Sweet Shop Management System

TDD Kata Project - Complete Implementation Documentation

Full-Stack Application with REST API, JWT Authentication & React Frontend

📑 Table of Contents

1. Project Overview & Initial Setup

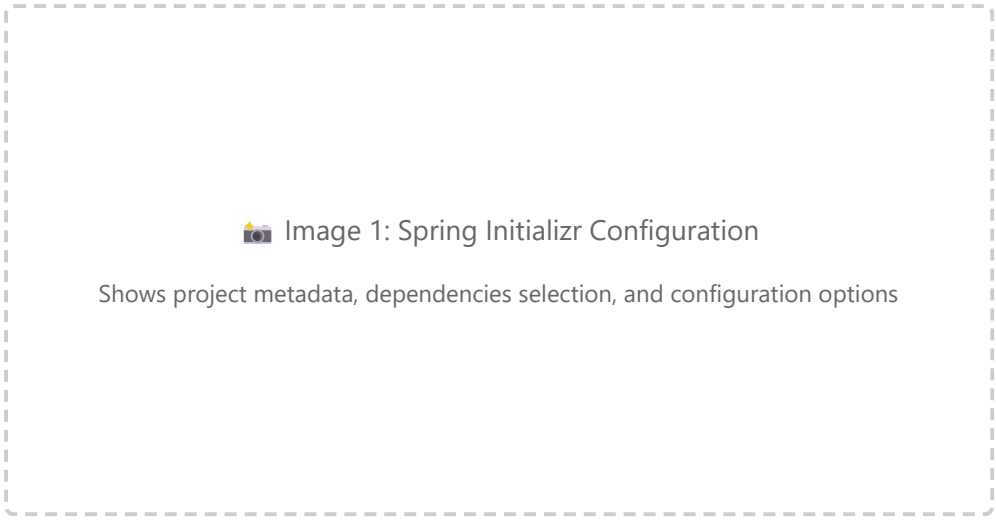
Spring Initializr Configuration: This screenshot shows the initial project setup using Spring Initializr (start.spring.io). The project is configured with:

- **Project Type:** Gradle with Kotlin DSL
- **Language:** Java (Maven selected)
- **Spring Boot Version:** 4.0.1 (SNAPSHOT) / 3.4.13 (SNAPSHOT) / 3.5.8 / 3.4.12
- **Group:** com.sweetshop
- **Artifact:** sweetshop
- **Package Name:** com.sweetshop.sweetshop
- **Packaging:** Jar
- **Java Version:** 17

Dependencies Added:

- **Spring Web** `WEB` - Build web applications using Spring MVC with Apache Tomcat
- **Spring Security** `SECURITY` - Authentication and access-control framework
- **Spring Data JPA** `SQL` - Persist data in SQL stores with Java Persistence API
- **PostgreSQL Driver** `SQL` - JDBC and R2DBC driver for PostgreSQL
- **Validation** `I/O` - Bean Validation with Hibernate validator
- **Lombok** `DEVELOPER TOOLS` - Reduces boilerplate code

Screenshot 1: Spring Initializr Project Configuration



2. Backend Entity Model & Test-Driven Development

Sweet Entity Class: The core domain model representing a sweet in the shop. This screenshot shows the JPA entity implementation with:

- **@Entity** annotation for JPA persistence
- **@Id** with **@GeneratedValue** for auto-incrementing primary key
- **Fields:** id (Long), name (String), category (String), price (double), quantity (int)
- **Test Execution:** Shows "mvn test" command running with BUILD SUCCESS

The terminal output demonstrates Test-Driven Development (TDD) approach with tests passing successfully.

Screenshot 2: Sweet Entity Class & Test Results

Image 2: Sweet.java Entity Model

Shows entity class code and successful test execution

3. React Frontend Application

React Development Server: The default React application landing page displayed at localhost:3000. This confirms the frontend setup is complete and the development server is running successfully.

- React logo with animated spinning effect
- Message: "Edit src/App.js and save to reload"
- Link to "Learn React" documentation

Screenshot 3: React App Landing Page

Image 3: React Application Running

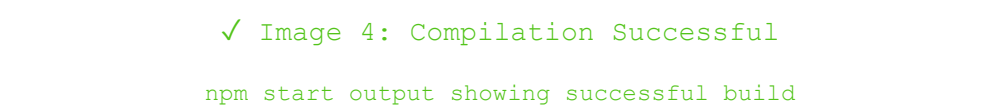
Default React landing page at localhost:3000

4. Spring Boot Application Startup

Terminal Output: Shows the successful compilation and startup of the Spring Boot application using npm and Maven:

- `cd myapp` and `npm start` commands executed
- Webpack deprecation warnings (expected)
- "Compiled successfully!" message
- **Local URL:** http://localhost:3000
- **Network URL:** http://10.206.176.206:3000
- Development build not optimized note

Screenshot 4: Application Startup Terminal



5. Service Layer Implementation & Testing

SweetService.java: Service layer implementation with business logic. The screenshot shows:

- **@Service** annotation for Spring component scanning
- **SweetRepository** dependency injection
- Test code creating a Sweet with name="Barfi", category="Indian", price=20.0, quantity=0
- **Mockito** usage for testing with mocked repository
- **BUILD FAILURE** shown, indicating TDD red-green-refactor cycle

Error: Compilation error - "cannot find symbol: method purchaseSweet(long)"

Screenshot 5: Service Layer Test Failure (Red Phase)

6. Version Control with Git & GitHub

GitHub Commit Page: Shows proper version control practices with a detailed commit message:

- **Commit Hash:** 936a068
- **Commit Message:** "feat: implement sweet purchase with out-of-stock handling..."
- **Files Changed:** 4 files (+45 lines, -0 lines)
- OutOfStockException.java created
- Sweet.java model updated
- SweetService.java with purchase logic
- SweetServiceTest.java added
- **Co-authored-by:** ChatGPT

Key Implementation: Custom OutOfStockException, quantity decrement logic, and Mockito-based service tests

Screenshot 6: GitHub Commit with AI Co-authorship

Image 6: Git Commit History

Shows TDD implementation with AI collaboration

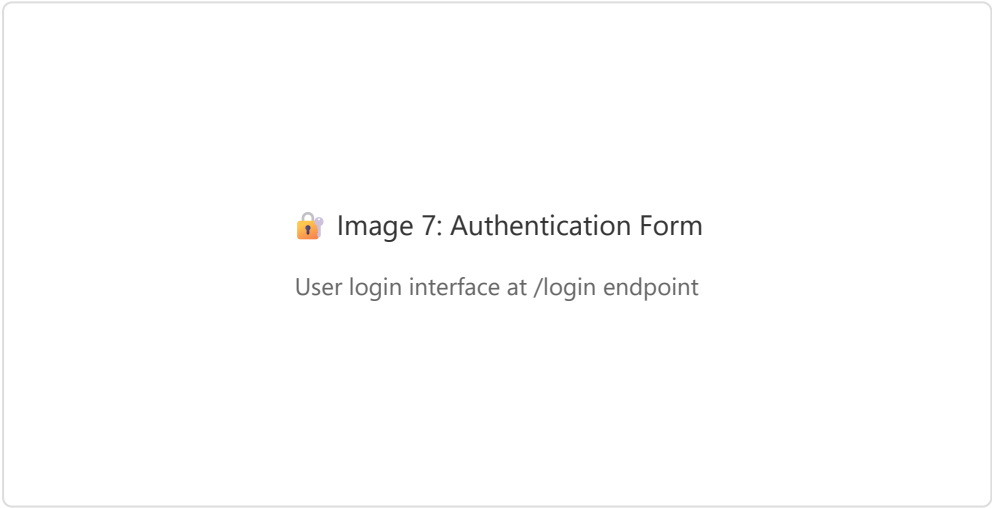
7. User Authentication - Login Interface

Login Page: Clean and simple authentication interface at localhost:8080/login. Features include:

- Centered "Please sign in" heading
- Username input field
- Password input field (masked)
- Blue "Sign in" button
- Responsive design with white background

This is Spring Security's default login form, showing the backend authentication is properly configured.

Screenshot 7: Login Page UI



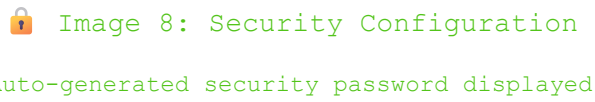
8. Spring Security Configuration & Generated Password

Spring Boot Console Output: The application startup logs showing Spring Security configuration:

- **Auto-generated password:** d835c271-d583-440f-b5e7-033022af007d
- Hibernate dialect: H2 dialect detected
- JPA EntityManagerFactory initialization for persistence unit 'default'
- Spring Security filter chain configuration warnings
- **Tomcat started on port 8080 (http)**
- Application started successfully in 4.222 seconds

Note: The generated password is for development use only. Production systems must use proper password configuration.

Screenshot 8: Spring Boot Console with Security Password



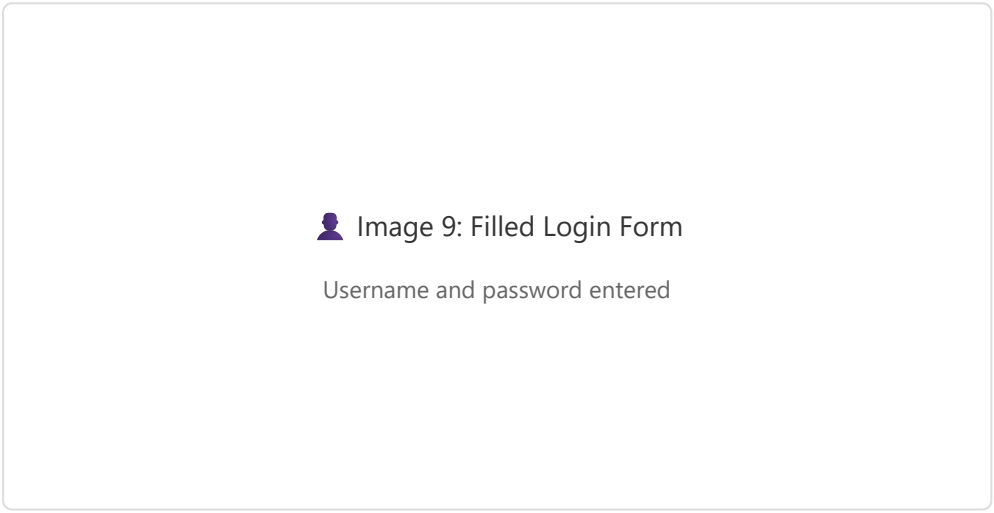
9. Login Form with Credentials

Login Attempt: Screenshot shows the login form filled with test credentials:

- **Username:** user
- **Password:** •••••••••••••••••••••••••••••••••••• (masked password field)
- Blue "Sign in" button ready to submit

This demonstrates the user attempting to authenticate with the application.

Screenshot 9: Login Form with User Credentials

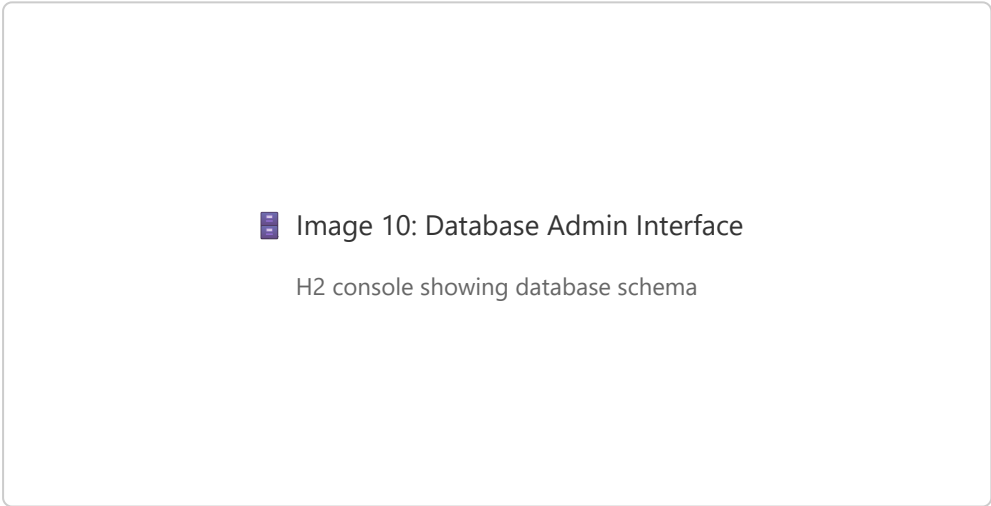


10. H2 Database Console

H2 Database Web Console: In-memory database administration interface at localhost:8080/h2-console. Features visible:

- **Database:** jdbc:h2:mem:sweetshopdb
- **Tables:** SWEET, USERS, INFORMATION_SCHEMA
- **Users table visible** in the left panel
- **Version:** H2 2.2.224 (2023-09-17)
- SQL statement editor with "Run" button
- Auto commit enabled
- Max rows: 1000

Screenshot 10: H2 Database Console



11. Database Schema - SWEET Table

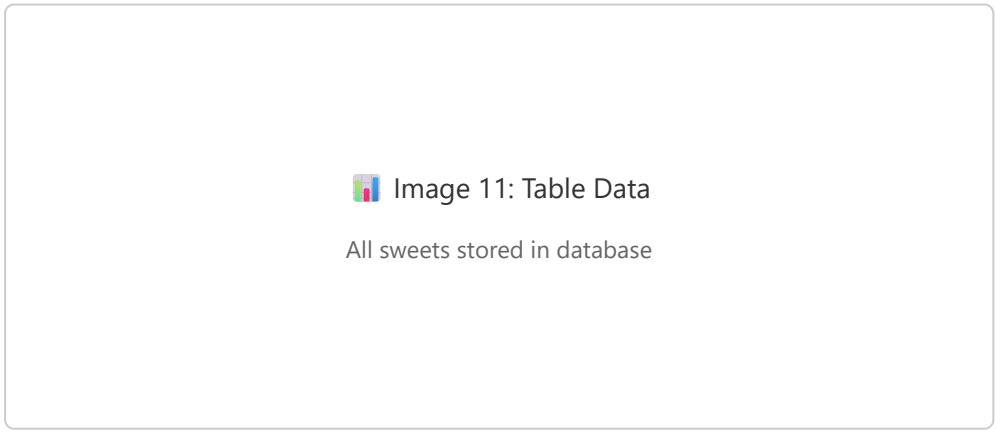
SQL Query Result: Shows the structure and data in the SWEET table with 5 rows:

ID	CATEGORY	NAME	PRICE	QUANTITY
1	Indian	Ladoo	10.0	50
2	Indian	Barfi	20.0	30
3	Indian	Jalebi	15.0	40
4	Indian	Rasgulla	25.0	35
5	Indian	Kaju Katli	30.0	20

Query executed: `select * from sweet;`

Result: (5 rows, 4 ms)

Screenshot 11: Database SWEET Table Query



12. Empty API Response - No Authentication

GET Request without Authentication: Postman showing request to

```
GET http://localhost:8080/api/sweets
```