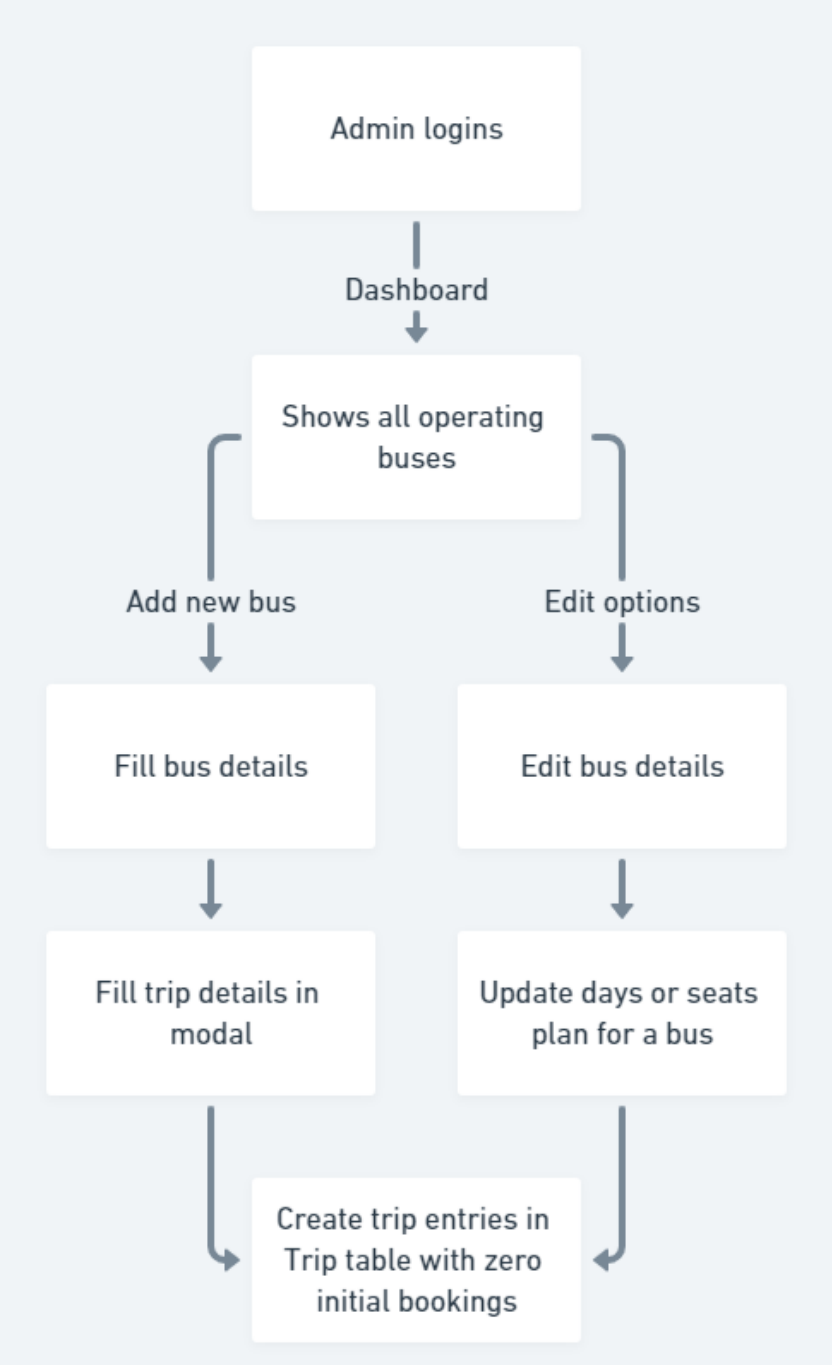
Bus Booking System

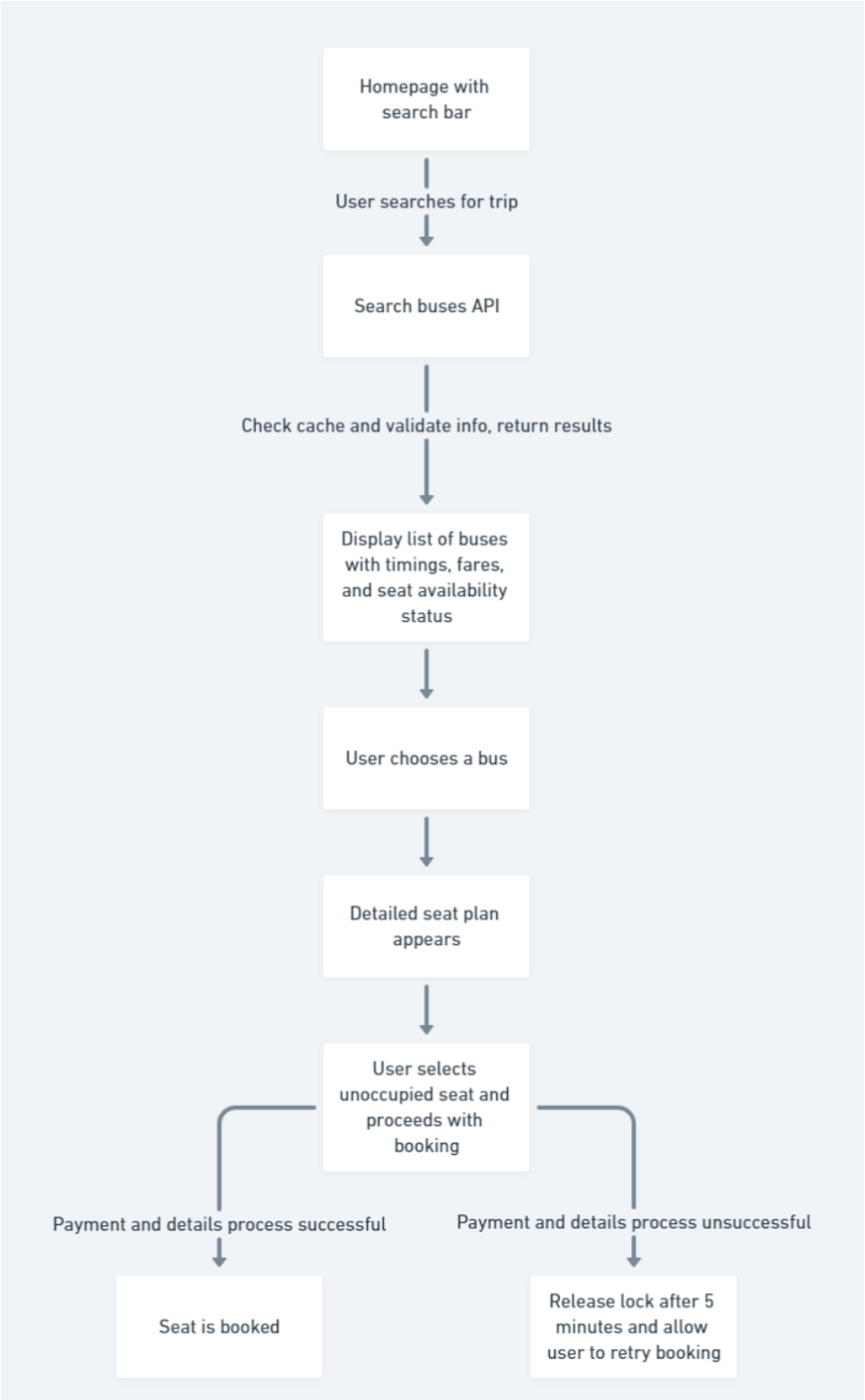
Problem Statement:

Our aim is to establish an effective Bus Booking System that enables users to effortlessly secure bus reservations for their trips between diverse locations. Users will have the ability to verify bus availability, make ticket reservations, cancel bookings, and carry out these activities in real-time.

Admin Flow:



Traveler Flow:



- **System**

The bus booking system will be a centralized database, distributed server architecture so that we ensure consistency along with good performance.

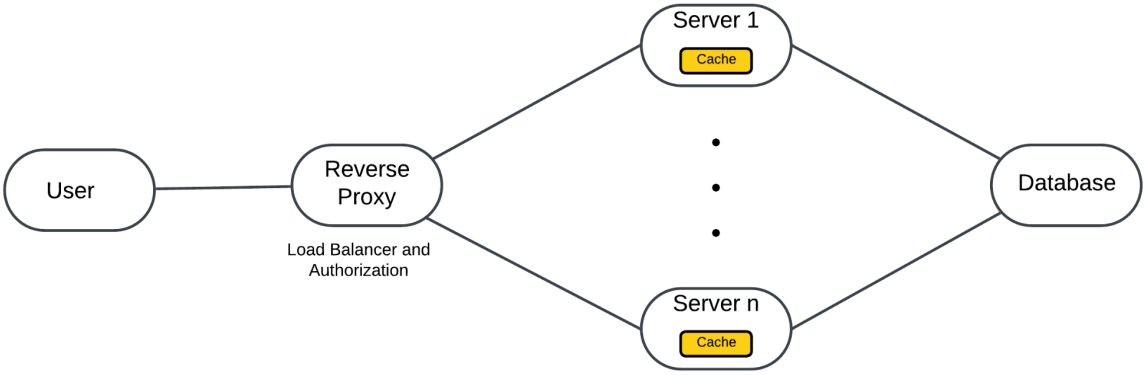


Fig. System Architecture

Components:

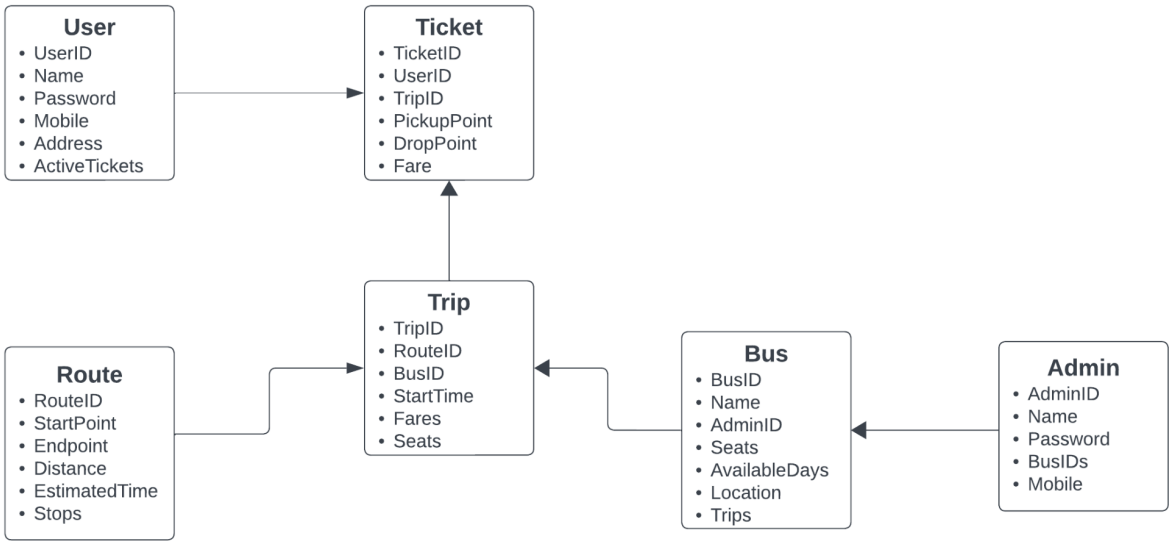
- **User:** The client using our platform.
- **Servers:** For efficiently handling the large volume of requests to our system, we implement a distributed server approach. There is a local cache.
- **Reverse Proxy:**

  For load balancing we have a reverse proxy whose function is to balance load among servers and forward the request to the server with minimum load. The request forwarding in done on two parameters:

  - The loads on the servers at the moment
  - The location of the request (to maintain spatial locality).

- **Database**

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



Various Functionalities:

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.

Low level Implementation:

We implement a Object oriented approach to achieve the functionality for promoting reusability of code and improve code structure and understanding.

- User Interface:
    - Attributes:
        - userID - string
        - Password - string
        - Name - string
        - Mobile - string
    - Methods:
        - abstract register()
        - login() - validates credentials and returns a token
        - changePasswords() - returns success or failure
        - editDetails() - returns success or failure
- Customer Class (**inherited** from User Interface):
    - Attributes:
        - Address: string
        - CompletedTrips: array(ticketIds)
        - Tickets: array(ticketIds)
    - Methods:
        - getPastTrips() - returns past trips
        - Implement register()
        - searchBuses() - returns buses nearest to you for given route.
- Admin Class (**inherited** from User Interface):
    - Attributes:
        - busIds - array(BusID)
    - Methods:
        - createTrip() - create a new trip
        - Implement register()
        - addBus() - add a new Bus.
- Ticket Class:
    - Attributes:

        - TicketID: string
        - userID: UserID
        - tripID: TripID
        - Details: object
    - Methods:
        - getDetails() - returns ticket details
        - bookTicket() - book a ticket
- **Trip Class:**
    - Attributes:
        - tripID: TripID
        - Details: object
        - routeID: RouteID
        - startTime
        - Fares
        - totalSeats
        - occupiedSeats
        - Seats
    - Methods:
        - getOccupancy(): returns the percent of occupied seats.
- **Route Class:**
    - Attributes:
        - routeID
        - Startpoint
        - Endpoint
        - Distance
        - estimatedTime
        - Stops
    - Methods:
        - subRoute() - checks whether the points lie on the route
- Bus Class:
    - Attributes
        - busID
        - adminID
        - Location
        - Trips
        - Seats
    - Methods:
        - editDetails()
        - getLocation()
        - editSeatPlan()

- Seat Class:
    - Attributes:
        - seatNumber
        - Type
        - Berth
        - capacity
        - Status

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 robustness we have used multiple replicas of the server, so in cases where a certain server instance fails we immediately redirect the traffic to other servers, this ensures that the system as overall never goes down.

We have implemented the caching mechanism and designed the database to optimize the memory usage by minimizing the redundancy of data stored.