Group 20

Booking Hotel
Software Architecture Document

Version <1.1>

Revision History

Date	Version	Description	Author
30/11/2022	1.0	3-tier architecture, component database, component services integration	Nguyen Long Vu
30/11/2022	1.0	Component searching, component user management, component payment	Le Cong Huu
14/12/2022	1.1	Implementation View	Nguyen Long Vu
14/12/2022	1.1	Update Section 1 & 2	Ngo Thanh Luc

Table of Contents

1. Introduction 4
   - 1.1. Purpose 4
   - 1.2. Scope 4
   - 1.3. Definitions, Acronyms and Abbreviations 4

2. Architectural Goals and Constraints 4

3. Use-Case Model 5

4. Logical View 6
   - 4.1. Component: Searching 6
   - 4.2. Component: User Management 8
   - 4.3. Component: Payment 8
   - 4.4. Component: Authentication 9
   - 4.5. Component: User Action 11
   - 4.6. Component: Hotel management 13
   - 4.7. Component: Database 14
   - 4.8. Component: Service Integration 15

5. Deployment 16

6. Implementation View 17

Software Architecture Document

1. Introduction

1.1. Purpose

- The document provides an architectural overview of the system using several different architectural views to illustrate several aspects of the system. It captures and conveys the important architectural decisions made on the system

1.2. Scope

- This document is an architectural overview of the Booking Hotel website. Providing online booking services, advertising for hotels, and reservation management, this website is hosted online.
- This document describes the current architectural structure of our website. It is intended to be read by future developers who will join the project and create and interface their own website with this project.

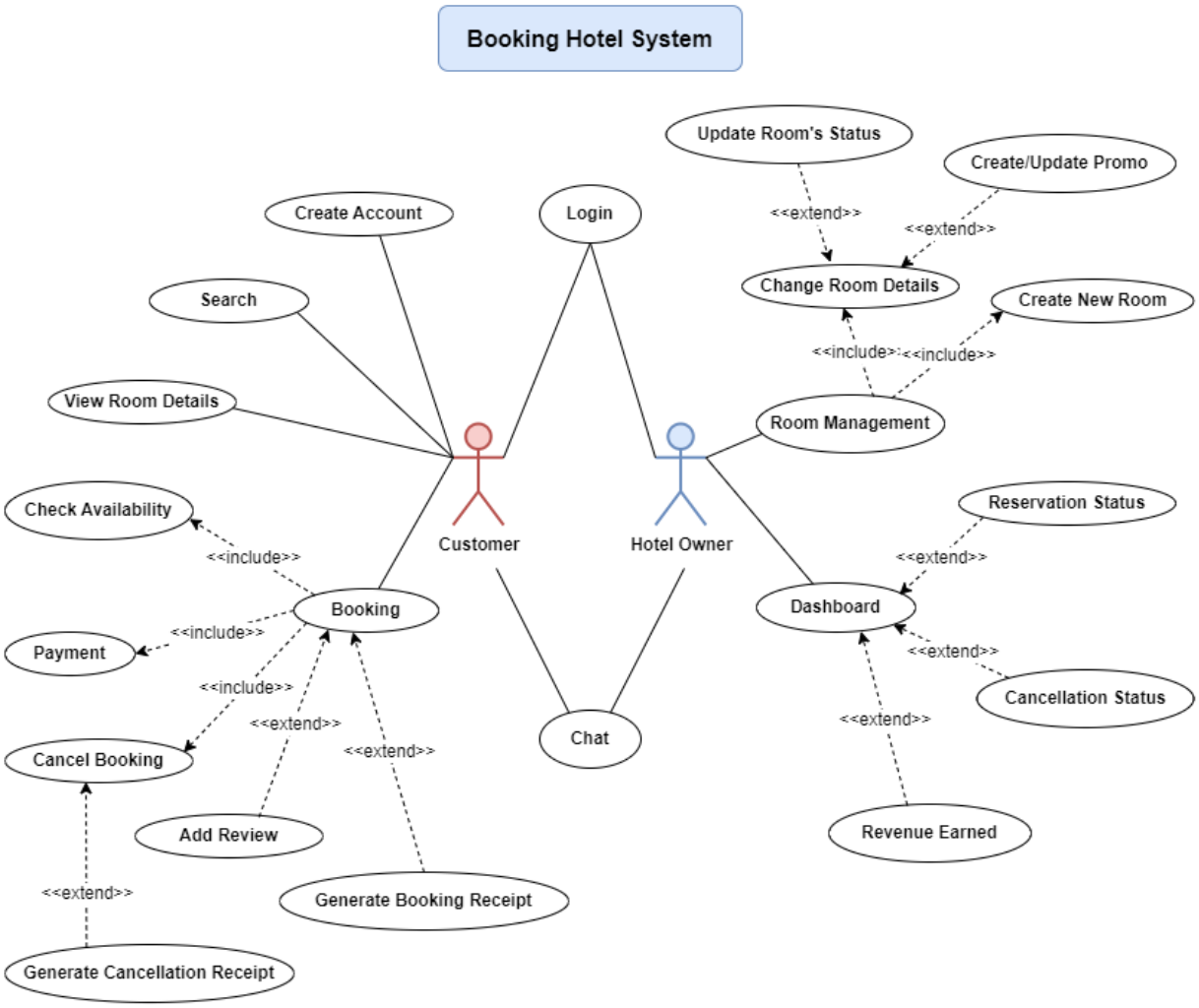
1.3. Definitions, Acronyms and Abbreviations

- In this document, we use the following abbreviations and acronyms:
  + **PCI DSS:** Payment Card Industry Data Security Standard a widely accepted set of policies and procedures intended to optimize the security of credit, debit and cash card transactions and protect cardholders against misuse of their personal information
  + **MTTRS:** Mean time to respond is the average time it takes to recover from a product or system failure from the time when you are first alerted to that failure. This does not include any lag time in your alert system.
  + **APIs:** API stands for Application Programming Interface. In the context of APIs, the word Application refers to any software with a distinct function. Interface can be thought of as a contract of service between two applications. This contract defines how the two communicate with each other using requests and responses.
  + **GUI:** A graphical user interface (GUI) is an interface through which a user interacts with electronic devices such as computers and smartphones through the use of icons, menus and other visual indicators or representations (graphics).

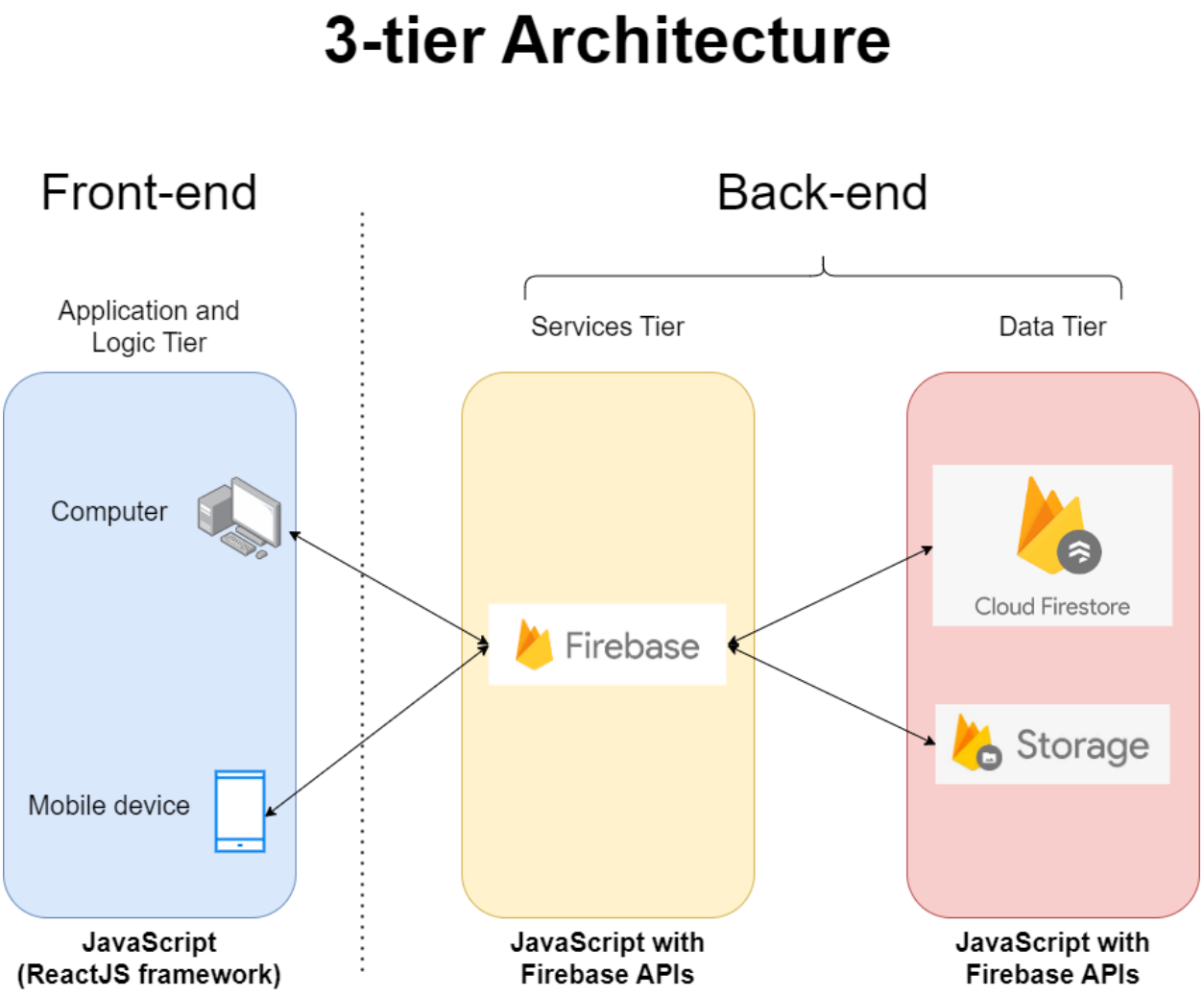
2. Architectural Goals and Constraints

- The purpose of this section is to describe the software requirements and objectives that have a significant impact on the architecture and to identify any special constraints that may apply.
  - **Safety:** PCI DSS compliance is required for payment gateways
  - **Security**: The system must be secure, so that a customer can make online payments.
    - The application must implement basic security behaviors:
      - **Authentication:** Login using at least a username and a password
    - For internet access, the following requirements are mandatory
      - **Confidentiality:** sensitive data must be encrypted (credit card payments, password)
      - **Data integrity:** Data sent across the network cannot be modified by a tier
  - **Privacy:** information about users cannot be leaked without their permission.
  - **Maintenance:** Mean Time to Respond (MTTRS) after system failure should not be greater than 10 minutes. MTTRS includes all corrective maintenance and deferred work.
  - **Performance:** The web should reply to the clients' commands as soon as possible
  - **Availability:** The web must be available all the time (24/7) except when there are maintenance activities.
  - **Capacity:** Not only does the system support one million concurrent visits, but it also maintains optimal performance.

3. Use-Case Model



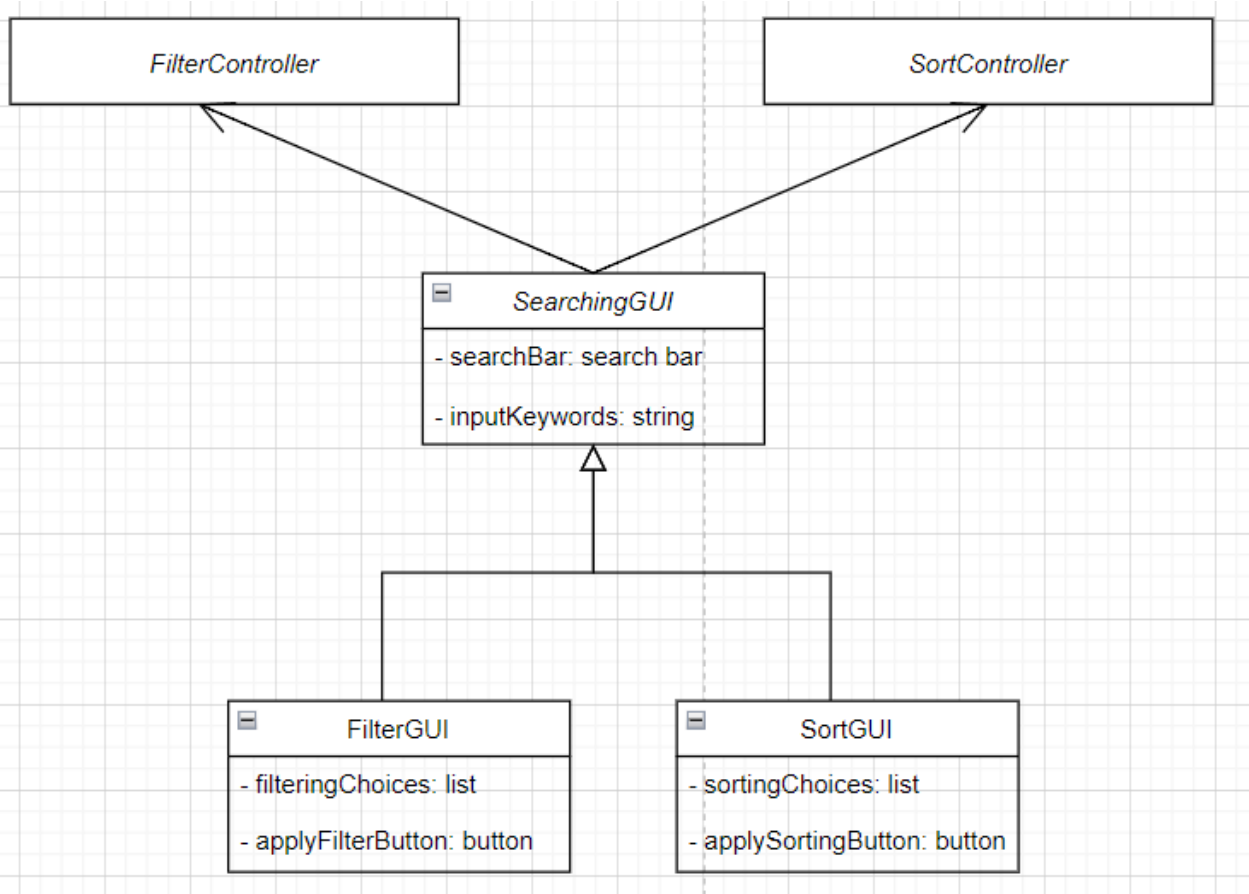
4. Logical View



4.1. Component: Searching

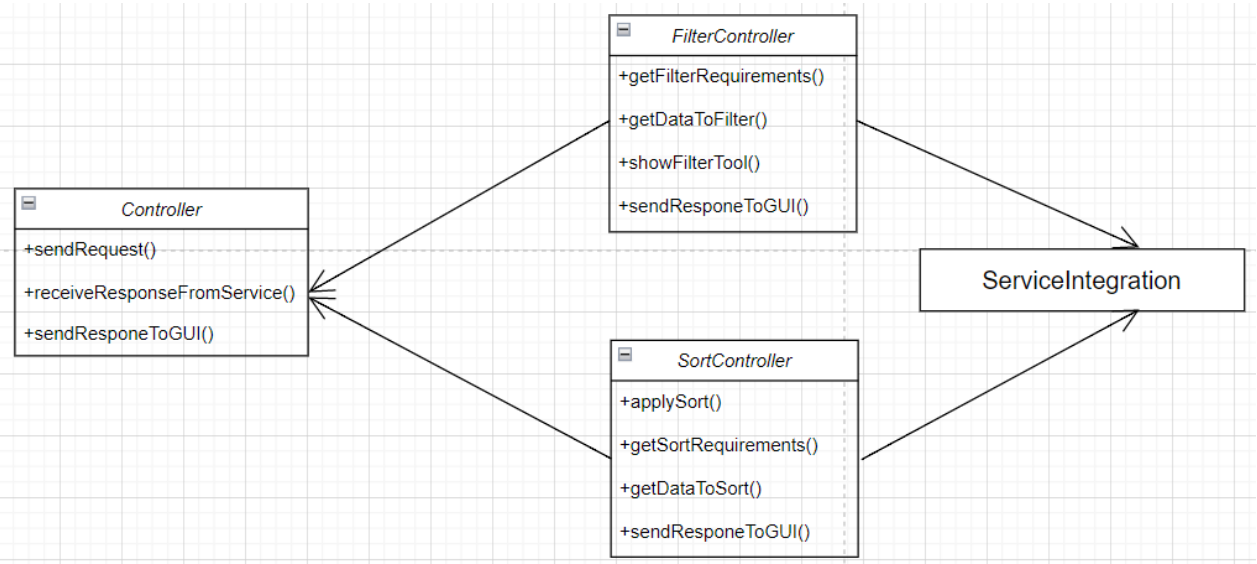
- **GUI:**

+ In Presentation tier, there is a search bar for the user to enter the keywords. The user can search: hotel names or hotel address. There will be a filter for input requirements below before searching for hotel names and hotel address. After the user receives the data from the server, their can sort or filter for the list of results. All will be processed by SearchingController.



- **Controller:**

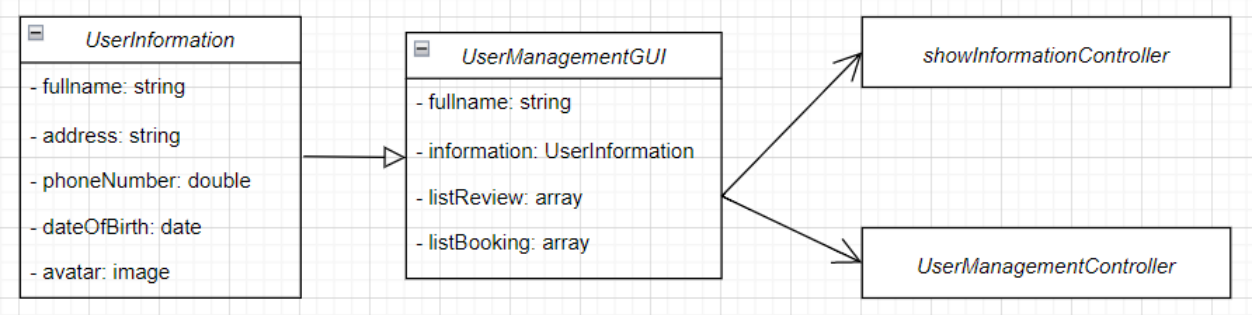
+ In the FilterController and SortController classes, these access the input data in the search bar and chosen criteria in tool (filter or sort). After the data will be sent to Service Integrations to handle. Final accessing the Database to get data and display on user's screen.



4.2. Component: User Management

- **GUI:**

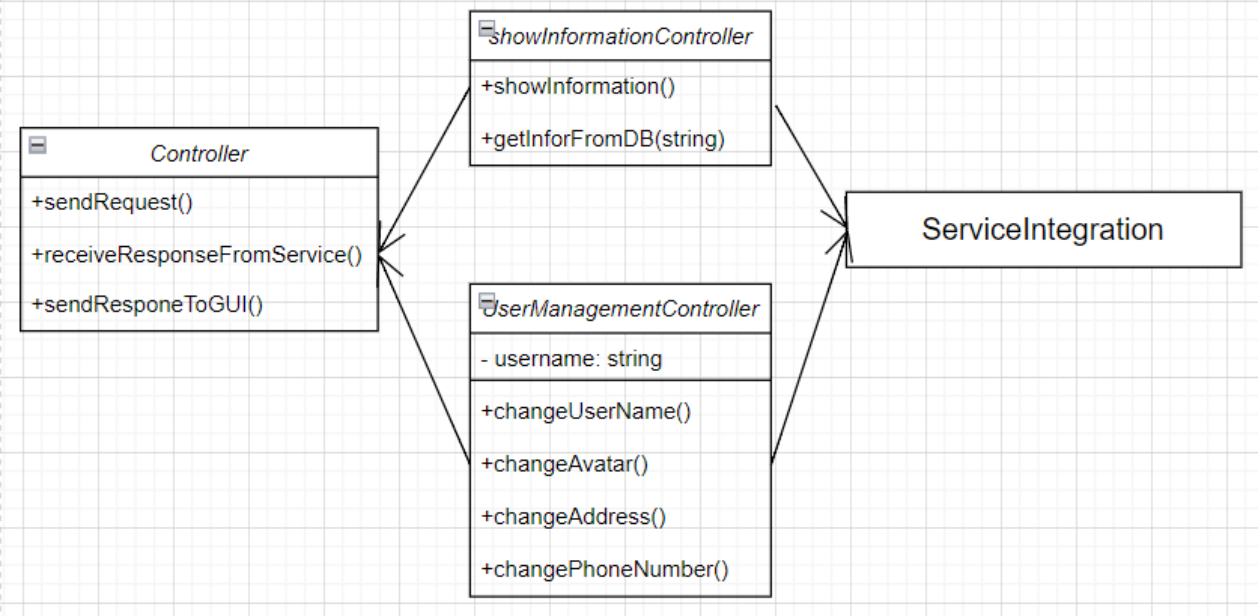
+ In Presentation tier, the system will show all information of the user includes their username, detailed information, avatar, cover and view their review/booking. ShowInformationController will handle the information in user's account page. If there is edited, UserMangementController will process the information. The information will be updated and show on screen after refresh.



- **Controller:**

+ In the showInformationController class, it will get the user's information based on username through the Service and give it to the private variable of this class. After accessing the database to get data and display the user information on screen.

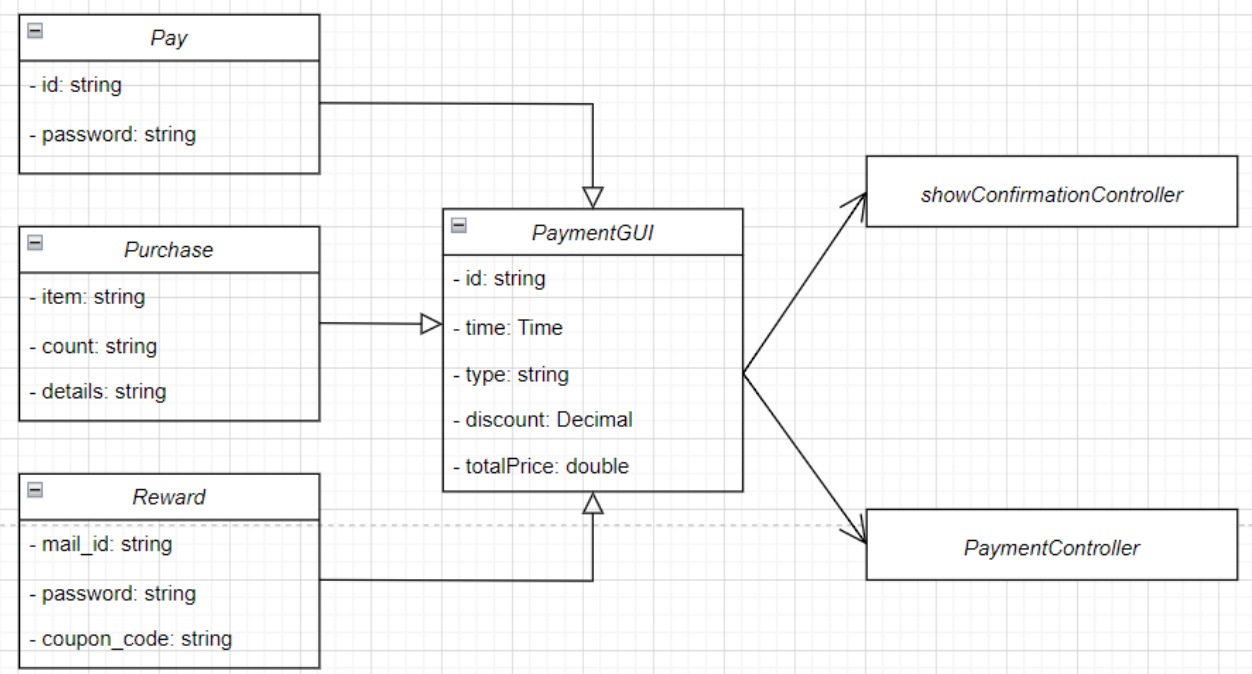
+ In the userManagementController class, the user can change information through the functions. These the function ask the Service to access the Database to update database for user's account.



4.3. Component: Payment

- **GUI:**

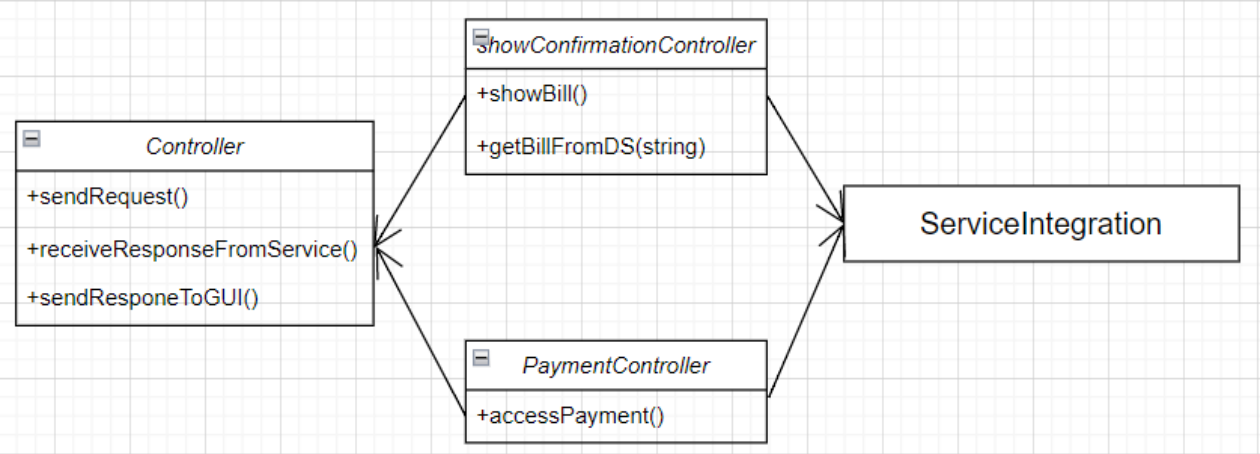
+ In Presentation tier, when the user makes a payment, the user first supplies details of the hotel that need to be registered, then provides payment details and the enters delivery detail. Then the user will confirmation their bill. If the user pay successfully, the system will display on user's screen. All will be processed by PaymentController.



- **Controller;**

+ In the showConfirmationController class, it will get the bill based on id through the Service and give it to the private variable of this class. After accessing the database to get data and display the user bill on screen.

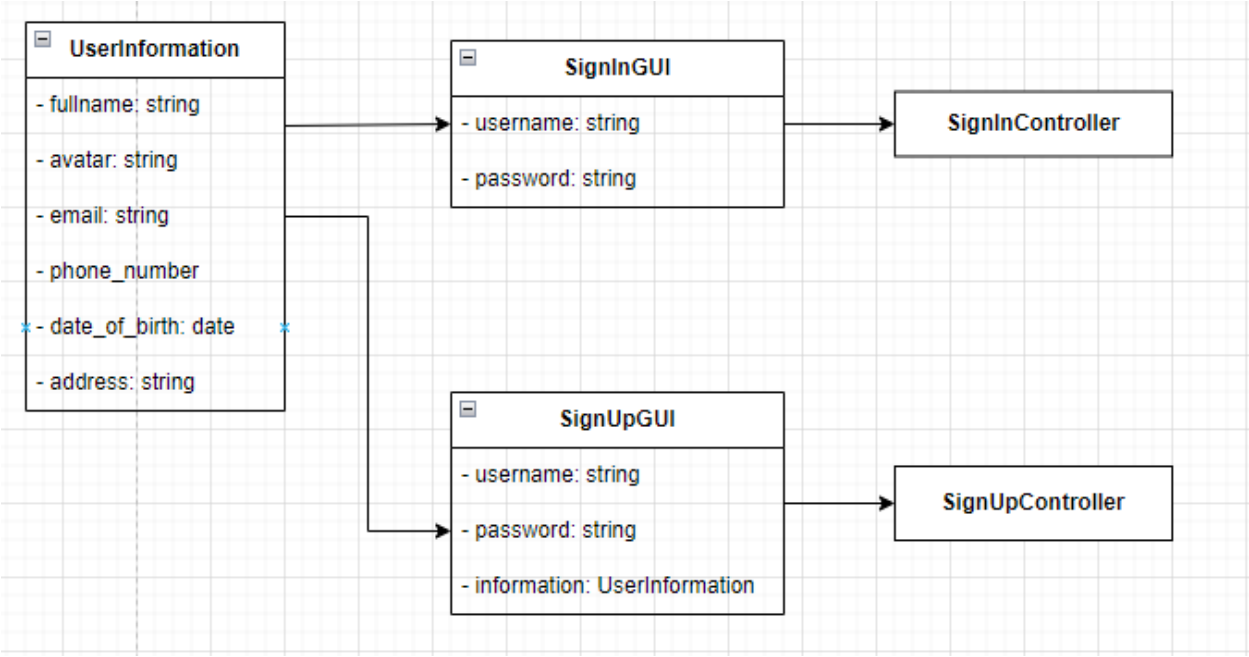
+ In the PaymentController class, the user payment through the functions. If paying successfully, these the function will ask the Service to access the Database to update database for user's account.



4.4. Component: Authentication

- **GUI:**

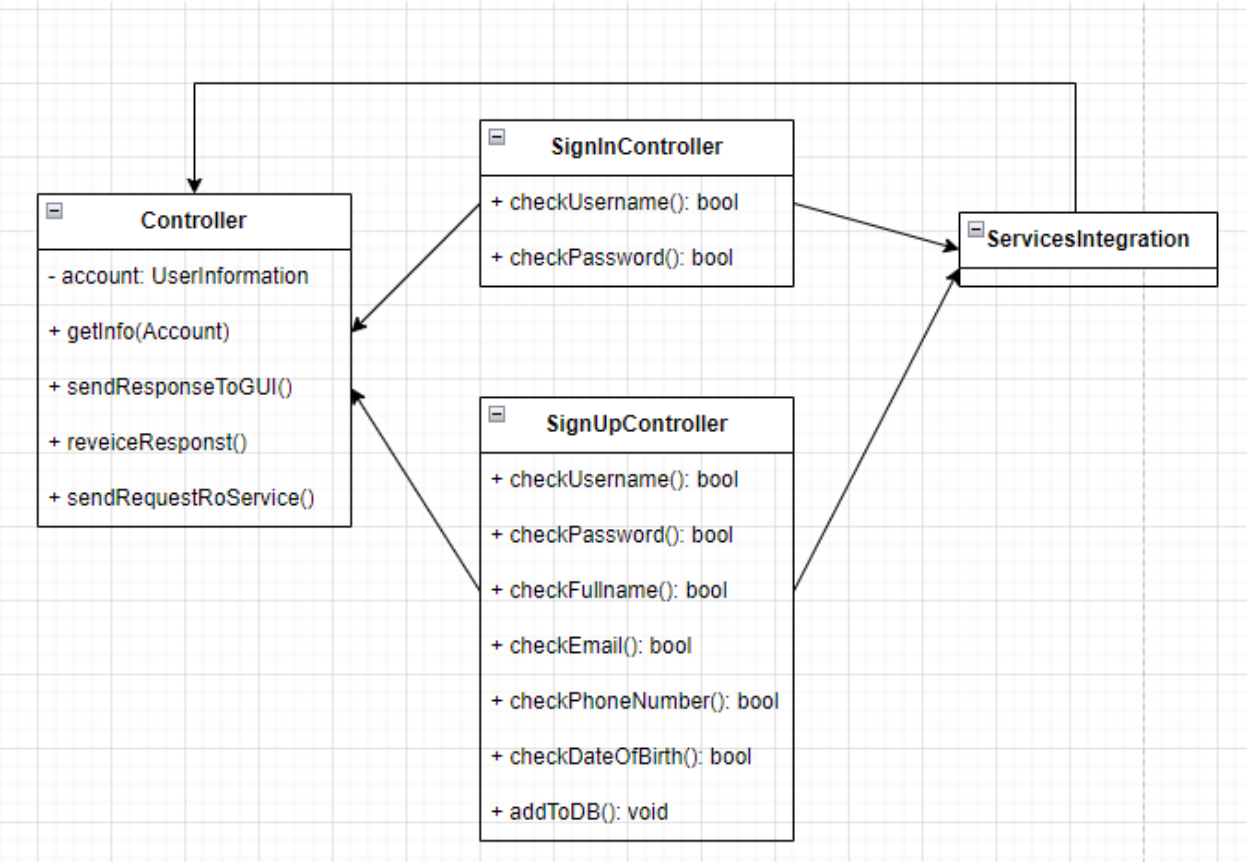
+ In the Presentation tier, the user will see the authentication form. There are two forms: Sign In and Sign Up. Initially, the user is directed to Sign In page. They are required to enter the username and password in text bars, then click the "Sign In" button to submit. If they don't have an account, the user will be moved to Sign Up page by clicking to "Don't have any account?". After signing up successfully, they will be redirected to Home page.



- **Controller:**

+ For Authentication Controller, we build a father class called Controller. This class will receive requests from the GUI and get username and password entered by the user through function getInfo(). This class also receives responses from deeper processing components (or more detailed - Services Integration) and sends these responses to GUI to display on the user interface.

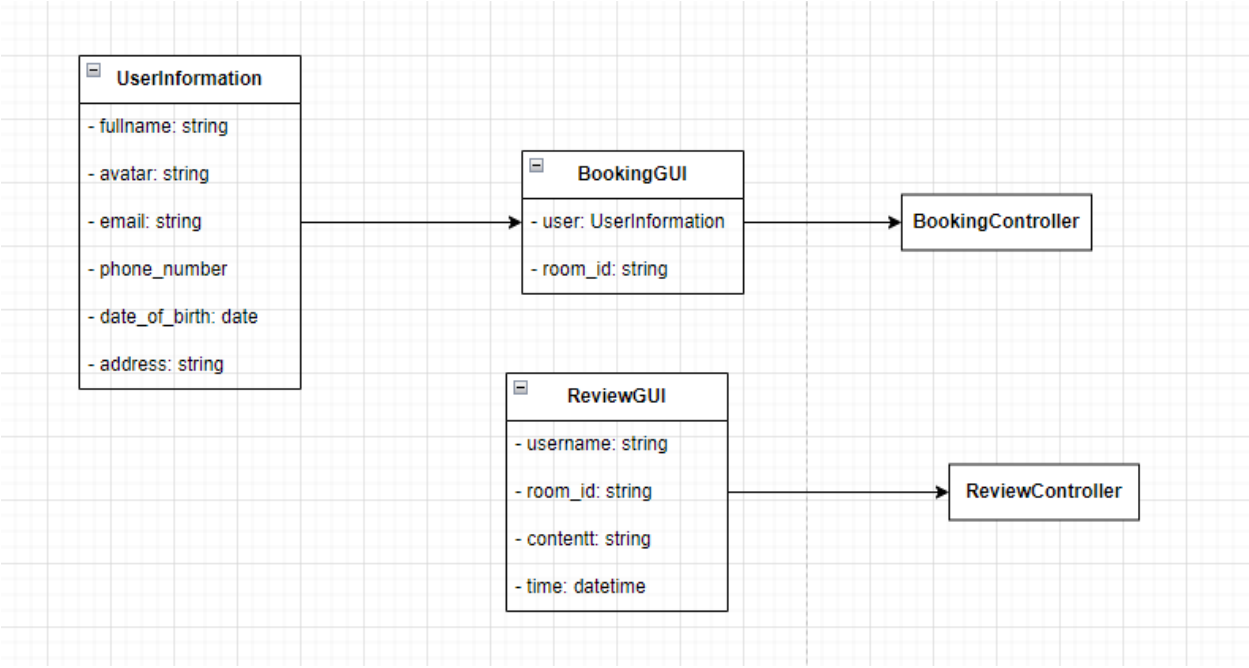
+ SignInController class - child class of Controller, used for checking account and password that the user enters through calling the Service so that the Service can access the Database to get all the accounts with the same username and check the password. Services Integration will send the response to the Controller whether sign in is successful or not, then the Controller will send that response to the GUI.



4.5. Component: User Action

- **GUI:**

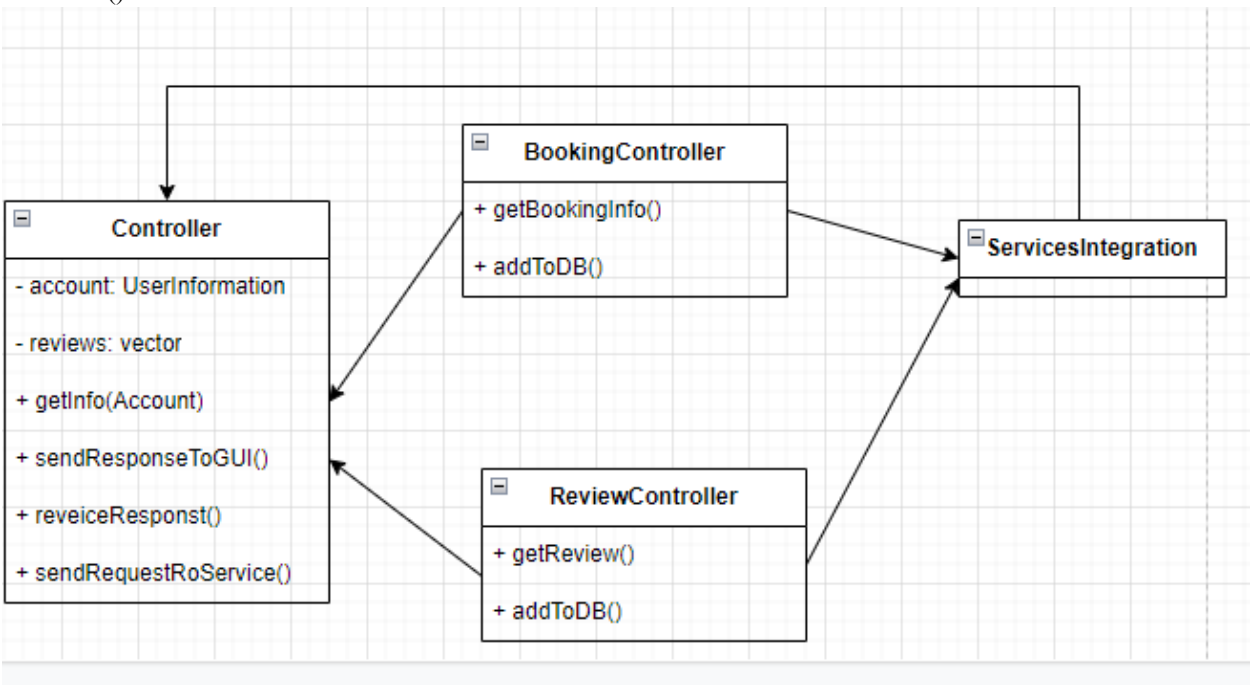
+ In Presentation tier, the user is allowed to see a list of suggested hotel cards. Each card will contain its name, images, address. If users login successfully, they can see details of it by clicking the “See detail” button. Here they will see the number of empty rooms, price of each room and can book a room if available. After booking a hotel, they can review their experience to anyone in the comment area of the hotel.



- **Controller:**

+ With the BookingController class, we get the content of the rental into the Controller through the function getBookingInfo(). Then pass it to the Services Integration to add to the Database by using the function addToDB().

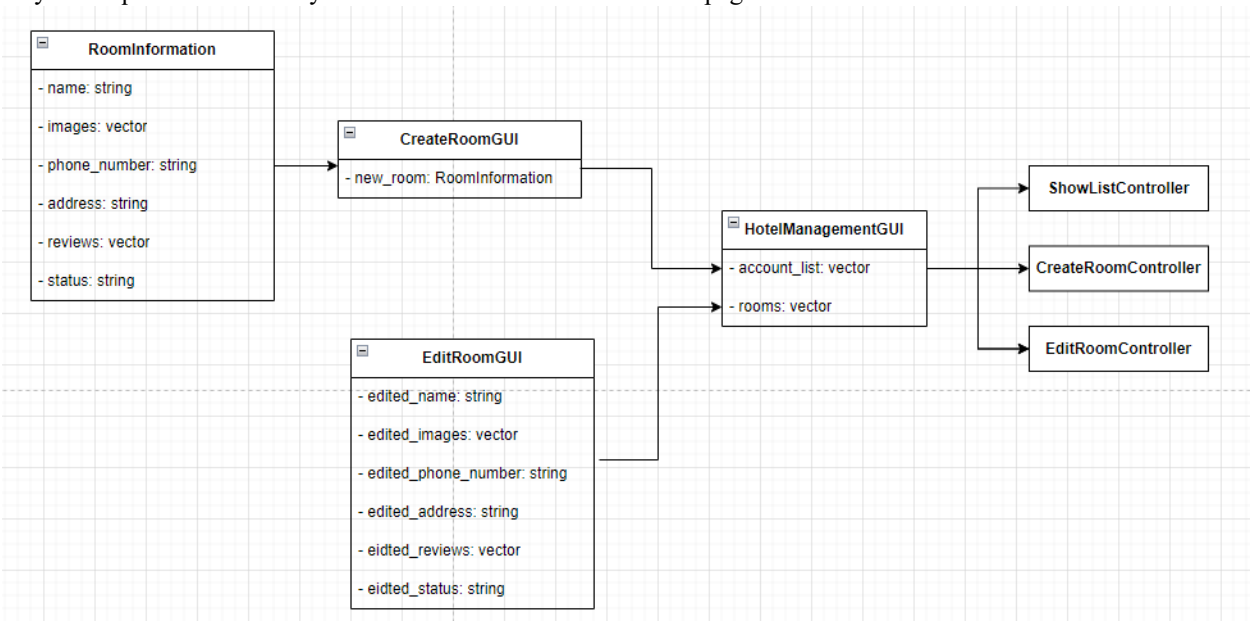
+ With the ReviewController class, we get the content of the review entered by the user into the Controller through the function getReview(). Then pass it to the Services Integration to add to the Database by using the function addToDB().



4.6. Component: Hotel management

- **GUI:**

+ In Presentation tier, the hotel owners can see list of their user accounts and their hotel information. They can create a new room or change the room details. If they want to create a new room, they are required to enter the information of that room in Create room page then click “Create” button. If they want to change the room details, they are required to edit many fields of room details in Edit room page.

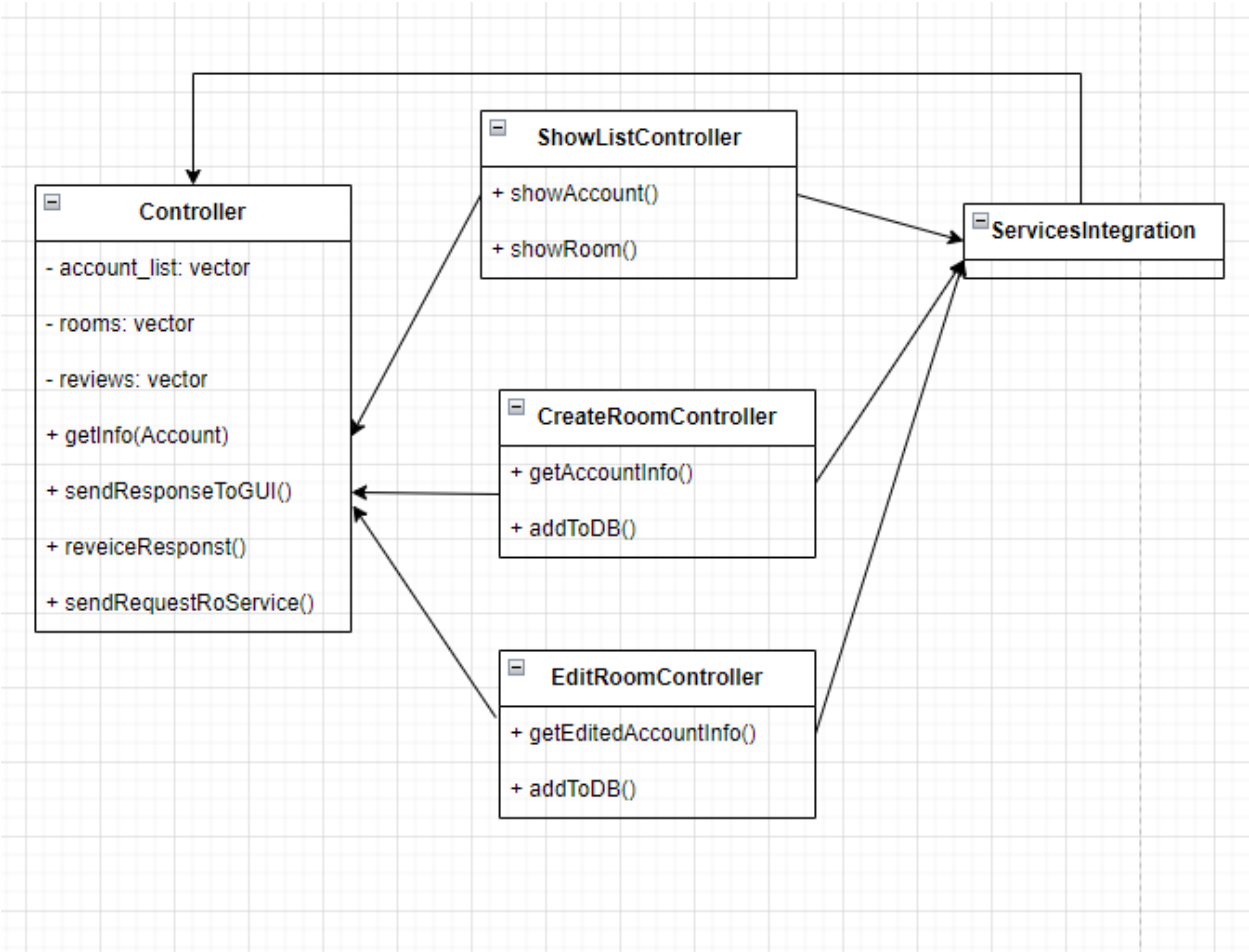


- **Controller:**

+ In the ShowListController class, we call the function to display the list of rooms and hotels. The above functions will call Service Integration to access the Database to update data into the private variables in the Controller (parent class) as well as give the results (send response) to the GUI to display to the admin.

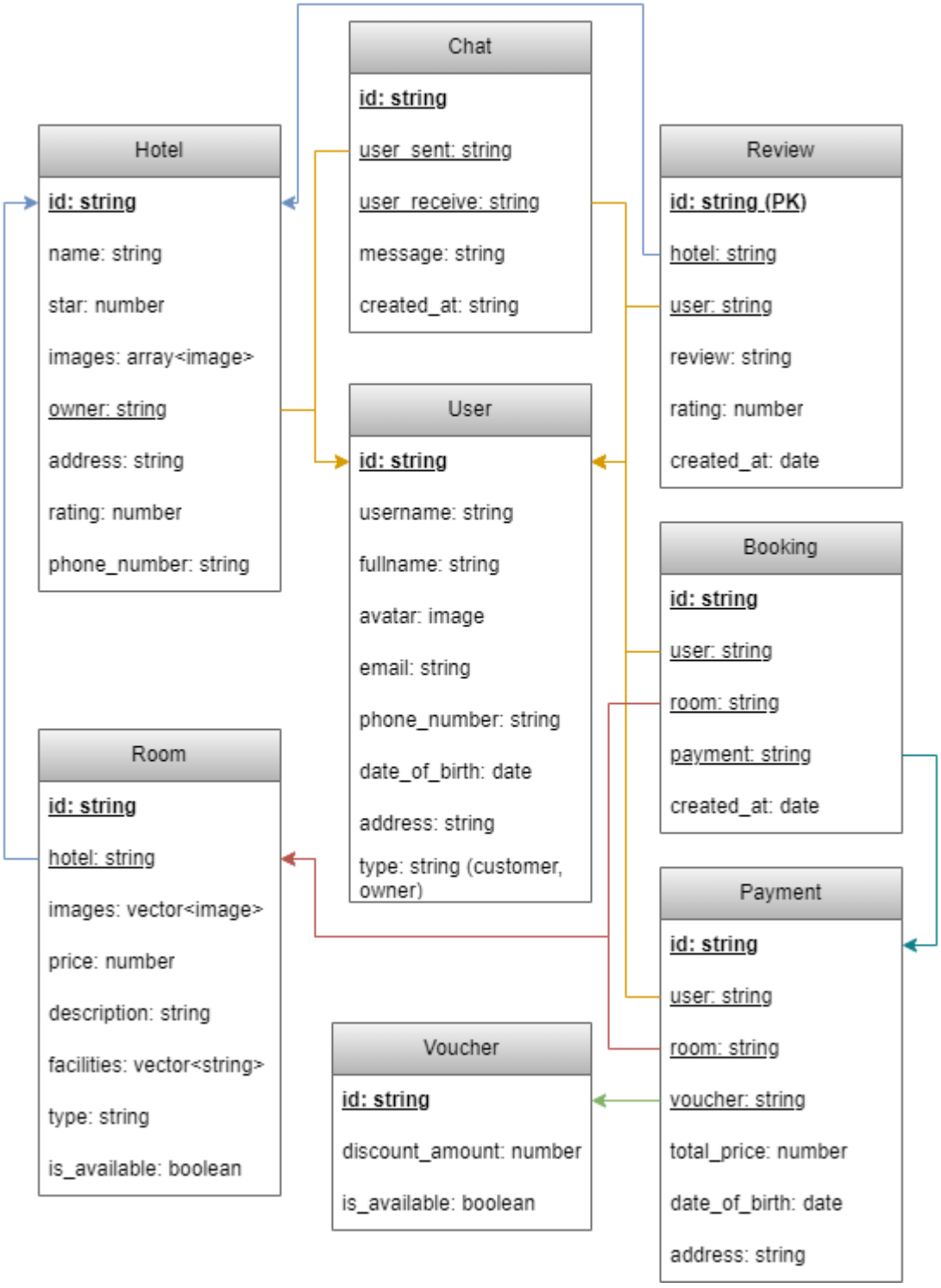
+ In the CreateRoomController class, we call the function to get information about the new room then create a new room in the database.

+ In the EditRoomController class, we call the function to get the edited information and update them in the database.



4.7. Component: Database

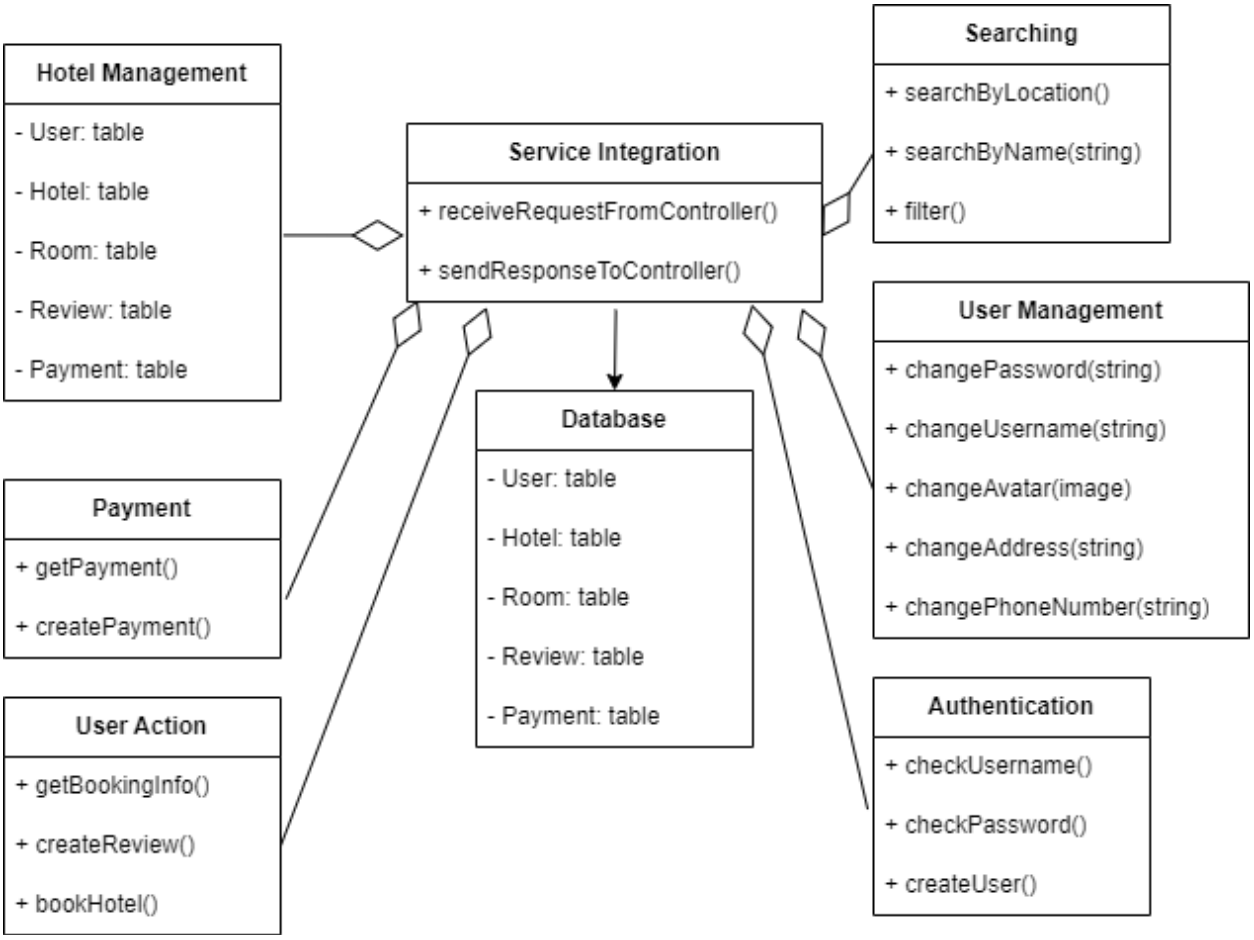
- The database of our website is sketched in the class diagram below. It shows how each table is constructed and how they connect to each other.



4.8. Component: Service Integration

- The Service Integration is constructed to communicate between client and server side. From the client, the

Integration will receive the commands from the Controllers, send them to server (by using GET and POST methods) to request the server to access the database. Then, the accessed data will be sent back to the client side to display on the GUIs.

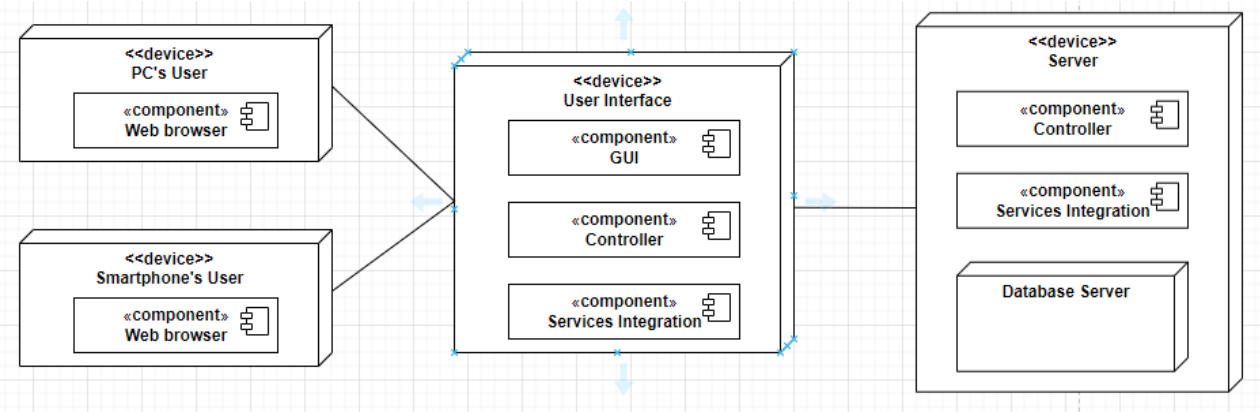


5. Deployment

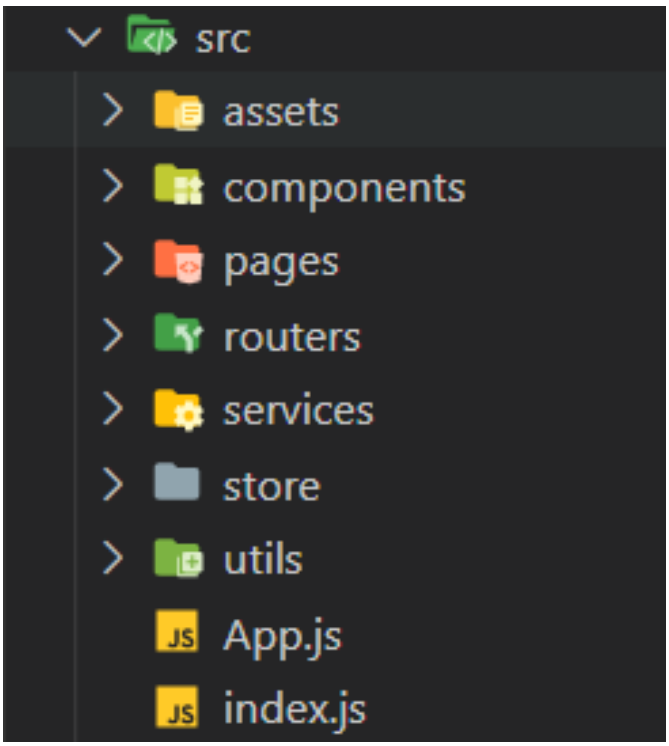
Our team will deploy the architecture on 2 different computers as the picture shown below.

- Users use their device (it can be PC or Smartphone) to access our website through the web browser. Web browser will access the Front End (or in the deployment, this is the User Interface).
- At this time, the computer which takes care of the interface for the user while using this website (User Interface) receives a signal from the user's device. It will take the data from the Server and show them to the screen for the user
- We use 1 Server for this architecture, so we deploy 1 computer to operate this Server
- By HTTP connection, we connect the User Interface with Server. This Server will get data from Database Server.

By this deployment, if 1 computer in the architecture faces an error, we only need to fix that computer without affecting other computers.



6. Implementation View



- Assets: Raw files we need, like images, text, json,…
- Components: Common components can re-usable.
- Pages: Pages component, each page component contain multiple sub components.
- Routers: Routing files.
- Services: Config and service files.
- Store: Config global state.
- Utils: Utilities files.