Full-stack Web Shop

MySQL, Spring Boot, Vue.js

bike X optimum

Introduction

This documentation provides a comprehensive overview of the "Bike X Optimum" web shop, a collaborative project developed by a team of three students. The web shop is built on the MySQL database management system, utilizing the Spring Boot framework for the backend and Vue.js for the frontend development.

Team members:

- Filip Stanojković – Singidunum University Faculty of Informatics and Computing (Факултет информатике и рачунарства Универизета Сингидунум)
- Dimitrije Rankov – University of Belgrade Faculty of Organizational Sciences (Факултет организационих наука Универзитета у Београду)
- Igor Stanković - College of Information and Communication Technology (Висока школа за информационе и комуникационе технологије)

28/05/2025

System Architecture

The system architecture of the web shop is structured as follows:

Frontend Components:

- Vue.js (Vue2) is utilized for the frontend development.
- CRUD operations are implemented for user account management, including account creation, deletion, password modification, and bike purchase.

Backend Components:

- Spring Boot is employed for the backend development.
- The backend consists of controllers, entities, services, and repositories.

Controllers:

- Controllers act as intermediaries between the user interface (UI) and the backend logic of an application.
- They receive incoming requests from the frontend, process them, and formulate appropriate responses.
- Controllers typically contain methods (or actions) that handle specific HTTP requests (e.g., GET, POST, PUT, DELETE).
- In the context of a web shop, controllers manage various functionalities such as user account management, product listings, and transaction processing.

Entities:

- Entities represent the core data objects or business entities within an application.
- They encapsulate the structure and behavior of real-world concepts that the application needs to model and manipulate.
- Entities often correspond to database tables, with attributes representing the properties of the entity and methods defining its behavior.
- In a web shop environment, entities could include User, Bike, and Transaction, each representing a distinct concept relevant to the business domain.

Services:

- Services encapsulate the business logic of an application that isn't specific to any particular user interface or data access mechanism.
- They define operations or workflows that involve multiple steps and may require coordination between different parts of the system.
- Services abstract away complex logic from controllers and facilitate code reuse and maintainability.
- In a web shop scenario, services might handle tasks such as user authentication, order processing, and inventory management.

Repositories:

- Repositories provide an abstraction layer for accessing and manipulating data stored in a persistent storage mechanism, such as a database.
- They encapsulate the details of data access, including querying, saving, updating, and deleting entities.
- Repositories promote separation of concerns by isolating data access code from the rest of the application logic.
- In the context of a web shop, repositories would interact with the underlying database tables (e.g., User, Bike, Transaction) to perform CRUD operations and manage data persistence.

The interaction between the frontend and backend components is orchestrated through HTTP requests, with Vue.js handling the presentation layer and Spring Boot managing the business logic and data access. The CRUD operations ensure seamless user interaction for account management and bike purchasing functionalities.

Database Schema

Overview:

The database contains three main tables: bikes, transactions, and users. These tables store information about available bikes, transaction history, and user accounts, respectively.

Tables:

1. ***bikes:***

- bike_id: INT (Primary key; Unique)
- model: VARCHAR(255) (Name or model of the bike)
- price: DOUBLE (Price of the bike)
- img_path: VARCHAR(255) (Path to the image of the bike)

2. ***transactions:***

- transaction_id: INT (Primary key; Unique)
- buyer: INT (Foreign key referencing the user_id of the buyer)
- product: INT (Foreign key referencing the bike_id of the purchased bike)
- datetime: TIMESTAMP (Date and time of the transaction)

3. ***users:***

- user_id: INT (Primary key; Unique identifier for each user)
- email: VARCHAR(255) (Email address of the user)
- password: VARCHAR(255) (Encrypted password of the user)
- balance: DOUBLE (User's balance or available funds)
- role: VARCHAR(10) (Role of the user; e.g., 'admin', 'user')

Relationships:

The transactions table has foreign key constraints referencing both the users and bikes tables. This ensures referential integrity, i.e., each transaction is associated with valid user and bike IDs.

- The buyer column in the transactions table references the user_id column in the users table, linking each transaction to the respective buyer (@ManyToOne).
- The product column in the transactions table references the bike_id column in the bikes table, indicating the bike purchased in each transaction (@ManyToOne).

API Endpoints

Bike Controller:

- *GET /bikes*

Retrieves a list of all bikes.

- *GET /bike/id/{id}*

Retrieves a bike by its unique identifier.

- *GET /bike/model/{model}*

Retrieves a bike by its model.

Transaction Controller:

- *POST /addTransaction*

Creates a new transaction.

- *GET /transactions*

Retrieves a list of all transactions.

- *GET /transaction/id/{id}*

Retrieves a transaction by its unique identifier.

User Controller:

- *POST /addUser*

Creates a new user.

- *GET /users*

Retrieves a list of all users.

- *GET /user/id/{id}*

Retrieves a user by their unique identifier.

- *GET /user/email/{email}*

Retrieves a user by their email address.

- *PUT /update*

Updates an existing user.

- *DELETE /delete/id/{id}*

Deletes a user by their unique identifier.

Refer to the API documentation for detailed information on request and response formats for each endpoint.

Deployment

Github Repository:

https://github.com/44filip/project-x

Database:

Import the data found in /db_dump to your relational database of choice (project choice is MySQL).

Frontend:

Through a terminal, navigate to .\frontend\ by typing “cd .\frontend” then execute the following commands in order.

- “npm install”
- “npm run serve”

Backend:

Run BackendApplication.java found in “.\backend\src\main\java\rs\ac\singidunum\backend\BackendApplication.java” either through a Java environment or through the terminal (the execution will vary from system to system).

Browser:

Navigate to “localhost:8080” in the address bar.

Thank you for reading through the Project X documentation.