Group 01

KOOLB
Software Architecture Document

Version 1.0

Revision History

Date	Version	Description	Author
29/11/22	1.0	Update section 1, 2, 3	Tường Vy
30/11/22	1.1	Update section 1, 3	Ngọc Linh
1/12/22	1.2	Update section 4	Tường Vy
2/12/22	1.3	Update section 4	Gia Khánh
16/12/22	1.4	Update section 5, 6	Ngọc Linh
16/12/22	1.5	Update section 5, 6	Gia Khánh

Table of Contents

Software Architecture Document

1. Introduction

a. Purpose

With the use of several alternative architectural perspectives, this document illustrates each component of the system's architecture in detail. It aims to capture and convey the key system architecture decisions that have been made.

b. Scope

An architectural overview of the KoolB App System is provided in this software architecture document. KoolB is being developed to support online accommodation booking.

c. Reference

1. Tutorial Video
2. Software Architecture Document template
3. Vision Document.docx

d. Document Content Overview

This document has used several architectural views such as use case, logical view, deployment, and implementation view to compute size, performance and quality considerations.

2. Architectural Goals and Constraints

There are some key requirements and system constraints that have a significant bearing on the architecture. They are:

2. The app's environment will be IOS, Android and web
3. The app must be reliable and trustable in case of security by not leaking or transferring users' private information
4. Data must be completely protected from unauthorized access by the app. All remote accesses are subject to password protection and user identification.
5. Constantly up to date for compatibility with devices
6. The system should be easy to use and install for non-tech relevant users.
7. UI design is user-friendly, it is easy to learn and use
8. When developing the architecture, all performance and non-functional criteria as outlined in the Vision Document [3] must be taken into account.
9. The application shall be able to be deployed on a wide variety of software/hardware systems

3. Use-Case Model

KOOLB all use-cases are:

- **Register:**

  This use case describes how a new user registers into the KOOLB.
  The actors starting this use case are Host and Renter.

- **Log in:**

  The KoolB App login process is described in this use case.

  Host, Renter, and Admin are the actors who begin this use case.

- **Setting:**

  This use-case describes the setting process of the app KOOLB.
  This use-case is performed by host and renter.
  - **Update profile:**

    This use-case describes the update profile process of the app KOOLB.

    This use-case is performed by host and renter.
- **Notification:**

  This use case describes the notifications acting in Renter and Host.

  Renter and Host are the actors who begin this use-case.
- **Chatbox:**

  This use case allows a renter and a host to discuss more information about the room.

  The Renter and Host are the actors who begin this use case.
- **Booking:**

  This use case describes how a user books a room in the system..

  The actor starting use case is Renter.
- **Cancel booking:**

  This use case describes how a user cancels their bookings in the system..

  The actor starting use case is Renter.
- **Check Availability:**

  This use case describes the availability checking.

  Renters are the actors who begin this use case.
- **Reservation List View:**

  This use-case describes how the user views their reservations.

  The actor begins this use-case as Renter.
- **Checkin:**

  The KoolB App check-in process is described in this use case.

  Renters are the actors who begin this use case.
- **Checkout:**

  This use-case describes the checkout process of the app KOOLB.

  This use-case is performed by renter.
- **Rating:**

  This use-case describes the rating of a specific accommodation.

  The action of this use-case is performed by user type "Renter".
- **Search room:**

The use case allows a renter to search rooms with their preferences which includes destination, time and the number of people staying.

The actor starting this use case is the Renter.

- **Sort and Filter:**

  This use-case describes the process of filtering and sorting in the search bar in app KOOLB.

  This use-case is performed by Renter.

- **Favorite List:**

  This use case shows a list of saved accommodations.

  The actor starting use case is Renter.

- **Post room Information:**

  The KoolB App posting accommodation process is described in this use case.

  Host are the actors who begin this use case.

- **Customer Service:**

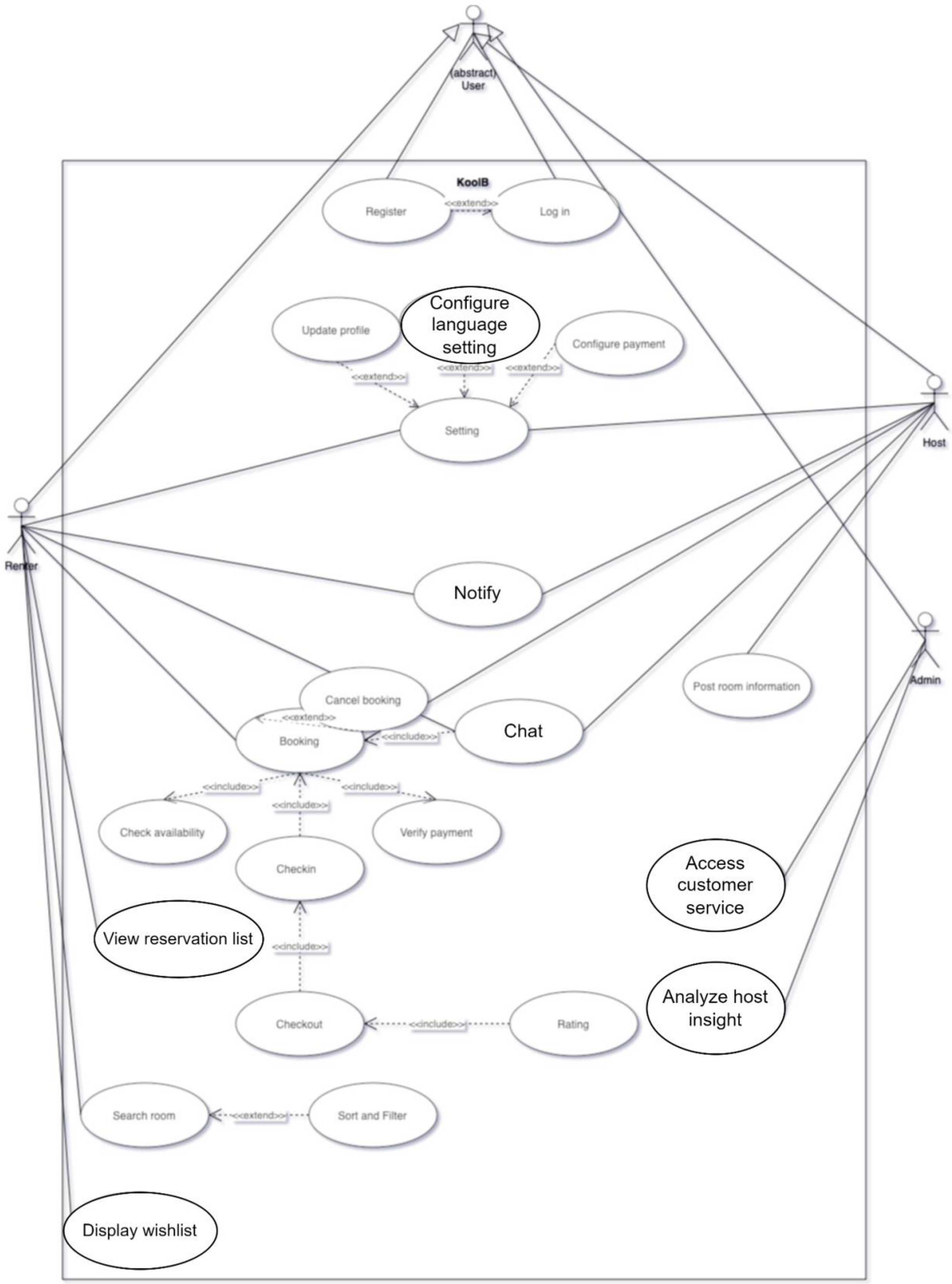
  The KoolB App Customer service process is described in this use case.

  Admin are the actors who begin this use case.

- **Host Insight:**

  This use-case describes the tools for hosts to manage information and data about their service.

  The action of this use-case is started by the user “Host”.

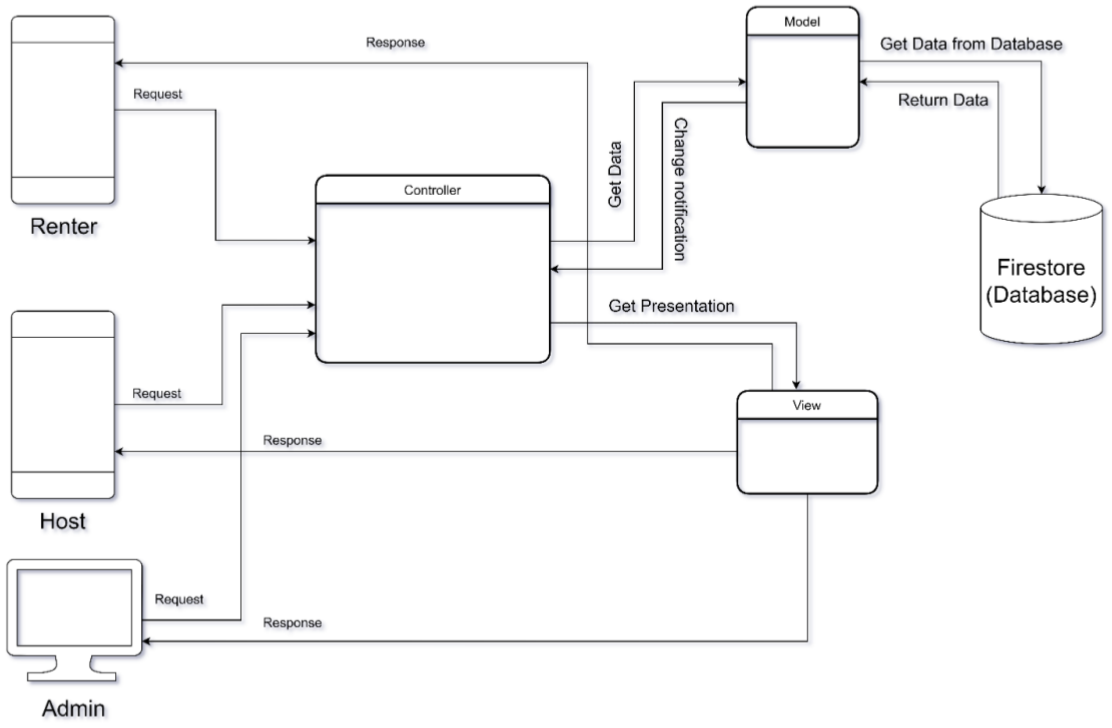

(abstract) User
KoolB
Register
<<extend>>
Log in
Update profile
Configure language setting
Configure payment
<<extend>>
<<extend>>
<<extend>>
Setting
Host
Renter
Notify
Admin
Cancel booking
<<extend>>
Booking
<<include>>
Chat
Post room information
<<include>>
<<include>>
<<include>>
Check availability
Verify payment
Checkin
Access customer service
View reservation list
<<include>>
Analyze host insight
Checkout
<<include>>
Rating
Search room
<<extend>>
Sort and Filter
Display wishlist

4. Logical View

4.1 Overview

This section describes the main application module, how it will interact with other components and how they implement the specification and profile.

4.2 Architectural Structure

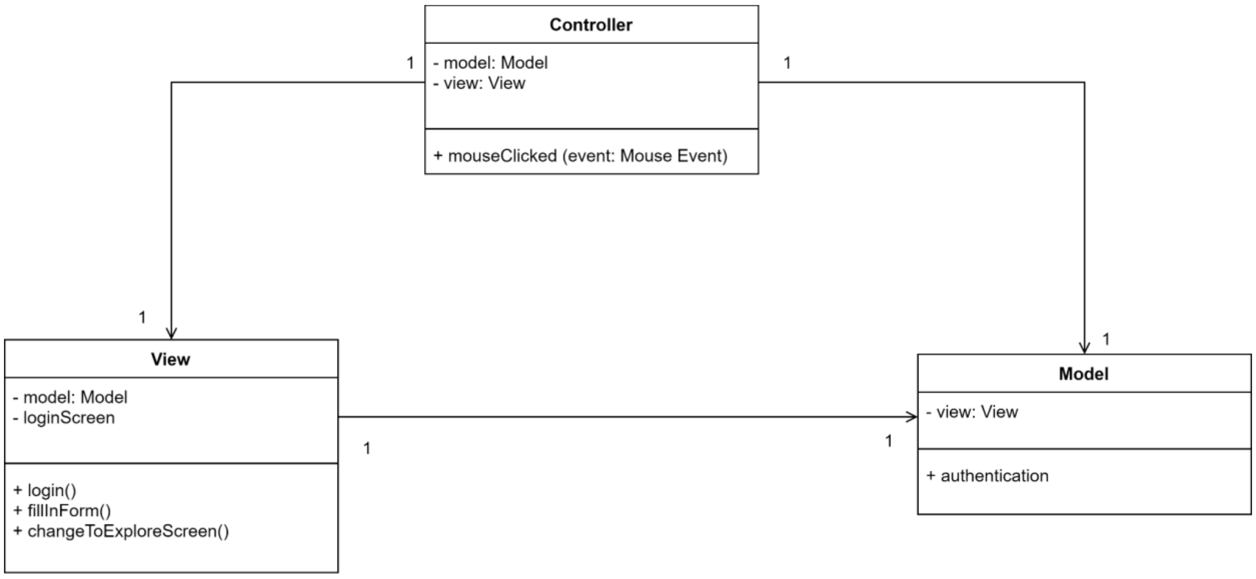


- The overall software is structured as a Client–server architecture. Since our app is a multi-user software, there will be a variety of queries coming from clients that need access to the server through the internet, which is the database that includes data about each user and all related information. Thus, by applying the Client-server architecture, the server can be distributed across a network

  Since there are multiple ways to view and interact with data in the app (such as when a user views a list of accommodation in a wishlist and in explore page, even though the information is the same, the way it is showed to users is different), the app itself is structured as an MVC (Model - View - Controller) architecture. The controller will be responsible for all user communications through user-interfaces. Which means that the controller will pass any queries, users' gestures, transactions, users' requirements to

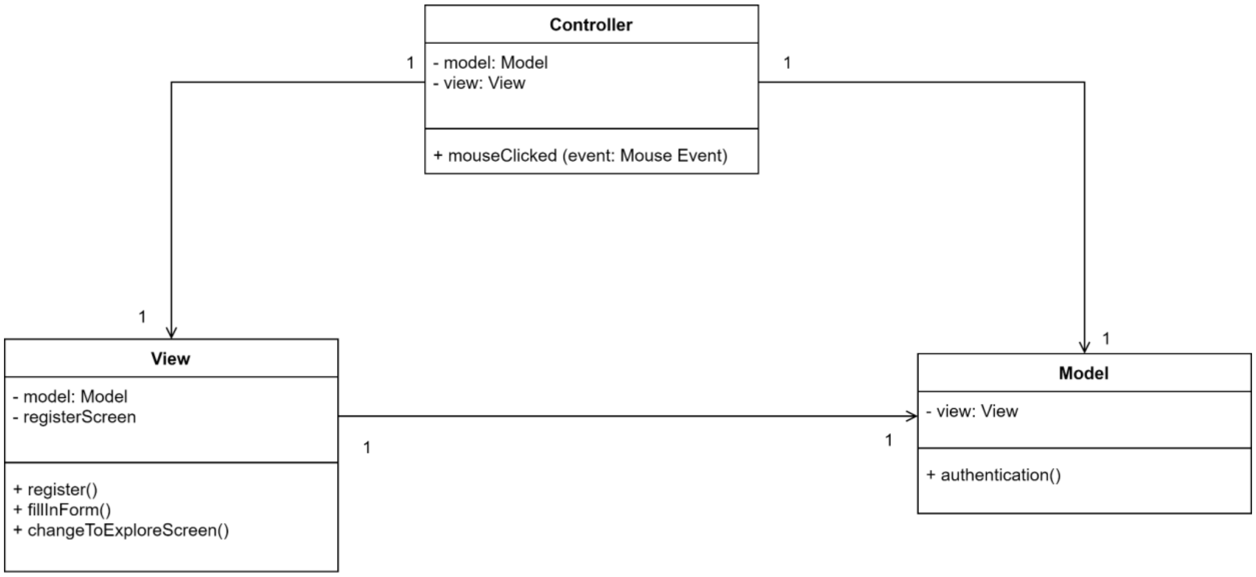
Model and View for processing them, the controller itself does not contain any logic. The Model is responsible for implementing application's specific logic as well as information storage and retrieval requests. Lastly, the View is responsible for displaying information to the user.

4.2.1 Component: MVC Diagram for LogIn Service



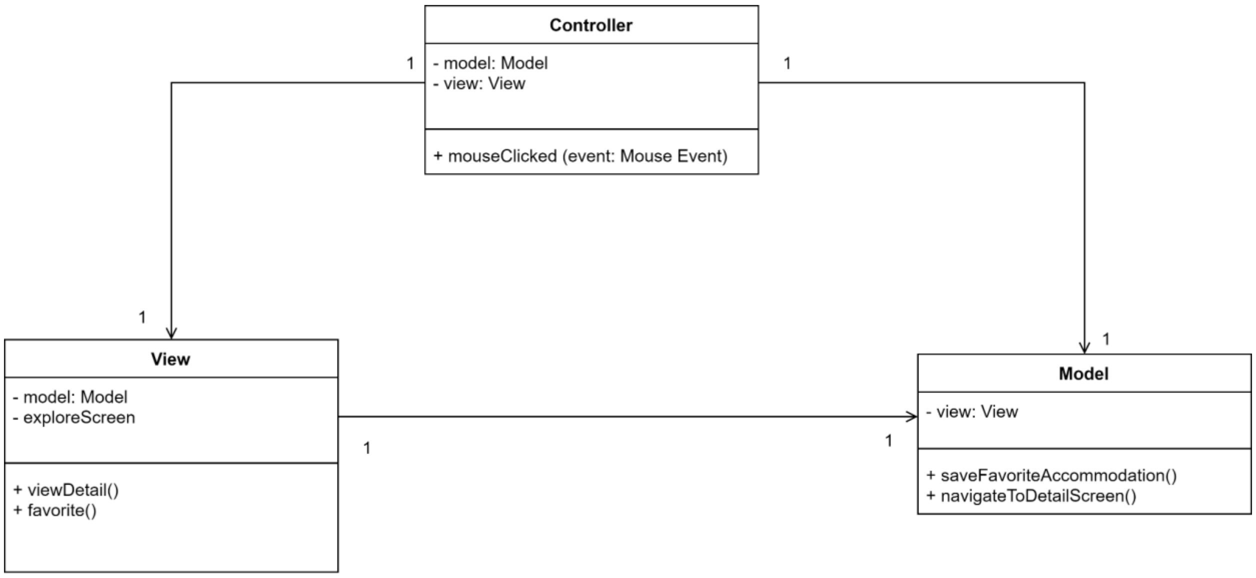
- Controller detect users' gestures
- View displays the login screen, which allows users to fill their username and password in the form, when they click the Login button, *login()* calls Model to authenticate that user by accessing the database and check whether that user exists or not or is the password correct. If both username and password are correct, View change from Login screen to the Explore page

4.2.2 Component: MVC Diagram for Register Service



- Controller detect users' gestures
- View displays the register screen, which allows users to fill their username and password in the form, when they click the Register button, *register()* calls Model to authenticate that user by checking whether username and password are in the right format. If yes, then the Model will access the database and require the database to add a user. Finally, View change from Register screen to the Explore page

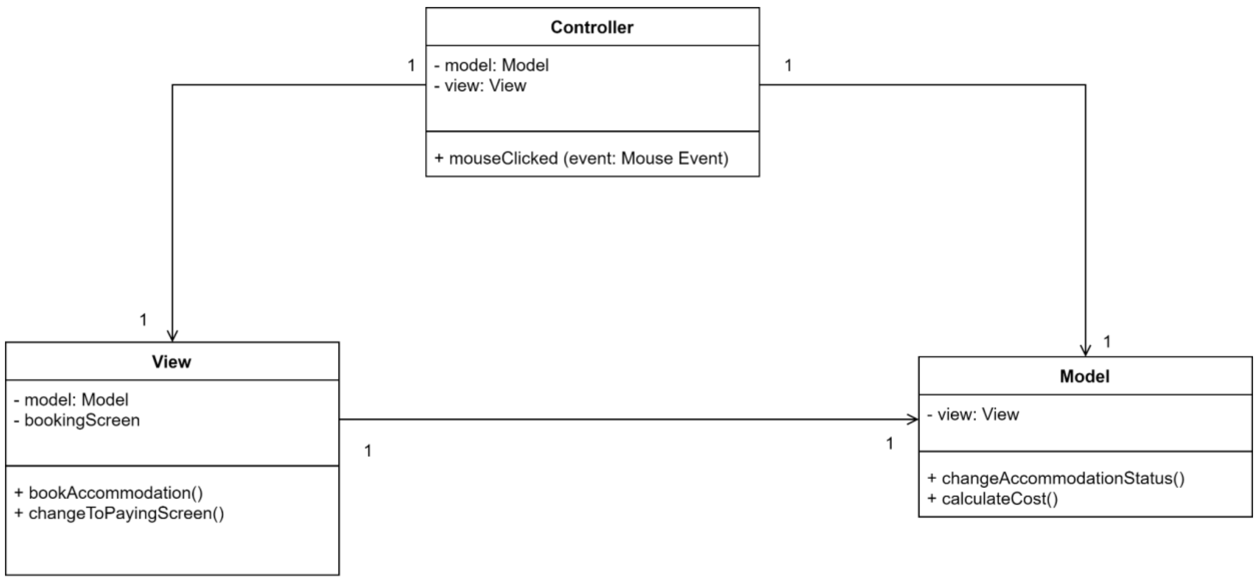
4.2.3 Component: MVC Diagram for Explore Service



- Controller detect users' gestures
- View displays the explore screen, which allows users to view a list of posted accommodations, when they click an accommodation, *viewDetail()* calls Model to navigate them to the Detail

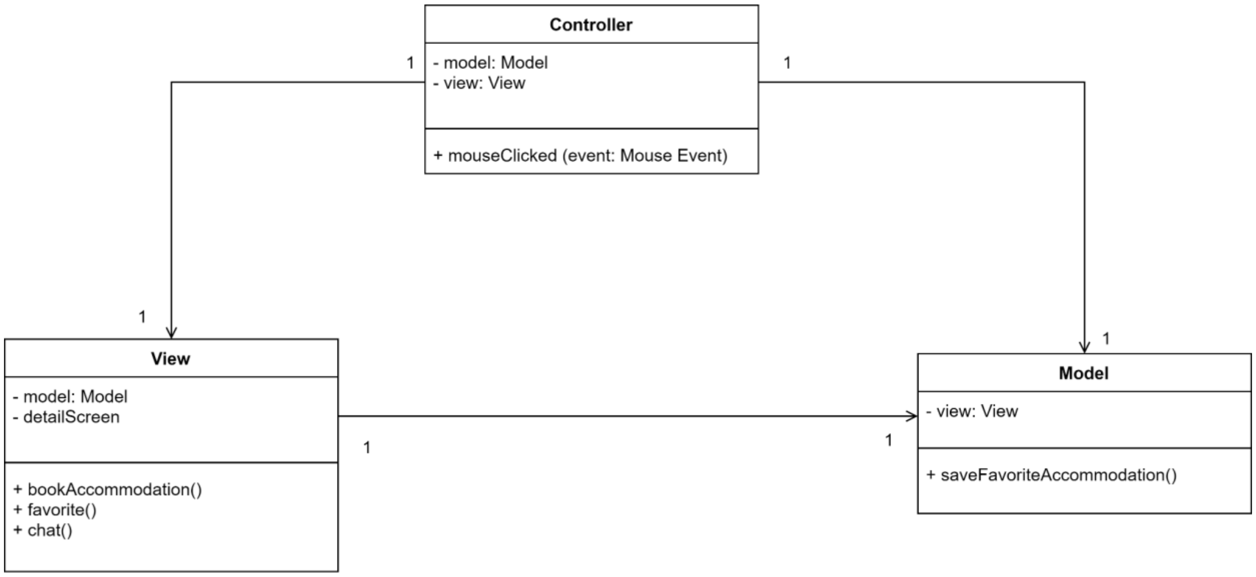
Screen of that accommodation by accessing the database and get all necessary information for displaying then the View will change to Detail Screen.
- Additionally, in the explore view, when the user taps the 'heart' icon, View triggers *favorite()* which calls the Model to add that accommodation to the user's wishlist in the database, then the View changes the icon to heart with a red color filled to indicate this accommodation is in the wishlist.

4.2.4 Component: MVC Diagram for Booking Service



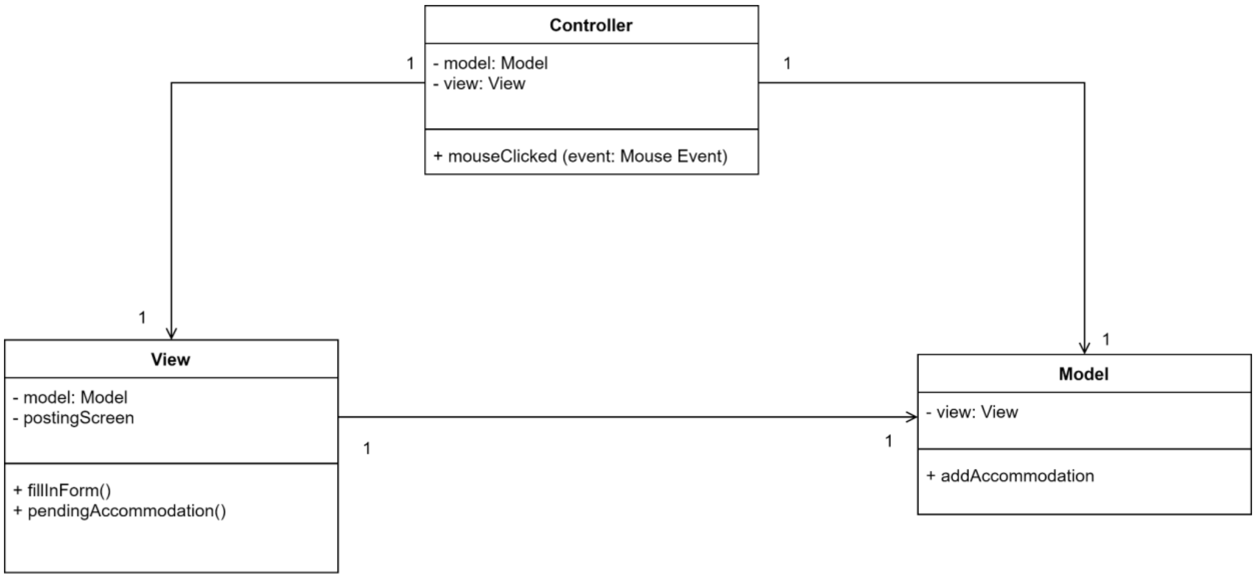
- Controller detect users' gestures
- View displays the booking screen for further transactions, which allows users to choose booking date, see the price and confirm the booking, when they click the 'Book this Accommodation' button, *bookAccommodation()* calls Model to change the accommodation's status in database from 'Vacant' to 'Booked'. Additionally, the Model also calculates the total price based on how many days the customers plan to stay and the basic price of accommodation itself. Finally, View pops up a window indicating that the payment is successful.

4.2.5 Component: MVC Diagram for Display Detail Service



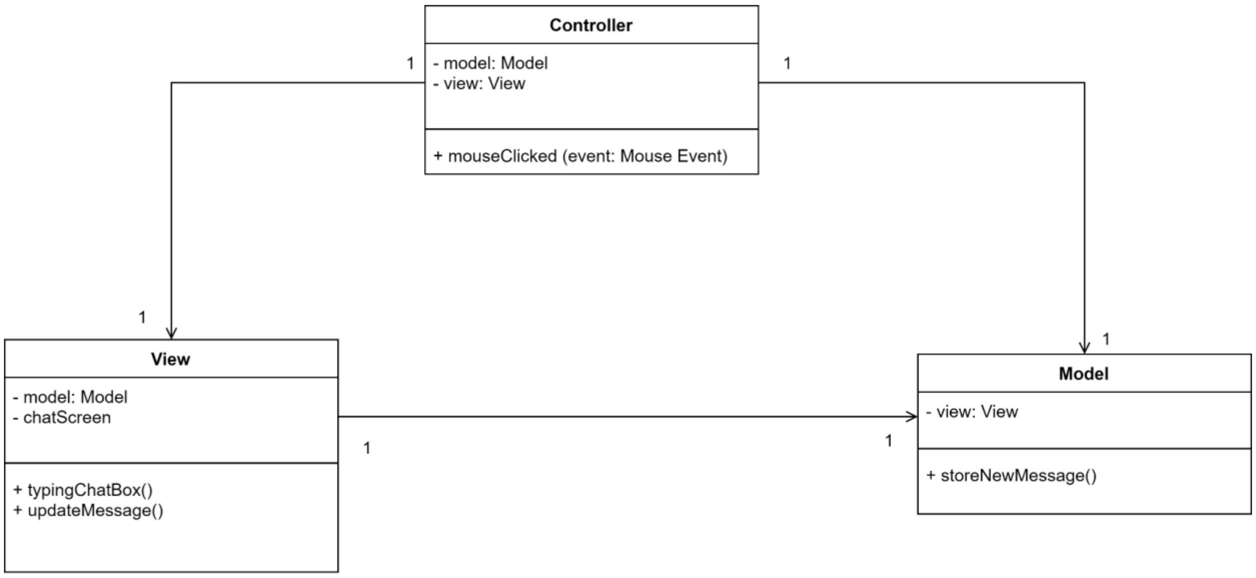
- Controller detect users' gestures
- View displays the detail screen, which allows users to have a more clearer and deeper view about the accommodation such as what services that accommodation provides, is it near the city center or not, etc; when they click the 'Book this Accommodation' button, the View changes to Booking Screen.
- Additionally, in the explore view, when the user taps the 'heart' icon, View triggers *favorite()* which calls the Model to add that accommodation to the user's wishlist in the database, then the View changes the icon to heart with a red color filled to indicate this accommodation is in the wishlist.
- When user click on the host's profile image, the View navigated them to their private chat box with that host

4.2.6 Component: MVC Diagram for posting Accommodation Service



- Controller detect users' gestures
- View displays the Posting screen for hosts to fill in necessary information about the Accommodation through *fillInForm()* method that displays what hosts type in, when they click the 'Posting' button, *pendingAccommodation()* pops up a dialog to announce that the accommodation is now in pending for acceptance from Admin, additionally, it calls Model to add this Accommodation to Database.

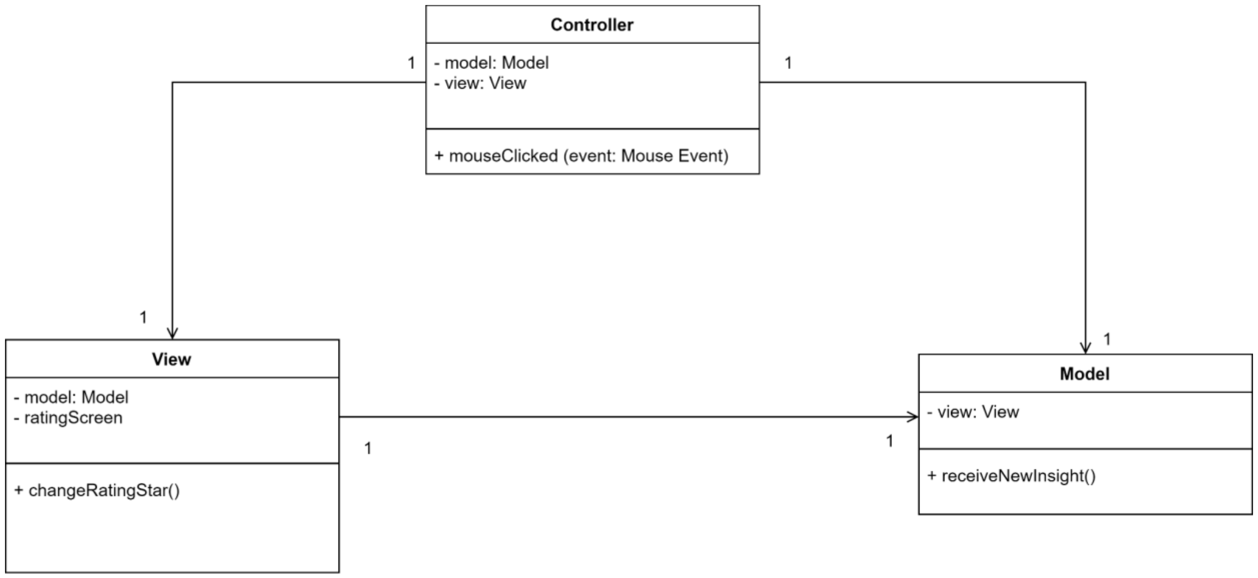
4.2.7 Component: MVC Diagram for Chat Service



- Controller detect users' gestures
- View displays the chat screen for text communication between hosts and renters, it allows users to type in the messages, when they click the 'Send' button, *updateMessage()* calls Model to store

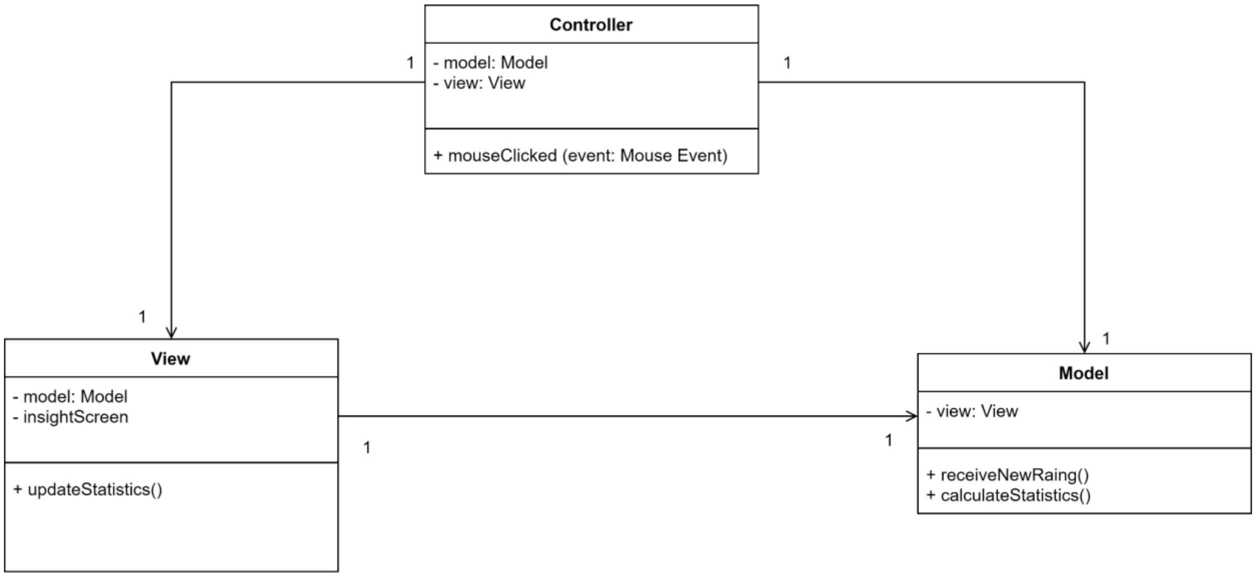
new messages to the database. Additionally, the View also immediately displays new messages on screen.

4.2.8 Component: MVC Diagram for Rating Service



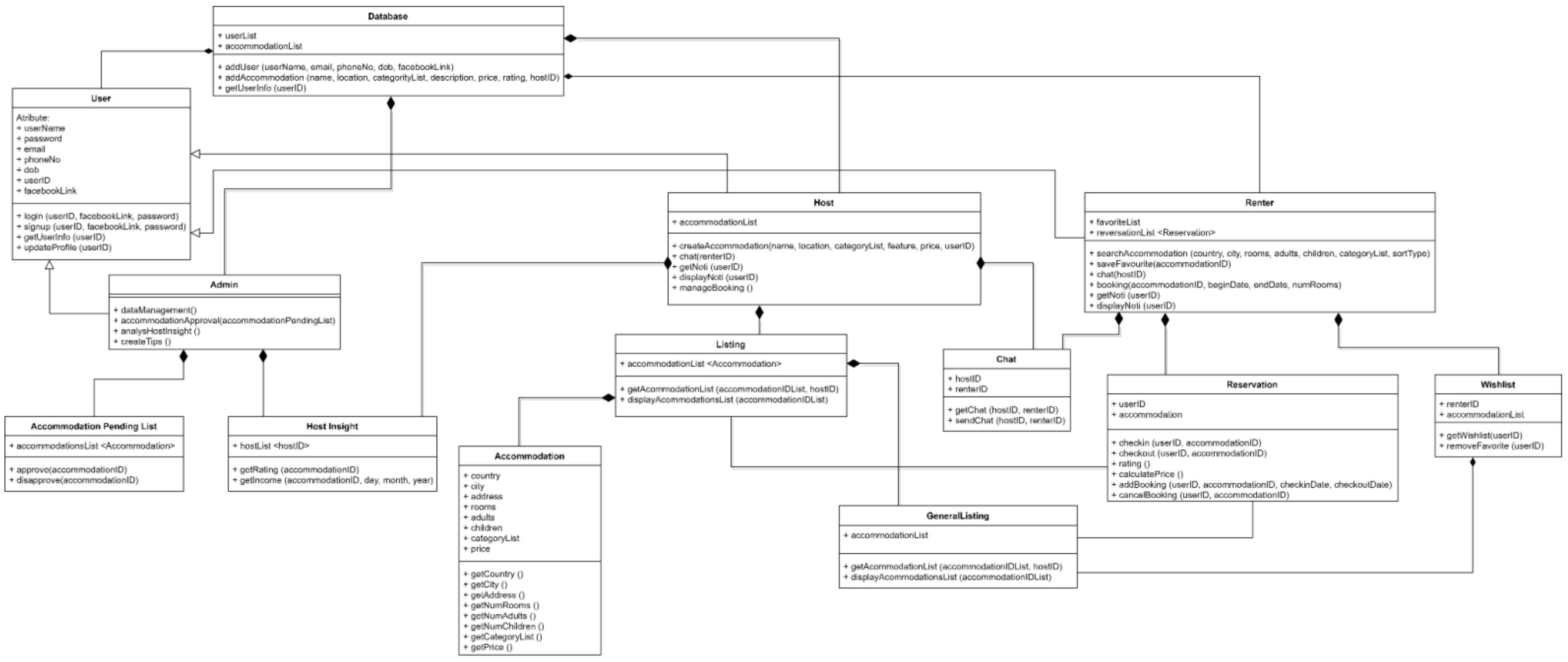
- Controller detect users' gestures
- View displays the rating screen after renters check out accommodations, which allows users to rate their services. The Controller will detect users' gestures to notify the View to display the amount of yellow filled stars out of 5 white stars. After the renters have rated the Accommodation, View calls Model to receive new feedback and add this to the database for re-calculating insight and statistics.

4.2.9 Component: MVC Diagram for Insight Service



- Controller detect users' gestures
- View displays the insight screen, which shows admins about an accommodation's statistics, the average rating, etc. When a user rates accommodation, View notifies Model to receive new rating and feedback, the Model then triggers *calculateStatistics()* to recalculate rating and statistics, then notifies back the View to display and update the insight.

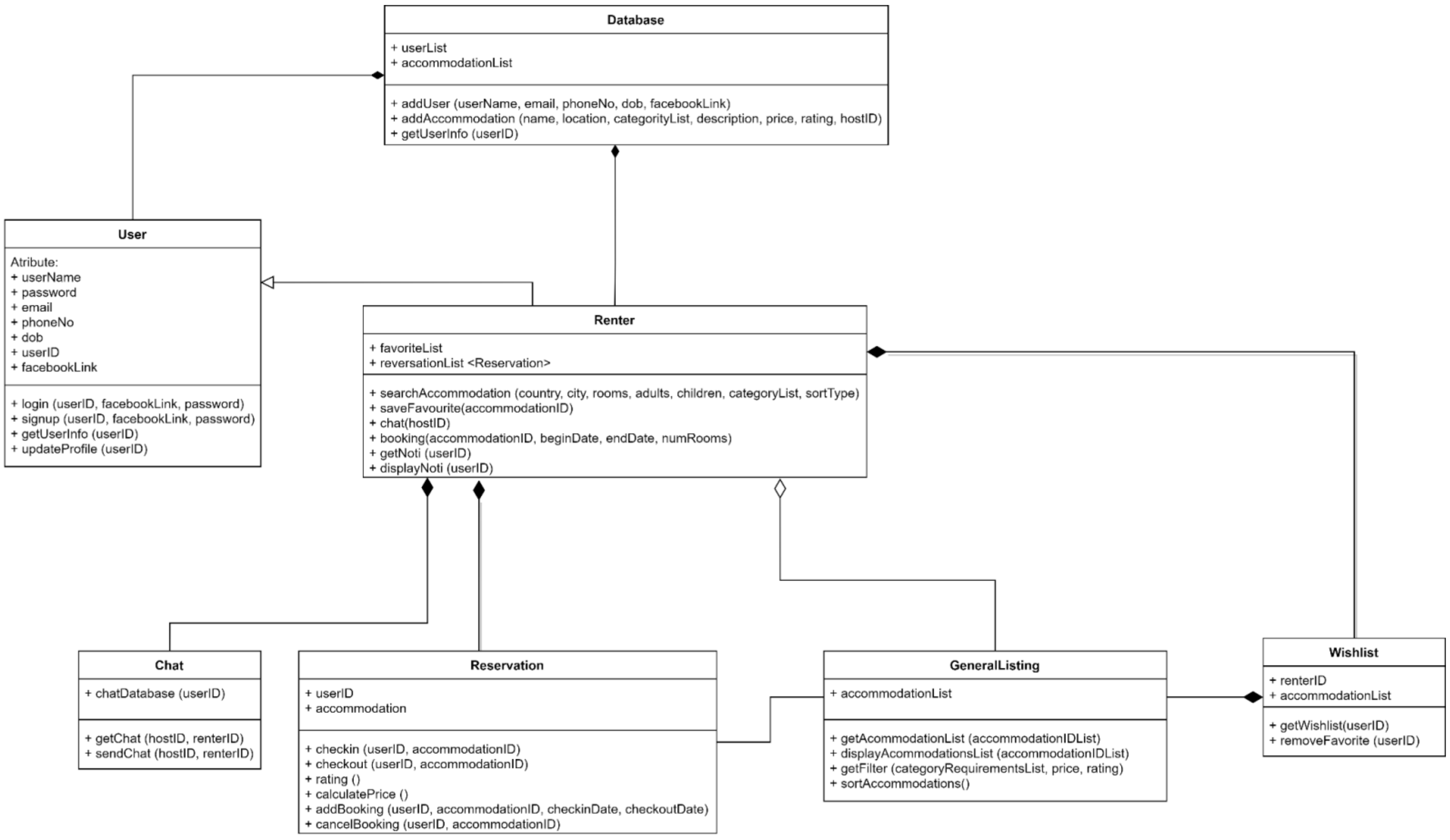
4.3 Class Diagram



Database
+ userList
+ accommodationList
+ addUser (userName, email, phoneNo, dob, facebookLink)
+ addAccommodation (name, location, categorityList, description, price, rating, hostID)
+ getUserInfo (userID)
User
Atribute:
+ userName
+ password
+ email
+ phoneNo
+ dob
+ userID
+ facebookLink
+ login (userID, facebookLink, password)
+ signup (userID, facebookLink, password)
+ getUserInfo (userID)
+ updateProfile (userID)
Admin
+ dataManagement()
+ accommodationApproval(accommodationPendingList)
+ analysHostInsight ()
+ createTips ()
Accommodation Pending List
+ accommodationsList <Accommodation>
+ approve(accommodationID)
+ disapprove(accommodationID)
Host Insight
+ hostList <hostID>
+ getRating (accommodationID)
+ getIncome (accommodationID, day, month, year)
Accommodation
+ country
+ city
+ address
+ rooms
+ adults
+ children
+ categoryList
+ price
+ getCountry ()
+ getCity ()
+ getAddress ()
+ getNumRooms ()
+ getNumAdults ()
+ getNumChildren ()
+ getCategoryList ()
+ getPrice ()
Host
+ accommodationList
+ createAccommodation(name, location, categoryList, feature, price, userID)
+ chat(renterID)
+ getNoti (userID)
+ displayNoti (userID)
+ manageBooking ()
Listing
+ accommodationList <Accommodation>
+ getAcommodationList (accommodationIDList, hostID)
+ displayAcommodationsList (accommodationIDList)
Chat
+ hostID
+ renterID
+ getChat (hostID, renterID)
+ sendChat (hostID, renterID)
Renter
+ favoriteList
+ reversationList <Reservation>
+ searchAccommodation (country, city, rooms, adults, children, categoryList, sortType)
+ saveFavourite(accommodationID)
+ chat(hostID)
+ booking(accommodationID, beginDate, endDate, numRooms)
+ getNoti (userID)
+ displayNoti (userID)
Reservation
+ userID
+ accommodation
+ checkin (userID, accommodationID)
+ checkout (userID, accommodationID)
+ rating ()
+ calculatePrice ()
+ addBooking (userID, accommodationID, checkinDate, checkoutDate)
+ cancelBooking (userID, accommodationID)
Wishlist
+ renterID
+ accommodationList
+ getWishlist(userID)
+ removeFavorite (userID)
GeneralListing
+ accommodationList
+ getAcommodationList (accommodationIDList, hostID)
+ displayAcommodationsList (accommodationIDList)

Diagram

4.3.1 Component: Renter:

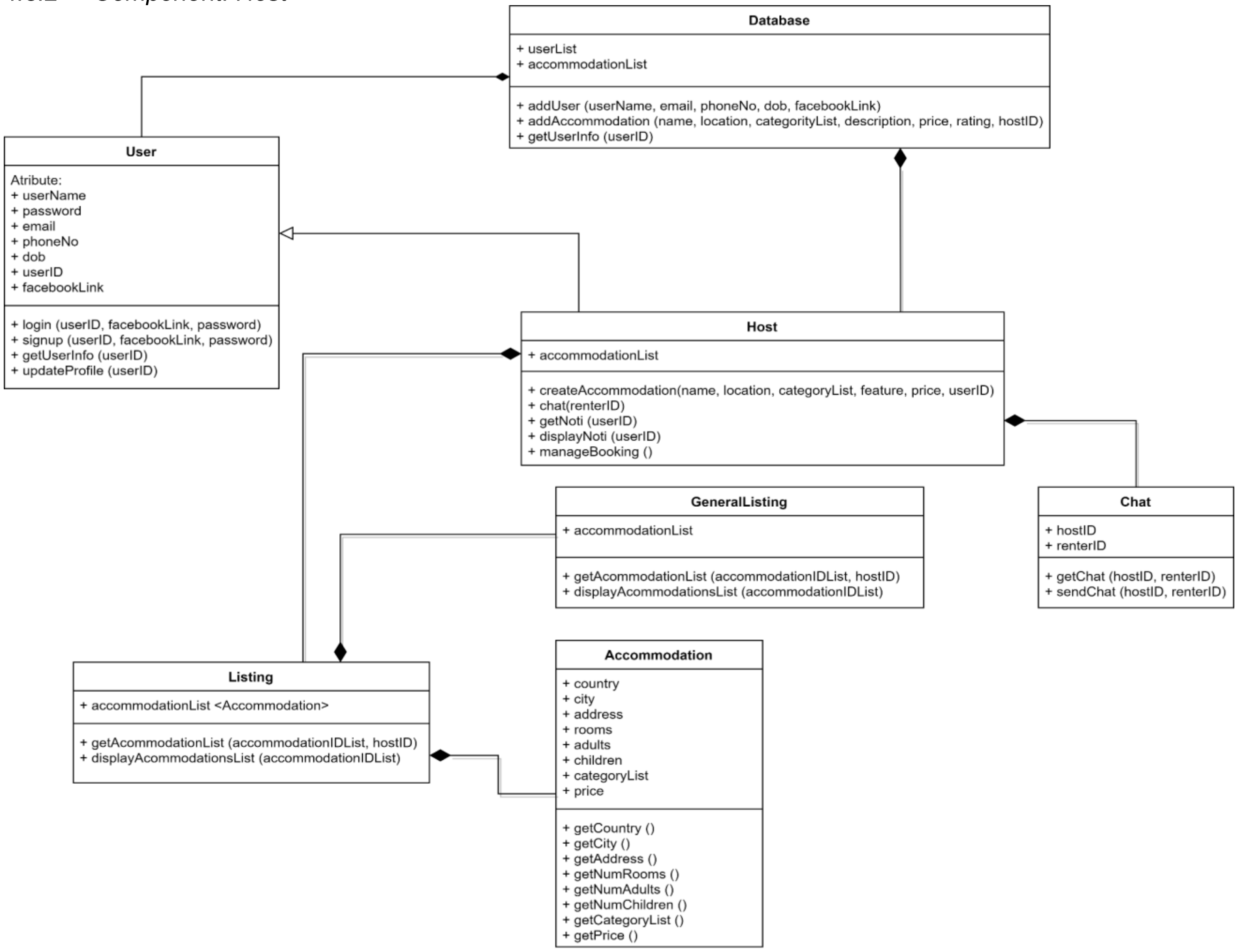

Database
+ userList
+ accommodationList
+ addUser (userName, email, phoneNo, dob, facebookLink)
+ addAccommodation (name, location, categorityList, description, price, rating, hostID)
+ getUserInfo (userID)
User
Atribute:
+ userName
+ password
+ email
+ phoneNo
+ dob
+ userID
+ facebookLink
+ login (userID, facebookLink, password)
+ signup (userID, facebookLink, password)
+ getUserInfo (userID)
+ updateProfile (userID)
Renter
+ favoriteList
+ reversationList <Reservation>
+ searchAccommodation (country, city, rooms, adults, children, categoryList, sortType)
+ saveFavourite(accommodationID)
+ chat(hostID)
+ booking(accommodationID, beginDate, endDate, numRooms)
+ getNoti (userID)
+ displayNoti (userID)
Chat
+ chatDatabase (userID)
+ getChat (hostID, renterID)
+ sendChat (hostID, renterID)
Reservation
+ userID
+ accommodation
+ checkin (userID, accommodationID)
+ checkout (userID, accommodationID)
+ rating ()
+ calculatePrice ()
+ addBooking (userID, accommodationID, checkinDate, checkoutDate)
+ cancelBooking (userID, accommodationID)
GeneralListing
+ accommodationList
+ getAcommodationList (accommodationIDList)
+ displayAcommodationsList (accommodationIDList)
+ getFilter (categoryRequirementsList, price, rating)
+ sortAccommodations()
Wishlist
+ renterID
+ accommodationList
+ getWishlist(userID)
+ removeFavorite (userID)

Responsibilities and/or Services:

- *User* is used as an abstraction class for class *Renter*, it provides basic information for any user such as name, user id, phone number, etc as well as basic methods for users:
    + *login*: To log in into the app
    + *signup*: help user to register a new account for future login
    + *getUserInfo*: return information of user
    + *updateProfile*: update information in personal profile
- *Renter* is a class for actors in the renter role and inherits the *'User'* abstract class. It is used to store wishlist and list of user's booked accommodation, search and save accommodation, chat with host, book new accommodation and display notification as well as notify users with new update
- *User* and *Renter* classes are a part of *userList* in class *Database*, which contains attributes and methods to manage the app's database such as add *Accommodations* and *User*.
- Each user owns a list of *Chat* classes indicating the chat function with hosts. Basically, with *Chat* class, we can get messages sent between user and hosts as well as sending messages between two actors

- *favoriteList* in class *User* is a list of *Wishlist* classes. This class helps users to view their favorite accommodation and remove them if they want to.
- *GeneralListing* is in independent class associates with both *User* and *Host* component, it is used to display any list of *Accommodations* in the app with necessary methods such as filtering accommodation result, display the list, get *Accommodation* element in the list and sort *Accommodation* list accordance to user's requirements.
- *Wishlist* class also owns *GeneralListing* class to display its elements in *AccommodationList* attribute
- reservationList in User class is a list of Reservations. This class contains an attribute of *UserID* and an *Accommodation*, together with essential methods for booking process such as check in, check out accommodation, rate accommodation after checking out, calculate the total price and cancel booking.
- Activities of renter such as searching and saving accommodations are related to retrieving and saving data from/to the *Database*.

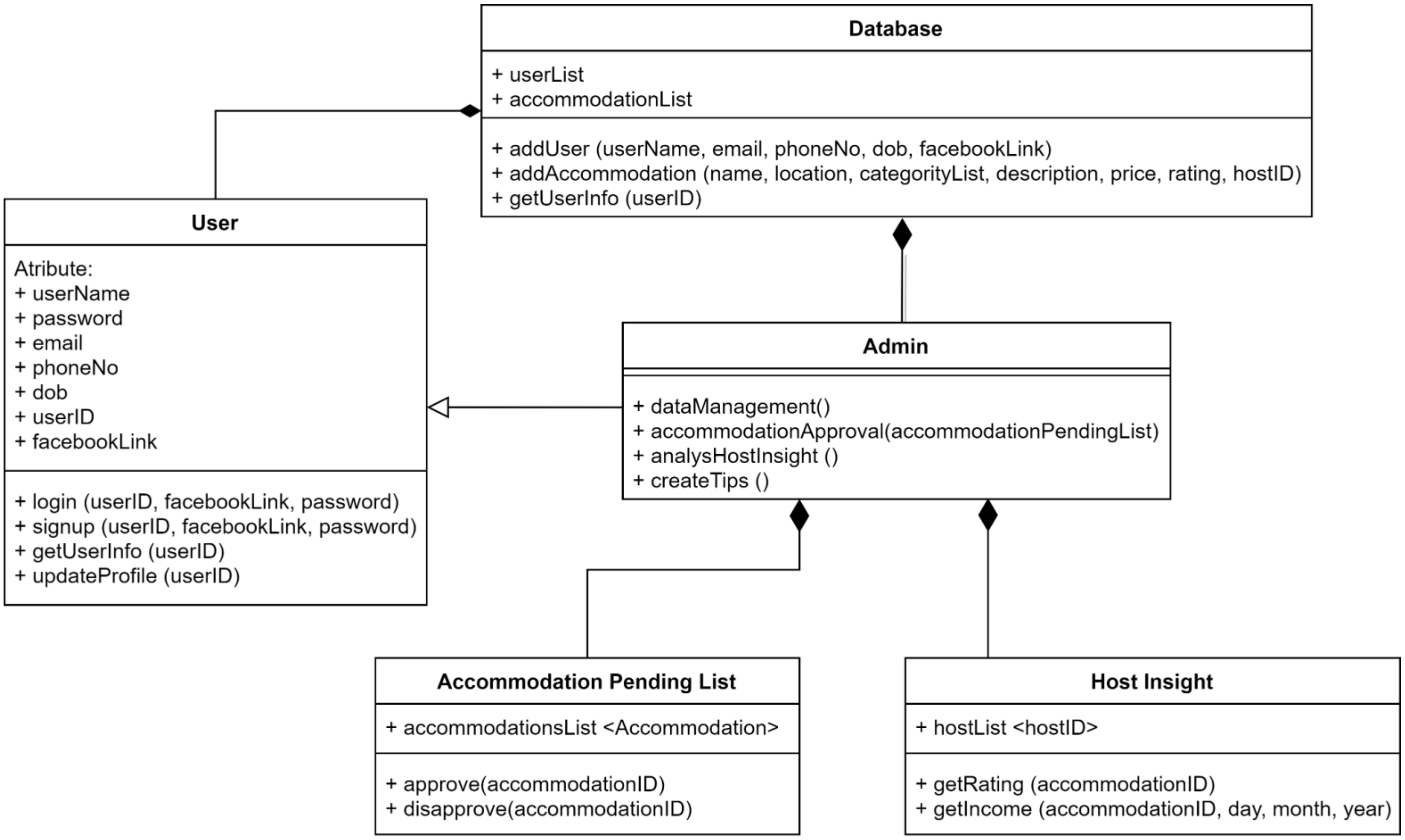
4.3.2 Component: Host


Database
+ userList
+ accommodationList
+ addUser (userName, email, phoneNo, dob, facebookLink)
+ addAccommodation (name, location, categorityList, description, price, rating, hostID)
+ getUserInfo (userID)
User
Atribute:
+ userName
+ password
+ email
+ phoneNo
+ dob
+ userID
+ facebookLink
+ login (userID, facebookLink, password)
+ signup (userID, facebookLink, password)
+ getUserInfo (userID)
+ updateProfile (userID)
Host
+ accommodationList
+ createAccommodation(name, location, categoryList, feature, price, userID)
+ chat(renterID)
+ getNoti (userID)
+ displayNoti (userID)
+ manageBooking ()
GeneralListing
+ accommodationList
+ getAcommodationList (accommodationIDList, hostID)
+ displayAcommodationsList (accommodationIDList)
Chat
+ hostID
+ renterID
+ getChat (hostID, renterID)
+ sendChat (hostID, renterID)
Listing
+ accommodationList <Accommodation>
+ getAcommodationList (accommodationIDList, hostID)
+ displayAcommodationsList (accommodationIDList)
Accommodation
+ country
+ city
+ address
+ rooms
+ adults
+ children
+ categoryList
+ price
+ getCountry ()
+ getCity ()
+ getAddress ()
+ getNumRooms ()
+ getNumAdults ()
+ getNumChildren ()
+ getCategoryList ()
+ getPrice ()

Responsibilities and/or Services:

- Similar to Renter component, *User* is used as an abstraction class for class *Admin*, it provides basic information for any user such as name, user id, phone number, etc as well as basic methods for users:
    - + *login*: To log in into the app
    - + *signup*: help user to register a new account for future login
    - + *getUserInfo*: return information of user
    - + *updateProfile*: update information in personal profile
- *Admin* is an inheriting class of *User*. Every Host has a list of his/her accommodations. It provides methods for host to create new accommodations, chat with customers, get notified with any changes or updates and display all of his/her notification, as well as manage all the booking process of each accommodation
- Host owns an attribute named *accommodationList*, which is a Listing class data type. This class has an attribute which is a list of Accommodation class and functions to retrieve and display a list of Accommodation and manage the list of their posted - rooms.
- Each element in the *accommodationList* attribute in the Listing class is a list of *Accommodation* instances which consists of attributes such as country, city, address, rooms,... and functions to get the information in *Accommodation* class.
- Host owns a list of *Chat* classes which are identified by the *hostID* and *renterID*, saving conversations between Host and Renter. With *Chat* class, we can get messages or send messages between these two actors.
- All the basic information of Host which is the attributes of *User* is stored in the Database. Database also stores the *accommodationList* of Host. Whenever the Host *createAccommodation()*, the accommodation's data will be saved in the Database by the function *addAccommodation()*.

4.3.3 *Component: Admin:*

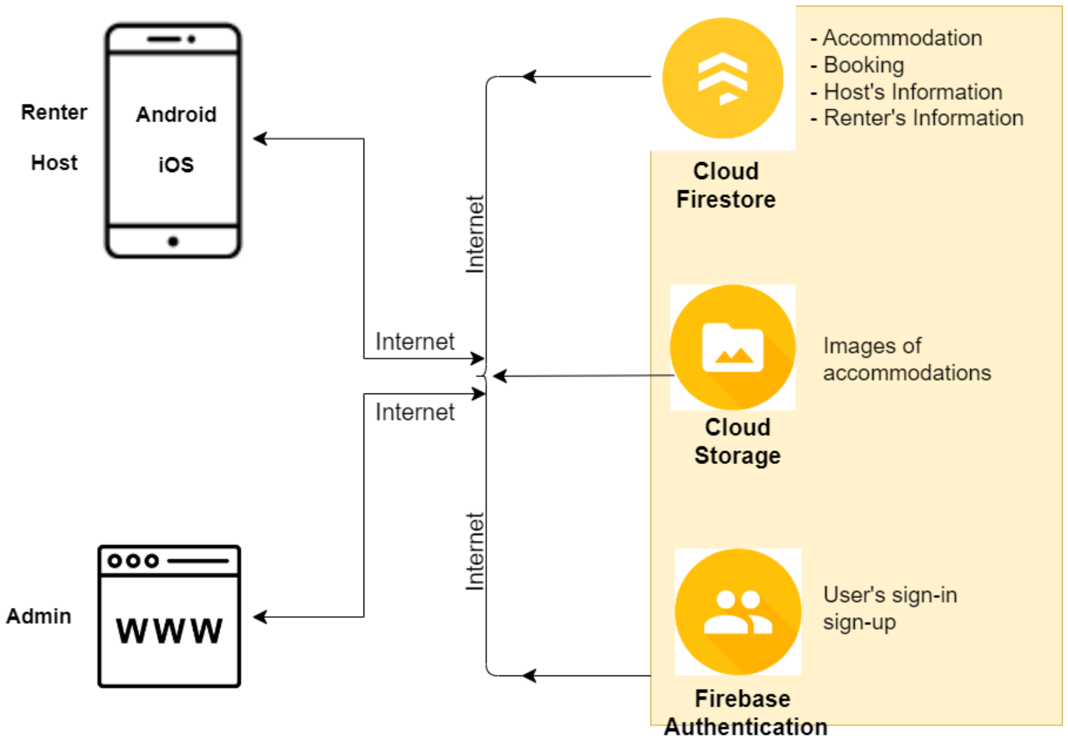

Database
+ userList
+ accommodationList
+ addUser (userName, email, phoneNo, dob, facebookLink)
+ addAccommodation (name, location, categorityList, description, price, rating, hostID)
+ getUserInfo (userID)
User
Atribute:
+ userName
+ password
+ email
+ phoneNo
+ dob
+ userID
+ facebookLink
+ login (userID, facebookLink, password)
+ signup (userID, facebookLink, password)
+ getUserInfo (userID)
+ updateProfile (userID)
Admin
+ dataManagement()
+ accommodationApproval(accommodationPendingList)
+ analysHostInsight ()
+ createTips ()
Accommodation Pending List
+ accommodationsList <Accommodation>
+ approve(accommodationID)
+ disapprove(accommodationID)
Host Insight
+ hostList <hostID>
+ getRating (accommodationID)
+ getIncome (accommodationID, day, month, year)

Responsibilities and/or Services:

- Similar to Renter component, *User* is used as an abstraction class for class *Admin*, it provides basic information for any user such as name, user id, phone number, etc as well as basic methods for users:
  + *login*: To log in into the app
  + *signup*: help user to register a new account for future login
  + *getUserInfo*: return information of user
  + *updateProfile*: update information in personal profile
- Admin is an inheriting class of *User*. Admin owns two methods. The first one is *dataManagement()* which is used for managing the data of the whole system. The second one is *accommodationApproval()*, in this method, Admin retrieves the *accommodationList* in the Database sent by Host to do the approval process.
- Admin owns a list of *Accommodation Pending list* class which contains a list of accommodation needed to be approved. Admin can approve an accommodation or disapprove.
- The *anaylysHostInsight* function interacts with the *Host Insight* class. Admin owns a list of Hosts named *hostList* which saves a list of *hostID* and is used for managing the Hosts.

5. Deployment

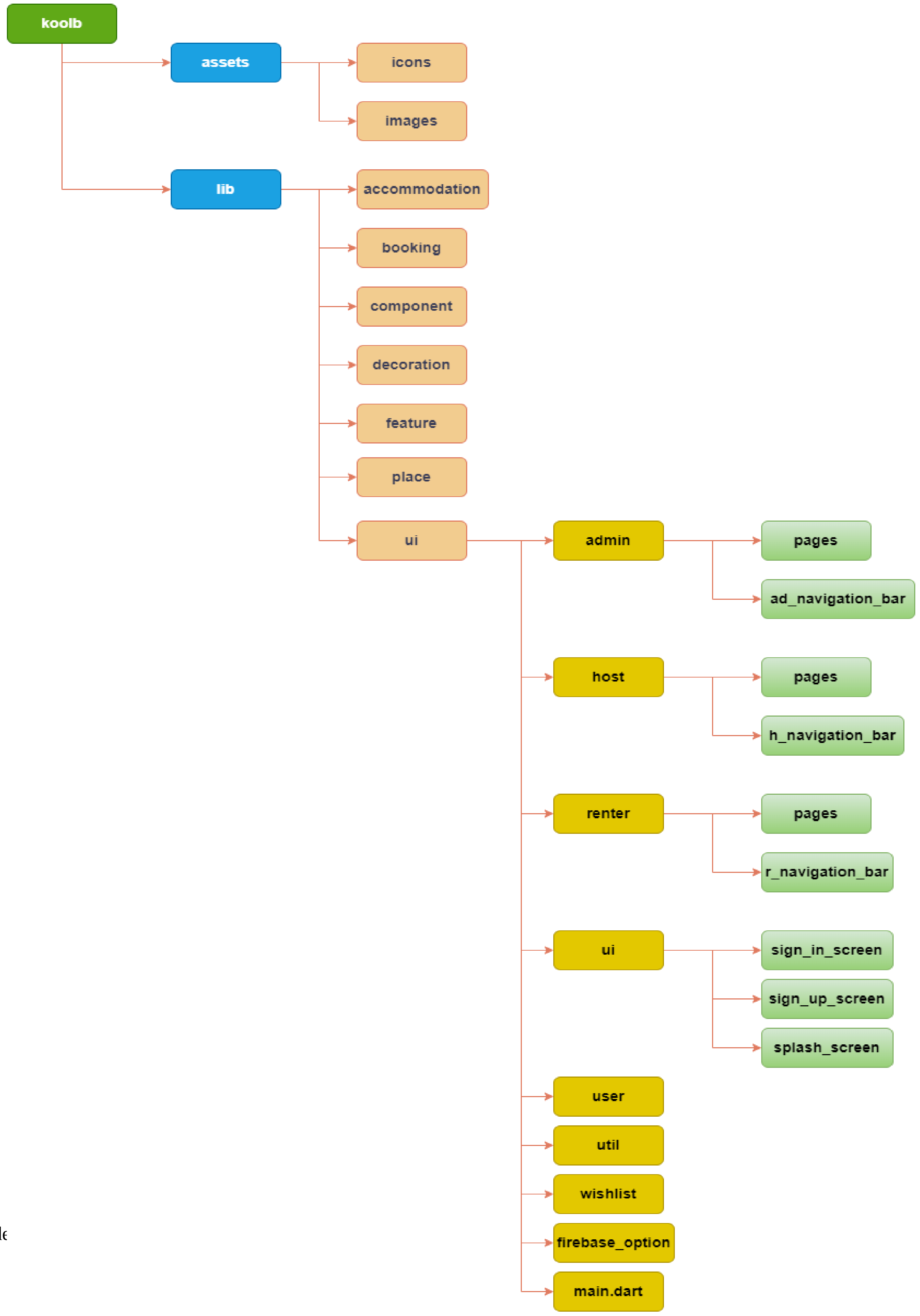
The following is a description of the hardware nodes running the execution environment for the application system.



These are the identify hardware components:

- Back-end as a Service Firebase. This is a NoSQL database program which stores our data in JSON-like documents. Our app can talk directly to the backend without a server. Firebase Backend provides us with:
  - Cloud Firestore: used to store data of accommodations, bookings, hosts' information and renters' information
  - Cloud Storage: used to store images of accommodations and user's profile image.
  - Firebase Authentication: used to store each user ID and helps us with the sign-in and sign-up process.
- Mobile devices (Android, IOS,...) to run the mobile application.

6. Implementation View


koolb
assets
icons
images
lib
accommodation
booking
component
decoration
feature
place
ui
admin
pages
ad_navigation_bar
host
pages
h_navigation_bar
renter
pages
r_navigation_bar
ui
sign_in_screen
sign_up_screen
splash_screen
user
util
wishlist
firebase_option
main.dart

The following is a description of the folder structure of our code for all the components:

- Renter's folder: This folder contains all file that implement the renter component
    - A class to implement the information that involves renters.
    - UI interface so that the renter can operate:
        - Sign-in, Register account as Renters
        - Homepage, Navigation
        - Chatbox
    - Other implementation that benefits the renters' role
        - Notification
        - Wishlist
        - Booking
        - Detail view
        - Setting
        - Checkin/ Checkout feature
        - Rating feature
- Host's folder: This folder contains all file that has been proposed on the host component
    - A class to implement the information that involves hosts.
    - UI interface so that the renter can operate:
        - Sign-in, Register account as Renters
        - Explore, Navigation
        - Chatbox
    - Other implementation that benefits the renters' role
        - Post Accommodation
        - Detail view
        - Insight
        - Notification
        - Reservation list
        -
- Admin's folder: This folder contains all files associated with admin component
    - A class to implement the information that involves hosts.
    - Implementation that benefits the admin's role:
        - Request page
        - Setting page
        - Statistic page
        - FAQ feature
        - Chatbox