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 |

Table of Contents

[**1. Introduction**](#_heading=h.gjdgxs) **5**

[Purpose](#_heading=h.tzqsbnhi77yd) 5

[Scope](#_heading=h.bwr2p6l2gnog) 5

[Reference](#_heading=h.ffxr3m1sitig) 5

[Document Content Overview](#_heading=h.jyua6w9u5dc8) 5

[**2. Architectural Goals and Constraints**](#_heading=h.30j0zll) **5**

[**3. Use-Case Model**](#_heading=h.1fob9te) **5**

[**4. Logical View**](#_heading=h.2et92p0) **9**

[4.1 Overview](#_heading=h.tyjcwt) 9

[4.2 Logical View](#_heading=h.12jb34z78a6d) 9

[4.2.1 Register:](#_heading=h.r891awinzee1) 9

[4.2.1.1 Relationship with ‘Log in’:](#_heading=h.s2s8lfi54efx) 9

[4.2.1.2 Relationship with ‘Setting’:](#_heading=h.n520yj51g91i) 9

[4.2.2 Log in:](#_heading=h.5qjp88wl7ckc) 9

[4.2.2.1 Relationship with ‘Homepage’:](#_heading=h.mmb2ftat8143) 9

[4.2.2.2 Relationship with ‘Booking’ and ‘Wishlist’:](#_heading=h.akr79jmpo7zd) 9

[4.2.3 Homepage:](#_heading=h.s3f8vedmq2fr) 9

[4.2.3.1 Relationship with ‘Profile’:](#_heading=h.mixfisk1mwa8) 10

[4.2.3.2 Relationship with ‘Chatbox’:](#_heading=h.fjr4za93bpha) 10

[4.2.3.3 Relationship with ‘Notification’:](#_heading=h.v12p9nemi3mn) 10

[4.2.3.4 Relationship with ‘Search Room;:](#_heading=h.bfdweuv01h6e) 10

[4.2.3.5 Relationship with ‘Booking’:](#_heading=h.oq6f8maiu26z) 10

[4.2.3.6 Relationship with ‘Favorite List’:](#_heading=h.xrjadcxypp1e) 10

[4.2.3.7 Relationship with ‘Reservation List View’](#_heading=h.t1l7jsmlju2g) 11

[4.2.3.8 Relationship with ‘Customer Service’](#_heading=h.y16bqswrsyye) 11

[4.2.3.9 Relationship with ‘Host Insight’](#_heading=h.qq0qiogm23fw) 11

[4.2.4 Booking:](#_heading=h.ovgsy5138wi5) 11

[4.2.4.1 Relationship with ‘Cancel Booking’:](#_heading=h.olsvatsv0y0f) 11

[4.2.4.2 Relationship with ‘Check In’:](#_heading=h.90p6jke39e0y) 11

[4.2.4.3 Relationship with ‘Check Out’:](#_heading=h.ocyh8ye6oyw) 11

[4.2.4.4 Relationship with ‘Accommodation Detail’:](#_heading=h.g4ls8ohjucw5) 11

[4.2.5 Setting:](#_heading=h.voczu0qq8xvr) 12

[4.2.5.1 Relationship with ‘Update profile’:](#_heading=h.466ozogrfctg) 12

[4.2.5.2 Relationship with ‘Language Setting’:](#_heading=h.5nbgmrkjr6gt) 12

[4.2.5.3 Relationship with ‘Configure payment’:](#_heading=h.kj0zbpaxota8) 12

[4.2.6 Checkout:](#_heading=h.4evqzxdvoeig) 12

[4.2.6.1 Relationship with ‘Rating’:](#_heading=h.ldvv9piwdup0) 12

[4.2.7 Rating:](#_heading=h.33024zk98ke3) 12

[4.2.7.1 Relationship with Accommodation Detail:](#_heading=h.q7m2emkoz9yk) 12

[4.2.8 Search room:](#_heading=h.aw4o2r5gim9g) 13

[4.2.8.1 Relationship with ‘Sort and Filter’:](#_heading=h.z0g0zr29crl3) 13

[4.2.9 Sort and Filter:](#_heading=h.8g5l44a2dlsp) 13

[4.2.9.1 Relationship with ‘Search room’:](#_heading=h.eic99398eitk) 13

[4.2.10 Post room Information:](#_heading=h.ubu4ibm8kn0z) 13

[4.2.10.1 Relationship with Accommodation Detail:](#_heading=h.gluymqnbmjlb) 13

[4.2.11 Host Insight:](#_heading=h.89qnmjy3pao3) 13

[4.2.11.1 Relationship with Rating:](#_heading=h.2h3qhwba6umx) 13

[**5. Deployment**](#_heading=h.3dy6vkm) **13**

[**6. Implementation View**](#_heading=h.1t3h5sf) **13**

Software Architecture Document

# Introduction

## 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.

## 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.

## Reference

1. Tutorial Video
2. Software Architecture Document template
3. [Vision Document.docx](https://docs.google.com/document/u/0/d/1fsYqurav5LSKLh1UMCeumz7CMvVELdna/edit)

## 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.

# Architectural Goals and Constraints

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

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

# 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:**

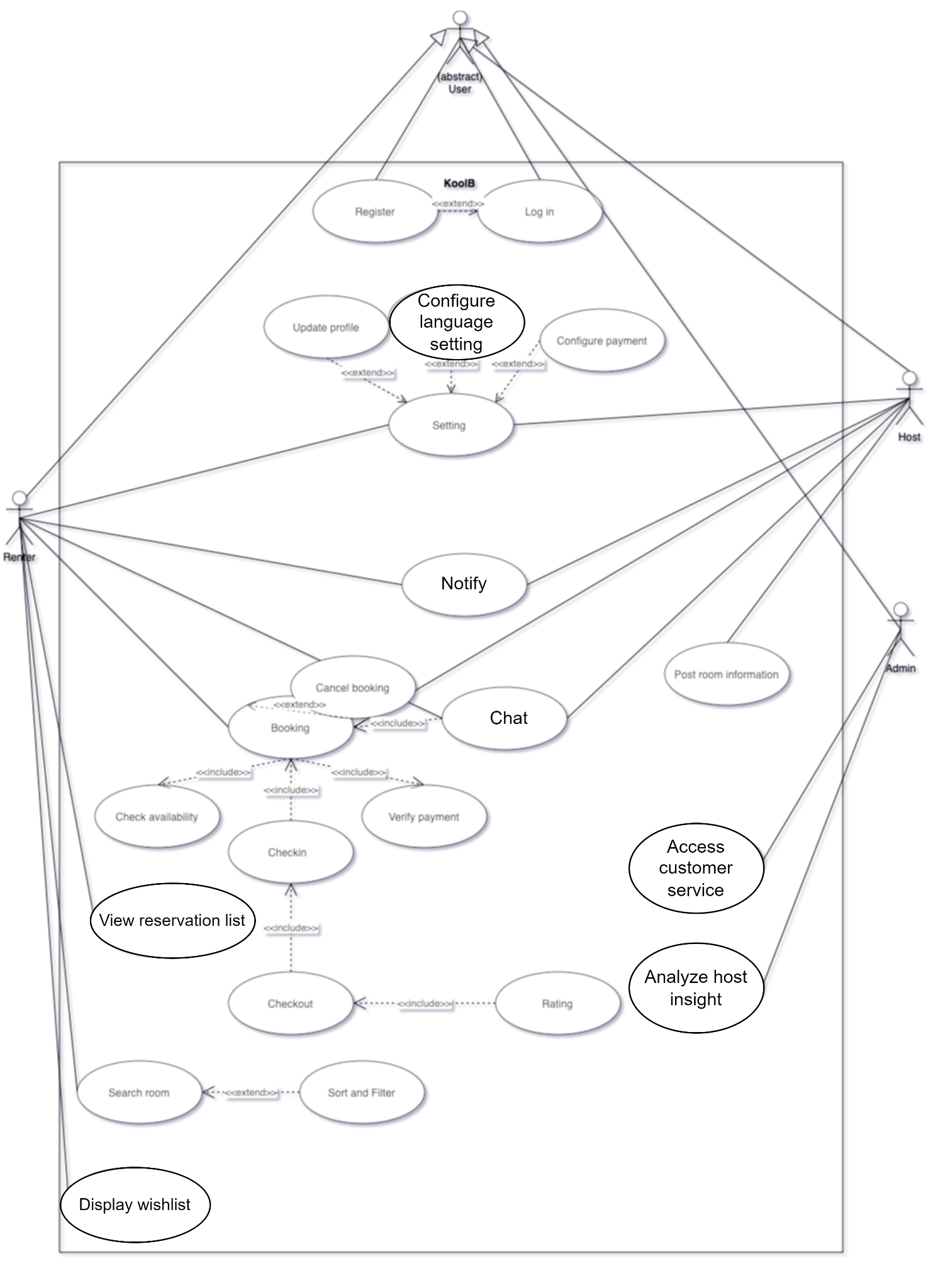
The KoolB App Customer service process is described in this use case.

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”.



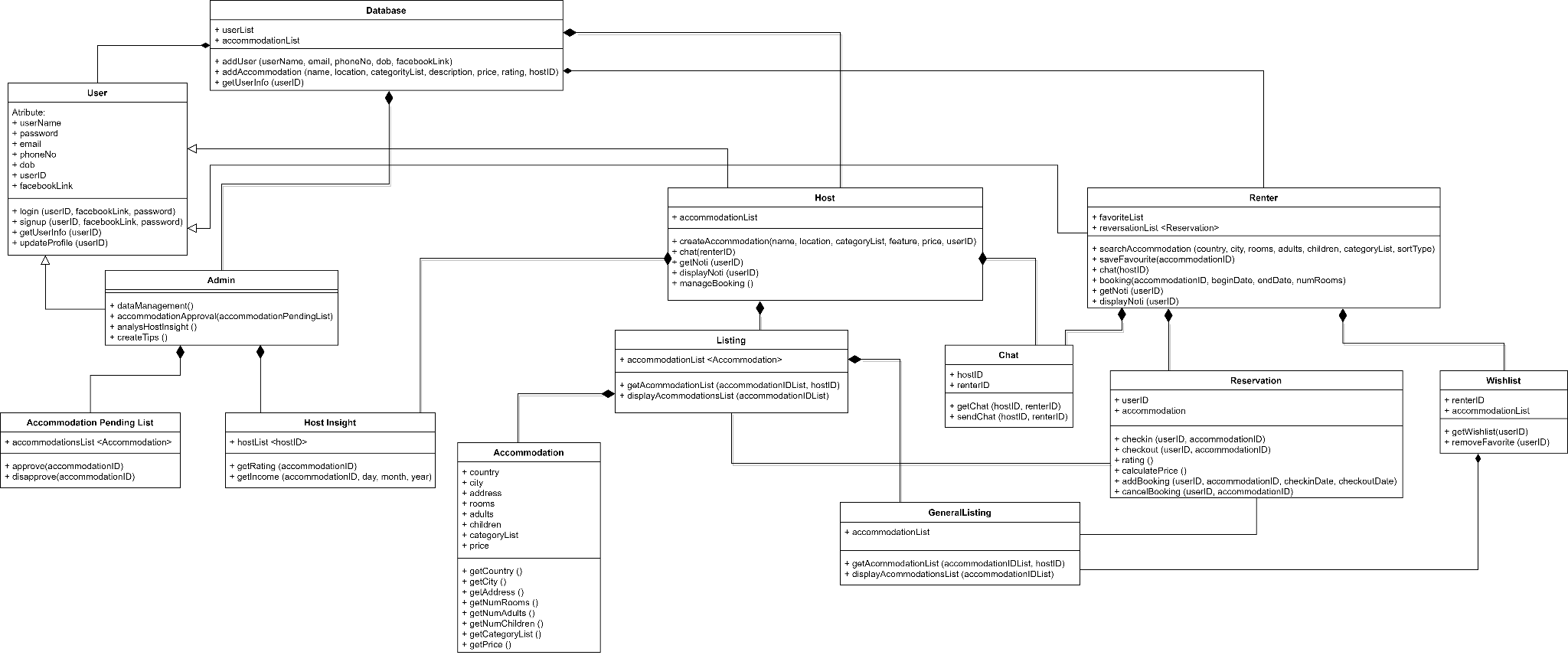
# Logical View

## Overview

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

## Logical View

Package diagram

Diagram

### Component: Renter:

**Responsibilities and/or Services:**

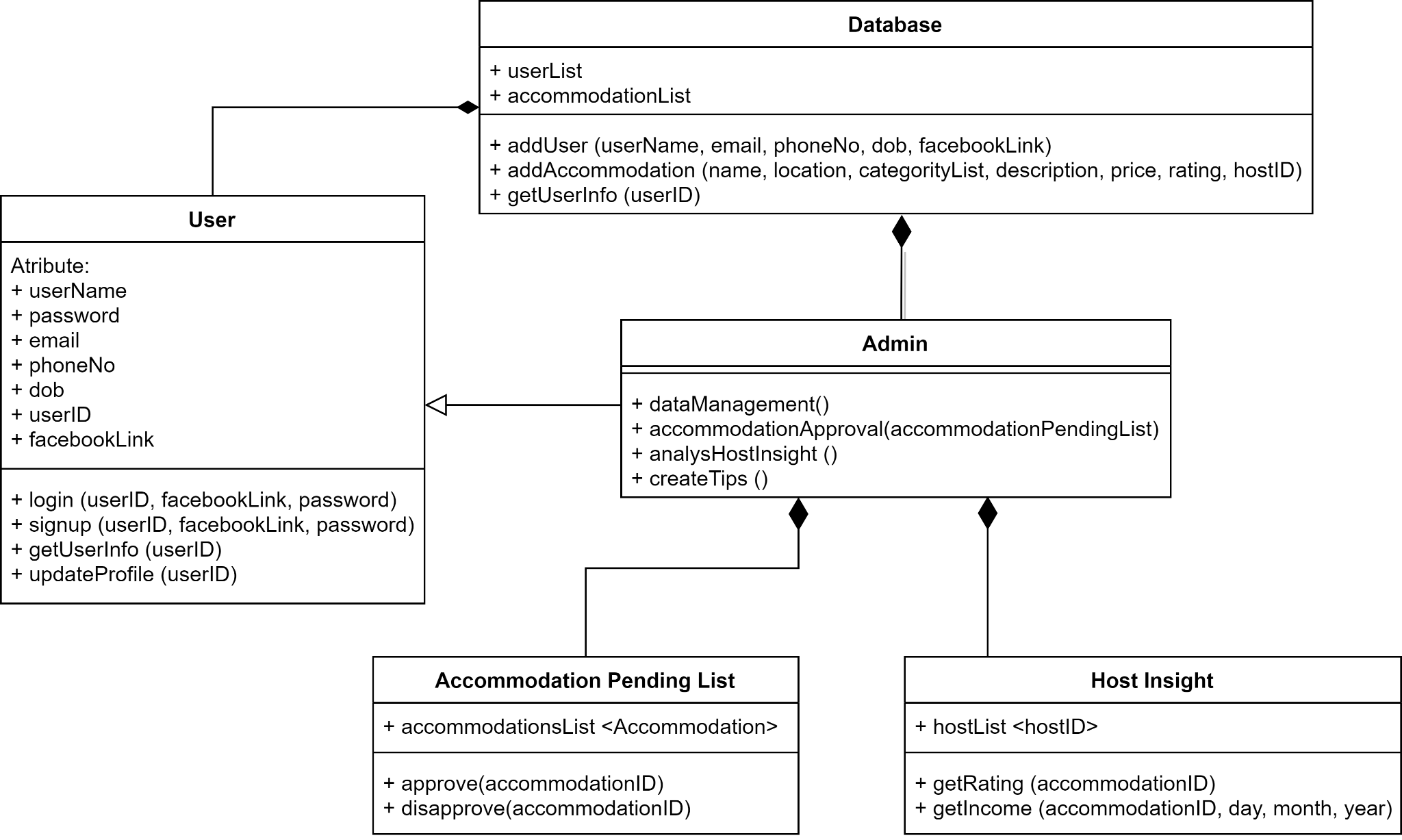
* *User* is used as an abstraction class for class *Renter*, it provides basic information for any user such as name, user id, phone number, etc as well as basic methods for users:
* *login*: To log in into the app
* *signup*: help user to register a new account for future login
* *getUserInfo*: return 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*.

### Component: Host

**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()*.

### Component: Admin:

**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.

# Deployment

# Implementation View