EECS 4413: Building E-commerce Systems

Design Document for <***Online Book Store***> Project

Contributors:

Mohammed Nasiful Haque, 214475461

Patrick Tan, 215150576

Dennis Phetsomphou, 214865745

Visit the repo on GitHub for everything latest: https://github.com/Nasiff/Online-Book-Store

TABLE OF CONTENTS

Architecture:

The online book store that was implemented follows a very simple architecture. The front-end runs on the **React** framework and it is hosted on the IBM Cloud whereas the backend is hosted on another instance of IBM Cloud and is written in its entirety in **Java** and the frontend communicates with the backend through the Jersey REST API framework.

That's all for the big picture. We're going to dig a little bit deeper into the specificities of the frontend and the backend.

Frontend (React):

The majority of the use cases and user flows are going to be taken care of, by the frontend. The frontend uses the 'fetch' library that's available as a node module on React which allows it to communicate directly with the backend through REST. Anything that requires the frontend to know the current state of a specific resource (Get all available books, Make a purchase, Registering a new user) from the database, it will do so by using this Fetch API to either make GET/POST calls to the database thus updating the current state of the user.

The frontend is neatly divided into components and services. The components are further divided based on a specific view that the user sees. These components are rendered when they need to and thus allows for high maintainability and scalability when adding new features/components. The services (Controller) are used to get data to these components when they need it, on-demand by sending in the appropriate query response to the backend.

Maintaining States is done in the upper-level component of the web application, these are meant to act as the session scope in java. Variables that need to be saved here include things like the shopping cart, login information (If logged in), or current redirection paths. Since these elements are saved here, we can have a single source of truth and pass down mutator functions to other screens/components to update them. For example, the shopping cart is updated through a handler function that is passed down to the catalogue component to update the cart. The frontend has also been deployed to the IBM Cloud here: https://mpop-bookstore.mybluemix.net/

Backend (Java & Jersey):

The backend as mentioned above uses Java and Jersey to handle all the queries made to the main database. The backend REST API has also been deployed to the IBM Cloud and queries can be made to this URL here: https://obs-4413.mybluemix.net/

The backend uses a combination of MVC + DAO design patterns to achieve this. The 'view' of the backend is the Jersey API client which is designed to catch incoming JSON requests and parse it accordingly. These JSON requests are further validated in the 'Controller' part of the backend which deals with calling the appropriate DAOs and validation (using custom schemas).

The backend is structurally divided into three sections. The first part contains all the Services that receive all the CRUD operations from the front-end and/or API. These services send these requests to the controller for further validation, parsing and re-routing. The controller then sends this JSON response to the DAO specific to the database collection that was specified in the path. The DAOs further cleans the query that came in using Prepared Statements which stops special characters from being run into the database. The DAOs work with the 'Model's of the appropriate collection to create a structured response which gets converted into a JSON Object (If it's a GET and the front end is expecting a response) and then sent to the frontend.

Tradeoff: Since the backend wasn't set as a Maven project as doing so ran into problems with sharing and whatnot. We only had a limited number of libraries to work with. Therefore we couldn't make use of some of the myriad of libraries that were available only to maven projects. This caused us to do our own manual validation over the usage of something like Marshalling on JSON schemas as we couldn't find a single .jar file that would let us achieve what we wanted. Going with a simple project resulted in having a tiny project size, therefore making it portable & usable on multiple computers thereby drastically reducing the time needed to configure and get the project running.

Sample SQL Files & Schema:

The default backend schema that was provided was modified slightly to meet all the requirements of the project. More data was added to cover even more test cases. The sample SQL file to populate a local database that follows the data model schema can be found ***here***. The schema is also accompanied by another SQL file that clears the database, in the event there comes the need of restarting over.

REST API Endpoints:

Address:
GET **/rest/address** (Retrieves address by uid)
POST **/rest/address** (Creates address given input)

Books:
GET **/rest/books** (Retrieve all books)
GET **/rest/books/{bid}** (Retrieve book by bid)
GET **/rest/books/categories** (Retrieve all categories)
GET **/rest/books/categories/{categoryName}** (Retrieve all books of categoryName)
GET **/rest/books/{bid}/review** (Retrieve book review by bid)
POST **/rest/books/{bids}/review** (CREATE review with bid)

Users:
GET /**rest/user** (Login user with email and password / Get uid)
POST **/rest/user** (Creates new user)
GET **/rest/user/admin/month** (get top selling books this month)
GET **/rest/user/admin/alltime** (get top selling books of all time)
GET **/rest/user/partner/{bid}** (all orders for a specific book)
GET **/rest/user/partner/{bid}/json** (get a book with a specific id in a json file)
GET **/rest/user/partner/books?search=<title>** (get a book by title)

Purchase:
POST **/rest/purchase** (create/make a purchase order)

For more information regarding these REST API endpoints and the headers & bodies needed to make those calls to the backend, Go to this link ***here***

Use Cases:

Please find the class diagrams ***here***. The files there need to be downloaded and then imported into draw.io. Click on File > Import From > Device… on draw.io to open it. Click on the Use Cases tab at the very bottom.

Class Diagrams:

Please find the class diagrams ***here***. The files there need to be downloaded and then imported into draw.io. Click on File > Import From > Device…on draw.io to open it. Click on the Class Diagram tab at the very bottom.

Sequence Diagrams:

Please find the sequence diagrams ***here***. The files there need to be downloaded and then imported into draw.io. Click on File > Import From > Device… on draw.io to open it. Click on the Sequence Diagram tab at the very bottom.

Testing:

Due to time constraints, most of the testing has been done manually. The backend has been stress tested both individually and in a group session to cover as many scenarios as is feasibly possible. Testing was done both individually and in a group session.

Testing scenarios include but not limited to: Empty Input, Is Alphabetic, Is Numerical, Valid User IDs, Validating Emails, Validating Passwords, Validating User Type. Tested for SQL injection attacks and also tested User, Admin & Partner authentication.

User Stories:

This project covers all the functionalities related to the user stories expanded below. The stories are divided into epics. Each epic relates to one actor in the system.

Epic 1

As a Visitor, I want to be able to access the book catalogue and add things to my cart **so that** I can browse the catalogue and potentially purchase the items.

User Story	Acceptance Criteria
1) As a Visitor, I need to browse the products in the catalogue **so that** I can find products I'm interested in	Ensure that the Visitor is able to: ● Log in to the bookstore ● Navigate to the catalogue ● Browse books by category
2) As a Visitor, I need to select bookstore products in the catalogue **so that** I can add it to my cart	Ensure that the Visitor is able to: ● Navigate to the catalogue ● Select books I want to view

	• View reviews posted for the book • Add the selected books to a cart
3) As a Visitor, I need to view my cart **so that** I can adjust and proceed with checkout.	Ensure that the Visitor is able to: • Navigate to the shopping cart • View items and the relevant expenses • Adjust the number of items in the cart
4) As a Visitor, I need to place my order **so that** I can purchase them.	Ensure that the Visitor is able to: • Navigate to the checkout screens • Sign in / Register or provide shipping information • Insert a valid Credit Card Number • Review Order Details • Submit Order and Receive Confirmation of Success or Failures
5) As a Visitor, I need to be able to sign in **so that** I access specific services.	Ensure that the Visitor is able to: • Navigate to the login screen • Input login information • Receive confirmation whether the login was a success or failure
6) As a Visitor, I need to be able to register **so that** I can have an account to access specific services.	Ensure that the Visitor is able to: • Navigate to the registration screen • Insert required information needed for registration • Select the type of user to be associated with (CUSTOMER, ADMIN, or PARTNER) • Submit information and receive a confirmation on whether it was successful

Epic 2

As a Customer, I want to be able to access the book catalogue through a secure login **so that** I can purchase products and leave reviews.

User Story	Acceptance Criteria
7) As a Customer, I need to be able to browse the catalogue and add items to the cart **so that** I can prepare for checkout.	Ensure that the Customer is able to: ● Perform User Stores 1-3
8) As a Customer, I need to be able to place my order **so that** I can complete the checkout process.	**Prerequisite**: Signed in as a Customer Ensure that the Visitor is able to: ● Navigate to the checkout screens ● Have pre-existing shipping data gathered from registration ● Insert a valid Credit Card Number ● Review Order Details ● Submit Order and Receive Confirmation of Success or Failures
9) As a Customer, I need to be able to place reviews **so that** I can give feedback on the items in the bookstore	**Prerequisite**: Signed in as a Customer Ensure that the Customer is able to: ● Navigate to the book they are interested in reviewing ● Post a Review for the specific book ● See their review reflected in the catalogue

Epic 3

As an Admin, I want to be able to view analytics for the site **so that** I can know which books are doing well in terms of sales.

User Story	Acceptance Criteria
7) As an Admin, I want to be able to view analytics for the site **so that** I can know which books are doing well in terms of sales.	**Prerequisite**: Signed in as an Admin Ensure that the Admin is able to: ● Navigate to the admin analytics ● screen ● View the top monthly and all-time selling books from the bookstore

Visit the repo on GitHub for everything latest: https://github.com/Nasiff/Online-Book-Store

Epic 4
As a Partner, I want to be able to get specific data on which of my books are being purchased. I would also want to be able to download the JSON files associated with them so I can feed it to another system. I would also want to be able to search the kind of books this system offers through REST

User Story	Acceptance Criteria
7) As a Partner, I want to be able to see orders for my books in JSON files and also search for specific or a list of books.	**Prerequisite**: Partner has a partner key in the headers of their HTTP requests Ensure that the Partner is able to: ● Get a list of purchase orders that include their books in a JSON file ● Search the list of books by providing a keyword ● Get more information about a specific book.

Team Contributions:

The team collaborated on Github and the workflow involved creating an issue to track the current thing being worked on. The individual assigned to the issue will have to branch out and create a PR which the other two members of the team would have to review and approve before getting it merged into the main branch. This allowed us to always know what the other teams are doing. We met once a week to prioritize & plan what needs to be done and stayed in constant communication through social media.

Dennis:

Designed the front end site and its flow through Adobe XD. Created the website and implemented the designs through the use of the react library. Following the designs, I incorporated following front-end functionalities in regards to the front end services, including…

- Credit Card Checks
- Rerouting and Redirection
- Session states and mutations (ex. shopping cart, login, and etc)
- CSS Styles and Transitions

Aside from those services, I worked with the team to design and incorporate the backend services into the site. This involved designing responses and requests that were needed to fulfill the project requirements, for example, building up a JSON object to be posted to the shipping service, or fetching the response from a login to determine a successful login. Lastly, I was in charge of deploying the front-end to the cloud.

Nasif:

Created the design and the API pathways that would allow the front-end and the backend to interact with each other. Re-iterated on the API design to make it more robust. Hosted and led the weekly meetings to prioritize tasks for the week ahead and also assign the necessary tasks to finish that week. Maintained the repo and laid down ground rules regarding as to how our work flow is going to be like. Had a hand in fixing anything Github related. Fixed some majors bugs relating to CORS and any server related issues. Deployed and was in charge of deploying the backend on the cloud, Wrote up all of the design documents which includes but not limited to the UML, Class & Sequence Diagrams. Took part in the stress testing to ensure there weren't any major bugs.

Patrick:

Designed a database schema (refined from the given template) to suit our bookstore application's needs and created a sample SQL script to start our database with existing data we could work with (inserted books with all their information such as isbn/bid, title, prices, author while also adding existing users, reviews, purchase orders, etc.).

Implemented and handled all the REST services, the model controllers and the Data Access Object (DAOs for user, address, books, purchase orders, reviews, etc.) for our application's business logic. This also included input validation, error checking and using prepared statements to prevent SQL injection attacks. Handled valid/invalid inputs accordingly and provided the appropriate formatted JSON response (successful or not). Tested backend services with sample inputs and documented the expected outputs (with the corresponding input) into text files. Communicated with the team to ensure backend and frontend logic was consistent while also looking out for and addressing any bugs/concerns.

Visit the repo on GitHub for everything latest: https://github.com/Nasiff/Online-Book-Store