<Group 01>

<Web Booking Movie>
Software Architecture Document

Version <1.0>

Revision History

Date	Version	Description	Author
09/12/2022	1.0	Define software architecture	Group 01
22/12/2022	1.1	Review architecture and class diagram, add deployment information and implementation view.	Group 01

Table of Contents

Software Architecture Document

1. Introduction

The introduction of the Software Architecture Document of Web Booking Movie provides an overview of the entire Software Architecture Document. It includes the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the Software Architecture Document.

2. Architectural Goals and Constraints

Performance:

- Focus should be on front-end optimizations by
- Using fast server to handle traffic
- Providing cross-browser compatibility
- Using minification
- Reducing HTTP request

Usability:

- User should be at ease in using website without any specialized training
- User should be able to relate further action needed for an interaction
- Web application should not have any ambiguity regarding the consequences of an action

Accessibility:

- The system has many user support functions (especially disabled person)

Security:

- Protecting information from external attacks by mean of
- Authorization
- Authentication
- Encryption
- Session Management
- Exception Management

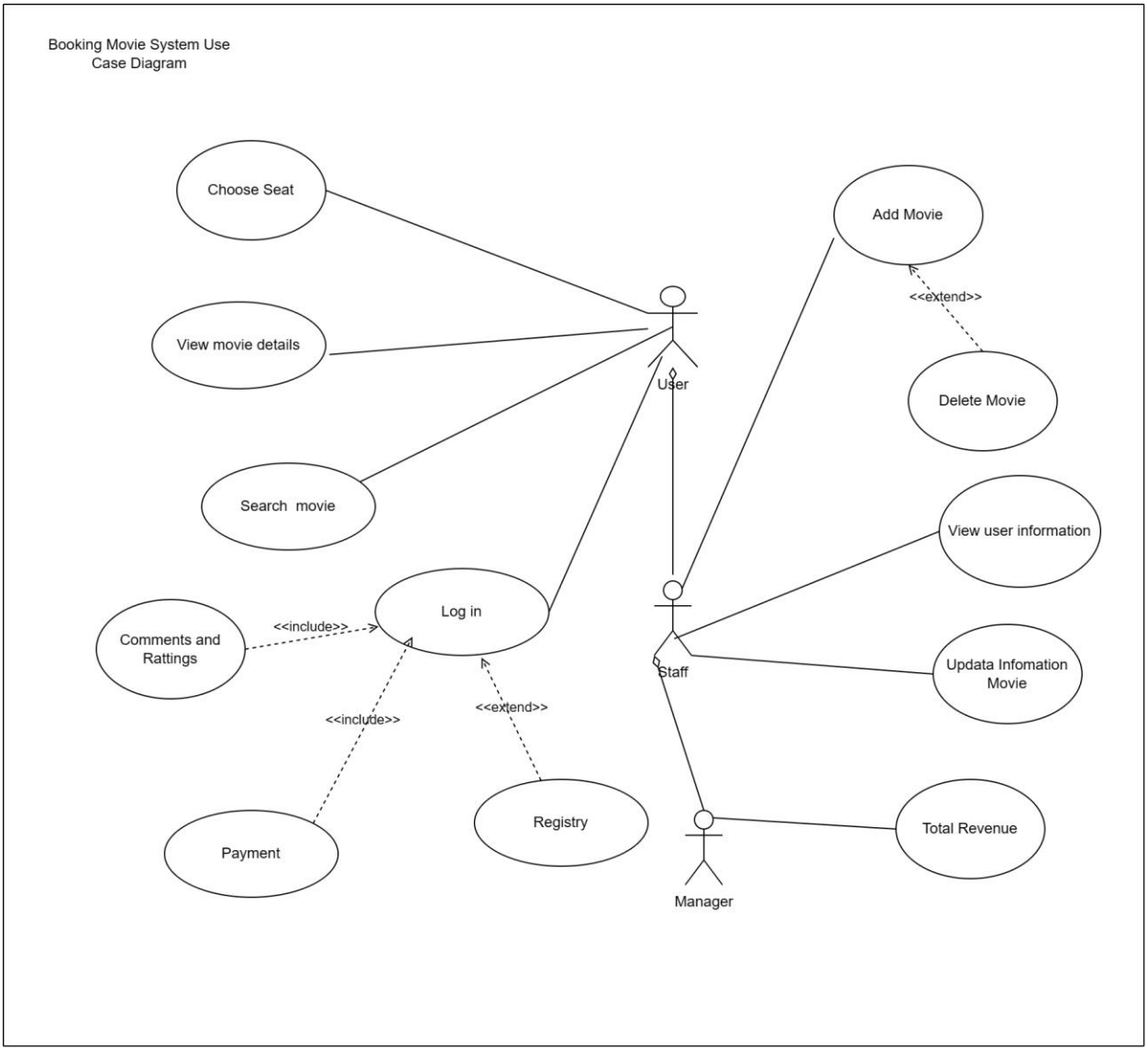
Maintainability

- It refers to capability of web system to maintain its performance over time.
- We can assure reliability by making website robust, recoverable and available under adversity.

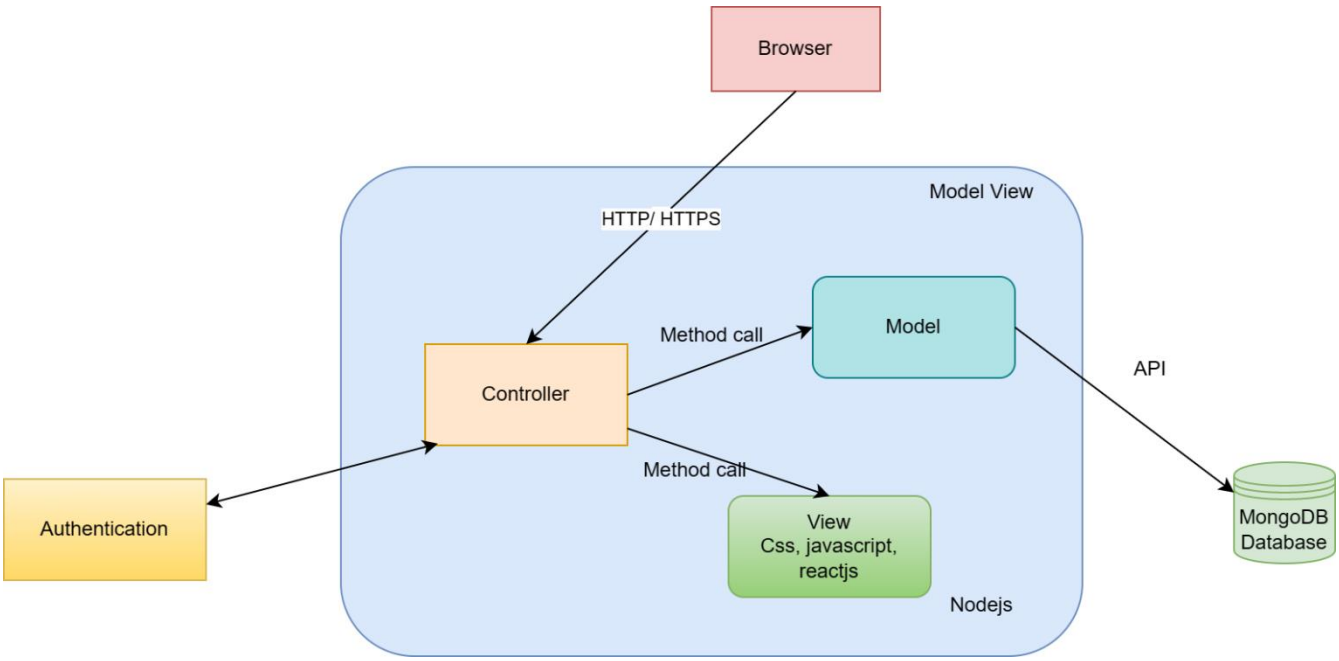
User Interface:

- Interface designs must accomplish any task with efficiency and effectiveness
- To reduce the weight of the users, interface designs must be lightweight and provides information in a more subtle, succinct manner.

3. Use-Case Model



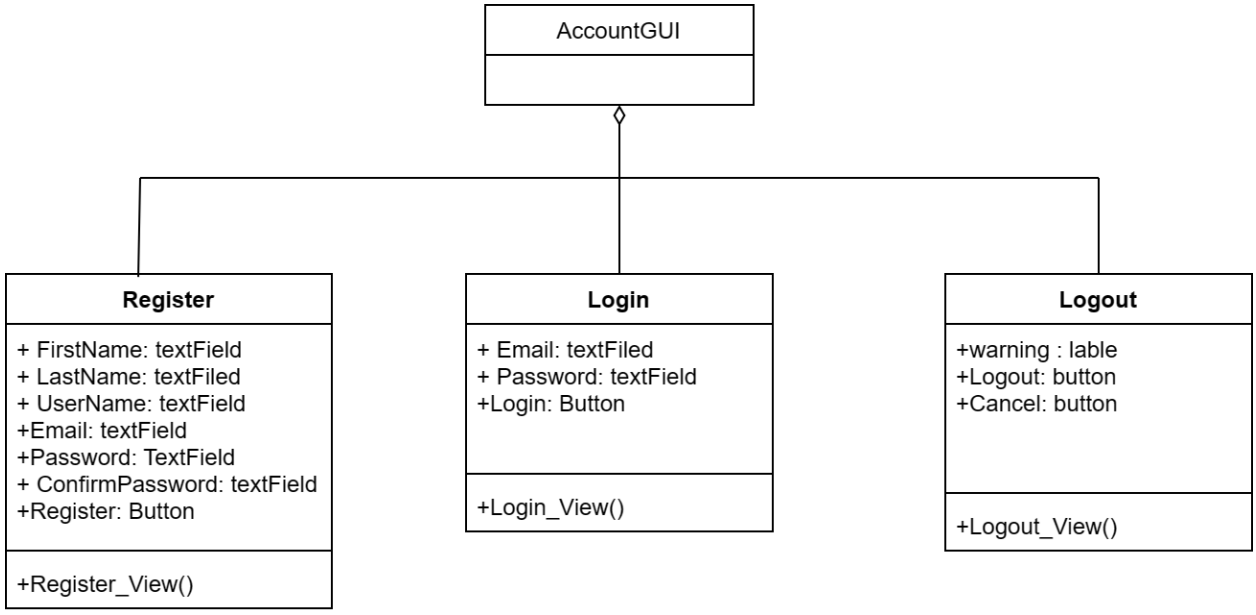
4. Logical View



4.1 Component: View

View includes 2 small components : AccountView, User/AdminView

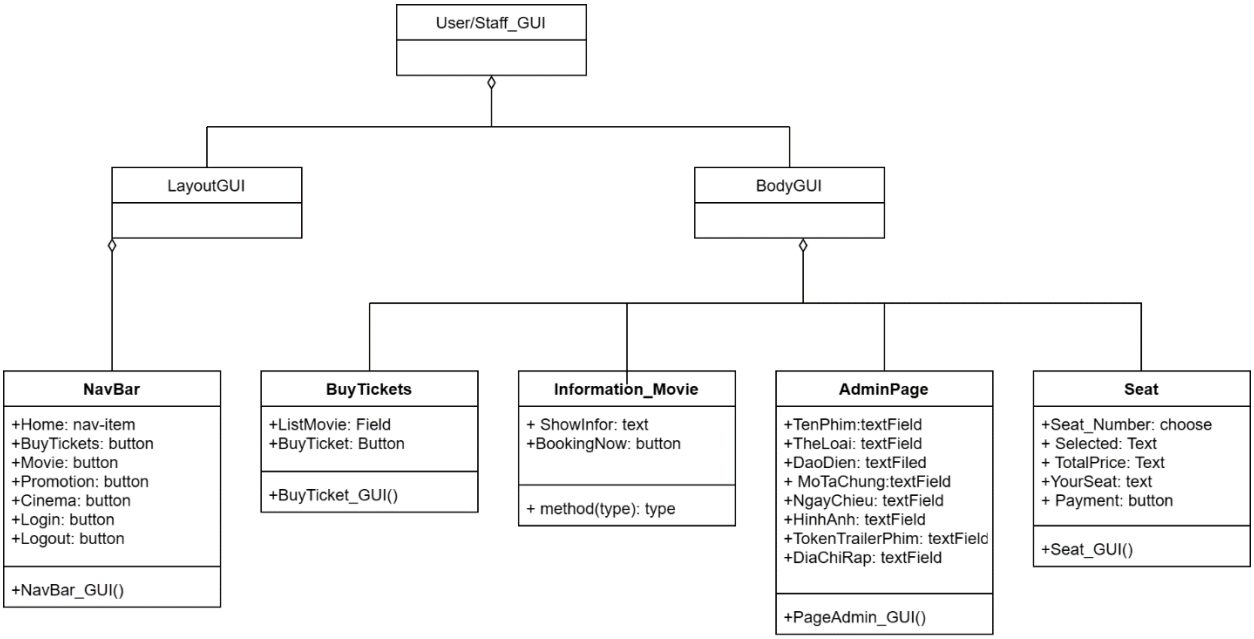
a. Account View:



AccountGUI has 5 main pages: **Login, Register** and **LogOut**

- **Register**: page where the user signs up an account. The user needs to fill registration and submit to create an account. Web Client will send a RESTful API (POST) request to the system. If the user has already created an account, click on “Login” to go to the **Login** page.
- **Login**: page where the user logs in. After the user fills the login form and submits, the system will check whether the account is valid. If successful, the user can go to the **Home** website, otherwise, the user has to login again. In case the user haven’t had an account, click on “Register” link to go to **Login** page Users . It will navigate to Google's website to authenticate the account.
- **LogOut**: user can log out by pressing this button.

b. User/Admin View:

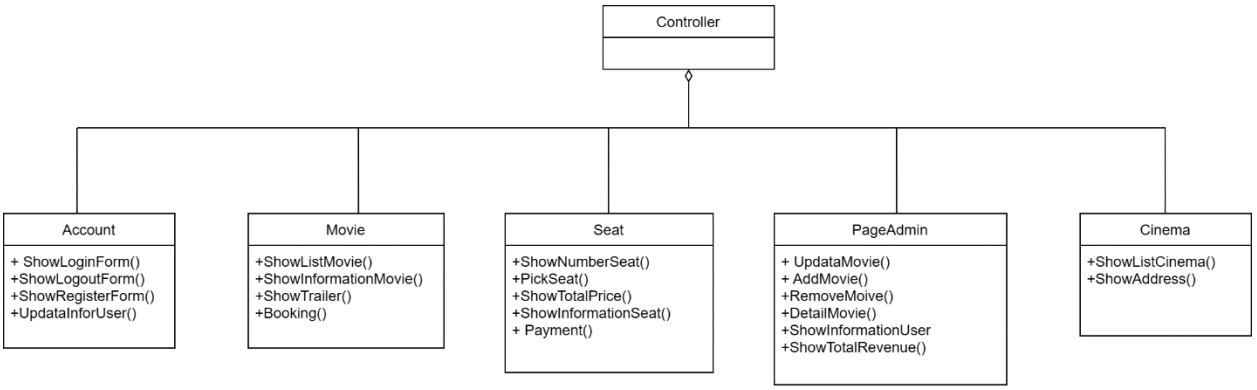


User/Staff_GUI have 2 main parts : **LayoutGUI & BodyGUI**

- **LayoutGUI**: This is the part which have a layout home and all button such as buytickets, movie, promotion, cinema, login, logout. When we press on the button, adjust direct into function page
- **BodyGUI**: + At the page BuyTicket_GUI, The user can see a list of movie and when wanting to watch film, then click BuyTickets, after that they see more information about the film and click BookingNow to reservation. At the page Seat_GUI, the user can choose their seat and payment total price.

  +At the page PageAdmin_GUI, The admin can add more movie by TenPhim, TheLoai, DaoDien, MoTaChung, NgayChieu, HinhAnh, TokenTrailerPhim, DiaChiRap.

4.2 Component: Controller



Controller has 5 main parts:

+ **Account:** This is the part in order to show login form, log out form, register form and update information of user

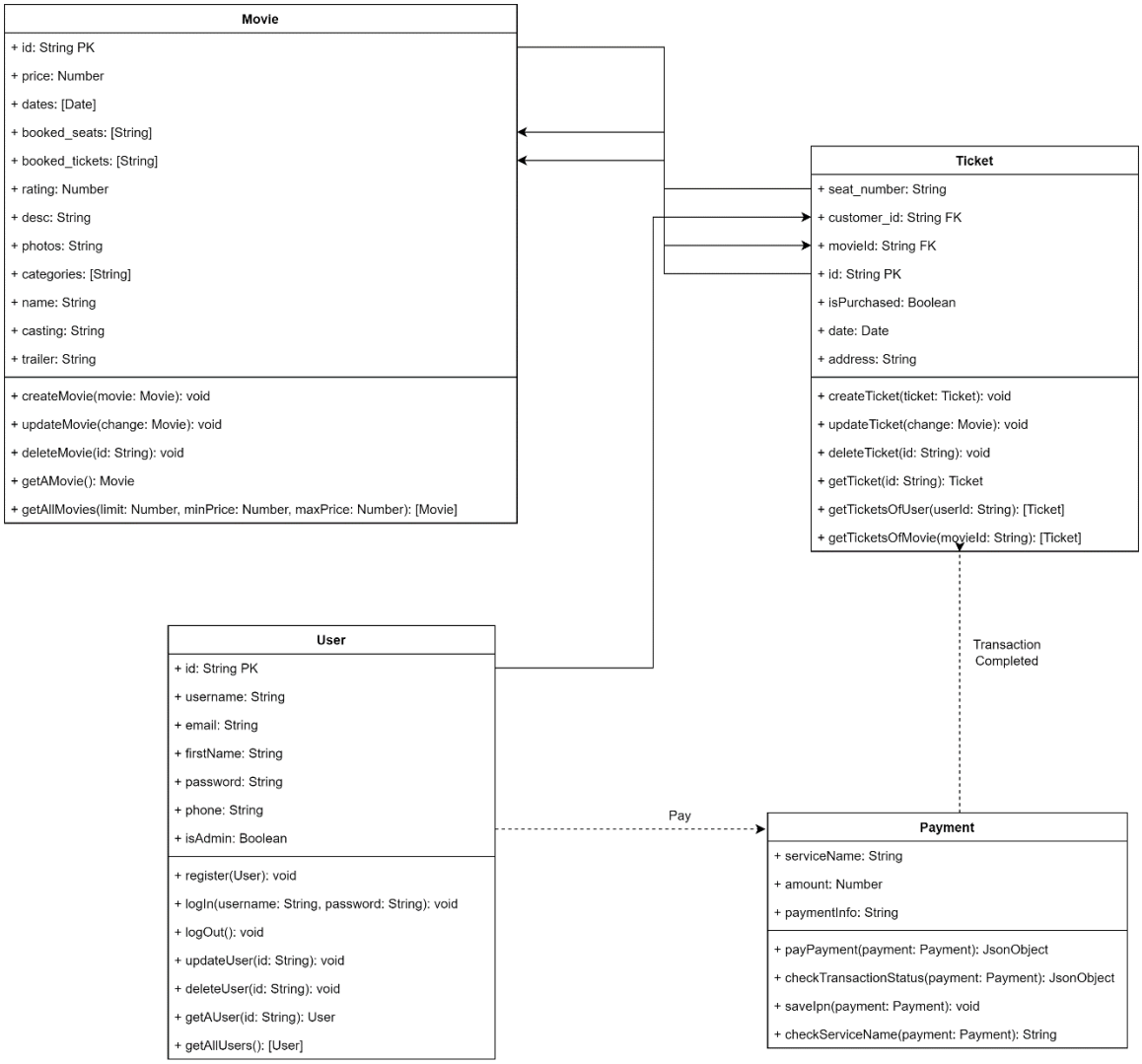
+ **Movie**: This is the part in order to show list of movie, information about movie, show trailer and booking the seat

+ **Seat**: This is the part in order to show number of seat, choose seat, show total price, show information about the seat, pay the bill.

+ **PageAdmin**: : This is the part in order to update movie, add movie, remove movie, information about movie, information about user and total revenue.

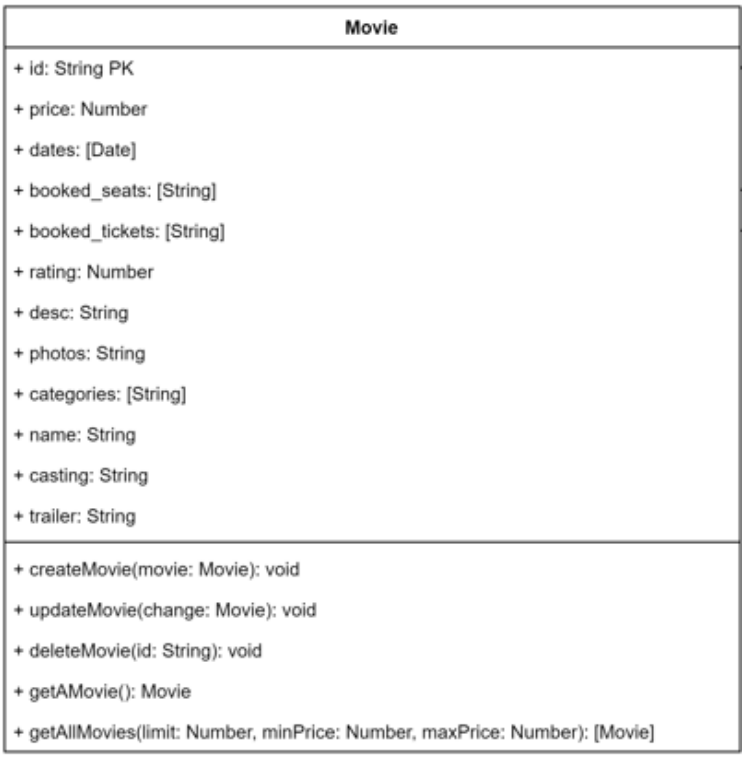
+ **Cinema**: This is the part in order to show list and address of the cinema

4.3 Component: Model



a. Movie

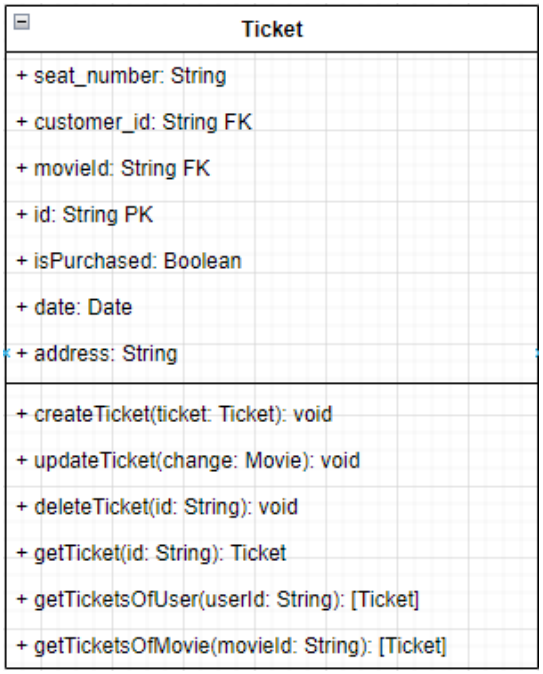
Class: Movie



Atributes:
- id: String → id of movie.
- price: Number → price of movie .
- date: [Date] → movie schedule.
- booking-seats: [Number] → number of seat booked.
- booking-tickets: [String] -> list of booked ticket id
- rating: Number → rating of movie .
- desc: String→ describe information about movie .
- photos: String → poster of movie .
- address: String → address of cinema where movie showing.
- categories: [string] → categories of movie
- name: string → name of movie.

Behaviors:
- createMovie(movie: Movie) : void → create a new movie in the system.
- updateMovie(change: Movie): void → update more information of movie.
- deleteMovie(id: String): void → delete a movie from the system.
- getAMovie(): Movie → get information on a movie in the system.
- getAllMovie(): [Movie] → get information on all movies in the system.

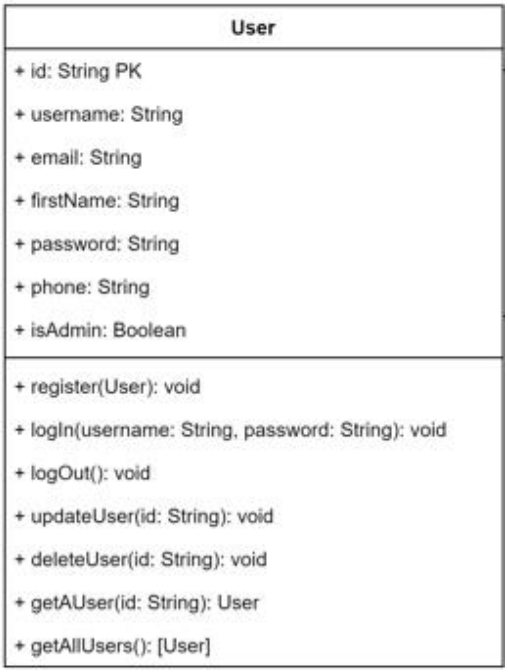
b. Ticket



Class: Ticket
Atributes:
- seatNumber: string → number of booked seat.
- customerId: string → id of user when buying the ticket.
- movieId: string → id of movie.
- id: string → id of ticket .
- isPurchased: boolean → Check the transaction status
- date: Date → date of ticket creation.
- address: String → address of cinema.

Behaviours:
- createTicket(movieId: string) : void → create a ticket in the system.
- updateTicket(id: string): void → update information of ticket.
- deleteTicket(id: string): void → delete ticket from database.
- getTicket(id: string): void → get information of a ticket in the system.
- getTicketOfUser(user_id: string): void → get information of all booked tickets by user id in the system.
- getTicketOfMovie(movieId: string): void → get information of all booked tickets by movie id in the system.

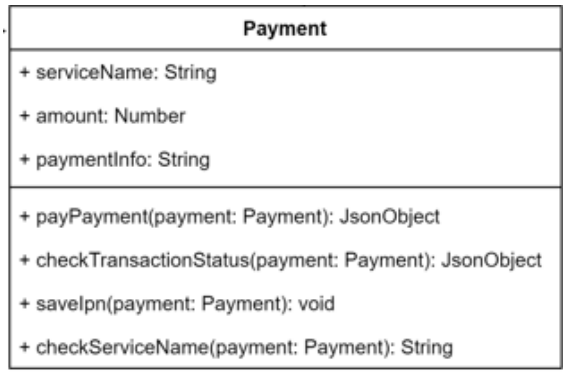
c. User



Class: User
Atributes:
- id: string → id of user
- usename: string → usename of user
- email: string → email's use
- firstName: string → firstname's user
- password: string → password
- phone: string → phone number
- lastName: string → lastname's user
- isAdmin: boolean → check if the user is admin or not

Behaviours:
- register(): void → register a new account if the user has not yet
- logIn(type): void → after user has account, user can login in the system
- logOut(type): void → logout, use can logout when the user's account has login.updateUser(id: string): void → update more information of user in the system
- deleteUser(id: string): void → delete user from the system
- getAUser(id: string): void → get information of user
- getAllUser(): [User] → get information of all user

d. PaymentService

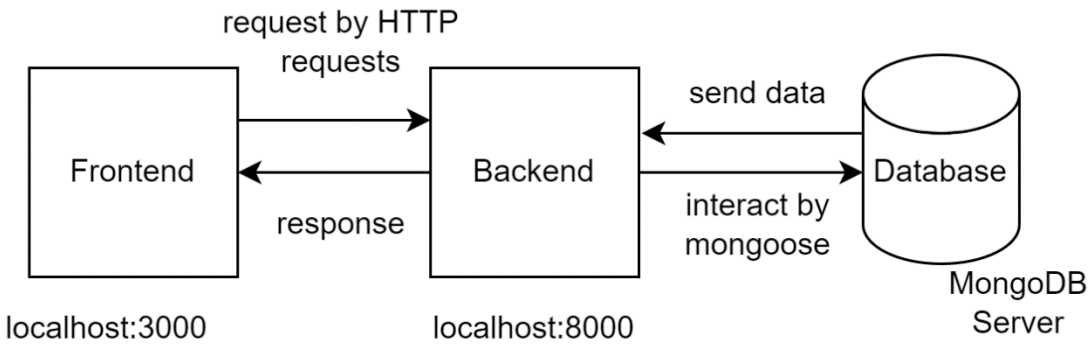


Class: Payment
Atributes:
- id: string → if of payment service.
- serviceName: string → name of service such as card, mono, vnpay,...

Behaviours:
- payPayment(payment: Payment): JsonObject → return link to third part service.
- checkTransationStatus(payment: Payment): JsonObject → JsonObject

5. Deployment

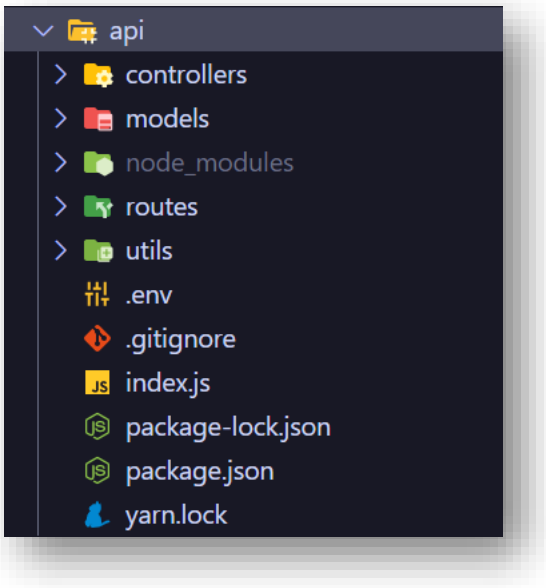
- Our webapp includes of 3 main parts: Frontend, Backend and Database.
- Database of system is in MongoDB server. By providing connection link in file environment variables, our Backend can connect with data in the server.
- Our backend will be deployed in another server, plays the role that supply APIs for frontend interaction by HTTP requests, and interacts with data in Database.
- Frontend interacts with users as well as backend, render and redirect to the appropriate site. It will be deployed in another server.



-

6. Implementation View

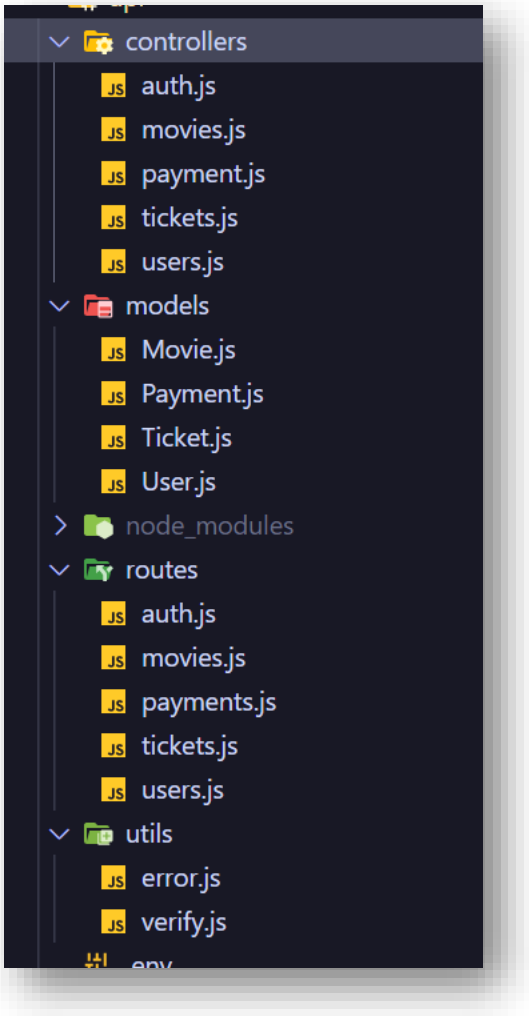
6.1 Backend



- Backend folder contains 4 main folders: controllers, models, routes, utils and is managed by Node package

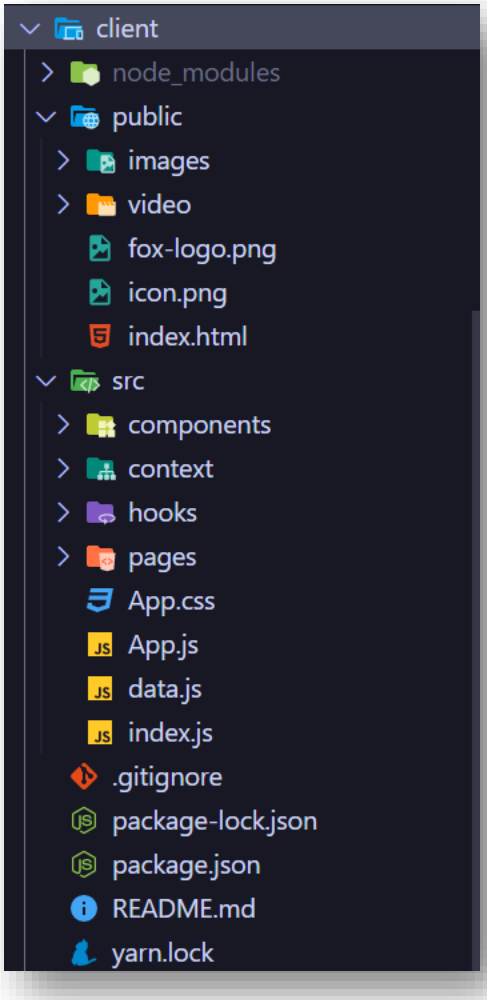
manager (npm).

- File .env stores important information like hash key for JWT.



- In folder routes, we define the api path for each route.
- In folder controllers, we implement logic code to return results to users for each api.
- In folder models, we define Schema model will be stored in MongoDb.
- In folder utils, we define support function for handling error as well as verify user for authentication.

6.2 Frontend



- We use React framework to develop our webapp, it separates client folder into two parts, public and src.
- In public, we store static images, logo, icon,…
- In main src, we have four folders components, context, hooks, pages.
- Context contains files Auth Context and Search Context to control authentication and search state over website.
- Hooks contains files that define custom hook function to use in development by React.
- Components contain files that define partials in a website like Header, Footer, Navbar,…
- Folder Pages store code rendering each page in website, coordinate with Context to render the appropriate site for users.