



# Cocktail Backend – Spring Boot (LB 295)

---



## Projektbeschreibung

Dieses Projekt ist ein **Spring Boot Backend** für eine Cocktail-Webanwendung, welche in [LB\\_Projekt\\_m294](#) zu finden ist.

Es stellt eine interne CRUD-API bereit, sodass das M294-Frontend mit kleinen Anpassungen (s. Abschnitt unten) weiterverwendet werden kann.

Das Backend nutzt **PostgreSQL** (Docker-Container) als Hauptdatenbank und **H2** als In-Memory-Datenbank für Tests.

---



## Änderungen im Frontend

Das M294-Frontend wurde ursprünglich für die öffentliche TheCocktailDB API entwickelt. Damit es mit dem M295-Backend funktioniert, müssen folgende Anpassungen vorgenommen werden:

### 1. API-URLs umstellen

Statt [https://www.thecocktaildb.com/...](https://www.thecocktaildb.com/) auf <http://localhost:8080/api/cocktails> verweisen.

### 2. Feldnamen anpassen

- `strDrink` → `name`
- `strInstructions` → `instructions`
- `strDrinkThumb` → `imageUrl`

### 3. Such- und Detailendpunkte ändern

- Suche: `?q={suchbegriff}`
- Details: `/api/cocktails/{id}`

### 4. Bilder sicherstellen

- Entweder `imageUrl`-Feld im Backend pflegen oder Frontend auf lokale Bilder umstellen.

Diese Änderungen sind optional und nur nötig, falls das M294-Frontend direkt mit diesem Backend betrieben werden soll.

---



## Projektstruktur

```
src/main/java/com/wiss/cocktailbackend
|
├── config/           # SwaggerConfig, WebConfig
├── controller/       # CocktailController
├── dto/              # CocktailDTO, CocktailListItemDTO
```

```
|— entity/          # JPA-Entities (Cocktail, CocktailIngredient, Ingredient)
|— mapper/          # CocktailMapper
|— repository/      # Repositories (CocktailIngredientRepository,
CocktailRepository, IngredientRepository)
|— service/         # CocktailService
|— CocktailBackendApplication.java
```

## Klassendiagramm

```
classDiagram
    direction LR

    class CocktailController {
        +list(q, alcoholic, page, size)
        +get(id)
        +create(dto)
        +update(id, dto)
        +delete(id)
    }

    class CocktailService {
        +list(q, alcoholic, pageable)
        +get(id)
        +create(dto)
        +update(id, dto)
        +delete(id)
    }

    class CocktailMapper {
        +toDto(Cocktail): CocktailDTO
        +toListItem(Cocktail): CocktailListItemDTO
        +apply(Cocktail, CocktailDTO)
    }

    class CocktailRepository
    class IngredientRepository
    class CocktailIngredientRepository

    class Cocktail {
        Long id
        String name
        String category
        String alcoholic
        String glass
        String instructions
        String imageUrl
    }

    class Ingredient {
        Long id
        String name
```

```
}

class CocktailIngredient {
    Long id
    String measure
}

class CocktailDTO {
    Long id
    String name
    String category
    String alcoholic
    String glass
    String instructions
    String imageUrl
}

class CocktailListItemDTO {
    Long id
    String name
    String alcoholic
    String imageUrl
}

%% Beziehungen
CocktailController --> CocktailService : calls
CocktailService --> CocktailRepository : uses
CocktailService --> IngredientRepository : uses
CocktailService --> CocktailIngredientRepository : uses
CocktailService --> CocktailMapper : uses

CocktailMapper ..> Cocktail : maps
CocktailMapper ..> CocktailDTO : maps
CocktailMapper ..> CocktailListItemDTO : maps

CocktailRepository ..> Cocktail : JpaRepository
IngredientRepository ..> Ingredient : JpaRepository
CocktailIngredientRepository ..> CocktailIngredient : JpaRepository

Cocktail "1" o-- "many" CocktailIngredient : has
CocktailIngredient "many" --> "1" Ingredient : uses
```

---

## Docker-Setup für PostgreSQL

Das Projekt nutzt PostgreSQL in einem Docker-Container.

Die Konfiguration ist in der Datei **docker-compose-cocktails.yml** enthalten.

### **1** Docker-Container starten

```
docker compose -f docker-compose-cocktails.yml up -d
```

- Erstellt einen Container mit dem Namen `cocktail_postgres`
- Nutzt die Zugangsdaten aus `application.properties`
- Standard-Port: **5433**

## 2 Umgebungsvariablen (aus `docker-compose-cocktails.yml`)

```
environment:  
  POSTGRES_DB: cocktail  
  POSTGRES_USER: postgres  
  POSTGRES_PASSWORD: postgres
```

## 3 Container stoppen

```
docker compose -f docker-compose-cocktails.yml down
```

## 4 Initialdaten

Beim ersten Start wird das SQL-Skript `data.sql` ausgeführt und befüllt die Datenbank mit Beispieldaten.

💡 **Hinweis:**

Falls die Initialdaten erneut geladen werden sollen, muss man das zugehörige Docker-Volume löschen:

```
docker volume rm <volume-name>
```

(Ersetze `<volume-name>` durch den Namen aus `docker volume ls`, z. B. `cocktail_postgres_data`)

---

## API-Endpunkte

### ◇ Interne CRUD-API (`/api/cocktails`)

Methode	Pfad	Beschreibung
GET	<code>/api/cocktails</code>	Liste aller Cocktails (Paged)
GET	<code>/api/cocktails/{id}</code>	Einzelcocktail abrufen
POST	<code>/api/cocktails</code>	Neuen Cocktail anlegen
PUT	<code>/api/cocktails/{id}</code>	Cocktail aktualisieren
DELETE	<code>/api/cocktails/{id}</code>	Cocktail löschen

---

## Tests

Das Projekt enthält **6 Unit- und Integrationstests**:

- `CocktailRepositoryTest` (Suchlogik)
- `CocktailServiceTest` (Fehlerfälle)
- `CocktailMapperTest` (Core-Mapping)
- `CocktailControllerCrudTest` (CRUD-Controller)
- `CocktailPersistenceTest` (JPA Persistenz)
- `CocktailBackendApplicationTests` (Testet das fehlerfreie Starten der Anwendung und das Zusammenspiel der Komponenten)

Tests ausführen:

```
mvn test
```

---

## Technologie-Stack

- **Java 21**
- **Spring Boot 3.5.x**
- **Spring Web**
- **Spring Data JPA**
- **PostgreSQL**
- **H2 Database (Tests)**
- **Jakarta Validation**
- **Swagger / Springdoc OpenAPI**
- **Maven**

---

## Hilfestellungen

- [Tutorialspoint](#) wurde als Nachschlagewerk für verschiedene Java- und Spring-Boot-Konzepte genutzt. Die Erklärungen dort inklusive Codebeispielen waren sehr hilfreich.
- ChatGPT wurde genutzt als:
  - Strukturierungshilfe beim modularen Aufbau des Projekts
  - Unterstützung bei der Fehlersuche allgemein, z. B. bei Fehlermeldungen im Zusammenhang mit Spring Boot, JPA und Maven
  - Hilfe bei der Gestaltung der API-Struktur und Testklassen
  - Hinweise zu Best Practices im Umgang mit Docker, PostgreSQL und H2 für Tests
- Google Classroom (Kurse 295\_BE 2-7) wurde als Nachschlagewerk für die Vorgaben und den Aufbau des Projekts genutzt.
- Zwei Markdown-Cheatsheets ([Allgemein](#) und [Mermaid-Diagramm](#)) von Github wurde für diese Dokumentation als Nachschlagewerke benutzt.