Bus Booking System

Problem Statement:

Our goal is to design a Bus Booking System where the users can book their buses for traveling from one location to another. The User can check bus availability, book tickets, cancel tickets and everything in real time.

Overview:

The general flow of our system is as follows:

- **Admin Flow:**
    1. Admin logins, dashboard shows all of his operating buses with options to edit them.
    2. Admin adds a new bus, includes details of bus plan, bus registration number, type, etc.
    3. After it a modal asking the number of available days, the startpoint and endpoint of the route, stops, fare, etc. is filled by the admin.
    4. This creates the trip entries for that many days in the Trip table with zero bookings initially.
    5. When the admin updates the days or seats plan for a bus the subsequent changes are reflected in the trips table.
- **Traveler Flow:**
    1. Traveler logins, reach on homepage with search bar.
    2. Searches for a trip from A to B for a date D.
    3. The search buses api is called, it first checks the cache, validates it and returns the results.
    4. The buses list is shown along with their timing, fares and seat availability status (green <60%, yellow < 90%, red >= 90% occupancy).
    5. User selects a bus, the detailed seat plan for the bus appears with booked seats highlighted.
    6. The user selects an unoccupied seat and moves forward with booking. The seat gets locked for a period of 5 min and the user should complete the payments and detail process in this time.
    7. If successful the seat is booked or else the lock gets released after 5 min and the user has to retry the booking process.

We have designed the system for inter city travel assuming certain things:

- We aim to give users a snappy experience trading off with a bit increase in memory.
- Also suppose the bus travels from A to C and B lies between them and the admin has added B as a stop then we will show this bus in search results of A to B and B to C as well, assuming bus stops there.

System

We will use a microservices architecture for our system to ensure that even if some of our services go down it would not affect the rest of the system.

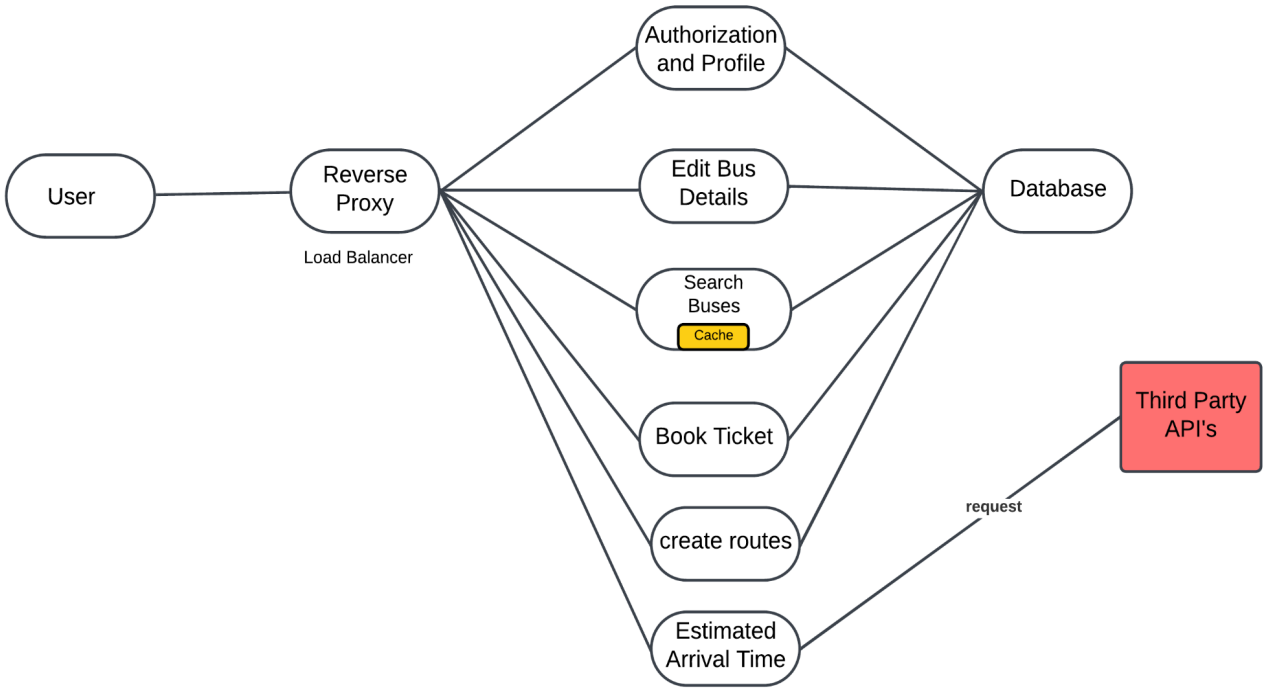


Fig. System Architecture

Services

Various microservices along with their functions are listed below:

- **Authorization and Profile**:
  Consists of all functionalities related to the register, login, reviewing and editing profile for both user and admin. Users can also see their booked tickets.

    - **Authorization:** We will be using the standard methods to authenticate:
        - JWT-Tokens for authentication and maintaining the session of the user.
        - OAuth for third party logins like google, facebook login etc.

    - **Profile:** Functions for editing and viewing my profile data.

- **Edit Bus Details**:
  Admin can schedule, reschedule trips of their buses. They can modify other bus details such as seat plans as well . This also includes adding new buses.

- **Create Routes:**
  Functions for creating new routes by admins. Routes include an array of stops.

- **Search Buses:**
  Users can search buses between two given points. This also features a cache as computing this is heavy and also the output doesn't change over a short time (until the bus starts on that route).

- **Book Tickets:**
  This container allows users to book tickets, cancel them.

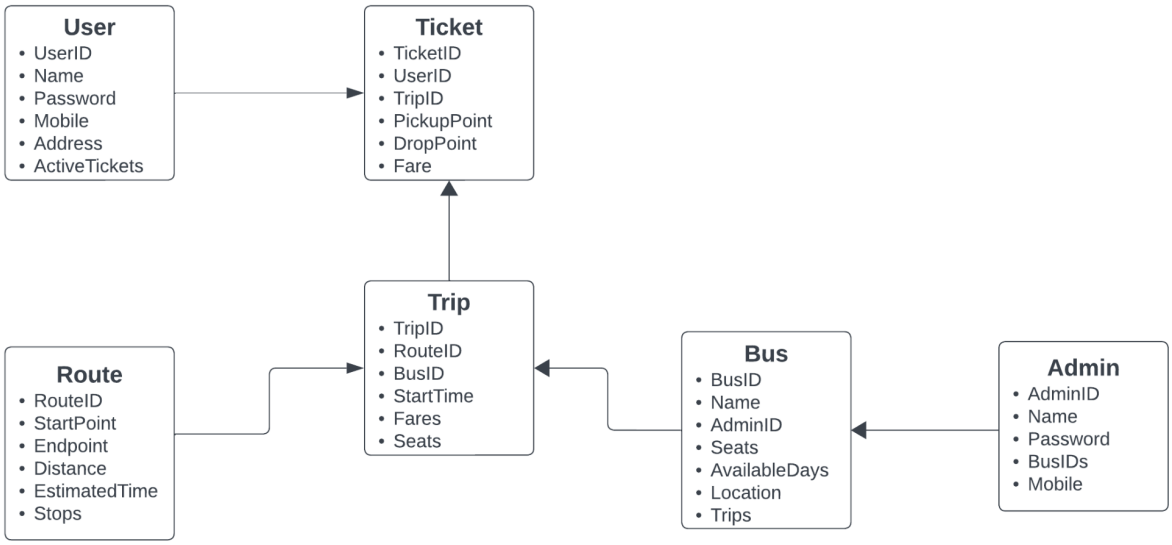
- **Bus Location:**
  Get the current location of the bus, the estimated arrival time, trip details etc.

Components:

- **User:** The client using our platform.
- **Services:** For efficiently handling the large volume of requests to our system and ensuring the robustness of the system we have divided the system into various microservices which run independent of each other (although some may interact with each other). Functionality of each is explained in the section below.
- **Reverse Proxy:** Along with different services there are also multiple instances of each microservice. The load balancer is used to distribute load among them.

Database

Following diagram shows the relations between various schemas and their attributes.



Schemas

- **User:**
  All user info is stored in this table, some minimal attributes are listed in the diagram above. UserID is a unique key and used to identify the user. The ActiveTickets is an array of TicketIDs which the user has booked and the journey is not completed yet. This leads to data redundancy (as the same data can be obtained by joins with the Ticket table), but since the number of active tickets for a user would be very small, we tradeoff memory for reducing the query time.

- **Ticket (TicketID, UserID, TripID, PickupPoint, DropPoint, Fare).**

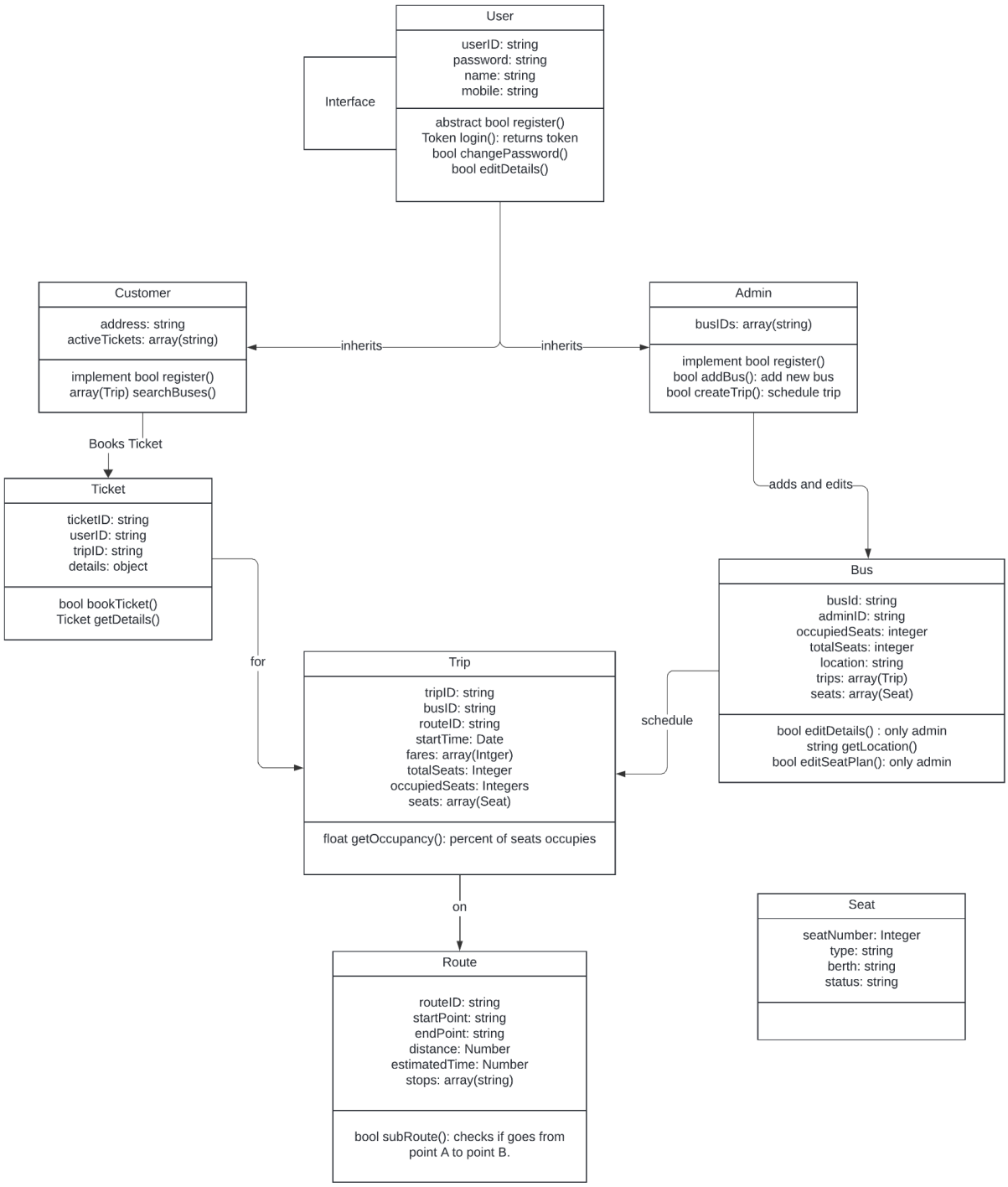
- **Route:**
  A startpoint, endpoint along with an array of stops (points where the bus halts) will make a route. Along with this we also store the distance of the route and estimated time required for the ride on the route.

- **Trip:**
  As soon as an admin creates or adds a bus the trips for the available number of days are created (if the available days is decreased the trips for that days are canceled) with the default startTime, fare and route provided. The admin can then edit the trip for any day he wants.

- **Bus:**
  The details of the bus will be stored which can be modified by the admin. Again we have maintained an array trip which stores the TripIDs of all the trips scheduled for that bus. This is redundant but again we have traded a small increase in memory to significantly faster queries. Location will store the current location of the Bus.

- **Admin:**
  The details of the admin along with an array of BusIDs of which he is the admin.

Low-level Design



Searching Buses

In the routes table we have routes with all stops. While searching for the buses from point A to point B we use the following approach:

1. Routes array contains all routes fetched from the database.
2. Iterate on all the routes and for every route
    a. Check if both the points A and B are there in the stops.
    b. Point A is earlier than point B.
3. If both the conditions are satisfied then add it to selectedRoutes.
   Till this step will be required only once in a day or two as new routes are not added frequently.
4. Then iterate over the trips and any trip on that route should be shown on as the search result.

The above algorithm takes significant time to execute and also the results for buses from Point A to point B would not change until some buses leave their start point. So we will use caching to store the results in **Cache**. From then on, if a similar query comes, just remove trips whose startTime is earlier to current time and iterate for only new trips added. You already have the selected routes.

Getting Estimated Arrival Time

We have the current location of any bus, we will use any third party api to get the estimated time for the bus to arrive at a particular location. Using third party API's will save us resources and give more accurate results as they will take into account real time traffic, weather, traffic signals, etc.

Concurrency Control

While the user is booking a particular seat the seat is locked for a period of 5 min to ensure that there is no race condition among users to book the same seat. In this time period no other user can select and book this ticket, while the one booking has to complete the procedure of adding details and payment. If a timeout occurs the transaction is rolled back, changes made reset and user, or any other user is free to book that ticket.

Robustness and Optimization of the System

To ensure that our system is robust and immune to system failures, we have used the microservices architecture, along with this we have multiple instances of each container so as to ensure redundancy in case of failures and for load balancing.

The most instances should be made of the authorization and Profile handling container, after which of the book ticket containers. This will receive a large amount of traffic and can efficiently operate independently. The search buses container should be used location specifically so the cache is used efficiently and less amount of redundant work is needed.