Group 08

MOVIE STREAMING WEBSITE
Software Architecture Document

Version <1.1>

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

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

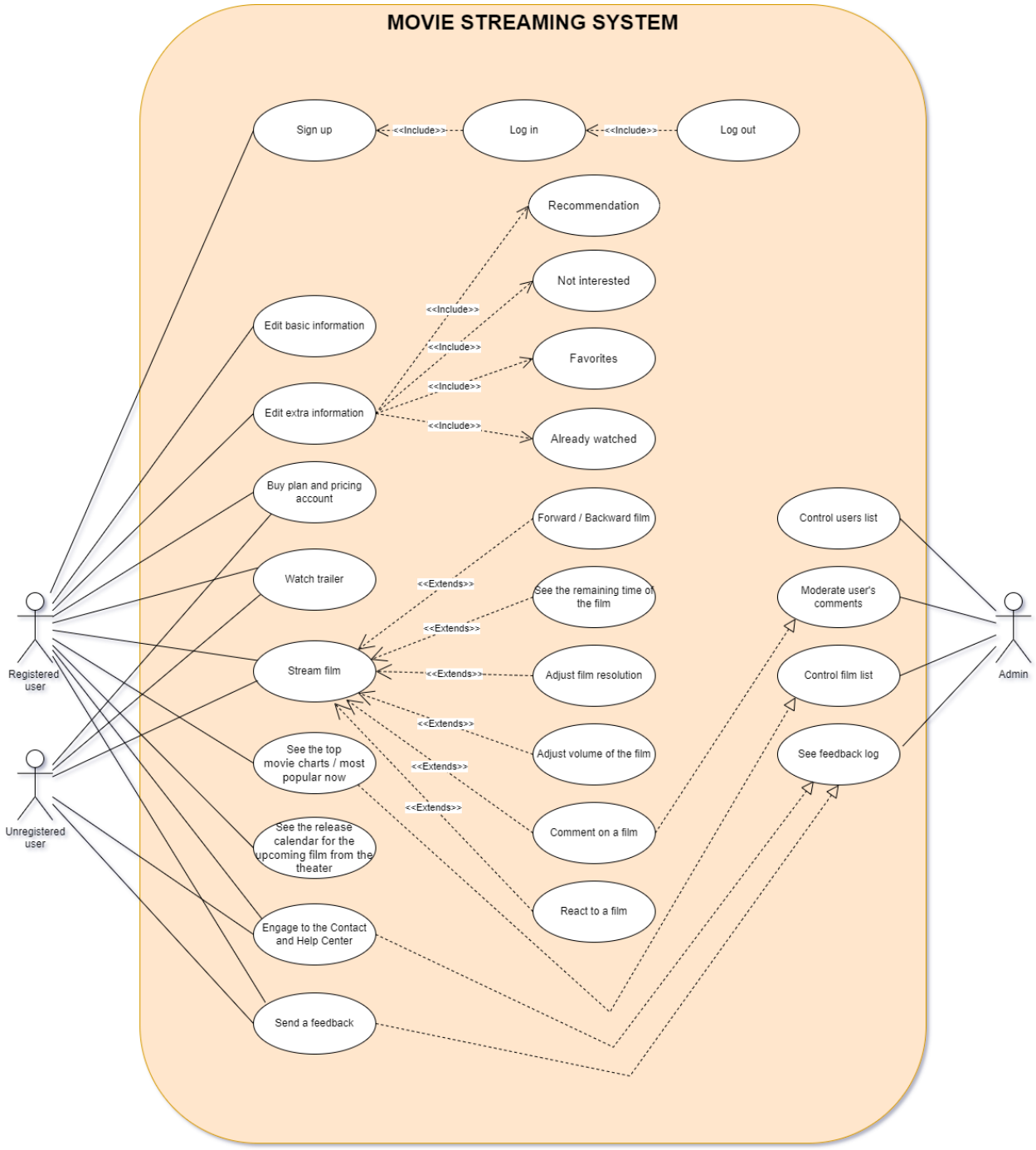
Our website is an off-the-shelf project such as our price which is affordable for everyone to use in order to have a better movie-watching experience. Alternatively, the website allows customers to view movies at any time and from any location. We have an expert team to test the product, which has been 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.

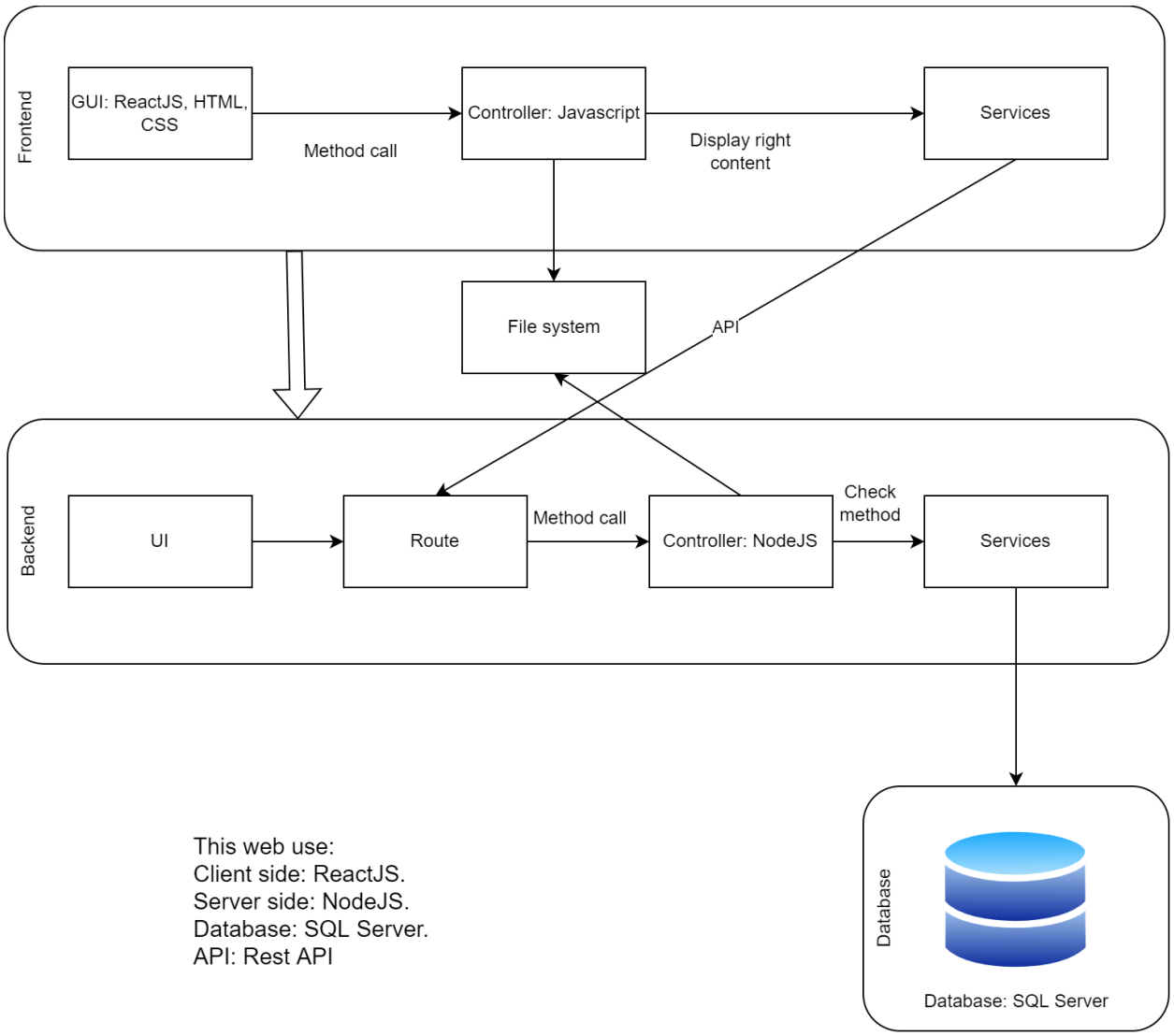
User’s privacy and information security: Protect all user personal information and only analyze user actions such as liked/disliked movies for our own movie recommendation usage and product improvement.

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


MOVIE STREAMING SYSTEM
Sign up
Log in
Log out
<<Include>>
<<Include>>
Recommendation
Not interested
Favorites
Already watched
Edit basic information
Edit extra information
Buy plan and pricing account
Watch trailer
Stream film
See the top movie charts / most popular now
See the release calendar for the upcoming film from the theater
Engage to the Contact and Help Center
Send a feedback
<<Include>>
<<Include>>
<<Include>>
<<Include>>
<<Extends>>
<<Extends>>
<<Extends>>
<<Extends>>
<<Extends>>
<<Extends>>
Forward / Backward film
See the remaining time of the film
Adjust film resolution
Adjust volume of the film
Comment on a film
React to a film
Control users list
Moderate user's comments
Control film list
See feedback log
Registered user
Unregistered user
Admin

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.

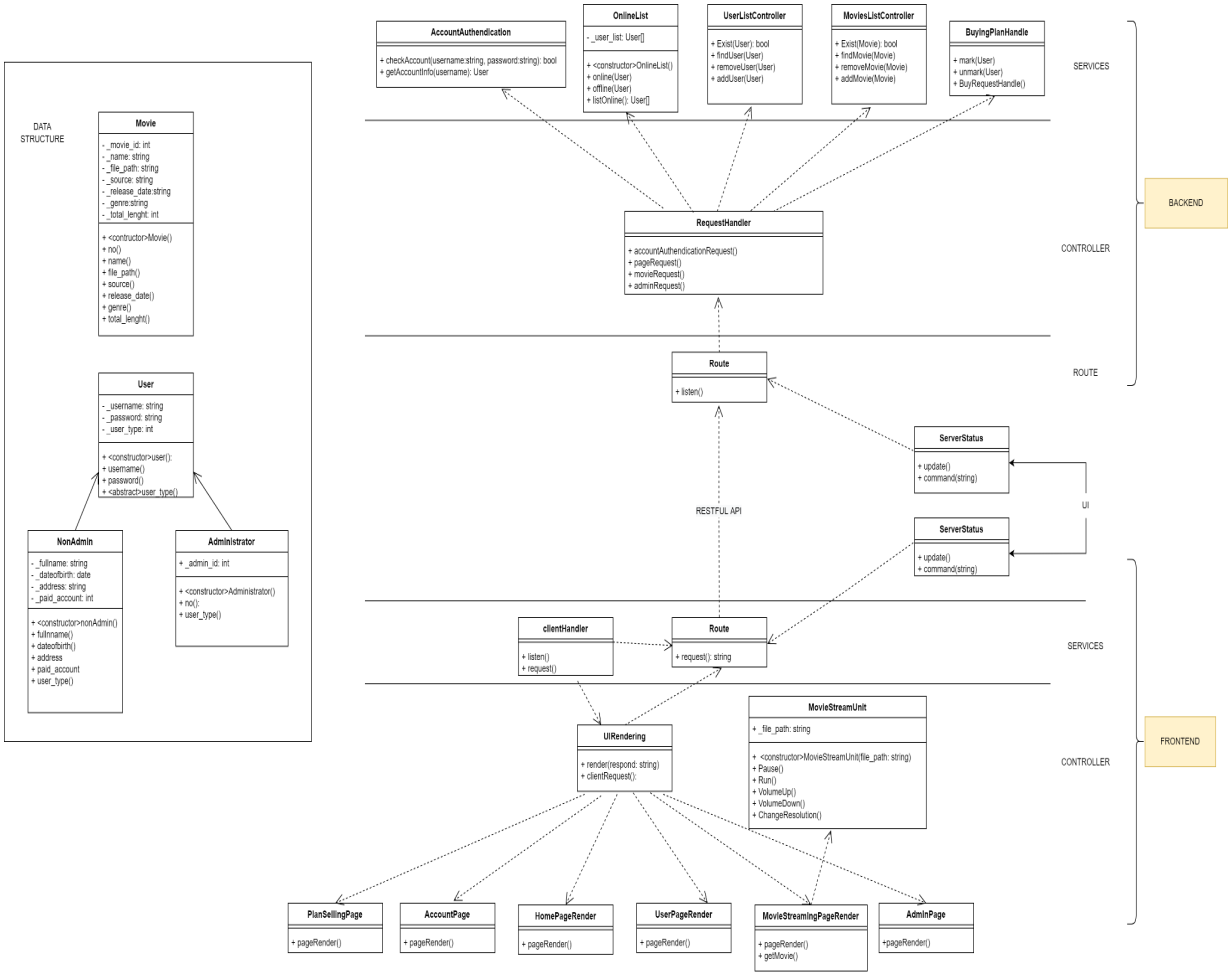
In the Route layer, we develop general APIs to handle, coordinate and distribute calls to other

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.

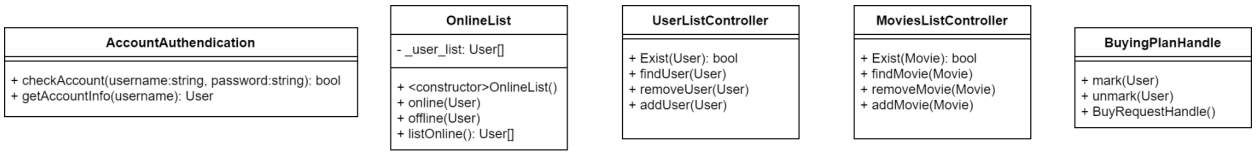
Then the Services layer queries to the Database and responds to all the requests from the Backend. We use SQL Server from Microsoft to query easily.

4.1 Class Diagram Overview



4.2 Component: Backend - Service

4.2.1 Class Diagram



4.2.2 Detail

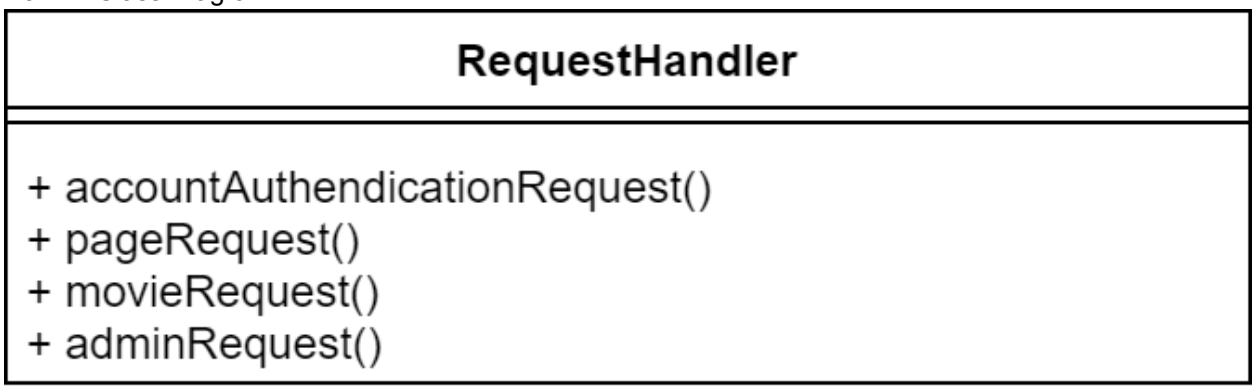
Backend service mainly accessing databases for all operations from the UI, all these services will be used by

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.

4.3 Component: Backend - Controller

4.3.1 Class Diagram

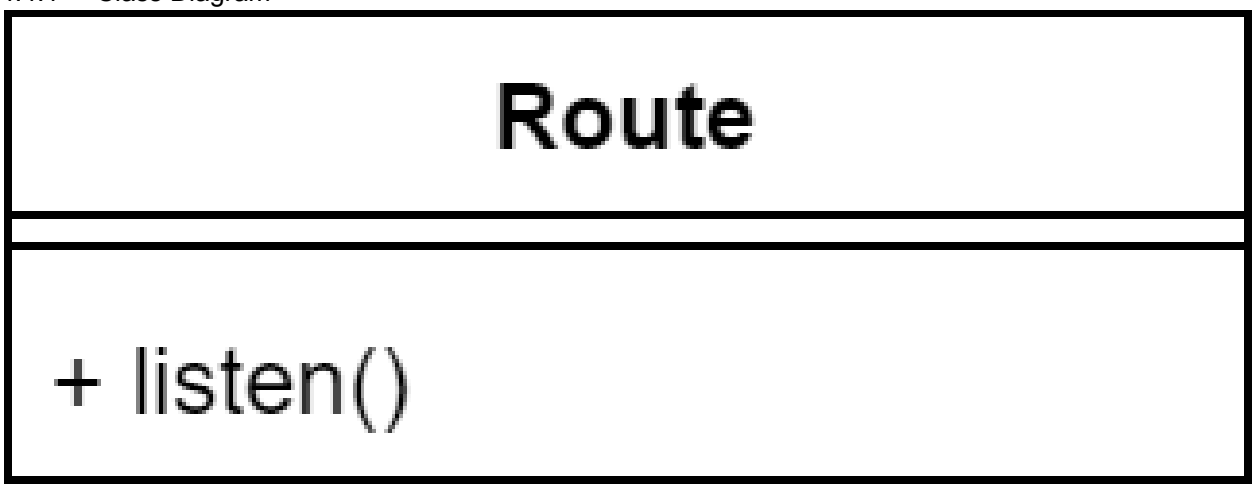


4.3.2 Detail

This component will take care of the incoming request from the route component, also accessing the file system if necessary; each method in **RequestHandler** will handle a specific task.

4.4 Component: Backend - Route

4.4.1 Class Diagram

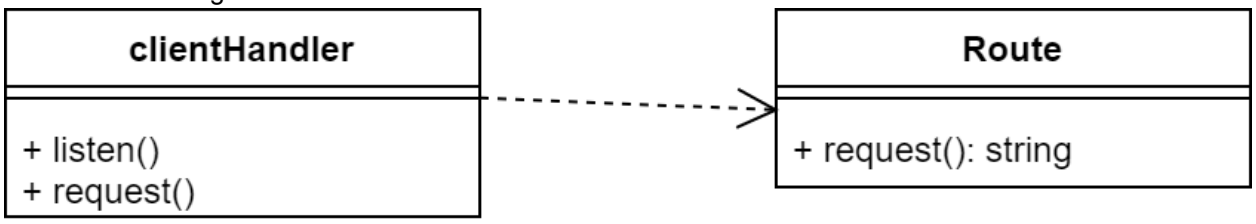


4.4.2 Detail

This component will handle incoming API requests from the frontend and parse it into a working command for the backend system to operate. When a request is complete, the route component will respond to the frontend side. This also will handle any command that the operator needs to use.

4.5 Component: Frontend - Services

4.5.1 Class Diagram



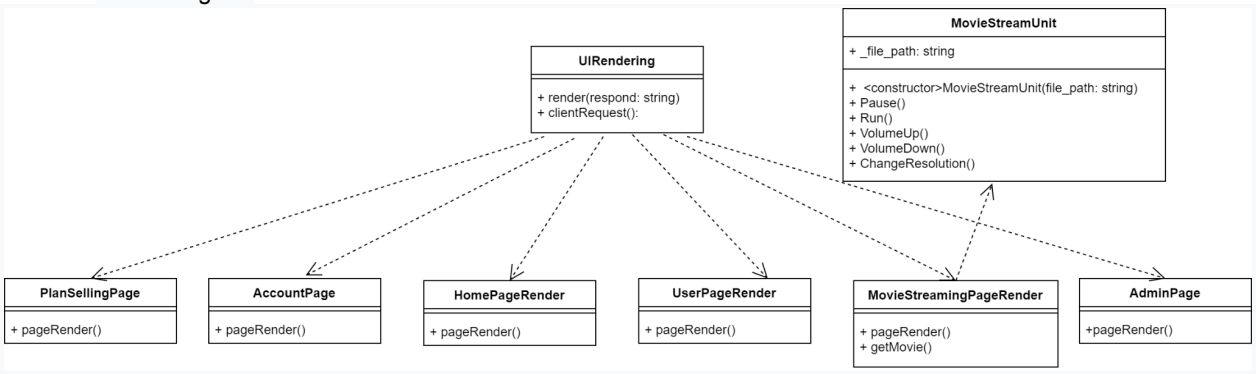
4.5.2 Detail

Service component will handle any incoming user request and interact directly with the backend side

- **clientHandler:** receive incoming request and parse it to the renderUI or Route for request.
- **Route:** 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

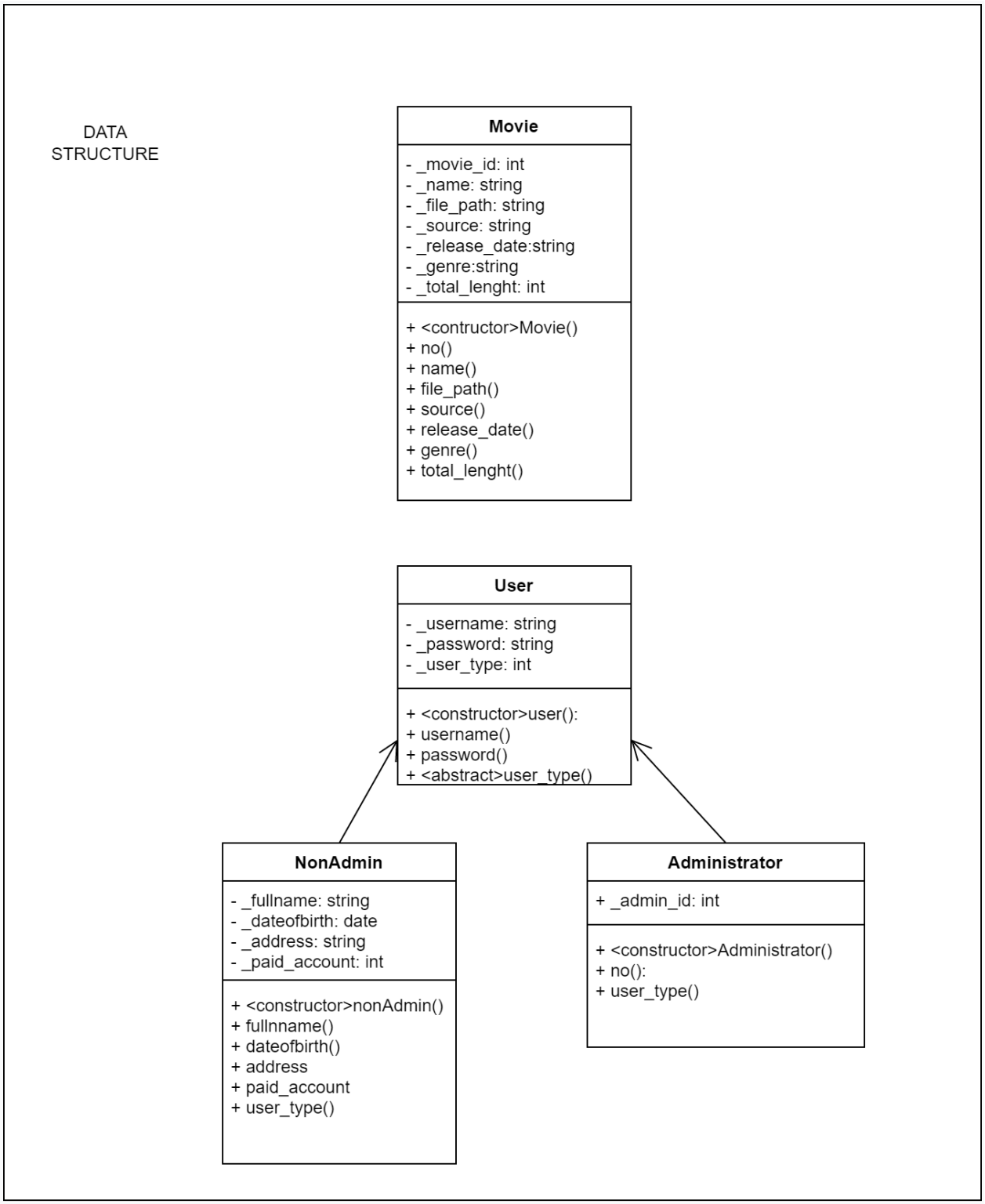


4.6.2 Detail

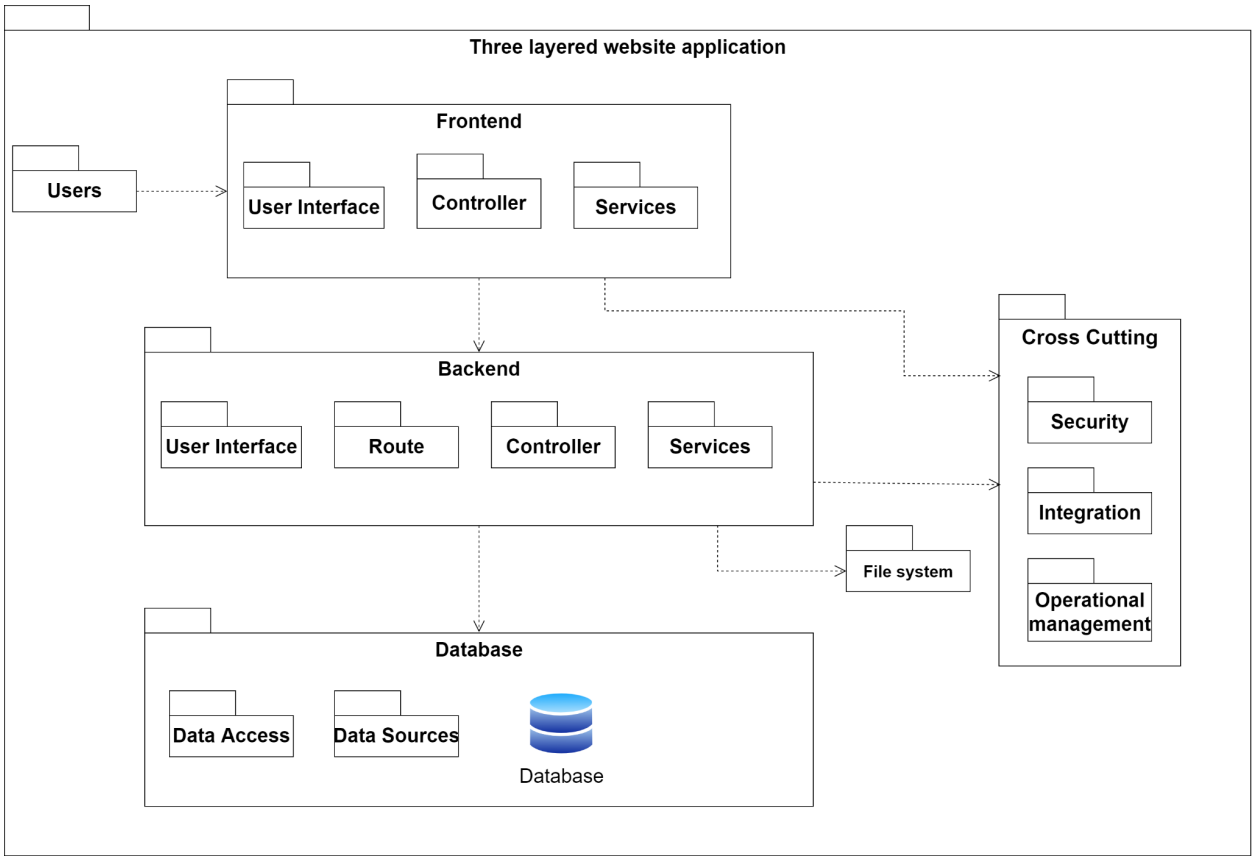
Controller will render the GUI for user using the user request and data get from backend side

- **UIRendering**: handle the GUI part.
- **PlanSellingPage:** handle the render of plan selling.
- **AccountPage:** handle the render of the user account.
- **HomePageRender:** handle the render of the home page.
- **UserPageRender:** handle the render of the user page.
- **MovieStreamingPageRender:** handle the render of movie streaming.
- **AdminPage**: handle the render of the admin page.
- **MovieStreamUnit**: access the file system for streaming and managing any streaming related command.

4.7 Data Structure Class Diagram



4.8 Package Diagram



5. Deployment

[Leave this section blank for PA3.]

6. Implementation View

[Leave this section blank for PA3.]