Dart Frog

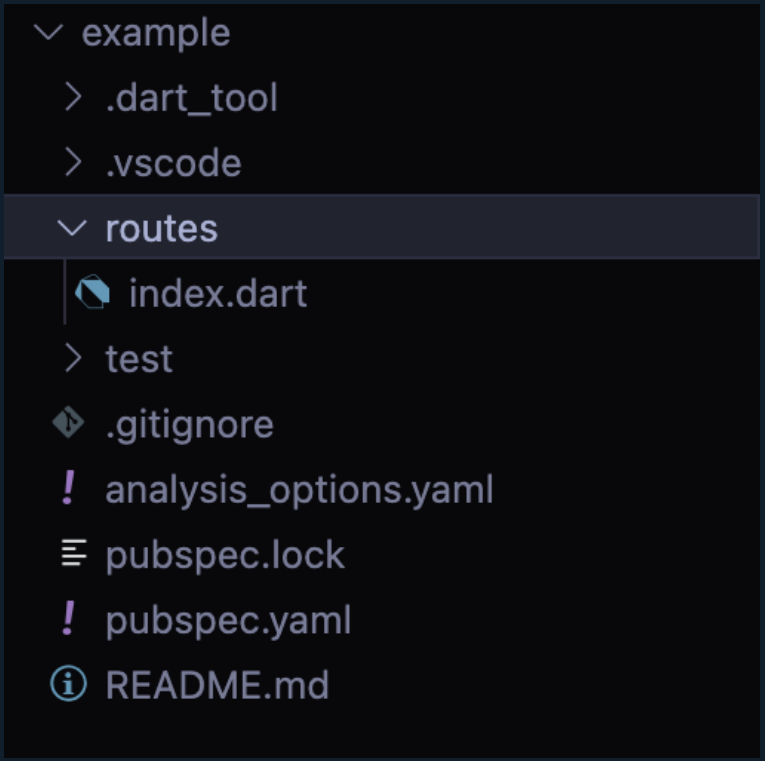
Intro & Basics

2025.3.25 | Lukianova Liza

Basics

Environment setting & Installations

- Dart package
  1. Installation: dart pub add dart_frog
  2. Project setting: dart_frog create <project name>
  3. Run (dev/build): dart_frog dev/build
     - Default port: 8080



Routes: Requests

dart_frog new route "/home"

Response

- Route = .dart file in [routes] folder
  - ~ index.dart → / endpoint
  - Each route includes onRequest function

example > routes > index.dart > ...

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

Response onRequest(RequestContext context) {
  return Response(body: 'Welcome to Dart Frog!');
}
```

Routes: Requests

- RequestContext class
  - Request itself is a property of RequestContext!

Request

Properties

connectionInfo

headers

method

uri

queryParameters

Methods

body

json

```dart
Future<Response> onRequest(RequestContext context) async {
  final request = context.request;
  //     [Request properties]
  // connection info
  final conn = request.connectionInfo;
  // headers
  final headers = request.headers; // Map <String, String>
  // method
  final method = request.method; // HttpMethod enum

  final params = request.uri.queryParameters;
  //  [Request  methods]
  final body = await request.body(); // Future<String>!!!
  final json_body = await request.json();


  return Response.json(body: {
    'conn_info': conn?.remotePort,
    'host_info': headers['host'],
    'original_method': method.toString(),
    'user_id': params['user_id'],
    'body': body,
    'body_json': json_body['greetings'],
  });
}
```

```
"conn_info": 52673,
"host_info": "localhost:8080",
"original_method": "HttpMethod.get",
"user_id": "1",
"body": "{\n    \"greetings\": \"hi\"\n}",
"body_json": "hi"
```

Routes: Responses

Constructors

Response.new({int statusCode = 200, String? body, Map<String, Object>? headers, Encoding? encoding})
Create a Response with a string body.

Response.bytes({int statusCode = 200, List<int>? body, Map<String, Object>? headers})
Create a Response with a byte array body.

Response.json({int statusCode = 200, Object? body = const <String, dynamic>{}, Map<String, Object> headers = const <String, Object>{}})
Create a Response with a json encoded body.

Response

- Dart object (classes) can also be put in json response...
  - ...if it has toJson method (i.e. serializable)

```dart
@JsonSerializable()
class User {
  const User({required this.name, required this.age});

  final String name;
  final int age;

  Map<String, dynamic> toJson() => _$UserToJson(this);
}
```

```dart
return Response.json(body: {
  'conn_info': conn?.remotePort,
  'host_info': headers['host'],
  'original_method': method.toString(),
  'user_id': params['user_id'],
  'body': body,
  'body_json': json_body['greetings'],
  'user_info': User(name: 'Dash', age: 42),
}); // Response.json
```

```
"conn_info": 55244,
"host_info": "localhost:8080",
"original_method": "HttpMethod.get",
"user_id": "1",
"body": "{\n    \"greetings\": \"hi\"\n}",
"body_json": "hi",
"user_info": {
    "name": "Dash",
    "age": 42
}
```

Routes: different routes

- Dynamic routes ~routes/users/[id].dart

```
Response onRequest(RequestContext context, String id) {
  return Response(body: 'post id: $id');
}
```

- Wildcard routes ~ routes/[...page].dart
  - Matches with any page of any level!
    /routes/today OR /routes/features/starred etc

- Rouge routes

```
routes
├── api
│   └── example.dart
└── api.dart
```

```
routes
├── api
│   ├── example.dart
│   └── index.dart
```

- Routes for static files

public / images
smiling_quokka.webp

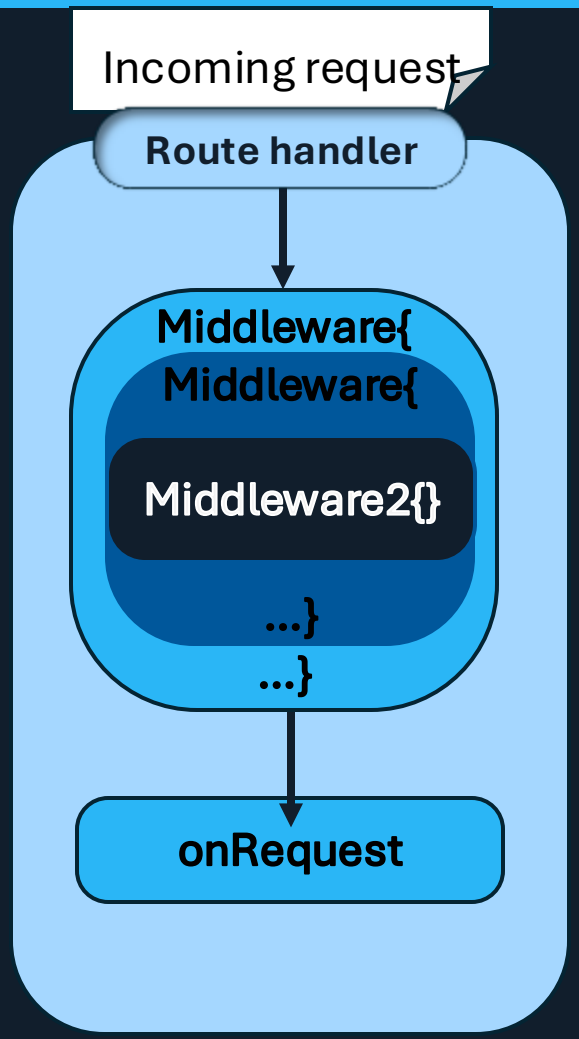
→ [host]/images/smiling_quokka.webp

Middleware

dart_frog new middleware "/hello"

Handler

- What is middleware???
  - Function running in between the incoming HTTP request and the final response being sent back to the client
  - Always triggered by incoming requests
  - Can modify the **request** (e.g., for validation, authentication, logging, etc.) before it reaches the route handler...
  - ...and response (e.g., for logging, adding headers, or transforming data) after the route handler has finished
  - <u>additional level of abstraction and flexibility</u>
  - <u>!!! There can only ever be one piece of middleware per route directory</u>



Middleware

Middleware | Handler | Response | Request

```
Middleware requestLogger({
  // ignore: avoid_positional_boolean_parameters
  void Function(String message, bool isError)? logger,
}) {
  return fromShelfMiddleware(shelf.logRequests(logger: logger));
}
```

```
Handler authMiddleware(Handler handler) {
  return (context) async {
    final request = context.request;

    // Check if the 'Authorization' header is present
    if (!request.headers.containsKey('Authorization')) {
      return Response.json(
        body: {'error': 'Authorization header is missing'},
        statusCode: HttpStatus.unauthorized,
      );
    }

    // Continue processing the request
    return await handler(context);
  };
}
```

- **Handler** is simply a function that takes a RequestContext as an argument and returns a Response
- Middleware is a function that wraps or modifies a Handler
- ... Middleware is a function that wraps function!
- Middleware is automatically applied to routes

```
Handler middleware(Handler handler) {
  return handler.use(authMiddleware).use(requestLogger());
}
```

```
2025-03-24T23:24:16.755638  0:00:00.001521 POST    [401] /
2025-03-24T23:25:08.296199  0:00:00.007332 GET     [200] /
```

Dependency Injection

```
Handler middleware(Handler handler) {
  return handler
      .use(provider<CardsRepository>((_) {
        return DatabaseClient(
          dbUrl: Platform.environment['DB_URL'],
          dbUser: Platform.environment['DB_USER'],
          dbPassword: Platform.environment['DB_PASSWORD'],
        );
      }),
    );
}
```

Provider **Middleware**

- Provider: built-in dependency injection mechanism (Middleware)
  - Inject an instance <T> in app by using <provider> on route middleware to request context
  - Accessed by **context.read<T>**. throughout *request lifecycle*

```
Handler middleware(Handler handler) {
  return handler
  .use(authMiddleware)
  .use(requestLogger())
  .use(provider<String>((context) =>
  'Liza')); //e.g. DB entry could be p
}
```

```
final name = context.read<String>();
```

```
return Response.json(body: {
  'conn_info': conn?.remotePort,
  'host_info': headers['host'],
  'original_method': method.toString(),
  'user_id': params['user_id'],
  'body': body,
  'body_json': json_body['greetings'],
  'body_json_name': "${json_body['greetings']}, $name",
```

```
"body_json": "hi",
"body_json_name": "hi, Liza",
```

Instances of the same type can not be injected!

- ✓ Use custom class
- ✓ Use List of same instances

Dependency Injection

Provider | Middleware

- Can be defined inline and as a separate Middleware
- Can insect asynchronous values (use <Future<Type>>)
- Lazy initialization: if context.read is not called, provider is not executed
- If providers use dependent instances, their providers are called **bottom-up!**
- Default: value is recreated on each read call
  - Use caching:

```
String? _greeting;   #global private value is declared

Middleware cachedGreetingProvider() {
  return provider<String>((context) => _greeting ??= 'Hello World');
}                    #assigned on first call → on next call _greeting is read
```

DB integration

Native clients

- e.g mysql_client

✅ easier to install

❌ uses native SQL commands

```
final pool = MySQLConnectionPool(
  host: '127.0.0.1',
  port: 3306,
  userName: 'your_user',
  password: 'your_password',
  maxConnections: 10,
  databaseName: 'your_database_name',
);
```

```
var result = await pool.execute("SELECT *
FROM book WHERE id = :id", {"id": 1});
```

ORM Clients

- e.g Prisma

❌ requires more dependencies and harder to set

✅ Synchronizes with DB and does ORM

```
final user = await prisma.user.create(
  data: PrismaUnion.$1(UserCreateInput(
    email: "seven@odroe.com",
    name: PrismaUnion.$1("Seven Du"),
  )),
);
```

Other [Dart] BE frameworks..?

- Lucifer Lightbringer
  - Built on top of native dart HttpServer
  - Last updated 3 years ago...
- Alfred
  - expressjs like server framework
- Serverpod
  - Automatically generates APIs