UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

Object Oriented Programming

5COSC019C

Coursework

Name - Nirodha Adithya Perera

UOW Number - W1953247

IIT Number - 20223021

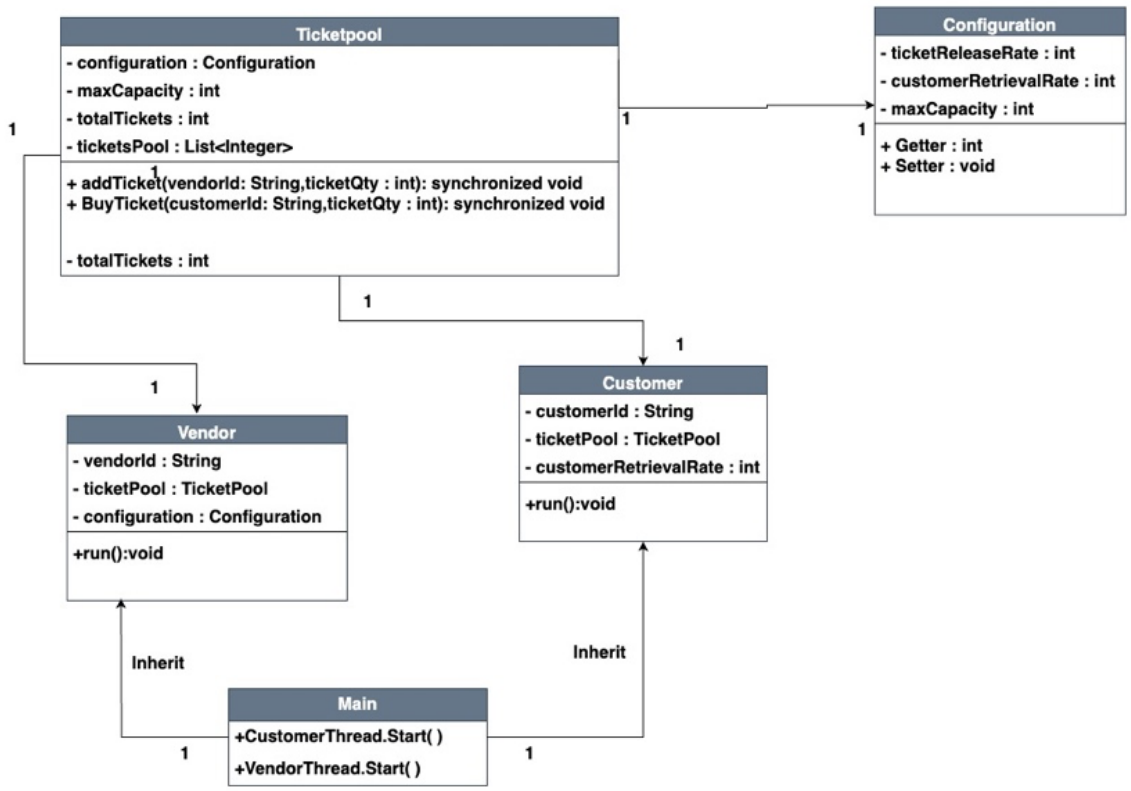
Introduction

The Real-Time Ticket Handling System is a modern web development platform that simplifies ticket purchasing and selling in a dynamic environment. It is built with Java Spring Boot on the backend and Angular on the frontend, and it follows Object-Oriented Programming (OOP) concepts and the Producer-Consumer concept.

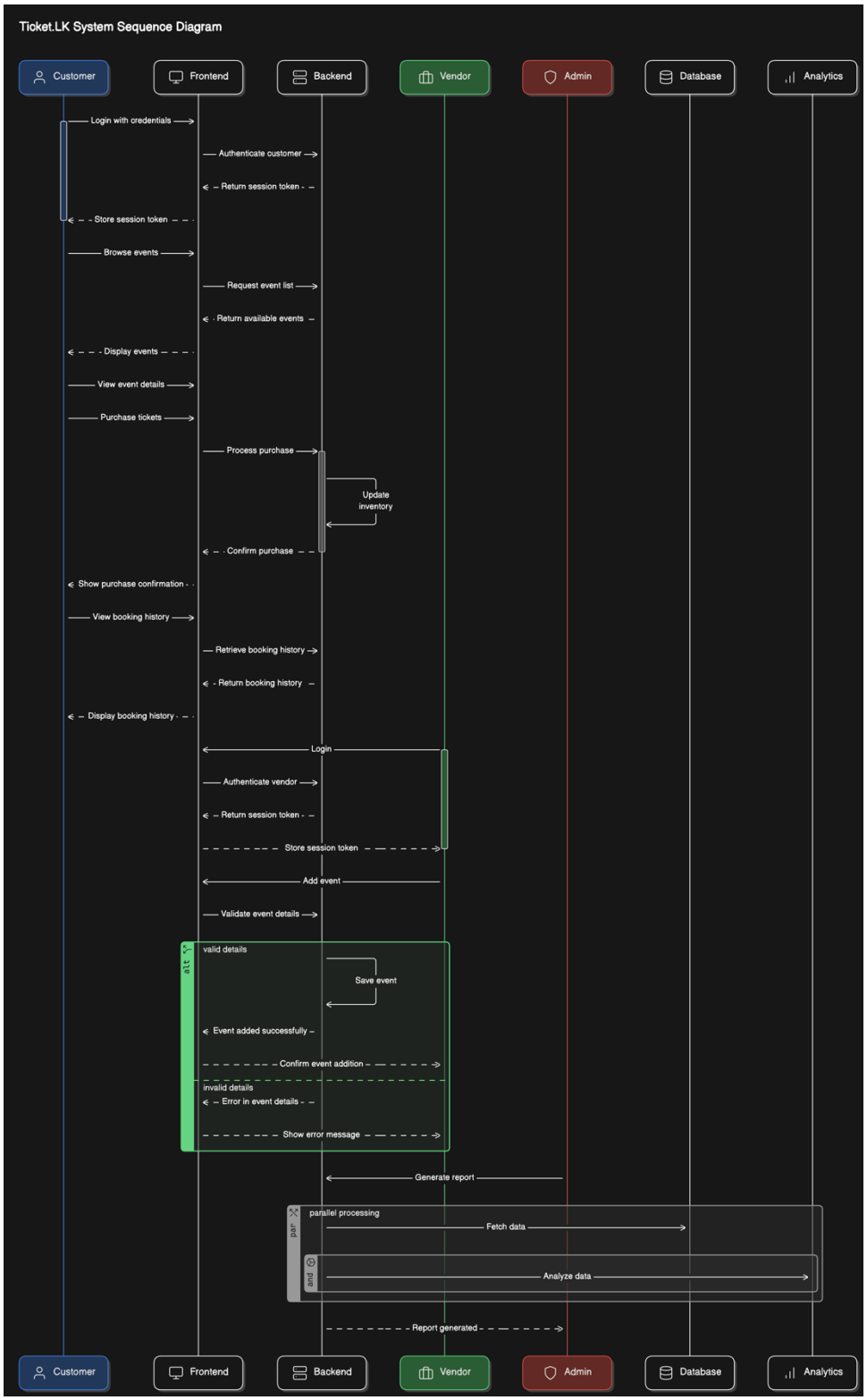
Vendors work as producers, releasing tickets, and customers work as consumers, purchasing available tickets. This design pattern ensures that ticket releases and purchases go smoothly, while maintaining confidentiality and simplicity. Multithreading is used to manage several ticket release and purchase events simultaneously, ensuring real-time updates and responsiveness in a dynamic ticketing system. The system also includes tools for managing customer and vendor data, modifying configuration parameters.

Diagrams

Class Diagram :



Sequence Diagram :



Test Case

Vendor Dashboard Test

- The `VendorController` facilitates vendor-related operations such as managing tickets, events, and viewing details of ticket purchases. The `addTicket` endpoint allows vendors to add tickets to the pool by providing the vendor ID and a `TicketDto` object. Before processing, the endpoint verifies if the system is active using the `SystemToggleDto`. If the system is active, the ticket is added, and a success message is returned. If the operation fails or the system is inactive, an appropriate failure message is sent.

- The saveEvent endpoint enables vendors to create new events. By providing the vendor ID and the event details in an EventDto, the system saves the event and returns the created event details with an HTTP status of 201 Created. Vendors can retrieve a list of all events using the getEventList endpoint, while the getEventListById endpoint retrieves events specific to a vendor based on their ID.

- The deleteEvent endpoint allows vendors to delete an event by providing its ID. A success or failure message is returned based on whether the event exists. Additionally, vendors can view details of purchased tickets through the getDetailsList endpoint, which returns a list of PurchaseTicket objects containing relevant details. This controller ensures that vendors can efficiently manage tickets, events, and view transactional details while adhering to the system's active state.

Admin Dashboard Test

- The AdminController provides various endpoints for managing system configurations and toggling the system's active state. The createConfiguration endpoint allows admins to create a new configuration by sending a ConfigurationDto object as the request body, which is then saved and returned with an HTTP status of 201 Created. The getConfigurationById endpoint retrieves a specific configuration by its ID and returns the corresponding ConfigurationDto object with an HTTP status of 200 OK. Admins can fetch a list of all configurations using the getConfigurationList endpoint, which returns a list of ConfigurationDto objects.

- To update an existing configuration, the `updateConfigurationById` endpoint accepts the configuration ID as a path variable and the updated `ConfigurationDto` object as the request body, returning the updated configuration with an HTTP status of `200 OK`. The `deleteConfiguration` endpoint allows the deletion of a configuration based on its ID. It returns a success or failure message in the response body, depending on whether the operation was successful.

- The controller also includes functionality to toggle the system's active state using the `toggleSystem` endpoint. This endpoint takes a boolean query parameter (`isActive`) to activate or deactivate the system, returning a message indicating the new state. Additionally, the `getToggleSystemState` endpoint retrieves the current active state of the system and returns it in the response body. Cross-origin requests are enabled for `http://localhost:4200` to allow integration with the Angular frontend

Customer Dashboard Test

- The `CustomerController` manages the operations related to customers, including purchasing tickets and retrieving event details. The `purchaseTicket` endpoint allows a customer to purchase tickets for a specific event by providing the customer's ID, the event ID, and the number of tickets as request parameters. Before processing the purchase, the method checks if the system is active using the `SystemToggleDto`. If the system is active, the ticket purchase is processed, and a success or failure message is returned depending on the operation's result. If the system is inactive, a message indicating that the system is not currently active is returned.
- The `getEventList` endpoint provides a list of all available events by fetching event details from the `EventService`. Additionally, the `getEvent` endpoint allows customers to retrieve details of a specific event by providing the event ID as a path variable. The response contains the details of the event wrapped in an `EventDto`. This controller ensures seamless interaction for customers to view events and make ticket purchases while adhering to the system's active state.

User Dashboard Test

- The UserController handles user-related operations such as account creation and login. The createUser endpoint allows new users to sign up by sending their details encapsulated in a userDto object in the request body. Upon successful registration, the user details are saved and returned with an HTTP status of 201 Created.

- The loginUser endpoint facilitates user authentication. It accepts the user's email and password as path variables and verifies the credentials using the UserService. If the login is successful, the corresponding user details wrapped in a userDto are returned with an HTTP status of 201 Created. This controller ensures basic user management functionality, including secure account creation and authentication.

Conclusion

In summary, the Real-Time Ticket Handling System, which makes usage of Angular and Java Spring Boot made trustworthy frontend and backend. The system maintains simplicity and confidentiality while guaranteeing smooth ticket releases and purchases through the combination of the Producer-Consumer concept and Object-Oriented Programming principles. Its responsiveness is improved by multithreading, which enables real-time updates and seamless management of multiple events. Furthermore, the development process was greatly simplified by tools like Visual Studio Code and IntelliJ IDEA, which offer strong functionality for coordination, debugging, and code management. This system is a stable and flexible solution that not only simplifies the ticket handling process for clients and vendors but also offers strong data management and system monitoring features.