Supreme Bookstore - BDS Project 3

Adam Kmeť, Benedek Komzsík

Contents

1 Introduction

The application which we have developed is called Supreme Bookstore. The core of the application is based on the created database from the first project assignment. During the first assignment, we created tables related to book store. Tables like customer, address, book, author, contact, and so on. Then we created an application that allows customers to buy these books and create their own profile.

1.1 Development Preparations

This application was developed in Java. It is an object-oriented, platform-independent programming language that is designed to create robust and versatile software applications. We used IntelliJ IDEA as our Integrated Development Environment. We also used the JavaFX SDK, which is used for building and deploying modern graphical user interfaces (GUIs) in Java. And then we built the project using Maven. Which is a great build automation tool used mainly for Java projects. It is based on the concept of POM (Project Object Model), which centralizes project configuration and provides a streamlined way to build, manage dependencies, and deploy applications. The most important is the pom.xml file, which is the heart of the Maven project, containing all the configurations and dependencies.

1.2 PreparedStatements

This is a feature of Java Database Connectivity API, which is used to interact with databases. The main key aspects are precompilation for performance, where the SQL statement is precompiled when the PreparedStatement is created. It reduces the overhead when executing the same statement for more times. PreparedStatements also provide prevention of SQL injections; it uses placeholders for parameters, which are later replaced by real values. And since the SQL is precompiled, we can execute the same statement more times with different values without recompiling. It also allows us to dynamically set parameters using methods like `setInt()` or `setString()`, which makes it easier to work with parameterized queries.

1.3 Database Changes

To support functionalities in the application, the following changes were made to the database schema:

Customer Table

- Added `username` column to store the username of the customer.
- Added `password` column to store the password of the customer.

Book Table

- Added `price` column so the customers knew the price of the book they wanted to buy.

2 Application Code Structure

It's important to understand the code structure and see how our application is implemented. So we have a better understanding of how the whole process functions.

Our code initialization starts with `Main.java` and `App.java` files.

- `Main.java` acts as the true entry point of the application. It sets up the database connection, ensuring backend (like the database) resources are prepared.
- `App.java` starts the JavaFX user interface after the backend has been initialized.

2.1 Application Programming Interface

The API in our application serves primarily as data containers; they make it easy to move information between different parts of the app. Each API file represents a specific type of data (like a customer, a book, or an order). In our App, we have 5 files.

- `BookView.java` file represents all the details about a book, like its name, ISBN, number of pages, release year, and author information.
- `CustomerCreateView.java` handles the information that are needed to create a new customer account, specifically their username and password.
- `CustomerView.java` represents detailed information about a customer.
- `OrderView.java` stores details about an order, like their ID, which customer placed it, when it was made, and what kind of book was ordered.
- `SignInView.java` this just handles login information, checks whether the credentials are correct when they try to log in.

2.2 Configuration File

Here, we have the `DataSourceConfig.java` file, which is responsible for configuring and managing the application's database connections. The way it works is that it sets up the connection using HikariCP, which is a high-performance JDBC connection pooling library; it ensures efficient and reliable access to the database throughout the application. We also have the logging classes from SLF4J in this file, which are used for logging messages about the configuration and potential errors. Included is also the properties class, which is meant for loading properties files that contain database configurations like username and password.

2.3 Controllers

Controllers are a crucial piece of the code; they are part of the MVC design pattern and act as intermediaries between the view (UI) and the model (data and logic). The main purpose of controllers is to handle user input, like clicking a button, and determine what needs to happen next. They take the input from the user and then send it to services or repositories for processing, and then update the user interface with the results. We have 7 controllers in total in our application.

- `AccountController.java` file manages actions and data related to user's account. It interacts with the `CustomerRepository` and is responsible for being able to display user information, add new account-related data, manage edits to contact and address details and to view shopping cart.

- `BookDetailController.java` controller is responsible for displaying book details and handling the purchase process. It ensures that only logged-in users can buy books, that the orders are recorded in the database and that the users can select the shipping options.
- `BookStoreController.java` is about displaying a list of books for users to browse. It also includes a search functionality, which filters books by user input. It also allows for user to click on certain book and showcase its details about it, information like author, release year and whatnot. And lastly it displays the sign in button and after signing in the profile button.
- `CartController.java` dynamically displays the logged-in user's order history in the cart view by fetching order data from the `OrderRepository` and creates labels for each order and adds them to the UI.
- `ContinueController.java` is only responsible for transitioning the user from the introductory screen to the Bookstore.
- `CreateAccountController.java` allows users to register new accounts by inputting username and password.
- `SignInController.java` handles signing in, it verifies credentials and opens the user profile upon success, it also displays user's account view and provides access to the account registration view. Shows appropriate dialog for successful and unsuccessful actions.

2.4 Repositories

Next part of the code are repositories, they are classes responsible for interacting with the database. They handle all the querying, saving, updating, inserting, or deleting data. Within this application, we have 3 repositories.

- `CustomerRepository.java` class handles all database interactions which are related to customers and performs CRUD operations. It includes tables like customer, contact or address, where we can insert, update or retrieve this information.
- `BookRepository.java` class provides data access for books in the database. It similarly does CRUD operations for books using SQL queries. We can retrieve detailed information about the books or search for them by name.
- `OrderRepository.java` is responsible for managing database operations related to orders. Repository can create new customer orders, linking them to books and shipping methods. It can also retrieve orders for specific user and provide detailed view of all orders in the system

2.5 Code Essentials

Furthermore, the code includes exceptions, which are used to handle unexpected issues that might arise during the execution of our application. The main purpose of them is to ensure that errors are detected and logged, allowing us to gather feedback.

- `DataAccessException.java` is here to encapsulate database-related errors, it occurs when database operations like insert or update fails. Or if there is a database connection issue.
- `ExceptionHandler.java` class centralizes the exception processing, ensuring consistency in how errors are handled and displayed. It uses JavaFX dialog that includes

a brief explanation of the issue. Stack trace provides detailed technical information for debugging.
- `ResourceNotFoundException.java` class is simply handling scenarios where a requested resource cannot be found. Like missing database entries or invalid identifiers.

Further down we have some utility and security-related classes.

- `Argon2Service.java` provides configured instance of Argon2 hashing algorithm. It provides reliable hashing and verification of passwords.
- `AuthService.java` is responsible for authentication in the application. It integrates with the `CustomerRepository` to fetch user data and uses the `Argon2Service` for secure password verification.
- `CustomerService.java` purpose of this class is to handle customer creation and ensure secure password hashing before storing the credentials in the database.

It is also very important to talk about the FXML files in our project. They are the XML-based files that define the Graphical User Interface (GUI) for us. They separate the UI design from application logic, allowing our application to have a clean structure. We used Scene Builder by Gluon; it helps us easily define the layout of the buttons, labels, and text fields.

Part of the application is also the usage of CSS, which is a stylesheet language used to describe the presentation of a document written in HTML or XML. It basically makes our application look better, allows us to apply styles like colors, layouts, and fonts, which is increasing the visual appeal of the whole application.

3 Application Utilization

The application serves as an online bookstore platform, allowing users to browse and search for books. Each book provides details about it, like its author, release date, price, and so on. The application allows users to purchase a book by adding it to an order and choosing shipping options. This requires signing up; after doing so, customers can customize their profiles with information that can include contact and address. Users can also view their order history to see their previous purchases.

3.1 Environment Set Up

Before we can dive into the application itself, we need to set up the environment. After we created the code, we need to build the project; we need Maven installed and imported. After doing so, we can simply execute the command: `mvn clean install` in the IntelliJ terminal. This will build the application. If the build was successful, we can initialize the application using the command: `java -jar .\target\bds-app-1.0.0.jar`.

3.2 User Experience

After we boot up the application, we are welcomed by our intro screen, which contains our logo with our name, and a button that says “Continue”. After pressing the button, a new window opens. This window has a list of available books in the store. We can scroll through them and look for the one that the customer is looking for, or we can simply use the “Search” functionality, and we can input the name of the book so it filters and we don’t have to look for it manually. Whenever we find the desired book, we can click on one, and we can see its details, like its author, the number of pages, ISBN, and release year. There is also a button that says “Buy” under these information. Whenever we click this button, a new window opens up which is a selection screen with shipping options; we can choose between GLS, UPS, Slovenska posta, Zasielkovna, or Odber na mieste. After we choose an option, we can press “Confirm”, and we get a pop-up window that says “Book successfully bought!”. The system will not allow us to buy the books unless we sign in first.

If we don’t have an account created, we can do so by pressing the “Create Account” text field, which opens up a new window where we can create an account. We enter the username and password and then press the “Create new account”. Now that the account was created, we can Sign In, where we insert the same username and password. After we successfully insert the right username and password, we get a pop-up that says “You were successfully logged in”. We can press OK and we are introduced to the User Profile window. Here we have multiple options. We can customize this User Profile. Once we press the “Insert Data” button, we can add details to our profile. Our Email, Telephone, Postal Code, Street, and House Number. After inserting the information, we also have the option to edit it. After we press “Edit Data” we get a pop-up window that asks us what data we want to change, whether contact data which is the email and phone, or our address, which is the remaining data. The last button on our User Profile page is the “View Cart”. After clicking on it, we get a pop-up with our orders in it.