Pink Fluffy Unicorns

Alborz Gharabaghi
Orion Nelson
Sidharth Sudarsan
Benedict Miguel

Table of Contents

I – Architecture and Design

- System Architecture

The architecture of our system is divided into three parts. The non-relational database structure based on NoSQL and implemented on MongoDB contains the following collections: User, Product, Order and Brand. The Pink Fluffy Unicorns web API is implemented with NodeJS and uses JSON Web Tokens. The Pink Fluffy Unicorns contains API endpoints for user login, user register, product add, product all and many more. Pink Fluffy Unicorns' API returns JSON objects for valid requests which are validated by authentication tokens generated by JSON web tokens. The front-end system is implemented using ReactJS and uses Axios for HTTP requests on the server. They serve as UI for making requests to the server, displaying responses from the server, filtering and sorting products, and is the main way of client to server communication. For more information on the structure of the subsystems refer to Section 3 of the document.

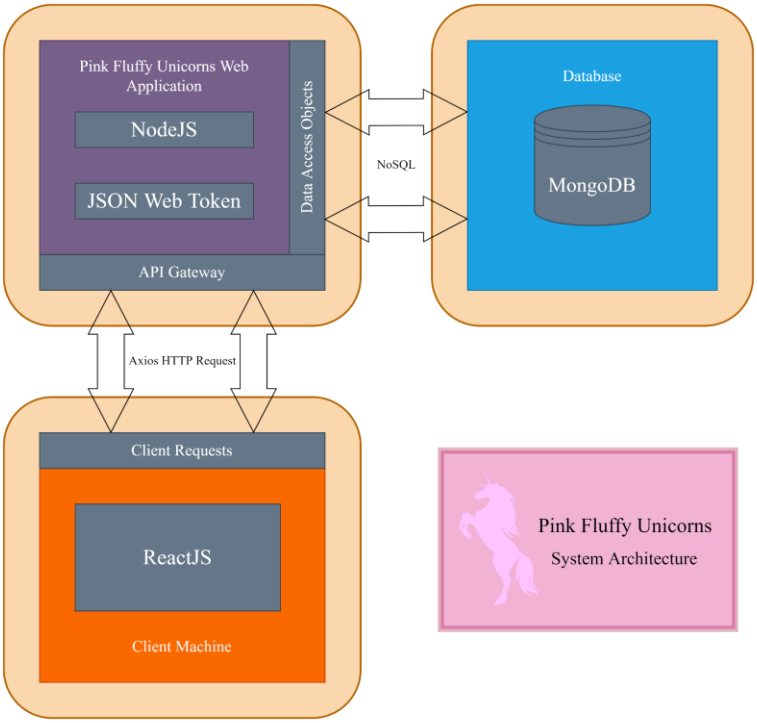

Figure 1: System Architecture

- Repository Structure (Organization: https://github.com/pink-fluffy)

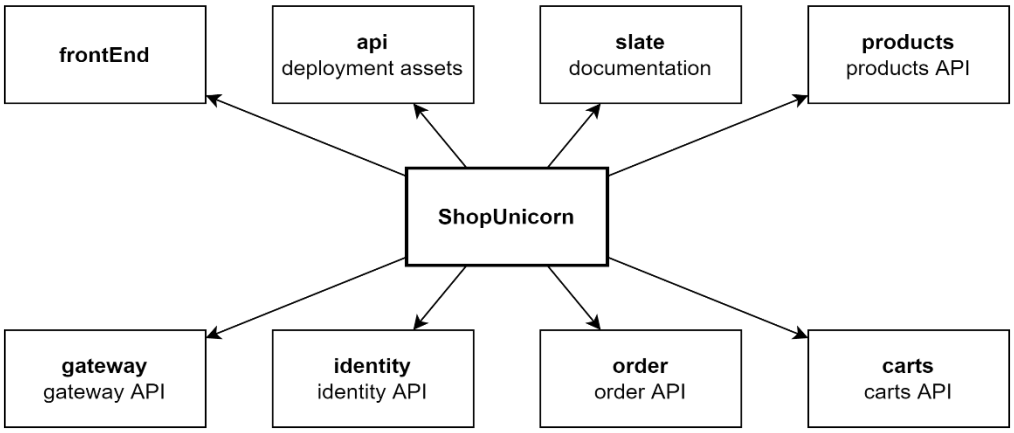

Figure 2: Repository Structure

- Design Patterns
  - Microservices Pattern
    Microservices is a design pattern categorized as a composition of different services, each of which can be accessed with API calls individually, as opposed to one API call that handles the request. In the design of our Web Services, we used microservices to decentralize and split services. Each API developed has a different functionality than other APIs and must be directly called for use. Each API call from the client must go through the gateway API for processing before being transferred to the API call. Some of the API calls developed are the user login and user register APIs. For a reference of the APIs developed refer to Section 3 of the document.

  - Model, View, Controller Pattern
    The MVC pattern was widely used throughout the application. There are several controllers for different API calls and different Models for the Users and Products. The view was created using ReactJS.

  - Data Access Object Pattern
    For all the different collections in the database, a different Data Access Object was created to create, modify and access these collections which are only accessible by the controllers.

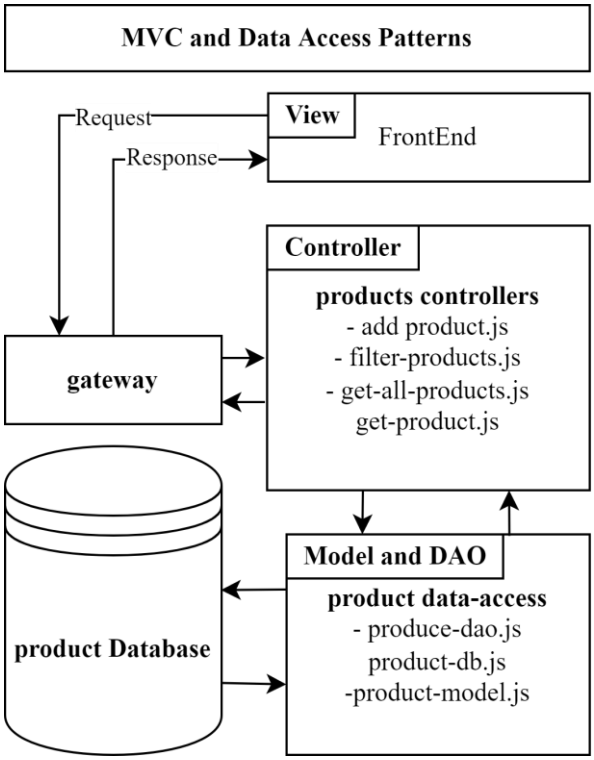

Figure 3: Design Patterns Visualized

  - Clean Architecture (Source: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html)
    The clean architecture pattern was use throughout each of the microservices. Dependency Injection was used for each of the microservices to remove any dependence related issues that the web application may have. Each concrete component, the microservices, may be needed for other services, but the concrete component does not.

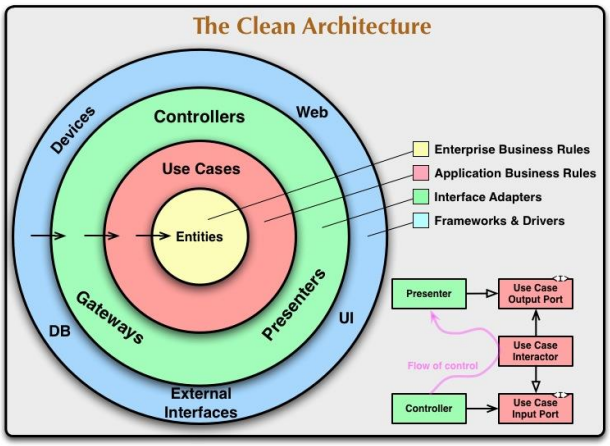

Figure 4:Clean Architecture Design Pattern

This architecture pattern allows the code to be easily maintainable and modifiable without risking dependency issues as the code base increases.

II – Implementation

- Database of choice - MongoDB (Website: https://www.mongodb.com/)
  In choosing the database with respect to the E-commerce nature of the project, we needed a database that is not complex, and is modifiable throughout the project. Because of this, using a non relational database would be beneficial for the project. The non relational database type allows us to rapidly store, add and modify items without worrying about relationships between the different collections. Choosing a non-relational database also saves us time as non-relational databases allow the storage of 'unstructured' items and allow us to directly see each item on the database as a JavaScript object already as opposed to creating a parser. In this project, the choice for a non-relational database came down to MongoDB for the sheer amount of experience the backend developers have. Database is running the cloud name MongoDB Atlas.

- Backend of Choice - NodeJS (Website: https://nodejs.org/en/)
  NodeJS is a JavaScript platform used to rapidly build and scale network applications. Because of this definition, using this existing framework allows us to rapidly create the backend of our application and use its existing features as opposed to creating a backend from JavaScript which would considerably take more time. NodeJS also contains proper documentation and is widely used for E-commerce applications which can help solve problems that we may encounter throughout development.

- Authentication of Choice - JSON Web Token JWT (Website: https://jwt.io/)
  JSON Web Token is an industry standard for authentication. By using JWT and its existing documentation, it allows for better understanding of the authentication service as opposed to creating our own which may lead to more problems as development continues.

- Frontend Choice - ReactJS (Website: https://reactjs.org/)
  ReactJS is a frontend framework that is used to quickly build the frontend components of a website. Using its existing features will save development time as reacts has a considerable amount of followers and contains proper documentation. ReactJS also works well with NodeJS, and the relationship between reacts and NodeJS is widely used throughout the web, giving us potential resources that we can consult throughout development.

- HTTP Request - Axios (Website: https://axios-http.com/docs/intro)
  Axios, being an HTTP request client for Node.js was a clear choice for any http requests. Axios was built for Node.js and seeing that we are using Node.js for backend development, it was the only practical choice we have.

- Server Framework - Express (Website: https://expressjs.com/)
  Express is a backend framework designed for Node.js. It allows rapid server-side development and with our choice of using Node.js, and seeing that Express is open

source, it allows for future errors to be solved relatively easier and allows rapid server-side development.

- Deployment – Docker
  The backend microservices and frontend client is deployed with Docker on a virtual private server running Ubuntu 20/04 and NGINX web server. The domains are SSL certified and support HTTPS only. The server also runs DDOS protection with fail2bain, Each of the microservice runs on individual containers and use docker compose to manage microservices and the gate API. Continuous deployment setup, where any changes on the GitHub get deployed.

- Limitations implied by the choice of implementation
  Limitations of the Web application is largely implied by the choice of implementation on the subsystems. For example, the choice of a non-relational database comes with the disadvantage of incompatibility with SQL instructions, poor management tools, and potential errors in the database due to its decentralized non-relational property (Sources: https://pandorafms.com/blog/nosql-vs-sql-key-differences/) . Node.js, ReactJS, Axios, Express and JSON Web Tokens are largely limited by the fact that they are frameworks, as to which can limit further customization and may become unstable due to framework changes, an increase in load or refactoring of code. With these said, the limitations are mitigated by its open sources structure, which users can find and detect potential shortcoming of these frameworks.

III.- Diagrams

API stack (Full API Reference: https://api.shopunicorn.live/docs/#introduction)

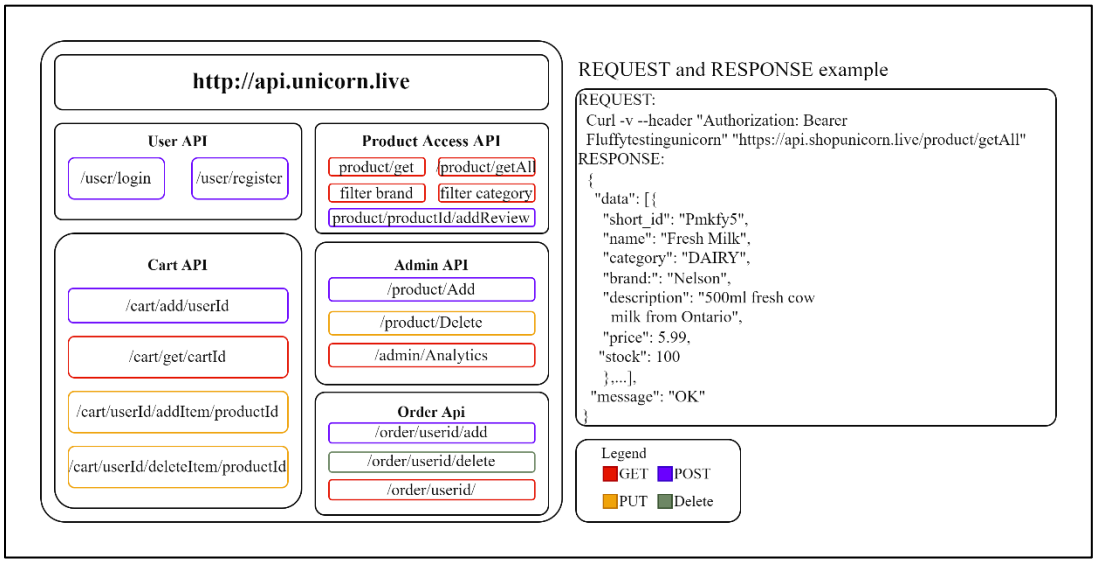

Figure 5: Pink Fluffy Unicorn's API visualized

Database Schema

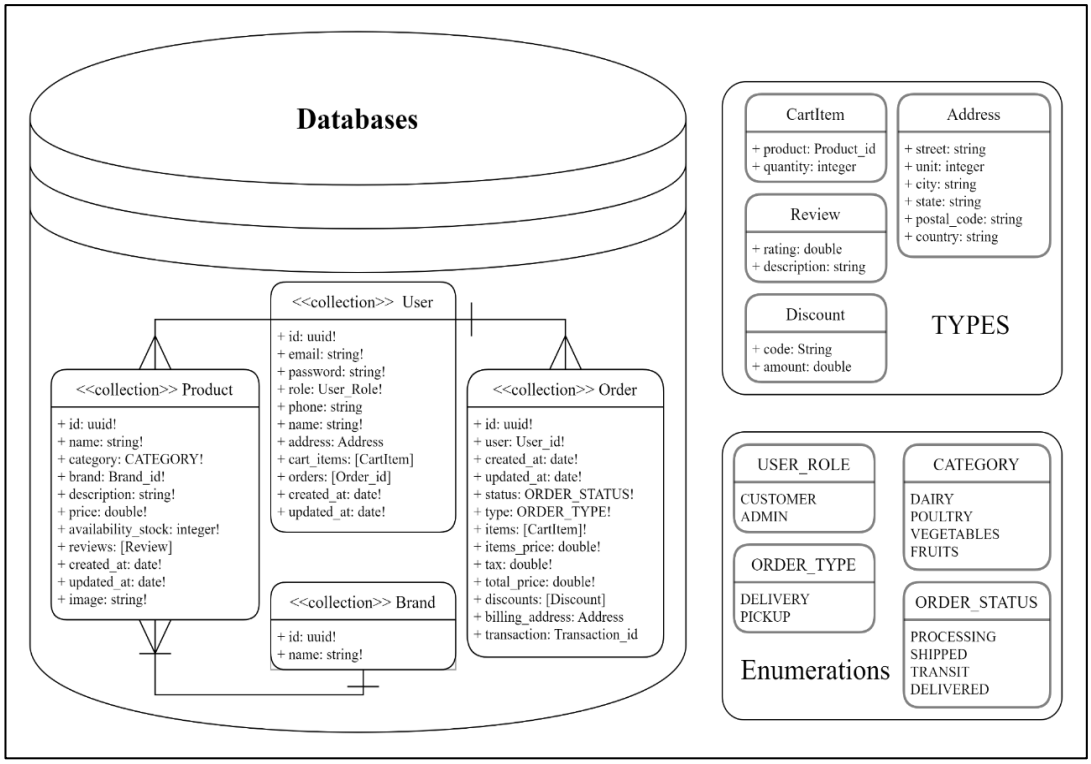

Figure 6: Pink Fluffy Unicorn Database Schema

Sequence Diagram – Retrieving objects from the database

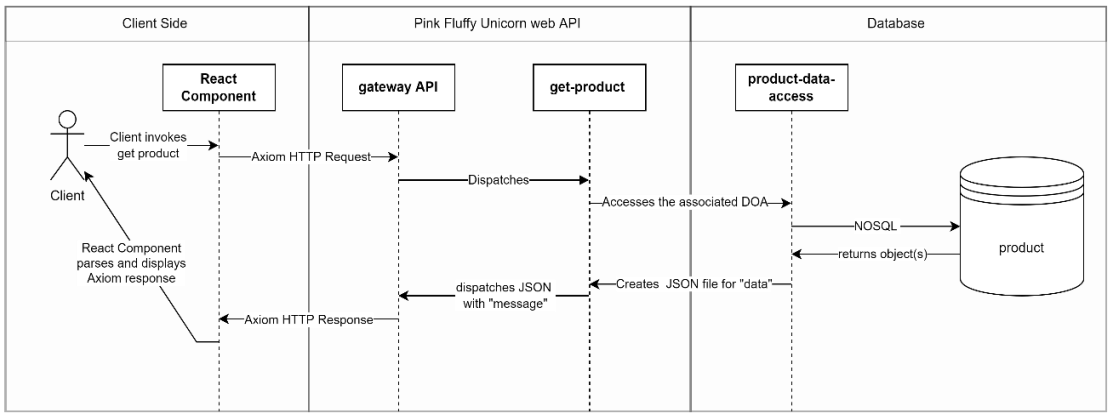

Figure 7: Sequence Diagram for retrieving objects

Sequence Diagram – Adding objects from the database

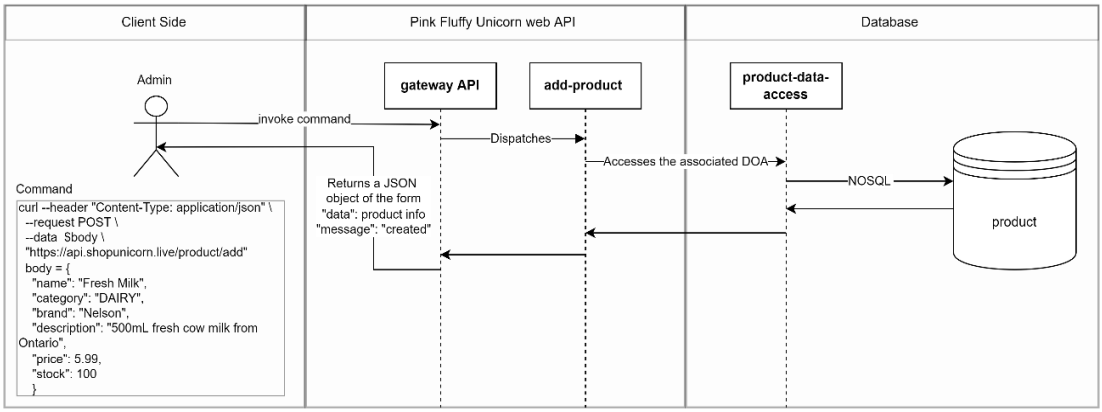

Figure 8: Sequence Diagram for adding to the product Database

Use Cases Diagram

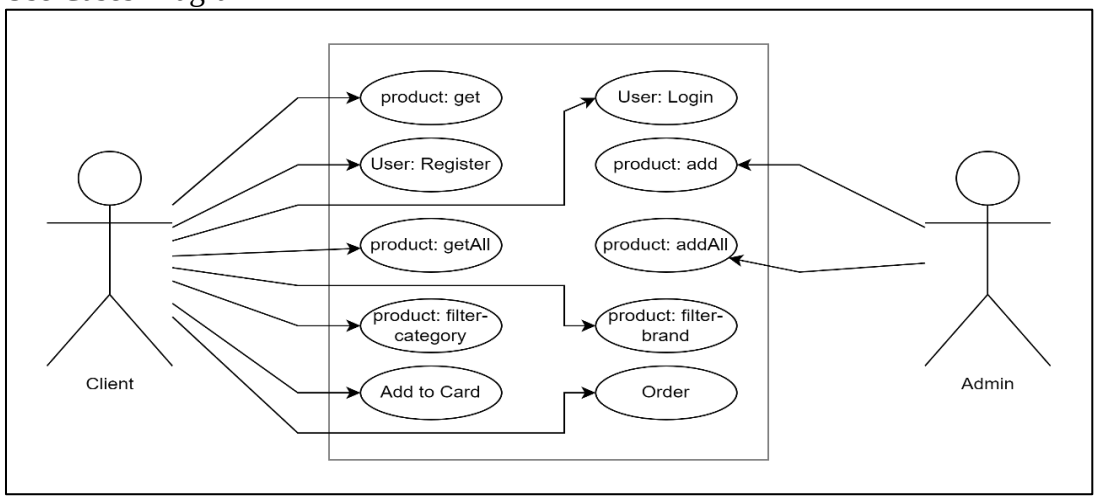

Figure 9: Client and Admin Use Cases

Class Diagram

The nature of our application includes not being object oriented, and thus, for UML or class diagrams, there is no object-oriented relations, but there are still relations in between the folders, subfolders and files. The following is an example of the relations in one of the repositories.

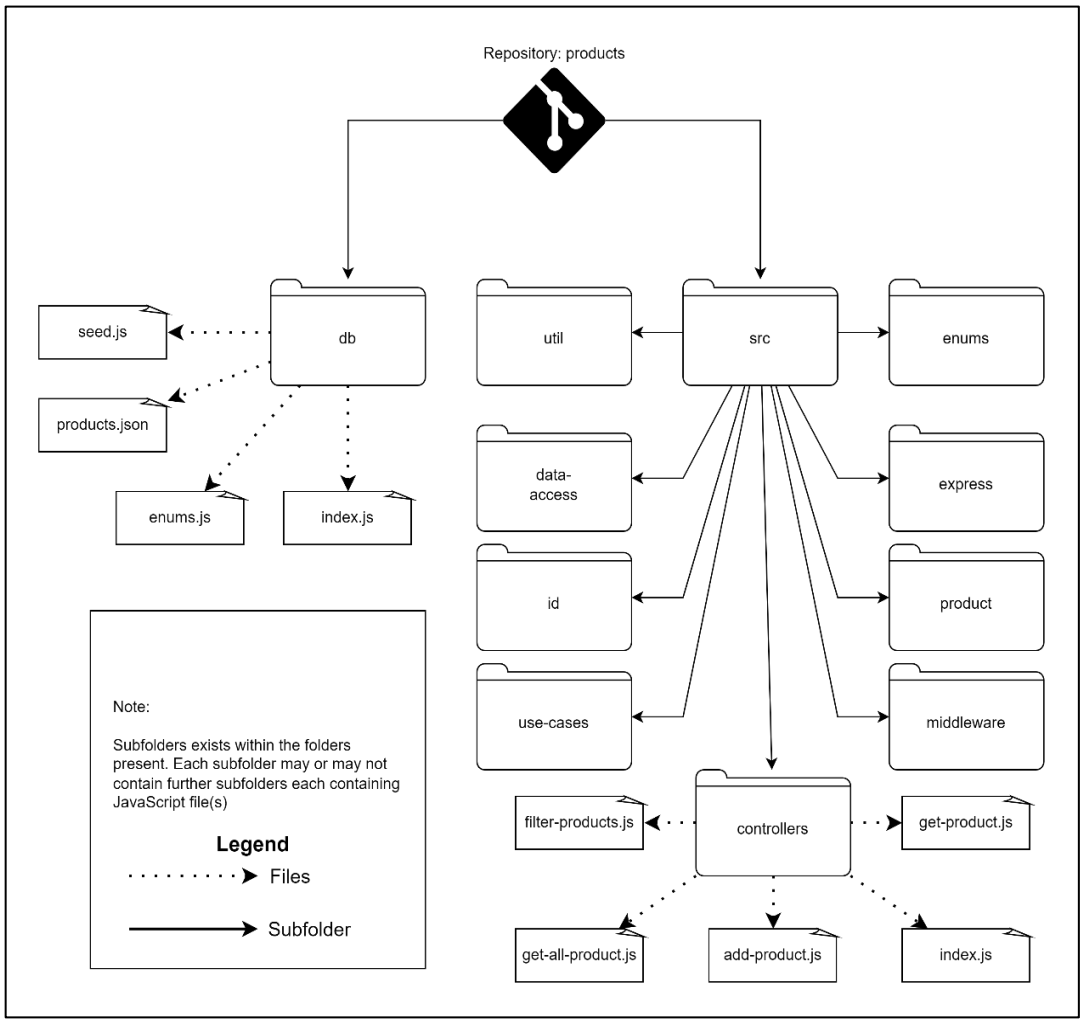

Figure 10: The following high-level diagram shows the hierarchy for one of the repositories, products

IV - Advance Features

There were two advanced features that were intended to be implemented. These two advanced features are widely used throughout the industry. The features are intuitive can be implemented without risk of injuring the subsystems. Both features also complement each other as they can be used in unison. For example, subscription models can be independent but can still be discounted, and/or discounted products can become discounted if they are bought using a subscription model.

Discounted Products

Products during a selected time can be discounted by a certain percentage. This discounting can be triggered by three events. An admin delegates a product as a

discounted item. Or after the client purchases the product after several times, the product can become discounted for each subsequent purchases between 1-25% depending on the original price of the product. The last trigger for a discounted product would be if they were purchased with a subscription model, where the product will be discounted depending on the length of the subscription, and the price of the scription with respect to the product.

Subscription Model
Subscription model allows clients to auto-purchase items in between time intervals of their choosing. Time intervals can be monthly, weekly, yearly or can be tailored to suit the client's needs. The subscription model can include any products. These products can be discounted in its entirety by the subscription model and/or can additionally be discounted if the individual item(s) are currently discounted at the time of purchase. Subscription models are priced on the length of the subscription, the length can be purchased on a monthly or yearly bases.

V – Performance Report

VI.- Security Testing and Vulnerabilities

- Security Testing Report
  Testing was done using the default authentication key 'fluffytestingunicorn'. The APIs were tested with respect to whether the API could or couldn't be accessed depending on the authentication key provided. The response from the request has parameter "message:", which determines whether the request was executed or returned error with respect to improper authentication. The following diagram returns the error code(s) for a request without a valid authentication

Error Code	Meaning
400	Bad Request -- Your request is invalid.
401	Unauthorized -- Your API key is wrong.
403	Forbidden -- The resource requested is hidden for administrators only.

Figure 11: Potential value of "message" parametre of an invalid request due to improper authentication

- Security Vulnerabilities
  Due to the rapid development of the web application, only certain security testing was done. Vulnerabilities can from rom accessing potential APIs with invalid credentials, potentially allow invalid credentials to modify the databases, SQL injection, cross site scription may also be a vulnerability.

  Other vulnerabilities come from the implementation using frameworks. Node.js with its nature of using more resources as the load increases, may cause the web application to crash due to DDOS attacks or if too many requests are made. The implementation of ReactJS may also cause the web application to develop several vulnerabilities due to its

rapid development, causing potential information leaks, information overload, information injection and several more problems pertaining to rapid development without robust testing. Further analysis also found the using MongoDB and its non-relational database may become a security concern as the web application grows, mainly the number of products, users, and order increase may put the robustness of the database in question.

VII – Team Member Contributions

Name	Work Description	Total Work Contribution
Alborz Gharabaghi	- Developed the front-end system with Orion Nelson - Developed communication between front-end and back-end	25%
Orion Nelson	- Developed the front-end system with Alborz Gharabaghi - Developed communication between front-end and back-end	25%
Sidharth Sudarsan	- Developed the back-end server - Developed the back-end API calls - Created the databases and relations between databases - Helped with the communication between the front-end and back-end.	27%
Benedict Miguel	- Created the Design Document - Created the diagrams associated with the web application - Scraped potential products from the web - Populated necessary databases	23%
Total		**100%**