Mom&Pop Online Bookstore

Critical Design and Implementation

April 9, 2021

Prepared for EECS 4413 by

Runtime Terror:
Daniel Kozlovsky
Kevin Nguyen
Jessanth Harrymanoharan
Nina Yanin

Table of Contents

Introduction and Overview

Team Runtime Terror has been commissioned by Mom&Pop Bookstore to create an online website that allows their customer to shop for books online. They want their customers to be able to do things like save a profile, keep a cart, read book info and leave reviews.

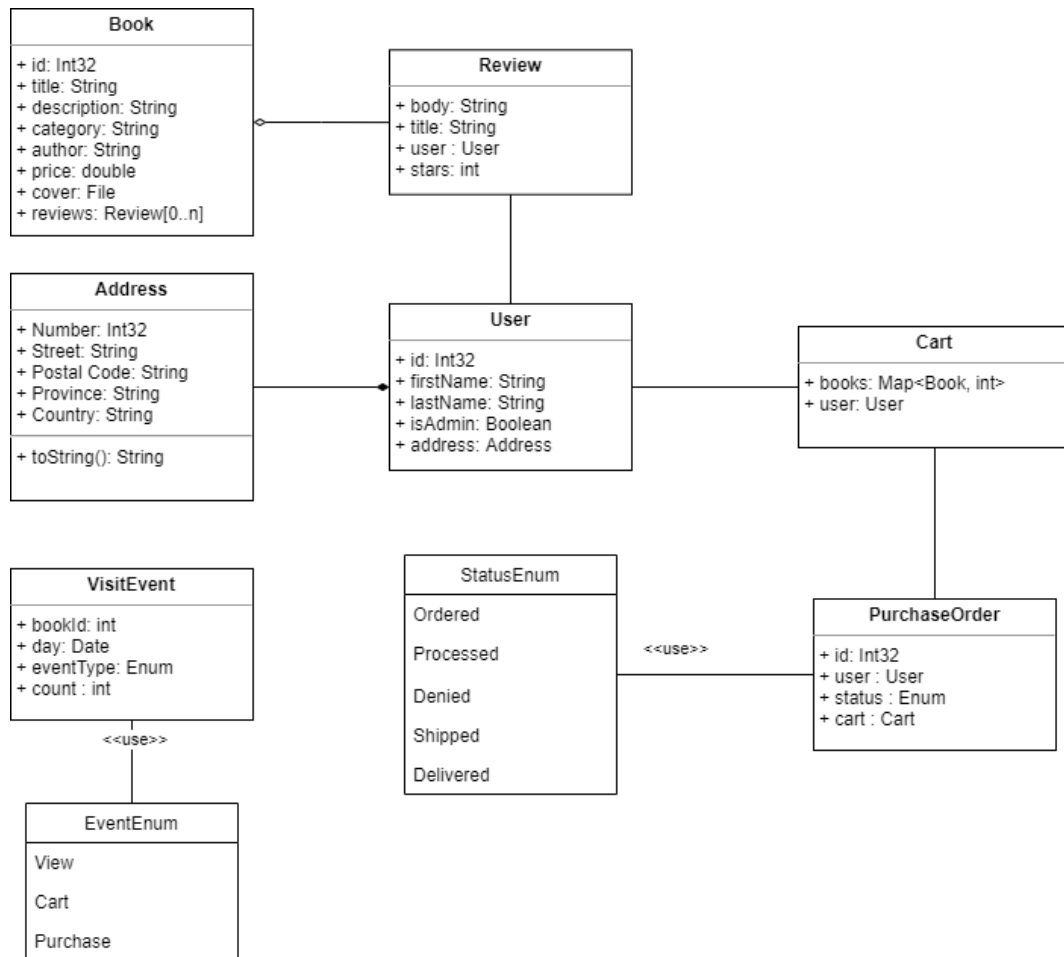# Architecture

To carry out Mom&Pop's requirements, the application is a three-tier client server application that serves online customers through a browser client.
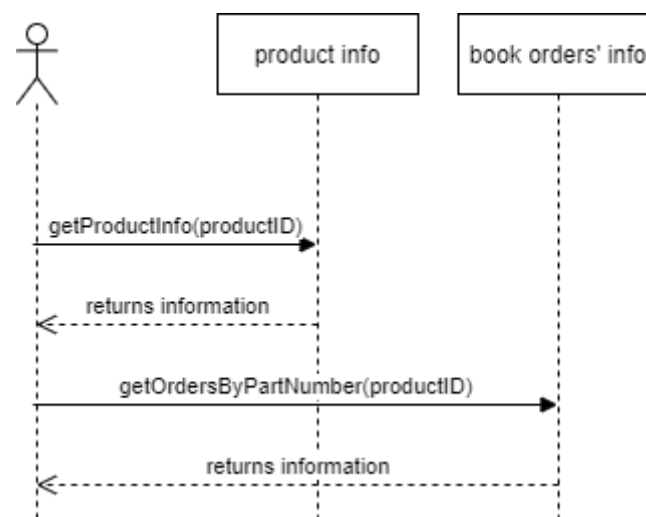
Use Cases
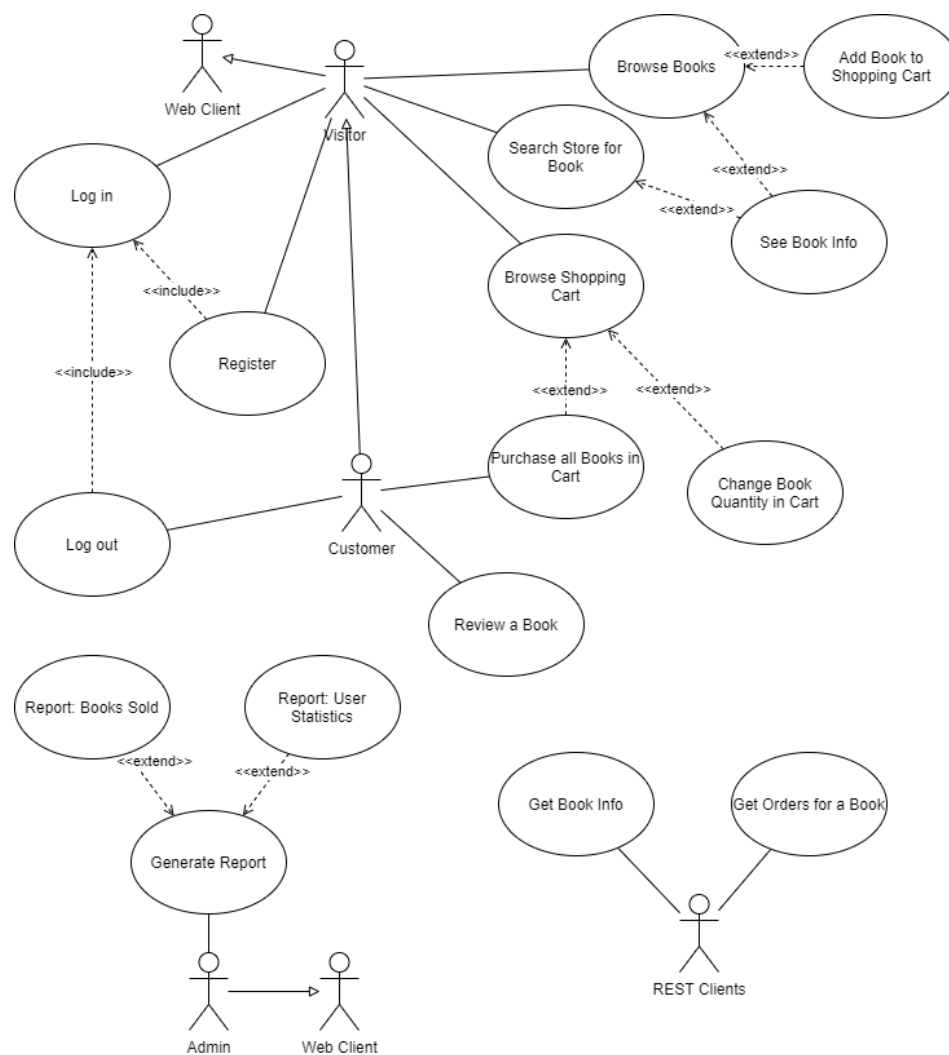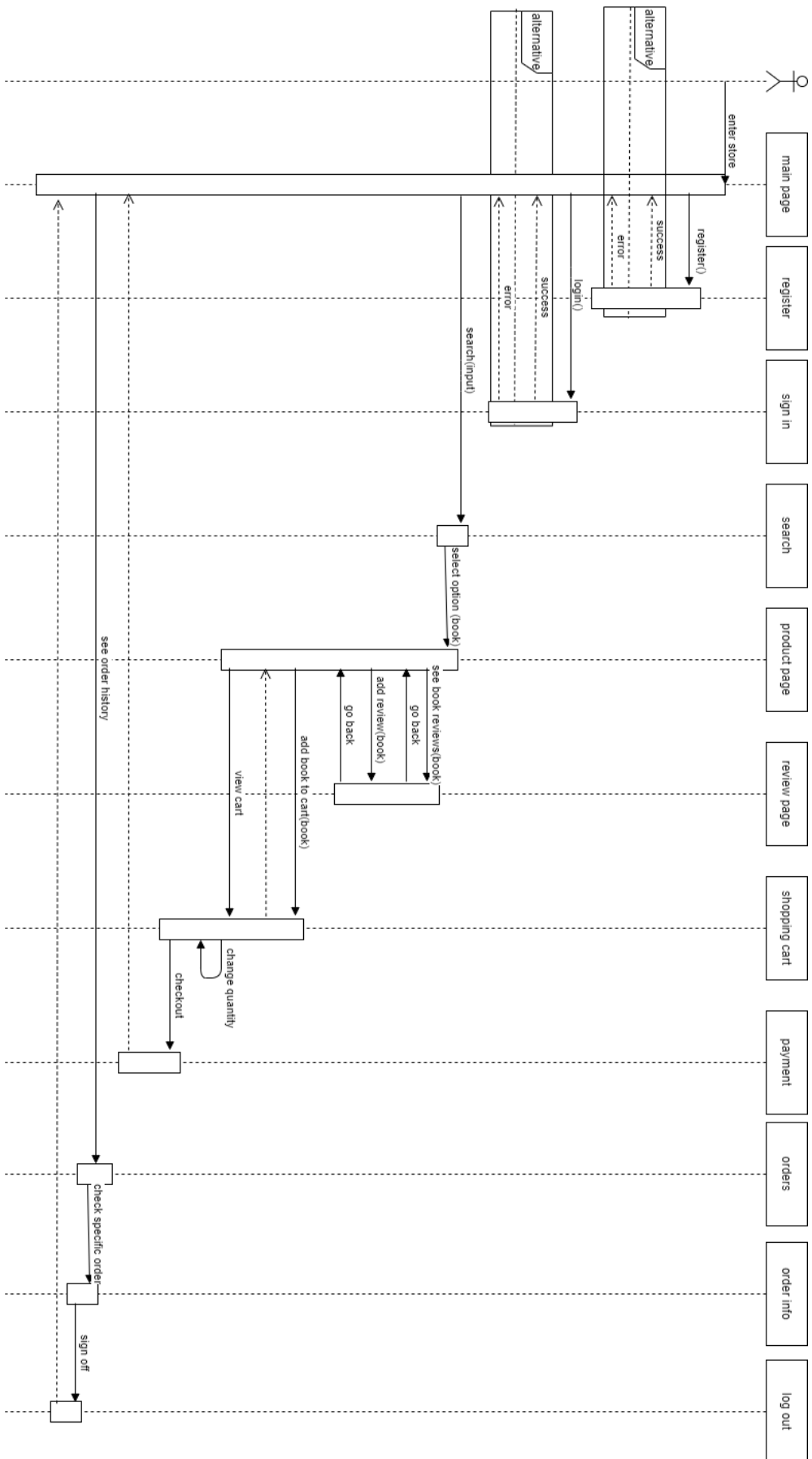

Use Cases for Mom&Pop Bookstore

Class Diagram



Class Diagram of our Data Entities

Sequence Diagrams

Sequence for getting book orders (REST API)

Sequence for main user flow. User purchases a book.

Design

The overall application is built in Java Enterprise Framework and follows a MVC pattern. It uses JSP and HTML for the view, and then Java and Servlets for the model and controller. There are two REST endpoints for use by external web clients. The resulting WAR format can be deployed virtually on any server, cloud or hosted. The following diagram gives an overview of our architecture.
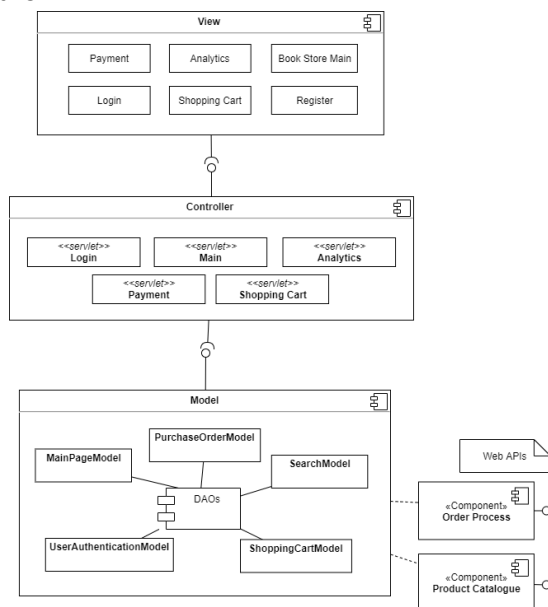
Component Diagram of our Architecture

Pattern Choices

There are a few prominent design choices we made, each with advantages and their trade-offs.

MVC

MVC is a ubiquitous pattern for a reason; it eases development. MVC provides separation of concern which allows us the team to collaborate easier. It is simpler and straightforward to divide work and for each developer to understand what is going on.

The trade-off is that it can be complicated to initially consolidate the features of the application after dividing up the work. It also complicates the flow of the app, making testing and debugging hard in some aspects.

Builder Pattern

To represent our data entity, or "beans", we used a builder pattern to construct the beans. This gave us flexibility in representation of each bean and enabled us to divide work easier

and accommodate many use cases. In addition, it enabled ease of use through the method-chaining nature of the builder pattern.

As a sacrifice, it took a longer and more laborious development effort to implement the builder pattern. It results in a complex system that not every developer can understand or debug. In such a complex system, bugs are prevalent as well. Also, since the builder classes are mutable, it introduces room for error and misuse on other developers' sides.

Multiple Structural Facade DAOs

Instead of a single data access object (DAO), we opted to create a structural facade of our DAO component using multiple DAOs. The actual DAO component is a complex system to accommodate many use cases and is managed through many sub-DAOs. To provide ease of use, we use a facade of prepared-statement-type method chaining. As a result we have a flexible, scalable and easy to use DAO component.

Disadvantageously, the complex system increases development time and effort. In addition, more bugs are introduced while developer understanding is decreased.

Limitations

In the interest of time, certain features were prioritized over others and the resulting application has certain goals that may have been missed.

Continuous Delivery / Continuous Integration

While the application is deployed to a cloud environment, we could not setup CI/CD and synchronize with our cloud.

Performance Testing

Formal performance testing was not completed.

Bugs

Apart from performance, it was not feasible to fully test all user flows on our application. As a result, uncaught bugs are more than likely.

Testing

Application

As part of our testing, we covered individual features, integration, and deployment. Features were manually tested while in development. Upon integration, the overall application was

also tested manually. In addition, we set up automatic integration tests as part of our maven build with selenium and JUnit. Once deployed, we also tested manually, probing as many user flows as we could.

Security

The security testing is minimal and covers the SSL/TLS setup. It was tested manually with different situations trying to force a non-TLS connection.

Not Tested

Some items could not be tested either due to time or feasibility. The items are:

- Performance (Cloud/Local)
- Database schema and values
- Penetration Testing
  - XSS
  - Request/Certificate forgery
  - SQL injection

Collaboration

Process

As a development team we used various collaboration tools to complete the project. We used discord to communicate and hold meetings, while we stored documents and resources in a google drive. Our application version control was done through git and hosted on GitHub. Features and issues were managed through GitHub as well.

Overall, each team member was invested in the project, and contributed passionately. No conflicts occured. Any technical issues were talked out.

Contribution

Daniel

Responsible for all functionality and requirements surrounding: Login/out, Register, Shopping Cart, Security and application deployment. I learned other teammates' code best by reviewing their pull requests, helping them debug and discussions during team meetings.

Nina

Responsible for the main page functionality, displaying bestseller products by category, in store search feature, product pages, product review pages, functionality of adding reviews, use page where customers can update their information, and customers' order pages. I

learned about others' work by constantly communicating via discord, observing issues and checking out different branches on github.
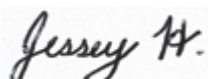
Jessanth

Responsible for analytics page functionality including generating a report for the books sold each month, implementing RT analytics on books sold and providing anonymized reports. And for REST functionality, including Product Catalog and Order Process Component/Service. I learned how to use git as a tool for development in a team project to better integrate code.

Kevin

Responsible for all data objects including the database setup, the builder pattern for the entities, the facade pattern for DAO access and several pages including purchase orders and payment. I learned my teammates' code by helping them debug and interfacing with some of their pages.

Attestations

Team Member	Signature
Daniel Kozlovsky	*(signature)*
Jessanth Harrymanoharan	*(signature)*
Kevin Nguyen	*(signature)*
Nina Yanin	NY

Conclusion

The result of this project, a MVC bookstore, is a final product that Mom&Pop can use to introduce their bookstore to the web for the first time. The project introduced some learnings and key takeaways that should be considered in the next iteration of the product. First, IBM cloud is more complex to work with compared to AWS or Firebase. Also, vanilla JS and JSP limit our options for dynamic, flexible, and aesthetic UI designs. Going forward, the application is best refactored into a single page application with some frontend framework to improve the UI. Deploying with amazon also provides more opportunities like load balancer and router management.