Group 08

MOVIE STREAMING WEBSITE
Software Architecture Document

Version <2.0>

Revision History

Date	Version	Description	Author
29 thg 6, 2022	<1.0>	Write Introduction	Khôi
		Add Use-case model	Nguyên
		Write Architectural Goals and Constraints	Khôi, Thịnh
2 thg 7, 2022	<1.1>	Write Logical View Introduction	Thịnh
		Write Component Detail	Khôi
		Add Component Image	Nguyên, Lam
		Check document font format and presentation style	Khanh
10 thg 7, 2022	<2.0>	Edit Class Diagram (add relationship, divide UML into separate area,...)	Khôi
		Split class-diagram into smaller for easy view.	Nguyên
		Draw Deployment Diagram and Implementation View.	Thịnh

Table of Contents

Software Architecture Document

1. Introduction

This document will describe our application's component and its logical perspective based on the use-case model described in the Use-case specifications document and also show how the application deployment will perform in a real-world scenario.

2. Architectural Goals and Constraints

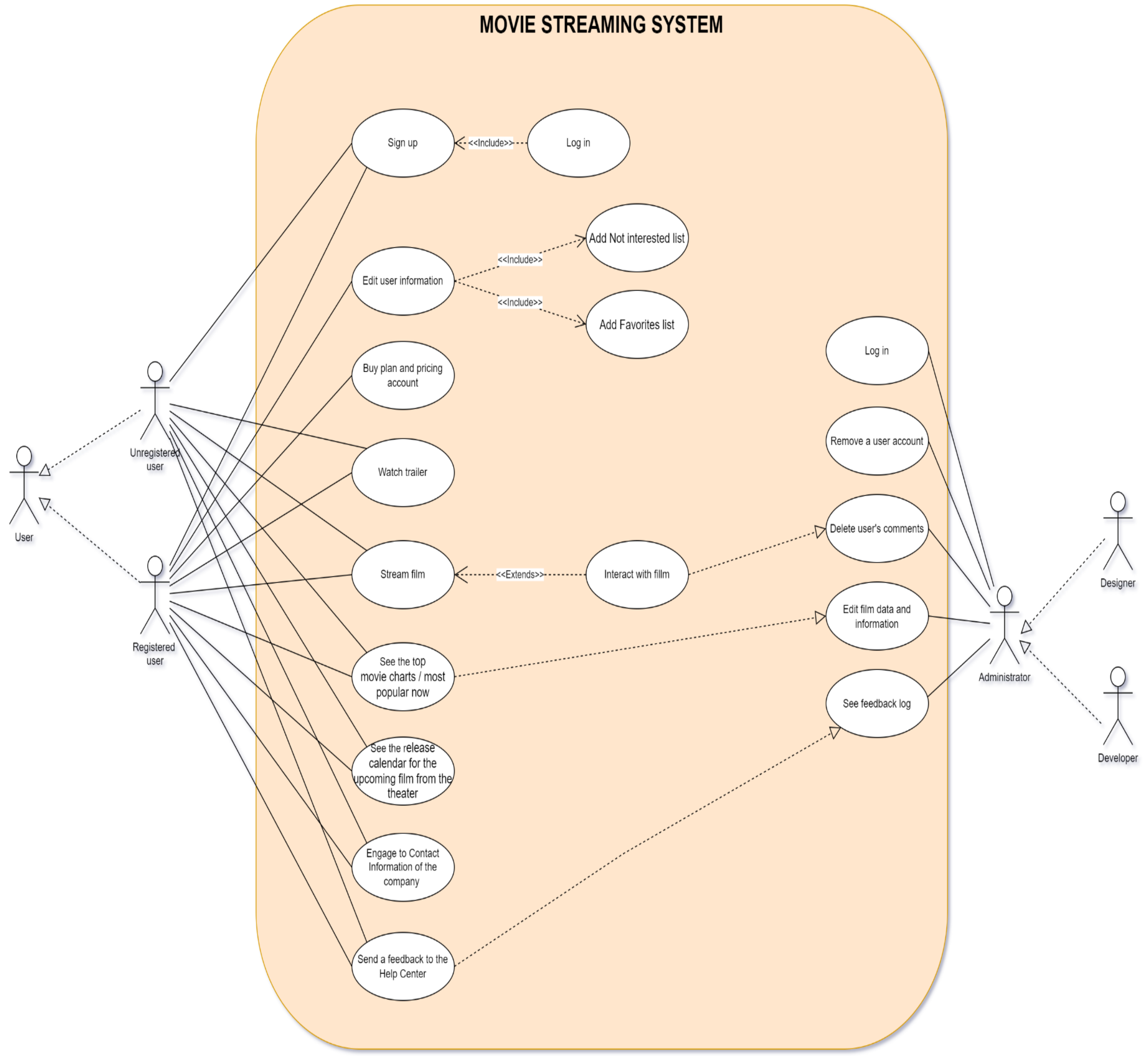
Our website is an off-the-shelf project such as our price which is affordable for everyone to use in order to have a better movie-watching experience. Alternatively, the website allows customers to view movies at any time and from any location. We have an expert team to test the product, which has been verified by the world's best testers using global standards. Additionally, users can customize the film categories based on their preferences.

Portability: We intend to create for the desktop web environment and do not plan to develop for the mobile environment. Users only need a web client such as Google Chrome, Microsoft Edge, or Firefox to use this website.

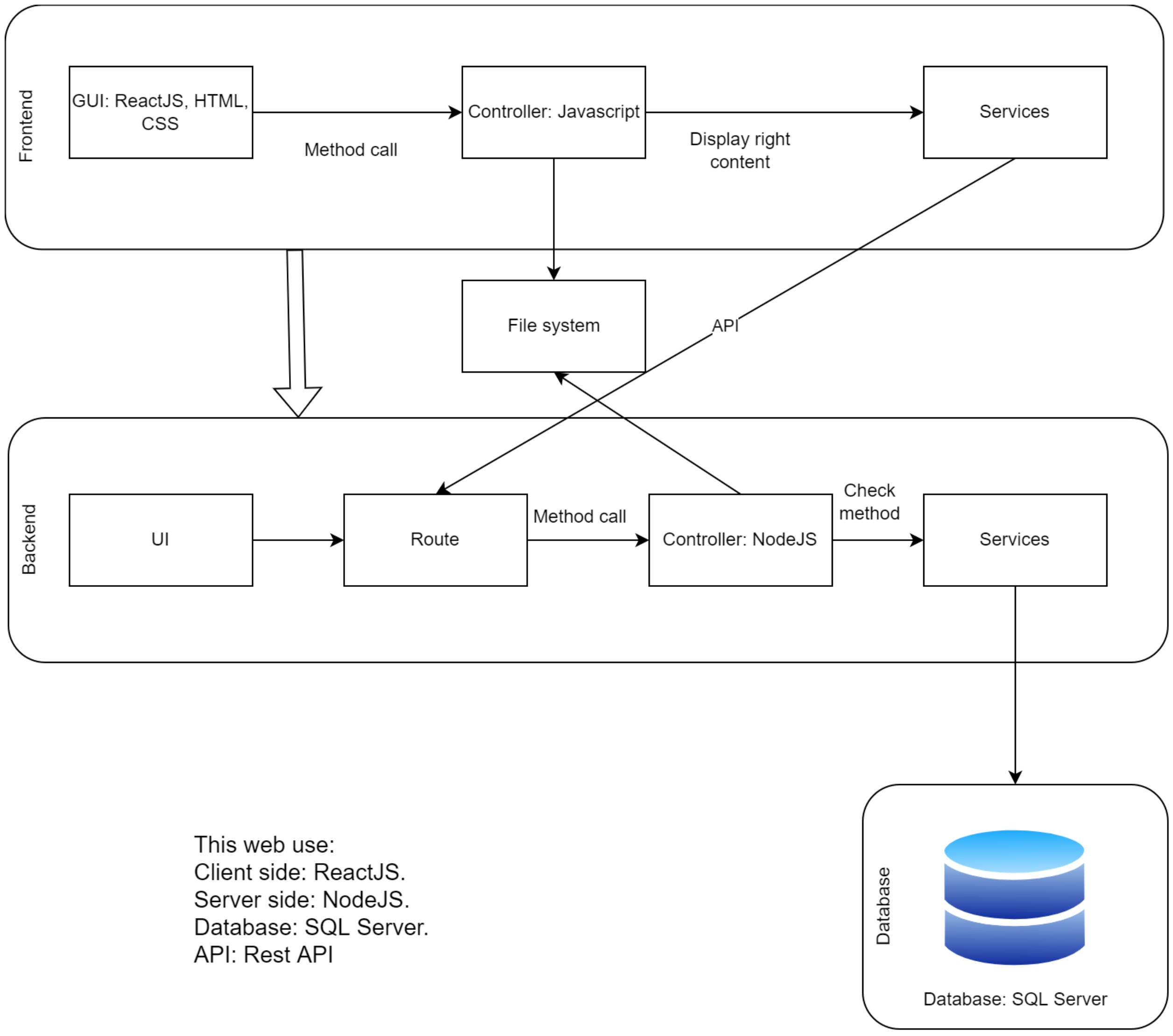
User's privacy and information security: Protect all user personal information and only analyze user actions such as liked/disliked movies for our own movie recommendation usage and product improvement.

Development strategy: First, we design the user interface and its components. Before constructing any other features, we focus on finalizing the streaming movies function and its related functions - database containing the movies for streaming itself, administration database control.

3. Use-Case Model



4. Logical View



Our group uses Three layered architecture for this project.

On the **Frontend side**, we separate into 3 layers:

- The first layer is GUI. We use some languages and frameworks to develop like: ReactJS, CSS, HTML, SASS. This layer shows the User Interface for our application and the user can see the navigation bar, the name of the movie, the poster of the movie,...
- Then next to the second layer is the Controller. This layer controls and contains functions to call from the GUI layer. The layer uses Javascript to declare functions and develop methods to call.
- After that, the Services layer uses API to communicate with the Backend side.

About the **Backend side**,

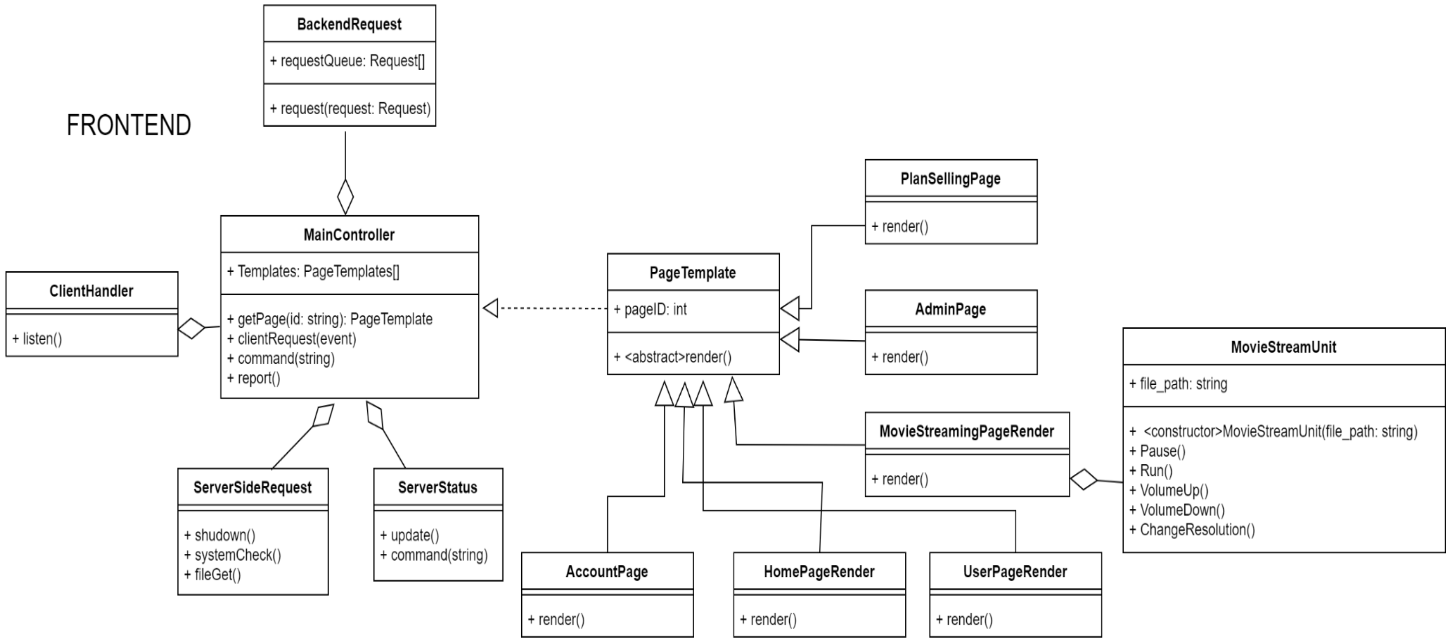
- We create the Backend with the Command Line Interface for admin to operate.
- In the Route layer, we develop general APIs to handle, coordinate and distribute calls to other functions. In this layer, we use Javascript and ExpressJS framework to create APIs.
- Next one is the Controller layer. It controls and calls methods from client to server and vice versa using the

NodeJS environment.

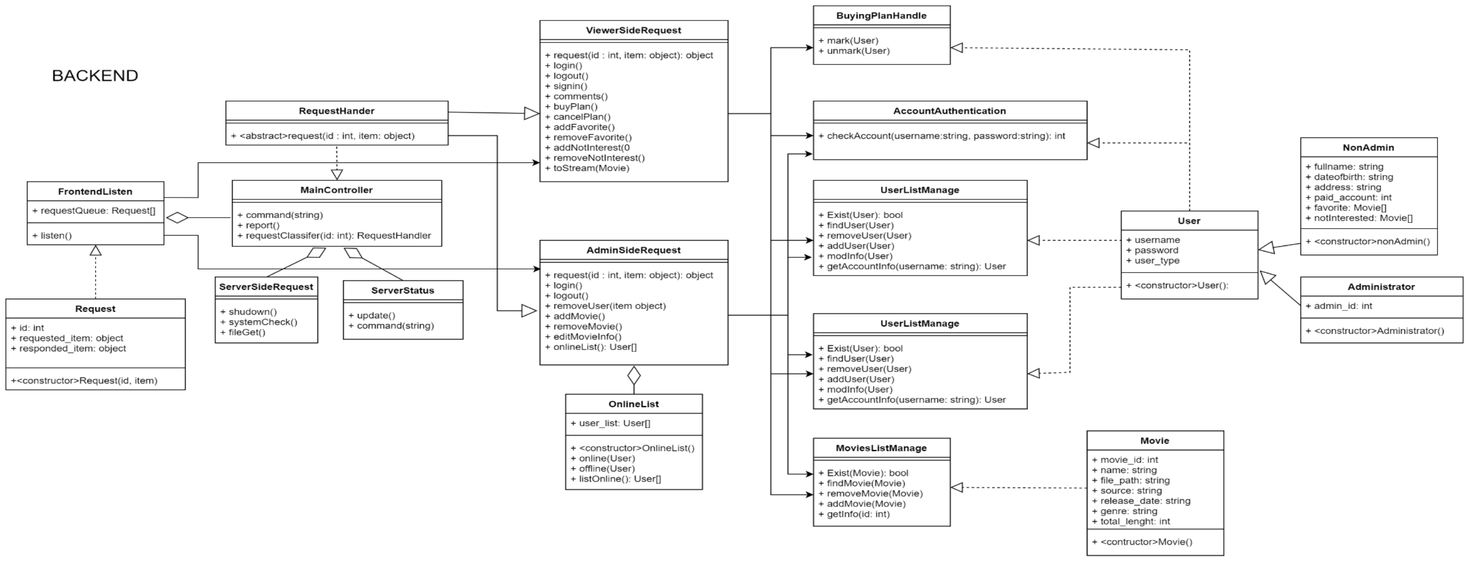
- Then the Services layer queries to the Database and responds to all the requests from the Backend. We use SQL Server from Microsoft to query easily.

4.1 Class Diagram Overview

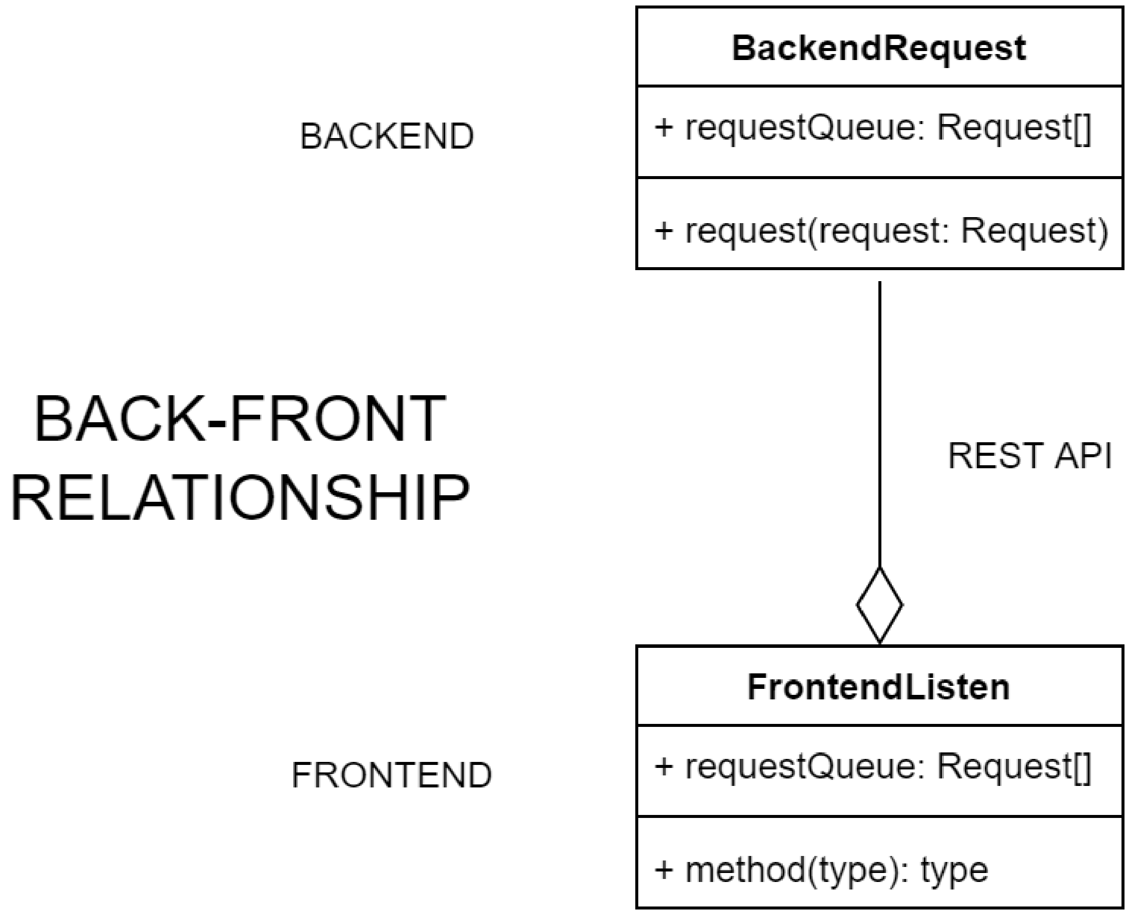
4.1.1 Frontend



4.1.2 Backend

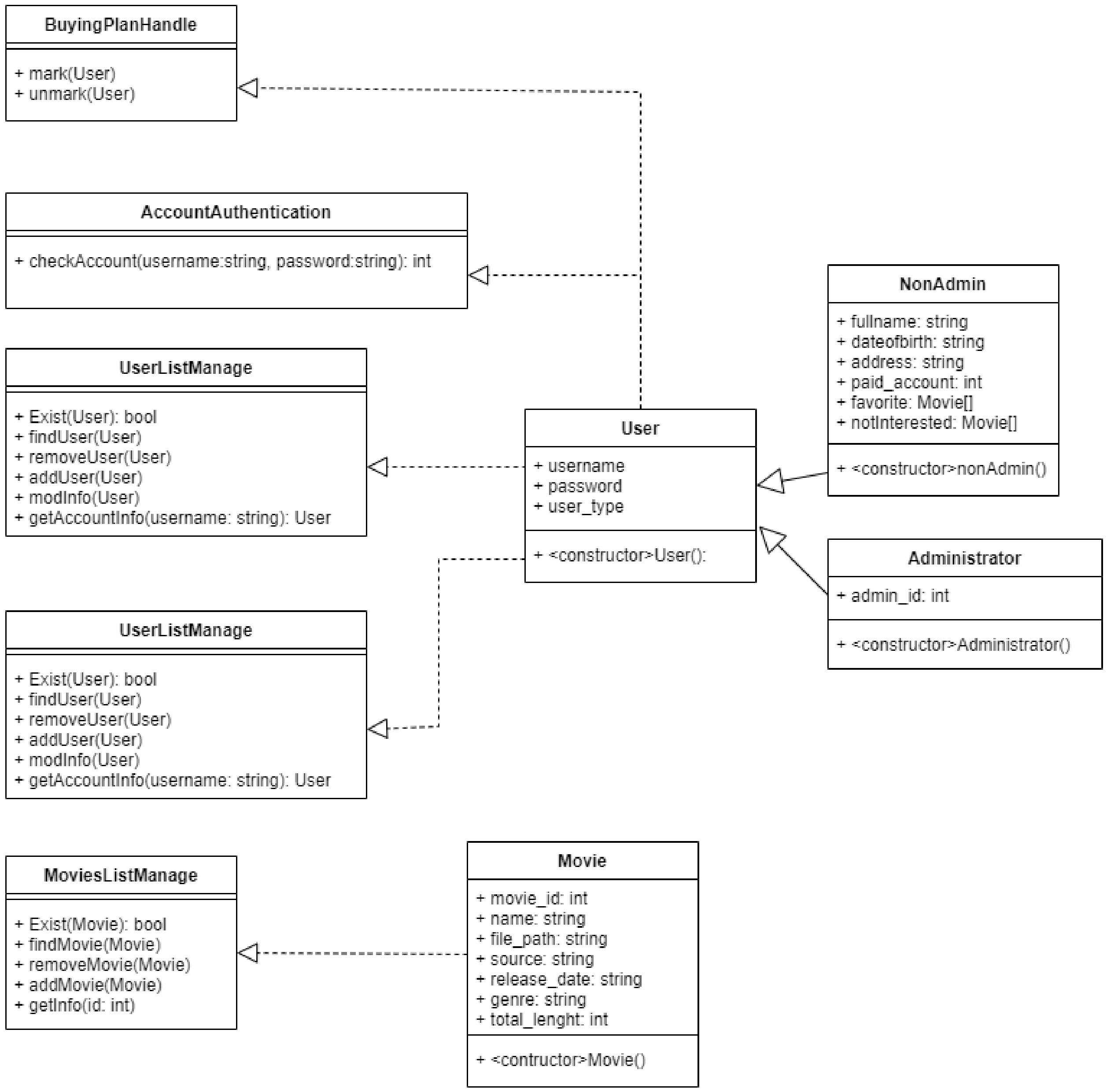


4.1.3 Frontend - Backend Relationship (How the Backend and Frontend side will communicate)



4.2 Component: Backend - Service

4.2.1 Class Diagram



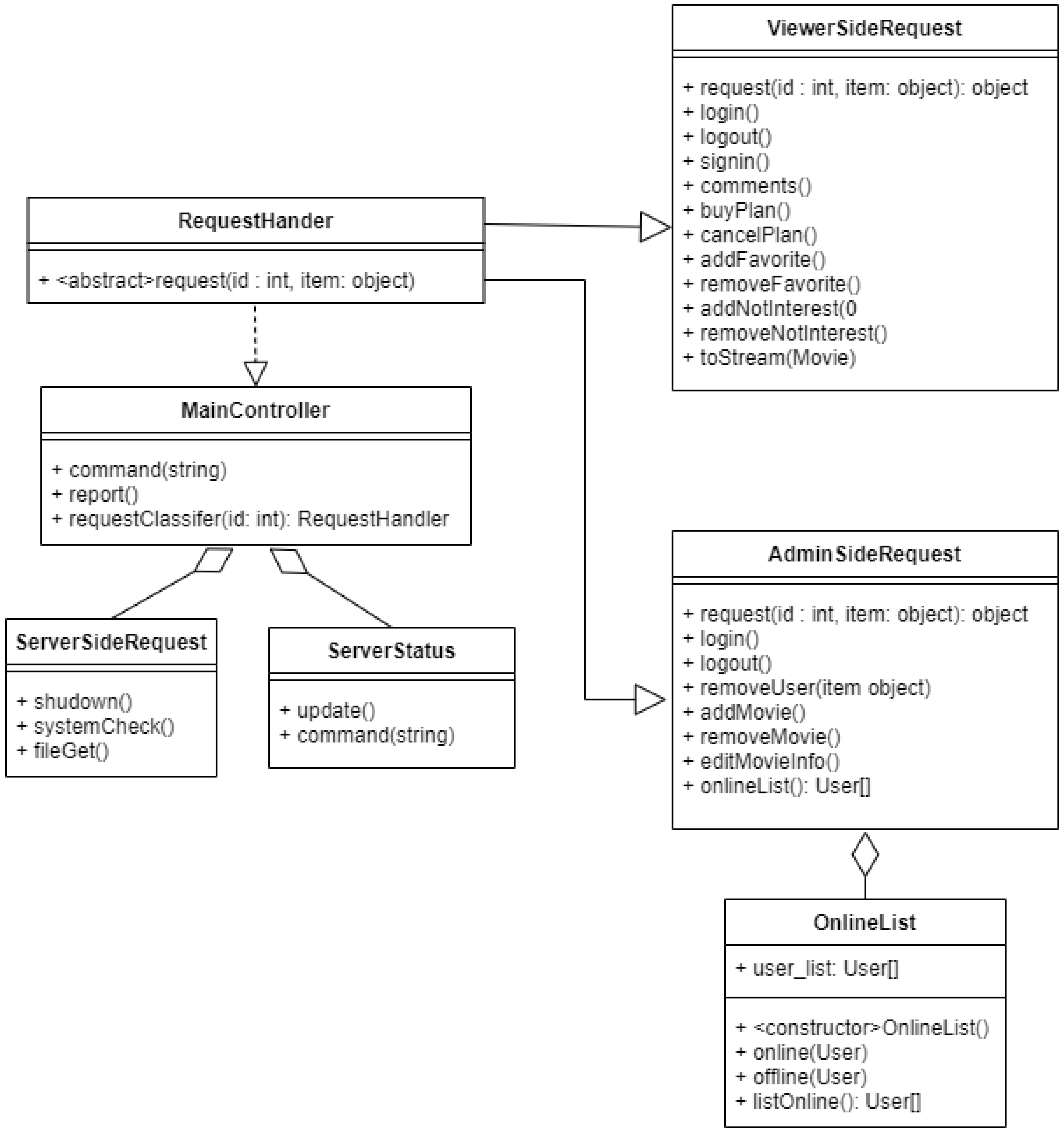
4.2.2 Detail

Backend services, which primarily access databases for all UI tasks, will be accessed by Backend controllers.

- **Account Authentication:** Use for checking accounts when there is a login request.
- **OnlineList:** Manage the current online user.
- **UserListController:** Use for adding or removing a user from the database.
- **MovieListController:** Use for adding or removing a movie from the database.
- **BuyingPlanHandle:** Use for marking and handing a plan subscription request.
- **Users <= NonAdmin/Administrator:** a data structure used for storing/ transfer data between components.

4.3 Component: Backend - Controller

4.3.1 Class Diagram



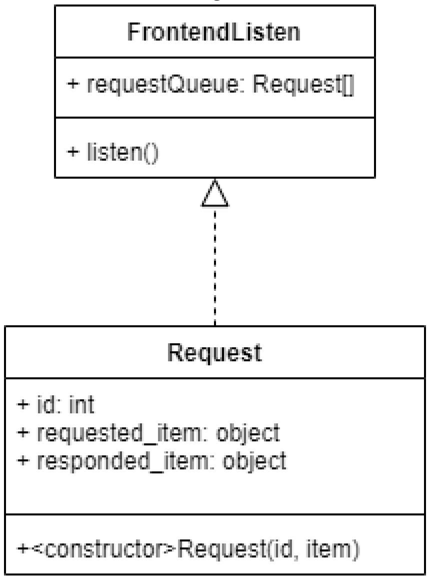
4.3.2 Detail

Backend controller will handle the request from the Frontend and manage the system.

- **MainController:** Receive the incoming request and specify what type of request before using any Backend service.
- **ServerSideRequest:** Handle an incoming command from the server operator.
- **ServerStatus:** An interface between the operator and the server itself, different from the admin page that the Frontend has.
- **RequestHandler <= ViewerSideRequest/AdminSideRequest:** Each handle a specific request based on its component name, the route component will ask directly with these components when receiving what type of request from the main controller.

4.4 Component: Backend - Route

4.4.1 Class Diagram



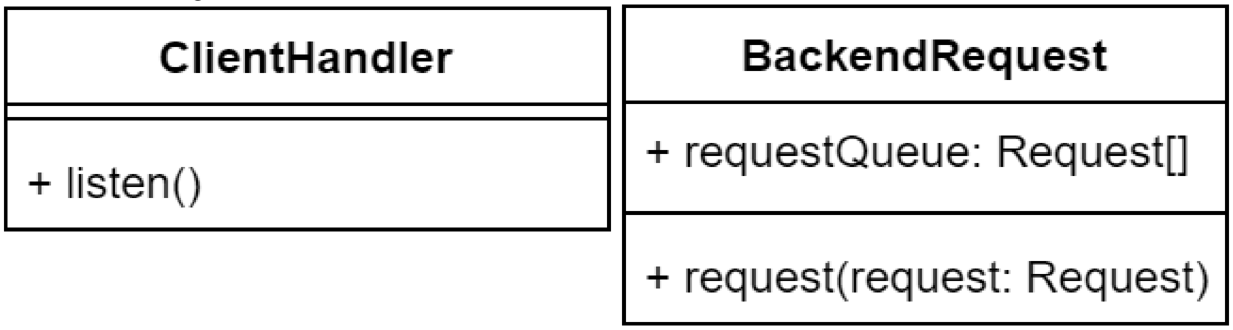
4.4.2 Detail

This component will handle incoming Frontend API requests and parse them into a functioning command for the backend system to operate. When a request is complete, the route component will provide a response to the Frontend.

- **FrontendListen:** Receive incoming request from Frontend, and communicate with controller.
- **Request:** A container used for specified which command/request that recieved.

4.5 Component: Frontend - Services

4.5.1 Class Diagram



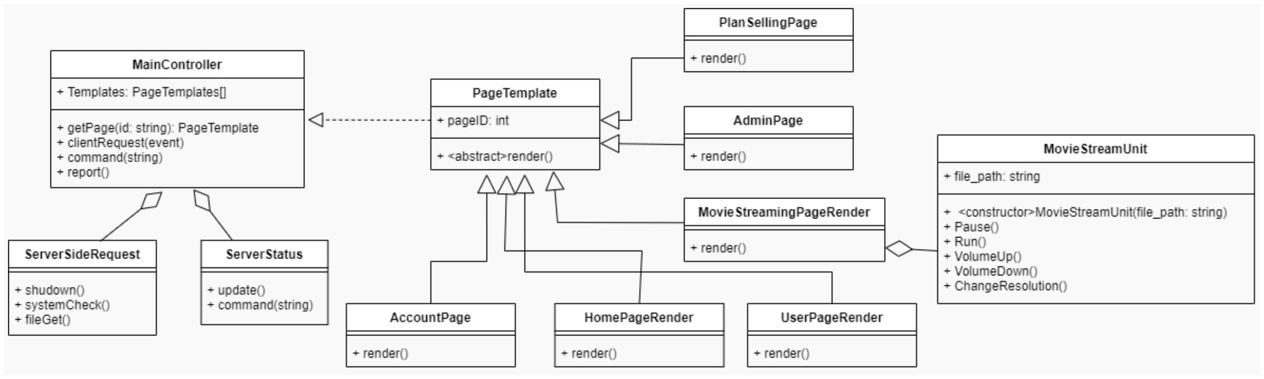
4.5.2 Detail

Any incoming user request will be handled by the service component, which will interface directly with the Backend.

- **ClientHandler:** Receive incoming request and parse it to the renderUI or Route for request.
- **BackendRequest:** Will send the request API to the backend side and wait for response.

4.6 Component: Frontend - Controller and UI

4.6.1 Class Diagram

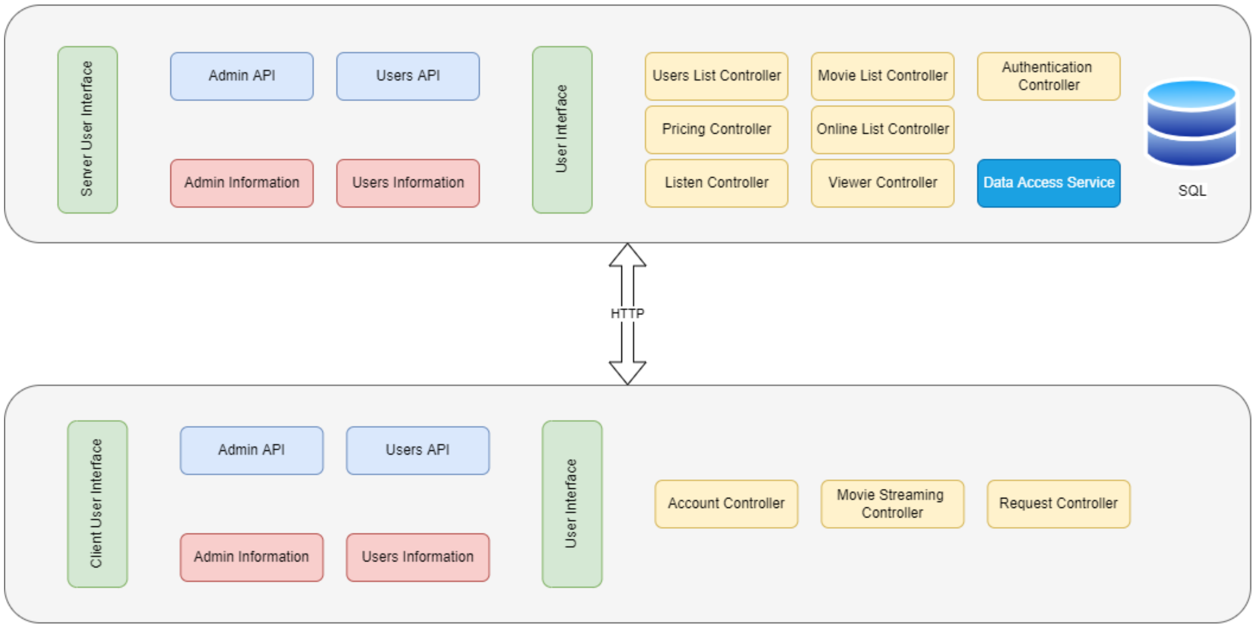


4.6.2 Detail

The controller will render the GUI for the user based on the user request and data received from the Backend.

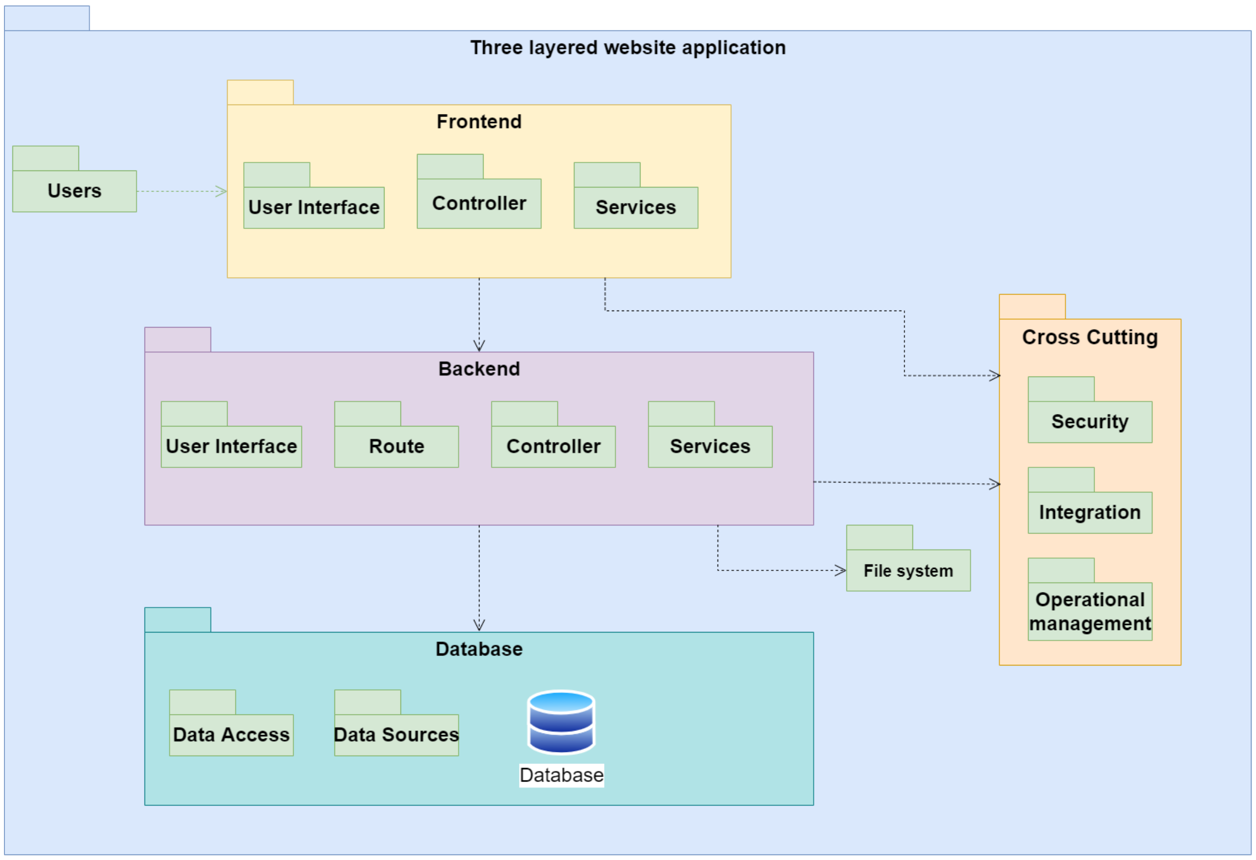
- **MainController:** manage the system and render pages depending on the client request, handle incoming requests from both the operator and the client.
- **ServerSideRequest:** Handle an incoming command from the server operator.
- **ServerStatus:** An interface between the operator and the server itself, different from the admin page that the Frontend has.
- **PageTemplate <= PlanSellingPage/AccountPage/HomePageRender/UserPageRender/MovieStreaminPageRender/AdminPage:** Container/render of a page based on its name**.**

4.7 Package Diagram

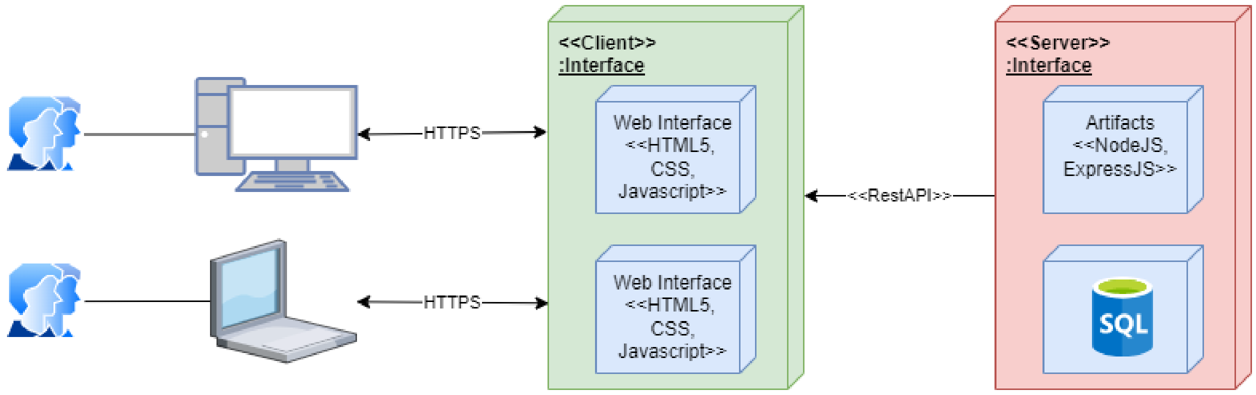


We built our web application using the following client-server model.

- To interface with the client, we have Users API and Admin API on the server side. We also saved Administrator and User information. We have controllers such as Users List, Movie List, Viewer, and so on, as well as a Database Access Service that allows us to access the SQL Server database.
- We also have Users API, Admin API, Admin information, Users information, and three controllers on the client side.



5. Deployment

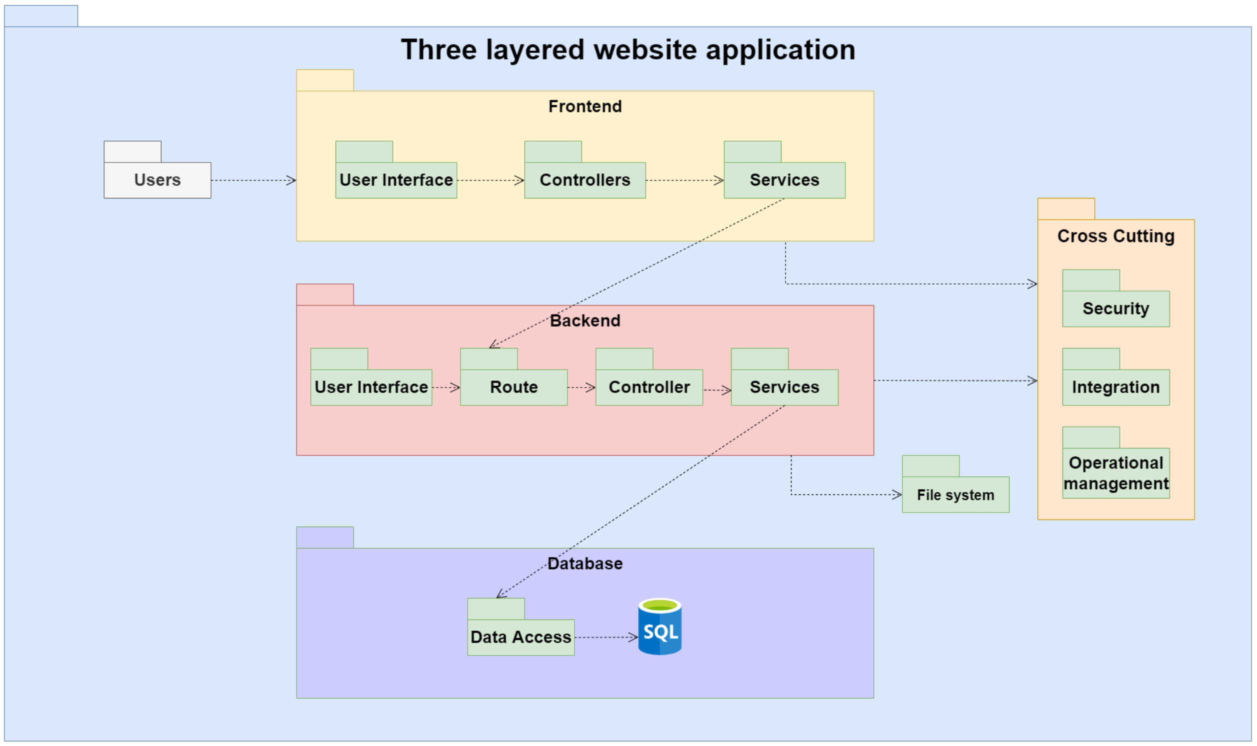


HTTPS
HTTPS
<<Client>>
:Interface
Web Interface <<HTML5, CSS, Javascript>>
Web Interface <<HTML5, CSS, Javascript>>
<<RestAPI>>
<<Server>>
:Interface
Artifacts <<NodeJS, ExpressJS>>
SQL

We aim our consumers to be able to use our product on any browser, including Google Chrome, Microsoft Edge, Firefox, Safari, Opera, etc on PCs or laptops running compatible operating systems like Windows, MacOS, Ubuntu, and so on.

We do not intend to develop our product on smartphones. Therefore it is a challenge for us because people nowadays use smartphones to watch or stream a movie anywhere and whenever they want rather than using their PC.

6. Implementation View



We saved certain information about the users in the **Users Folder**, such as username, password, fullname, DOB, address, and so on.
We separated the **Frontend** part into three subsections:

- The **User Interface Folder** has components, frameworks, libraries, etc.
- The **Controller Folder** includes the UI Render Controller, Movie Streaming Controller,...
- The **Services Folder** has a Listen and Request log, as well as a Route for interacting with the Backend.

We separated the **Backend** part into four subsections:

- The **User Interface Folder** has commands, components,...
- The **Route Folder** includes the Listen function and is used to respond from Frontend.
- The **Controller Folder** contains the RequestHandler function to manage services
- The **Services Folder** includes Users List controller, Movie List controller, Viewer controller, etc along with a Database Access Service that provides access to the Database folder.

We separated the **Database** folder into two independent folders:

- **The Data Sources Folder** includes the structure for the data and database.
- **The Data Access Folder** includes the way to query the SQL database.

We have three folders in the **Cross Cutting** section:

- The **Security Folder** includes information on how we encrypt user passwords using SHA-256 hash algorithms.
- The **Integration Folder** offers information on how we use GitHub and GitLab to integrate our application.
- The **Operational Management** section covers our policies for implementing or testing our application.

The File System Folder is where we split and manage our files.