jcxdc

CaRe
Software Architecture Document

Version <1.1>

Revision History

Date	Version	Description	Author
21/11/23	<1.0>	Initial version	Triệu Nhật Minh
06/12/23	<1.1>	Revised version for PA4	Triệu Nhật Minh

Table of Contents

Software Architecture Document

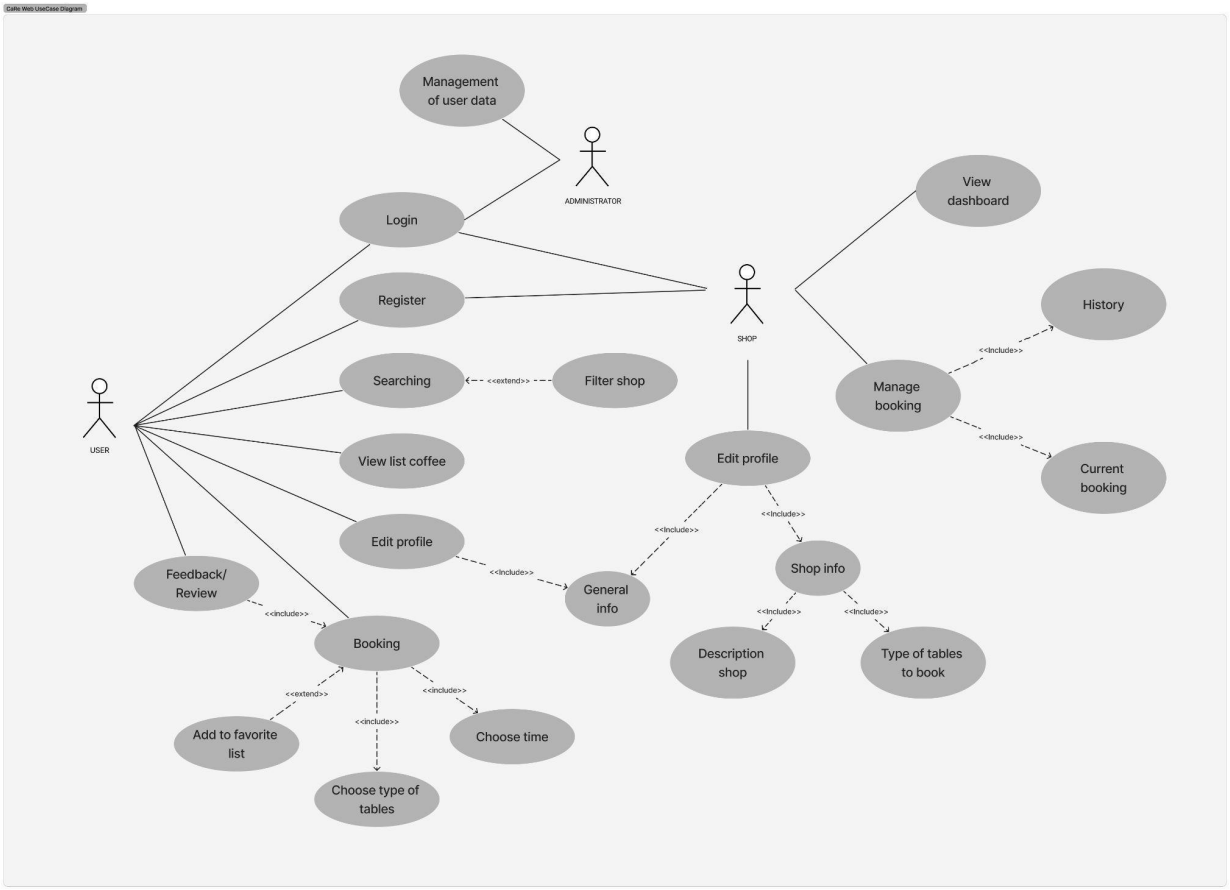
1. Introduction

The introduction of the **Software Architecture Document** 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**.

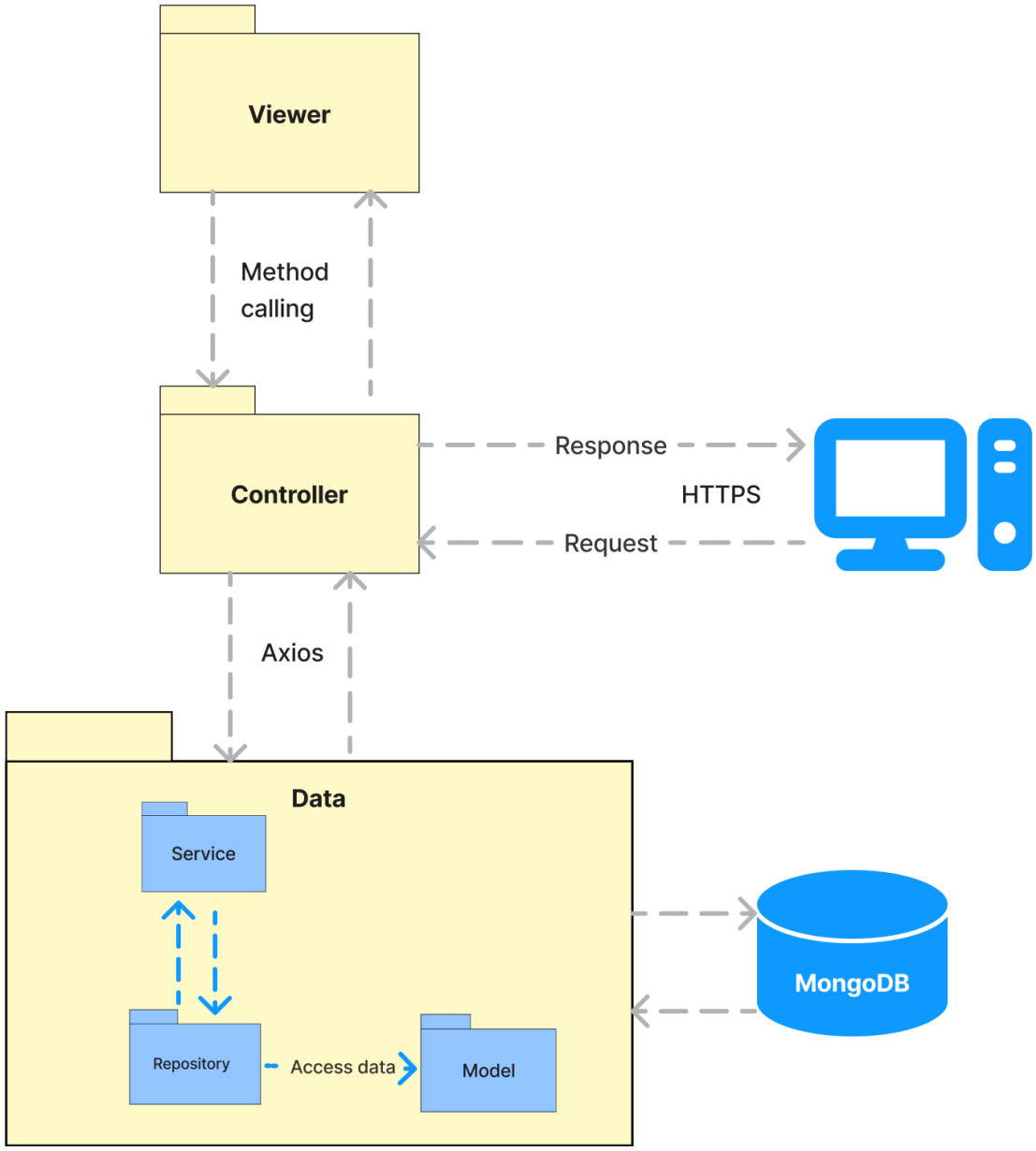
2. Architectural Goals and Constraints

1. Goals:
    a. Delivery: Finishing at least 3 features and delivery on time.
    b. Performance: Small time to load.
    c. Usability: Friendly interface and low perceived workload.
    d. Portability: Support in different types of devices (laptop, phone, tablet) and browsers (Firefox, Safari, Google Chrome).
2. Constraints:
    a. Backend technology: Node js (framework: Express js)
    b. Frontend technology: Reactjs
    c. Database: MongoDB
    d. Reliability: Verification is required.
    e. Interoperability: Password is encrypted.
    f. Efficiency: Limited the size of images uploaded.
    g. Legislative and ethical requirements: “trash” words are removed.

3. Use-Case Model



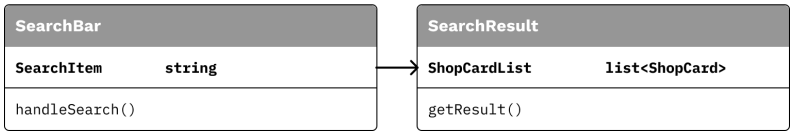
4. Logical View



4.1 Component: View

Our café critique platform, developed using ReactJS and MaterialUI, incorporates a View component that includes user interface elements such as the Café card component and the Navigation bar. These components enrich the user experience by offering pertinent information and features, like the ability to search for cafés and submit reviews. The “component view” not only boosts the visual attractiveness of the platform but also enhances its functionality and usability, thereby cultivating a vibrant community of café aficionados.

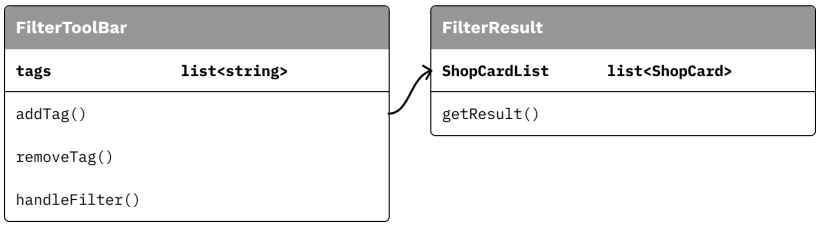
SEARCH



1. **Search:** The Search group is responsible for searching the coffee shops and then displaying them to the client.
   SeachBar:
   - Function: Get the search word from the client.
   - Flow: The client inputs the search word then the web will display the search result page.

   SearchResult:
   - Function: Display the search results based on the search word in the search bar.
   - Flow: Receive the search word from the search bar then get information from the database and display it.

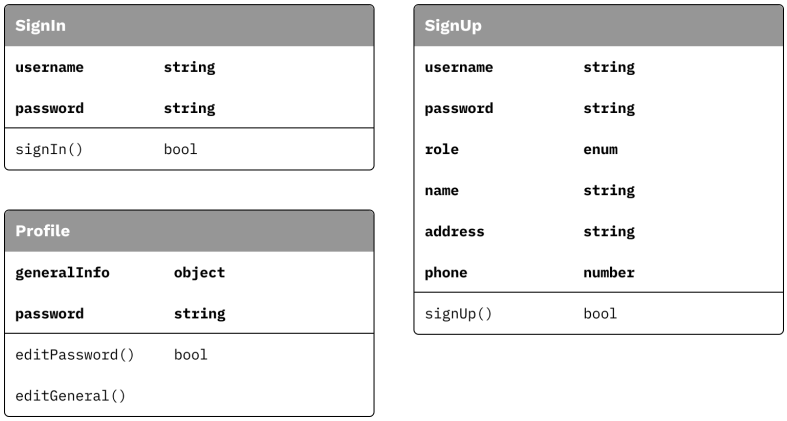
FILTER



2. **Filter:**
   FilterToolBar:
   - Function: Get the tag list of features which clients want in their coffee shops.
   - Flow: The clients click on some feature tags then the web will display the filter result page.

   FilterResult:
   - Function: Display the filter results based on the feature tags that have been chosen.
   - Flow: Receive the list of filter features from the list tags then get information from the database and display it.

USER



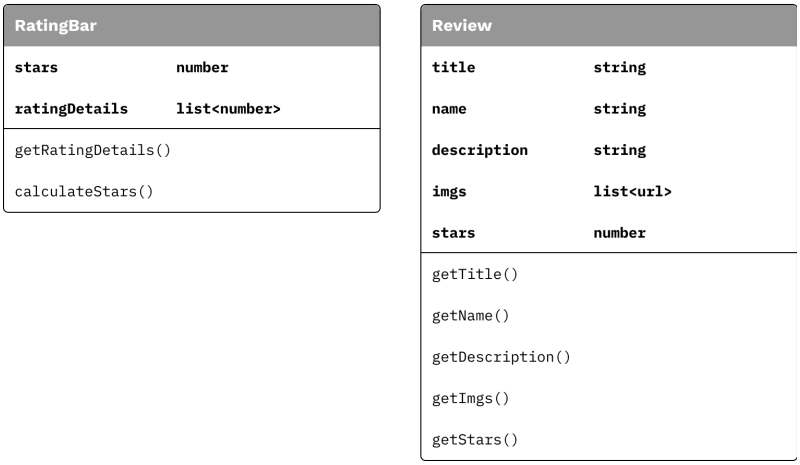
3. **User:** The User group is responsible for the client's login and signup.

SignUp
- Function: Display a form for clients to input information for their new account.
- Flow: The clients input information and then inform the client about their registration.

SignIn
- Function: Display a form to get a username and password of users.
- Flow: The client's input information then informs the client about their sign-in.

Profile
- Function: Display a section for users to view general information of their account and edit it if necessary.
- Flow: The client's click on the button edit to input new information then submit, information will be updated.

RATING

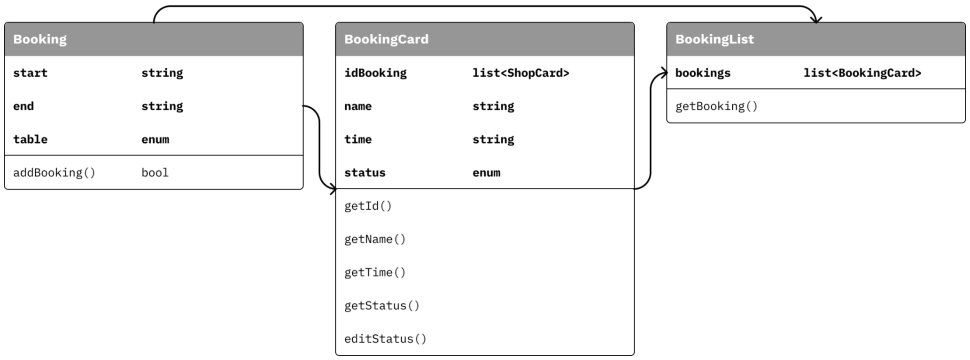


4. Rating:

RatingBar:
- Function: Display a rating bar for users in order to give them an overall view about the service of that coffee shop through the average number of stars. Moreover, it also displays all ratings that have been made by other customers in different categories from 1 to 5 stars.
- Flow: When the user clicks on any specific shop, the rating bar of that shop will be shown.

Review:
- Function: Display all reviews have been made by users (the number of stars, their comment and their feedback photos).
- Flow: Similar to the rating bar, it will be shown after users click on any specific shop.

BOOKING



5. **Booking:**

Booking:

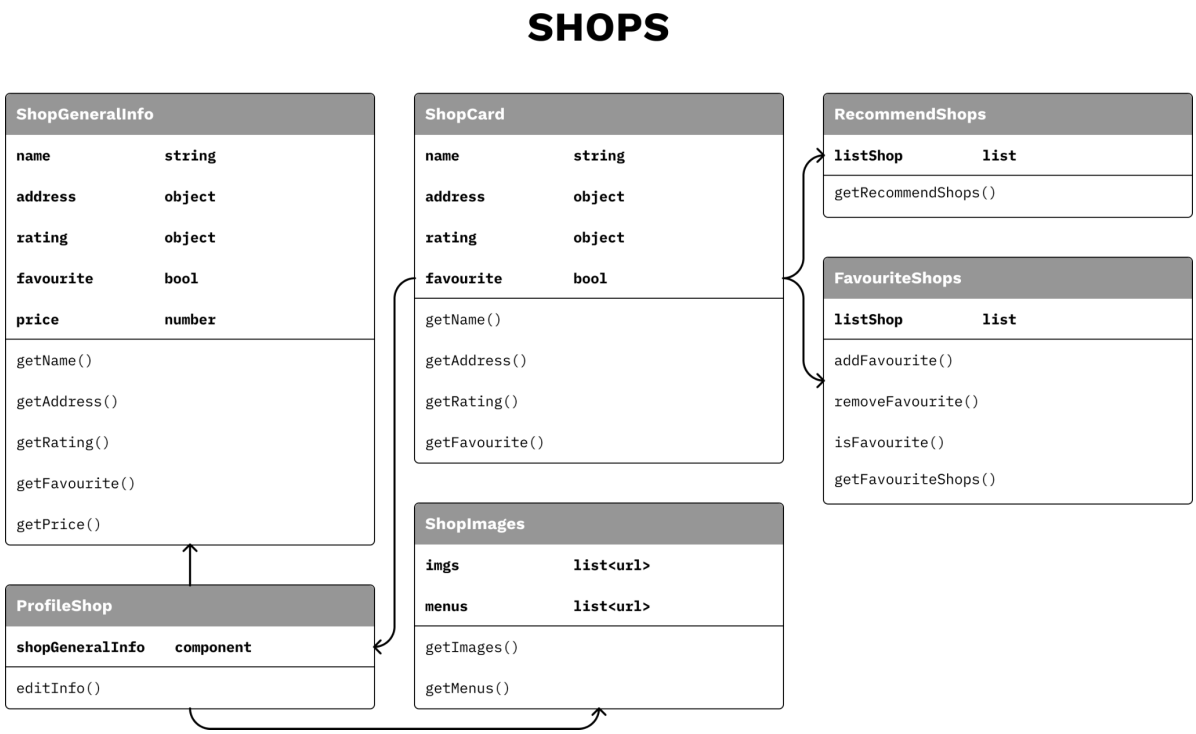
- Function: Display a form for clients to input information for their new booking.
- Flow: The client's input the start and end times as well as the type of table they want to choose before submitting and wait for the system confirmation.

BookingCard:

- Function: Display the order information of a user.
- Flow: When a user clicks on 'My order' field in their personal profile, each booking card is displayed as a row of a table.

BookingList:

- Function: Display all order information of a user.
- Flow: When a user clicks on 'My order' field in their personal profile, the booking list will be shown for the user in order to help them easily check their history of orders.



6. **Shops:**

ShopCard:

- Function: Show the information about shops like name, address, rating, favorite, etc.
- Flow: The information of the shop will be displayed at ShopCard, when users interact with the Shop Card, for example, by clicking the heart button, this event is forwarded to the Favorite shop component to add the song to the user's list of favorites. Users click on the whole card, the website navigates to Profile Page.

ProfileShop:

- Function: Show all the information about a shop, from general info to images.
- Flow: The profile shop includes the shop's images section and general info section.

ShopImages:

- Function: Show all the images of the shop including space and menus.
- Flow: The section receives from the database the list of images and renders it.

ShopGeneralInfo:

- Function: Show the information about shops like name, address, rating, price and whether the shop

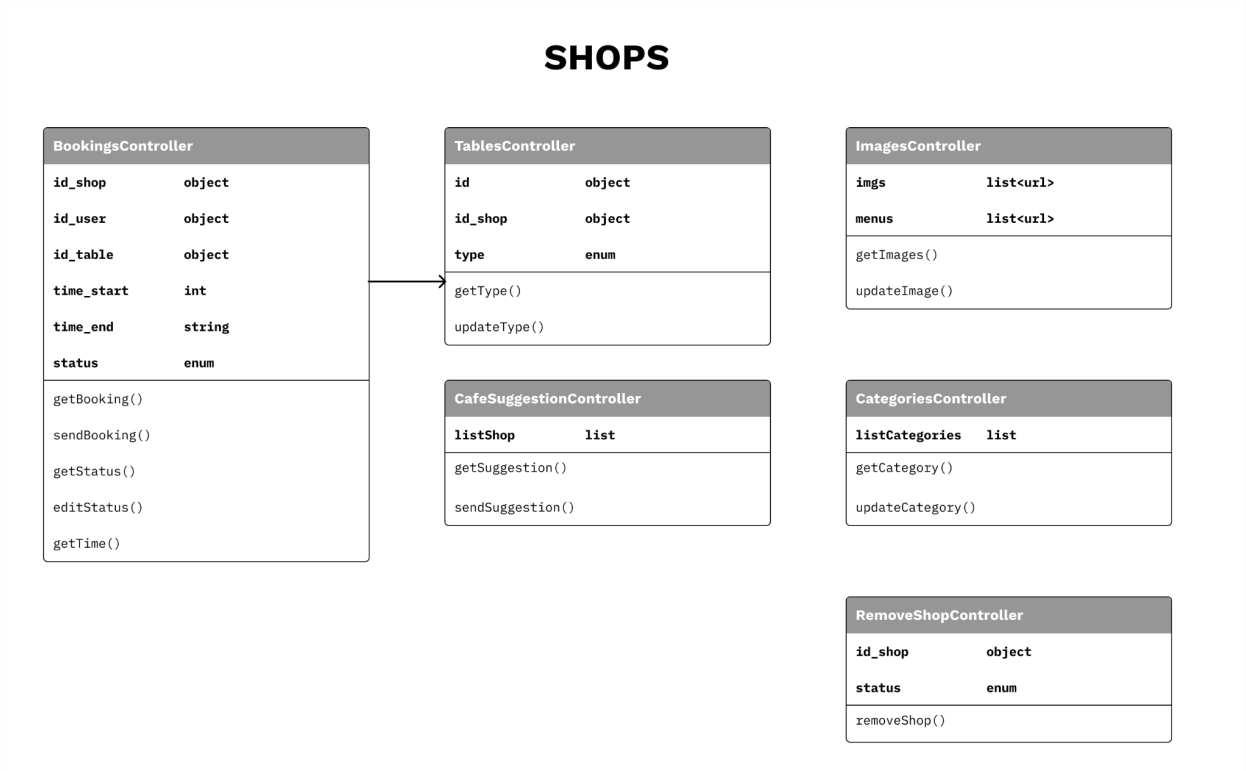
has free parking.
- Flow: The section receives from the database the list of images and renders it..

RecommendShops:
- Function: The recommended shops will be displayed in the Recommend section.
- Flow: The controller will get the information about name, address and rating of the shops then respond to a list of recommendations, then the viewer gets the list then displays it.

FavoriteSongs:
- Function: Managing the list of shops that users have marked as favorites. This helps users easily search for and access their favorite shops
- Flow: Favorite Shops receives information about favorite songs when users perform actions to add or remove shops from the favorites list

4.2 Component: Controller

The "Controller" component in our web application serves as the bridge between the frontend and backend. Developed with ReactJS and MaterialUI, it manages application logic, transforms raw data into a usable format, and ensures the user interface accurately reflects the data from the database. When a user interacts with the interface, the "Controller" processes the input, updates the backend, and synchronizes the frontend to reflect these changes. This component is crucial for maintaining a responsive, dynamic, and user-friendly application.



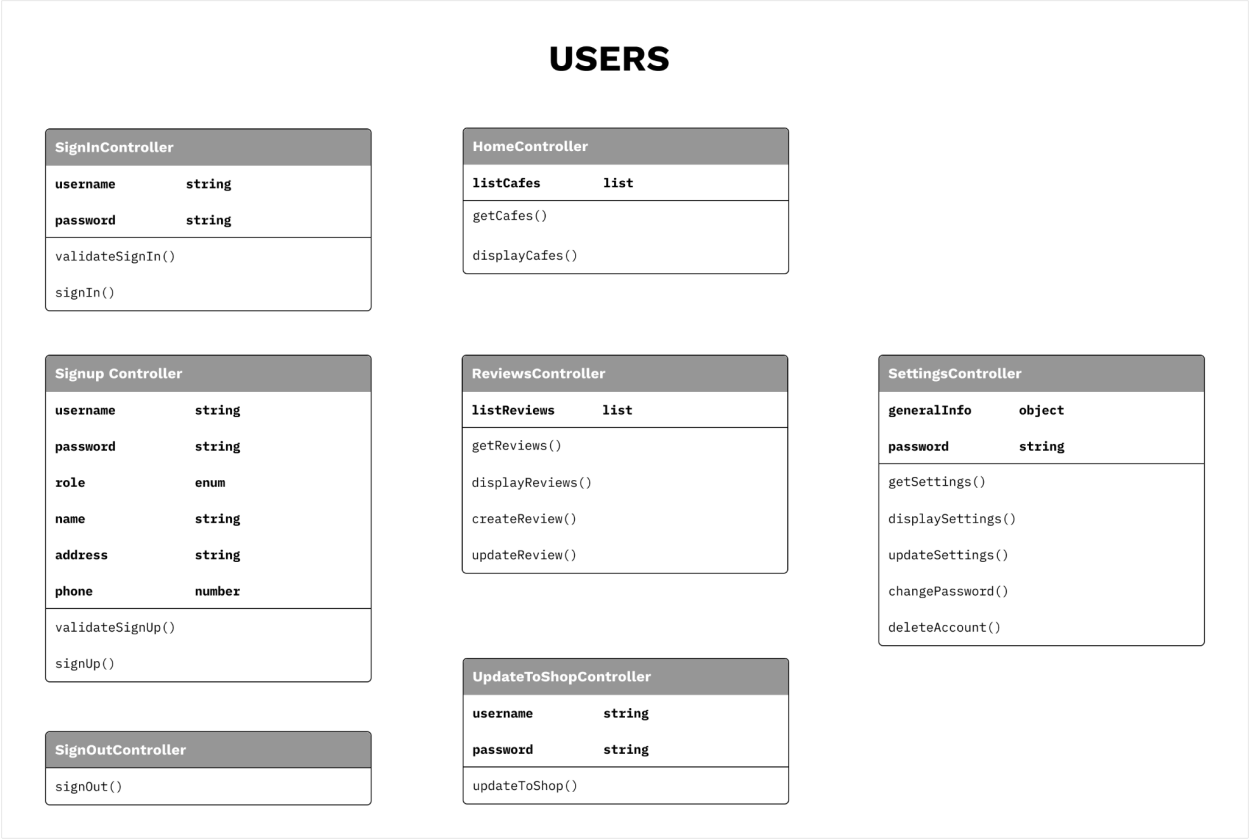
1. **Shops:**
   - **Bookings Controller:** This controller manages customer orders. It oversees the creation of orders, updates order statuses, and ensures synchronization between the frontend and backend when there are changes in the order status or content.
     i. **getBooking():** Retrieves the details of a specific booking using a booking ID as an input parameter.

ii. **sendBooking():** Sends a new booking to the system using the booking details as input parameters. Returns a confirmation upon successful creation of the booking.
iii. **getStatus():** Fetches the current status of a specific booking using a booking ID as an input parameter.
iv. **updateStatus():** Updates the status of a specific booking using a booking ID and the new status as input parameters. Returns a confirmation upon successful update.
v. **getTime():** Fetches the time of a specific booking using a booking ID as an input parameter.

- **Tables controller:** This controller oversees the allocation and status of tables in the cafe. It handles user interactions related to table selection, reservation, and order association. It ensures the frontend accurately reflects the current state of tables and reservations.
    i. **getType():** This method retrieves the type of a specific table. It could take a table ID as a parameter and return the type of the table (e.g., two-seater, four-seater, etc.).
    ii. **updateType():** This method updates the type of a specific table. It could take a table ID and the new type as parameters, update the table type in the system, and return a confirmation once the update is successful.
- **Images controller:** This controller oversees two lists: images and menus, both of which are of the list<url> type. The images list comprises URLs of images showcasing café views and spaces, while the menus list contains URLs of images depicting café menus. The controller's duties include adding, modifying, and deleting images linked to café views, spaces, and menus. It ensures that any modifications to the images are accurately displayed on the frontend, thereby enabling users to browse and interact with the updated views, spaces, and menus.
    i. **getImage():** This function fetches the image of a specific café view, space, or menu. It could accept an ID as a parameter, locate the corresponding URL in the images or menus list, and return the image.
    ii. **updateImage():** This function updates the image of a specific café view, space, or menu. It could accept an ID and the new image URL as parameters, locate the corresponding index in the images or menus list, update the image URL in the system, and return a confirmation once the update is successful. It ensures that the images and menus lists are always in sync. This way, any modifications made to the café views, spaces, or menus are accurately reflected. Please note that this controller does not handle individual menu items. It only manages images related to café views, spaces, and menus as a whole.
- **Café suggestion controller**: This controller provides functionality related to suggesting new cafes to users. It manages the submission of new cafe suggestions, handles validation and storage of these suggestions in the backend, and updates the frontend to inform users about their suggestion status.
    i. **getSuggestion():** This method retrieves a specific cafe suggestion. It could take a cafe ID as a parameter and return the details of the cafe suggestion.
    ii. **sendSuggestion():** This method sends a new cafe suggestion to the system. It could take the suggestion details as parameters and return a confirmation once the suggestion is successfully submitted.
- **Categories controller:** This controller manages the categorization of cafes, reviews, and menu items. It's responsible for creating, updating, and deleting categories. It ensures that cafes and their associated data are properly organized and displayed in the frontend based on their categories.
    i. **getCategory():** This method retrieves the details of a specific category. It could take a category ID as a parameter and return the details of the category.

ii. **updateCategory():** This method updates the details of a specific category. It could take a category ID and the new details as parameters, update the category details in the system, and return a confirmation once the update is successful.

- **Remove shop controller:** This controller changes the status of a shop in our database to a different state, effectively marking it as 'removed'. This allows for potential reinstatement of the shop if needed.
  i. **removeShop():** This method alters the status of a store in our database to a distinct state, effectively labeling it as 'removed'. This provides the possibility for the store to be potentially restored if necessary.



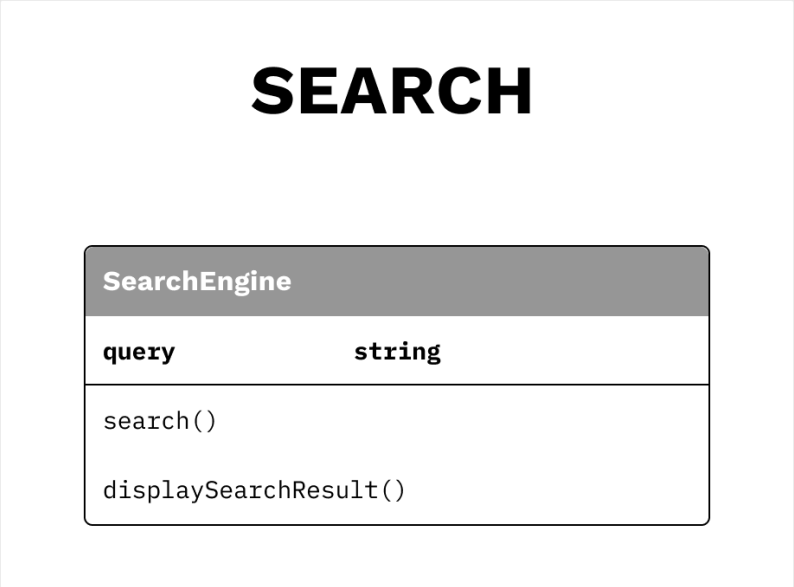
2. **Users:**

- **Sign-in controller:** This controller handles user sign-in operations. It receives user credentials from the view, verifies them with the model, and responds accordingly. On successful verification, it initiates a user session and redirects the user. On failure, it prompts the user to retry.
  i. **validateSignIn():** This method validates user credentials. It receives user credentials from the view and verifies them with the model. It returns a boolean value indicating whether the credentials are valid or not.
  ii. **signIn():** This method initiates a user session. It receives user credentials from the view and verifies them with the model. It returns a boolean value indicating whether the credentials are valid or not.
- **Sign-up controller:** This controller manages user registration. It receives user details from the view, creates a new user record with the model, and responds accordingly. On successful registration, it redirects the user. On failure, it prompts the user to correct their input.

i. **validateSignUp():** This method validates user details. It receives user details from the view and verifies them with the model. It returns a boolean value indicating whether the details are valid or not.
ii. **signUp():** This method creates a new user record. It receives user details from the view and creates a new user record with the model. It returns a boolean value indicating whether the details are valid or not.

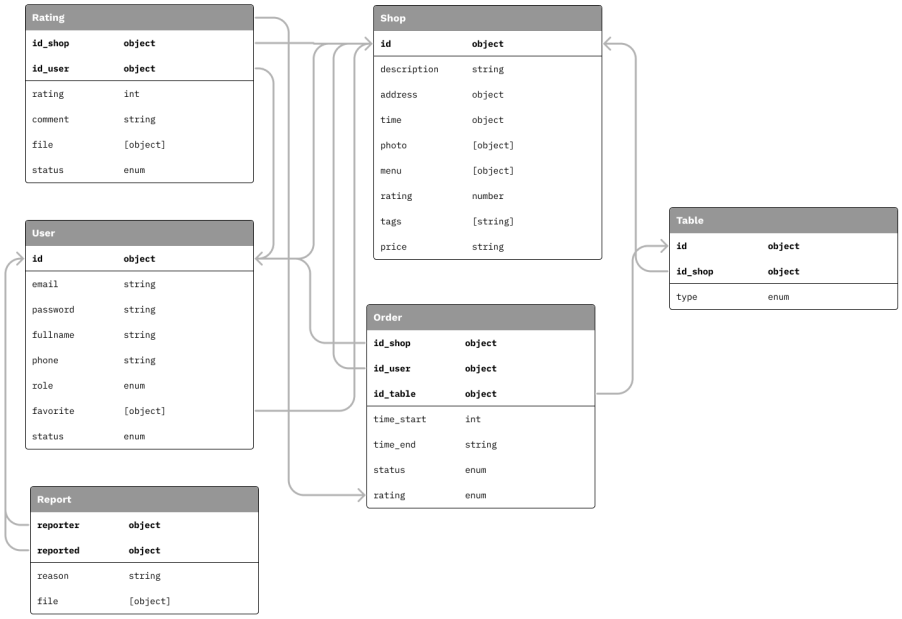
- **Sign-out controller:** This controller handles user sign-out operations. It receives a sign-out request from the view, terminates the user session, and responds accordingly. It redirects the user to the home page.
    i. **signOut():** This method terminates a user session. It receives a sign-out request from the view and terminates the user session. It returns a boolean value indicating whether the operation was successful or not.
- **Update to shop controller:** We would transition users to shop owners by migrating their profile data to the shop database. Thereby unlocking additional functionalities in our application for them.
    i. **updateToShop():** This method alters users into shop proprietors by shifting their profile data to the shop database, thus enabling them with extra features for their shop's administration within our application.
- **Home controller:** This controller manages the home page. It receives a request from the view, retrieves the latest cafes from the model, and responds accordingly. It ensures that the latest cafes are displayed on the home page.
    i. **getCafes():** This method retrieves the latest cafes. It receives a request from the view and retrieves the latest cafes from the model. It returns a list of cafes.
    ii. **displayCafes():** This method displays the latest cafes. It receives a list of cafes from the model and displays them on the home page.
- **Reviews controller**: This controller deals with user-generated reviews for cafes. It's involved in creating, modifying, and retrieving reviews. It ensures that reviews are stored in the backend and updates the frontend to accurately display the latest reviews.
    i. **getReviews():** This method retrieves reviews for a cafe. It receives a request from the view and retrieves reviews for a cafe from the model. It returns a list of reviews.
    ii. **displayReviews():** This method displays reviews for a cafe. It receives a list of reviews from the model and displays them on the cafe page.
    iii. **createReview():** This method creates a new review. It receives a review from the view and creates a new review with the model. It returns a boolean value indicating whether the operation was successful or not.
    iv. **updateReview():** This method updates an existing review. It receives a review from the view and updates an existing review with the model. It returns a boolean value indicating whether the operation was successful or not.
- **Settings controller:** This controller manages user settings. It allows users to customize their experience on the platform by adjusting various settings such as general information, change password, etc. It ensures that these settings are stored in the backend.
    i. **getSettings():** This method retrieves user settings. It receives a request from the view and retrieves user settings from the model. It returns a list of settings.
    ii. **displaySettings():** This method displays user settings. It receives a list of settings from the model and displays them on the settings page.
    iii. **updateSettings():** This method updates user settings. It receives a list of settings from the view and updates user settings with the model. It returns a boolean value indicating whether the operation was successful or not.
    iv. **changePassword():** This method changes user password. It receives a password from the view and updates the user password with the model. It returns a boolean value indicating whether the operation was successful or not.

v. **deleteAccount():** This method deletes a user account. It receives a request from the view and deletes a user account with the model. It returns a boolean value indicating whether the operation was successful or not.



3. **SearchEngine:** This controller is responsible for handling search operations within the platform. It receives search queries from the user, processes these queries to search the backend database, and returns the search results to the frontend. The SearchEngine controller ensures that users can effectively search for cafes, reviews, menu items, and other content within the platform.
    - search(): This method searches the backend database. It receives a search query from the view and searches the backend database with the model. It returns a list of search results.
    - displaySearchResults(): This method displays search results. It receives a list of search results from the model and displays them on the search results page.

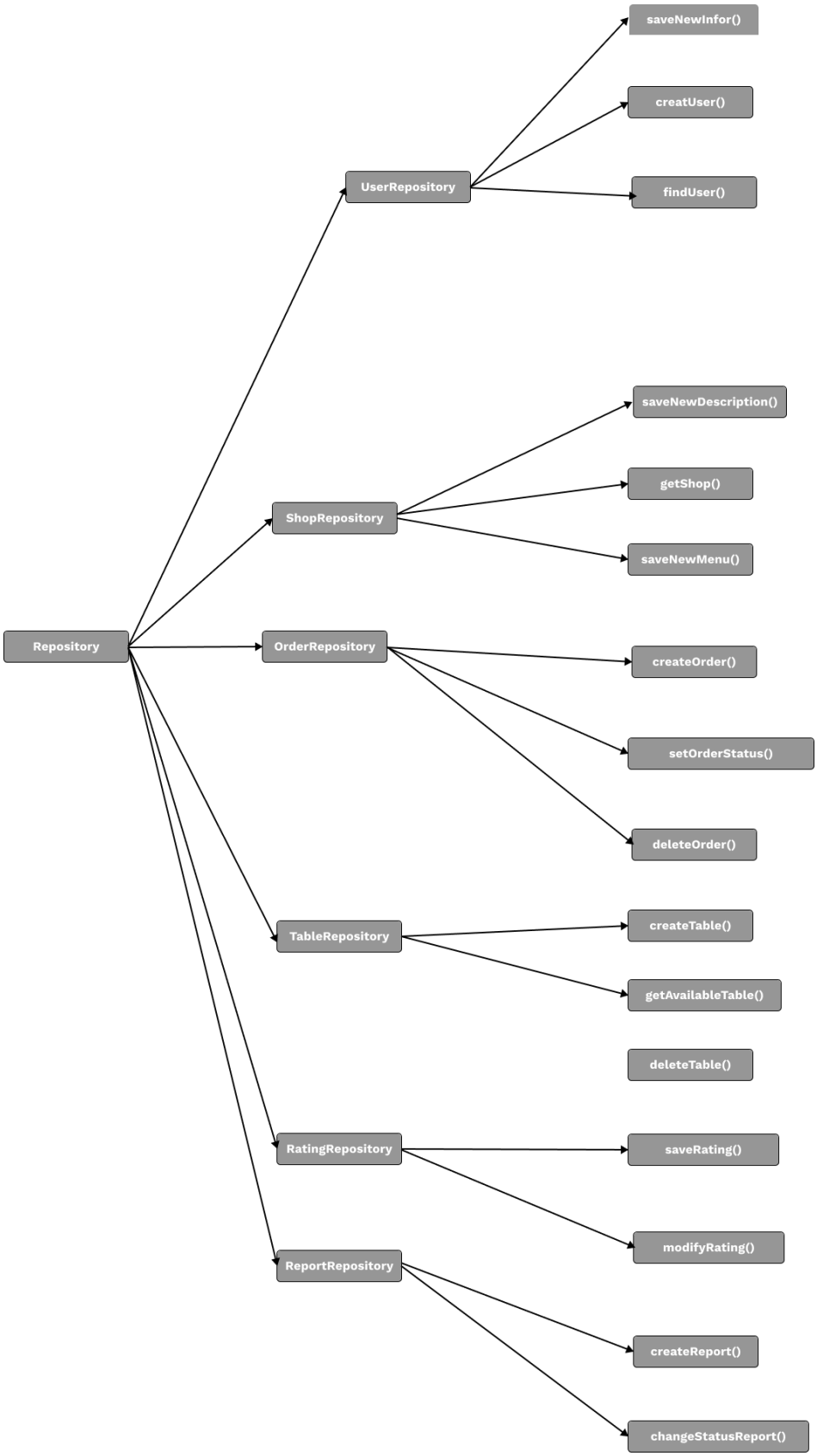
4.3 Component: Model



The "Model" in our web application, developed with JavaScript, Express, and Mongoose, is crucial as it defines the data structure for various entities. Each class in the model corresponds to a specific entity like users or playlists, and its attributes represent the fields in the database tables. This setup allows developers to interact with the database using JavaScript, performing essential operations like creating, reading, updating, and deleting data (CRUD operations).

1. **User model**
   - Manage users' information.
   - Manage authentication.
2. **Order model**
   - Manage information of order.
   - Decide to deny or accept order.
3. **Report model**
   - Save reports made by users.
4. **Rating model**
   - Manage the feedback that users send to shops.
5. **Shop model**
   - Save the shops' information so that users can view brief descriptions.
   - Note that a shop can be a user.
6. **Table model**
   - Manage different kinds of tables in each shop.

4.4 Component: Repository


saveNewInfor()
creatUser()
UserRepository
findUser()
saveNewDescription()
getShop()
ShopRepository
saveNewMenu()
Repository
OrderRepository
createOrder()
setOrderStatus()
deleteOrder()
TableRepository
createTable()
getAvailableTable()
deleteTable()
RatingRepository
saveRating()
modifyRating()
ReportRepository
createReport()
changeStatusReport()

1. **User Repository**
    - **createUser():** Systems will create a new user and save into the database (the password of the user account will be hashed before saved into the database) in service Register..
    - **findUser()**: Service sends the users' email, and the system will query the user having that email to check the login information.
    - **saveNewInfo()**: systems will handle to save new information that the user changes (phone number, address, …).
2. **Shop Repository**
    - **saveNewDescription()**: System will save the new description that shops make.
    - **getShop()**: While users are typing shops' names, the system will get the characters and query all the results containing those characters in finding shop service.
    - **saveNewMenu()**: System will query the menu of those shops to change by new menu.
3. **Order Repository**
    - **createOrder()**: System will save the order made by users into a database so that shops can get the information of orders.
    - **setOrderStatus()**: Shops can use service to change status orders which are accepted or denied.
    - **deleteOrder()**: Users can cancel their order before it is checked by shops, the orders are deleted in the database.
4. **Table Repository**
    - **createTable():** This function will receive information from the service before adding a table into the database with id of shop and table data.
    - **getAvailableTable():** When this function is called, an algorithm will be executed to find whether any table available for booking on the database or not based on the id of the shop.
    - **deleteTable():** When a shop wants to delete a table in the database due to reasons, this function will help them to do that. After this function is called, the system will delete the chosen table of that shop immediately from the database using id of shop and id of table.
5. **Rating Repository**
    - **saveRating():** System will save comment of user, rating star and recalculate the rating of shop based on the new rating using the star and comment given by a user and id of shop.
    - **modifyRating():** System will allow users to modify, re-comment or even delete a comment of a rating given by the user. However, it requires a user to make an order before rating.
6. **Report Repository**
    - **createReport():** This function allows both users and shops to report to each other if they do not satisfy one or more rules in community standards.
    - **changeStatusReport():** This function allows an administrator to resolve reports.

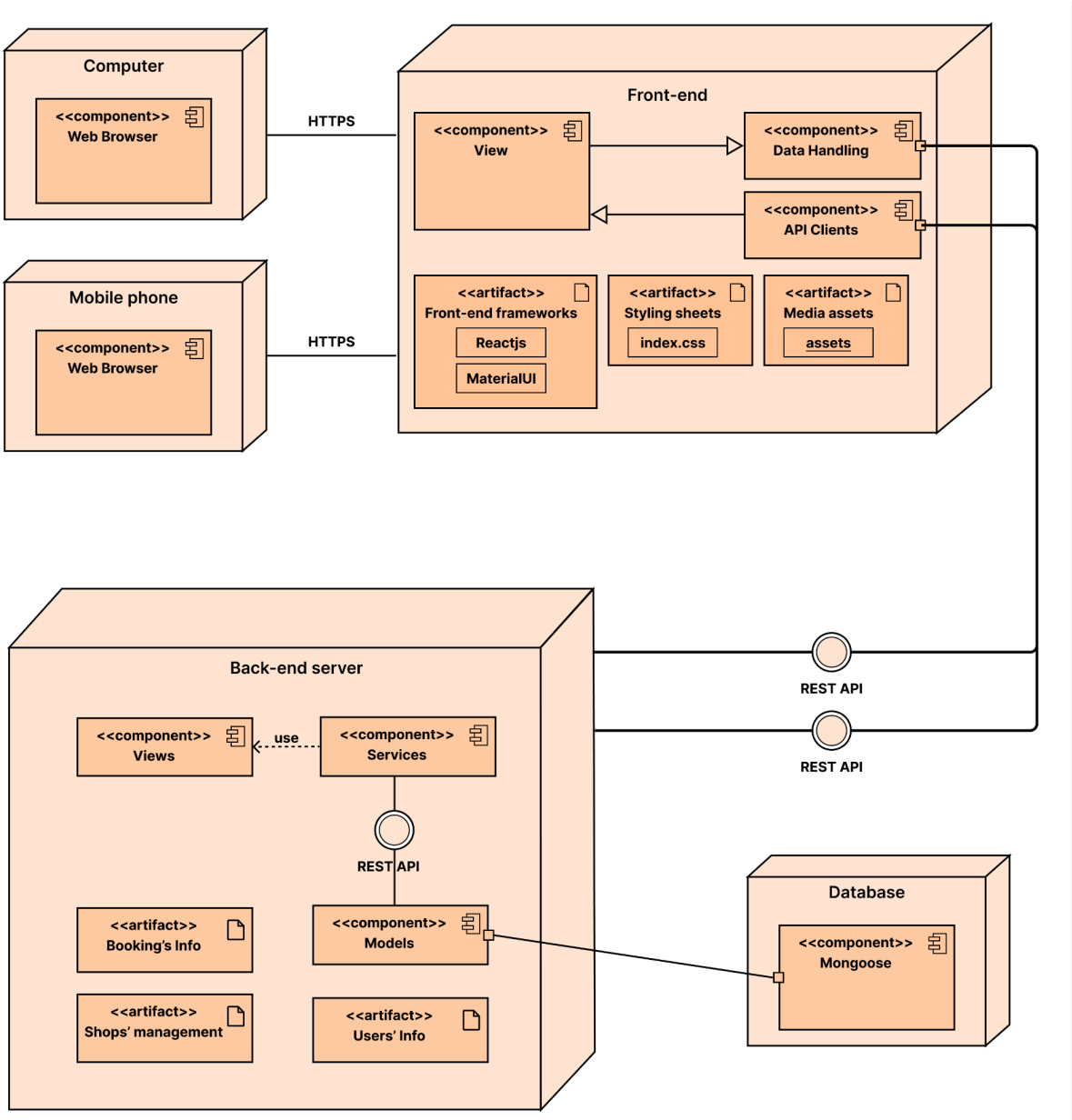
4.5 Component: Service

1. **User Registration and Login Service**
    - Responsible for processing user registration data.
    - Takes user inputs (email, password, role, general information,...) and stored hashed password into database.
    - Verify when users login by email, password and role, and then return an access token. This token will be valid for an hour.
    - Includes changing password and deleting account.
2. **Coffee Shop Service**
    - Manages information about coffee shops, such as location, open hours, and menu.
    - Includes adding new coffee shops, updating coffee shop information, and deleting coffee shops.
    - Handles searching for coffee shops based on their name, location, and filtering by ratings, and tags.
    - Providing users detailed information about coffee shops.
3. **Booking Service**
    - Responsible for managing users' reservations.
    - Stored booking information into the database, such as users' information, table's type and time.

- Allow coffee shops that have the table booked to check information and confirm or cancel the reservation. It will update users' booking status and notify them.
- Present for users their booking status, that is if it is confirmed or not.

4. **Rating Service**
   - Allow users to write reviews about their experience after their reservation. Each review includes rating, comment and upload image. These reviews will be stored in a database.
   - Support users to update or even delete their rating.
   - Provides users the list of shops that they have reviewed and allows shops to view list ratings from users.
   - Calculate average ratings for shops.
5. **Report Service**
   - Allow users to report shops that have low quality services after their reservation to admin.
   - Takes users input about why they report shop and store it in the database.
   - Based on information users provided, administrators can resolve the report.
6. **Shops' analytics Service**
   - Visualize the number of reservations, reviews (monthly, weekly, etc.), and average rating from database to shops' dashboard page.
   - Allow shops to filter data by various criteria.
   - Support export data for further analysis.

5. Deployment



The deployment diagram illustrates the distribution of components and artifacts across various node in the system. In this system, we have main nodes:

5.1. Computer & Mobile phone:

These nodes represent the devices on which users can access and interact with the application. They have the Web Browser component, allowing users to view and use the application through a web browser. Every interaction on the web browser will send HTTPS requests to the Front-end node to get the view and other data.

5.2 Front-end node:

This node includes 3 components View, Data handling and API Clients:

- View: responsible for rendering UI components of the application. This component interacts with the API Clients component to receive data and send actions to update.
- API Clients: this component interacts with the backend server by sending requests to retrieve the data as well as perform other actions.
- Data handling: This component receives data from the backend server and processes data to the correct form or needed form for the frontend to use and render.
  This node also contains some artifacts:
- Frontend frameworks:
    - Reactjs: an open-source JavaScript library and frontend framework that is used to create user interfaces in a declarative and efficient way. It is a component-based front-end library that is responsible only for the view layer of a Model View Controller (MVC) architecture. The application uses this framework to build UI components.
    - MaterialUI: a customizable and themable React component library that implements Google's Material Design. This framework has its own components with pre-built UI and logic. This framework also provides a system design for the application.
    - Styling files: the most general styling files are stored in '/src/libs/ui…' to determine the most overall style for the entire application.
    - Media assets: all images, icons, banners are stored in '/src/assets/…' and always in place to serve the users.

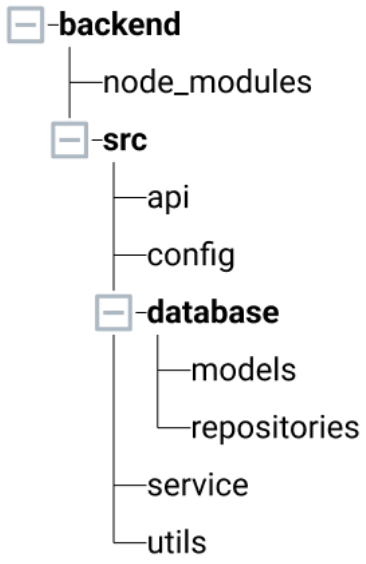
5.3 Back-end node:

It consists of several components:

- Views: These handle the interaction between the front end and the back end, processing user requests and providing appropriate responses. They work in conjunction with the Services component to ensure smooth communication.
- Services: providing services for handling data to render on the front-end. They are used to communicate with the Views component through REST API.
- Models: These define the data structures and manage the interaction with the Database node. In this case, the database is Mongoose, a MongoDB object modeling tool designed to work in an asynchronous environment.
- Artifacts used by the Backend Node:
    - Users' info, Booking's info, and Shop's management: These are resources used in the application that are stored and managed by the backend.

6. Implementation View

Supplement



backend:
- package.json
- .env

utils:
- ErrorHandle.js
- ResponedData.js

models:
- index.js
- OrderRoutes.js
- RatingRoutes.js
- ReportRoutes.js
- ShopRoutes.js
- TableRoutes.js
- UserRoutes.js

models:
- index.js
- Order.js
- Rating.js
- Report.js
- Shop.js
- Table.js
- User.js

service:
- OrderService.js
- RatingService.js
- ReportService.js
- ShopService.js
- TableService.js
- UserService.js

config:
- index.js

src:
- Connection.js
- index.js

src:
- ExpressApp.js
- index.js

repositories:
- OrderRepository.js
- RatingRepository.js
- ReportRepository.js
- ShopRepository.js
- TableRepository.js
- UserRepository.js