Group109

USCoffee
Software Architecture Document

Version <1.0>

Revision History

Date	Version	Description	Author
19/07/2023	1.0	First version of Software architecture document	Group 109

Table of Contents

Software Architecture Document

1. Introduction

- **Purpose:** The primary objective of the Software Architecture Document (SAD) is to furnish a comprehensive manual that delineates the architectural blueprint and arrangement of the software system under development. It aims to impart a clear comprehension of the essential components, their interactions, and the fundamental design principles governing the system. Serving as a crucial reference for all stakeholders involved in the project, including developers, designers, project managers, and clients, the document ensures a unified vision and understanding of the software's architecture.

- **Scope:** The scope of the Software Architecture Document (SAD) encompasses a detailed overview of the software system's architecture, which includes its various modules, components, and their interrelationships. This document outlines high-level design decisions, architectural patterns, and the technologies employed in the system's development. It covers both the functional and non-functional aspects of the software architecture, providing a comprehensive view of the system's structure and design considerations.

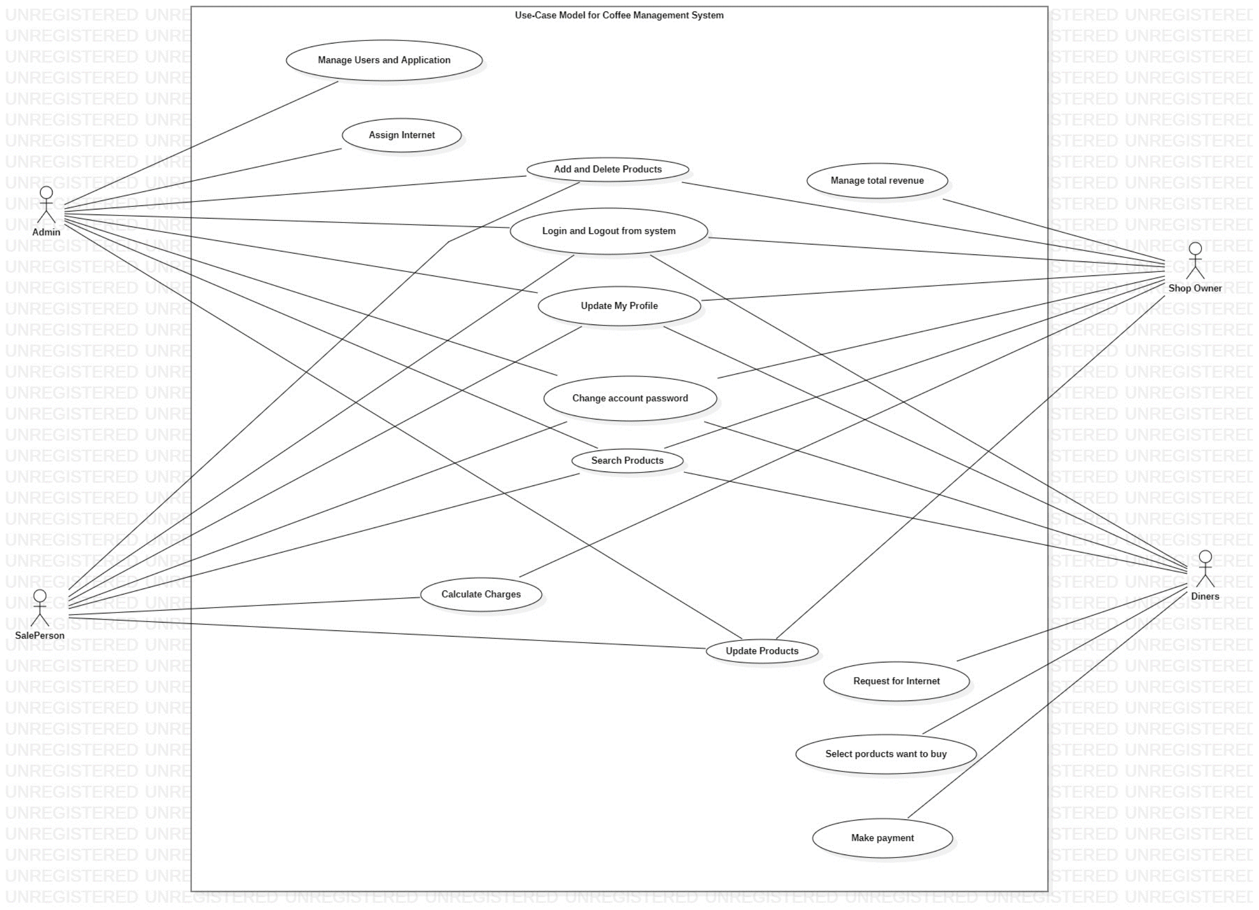
- **Definitions:** We will define important words and concepts related to the software system to make sure everyone involved understands them the same way.
- **Acronyms:** We will list the full meanings of any short forms used in the document to avoid confusion.
- **Abbreviations:** We will provide easy-to-find explanations for common short forms that are used in the software system.

2. Architectural Goals and Constraints

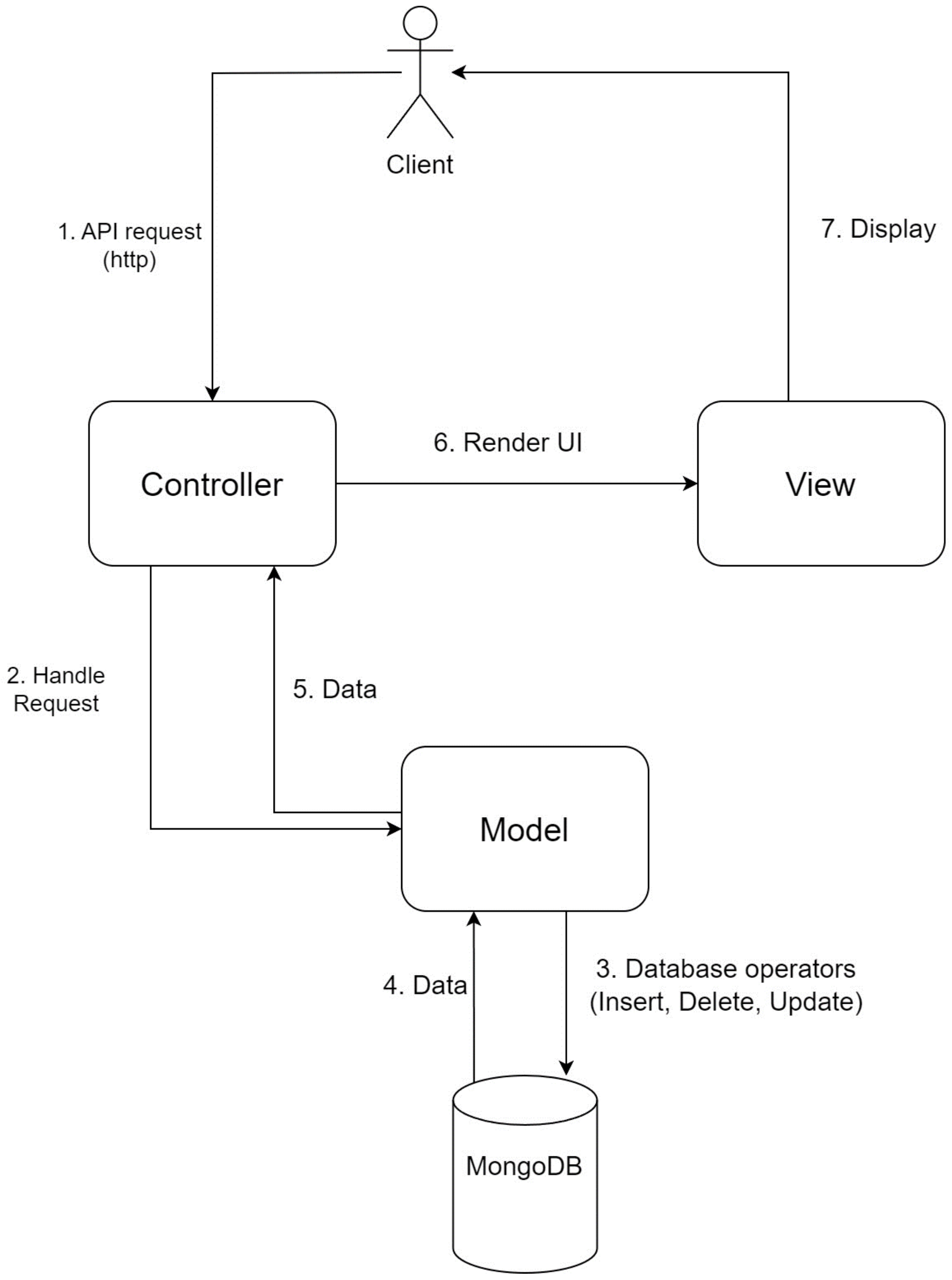
Some constraints and requitement requested for architecture:

- User's personal information should remain confidential and inaccessible to other users.
- All features request internet access to be used.
- About UI:
    - Developed by React
    - Simple, clean, friendly
- MongoDB is use for database

3. Use-Case Model



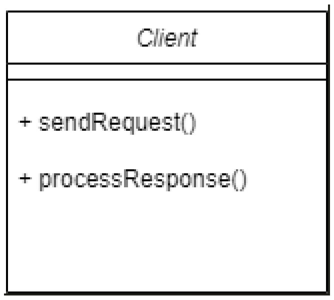
4. Logical View



4.1 Component: Client

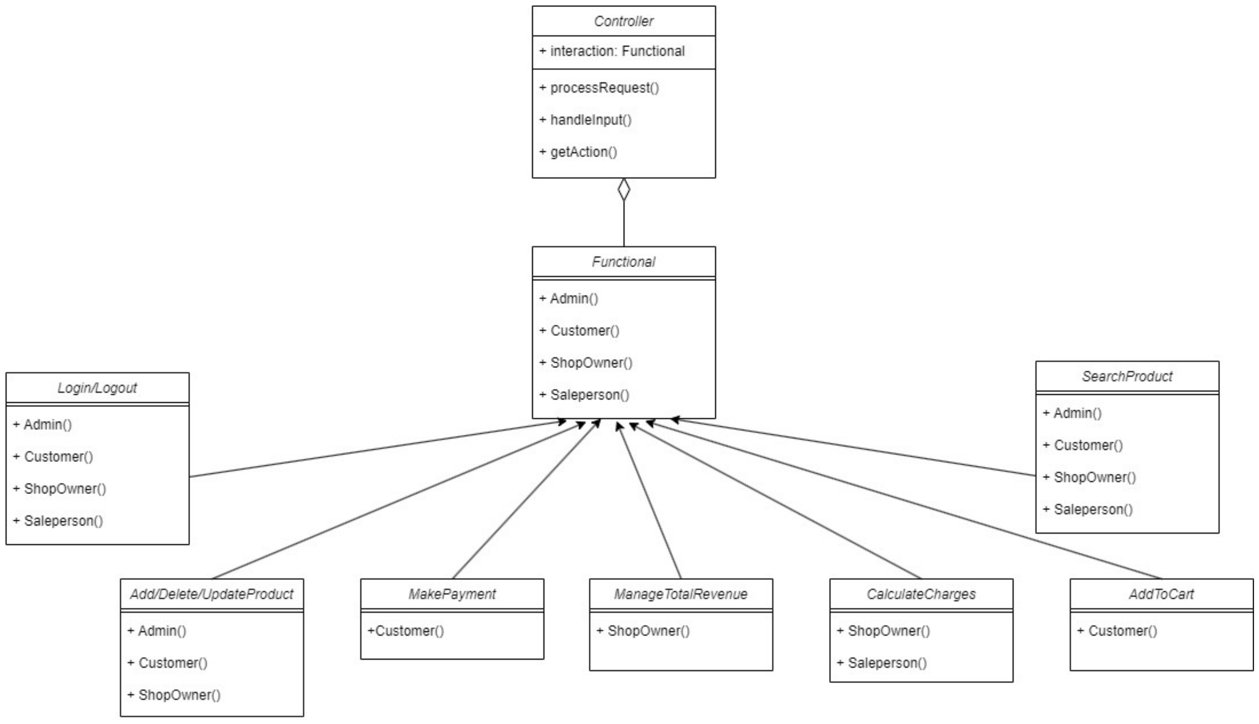
- **Client:** This represents the users' browsers or any client-side devices that interact with the website. The Client sends HTTP requests to the server to request data or perform actions. This class has 2 methods to process response and send HTTP request to controller

Functional



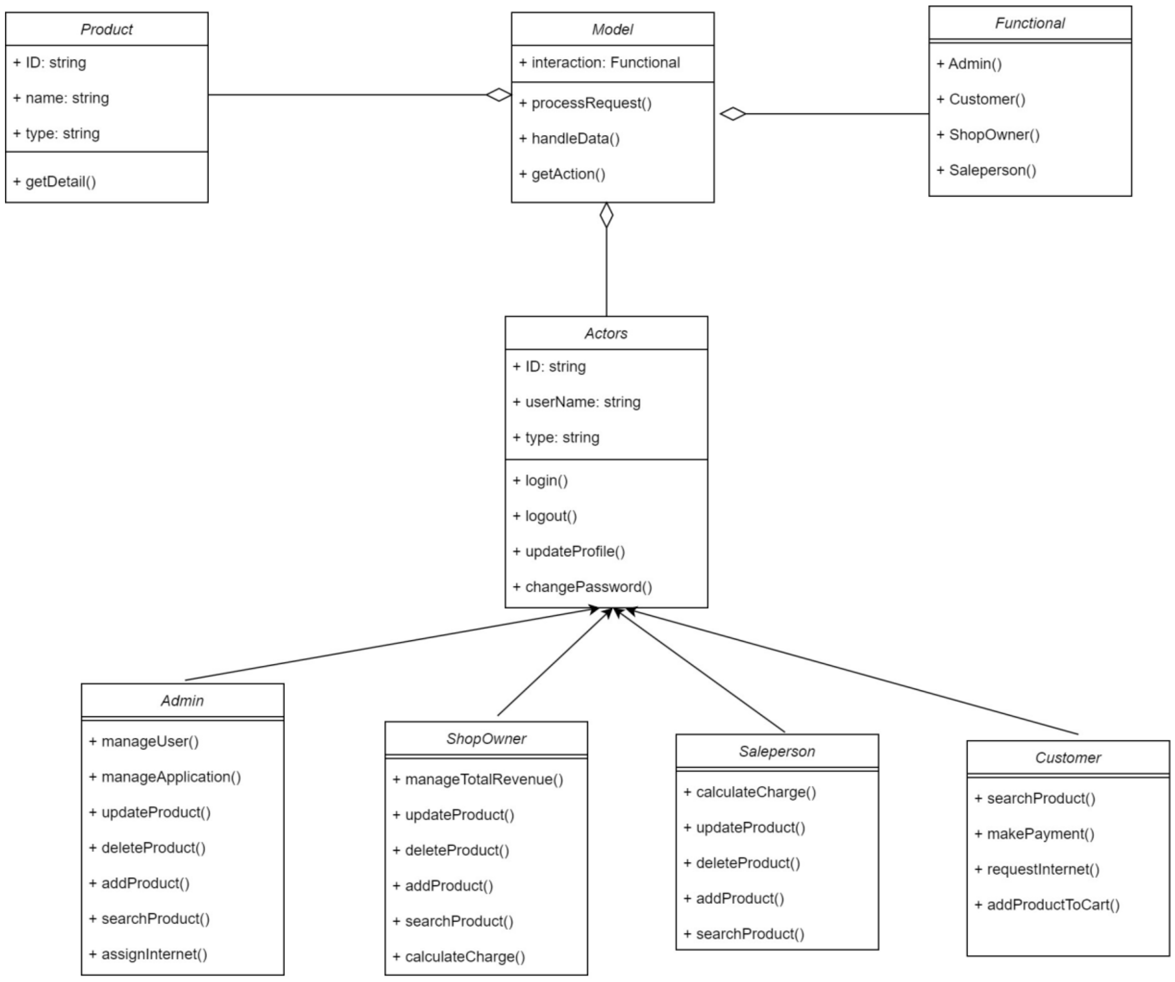
4.2 Component: Controller

- **Controller:** The Controller receives and handles the incoming HTTP requests from the Client. It acts as an intermediary between the Client and the Model. It processes the requests, fetches data from the Model, performs any necessary business logic, and sends the response back to the Client. The framework use for controller component is Next.js. In this diagram, the "Controller" class contains an attribute of type "Functional" and various methods for handling input, requests, and receiving actions from client
- **Functional:** An abstract class which has 4 functions representing 4 actors of the website. The classes below represent different use-cases, depending on each use-case inheriting from the "Functional" class with corresponding methods.



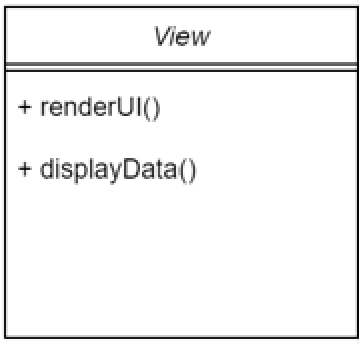
4.3 Component: Model

- **Model:** represents the data and the business logic of the application. It is responsible for interacting with the database (MongoDB) and performing various database operations like CRUD (Create, Read, Update, Delete) operations. The Model contains the application's data and state. Because the controller use Next.js, the Model will use MongoDB with Mongoose (an ORM library) for a MongoDB database. Similar to the "Controller" class, the "Model" class includes an attribute of type "Functional" and methods for processing requests, handling data, and receiving actions from "Controller"
- **Product:** This class includes attributes representing product details and a method called getDetail() to retrieve detailed information about the products.
- **Actors:** This class contains common attributes and general methods for users. The classes below represent different types of users, inheriting from the "Actors" class, and each user type has additional methods representing the services that each user type can use.



4.4 Component: View

- **View:** the presentation layer responsible for rendering the user interface (UI) components. In this case, React is used as the front-end library to build the View. It consumes data from the Model and displays it to the users. Because Front-end uses React, the View component would primarily be developed using JSX for rendering UI components and CSS for styling. This class has 2 methods, one takes information provided by the Controller and uses it to render UI, and the other methods are responsible for displaying the rendered UI to the user.



5. Deployment

6. Implementation View