Comet Commuter Carpool

Aman Balam, Vincent Jones, Neal Kapadia, Shivani Kumar, Alan Edward Roybal, Aarush Shintre, Andy Weng

Objectives

01

Create Carpools

02

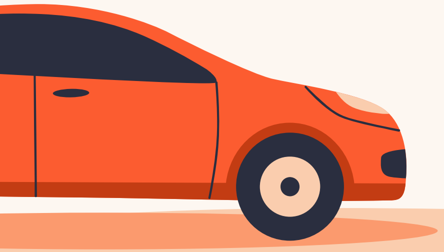
Cut Costs

03

Reduce Stress

04

Connect Commuters



Project Timeline

Project Proposal	Nov 3 - 7, 2025
Requirements Engineering	Nov 10 – 14, 2025
System Design	Nov 17 – 28, 2025
Implementation and Unit Testing	Dec 1 –12, 2025
System Testing	Dec 15 – 19, 2025
Present to Stakeholders	Dec 29 – 31, 2025
Deploy to Servers	Dec 31, 2025 – Jan 7, 2026

Start Date: Nov 3, 2025
End Date: Jan 7, 2026

Requirements

Functional

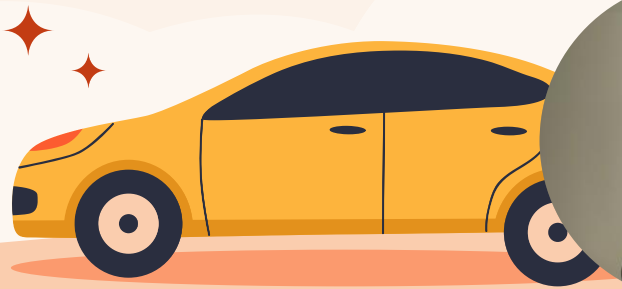
- Secure user login via email/password or phone OTP.
- Match users by location and schedule overlap.
- Create/edit commute profiles with schedules and routes.
- Calculate and display ride costs; support digital payments.

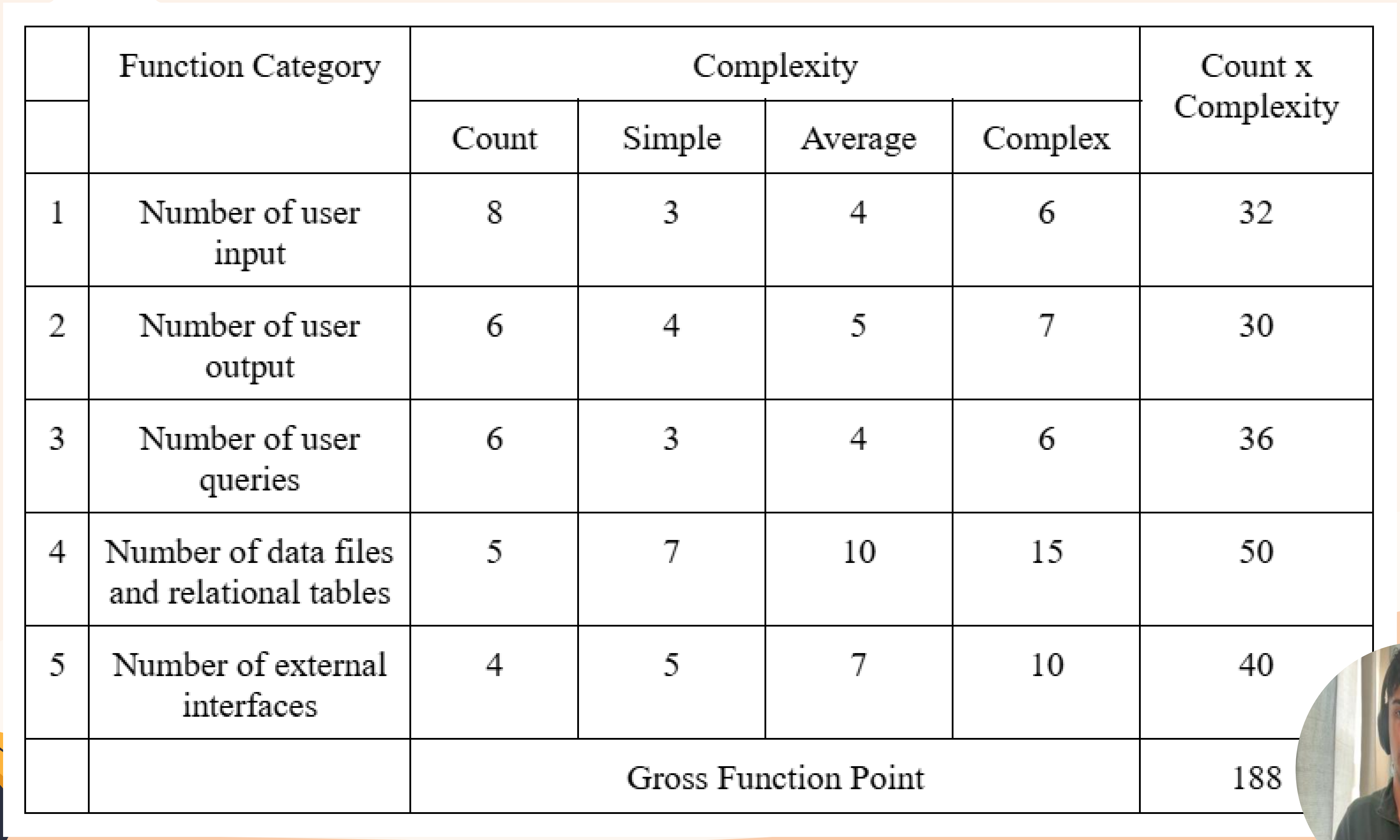
Non-Functional

- Security: Encryption of personal/session data; authorized access only.
- Safety: Provide user tools for reporting/blocking; verify student status.
- Performance: Core actions respond within 3 seconds.
- Usability: Key features accessible within five clicks.

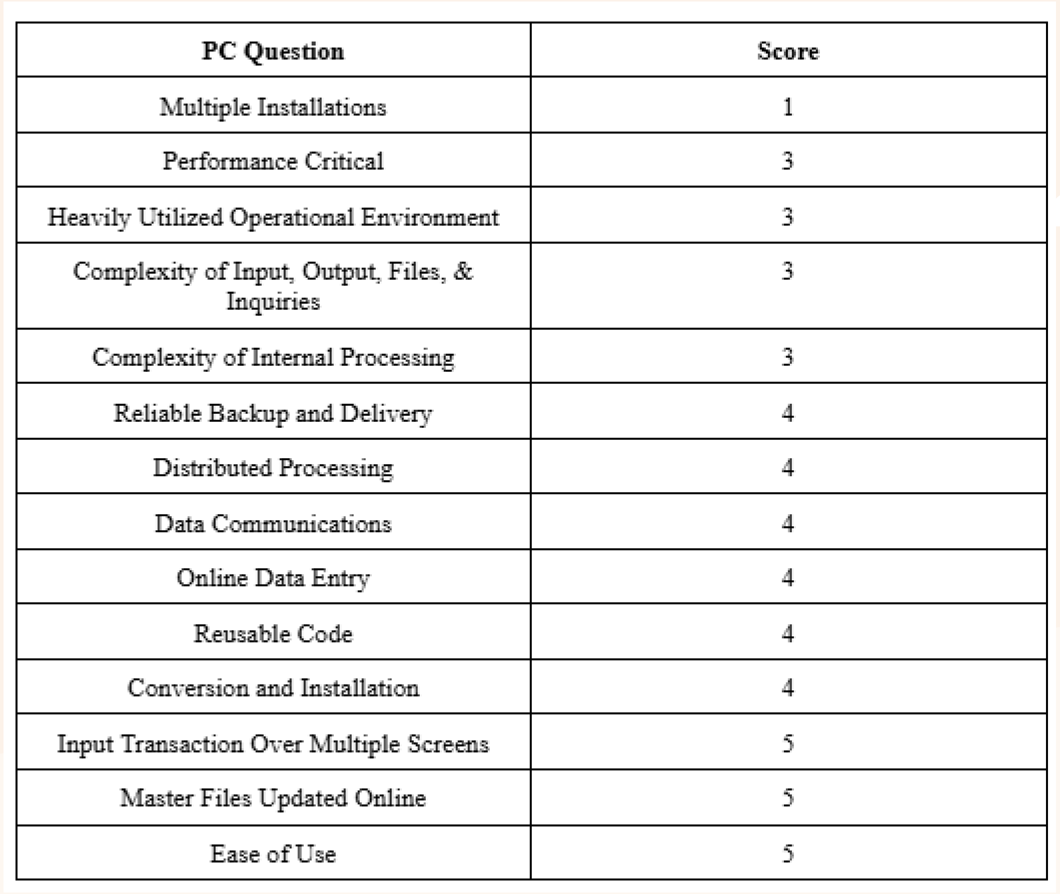
Cost, Effort, & Pricing

We used the function point technique to model the cost of our project.



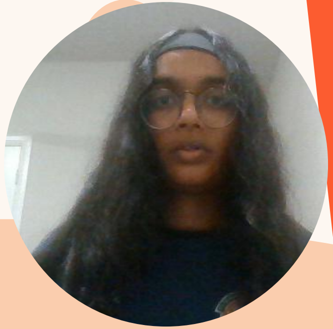


	Function Category	Complexity				Count x Complexity
		Count	Simple	Average	Complex	
1	Number of user input	8	3	4	6	32
2	Number of user output	6	4	5	7	30
3	Number of user queries	6	3	4	6	36
4	Number of data files and relational tables	5	7	10	15	50
5	Number of external interfaces	4	5	7	10	40
		Gross Function Point				188

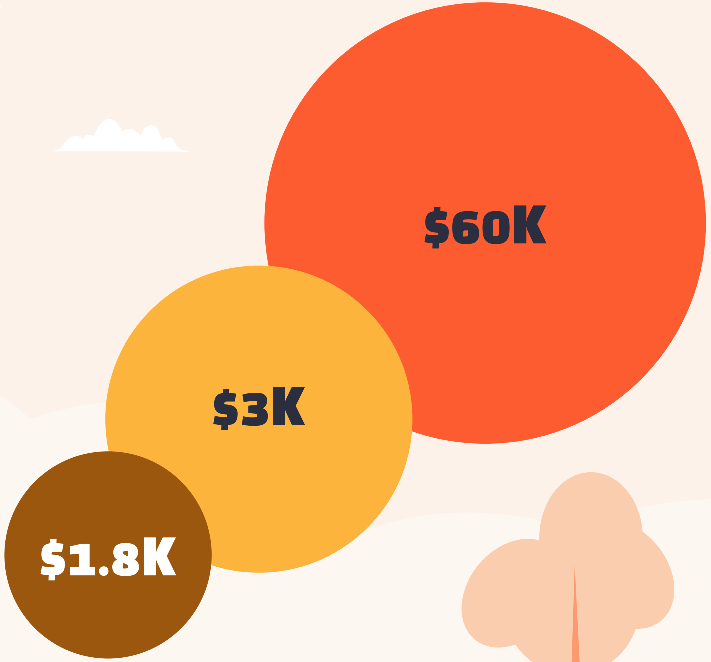


PC Question	Score
Multiple Installations	1
Performance Critical	3
Heavily Utilized Operational Environment	3
Complexity of Input, Output, Files, & Inquiries	3
Complexity of Internal Processing	3
Reliable Backup and Delivery	4
Distributed Processing	4
Data Communications	4
Online Data Entry	4
Reusable Code	4
Conversion and Installation	4
Input Transaction Over Multiple Screens	5
Master Files Updated Online	5
Ease of Use	5

8 weeks



Cost, Effort, & Pricing Estimate



Personnel

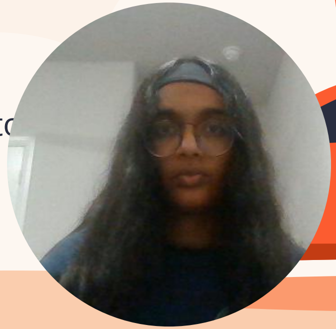
7 people working for 8 weeks earning 1k a week, and 4k for training after deployment

Hardware

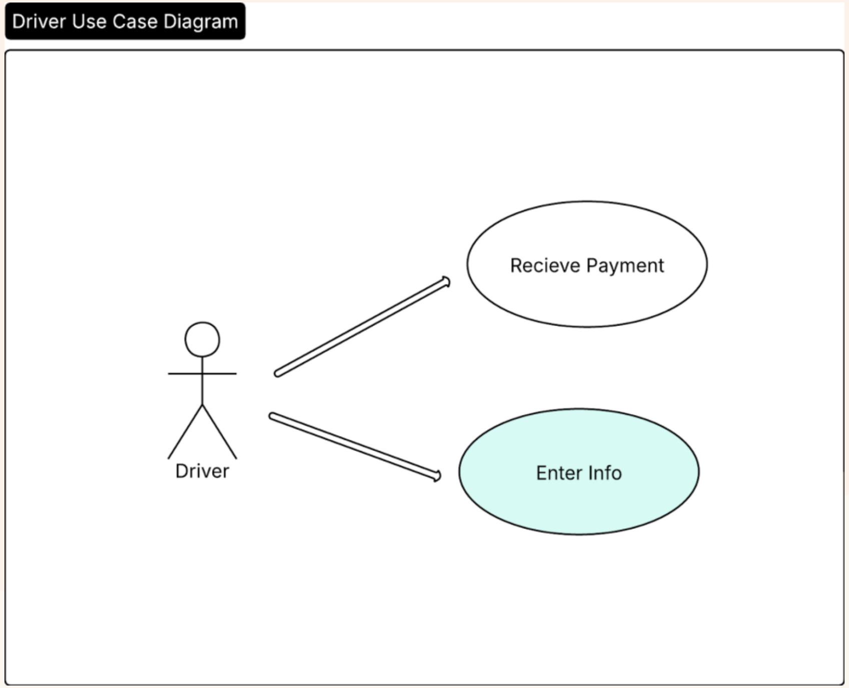
Server, Database, Backup

Software

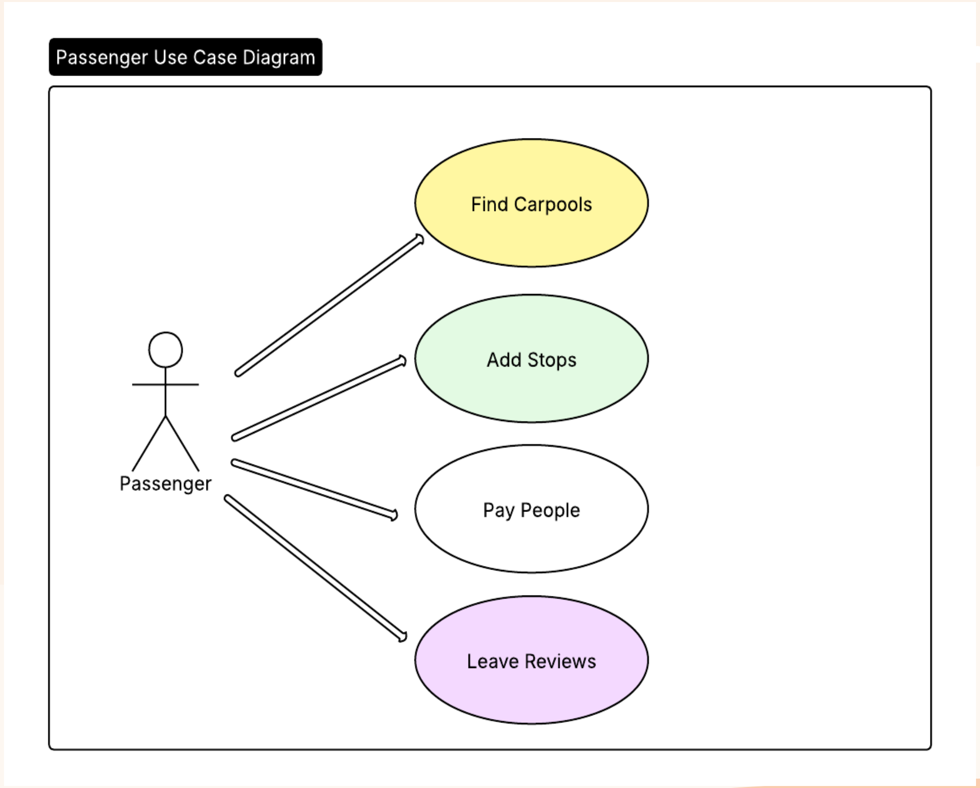
OS/DB License, Dev Tools, Monit
Tools, Deployment Tools



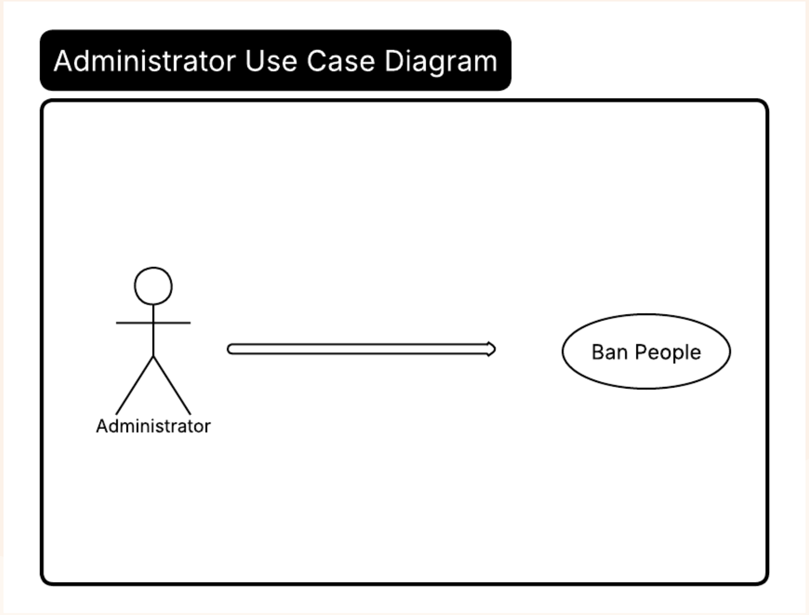
Driver Use Case Diagram



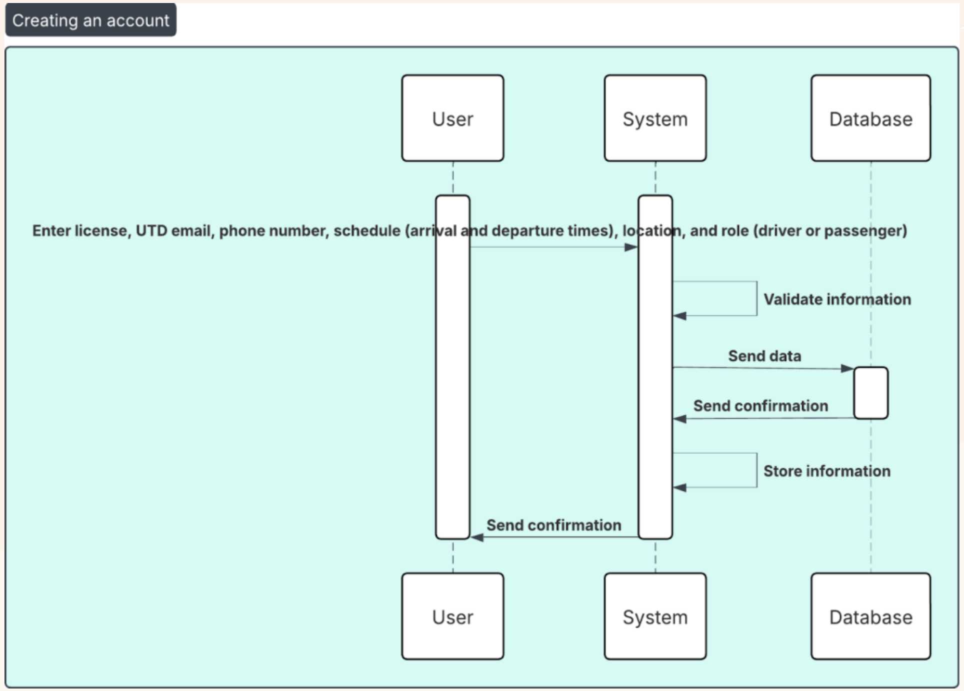
Passenger Use Case Diagram



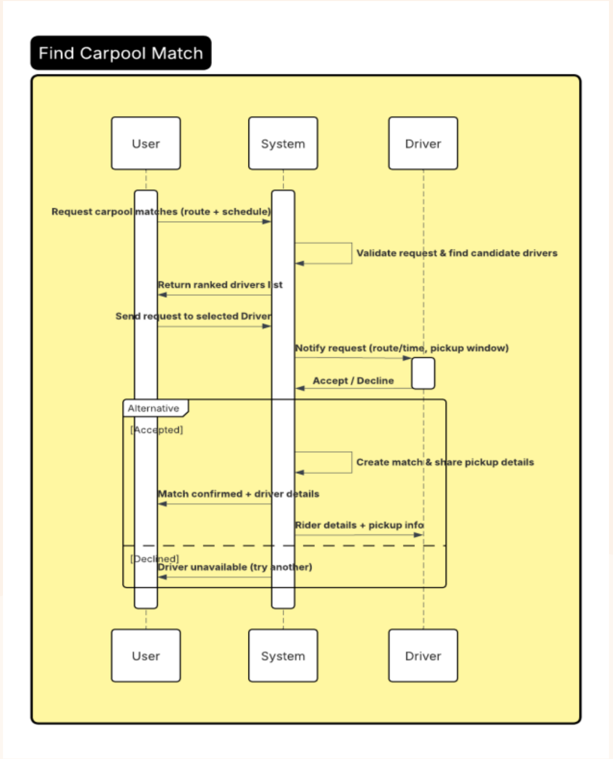
Administrator Use Case Diagram



Sequence Diagram

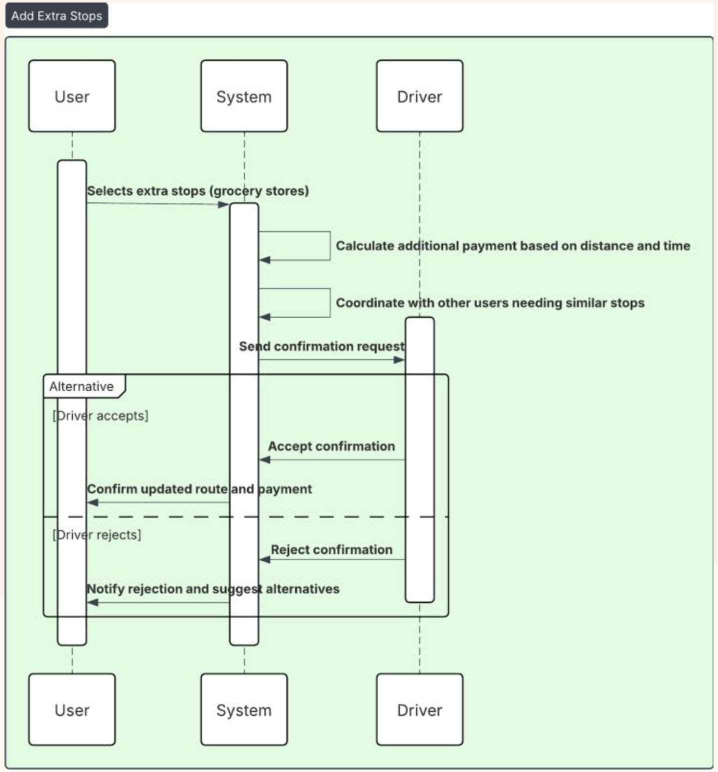


Sequence Diagram

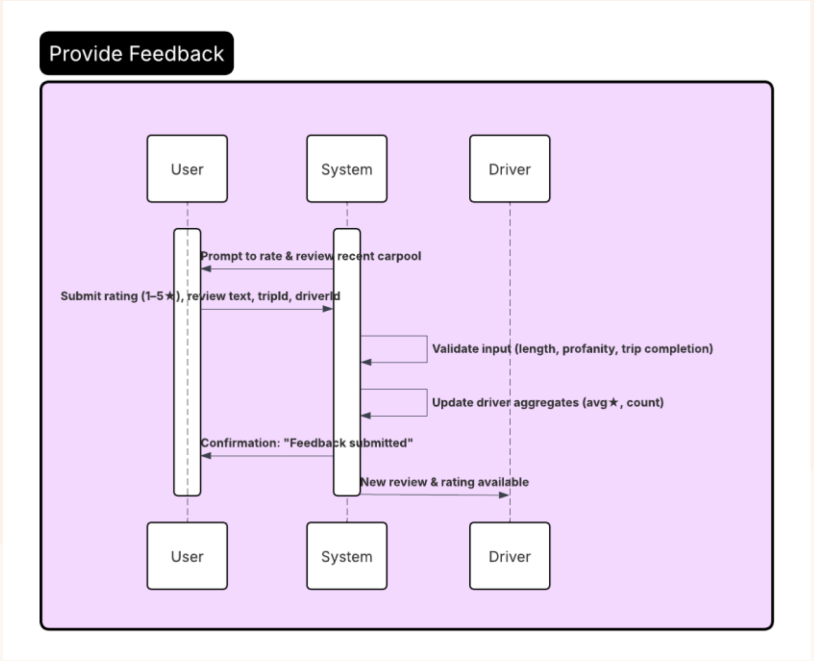


Find Carpool Match
User
System
Driver
Request carpool matches (route + schedule)
Validate request & find candidate drivers
Return ranked drivers list
Send request to selected Driver
Notify request (route/time, pickup window)
Accept / Decline
Alternative
[Accepted]
Create match & share pickup details
Match confirmed + driver details
Rider details + pickup info
[Declined]
Driver unavailable (try another)
User
System
Driver

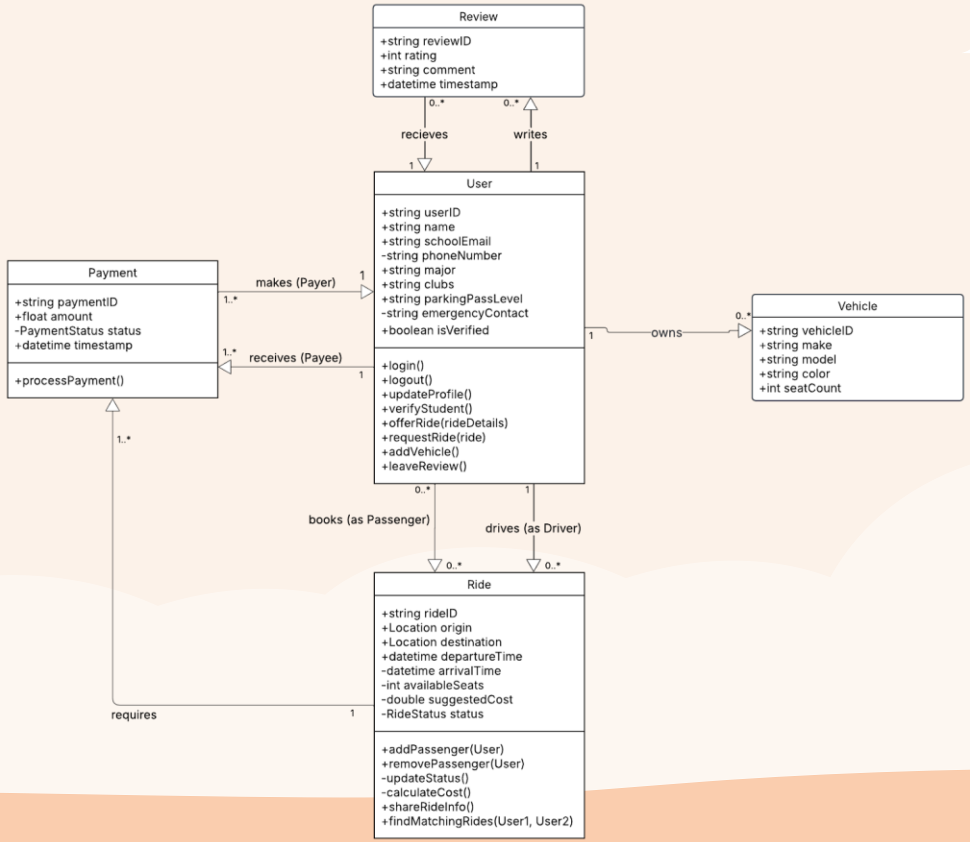
Sequence Diagram

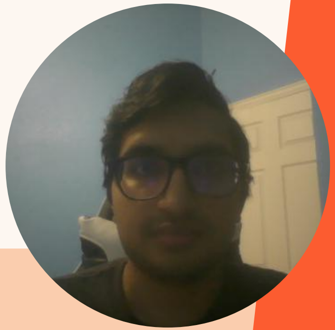

Add Extra Stops
User
System
Driver
Selects extra stops (grocery stores)
Calculate additional payment based on distance and time
Coordinate with other users needing similar stops
Send confirmation request
Alternative
[Driver accepts]
Accept confirmation
Confirm updated route and payment
[Driver rejects]
Reject confirmation
Notify rejection and suggest alternatives
User
System
Driver

Sequence Diagram

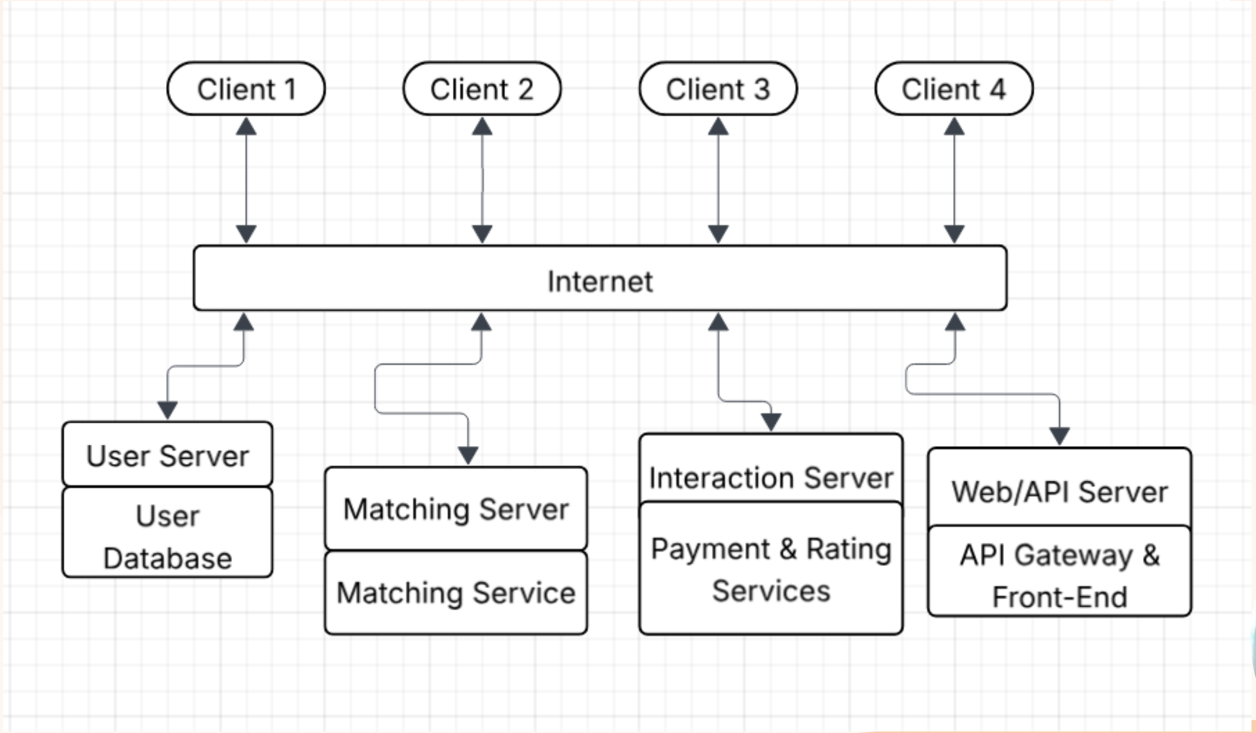

Provide Feedback
User
System
Driver
Prompt to rate & review recent carpool
Submit rating (1–5★), review text, tripId, driverId
Validate input (length, profanity, trip completion)
Update driver aggregates (avg★, count)
Confirmation: "Feedback submitted"
New review & rating available
User
System
Driver

Class Diagram





Architectural Diagram – Client-server



Test Plan

- Ensure the platform works **reliably**, **securely**, and **as intended** before expansion.
- Validate all **core user flows**: profile creation, matching, payments, and issue reporting.
- Reduce the risk of **service disruption**, **incorrect recommendations**, or **payment errors**.
- Build confidence in the platform’s **scalability** and **data integrity**.

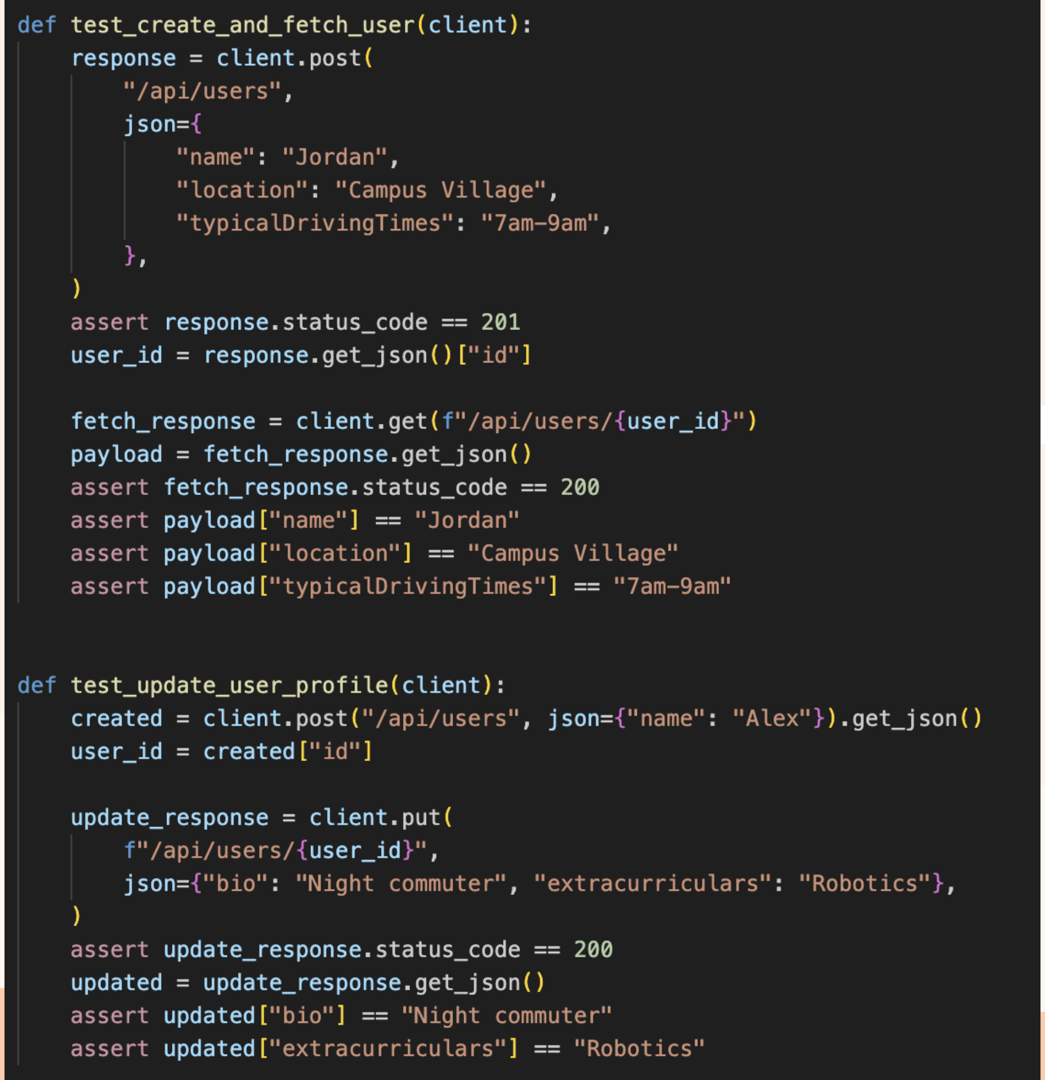
https://github.com/AarushShintre/3354-Team8/blob/main/backend/tests/test_ap

What We Test

- **User Account Management**
  - Creating users
  - Fetching user profiles
  - Updating user details
- **Matching & Recommendation Engine**
  - Compatibility scoring
  - Recommendation ordering
  - Handling differing user attributes
- **Payment Suggestion System**
  - Calculating suggested contribution
  - Consistent output between algorithm and API
- **Issue Reporting System**
  - Submitting issues
  - Retrieving issues
  - Data persistence across flows
- **Core Algorithms**
  - Compatibility scoring logic
  - Matching conditions
  - Overlap reward system

How We Test

- **User API Tests**
  - Validate 201 response on creation
  - Verify response body contains correct fields
  - Check correct retrieval of user profiles
  - Confirm updates persist properly
- **Recommendation Tests**
  - Confirm best match is ranked first
  - Ensure score ordering is correct
  - Validate data integrity between stored users and returned matches
- **Payment Tests**
  - Check correctness of price-suggestion formula
  - Ensure API returns consistent results with backend function
- **Issue Reporting Tests**
  - Validate issue creation
  - Ensure issues appear in feed
  - Confirm IDs are unique and persistent
- **Algorithm Unit Tests**
  - Confirm compatibility scores > 0 when attributes overlap
  - Validate logic for location, schedule, major, and extracurriculars

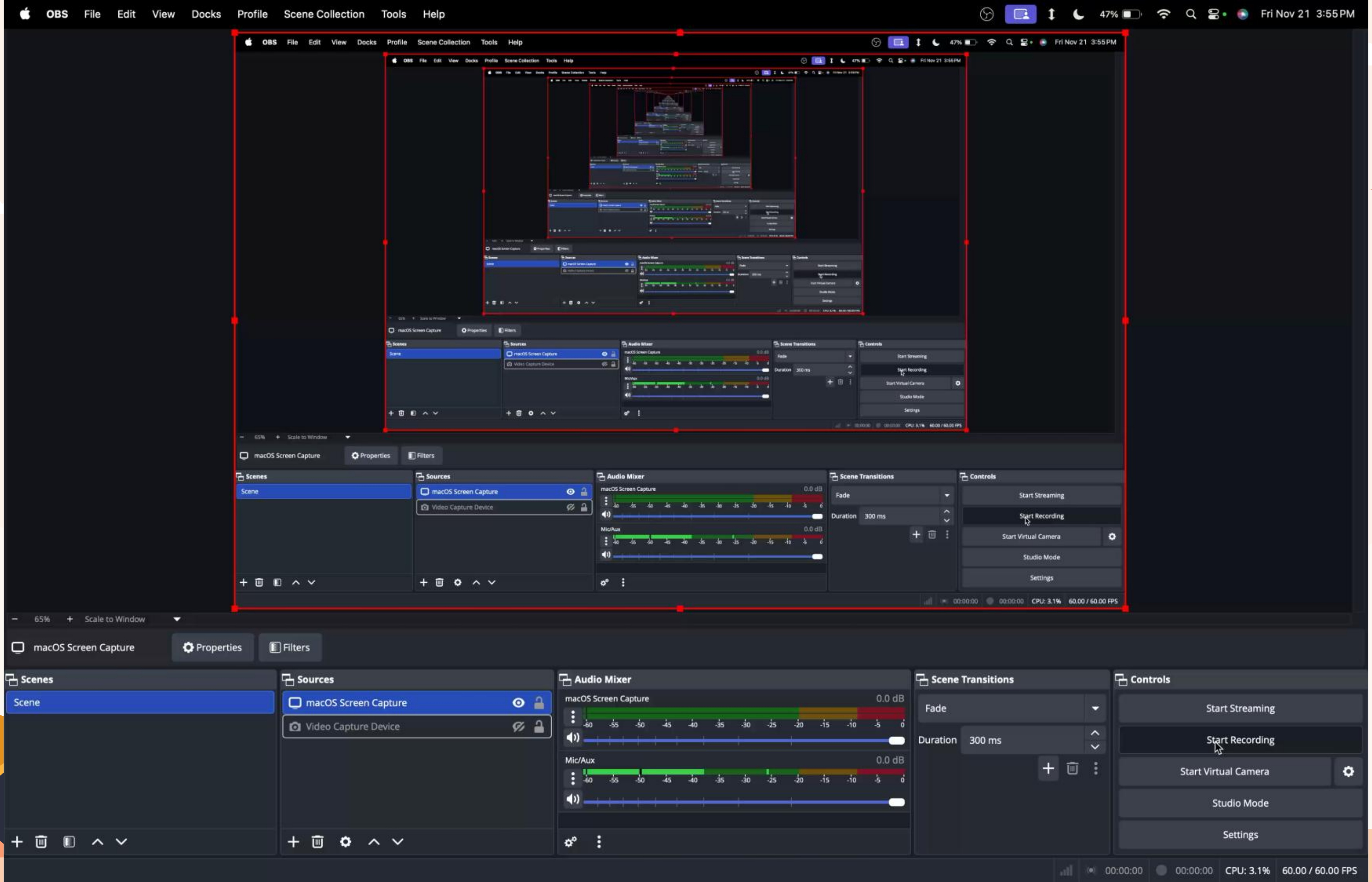


```python
def test_create_and_fetch_user(client):
    response = client.post(
        "/api/users",
        json={
            "name": "Jordan",
            "location": "Campus Village",
            "typicalDrivingTimes": "7am–9am",
        },
    )
    assert response.status_code == 201
    user_id = response.get_json()["id"]

    fetch_response = client.get(f"/api/users/{user_id}")
    payload = fetch_response.get_json()
    assert fetch_response.status_code == 200
    assert payload["name"] == "Jordan"
    assert payload["location"] == "Campus Village"
    assert payload["typicalDrivingTimes"] == "7am–9am"


def test_update_user_profile(client):
    created = client.post("/api/users", json={"name": "Alex"}).get_json()
    user_id = created["id"]

    update_response = client.put(
        f"/api/users/{user_id}",
        json={"bio": "Night commuter", "extracurriculars": "Robotics"},
    )
    assert update_response.status_code == 200
    updated = update_response.get_json()
    assert updated["bio"] == "Night commuter"
    assert updated["extracurriculars"] == "Robotics"
```



65% Scale to Window

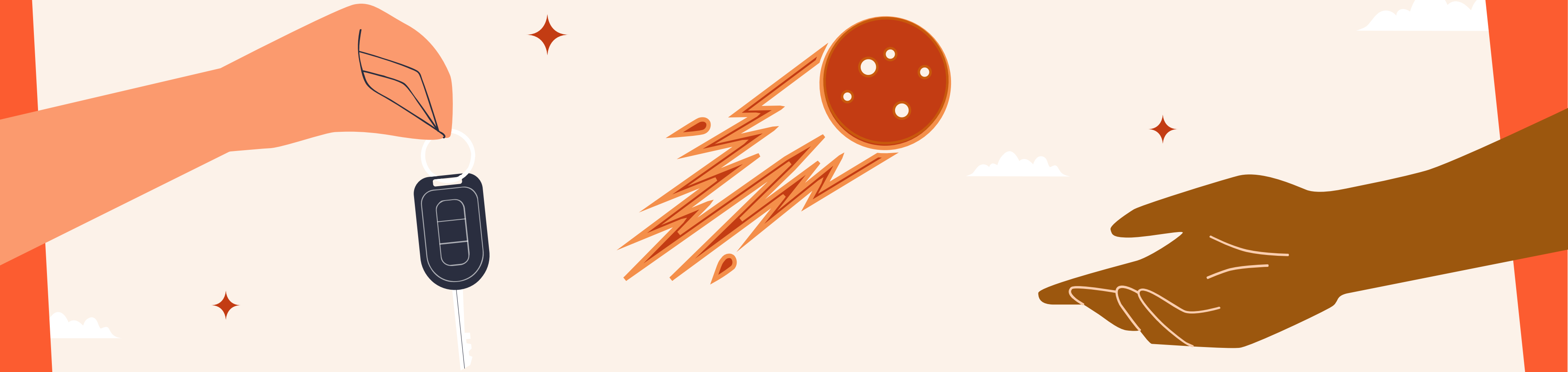
macOS Screen Capture 0.0 dB

Mic/Aux 0.0 dB

Fade

Duration 300 ms

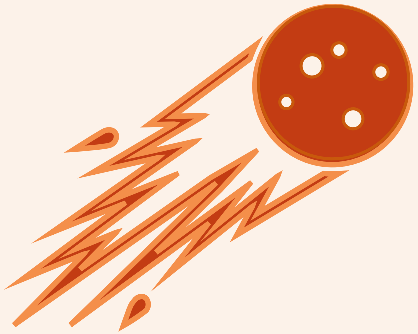
00:00:00 00:00:00 CPU: 3.1% 60.00 / 60.00 FPS



Prototype

https://github.com/AarushShintre/3354-Team8/tree/main

Thanks!



CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**