Request request) {
  return Response.ok(
    jsonEncode(metrics.toJson()),
    headers: {'content-type': 'application/json'},
  );
});

// Pipeline
final handler = const Pipeline()
  .addMiddleware(requestLoggingMiddleware(logger))
  .addMiddleware(httpMetricsMiddleware(httpMetrics))
  .addMiddleware(_errorHandler(logger))
  .addHandler(router.call);

// Server starten
final port = int.parse(Platform.environment['PORT'] ?? '8080');
final server = await io.serve(handler, InternetAddress.anyIPv4, port);

logger.info('Server started', {
  'port': port,
  'pid': pid,
  'dart_version': Platform.version.split(' ').first,
});
```

```

print('');
print('');
print('Monitoring Server on port $port');
print('');
print('Endpoints:');
print('  GET /api/hello      - Hello World');
print('  POST /api/users     - Create user');
print('  GET /api/slow       - Slow endpoint (2s)');
print('  GET /api/error      - Error endpoint');
print('  GET /health        - Full health check');
print('  GET /health/live    - Liveness probe');
print('  GET /health/ready   - Readiness probe');
print('  GET /metrics       - Prometheus metrics');
print('  GET /metrics/json   - JSON metrics');
print('');
print('');

// Graceful Shutdown
ProcessSignal.sigint.watch().listen((_) async {
  logger.info('Shutdown signal received');
  await server.close();
  logger.info('Server stopped');
  exit(0);
});
}

Middleware _errorHandler(Logger logger) {
  return (Handler innerHandler) {
    return (Request request) async {
      try {
        return await innerHandler(request);
      } catch (e, stack) {
        logger.error('Unhandled error', e, stack);

        return Response.internalServerError(
          body: jsonEncode({
            'error': 'Internal Server Error',
            'request_id': Logger.currentRequestId,
          }),
          headers: {'content-type': 'application/json'},
        );
      }
    };
  };
}
}

```

### 5.3.2.9 Projektstruktur

```

monitoring_server/
+-- bin/
|   +-- server.dart

```

```
+-- lib/
|   +-- logging/
|   |   +-- logger.dart
|   |   +-- request_logger.dart
|   +-- health/
|   |   +-- health_check.dart
|   |   +-- health_service.dart
|   +-- metrics/
|       +-- metrics.dart
|       +-- registry.dart
|       +-- http_metrics.dart
+-- pubspec.yaml
+-- README.md
```

#### 5.3.2.10 Test-Befehle

```
# Server starten
dart run bin/server.dart

# Verschiedene Endpoints aufrufen
curl http://localhost:8080/api/hello
curl -X POST http://localhost:8080/api/users
curl http://localhost:8080/api/slow
curl http://localhost:8080/api/error

# Health Checks
curl http://localhost:8080/health | jq
curl http://localhost:8080/health/live
curl http://localhost:8080/health/ready
curl http://localhost:8080/health/memory

# Metrics
curl http://localhost:8080/metrics
curl http://localhost:8080/metrics/json | jq

# Last erzeugen
for i in {1..50}; do
  curl -s http://localhost:8080/api/hello &
done
wait

# Metrics nach Last prüfen
curl http://localhost:8080/metrics/json | jq
```

### 5.3.3 Ressourcen

#### 5.3.3.1 Offizielle Dokumentation

- Dart logging Package
- Prometheus Data Model
- OpenTelemetry
- 12 Factor App - Logs

### 5.3.3.2 Cheat Sheet: Log Levels

```
// Wann welches Level?

// DEBUG - Detaillierte Debugging-Infos
logger.debug('Query executed', {'sql': query, 'params': params,
  ↪ 'duration_ms': 5});

// INFO - Normale Geschäftsereignisse
logger.info('User logged in', {'user_id': 123});
logger.info('Order created', {'order_id': 456, 'total': 99.99});

// WARNING - Unerwartetes, aber behandeltes Verhalten
logger.warning('Cache miss', {'key': 'user:123'});
logger.warning('Retry attempt', {'attempt': 2, 'max': 3});

// ERROR - Fehler die behandelt wurden
logger.error('Payment failed', exception, stackTrace);
logger.error('External API error', {'status': 503, 'service': 'stripe'});

// FATAL - Kritische Fehler, System instabil
logger.fatal('Database connection lost');
logger.fatal('Out of memory');
```

### 5.3.3.3 Cheat Sheet: Strukturierte Logs

```
// JSON Format (für Produktion)
{
  "timestamp": "2024-01-15T10:30:00.123Z",
  "level": "INFO",
  "logger": "auth",
  "message": "User logged in",
  "request_id": "abc123",
  "user_id": "456",
  "ip": "192.168.1.1",
  "duration_ms": 45
}

// Text Format (für Entwicklung)
[10:30:00.123] INFO [auth] User logged in (req=abc123 user=456)
```

### 5.3.3.4 Cheat Sheet: Health Checks

```
// Kubernetes Probes
// Liveness: Ist der Prozess am Leben?
GET /health/live
Response: 200 OK

// Readiness: Kann der Service Traffic annehmen?
GET /health/ready
```

Response: 200 OK oder 503 Service Unavailable

// Startup: Ist der Service gestartet? (für langsame Starts)

GET /health/startup

Response: 200 OK

// Vollständiger Health Check

GET /health

```
{
  "status": "healthy",
  "checks": [
    {"name": "database", "status": "healthy", "duration_ms": 5},
    {"name": "redis", "status": "healthy", "duration_ms": 2},
    {"name": "memory", "status": "healthy", "used_mb": 150}
  ]
}
```

#### 5.3.3.5 Cheat Sheet: Metriken

// Counter (nur aufwärts)

http\_requests\_total

errors\_total

users\_created\_total

// Gauge (auf/ab)

http\_requests\_active

memory\_used\_bytes

queue\_size

// Histogram (Verteilung)

http\_request\_duration\_seconds

response\_size\_bytes

// Prometheus Format

# HELP http\_requests\_total Total HTTP requests

# TYPE http\_requests\_total counter

http\_requests\_total 1234

# HELP http\_request\_duration\_seconds Request duration

# TYPE http\_request\_duration\_seconds histogram

http\_request\_duration\_seconds\_bucket{le="0.01"} 100

http\_request\_duration\_seconds\_bucket{le="0.05"} 200

http\_request\_duration\_seconds\_bucket{le="0.1"} 250

http\_request\_duration\_seconds\_bucket{le="+Inf"} 300

http\_request\_duration\_seconds\_sum 15.5

http\_request\_duration\_seconds\_count 300

### 5.3.3.6 Cheat Sheet: Request Context

```
// Request-ID durch alle Logs
Middleware requestContext() {
    return (Handler inner) {
        return (Request request) async {
            final requestId = request.headers['x-request-id']
                ?? Uuid().v4().substring(0, 8);

            Logger.currentRequestId = requestId;

            try {
                final response = await inner(request);
                return response.change(headers: {'x-request-id': requestId});
            } finally {
                Logger.currentRequestId = null;
            }
        };
    };
}
```

### 5.3.3.7 Cheat Sheet: Error Logging

```
// Fehler mit Kontext
try {
    await riskyOperation();
} catch (e, stack) {
    logger.error('Operation failed', e, stack);

    // Oder mit mehr Kontext
    logger.error('Operation failed', {
        'error': e.toString(),
        'stack': stack.toString().split('\n').take(10).toList(),
        'user_id': currentUserId,
        'operation': 'riskyOperation',
    });

    rethrow;
}
```

### 5.3.3.8 Cheat Sheet: Performance Logging

```
// Timing messen
Future<T> timed<T>(String name, Future<T> Function() fn) async {
    final stopwatch = Stopwatch()..start();
    try {
        return await fn();
    } finally {
        stopwatch.stop();
        logger.debug('$name completed', {
```

```

        'duration_ms': stopwatch.elapsedMilliseconds,
    });
    metrics.histogram('operation_duration_seconds').observe(
        stopwatch.elapsedMilliseconds / 1000,
    );
}
}

// Verwendung
final user = await timed('fetch_user', () => db.findUser(id));

```

#### 5.3.3.9 Best Practices

##### DO

1. **Strukturierte Logs** - JSON für maschinelle Verarbeitung
2. **Request-ID** - Durch alle Services tracken
3. **Sensitive Daten maskieren** - Passwörter, Tokens
4. **Log-Level nach Umgebung** - DEBUG in Dev, INFO in Prod
5. **Health Checks** - Liveness + Readiness trennen
6. **Timeouts** - Für Health Checks
7. **Metriken für wichtige Operationen** - Request-Zeit, Fehlerrate

##### DON'T

1. **print() in Produktion** - Unstrukturiert, kein Level
2. **Sensitive Daten loggen** - Passwörter, API Keys
3. **Zu viel loggen** - Performance-Impact
4. **Blocking I/O** - Logs asynchron schreiben
5. **Health Check = Geschäftslogik** - Nur Infrastruktur prüfen

#### 5.3.3.10 Sensitive Daten maskieren

```

String maskSensitive(String input, {int visibleChars = 4}) {
    if (input.length <= visibleChars) return '***';
    return '${input.substring(0, visibleChars)}${'*' * (input.length -
↪   visibleChars)}';
}

Map<String, dynamic> sanitizeForLogging(Map<String, dynamic> data) {
    final sanitized = Map<String, dynamic>.from(data);

    final sensitiveKeys = ['password', 'token', 'secret', 'api_key',
↪   'authorization'];

    for (final key in sensitized.keys.toList()) {
        if (sensitiveKeys.any((s) => key.toLowerCase().contains(s))) {
            sanitized[key] = '***REDACTED***';
        }
    }
}

```

```
    return sanitized;
}
```

#### 5.3.3.11 Log Aggregation

```
# docker-compose.yml mit ELK Stack
services:
  app:
    build: .
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"

# Logs sammeln mit Filebeat
filebeat:
  image: elastic/filebeat:8.0.0
  volumes:
    - /var/lib/docker/containers:/var/lib/docker/containers:ro
```

#### 5.3.3.12 Alerting Beispiele

```
// Fehlerrate > 5%
if (errors / requests > 0.05) {
  alert('High error rate', AlertSeverity.critical);
}

// Response Zeit > 1s
if (histogram.percentile(0.95) > 1.0) {
  alert('Slow responses', AlertSeverity.warning);
}

// Memory > 80%
if (memoryUsedPercent > 80) {
  alert('High memory usage', AlertSeverity.warning);
}

// Service unhealthy
if (health.status == HealthStatus.unhealthy) {
  alert('Service unhealthy', AlertSeverity.critical);
}
```

#### 5.3.3.13 Tools

- **Prometheus** - Metrics Collection
- **Grafana** - Dashboards
- **ELK Stack** - Log Aggregation (Elasticsearch, Logstash, Kibana)
- **Loki** - Log Aggregation (leichtgewichtig)
- **Jaeger/Zipkin** - Distributed Tracing

- PagerDuty/Opsgenie - Alerting

## 5.4 Einheit 9.4: Deployment & Docker

### 5.4.0.1 Lernziele

- Docker-Container für Dart-Anwendungen erstellen
- Multi-Stage Builds für optimale Images
- Docker Compose für lokale Entwicklung
- Cloud-Deployment (Railway, Fly.io)
- CI/CD Pipeline Grundlagen

### 5.4.0.2 Docker Grundlagen

Warum Docker?

"Es funktioniert auf meinem Rechner"

↓

Docker Container

↓

"Es funktioniert überall gleich"

Vorteile

Aspekt	Ohne Docker	Mit Docker
Umgebung	"Installiere Dart 3.2, Redis, PostgreSQL..."	<code>docker compose up</code>
Konsistenz	Unterschiede Dev/Prod	Identische Container
Isolation	Konflikte zwischen Projekten	Isolierte Umgebungen
Deployment	Manuelle Server-Konfiguration	Container starten

### 5.4.0.3 Dockerfile für Dart

Einfaches Dockerfile

```
# Dockerfile

# Basis-Image mit Dart SDK
FROM dart:stable AS build

# Arbeitsverzeichnis
WORKDIR /app

# Dependencies zuerst (Cache-Optimierung)
COPY pubspec.* ./
RUN dart pub get

# Quellcode kopieren
COPY . .

# Kompilieren
RUN dart compile exe bin/server.dart -o bin/server
```

```
# Runtime-Image (minimal)
FROM scratch

# Kompilierte Binary kopieren
COPY --from=build /runtime/ /
COPY --from=build /app/bin/server /app/bin/server

# Port freigeben
EXPOSE 8080

# Startbefehl
CMD ["/app/bin/server"]
```

## Multi-Stage Build erklärt

```
+-----+
| Stage 1: build (dart:stable ~800MB) |
| +-- dart pub get |
| +-- dart compile exe |
| +-- Ergebnis: /app/bin/server |
+-----+
|
| ▼ Nur Binary kopieren
+-----+
| Stage 2: runtime (scratch ~0MB) |
| +-- /app/bin/server |
| +-- Finale Größe: ~10-20MB |
+-----+
```

## Optimiertes Dockerfile

```
# Dockerfile.optimized

# ===== BUILD STAGE =====
FROM dart:stable AS build

WORKDIR /app

# Dependencies cachen
COPY pubspec.* ./
RUN dart pub get --no-precompile

# Code kopieren und kompilieren
COPY . .
RUN dart pub get --offline
RUN dart compile exe bin/server.dart -o bin/server

# ===== RUNTIME STAGE =====
FROM debian:bookworm-slim AS runtime

# Nur notwendige Runtime-Dateien
RUN apt-get update && apt-get install -y --no-install-recommends \
```

```
ca-certificates \
&& rm -rf /var/lib/apt/lists/*

# Non-root User
RUN useradd -r -u 1001 appuser
USER appuser

WORKDIR /app

# Binary kopieren
COPY --from=build /app/bin/server /app/bin/server

# Environment
ENV PORT=8080
EXPOSE 8080

# Health Check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s \
  CMD curl -f http://localhost:8080/health || exit 1

# Startbefehl
ENTRYPOINT ["/app/bin/server"]
```

.dockerignore

```
# .dockerignore
```

```
# Dart/Pub
.dart_tool/
.packages
pubspec.lock
build/
```

```
# IDE
.idea/
.vscode/
*.iml
```

```
# Git
.git/
.gitignore
```

```
# Docs
*.md
LICENSE
```

```
# Tests
test/
coverage/
```

```
# Local files
```

```
.env
.env.local
docker-compose.override.yml
```

#### 5.4.0.4 Docker Compose

Lokale Entwicklungsumgebung

```
# docker-compose.yml

services:
  # Dart API Server
  api:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    environment:
      - PORT=8080
      - DATABASE_URL=postgres://postgres:postgres@db:5432/myapp
      - REDIS_URL=redis://redis:6379
      - JWT_SECRET=dev-secret-change-in-prod
    depends_on:
      db:
        condition: service_healthy
      redis:
        condition: service_started
    volumes:
      # Hot-reload für Entwicklung (optional)
      - ./app:delegated
    restart: unless-stopped

  # PostgreSQL Datenbank
  db:
    image: postgres:15-alpine
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: myapp
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql:ro
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U postgres"]
      interval: 5s
      timeout: 5s
      retries: 5
```

```
# Redis Cache
redis:
  image: redis:7-alpine
  ports:
    - "6379:6379"
  volumes:
    - redis_data:/data

# Adminer (DB UI)
adminer:
  image: adminer
  ports:
    - "8081:8080"
  depends_on:
    - db

volumes:
  postgres_data:
  redis_data:
```

Compose-Befehle

```
# Starten
docker compose up -d

# Logs anzeigen
docker compose logs -f api

# Stoppen
docker compose down

# Mit Volumes löschen
docker compose down -v

# Neu bauen
docker compose build --no-cache

# In Container ausführen
docker compose exec api dart run bin/migrate.dart
```

Entwicklung vs. Produktion

```
# docker-compose.override.yml (automatisch geladen)

services:
  api:
    build:
      target: build # Verwende Build-Stage für Dev
    command: dart run bin/server.dart
    volumes:
      - ./app
    environment:
```

```
- LOG_LEVEL=debug
```

```
# docker-compose.prod.yml
```

```
services:
  api:
    image: myapp/api:${VERSION:-latest}
    deploy:
      replicas: 3
      resources:
        limits:
          cpus: '0.5'
          memory: 512M
    environment:
      - LOG_LEVEL=info
      - LOG_FORMAT=json
```

```
# Produktion starten
```

```
docker compose -f docker-compose.yml -f docker-compose.prod.yml up -d
```

#### 5.4.0.5 Cloud Deployment

Railway

Railway ist eine einfache PaaS für Container-Deployment.

```
# railway.toml
```

```
[build]
builder = "dockerfile"
dockerfilePath = "Dockerfile"

[deploy]
healthcheckPath = "/health"
healthcheckTimeout = 100
restartPolicyType = "ON_FAILURE"
restartPolicyMaxRetries = 3

[[services]]
name = "api"
```

```
# Railway CLI installieren
npm install -g @railway/cli
```

```
# Login
railway login
```

```
# Projekt erstellen
railway init
```

```
# Deployen
railway up
```

```
# Umgebungsvariablen setzen
railway variables set JWT_SECRET=my-secret
railway variables set DATABASE_URL=${Postgres.DATABASE_URL}}
```

Fly.io

Fly.io deployed Container weltweit.

```
# fly.toml

app = "my-dart-api"
primary_region = "fra" # Frankfurt

[build]
  dockerfile = "Dockerfile"

[env]
  PORT = "8080"
  LOG_FORMAT = "json"

[http_service]
  internal_port = 8080
  force_https = true
  auto_stop_machines = true
  auto_start_machines = true

[[services]]
  protocol = "tcp"
  internal_port = 8080

  [[services.ports]]
    port = 80
    handlers = ["http"]

  [[services.ports]]
    port = 443
    handlers = ["tls", "http"]

  [[services.tcp_checks]]
    grace_period = "5s"
    interval = "15s"
    timeout = "2s"
```

```
# Fly CLI installieren
curl -L https://fly.io/install.sh | sh

# Login
fly auth login

# App erstellen
fly launch
```

```
# Secrets setzen
fly secrets set JWT_SECRET=my-secret
fly secrets set DATABASE_URL=postgres://...

# Deployen
fly deploy

# Logs anzeigen
fly logs

# Skalieren
fly scale count 3
```

Datenbank-Provisioning

```
# Fly.io - PostgreSQL
fly postgres create
fly postgres attach my-dart-api

# Railway - PostgreSQL hinzufügen
# (Im Web UI: New -> Database -> PostgreSQL)
```

#### 5.4.0.6 CI/CD Pipeline

GitHub Actions

```
# .github/workflows/deploy.yml

name: Build and Deploy

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${github.repository}

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4

      - uses: dart-lang/setup-dart@v1
        with:
          sdk: stable

      - name: Install dependencies
```

```
    run: dart pub get

  - name: Analyze
    run: dart analyze

  - name: Test
    run: dart test

build:
  needs: test
  runs-on: ubuntu-latest
  permissions:
    contents: read
    packages: write

  steps:
    - uses: actions/checkout@v4

    - name: Log in to Container Registry
      uses: docker/login-action@v3
      with:
        registry: ${ env.REGISTRY }
        username: ${ github.actor }
        password: ${ secrets.GITHUB_TOKEN }

    - name: Extract metadata
      id: meta
      uses: docker/metadata-action@v5
      with:
        images: ${ env.REGISTRY }/${ env.IMAGE_NAME }
        tags: |
          type=sha,prefix=
          type=ref,event=branch
          type=semver,pattern={{version}}

    - name: Build and push
      uses: docker/build-push-action@v5
      with:
        context: .
        push: ${ github.event_name != 'pull_request' }
        tags: ${ steps.meta.outputs.tags }
        labels: ${ steps.meta.outputs.labels }

deploy:
  needs: build
  runs-on: ubuntu-latest
  if: github.ref == 'refs/heads/main'

  steps:
    - uses: actions/checkout@v4
```

```
- name: Deploy to Fly.io
  uses: superfly/flyctl-actions/setup-flyctl@master

- run: flyctl deploy --remote-only
  env:
    FLY_API_TOKEN: ${ secrets.FLY_API_TOKEN }
```

#### GitLab CI

```
# .gitlab-ci.yml

stages:
  - test
  - build
  - deploy

variables:
  DOCKER_IMAGE: $CI_REGISTRY_IMAGE:$CI_COMMIT_SHA

test:
  stage: test
  image: dart:stable
  script:
    - dart pub get
    - dart analyze
    - dart test

build:
  stage: build
  image: docker:latest
  services:
    - docker:dind
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
    - docker build -t $DOCKER_IMAGE .
    - docker push $DOCKER_IMAGE
  only:
    - main

deploy:
  stage: deploy
  image: alpine:latest
  script:
    - apk add --no-cache curl
    - curl -L https://fly.io/install.sh | sh
    - flyctl deploy --image $DOCKER_IMAGE
  environment:
    name: production
  only:
    - main
```

### 5.4.0.7 Best Practices

#### Security

```
# Non-root User
RUN useradd -r -u 1001 appuser
USER appuser

# Keine Secrets im Image
# Stattdessen: Environment Variables

# Minimales Base Image
FROM debian:bookworm-slim
# oder: FROM distroless/base
```

#### Secrets Management

```
# NIE im Dockerfile oder docker-compose.yml:
ENV JWT_SECRET=my-super-secret # []

# Stattdessen:
# 1. .env Datei (nicht committen!)
# 2. Docker Secrets
# 3. Cloud Provider Secrets (Railway, Fly.io)
```

```
# docker-compose.yml
services:
  api:
    env_file:
      - .env # Gitignored!
    secrets:
      - jwt_secret

secrets:
  jwt_secret:
    file: ./secrets/jwt_secret.txt
```

#### Logging

```
# Logs nach stdout/stderr (nicht in Dateien)
# Docker/Kubernetes sammeln diese automatisch

# In Dart:
// Nicht: File('app.log').writeAsStringSync(...)
// Sondern: print(...) oder stdout.writeln(...)
```

#### Health Checks

```
HEALTHCHECK --interval=30s --timeout=3s --start-period=10s --retries=3 \
  CMD curl -f http://localhost:8080/health || exit 1
```

### 5.4.0.8 Zusammenfassung

- **Docker** für konsistente Umgebungen

- **Multi-Stage Builds** für kleine Images
- **Docker Compose** für lokale Entwicklung
- **Railway/Fly.io** für einfaches Cloud-Deployment
- **CI/CD** für automatisierte Deployments

#### Deployment-Checkliste

##### #### Pre-Deployment

- [ ] Tests grün
- [ ] Dockerfile optimiert
- [ ] .dockerignore vorhanden
- [ ] Health Checks implementiert
- [ ] Environment Variables dokumentiert
- [ ] Secrets sicher gespeichert

##### #### Deployment

- [ ] Container baut erfolgreich
- [ ] Health Check erfolgreich
- [ ] Logs verfügbar
- [ ] Metriken verfügbar
- [ ] Rollback-Plan vorhanden

##### #### Post-Deployment

- [ ] Smoke Tests durchgeführt
- [ ] Monitoring prüfen
- [ ] Alerting aktiv

### 5.4.1 Übung

#### 5.4.1.1 Ziel

Erstelle ein produktionsreifes Docker-Setup für eine Dart API.

#### 5.4.1.2 Aufgabe 1: Einfaches Dockerfile (15 min)

Erstelle ein Dockerfile für eine Dart-Anwendung.

```
# Dockerfile

# TODO: Dart Base Image verwenden
FROM ???

# TODO: Arbeitsverzeichnis setzen
WORKDIR ???

# TODO: pubspec.yaml und pubspec.lock kopieren
COPY ???

# TODO: Dependencies installieren
RUN ???
```

```
# TODO: Restlichen Code kopieren
COPY ???

# TODO: Anwendung kompilieren
RUN dart compile exe bin/server.dart -o bin/server

# TODO: Port freigeben
EXPOSE ???

# TODO: Startbefehl
CMD ???
```

Test:

```
docker build -t my-api .
docker run -p 8080:8080 my-api
curl http://localhost:8080/health
```

#### 5.4.1.3 Aufgabe 2: Multi-Stage Build (20 min)

Optimiere das Dockerfile mit Multi-Stage Build.

```
# Dockerfile.optimized

# ===== BUILD STAGE =====
FROM dart:stable AS build

WORKDIR /app

# Dependencies cachen (Layer-Optimierung)
COPY pubspec.* ./
RUN dart pub get

# Code kopieren
COPY . .

# Kompilieren
RUN dart compile exe bin/server.dart -o bin/server

# ===== RUNTIME STAGE =====
# TODO: Minimales Base Image wählen
FROM ???

# TODO: CA-Zertifikate für HTTPS installieren (falls debian-slim)
RUN ???

# TODO: Non-root User erstellen
RUN ???
USER ???

WORKDIR /app
```

```
# TODO: Nur die kompilierte Binary kopieren
COPY --from=build ???

# TODO: Environment Variable für Port
ENV ???

# TODO: Port freigeben
EXPOSE ???

# TODO: Health Check hinzufügen
HEALTHCHECK ???

# TODO: Startbefehl
ENTRYPOINT ???
```

Test:

```
# Image-Größe vergleichen
docker build -t my-api:simple -f Dockerfile .
docker build -t my-api:optimized -f Dockerfile.optimized .
docker images | grep my-api
```

#### 5.4.1.4 Aufgabe 3: .dockerignore (10 min)

Erstelle eine .dockerignore Datei.

```
# .dockerignore

# TODO: Dart-spezifische Dateien/Ordner ausschließen
# (.dart_tool, .packages, build, etc.)

# TODO: IDE-Dateien ausschließen
# (.idea, .vscode, etc.)

# TODO: Git-Dateien ausschließen

# TODO: Test-Dateien ausschließen

# TODO: Dokumentation ausschließen

# TODO: Lokale Konfiguration ausschließen
# (.env, docker-compose.override.yml, etc.)
```

#### 5.4.1.5 Aufgabe 4: Docker Compose Setup (25 min)

Erstelle ein docker-compose.yml für die komplette Entwicklungsumgebung.

```
# docker-compose.yml

services:
  # TODO: API Service
```

```
api:
  build:
    context: .
    dockerfile: Dockerfile.optimized
  ports:
    # TODO: Port-Mapping
  environment:
    # TODO: Environment Variables
    # PORT, DATABASE_URL, REDIS_URL, JWT_SECRET
  depends_on:
    # TODO: Abhängigkeiten mit health condition
  restart: unless-stopped

# TODO: PostgreSQL Service
db:
  image: postgres:15-alpine
  ports:
    # TODO: Port-Mapping
  environment:
    # TODO: Postgres Environment
  volumes:
    # TODO: Daten-Volume
    # TODO: Init-Script mounten
  healthcheck:
    # TODO: pg_isready Check

# TODO: Redis Service
redis:
  image: redis:7-alpine
  ports:
    # TODO: Port-Mapping
  volumes:
    # TODO: Daten-Volume

# Optional: Adminer für DB-Management
adminer:
  image: adminer
  ports:
    - "8081:8080"
  depends_on:
    - db

# TODO: Volumes definieren
volumes:
  ???
```

**Test:**

```
docker compose up -d
docker compose ps
docker compose logs api
```

```
curl http://localhost:8080/health
```

#### 5.4.1.6 Aufgabe 5: Init-Script für Datenbank (10 min)

Erstelle ein SQL-Script für die Datenbank-Initialisierung.

```
-- init.sql

-- TODO: Datenbank erstellen (falls nicht existiert)
-- CREATE DATABASE IF NOT EXISTS ...

-- TODO: Users-Tabelle erstellen
CREATE TABLE IF NOT EXISTS users (
    -- TODO: Spalten definieren
);

-- TODO: Weitere Tabellen erstellen

-- TODO: Test-Daten einfügen (optional, nur für Entwicklung)
INSERT INTO users (email, password_hash, name)
VALUES ('admin@example.com', '$2a$12$...', 'Admin')
ON CONFLICT DO NOTHING;
```

#### 5.4.1.7 Aufgabe 6: Environment-Konfiguration (15 min)

Erstelle Konfigurationsdateien für verschiedene Umgebungen.

```
# .env.example (committen, als Vorlage)

PORT=8080
LOG_LEVEL=info
LOG_FORMAT=json

# Database
DATABASE_URL=postgres://user:pass@host:5432/db
DATABASE_POOL_SIZE=10

# Redis
REDIS_URL=redis://host:6379

# Auth
JWT_SECRET=change-me-in-production
JWT_ACCESS_TOKEN_DURATION=15m
JWT_REFRESH_TOKEN_DURATION=7d

# External Services
SMTP_HOST=
SMTP_PORT=
SMTP_USER=
SMTP_PASS=
```

```
# .env (NICHT committen!)
# Kopiere .env.example und passe an

PORT=8080
DATABASE_URL=postgres://postgres:postgres@db:5432/myapp
REDIS_URL=redis://redis:6379
JWT_SECRET=dev-secret-for-local-development
```

```
# docker-compose.override.yml (für lokale Entwicklung)

services:
  api:
    # TODO: Entwicklungs-spezifische Konfiguration
    # - Source Code mounten für Hot-Reload
    # - Debug Log Level
    # - Build Target auf 'build' Stage setzen
```

#### 5.4.1.8 Aufgabe 7: Fly.io Deployment (20 min)

Konfiguriere Deployment auf Fly.io.

```
# fly.toml

# TODO: App-Name
app = "???"

# TODO: Region
primary_region = "???"

[build]
  # TODO: Dockerfile referenzieren

[env]
  # TODO: Nicht-sensitive Environment Variables

[http_service]
  # TODO: Port konfigurieren
  # TODO: HTTPS erzwingen
  # TODO: Auto-Scaling

[[services]]
  # TODO: TCP Service konfigurieren

[[services.ports]]
  # TODO: HTTP Port

[[services.ports]]
  # TODO: HTTPS Port

[[services.http_checks]]
  # TODO: Health Check konfigurieren
```

```
# Deployment-Schritte dokumentieren

# 1. Fly CLI installieren
# TODO: Befehl

# 2. Login
# TODO: Befehl

# 3. App erstellen
# TODO: Befehl

# 4. PostgreSQL erstellen und verbinden
# TODO: Befehle

# 5. Secrets setzen
# TODO: Befehle

# 6. Deployen
# TODO: Befehl

# 7. Logs prüfen
# TODO: Befehl
```

#### 5.4.1.9 Aufgabe 8: GitHub Actions CI/CD (20 min)

Erstelle eine GitHub Actions Pipeline.

```
# .github/workflows/ci.yml

name: CI/CD

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  # TODO: Test Job
  test:
    runs-on: ubuntu-latest
    steps:
      # TODO: Checkout
      # TODO: Dart Setup
      # TODO: Dependencies installieren
      # TODO: Analyze
      # TODO: Tests

  # TODO: Build Job
  build:
    needs: test
```

```
runs-on: ubuntu-latest
steps:
  # TODO: Checkout
  # TODO: Docker Login (zu GitHub Container Registry)
  # TODO: Docker Build & Push

# TODO: Deploy Job (nur auf main)
deploy:
  needs: build
  runs-on: ubuntu-latest
  if: github.ref == 'refs/heads/main'
  steps:
    # TODO: Checkout
    # TODO: Fly CLI Setup
    # TODO: Deploy
```

#### 5.4.1.10 Bonus: Makefile (Optional)

```
# Makefile

.PHONY: dev build test deploy

# Lokale Entwicklung starten
dev:
  docker compose up -d
  docker compose logs -f api

# Image bauen
build:
  docker build -t my-api:latest -f Dockerfile.optimized .

# Tests ausführen
test:
  dart test

# Linting
lint:
  dart analyze

# Deployment
deploy:
  flyctl deploy

# Cleanup
clean:
  docker compose down -v
  docker system prune -f
```

#### 5.4.1.11 Testen

Lokale Entwicklung

```
# Compose starten
docker compose up -d

# Status prüfen
docker compose ps

# API testen
curl http://localhost:8080/health
curl http://localhost:8080/api/hello

# Logs
docker compose logs -f api

# In Container shell
docker compose exec api sh

# Stoppen
docker compose down
```

Image-Optimierung prüfen

```
# Größen vergleichen
docker images | grep my-api

# Layer inspizieren
docker history my-api:optimized

# Security Scan (optional)
docker scout cves my-api:optimized
```

#### 5.4.1.12 Abgabe-Checkliste

- ☐ Einfaches Dockerfile funktioniert
- ☐ Multi-Stage Dockerfile mit kleinem Image
- ☐ .dockerignore vorhanden
- ☐ Docker Compose mit API, DB, Redis
- ☐ Volumes für Persistenz
- ☐ Health Checks konfiguriert
- ☐ Init-Script für Datenbank
- ☐ Environment-Konfiguration (.env.example)
- ☐ fly.toml für Cloud-Deployment
- ☐ GitHub Actions Pipeline
- ☐ README mit Deployment-Anleitung

## 5.4.2 Lösung

### 5.4.2.1 Aufgabe 1: Einfaches Dockerfile

```
# Dockerfile

FROM dart:stable

WORKDIR /app

# Dependencies zuerst (Cache-Optimierung)
COPY pubspec.yaml pubspec.lock ./
RUN dart pub get

# Code kopieren
COPY . .

# Kompilieren
RUN dart compile exe bin/server.dart -o bin/server

# Port
EXPOSE 8080

# Start
CMD ["/bin/server"]
```

### 5.4.2.2 Aufgabe 2: Multi-Stage Build

```
# Dockerfile.optimized

# ===== BUILD STAGE =====
FROM dart:stable AS build

WORKDIR /app

# Dependencies cachen
COPY pubspec.yaml pubspec.lock ./
RUN dart pub get

# Code kopieren
COPY . .

# Dependencies erneut (für AOT)
RUN dart pub get --offline

# Kompilieren
RUN dart compile exe bin/server.dart -o bin/server

# ===== RUNTIME STAGE =====
FROM debian:bookworm-slim
```

```
# CA-Zertifikate für HTTPS
RUN apt-get update && apt-get install -y --no-install-recommends \
    ca-certificates \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Non-root User
RUN useradd -r -u 1001 -s /sbin/nologin appuser
USER appuser

WORKDIR /app

# Nur Binary kopieren
COPY --from=build /app/bin/server /app/bin/server

# Environment
ENV PORT=8080

# Port
EXPOSE 8080

# Health Check
HEALTHCHECK --interval=30s --timeout=3s --start-period=10s --retries=3 \
    CMD curl -f http://localhost:8080/health || exit 1

# Start
ENTRYPOINT ["/app/bin/server"]
```

#### 5.4.2.3 Aufgabe 3: .dockerignore

```
# .dockerignore

# Dart/Pub
.dart_tool/
.packages
build/
.pub-cache/
pubspec.lock

# IDE
.idea/
.vscode/
*.iml
*.ipr
*.iws

# Git
.git/
.gitignore
.gitattributes
```

```
# Tests
test/
coverage/
.test_runner.yaml

# Dokumentation
*.md
!README.md
LICENSE
docs/
CHANGELOG

# Lokale Konfiguration
.env
.env.local
.env.*.local
docker-compose.override.yml
*.local.yml

# CI/CD
.github/
.gitlab-ci.yml
Makefile

# Misc
*.log
*.tmp
.DS_Store
Thumbs.db
```

#### 5.4.2.4 Aufgabe 4: Docker Compose Setup

```
# docker-compose.yml

services:
  api:
    build:
      context: .
      dockerfile: Dockerfile.optimized
    ports:
      - "8080:8080"
    environment:
      - PORT=8080
      - DATABASE_URL=postgres://postgres:postgres@db:5432/myapp
      - REDIS_URL=redis://redis:6379
      - JWT_SECRET=${JWT_SECRET:-dev-secret-change-in-prod}
      - LOG_LEVEL=${LOG_LEVEL:-info}
    depends_on:
      db:
```

```
    condition: service_healthy
  redis:
    condition: service_started
  restart: unless-stopped
  networks:
    - app-network

db:
  image: postgres:15-alpine
  ports:
    - "5432:5432"
  environment:
    POSTGRES_USER: postgres
    POSTGRES_PASSWORD: postgres
    POSTGRES_DB: myapp
  volumes:
    - postgres_data:/var/lib/postgresql/data
    - ./init.sql:/docker-entrypoint-initdb.d/init.sql:ro
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U postgres -d myapp"]
    interval: 5s
    timeout: 5s
    retries: 5
    start_period: 10s
  networks:
    - app-network

redis:
  image: redis:7-alpine
  ports:
    - "6379:6379"
  volumes:
    - redis_data:/data
  command: redis-server --appendonly yes
  networks:
    - app-network

adminer:
  image: adminer
  ports:
    - "8081:8080"
  depends_on:
    - db
  networks:
    - app-network

volumes:
  postgres_data:
  redis_data:

networks:
```

```
app-network:
  driver: bridge
```

```
# docker-compose.override.yml (für lokale Entwicklung)
```

```
services:
  api:
    build:
      target: build
    command: dart run bin/server.dart
    volumes:
      - ./app:delegated
      - dart_packages:/app/.dart_tool
    environment:
      - LOG_LEVEL=debug
      - LOG_FORMAT=text

volumes:
  dart_packages:
```

#### 5.4.2.5 Aufgabe 5: Init-Script

```
-- init.sql

-- Extensions
CREATE EXTENSION IF NOT EXISTS "uuid-osspl";

-- Users Tabelle
CREATE TABLE IF NOT EXISTS users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  name VARCHAR(100),
  role VARCHAR(50) DEFAULT 'user',
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Refresh Tokens Tabelle
CREATE TABLE IF NOT EXISTS refresh_tokens (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
  token_hash VARCHAR(255) NOT NULL,
  expires_at TIMESTAMP NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  revoked_at TIMESTAMP
);

-- Index für häufige Queries
```

```
CREATE INDEX IF NOT EXISTS idx_users_email ON users(email);
CREATE INDEX IF NOT EXISTS idx_refresh_tokens_user ON refresh_tokens(user_id);
CREATE INDEX IF NOT EXISTS idx_refresh_tokens_hash ON refresh_tokens(token_hash);

-- Updated At Trigger
CREATE OR REPLACE FUNCTION update_updated_at()
RETURNS TRIGGER AS $$
BEGIN
    NEW.updated_at = CURRENT_TIMESTAMP;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS users_updated_at ON users;
CREATE TRIGGER users_updated_at
    BEFORE UPDATE ON users
    FOR EACH ROW
    EXECUTE FUNCTION update_updated_at();

-- Test-Admin (nur für Entwicklung)
-- Password: admin123 (bcrypt hash)
INSERT INTO users (email, password_hash, name, role)
VALUES (
    'admin@example.com',
    '$2a$12$LQv3c1yqBWHxkd0LHAKC0Yz6TtxMQJqhN8/X4.nqYBFqJWvyYGis',
    'Admin',
    'admin'
) ON CONFLICT (email) DO NOTHING;
```

#### 5.4.2.6 Aufgabe 6: Environment-Konfiguration

```
# .env.example

# Server
PORT=8080
HOST=0.0.0.0
LOG_LEVEL=info
LOG_FORMAT=json

# Database
DATABASE_URL=postgres://user:password@host:5432/database
DATABASE_POOL_SIZE=10
DATABASE_TIMEOUT=30

# Redis
REDIS_URL=redis://host:6379
REDIS_PREFIX=myapp:

# Authentication
JWT_SECRET=change-this-to-a-secure-random-string
```

```
JWT_ACCESS_TOKEN_DURATION=15m
JWT_REFRESH_TOKEN_DURATION=7d

# Email (optional)
SMTP_HOST=smtp.example.com
SMTP_PORT=587
SMTP_USER=
SMTP_PASS=
SMTP_FROM=noreply@example.com

# External APIs (optional)
SENTRY_DSN=

# .env (lokale Entwicklung - NICHT committen!)

PORT=8080
HOST=0.0.0.0
LOG_LEVEL=debug
LOG_FORMAT=text

DATABASE_URL=postgres://postgres:postgres@db:5432/myapp
DATABASE_POOL_SIZE=5

REDIS_URL=redis://redis:6379

JWT_SECRET=dev-secret-for-local-development-only
JWT_ACCESS_TOKEN_DURATION=1h
JWT_REFRESH_TOKEN_DURATION=30d
```

#### 5.4.2.7 Aufgabe 7: Fly.io Deployment

```
# fly.toml

app = "my-dart-api"
primary_region = "fra"

[build]
  dockerfile = "Dockerfile.optimized"

[env]
  PORT = "8080"
  LOG_LEVEL = "info"
  LOG_FORMAT = "json"

[http_service]
  internal_port = 8080
  force_https = true
  auto_stop_machines = true
  auto_start_machines = true
  min_machines_running = 1
```

```
[http_service.concurrency]
  type = "requests"
  hard_limit = 250
  soft_limit = 200

[[services]]
  protocol = "tcp"
  internal_port = 8080

[[services.ports]]
  port = 80
  handlers = ["http"]

[[services.ports]]
  port = 443
  handlers = ["tls", "http"]

[[services.http_checks]]
  interval = "15s"
  timeout = "5s"
  grace_period = "10s"
  path = "/health"
  method = "GET"

[[vm]]
  cpu_kind = "shared"
  cpus = 1
  memory_mb = 512
```

#### Deployment-Befehle

```
# 1. Fly CLI installieren
curl -L https://fly.io/install.sh | sh

# Oder via npm
npm install -g flyctl

# 2. Login
fly auth login

# 3. App erstellen (interaktiv)
fly launch

# Oder manuell
fly apps create my-dart-api

# 4. PostgreSQL erstellen
fly postgres create --name my-dart-api-db

# PostgreSQL mit App verbinden
```

```
fly postgres attach my-dart-api-db

# 5. Secrets setzen
fly secrets set JWT_SECRET="$(openssl rand -base64 32)"
fly secrets set REDIS_URL="redis://..." # Falls Redis benötigt

# Secrets auflisten
fly secrets list

# 6. Deployen
fly deploy

# Mit bestimmtem Image
fly deploy --image ghcr.io/user/repo:tag

# 7. Status prüfen
fly status
fly logs
fly logs --app my-dart-api

# 8. Skalieren
fly scale count 2 # 2 Instanzen
fly scale vm shared-cpu-1x # VM-Größe

# 9. SSH in Container
fly ssh console

# 10. Rollback
fly releases list
fly deploy --image <previous-image>
```

#### 5.4.2.8 Aufgabe 8: GitHub Actions CI/CD

```
# .github/workflows/ci.yml

name: CI/CD

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${github.repository}
  FLY_API_TOKEN: ${secrets.FLY_API_TOKEN}

jobs:
  test:
```

```
name: Test
runs-on: ubuntu-latest

services:
  postgres:
    image: postgres:15
    env:
      POSTGRES_USER: test
      POSTGRES_PASSWORD: test
      POSTGRES_DB: test
    ports:
      - 5432:5432
    options: >-
      --health-cmd pg_isready
      --health-interval 10s
      --health-timeout 5s
      --health-retries 5

steps:
  - name: Checkout
    uses: actions/checkout@v4

  - name: Setup Dart
    uses: dart-lang/setup-dart@v1
    with:
      sdk: stable

  - name: Install dependencies
    run: dart pub get

  - name: Analyze
    run: dart analyze --fatal-infos

  - name: Format check
    run: dart format --set-exit-if-changed .

  - name: Run tests
    run: dart test --coverage=coverage
    env:
      DATABASE_URL: postgres://test:test@localhost:5432/test

  - name: Upload coverage
    uses: codecov/codecov-action@v3
    with:
      files: coverage/lcov.info

build:
  name: Build
  needs: test
  runs-on: ubuntu-latest
  permissions:
```

```
  contents: read
  packages: write

outputs:
  image: ${ steps.meta.outputs.tags }

steps:
  - name: Checkout
    uses: actions/checkout@v4

  - name: Set up Docker Buildx
    uses: docker/setup-buildx-action@v3

  - name: Log in to Container Registry
    uses: docker/login-action@v3
    with:
      registry: ${ env.REGISTRY }
      username: ${ github.actor }
      password: ${ secrets.GITHUB_TOKEN }

  - name: Extract metadata
    id: meta
    uses: docker/metadata-action@v5
    with:
      images: ${ env.REGISTRY }/${ env.IMAGE_NAME }
      tags: |
        type=sha,prefix=
        type=ref,event=branch
        type=semver,pattern={{version}}
        type=raw,value=latest,enable={{is_default_branch}}

  - name: Build and push
    uses: docker/build-push-action@v5
    with:
      context: .
      file: Dockerfile.optimized
      push: ${ github.event_name != 'pull_request' }
      tags: ${ steps.meta.outputs.tags }
      labels: ${ steps.meta.outputs.labels }
      cache-from: type=gha
      cache-to: type=gha,mode=max

deploy:
  name: Deploy
  needs: build
  runs-on: ubuntu-latest
  if: github.ref == 'refs/heads/main' && github.event_name == 'push'

environment:
  name: production
  url: https://my-dart-api.fly.dev
```

```
steps:
  - name: Checkout
    uses: actions/checkout@v4

  - name: Setup Fly
    uses: superfly/flyctl-actions/setup-flyctl@master

  - name: Deploy to Fly.io
    run: flyctl deploy --remote-only
    env:
      FLY_API_TOKEN: ${ secrets.FLY_API_TOKEN }

  - name: Smoke Test
    run: |
      sleep 10
      curl -f https://my-dart-api.fly.dev/health || exit 1
```

#### 5.4.2.9 Bonus: Makefile

```
# Makefile

.PHONY: help dev build test lint deploy clean

# Default target
help:
  @echo "Available commands:"
  @echo "  make dev      - Start development environment"
  @echo "  make build    - Build Docker image"
  @echo "  make test     - Run tests"
  @echo "  make lint     - Run linter"
  @echo "  make deploy   - Deploy to Fly.io"
  @echo "  make clean    - Clean up"

# Entwicklung
dev:
  docker compose up -d
  docker compose logs -f api

dev-down:
  docker compose down

dev-restart:
  docker compose restart api

# Build
build:
  docker build -t my-api:latest -f Dockerfile.optimized .

build-no-cache:
```

```
    docker build --no-cache -t my-api:latest -f Dockerfile.optimized .

# Tests
test:
    dart test

test-coverage:
    dart test --coverage=coverage
    dart pub global run coverage:format_coverage \
        --lcov --in=coverage --out=coverage/lcov.info

# Linting
lint:
    dart analyze
    dart format --set-exit-if-changed .

lint-fix:
    dart format .

# Deployment
deploy:
    flyctl deploy

deploy-staging:
    flyctl deploy --config fly.staging.toml

# Secrets
secrets:
    flyctl secrets list

set-secret:
    @read -p "Secret name: " name; \
    read -p "Secret value: " value; \
    flyctl secrets set $$name=$$value

# Logs
logs:
    flyctl logs

logs-follow:
    flyctl logs -f

# Database
db-shell:
    docker compose exec db psql -U postgres -d myapp

db-migrate:
    dart run bin/migrate.dart

# Cleanup
clean:
```

```
docker compose down -v
docker system prune -f
rm -rf coverage/ .dart_tool/ build/

# CI lokall testen
ci:
  act -j test
```

#### 5.4.2.10 Projektstruktur

```
my-api/
+-- .github/
|   +-- workflows/
|       +-- ci.yml
+-- bin/
|   +-- server.dart
|   +-- migrate.dart
+-- lib/
|   +-- ...
+-- test/
|   +-- ...
+-- .dockerignore
+-- .env.example
+-- .gitignore
+-- Dockerfile
+-- Dockerfile.optimized
+-- docker-compose.yml
+-- docker-compose.override.yml
+-- fly.toml
+-- init.sql
+-- Makefile
+-- pubspec.yaml
+-- README.md
```

#### 5.4.2.11 README.md Template

```
# My Dart API

#### Lokale Entwicklung

##### Voraussetzungen

- Docker & Docker Compose
- Dart SDK (für Tests ohne Docker)

##### Starten

```bash
# Umgebungsvariablen kopieren
cp .env.example .env
```

```
# Services starten
docker compose up -d

# Logs verfolgen
docker compose logs -f api
```

Endpoints

- API: <http://localhost:8080>
- Adminer (DB UI): <http://localhost:8081>
- Health: <http://localhost:8080/health>

#### 5.4.2.12 Tests

```
dart test
```

#### 5.4.2.13 Deployment

Deployment erfolgt automatisch bei Push auf `main` via GitHub Actions.

Manuelles Deployment:

```
flyctl deploy
```

#### 5.4.2.14 Umgebungsvariablen

Siehe `.env.example` für alle verfügbaren Variablen.

### Ressourcen

#### Offizielle Dokumentation

- [Docker Documentation](<https://docs.docker.com/>)
- [Docker Compose](<https://docs.docker.com/compose/>)
- [Fly.io Documentation](<https://fly.io/docs/>)
- [Railway Documentation](<https://docs.railway.app/>)
- [GitHub Actions](<https://docs.github.com/en/actions>)

#### Cheat Sheet: Dockerfile

```
```dockerfile
# Base Images
FROM dart:stable           # Volle Dart SDK (~800MB)
FROM dart:stable-sdk       # Nur SDK
FROM debian:bookworm-slim  # Minimales Debian (~80MB)
FROM alpine:3.18           # Noch kleiner (~5MB)
FROM scratch               # Leer (nur für statische Binaries)

# Arbeitsverzeichnis
WORKDIR /app
```

```

# Dateien kopieren
COPY . .                # Alles
COPY pubspec.* ./        # Nur pubspec
COPY --from=build /app/bin/server . # Aus anderem Stage

# Befehle ausführen
RUN dart pub get
RUN dart compile exe bin/server.dart -o server

# User
RUN useradd -r -u 1001 appuser
USER appuser

# Environment
ENV PORT=8080
EXPOSE 8080

# Health Check
HEALTHCHECK --interval=30s --timeout=3s \
  CMD curl -f http://localhost:8080/health || exit 1

# Start
CMD ["/server"]
ENTRYPOINT ["/server"] # Unveränderlich

```

#### 5.4.2.15 Cheat Sheet: Multi-Stage Build

```

# Stage 1: Build
FROM dart:stable AS build
WORKDIR /app
COPY . .
RUN dart compile exe bin/server.dart -o server

# Stage 2: Runtime (nur Binary)
FROM debian:bookworm-slim
COPY --from=build /app/server /app/server
CMD ["/app/server"]

# Ergebnis: ~20MB statt ~800MB

```

#### 5.4.2.16 Cheat Sheet: Docker Compose

```

services:
  app:
    build: .                # Dockerfile im aktuellen Verzeichnis
    build:
      context: .
      dockerfile: Dockerfile.prod
    image: myapp:latest     # Fertiges Image verwenden

```

```
ports:
  - "8080:8080"          # host:container
environment:
  - KEY=value
env_file:
  - .env
volumes:
  - ./data:/app/data      # Bind Mount
  - myvolume:/app/storage # Named Volume
depends_on:
  db:
    condition: service_healthy
restart: unless-stopped
networks:
  - backend

volumes:
  myvolume:

networks:
  backend:
```

#### 5.4.2.17 Cheat Sheet: Docker Befehle

```
# Images
docker build -t name:tag .
docker build -f Dockerfile.prod -t name:prod .
docker images
docker rmi image:tag

# Container
docker run -d -p 8080:8080 --name myapp image:tag
docker ps
docker ps -a
docker logs -f myapp
docker exec -it myapp sh
docker stop myapp
docker rm myapp

# Compose
docker compose up -d
docker compose down
docker compose logs -f
docker compose exec app sh
docker compose build --no-cache
docker compose ps

# Cleanup
docker system prune -f
docker volume prune -f
```

```
docker image prune -a
```

#### 5.4.2.18 Cheat Sheet: Fly.io

```
# Setup
fly auth login
fly launch           # Neue App erstellen
fly deploy           # Deployen

# Status
fly status
fly logs
fly logs -f

# Secrets
fly secrets set KEY=value
fly secrets list

# Database
fly postgres create
fly postgres attach app-db

# Skalierung
fly scale count 3      # 3 Instanzen
fly scale vm shared-cpu-1x  # VM-Größe
fly scale memory 512     # RAM

# SSH
fly ssh console
fly ssh console -C "ls -la"

# Rollback
fly releases list
fly deploy --image registry/app:v1
```

#### 5.4.2.19 Cheat Sheet: fly.toml

```
app = "my-app"
primary_region = "fra"

[build]
  dockerfile = "Dockerfile"

[env]
  PORT = "8080"

[http_service]
  internal_port = 8080
  force_https = true
```

```
auto_stop_machines = true
auto_start_machines = true
min_machines_running = 1

[[services]]
protocol = "tcp"
internal_port = 8080

[[services.ports]]
port = 443
handlers = ["tls", "http"]

[[services.http_checks]]
interval = "15s"
timeout = "5s"
path = "/health"
```

#### 5.4.2.20 Cheat Sheet: GitHub Actions

```
name: CI/CD

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: dart-lang/setup-dart@v1
      - run: dart pub get
      - run: dart analyze
      - run: dart test

  build:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${ github.actor }
          password: ${ secrets.GITHUB_TOKEN }
      - uses: docker/build-push-action@v5
        with:
          push: true
```

```

    tags: ghcr.io/${{ github.repository }}:latest

deploy:
  needs: build
  if: github.ref == 'refs/heads/main'
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v4
    - uses: superfly/flyctl-actions/setup-flyctl@master
    - run: flyctl deploy --remote-only
  env:
    FLY_API_TOKEN: ${ secrets.FLY_API_TOKEN }

```

#### 5.4.2.21 Best Practices

##### DO

1. **Multi-Stage Builds** - Kleine Images
2. **Non-root User** - Security
3. **Layer-Optimierung** - COPY pubspec.\* vor COPY .
4. **Health Checks** - Für Orchestrierung
5. **.dockerignore** - Unnötiges ausschließen
6. **Secrets extern** - Nicht im Image
7. **Immutable Tags** - SHA statt :latest in Prod
8. **Graceful Shutdown** - SIGTERM behandeln

##### DON'T

1. **Secrets im Dockerfile** - ENV SECRET=xxx
2. **:latest in Produktion** - Nicht reproduzierbar
3. **Root User** - Security-Risiko
4. **Große Base Images** - Langsamer, mehr CVEs
5. **Volumes für Code** - Nur für persistente Daten
6. **docker-compose.override.yml committen** - Lokal only

#### 5.4.2.22 Troubleshooting

```

# Build-Probleme
docker build --no-cache -t app .
docker build --progress=plain -t app .

# Container-Probleme
docker logs myapp
docker exec -it myapp sh
docker inspect myapp

# Netzwerk-Probleme
docker network ls
docker network inspect bridge

# Speicher-Probleme

```

```
docker system df
docker system prune -a
```

#### 5.4.2.23 Security Checklist

- [ ] Non-root User im Container
- [ ] Minimales Base Image (debian-slim, alpine)
- [ ] Keine Secrets im Image
- [ ] HEALTHCHECK definiert
- [ ] Nur notwendige Ports exposed
- [ ] Read-only Filesystem wo möglich
- [ ] Security Scan durchgeführt (docker scout, trivy)
- [ ] Keine hardcoded Credentials

## 5.5 Einheit 9.5: Abschlussprojekt Backend

### 5.5.0.1 Projektübersicht

Task Management API

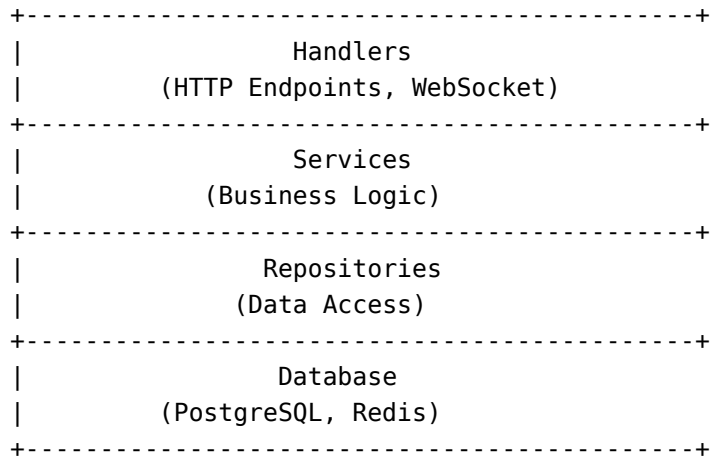
Du entwickelst eine vollständige Task-Management-API, die alle gelernten Konzepte aus dem Backend-Teil zusammenführt.

Features

Task Management API		
□ Benutzer	□ Projekte	
+++ Registrierung	+++ CRUD	
+++ Login/Logout	+++ Mitglieder	
+++ JWT Auth	+++ Berechtigungen	
+++ Profil		
□ Tasks	□ Real-time	
+++ CRUD	+++ WebSocket	
+++ Zuweisung	+++ Live-Updates	
+++ Status	+++ Notifications	
+++ Priorität		
+++ Kommentare		
□ Sicherheit	□ Infrastruktur	
+++ Rate Limiting	+++ PostgreSQL	
+++ Input Validation	+++ Redis Caching	
+++ CORS	+++ Docker	
+++ Security Headers	+++ Health Checks	

### 5.5.0.2 Architektur

#### Layered Architecture



#### Projektstruktur

```

task_api/
+-- bin/
|   +-- server.dart           # Hauptserver
|   +-- migrate.dart         # DB Migrationen
|   +-- seed.dart            # Test-Daten
+-- lib/
|   +-- config/
|   |   +-- config.dart       # Konfiguration
|   |   +-- dependencies.dart # DI Container
|   +-- models/
|   |   +-- user.dart
|   |   +-- project.dart
|   |   +-- task.dart
|   |   +-- comment.dart
|   +-- repositories/
|   |   +-- user_repository.dart
|   |   +-- project_repository.dart
|   |   +-- task_repository.dart
|   |   +-- comment_repository.dart
|   +-- services/
|   |   +-- auth_service.dart
|   |   +-- project_service.dart
|   |   +-- task_service.dart
|   |   +-- notification_service.dart
|   +-- handlers/
|   |   +-- auth_handler.dart
|   |   +-- project_handler.dart
|   |   +-- task_handler.dart
|   |   +-- websocket_handler.dart
|   +-- middleware/
|   |   +-- auth_middleware.dart
|   |   +-- rate_limit_middleware.dart
|   |   +-- cors_middleware.dart

```

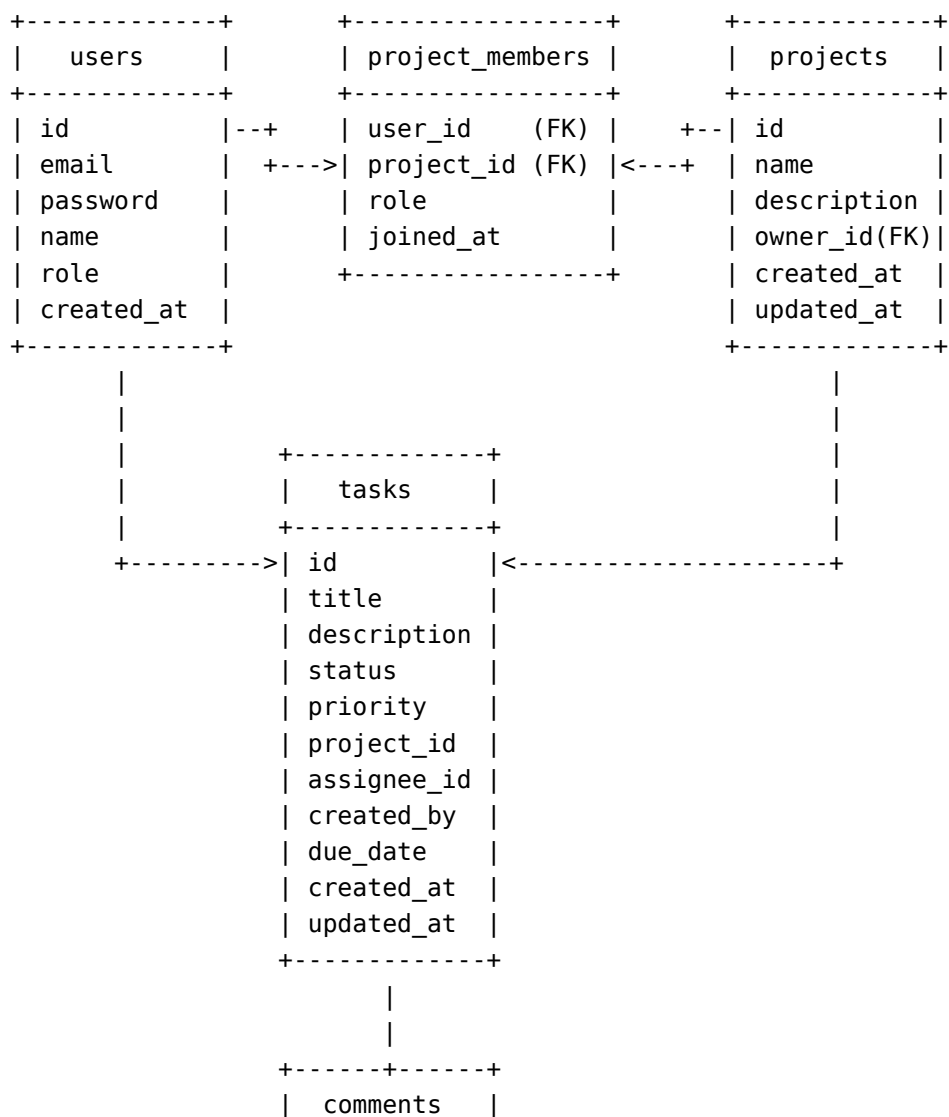
```

| | +-- error_middleware.dart
| +-- utils/
| | +-- logger.dart
| | +-- validator.dart
| | +-- jwt_utils.dart
| +-- app.dart          # App zusammenbauen
+-- test/
| +-- unit/
| +-- integration/
| +-- fixtures/
+-- docker/
| +-- Dockerfile
| +-- docker-compose.yml
+-- .env.example
+-- pubspec.yaml
+-- README.md

```

### 5.5.0.3 Datenmodell

Entity-Relationship-Diagramm



```

+-----+
| id      |
| task_id(FK) |
| user_id(FK) |
| content  |
| created_at |
+-----+

```

SQL Schema

```

-- users
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  name VARCHAR(100) NOT NULL,
  role VARCHAR(50) DEFAULT 'user',
  avatar_url VARCHAR(500),
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- projects
CREATE TABLE projects (
  id SERIAL PRIMARY KEY,
  name VARCHAR(200) NOT NULL,
  description TEXT,
  owner_id INTEGER REFERENCES users(id),
  is_archived BOOLEAN DEFAULT false,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- project_members
CREATE TABLE project_members (
  id SERIAL PRIMARY KEY,
  project_id INTEGER REFERENCES projects(id) ON DELETE CASCADE,
  user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
  role VARCHAR(50) DEFAULT 'member', -- owner, admin, member, viewer
  joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(project_id, user_id)
);

-- tasks
CREATE TABLE tasks (
  id SERIAL PRIMARY KEY,
  title VARCHAR(300) NOT NULL,
  description TEXT,
  status VARCHAR(50) DEFAULT 'todo', -- todo, in_progress, review, done
  priority VARCHAR(50) DEFAULT 'medium', -- low, medium, high, urgent
  project_id INTEGER REFERENCES projects(id) ON DELETE CASCADE,

```

```

    assignee_id INTEGER REFERENCES users(id),
    created_by INTEGER REFERENCES users(id),
    due_date TIMESTAMP,
    completed_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- comments
CREATE TABLE comments (
    id SERIAL PRIMARY KEY,
    task_id INTEGER REFERENCES tasks(id) ON DELETE CASCADE,
    user_id INTEGER REFERENCES users(id),
    content TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- refresh_tokens
CREATE TABLE refresh_tokens (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
    token_hash VARCHAR(255) NOT NULL,
    expires_at TIMESTAMP NOT NULL,
    revoked_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

#### 5.5.0.4 API Endpoints

##### Authentication

Method	Endpoint	Beschreibung
POST	/api/auth/register	Registrierung
POST	/api/auth/login	Login
POST	/api/auth/refresh	Token erneuern
POST	/api/auth/logout	Logout
GET	/api/auth/me	Aktueller User

##### Users

Method	Endpoint	Beschreibung
GET	/api/users	Alle User (Admin)
GET	/api/users/:id	User Details
PUT	/api/users/:id	User aktualisieren
DELETE	/api/users/:id	User löschen (Admin)

##### Projects

Method	Endpoint	Beschreibung
GET	/api/projects	Meine Projekte
POST	/api/projects	Projekt erstellen
GET	/api/projects/:id	Projekt Details
PUT	/api/projects/:id	Projekt aktualisieren
DELETE	/api/projects/:id	Projekt löschen
GET	/api/projects/:id/members	Mitglieder
POST	/api/projects/:id/members	Mitglied hinzufügen
DELETE	/api/projects/:id/members/:userId	Mitglied entfernen

#### Tasks

Method	Endpoint	Beschreibung
GET	/api/projects/:id/tasks	Tasks eines Projekts
POST	/api/projects/:id/tasks	Task erstellen
GET	/api/tasks/:id	Task Details
PUT	/api/tasks/:id	Task aktualisieren
PATCH	/api/tasks/:id/status	Status ändern
PATCH	/api/tasks/:id/assign	Zuweisen
DELETE	/api/tasks/:id	Task löschen

#### Comments

Method	Endpoint	Beschreibung
GET	/api/tasks/:id/comments	Kommentare
POST	/api/tasks/:id/comments	Kommentar hinzufügen
PUT	/api/comments/:id	Kommentar bearbeiten
DELETE	/api/comments/:id	Kommentar löschen

#### System

Method	Endpoint	Beschreibung
GET	/health	Health Check
GET	/health/ready	Readiness Check
GET	/metrics	Prometheus Metrics
WS	/ws	WebSocket Connection

#### 5.5.0.5 Implementierungs-Roadmap

##### Phase 1: Grundgerüst (Tag 1)

- [ ] Projektstruktur erstellen
- [ ] pubspec.yaml konfigurieren
- [ ] Basis-Konfiguration (Config-Klasse)
- [ ] Logger einrichten
- [ ] Datenbank-Connection
- [ ] Basis-Middleware (CORS, Logging)
- [ ] Health Check Endpoint

Phase 2: Authentifizierung (Tag 1-2)

- [ ] User Model
- [ ] User Repository
- [ ] Password Service (bcrypt)
- [ ] JWT Service
- [ ] Auth Service
- [ ] Auth Handler (Register, Login, Logout)
- [ ] Auth Middleware
- [ ] Refresh Token Flow

Phase 3: Projekte & Tasks (Tag 2-3)

- [ ] Project Model & Repository
- [ ] Task Model & Repository
- [ ] Comment Model & Repository
- [ ] Project Service mit Berechtigungen
- [ ] Task Service
- [ ] CRUD Handler für Projekte
- [ ] CRUD Handler für Tasks
- [ ] Kommentar-Funktionalität

Phase 4: Real-time (Tag 3)

- [ ] WebSocket Handler
- [ ] Connection Manager
- [ ] Notification Service
- [ ] Live Updates bei Task-Änderungen
- [ ] Präsenz-Anzeige (optional)

Phase 5: Produktion (Tag 4)

- [ ] Rate Limiting
- [ ] Input Validation
- [ ] Security Headers
- [ ] Redis Caching
- [ ] Docker Setup
- [ ] Tests schreiben
- [ ] Dokumentation

#### 5.5.0.6 Technologie-Stack

Packages

```
dependencies:
  # Server
  shelf: ^1.4.0
  shelf_router: ^1.1.0
  shelf_web_socket: ^2.0.0

  # Database
  postgres: ^3.0.0

  # Cache
  redis: ^3.0.0
```

```
# Auth
bcrypt: ^1.1.0
dart_jsonwebtoken: ^2.8.0

# Utilities
uuid: ^4.0.0
dotenv: ^4.1.0

dev_dependencies:
  test: ^1.24.0
  mocktail: ^1.0.0
```

#### Infrastruktur

- **PostgreSQL 15** - Primäre Datenbank
- **Redis 7** - Caching & Sessions
- **Docker** - Containerisierung
- **GitHub Actions** - CI/CD

#### 5.5.0.7 Bewertungskriterien

##### Funktionalität (40%)

- ☐ Vollständige Auth-Flows
- ☐ CRUD für alle Entitäten
- ☐ Korrekte Berechtigungen
- ☐ WebSocket-Updates
- ☐ Fehlerbehandlung

##### Code-Qualität (30%)

- ☐ Saubere Architektur
- ☐ Separation of Concerns
- ☐ Keine Code-Duplikation
- ☐ Aussagekräftige Benennung
- ☐ Dokumentation

##### Sicherheit (15%)

- ☐ Sichere Passwort-Speicherung
- ☐ JWT korrekt implementiert
- ☐ Input Validation
- ☐ SQL Injection geschützt
- ☐ Rate Limiting

##### Infrastruktur (15%)

- ☐ Docker funktioniert
- ☐ Health Checks
- ☐ Logging strukturiert
- ☐ Metriken vorhanden
- ☐ Tests vorhanden

#### 5.5.0.8 Tipps

Starte klein

```
// Erst ein Endpoint, dann erweitern
router.get('/api/projects', (Request request) async {
    return Response.ok('[]');
});
```

Nutze Dependency Injection

```
class Dependencies {
    late final Connection db;
    late final UserRepository userRepo;
    late final AuthService authService;

    Future<void> init() async {
        db = await Connection.open(...);
        userRepo = UserRepository(db);
        authService = AuthService(userRepo, ...);
    }
}
```

Teste früh

```
test('login returns tokens', () async {
    final result = await authService.login('test@test.com', 'password');
    expect(result.accessToken, isEmpty);
});
```

Dokumentiere während du entwickelst

```
/// Erstellt einen neuen Task im angegebenen Projekt.
///
/// Erfordert Projekt-Mitgliedschaft.
/// Sendet WebSocket-Notification an alle Projekt-Mitglieder.
Future<Task> createTask(CreateTaskDto dto, int userId) async {
    // ...
}
```

### 5.5.0.9 Ressourcen

- Alle Lehrstoffe aus Block 5-9
- Shelf Dokumentation
- PostgreSQL Dokumentation
- JWT.io - Token debuggen
- Postman - API testen

## 5.5.1 Übung

### 5.5.1.1 Task Management API

Entwickle eine vollständige Task-Management-API mit allen Features aus dem Backend-Curriculum.

**Geschätzter Zeitaufwand:** 8-12 Stunden

### 5.5.1.2 Vorbereitung

#### 1. Projekt erstellen

```
mkdir task_api
cd task_api
dart create -t server-shelf .
```

#### 2. Dependencies

```
# pubspec.yaml
name: task_api
description: Task Management API
version: 1.0.0

environment:
  sdk: ^3.0.0

dependencies:
  shelf: ^1.4.0
  shelf_router: ^1.1.0
  shelf_web_socket: ^2.0.0
  postgres: ^3.0.0
  redis: ^3.0.0
  bcrypt: ^1.1.0
  dart_jsonwebtoken: ^2.8.0
  uuid: ^4.0.0
  dotenv: ^4.1.0

dev_dependencies:
  test: ^1.24.0
  mocktail: ^1.0.0
```

#### 3. Docker-Umgebung

```
# docker-compose.yml
services:
  db:
    image: postgres:15-alpine
    ports:
      - "5432:5432"
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
      POSTGRES_DB: taskapi
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./migrations:/docker-entrypoint-initdb.d:ro

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
```

```
volumes:
  postgres_data:
```

```
docker compose up -d
```

### 5.5.1.3 Phase 1: Grundgerüst

Aufgabe 1.1: Konfiguration (30 min)

```
// lib/config/config.dart

class AppConfig {
  final int port;
  final String databaseUrl;
  final String redisUrl;
  final String jwtSecret;
  final Duration jwtAccessDuration;
  final Duration jwtRefreshDuration;
  final LogLevel logLevel;

  AppConfig({
    required this.port,
    required this.databaseUrl,
    required this.redisUrl,
    required this.jwtSecret,
    this.jwtAccessDuration = const Duration(minutes: 15),
    this.jwtRefreshDuration = const Duration(days: 7),
    this.logLevel = LogLevel.info,
  });

  factory AppConfig.fromEnvironment() {
    // TODO: Aus Environment Variables laden
    // TODO: Defaults für Entwicklung
  }
}
```

Aufgabe 1.2: Logger (20 min)

```
// lib/utils/logger.dart

// TODO: Logger aus Einheit 9.3 implementieren
// - Log Level (debug, info, warning, error)
// - JSON-Format für Produktion
// - Request-ID Context
```

Aufgabe 1.3: Datenbank-Setup (30 min)

```
-- migrations/001_initial.sql

-- TODO: Schema erstellen
-- - users
```

```
-- - projects
-- - project_members
-- - tasks
-- - comments
-- - refresh_tokens

-- TODO: Indizes

-- TODO: Trigger für updated_at
```

Aufgabe 1.4: Health Check (20 min)

```
// lib/handlers/health_handler.dart

class HealthHandler {
  final Connection db;
  final RedisConnection? redis;

  // TODO: /health Endpoint
  // TODO: /health/ready Endpoint
  // TODO: /health/live Endpoint
}
```

#### 5.5.1.4 Phase 2: Authentifizierung

Aufgabe 2.1: User Model (20 min)

```
// lib/models/user.dart

class User {
  final int id;
  final String email;
  final String passwordHash;
  final String name;
  final String role;
  final String? avatarUrl;
  final bool isActive;
  final DateTime createdAt;
  final DateTime updatedAt;

  // TODO: Konstruktor
  // TODO: fromRow (DB Row)
  // TODO: toJson (ohne passwordHash!)
  // TODO: copyWith
}

class CreateUserDto {
  final String email;
  final String password;
  final String name;

  // TODO: Validation
}
```

```
}
```

Aufgabe 2.2: Password & JWT Service (30 min)

```
// lib/services/password_service.dart
// TODO: hash(), verify(), needsRehash()

// lib/services/jwt_service.dart
// TODO: generateAccessToken(), generateRefreshToken()
// TODO: verifyToken(), extractFromHeader()
```

Aufgabe 2.3: User Repository (30 min)

```
// lib/repositories/user_repository.dart

class UserRepository {
  final Connection db;

  // TODO: create(CreateUserDto)
  // TODO: findById(int id)
  // TODO: findByEmail(String email)
  // TODO: update(int id, UpdateUserDto)
  // TODO: delete(int id)
  // TODO: exists(String email)
}
```

Aufgabe 2.4: Auth Service (45 min)

```
// lib/services/auth_service.dart

class AuthService {
  // TODO: register(email, password, name) -> User
  // TODO: login(email, password) -> TokenPair
  // TODO: refreshToken(refreshToken) -> TokenPair
  // TODO: logout(refreshToken) -> void
  // TODO: getCurrentUser(accessToken) -> User
}
```

Aufgabe 2.5: Auth Handler (30 min)

```
// lib/handlers/auth_handler.dart

class AuthHandler {
  Router get router {
    final router = Router();

    router.post('/register', _register);
    router.post('/login', _login);
    router.post('/refresh', _refresh);
    router.post('/logout', _logout);
    router.get('/me', _me);

    return router;
  }
}
```

```
}

// TODO: Implementiere alle Handler
}
```

Aufgabe 2.6: Auth Middleware (20 min)

```
// lib/middleware/auth_middleware.dart

// TODO: authMiddleware - Token validieren, User in Context
// TODO: requireRole(String role) - Rollen-Check
// TODO: optionalAuth - Auth optional (für öffentliche Endpoints)
```

#### 5.5.1.5 Phase 3: Projekte & Tasks

Aufgabe 3.1: Project Model & Repository (30 min)

```
// lib/models/project.dart

class Project {
  final int id;
  final String name;
  final String? description;
  final int ownerId;
  final bool isArchived;
  final DateTime createdAt;
  final DateTime updatedAt;

  // TODO: Implementieren
}

// lib/repositories/project_repository.dart

class ProjectRepository {
  // TODO: create, findById, findByUser, update, delete
  // TODO: addMember, removeMember, getMembers
  // TODO: getUserRole(projectId, userId)
}
```

Aufgabe 3.2: Task Model & Repository (30 min)

```
// lib/models/task.dart

enum TaskStatus { todo, inProgress, review, done }
enum TaskPriority { low, medium, high, urgent }

class Task {
  final int id;
  final String title;
  final String? description;
  final TaskStatus status;
  final TaskPriority priority;
```

```
    final int projectId;
    final int? assigneeId;
    final int createdBy;
    final DateTime? dueDate;
    final DateTime? completedAt;
    final DateTime createdAt;
    final DateTime updatedAt;

    // TODO: Implementieren
}

// lib/repositories/task_repository.dart

class TaskRepository {
    // TODO: create, findById, findByProject, update, delete
    // TODO: updateStatus, assign
    // TODO: findByAssignee, findOverdue
}
```

Aufgabe 3.3: Comment Model & Repository (20 min)

```
// lib/models/comment.dart
// lib/repositories/comment_repository.dart

// TODO: Analog zu Task
```

Aufgabe 3.4: Project Service (45 min)

```
// lib/services/project_service.dart

class ProjectService {
    // TODO: Berechtigungsprüfung einbauen
    // - Nur Mitglieder sehen Projekte
    // - Nur Owner/Admin können bearbeiten
    // - Nur Owner kann löschen
}
```

Aufgabe 3.5: Task Service (45 min)

```
// lib/services/task_service.dart

class TaskService {
    // TODO: Task-Logik
    // - Nur Projekt-Mitglieder können Tasks erstellen
    // - Benachrichtigung bei Zuweisung
    // - completedAt setzen bei Status=done
}
```

Aufgabe 3.6: Handler (60 min)

```
// lib/handlers/project_handler.dart
// lib/handlers/task_handler.dart
// lib/handlers/comment_handler.dart
```

```
// TODO: CRUD Endpoints für alle Entitäten
```

#### 5.5.1.6 Phase 4: Real-time

Aufgabe 4.1: WebSocket Handler (45 min)

```
// lib/handlers/websocket_handler.dart

class WebSocketHandler {
  final ConnectionManager _connections;
  final JwtService _jwtService;

  // TODO: handleConnection(WebSocketChannel)
  // TODO: Authentifizierung via Token
  // TODO: Subscription zu Projekten
}
```

Aufgabe 4.2: Notification Service (30 min)

```
// lib/services/notification_service.dart

class NotificationService {
  // TODO: notifyTaskCreated(task)
  // TODO: notifyTaskUpdated(task, changes)
  // TODO: notifyTaskAssigned(task, assignee)
  // TODO: notifyCommentAdded(comment)
}
```

Aufgabe 4.3: Integration (30 min)

```
// TODO: NotificationService in TaskService integrieren
// TODO: Bei Task-Änderungen WebSocket-Updates senden
```

#### 5.5.1.7 Phase 5: Produktion

Aufgabe 5.1: Security (45 min)

```
// lib/middleware/rate_limit_middleware.dart
// TODO: Rate Limiting implementieren

// lib/middleware/security_middleware.dart
// TODO: Security Headers

// lib/utils/validator.dart
// TODO: Input Validation
```

Aufgabe 5.2: Caching (30 min)

```
// lib/services/cache_service.dart

class CacheService {
  final RedisConnection redis;
```

```
// TODO: get, set, delete
// TODO: Cache für häufige Queries (z.B. User-Profil)
}
```

Aufgabe 5.3: Docker (30 min)

```
# Dockerfile

# TODO: Multi-Stage Build
# TODO: Non-root User
# TODO: Health Check
```

Aufgabe 5.4: Tests (60 min)

```
// test/services/auth_service_test.dart
// test/handlers/auth_handler_test.dart
// test/integration/auth_flow_test.dart

// TODO: Mindestens:
// - Auth Service Unit Tests
// - Ein Integration Test für Login-Flow
```

Aufgabe 5.5: Dokumentation (30 min)

```
# README.md

#### Setup
#### API Endpoints
#### Environment Variables
#### Development
#### Deployment
```

### 5.5.1.8 Abgabe

Funktionale Anforderungen

- ☐ User Registration & Login
- ☐ JWT Access/Refresh Token Flow
- ☐ Projekte erstellen, bearbeiten, löschen
- ☐ Projekt-Mitglieder verwalten
- ☐ Tasks mit Status und Priorität
- ☐ Task-Zuweisung
- ☐ Kommentare zu Tasks
- ☐ WebSocket für Live-Updates
- ☐ Health Checks

Technische Anforderungen

- ☐ PostgreSQL Datenbank
- ☐ Redis (mindestens für Rate Limiting)
- ☐ Strukturiertes Logging
- ☐ Input Validation
- ☐ Error Handling

- ☐ Docker Compose funktioniert
- ☐ Mindestens 5 Tests

#### Bonus

- ☐ Pagination für Listen
- ☐ Task-Filter (Status, Assignee, etc.)
- ☐ File Uploads für Avatare
- ☐ Email-Benachrichtigungen
- ☐ API-Dokumentation (OpenAPI)
- ☐ CI/CD Pipeline

#### 5.5.1.9 Hilfestellung

Wenn du nicht weiterkommst

1. Schaue in die entsprechende Lerneinheit
2. Prüfe die Lösungsdatei dieser Übung
3. Vereinfache das Problem
4. Teste mit Postman/curl

#### Debugging

```
# Logs
docker compose logs -f

# DB Shell
docker compose exec db psql -U postgres -d taskapi

# Redis CLI
docker compose exec redis redis-cli
```

#### API Testen

```
# Register
curl -X POST http://localhost:8080/api/auth/register \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"test123","name":"Test"}'

# Login
curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@test.com","password":"test123"}'

# Mit Token
curl http://localhost:8080/api/projects \
  -H "Authorization: Bearer <token>"
```

#### 5.5.2 Lösung

Diese Lösung zeigt die wichtigsten Komponenten. Der vollständige Code wäre zu umfangreich für eine einzelne Datei.

### 5.5.2.1 Projektstruktur

```
task_api/  
+-- bin/  
|   +-- server.dart  
+-- lib/  
|   +-- config/  
|   |   +-- config.dart  
|   +-- models/  
|   |   +-- user.dart  
|   |   +-- project.dart  
|   |   +-- task.dart  
|   +-- repositories/  
|   |   +-- user_repository.dart  
|   |   +-- project_repository.dart  
|   |   +-- task_repository.dart  
|   +-- services/  
|   |   +-- auth_service.dart  
|   |   +-- password_service.dart  
|   |   +-- jwt_service.dart  
|   |   +-- project_service.dart  
|   +-- handlers/  
|   |   +-- auth_handler.dart  
|   |   +-- project_handler.dart  
|   |   +-- task_handler.dart  
|   +-- middleware/  
|   |   +-- auth_middleware.dart  
|   |   +-- error_middleware.dart  
|   +-- app.dart  
+-- migrations/  
|   +-- 001_initial.sql  
+-- docker-compose.yml  
+-- Dockerfile  
+-- pubspec.yaml
```

### 5.5.2.2 Konfiguration

```
// lib/config/config.dart  
  
import 'dart:io';  
  
class AppConfig {  
  final int port;  
  final String databaseUrl;  
  final String? redisUrl;  
  final String jwtSecret;  
  final Duration jwtAccessDuration;  
  final Duration jwtRefreshDuration;  
  final bool isDevelopment;  
  
  AppConfig({  
    required this.port,
```

```
    required this.databaseUrl,
    this.redisUrl,
    required this.jwtSecret,
    this.jwtAccessDuration = const Duration(minutes: 15),
    this.jwtRefreshDuration = const Duration(days: 7),
    this.isDevelopment = false,
  });

  factory AppConfig.fromEnvironment() {
    return AppConfig(
      port: int.parse(Platform.environment['PORT'] ?? '8080'),
      databaseUrl: Platform.environment['DATABASE_URL'] ??
        'postgres://postgres:postgres@localhost:5432/taskapi',
      redisUrl: Platform.environment['REDIS_URL'],
      jwtSecret: Platform.environment['JWT_SECRET'] ?? 'dev-secret-change-me',
      isDevelopment: Platform.environment['ENV'] != 'production',
    );
  }
}
```

### 5.5.2.3 Models

```
// lib/models/user.dart

class User {
  final int id;
  final String email;
  final String passwordHash;
  final String name;
  final String role;
  final bool isActive;
  final DateTime createdAt;

  User({
    required this.id,
    required this.email,
    required this.passwordHash,
    required this.name,
    this.role = 'user',
    this.isActive = true,
    required this.createdAt,
  });

  factory User.fromRow(Map<String, dynamic> row) {
    return User(
      id: row['id'] as int,
      email: row['email'] as String,
      passwordHash: row['password_hash'] as String,
      name: row['name'] as String,
      role: row['role'] as String? ?? 'user',
    );
  }
}
```

```
        isActive: row['is_active'] as bool? ?? true,
        createdAt: row['created_at'] as DateTime,
    );
}

Map<String, dynamic> toJson() => {
    'id': id,
    'email': email,
    'name': name,
    'role': role,
    'isActive': isActive,
    'createdAt': createdAt.toIso8601String(),
};
}

// lib/models/project.dart

class Project {
    final int id;
    final String name;
    final String? description;
    final int ownerId;
    final bool isArchived;
    final DateTime createdAt;
    final DateTime updatedAt;

    Project({
        required this.id,
        required this.name,
        this.description,
        required this.ownerId,
        this.isArchived = false,
        required this.createdAt,
        required this.updatedAt,
    });

    factory Project.fromRow(Map<String, dynamic> row) {
        return Project(
            id: row['id'] as int,
            name: row['name'] as String,
            description: row['description'] as String?,
            ownerId: row['owner_id'] as int,
            isArchived: row['is_archived'] as bool? ?? false,
            createdAt: row['created_at'] as DateTime,
            updatedAt: row['updated_at'] as DateTime,
        );
    }

    Map<String, dynamic> toJson() => {
        'id': id,
        'name': name,
```

```

        'description': description,
        'ownerId': ownerId,
        'isArchived': isArchived,
        'createdAt': createdAt.toIso8601String(),
        'updatedAt': updatedAt.toIso8601String(),
    };
}

// lib/models/task.dart

enum TaskStatus { todo, inProgress, review, done }
enum TaskPriority { low, medium, high, urgent }

class Task {
    final int id;
    final String title;
    final String? description;
    final TaskStatus status;
    final TaskPriority priority;
    final int projectId;
    final int? assigneeId;
    final int createdBy;
    final DateTime? dueDate;
    final DateTime? completedAt;
    final DateTime createdAt;
    final DateTime updatedAt;

    Task({
        required this.id,
        required this.title,
        this.description,
        this.status = TaskStatus.todo,
        this.priority = TaskPriority.medium,
        required this.projectId,
        this.assigneeId,
        required this.createdBy,
        this.dueDate,
        this.completedAt,
        required this.createdAt,
        required this.updatedAt,
    });

    factory Task.fromRow(Map<String, dynamic> row) {
        return Task(
            id: row['id'] as int,
            title: row['title'] as String,
            description: row['description'] as String?,
            status: TaskStatus.values.byName(row['status'] as String? ?? 'todo'),
            priority: TaskPriority.values.byName(row['priority'] as String? ??
↵ 'medium'),
            projectId: row['project_id'] as int,

```

```
    assigneeId: row['assignee_id'] as int?,
    createdBy: row['created_by'] as int,
    dueDate: row['due_date'] as DateTime?,
    completedAt: row['completed_at'] as DateTime?,
    createdAt: row['created_at'] as DateTime,
    updatedAt: row['updated_at'] as DateTime,
  );
}

Map<String, dynamic> toJson() => {
  'id': id,
  'title': title,
  'description': description,
  'status': status.name,
  'priority': priority.name,
  'projectId': projectId,
  'assigneeId': assigneeId,
  'createdBy': createdBy,
  'dueDate': dueDate?.toIso8601String(),
  'completedAt': completedAt?.toIso8601String(),
  'createdAt': createdAt.toIso8601String(),
  'updatedAt': updatedAt.toIso8601String(),
};
}
```

#### 5.5.2.4 Services

```
// lib/services/password_service.dart

import 'package:bcrypt/bcrypt.dart';

class PasswordService {
  final int costFactor;

  PasswordService({this.costFactor = 12});

  String hash(String password) {
    return BCrypt.hashpw(password, BCrypt.gensalt(logRounds: costFactor));
  }

  bool verify(String password, String hash) {
    return BCrypt.checkpw(password, hash);
  }
}

// lib/services/jwt_service.dart

import 'package:dart_jsonwebtoken/dart_jsonwebtoken.dart';

class JwtService {
```

```
final String secret;
final Duration accessDuration;
final Duration refreshDuration;

JwtService({
  required this.secret,
  this.accessDuration = const Duration(minutes: 15),
  this.refreshDuration = const Duration(days: 7),
});

String generateAccessToken(User user) {
  final jwt = JWT({
    'sub': user.id.toString(),
    'email': user.email,
    'name': user.name,
    'role': user.role,
    'type': 'access',
  });
  return jwt.sign(SecretKey(secret), expiresIn: accessDuration);
}

String generateRefreshToken(User user) {
  final jwt = JWT({
    'sub': user.id.toString(),
    'type': 'refresh',
  });
  return jwt.sign(SecretKey(secret), expiresIn: refreshDuration);
}

Map<String, dynamic>? verifyToken(String token) {
  try {
    final jwt = JWT.verify(token, SecretKey(secret));
    return jwt.payload as Map<String, dynamic>;
  } catch (e) {
    return null;
  }
}

String? extractFromHeader(String? header) {
  if (header == null || !header.startsWith('Bearer ')) return null;
  return header.substring(7);
}

// lib/services/auth_service.dart

class AuthService {
  final UserRepository _userRepo;
  final RefreshTokenRepository _tokenRepo;
  final PasswordService _password;
  final JwtService _jwt;
```

```
AuthService(this._userRepo, this._tokenRepo, this._password, this._jwt);

Future<User> register(String email, String password, String name) async {
  if (await _userRepo.existsByEmail(email)) {
    throw AuthException('Email already exists');
  }

  final hash = _password.hash(password);
  return await _userRepo.create(email: email, passwordHash: hash, name: name);
}

Future<TokenPair> login(String email, String password) async {
  final user = await _userRepo.findByEmail(email);

  if (user == null || !user.isActive) {
    // Timing attack prevention
    _password.hash(password);
    throw AuthException('Invalid credentials');
  }

  if (!_password.verify(password, user.passwordHash)) {
    throw AuthException('Invalid credentials');
  }

  return _generateTokens(user);
}

Future<TokenPair> refreshToken(String refreshToken) async {
  final payload = _jwt.verifyToken(refreshToken);
  if (payload == null || payload['type'] != 'refresh') {
    throw AuthException('Invalid refresh token');
  }

  final userId = int.parse(payload['sub'] as String);
  final storedToken = await _tokenRepo.findByUserAndHash(userId,
↵ _hashToken(refreshToken));

  if (storedToken == null || storedToken.isRevoked) {
    throw AuthException('Token revoked');
  }

  final user = await _userRepo.findById(userId);
  if (user == null || !user.isActive) {
    throw AuthException('User not found');
  }

  // Rotate token
  await _tokenRepo.revoke(storedToken.id);

  return _generateTokens(user);
}
```

```
}

Future<void> logout(String refreshToken) async {
  final payload = _jwt.verifyToken(refreshToken);
  if (payload != null) {
    final userId = int.parse(payload['sub'] as String);
    final hash = _hashToken(refreshToken);
    final token = await _tokenRepo.findByUserAndHash(userId, hash);
    if (token != null) {
      await _tokenRepo.revoke(token.id);
    }
  }
}

Future<TokenPair> _generateTokens(User user) async {
  final accessToken = _jwt.generateAccessToken(user);
  final refreshToken = _jwt.generateRefreshToken(user);

  await _tokenRepo.create(
    userId: user.id,
    tokenHash: _hashToken(refreshToken),
    expiresAt: DateTime.now().add(_jwt.refreshDuration),
  );

  return TokenPair(
    accessToken: accessToken,
    refreshToken: refreshToken,
    expiresIn: _jwt.accessDuration.inSeconds,
  );
}

String _hashToken(String token) {
  return sha256.convert(utf8.encode(token)).toString();
}

class TokenPair {
  final String accessToken;
  final String refreshToken;
  final int expiresIn;

  TokenPair({
    required this.accessToken,
    required this.refreshToken,
    required this.expiresIn,
  });

  Map<String, dynamic> toJson() => {
    'access_token': accessToken,
    'refresh_token': refreshToken,
    'expires_in': expiresIn,
  }
}
```

```
        'token_type': 'Bearer',  
      };  
    }  
  }
```

#### 5.5.2.5 Handlers

```
// lib/handlers/auth_handler.dart  
  
import 'dart:convert';  
import 'package:shelf/shelf.dart';  
import 'package:shelf_router/shelf_router.dart';  
  
class AuthHandler {  
  final AuthService _authService;  
  
  AuthHandler(this._authService);  
  
  Router get router {  
    final router = Router();  
  
    router.post('/register', _register);  
    router.post('/login', _login);  
    router.post('/refresh', _refresh);  
    router.post('/logout', _logout);  
    router.get('/me', _me);  
  
    return router;  
  }  
  
  Future<Response> _register(Request request) async {  
    final body = jsonDecode(await request.readAsString());  
  
    final email = body['email'] as String?;  
    final password = body['password'] as String?;  
    final name = body['name'] as String?;  
  
    if (email == null || password == null || name == null) {  
      return Response.badRequest(  
        body: jsonEncode({'error': 'email, password, and name required'}),  
      );  
    }  
  
    try {  
      final user = await _authService.register(email, password, name);  
      return Response(201,  
        body: jsonEncode(user.toJson()),  
        headers: {'content-type': 'application/json'});  
    } on AuthException catch (e) {  
      return Response(409,  
        body: jsonEncode({'error': e.message}),  
      );  
    }  
  }  
}
```

```
        headers: {'content-type': 'application/json'});
    }
}

Future<Response> _login(Request request) async {
    final body = jsonDecode(await request.readAsString());

    final email = body['email'] as String?;
    final password = body['password'] as String?;

    if (email == null || password == null) {
        return Response.badRequest(
            body: jsonEncode({'error': 'email and password required'}),
        );
    }

    try {
        final tokens = await _authService.login(email, password);
        return Response.ok(
            jsonEncode(tokens.toJson()),
            headers: {'content-type': 'application/json'},
        );
    } on AuthException catch (e) {
        return Response(401,
            body: jsonEncode({'error': e.message}),
            headers: {'content-type': 'application/json'});
    }
}

Future<Response> _refresh(Request request) async {
    final body = jsonDecode(await request.readAsString());
    final refreshToken = body['refresh_token'] as String?;

    if (refreshToken == null) {
        return Response.badRequest(
            body: jsonEncode({'error': 'refresh_token required'}),
        );
    }

    try {
        final tokens = await _authService.refreshToken(refreshToken);
        return Response.ok(
            jsonEncode(tokens.toJson()),
            headers: {'content-type': 'application/json'},
        );
    } on AuthException catch (e) {
        return Response(401,
            body: jsonEncode({'error': e.message}),
            headers: {'content-type': 'application/json'});
    }
}
```

```
Future<Response> _logout(Request request) async {
  final body = jsonDecode(await request.readAsString());
  final refreshToken = body['refresh_token'] as String?;

  if (refreshToken != null) {
    await _authService.logout(refreshToken);
  }

  return Response.ok(jsonEncode({'message': 'Logged out'}));
}

Future<Response> _me(Request request) async {
  final user = request.context['user'] as User?;
  if (user == null) {
    return Response(401, body: jsonEncode({'error': 'Not authenticated'}));
  }
  return Response.ok(
    jsonEncode(user.toJson()),
    headers: {'content-type': 'application/json'},
  );
}
```

#### 5.5.2.6 Middleware

```
// lib/middleware/auth_middleware.dart

import 'package:shelf/shelf.dart';

Middleware authMiddleware(JwtService jwt, UserRepository userRepo) {
  return (Handler innerHandler) {
    return (Request request) async {
      final token = jwt.extractFromHeader(request.headers['authorization']);

      if (token == null) {
        return Response(401,
          body: jsonEncode({'error': 'Authorization required'}),
          headers: {'content-type': 'application/json'});
      }

      final payload = jwt.verifyToken(token);
      if (payload == null || payload['type'] != 'access') {
        return Response(401,
          body: jsonEncode({'error': 'Invalid token'}),
          headers: {'content-type': 'application/json'});
      }

      final userId = int.parse(payload['sub'] as String);
      final user = await userRepo.findById(userId);
    };
  };
}
```

```

        if (user == null || !user.isActive) {
            return Response(401,
                body: jsonEncode({'error': 'User not found'}),
                headers: {'content-type': 'application/json'});
        }

        return innerHandler(request.change(context: {'user': user}));
    };
};
}

Middleware requireRole(String role) {
    return (Handler innerHandler) {
        return (Request request) async {
            final user = request.context['user'] as User?;

            if (user == null) {
                return Response(401, body: jsonEncode({'error': 'Not authenticated'}));
            }

            if (user.role != role && user.role != 'admin') {
                return Response(403, body: jsonEncode({'error': 'Insufficient
↪ permissions'}));
            }

            return innerHandler(request);
        };
    };
}

```

### 5.5.2.7 App Assembly

```

// lib/app.dart

import 'package:shelf/shelf.dart';
import 'package:shelf_router/shelf_router.dart';

class App {
    final AppConfig config;
    final Connection db;
    final RedisConnection? redis;

    // Repositories
    late final UserRepository userRepo;
    late final ProjectRepository projectRepo;
    late final TaskRepository taskRepo;
    late final RefreshTokenRepository tokenRepo;

    // Services

```

```
late final PasswordService passwordService;
late final JwtService jwtService;
late final AuthService authService;
late final ProjectService projectService;
late final TaskService taskService;

// Handlers
late final AuthHandler authHandler;
late final ProjectHandler projectHandler;
late final TaskHandler taskHandler;
late final HealthHandler healthHandler;

App(this.config, this.db, this.redis);

void init() {
    // Init repositories
    userRepo = UserRepository(db);
    projectRepo = ProjectRepository(db);
    taskRepo = TaskRepository(db);
    tokenRepo = RefreshTokenRepository(db);

    // Init services
    passwordService = PasswordService();
    jwtService = JwtService(
        secret: config.jwtSecret,
        accessDuration: config.jwtAccessDuration,
        refreshDuration: config.jwtRefreshDuration,
    );
    authService = AuthService(userRepo, tokenRepo, passwordService, jwtService);
    projectService = ProjectService(projectRepo, userRepo);
    taskService = TaskService(taskRepo, projectService);

    // Init handlers
    authHandler = AuthHandler(authService);
    projectHandler = ProjectHandler(projectService);
    taskHandler = TaskHandler(taskService);
    healthHandler = HealthHandler(db, redis);
}

Handler get handler {
    final router = Router();

    // Public routes
    router.mount('/api/auth/', authHandler.router.call);

    // Health
    router.get('/health', healthHandler.check);
    router.get('/health/ready', healthHandler.ready);
    router.get('/health/live', healthHandler.live);

    // Protected routes
```

```
final protectedRouter = Router();
protectedRouter.mount('/projects/', projectHandler.router.call);
protectedRouter.mount('/tasks/', taskHandler.router.call);

router.mount(
  '/api/',
  const Pipeline()
    .addMiddleware(authMiddleware(jwtService, userRepo))
    .addHandler(protectedRouter.call),
);

// Pipeline
return const Pipeline()
  .addMiddleware(logRequests())
  .addMiddleware(corsMiddleware())
  .addMiddleware(errorMiddleware())
  .addHandler(router.call);
}
}
```

#### 5.5.2.8 Server

```
// bin/server.dart

import 'dart:io';
import 'package:shelf/shelf_io.dart' as io;
import 'package:postgres/postgres.dart';

void main() async {
  final config = AppConfig.fromEnvironment();

  // Database
  final db = await Connection.open(
    Endpoint.parse(config.databaseUrl),
    settings: ConnectionSettings(sslMode: SslMode.disable),
  );

  // Redis (optional)
  RedisConnection? redis;
  if (config.redisUrl != null) {
    redis = await RedisConnection.connect(config.redisUrl!);
  }

  // App
  final app = App(config, db, redis);
  app.init();

  // Server
  final server = await io.serve(
    app.handler,
```

```
    InternetAddress.anyIPv4,
    config.port,
  );

  print('Server running on http://localhost:${server.port}');

  // Graceful shutdown
  ProcessSignal.sigint.watch().listen((_) async {
    print('Shutting down...');
    await server.close();
    await db.close();
    await redis?.close();
    exit(0);
  });
}
```

#### 5.5.2.9 Migration

```
-- migrations/001_initial.sql

CREATE EXTENSION IF NOT EXISTS "uuid-ossf";

CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  name VARCHAR(100) NOT NULL,
  role VARCHAR(50) DEFAULT 'user',
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE projects (
  id SERIAL PRIMARY KEY,
  name VARCHAR(200) NOT NULL,
  description TEXT,
  owner_id INTEGER REFERENCES users(id),
  is_archived BOOLEAN DEFAULT false,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE project_members (
  id SERIAL PRIMARY KEY,
  project_id INTEGER REFERENCES projects(id) ON DELETE CASCADE,
  user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
  role VARCHAR(50) DEFAULT 'member',
  joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(project_id, user_id)
```

```
);

CREATE TABLE tasks (
  id SERIAL PRIMARY KEY,
  title VARCHAR(300) NOT NULL,
  description TEXT,
  status VARCHAR(50) DEFAULT 'todo',
  priority VARCHAR(50) DEFAULT 'medium',
  project_id INTEGER REFERENCES projects(id) ON DELETE CASCADE,
  assignee_id INTEGER REFERENCES users(id),
  created_by INTEGER REFERENCES users(id),
  due_date TIMESTAMP,
  completed_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE comments (
  id SERIAL PRIMARY KEY,
  task_id INTEGER REFERENCES tasks(id) ON DELETE CASCADE,
  user_id INTEGER REFERENCES users(id),
  content TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE refresh_tokens (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id) ON DELETE CASCADE,
  token_hash VARCHAR(255) NOT NULL,
  expires_at TIMESTAMP NOT NULL,
  revoked_at TIMESTAMP,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indices
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_tasks_project ON tasks(project_id);
CREATE INDEX idx_tasks_assignee ON tasks(assignee_id);
CREATE INDEX idx_comments_task ON comments(task_id);
CREATE INDEX idx_refresh_tokens_user ON refresh_tokens(user_id);
```

#### 5.5.2.10 Docker

```
# Dockerfile

FROM dart:stable AS build
WORKDIR /app
COPY pubspec.* ./
RUN dart pub get
COPY . .
```

```

RUN dart compile exe bin/server.dart -o bin/server

FROM debian:bookworm-slim
RUN apt-get update && apt-get install -y ca-certificates && rm -rf ↵
  ↵ /var/lib/apt/lists/*
RUN useradd -r -u 1001 appuser
USER appuser
WORKDIR /app
COPY --from=build /app/bin/server /app/bin/server
ENV PORT=8080
EXPOSE 8080
HEALTHCHECK --interval=30s --timeout=3s CMD curl -f ↵
  ↵ http://localhost:8080/health || exit 1
ENTRYPOINT ["/app/bin/server"]

```

```

# docker-compose.yml

services:
  api:
    build: .
    ports:
      - "8080:8080"
    environment:
      - DATABASE_URL=postgres://postgres:postgres@db:5432/taskapi
      - JWT_SECRET=dev-secret
    depends_on:
      db:
        condition: service_healthy

  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: taskapi
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
    volumes:
      - ./migrations:/docker-entrypoint-initdb.d:ro
      - postgres_data:/var/lib/postgresql/data
    healthcheck:
      test: pg_isready -U postgres
      interval: 5s

volumes:
  postgres_data:

```

### 5.5.3 Ressourcen

#### 5.5.3.1 Referenz zu allen Lerneinheiten

Thema	Einheit	Wichtige Konzepte
HTTP Server	5.1-5.2	dart:io, Shelf, Handler
Routing	5.3	shelf_router, URL-Parameter
Middleware	5.4	Pipeline, CORS, Logging
Konfiguration	5.5	Environment Variables, .env
Architektur	5.6	Layered Architecture, DI
REST Design	6.1-6.4	HTTP Methoden, CRUD, Status Codes
Validierung	6.5	Input Validation, Error Handling
Pagination	6.7	Offset/Limit, Cursor
PostgreSQL	7.1-7.2	SQL, postgres Package
Repository	7.3	Repository Pattern
Relationen	7.4	JOINS, Foreign Keys
Migrations	7.5	Schema Versionierung
Redis	7.8	Caching, Sessions
Passwörter	8.1	bcrypt, Hashing
JWT	8.2	Access/Refresh Tokens
Auth Middleware	8.3	RBAC, Guards
API Security	8.5	Rate Limiting, CORS, Headers
Testing	8.6	Unit Tests, Mocks
WebSockets	9.1	Real-time, shelf_web_socket
Background Jobs	9.2	Queues, Scheduling
Logging	9.3	Strukturierte Logs, Metriken
Docker	9.4	Container, Compose, CI/CD

### 5.5.3.2 Cheat Sheet: Projekt-Setup

```
# Projekt erstellen
dart create -t server-shelf task_api
cd task_api

# Dependencies hinzufügen
# pubspec.yaml bearbeiten...
dart pub get

# Docker starten
docker compose up -d

# Server starten
dart run bin/server.dart
```

### 5.5.3.3 Cheat Sheet: API Response Format

```
// Erfolg (200, 201)
{
  "data": { ... },
  "meta": {
    "page": 1,
    "limit": 10,
    "total": 100
  }
}
```

```

    }
}

// Fehler (4xx, 5xx)
{
  "error": {
    "code": "VALIDATION_ERROR",
    "message": "Invalid input",
    "details": {
      "email": "Invalid email format"
    }
  }
}
}

```

#### 5.5.3.4 Cheat Sheet: Berechtigungen

```

// Projekt-Rollen
enum ProjectRole { owner, admin, member, viewer }

// Berechtigungsmatrix
//
//          owner  admin  member  viewer
// projekt.update  [x]    [x]    -      -
// projekt.delete  [x]    -      -      -
// member.add      [x]    [x]    -      -
// member.remove   [x]    [x]    -      -
// task.create     [x]    [x]    [x]    -
// task.update     [x]    [x]    [x]    -
// task.delete     [x]    [x]    -      -
// task.view       [x]    [x]    [x]    [x]
// comment.add     [x]    [x]    [x]    [x]

```

#### 5.5.3.5 Cheat Sheet: WebSocket Messages

```

// Client -> Server
{"type": "auth", "token": "..."}
{"type": "subscribe", "project_id": 1}
{"type": "unsubscribe", "project_id": 1}

// Server -> Client
{"type": "auth_success", "user_id": 1}
{"type": "task_created", "task": {...}}
{"type": "task_updated", "task": {...}, "changes": [...]}
{"type": "task_deleted", "task_id": 1}
{"type": "comment_added", "comment": {...}}

```

#### 5.5.3.6 Cheat Sheet: Test-Befehle

```

# Register
curl -X POST http://localhost:8080/api/auth/register \

```

```
-H "Content-Type: application/json" \
-d '{"email":"test@test.com","password":"password123","name":"Test User"}'

# Login
curl -X POST http://localhost:8080/api/auth/login \
-H "Content-Type: application/json" \
-d '{"email":"test@test.com","password":"password123"}'

# Token speichern
TOKEN="eyJ..."

# Projekte auflisten
curl http://localhost:8080/api/projects \
-H "Authorization: Bearer $TOKEN"

# Projekt erstellen
curl -X POST http://localhost:8080/api/projects \
-H "Authorization: Bearer $TOKEN" \
-H "Content-Type: application/json" \
-d '{"name":"My Project","description":"A test project"}'

# Task erstellen
curl -X POST http://localhost:8080/api/projects/1/tasks \
-H "Authorization: Bearer $TOKEN" \
-H "Content-Type: application/json" \
-d '{"title":"First Task","priority":"high"}'

# Task Status ändern
curl -X PATCH http://localhost:8080/api/tasks/1/status \
-H "Authorization: Bearer $TOKEN" \
-H "Content-Type: application/json" \
-d '{"status":"in_progress"}'
```

### 5.5.3.7 Cheat Sheet: Debugging

```
# Server Logs
docker compose logs -f api

# Database Shell
docker compose exec db psql -U postgres -d taskapi

# Alle Tables anzeigen
\dt

# Table Schema
\d users
\d tasks

# Query testen
SELECT * FROM users;
```

```
SELECT t.*, u.name as assignee_name
FROM tasks t
LEFT JOIN users u ON t.assignee_id = u.id;
```

*# Redis CLI*

```
docker compose exec redis redis-cli
KEYS *
GET key
```

### 5.5.3.8 Best Practices Checklist

#### Code-Qualität

- ☐ Separation of Concerns (Handler -> Service -> Repository)
- ☐ Keine Business-Logik in Handlern
- ☐ Keine Datenbankzugriffe in Handlern
- ☐ DTOs für Input/Output
- ☐ Aussagekräftige Variablennamen

#### Sicherheit

- ☐ Passwörter mit bcrypt hashen
- ☐ JWT mit sicherem Secret
- ☐ Input validieren
- ☐ SQL Injection verhindern (Prepared Statements)
- ☐ Rate Limiting
- ☐ CORS konfiguriert
- ☐ Security Headers

#### API Design

- ☐ RESTful Endpoints
- ☐ Konsistente Response-Struktur
- ☐ Korrekte HTTP Status Codes
- ☐ Aussagekräftige Fehlermeldungen
- ☐ Pagination für Listen

#### Infrastruktur

- ☐ Docker Compose funktioniert
- ☐ Health Checks implementiert
- ☐ Strukturiertes Logging
- ☐ Environment Variables für Secrets
- ☐ Graceful Shutdown

### 5.5.3.9 Häufige Fehler

#### 1. Token nicht im Header

*# Falsch*

```
curl http://localhost:8080/api/projects
```

*# Richtig*

```
curl http://localhost:8080/api/projects \
-H "Authorization: Bearer $TOKEN"
```

## 2. Content-Type vergessen

```
# Falsch
curl -X POST http://localhost:8080/api/auth/login \
  -d '{"email":"..."}'

# Richtig
curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"..."}'
```

## 3. Datenbank nicht bereit

```
# Warten bis DB healthy
docker compose up -d
docker compose exec db pg_isready -U postgres
```

## 4. Migrations nicht ausgeführt

```
# SQL manuell ausführen
docker compose exec db psql -U postgres -d taskapi -f
↪ /docker-entrypoint-initdb.d/001_initial.sql
```

**5.5.3.10 Hilfreiche Links**

- Shelf Documentation
- postgres Package
- JWT.io - Token debuggen
- HTTP Status Codes
- REST API Design