PGR209 Exam - Fall 2025

Successes and Difficulties

One aspect that worked particularly well was structuring the application according to best practices. We clearly separated responsibilities between controllers, services, repositories, and entities, which made the codebase easier to understand and develop. Using JPA annotations together with Flyway migrations gave us confidence that the database schema was consistent and versioned correctly. Lombok also helped us greatly with reducing redundant time spent on writing boilerplate. Implementing domain rules, such as preventing orders for products that were out of stock, was also a good exercise in placing business logic in the service layer rather than in controllers. Having the entities separated into their own folders made it easy to assign each group member to work on their own parts before putting it all together. Testing was both one of the most valuable and most time-consuming parts of the project. Unit tests helped verify service logic, while integration tests with Testcontainers ensured that our JPA mappings and database interactions behaved correctly against a real PostgreSQL instance. Setting up Testcontainers initially required some trial and error, especially around configuration and startup time, but once in place it significantly increased our confidence in the application. Jacoco coverage reports were useful for identifying untested areas of the codebase.

Some parts of the project were more difficult than expected. We first tried to set a many-to-one relationship between addresses/customers and orders. However, this made us unable to delete addresses or customers connected to an order due to foreign key constraints. We wanted to keep a log of all orders, including to where and to whom it was sent, so we did some research and found that the most common solution was snapshotting the customer/address info to the order without a direct connection in the database. Implementing this for both addresses and customers on order creation enables us to delete user info for GDPR reasons, while keeping a full record history of orders made.

As an extension beyond the core requirements, we implemented a Thymeleaf-based order receipt view. This helped us better understand how a backend API can also serve rendered content, and it added a more complete, realistic feel to the application. Throughout delevopment, we relied on official documentation from Spring and our application dependencies, course materials, and community resources such as Stack Overflow and random blog posts. Overall, this project strengthened our understanding of backend development with Spring and highlighted the importance of clean architecture, testing, and incremental development.