UCS503 SOFTWARE ENGINEERING PROJECT REPORT

CampusRide — A Smart, Secure Campus Carpooling App

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Group: 3C43

Submitted To: Dr. Mahak Gambhir

Submitted By:

Avani – 102303587
Ananya – 102303594
Priyanshu – 102303595
Yash – 102303590

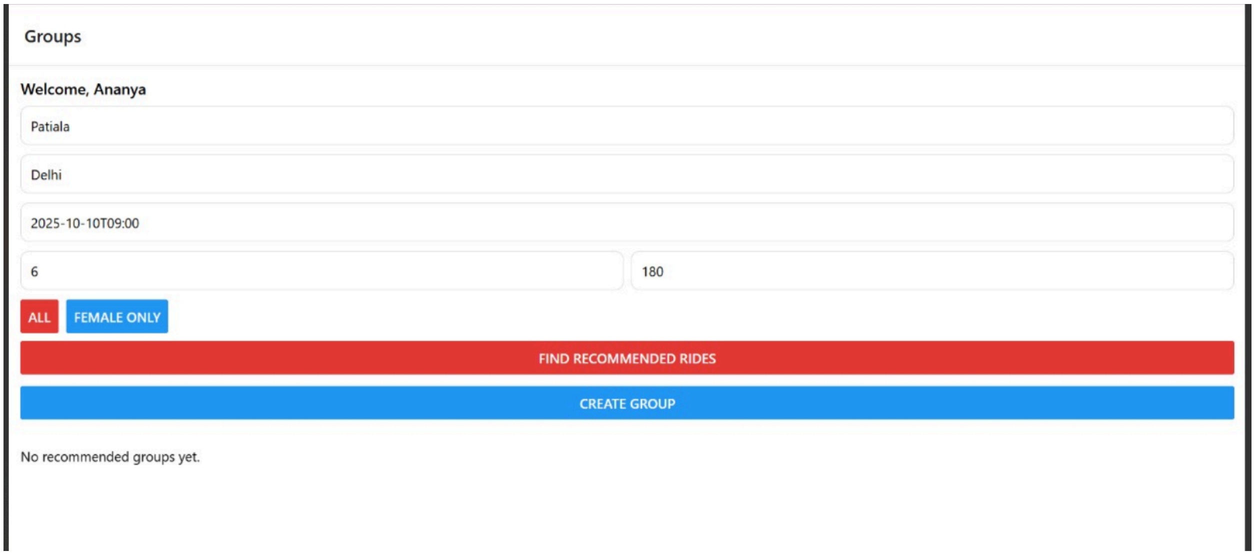
Table of Contents

1. Project Overview

Project Title: CampusRide – Smart Campus Carpooling App
Objective: To make campus travel safer, cheaper, and more convenient through secure, verified carpooling among Thapar University students.
Short Description: CampusRide is a campus-exclusive rideshare and carpooling platform that groups students traveling on similar routes and times into shared trips. It enforces authenticity, privacy, and trust through OTP-verified @thapar.edu login and a Circle of Trust contact-mining feature that marks 1°, 2°, and 3° connections for safer matching.

Scope:

- Students: Create, join, and coordinate verified ride groups with known or connected peers via Circle of Trust.
- Admin: Monitor system health, moderate complaints, view audit logs, and manage onboarding policies.



Groups

Welcome, Ananya

Patiala

Delhi

2025-10-10T09:00

6

180

No recommended groups yet.

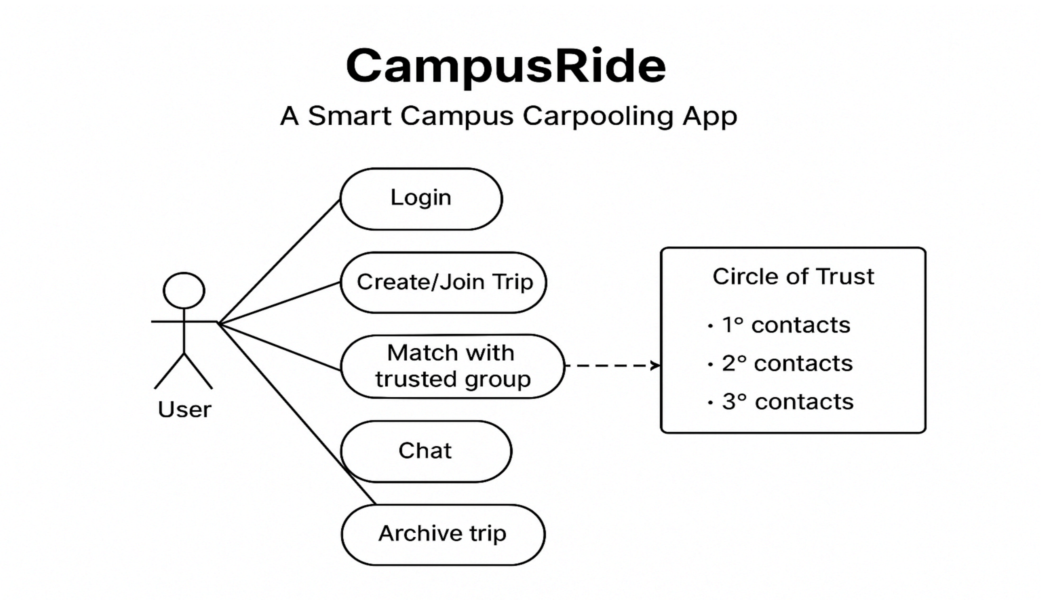
- Optional future extensions: integration with campus taxis, driver network, and payments.

2. Analysis Phase

2.1 Use Case Diagram & Templates

Overview: The Use Case set highlights primary actors and goals: Student (primary actor), Admin (secondary actor). Primary use cases include Login (OTP verified), Create Trip, Search/Join Trip, Coordinate via In-App Chat, Complete & Archive Trip, and Manage Profile. The Circle of Trust (1°, 2°, 3° contacts) is integrated into Trip Matching — not as a separate use case.

Use Case: Create / Join Trip (including Circle of Trust)



Actor: Student
Precondition: Student is registered and OTP-verified with @thapar.edu
Trigger: Student opens the app and selects Create Trip or searches for existing trips
Main Flow:

1. Student enters source, destination, date, time, and preferences.
2. System parses route and normalizes stops (Route Parser).
3. Match Engine runs matching algorithm; Contact Mining component computes 1°, 2°, 3° trust scores for potential matches.
4. System presents recommended groups prioritized by trust level, route overlap, and time proximity.
5. Student selects a group to join or creates a new group; in-app chat activates when group forms.

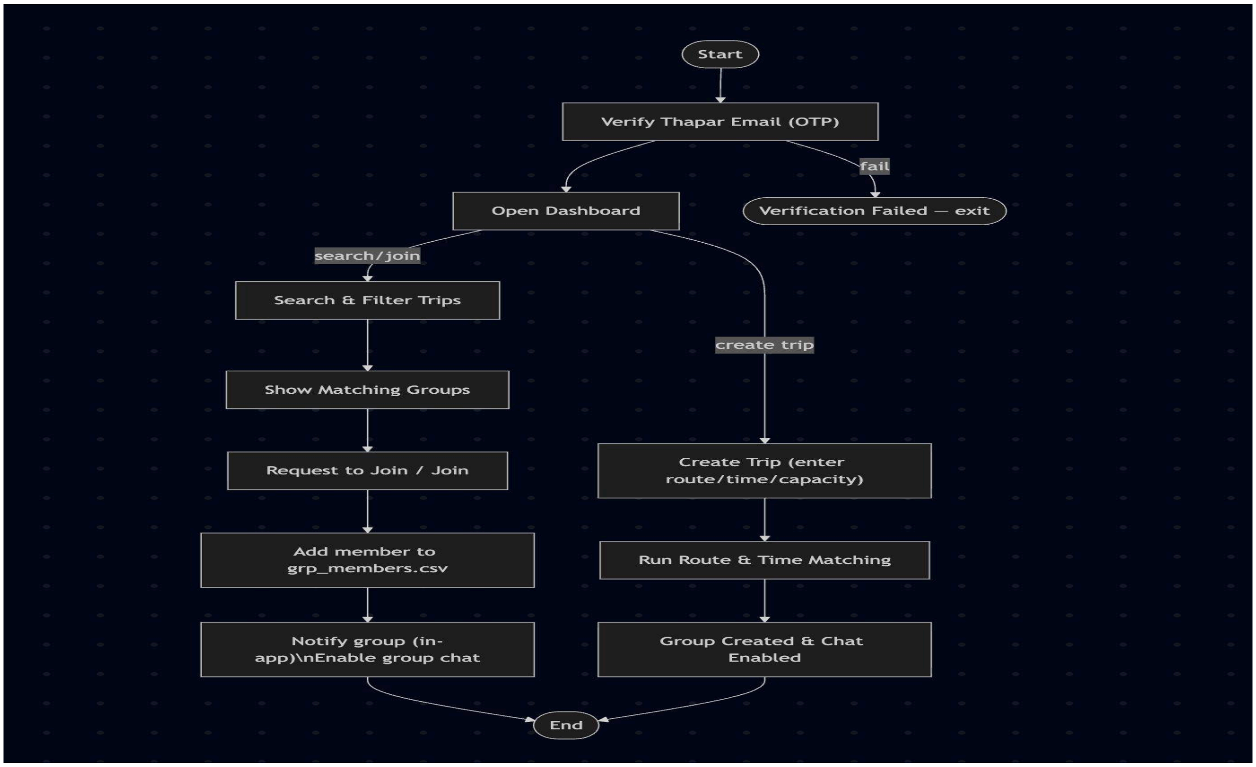
Alternate Flows:

- If no matches found, system suggests to create a new trip and optionally share a secure join link.

Postcondition: Student is a member of a trip group; chat is enabled until trip archival.

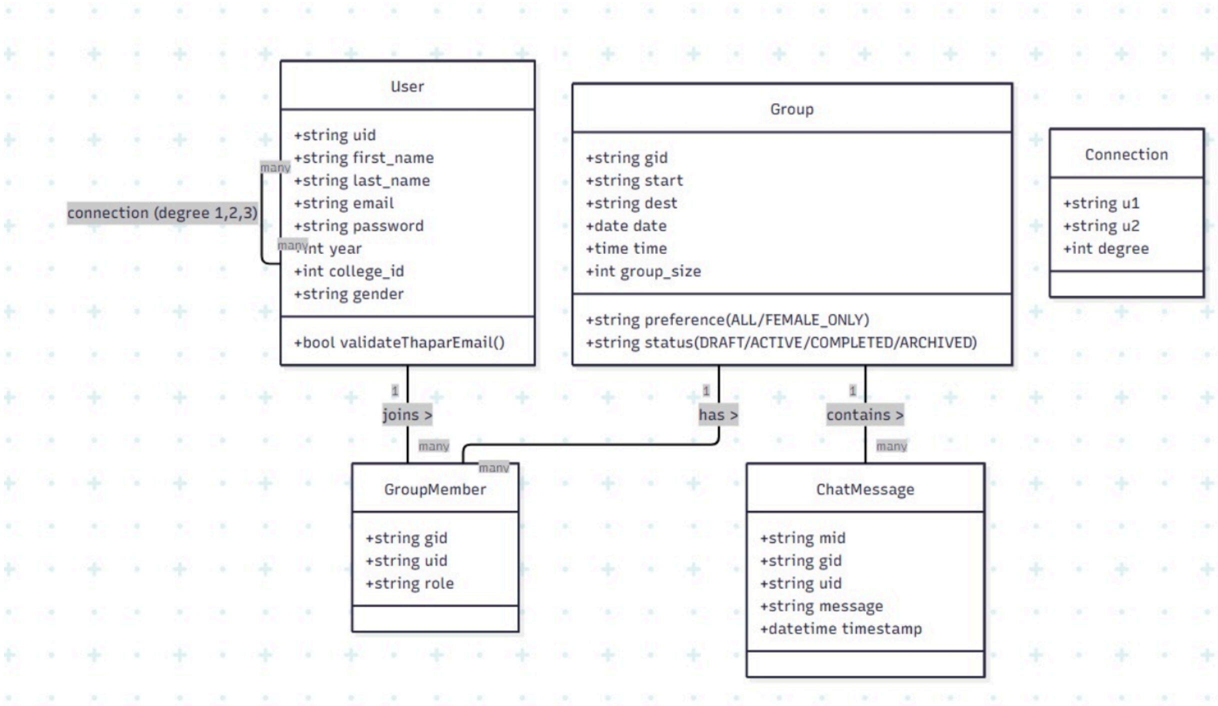
2.2 Activity Diagram

The activity flow below captures end-to-end user actions from Login to Trip Archival. For a visual diagram, use the Mermaid code in Appendix (Section 8).


Start
Verify Thapar Email (OTP)
fail
Open Dashboard
Verification Failed — exit
search/join
Search & Filter Trips
create trip
Show Matching Groups
Request to Join / Join
Create Trip (enter route/time/capacity)
Add member to grp_members.csv
Run Route & Time Matching
Notify group (in-app)\nEnable group chat
Group Created & Chat Enabled
End

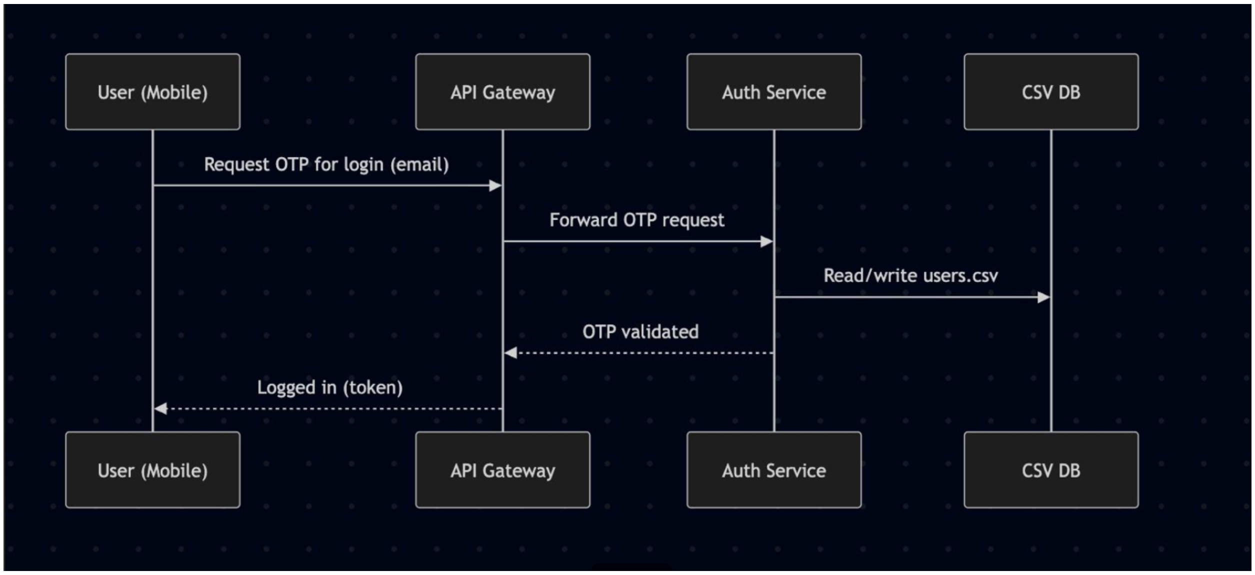
2.3 Class Diagram (Core Entities)

Core classes and attributes (brief):
- User: userId, name, thaparEmail, gender, phoneHash, trustEdges
- Trip: tripId, creatorId, source, destination, stops[], dateTime, seatsAvailable, preferences
- Group: groupId, tripId, members[], status
- MatchEngine: matchId, algorithmVersion, routeIndex
- ContactGraph: adjacencyList, trustScores (1°,2°,3°)
- ChatMessage: messageId, senderId, groupId, timestamp, contentHash

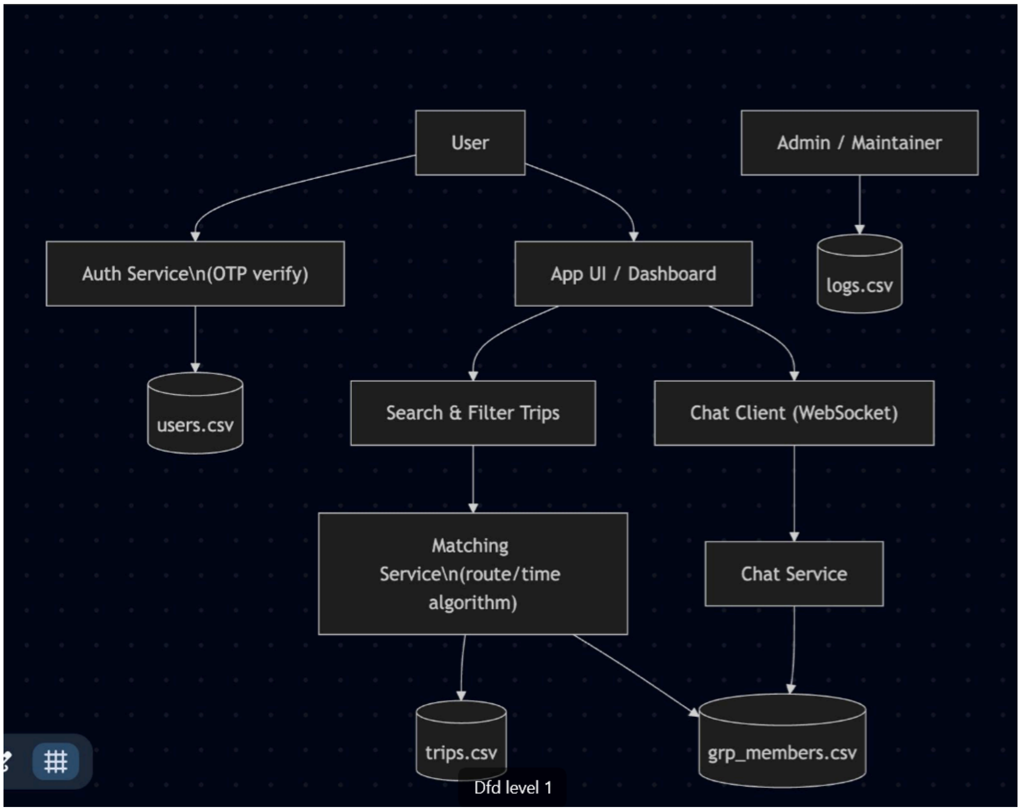


2.4 DFD Level 0, Level 1 & Level 2

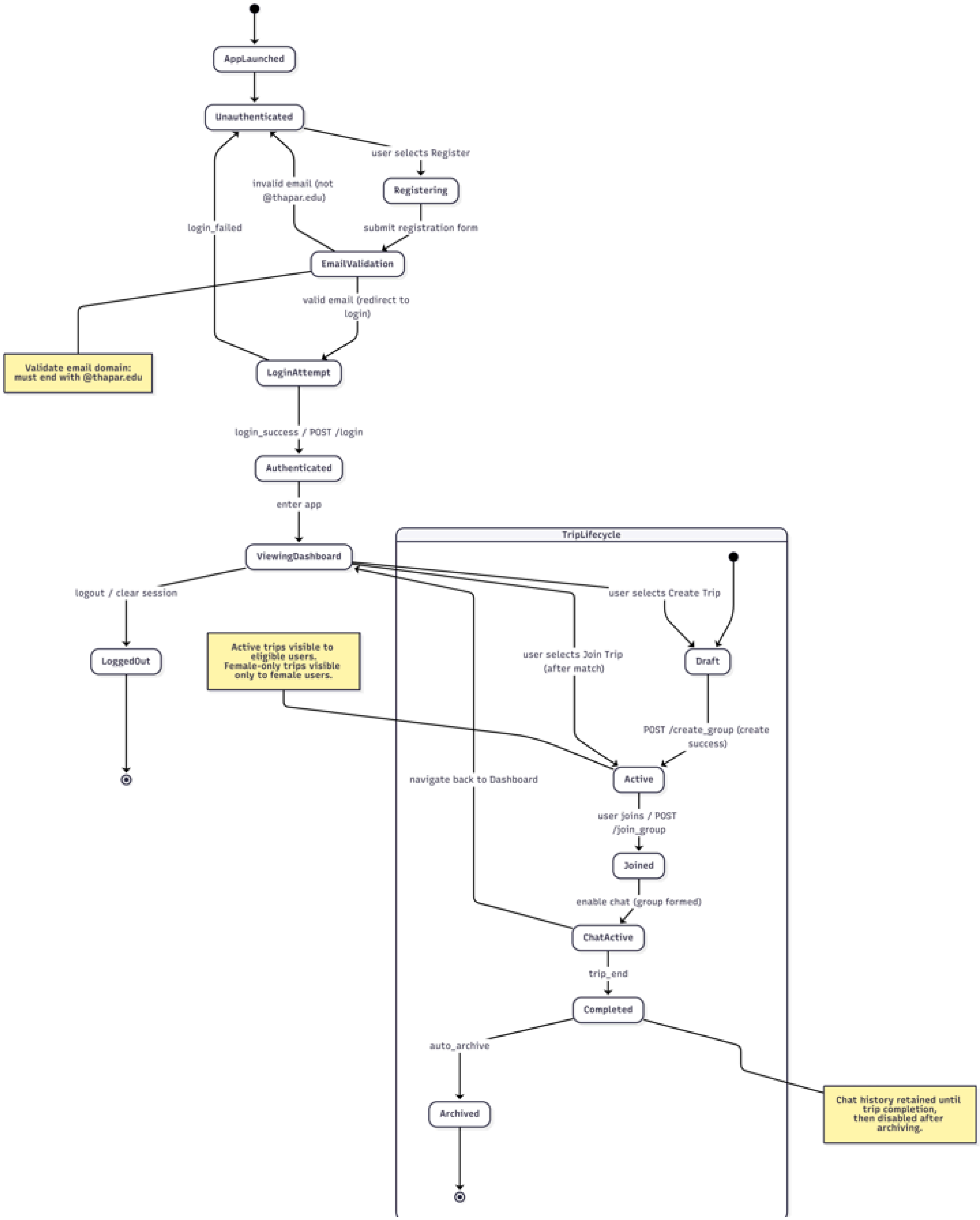
DFD Level 0 (context): User interacts with CampusRide System (Auth, Trip Matching, Chat).



DFD Level 1 expands into Authentication, Trip Management & Matching (includes Circle of Trust contact mining), In-App Chat, Notification, and Payment (future).

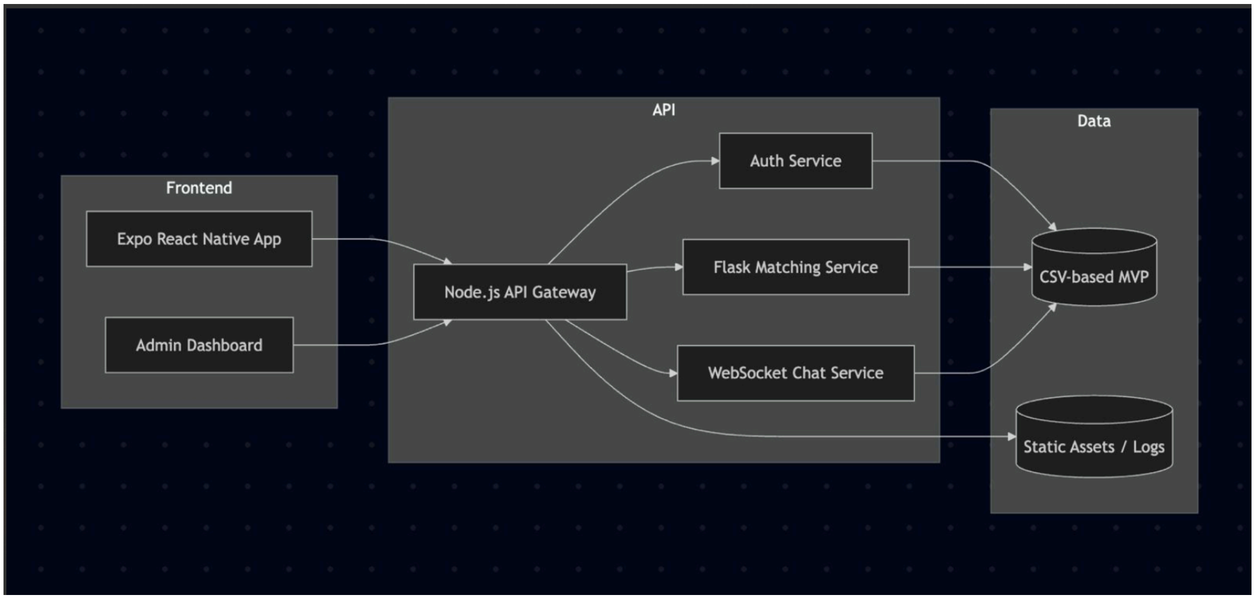


3. State Diagram


AppLaunched
Unauthenticated
user selects Register
Registering
invalid email (not @thapar.edu)
login_failed
submit registration form
EmailValidation
valid email (redirect to login)
Validate email domain: must end with @thapar.edu
LoginAttempt
login_success / POST /login
Authenticated
enter app
TripLifecycle
ViewingDashboard
logout / clear session
user selects Create Trip
LoggedOut
Active trips visible to eligible users. Female-only trips visible only to female users.
user selects Join Trip (after match)
Draft
POST /create_group (create success)
navigate back to Dashboard
Active
user joins / POST /join_group
Joined
enable chat (group formed)
ChatActive
trip_end
Completed
auto_archive
Archived
Chat history retained until trip completion, then disabled after archiving.

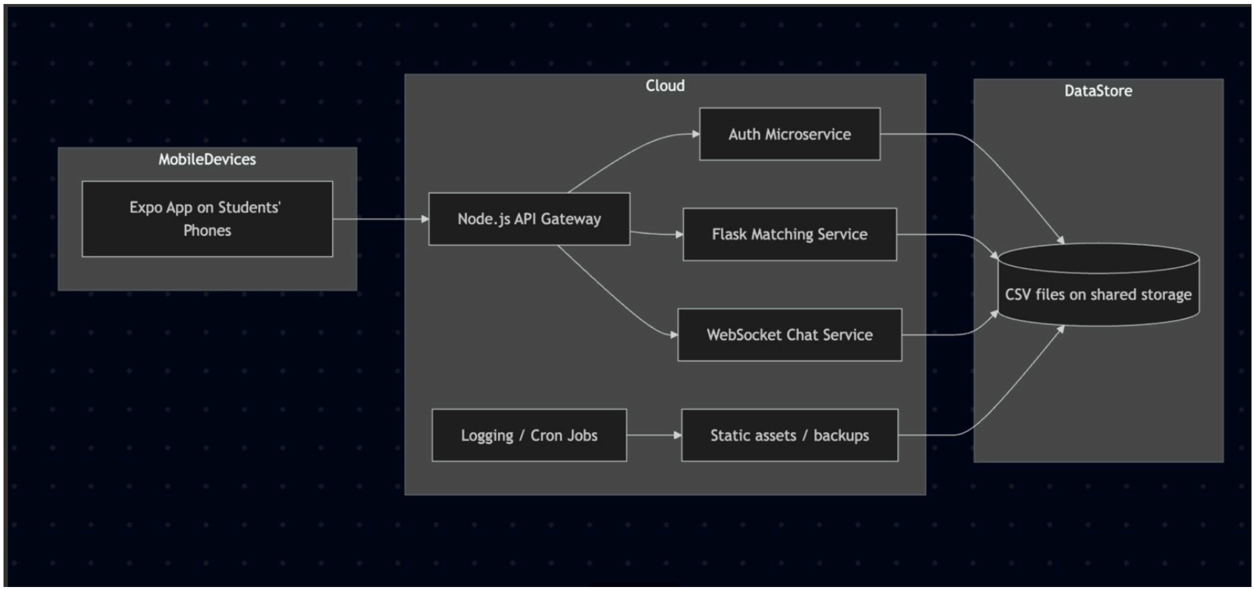
3.1 Component Diagram

High-level components:
- Mobile App (Expo React Native)
- API Gateway (Node.js)
- Auth Service (OTP, domain verification)
- Match Service (route parsing, contact mining, recommendation)
- Chat Service (SocketIO/Flask)
- Notification Service
- Storage (Managed SQL / Blob storage for media)



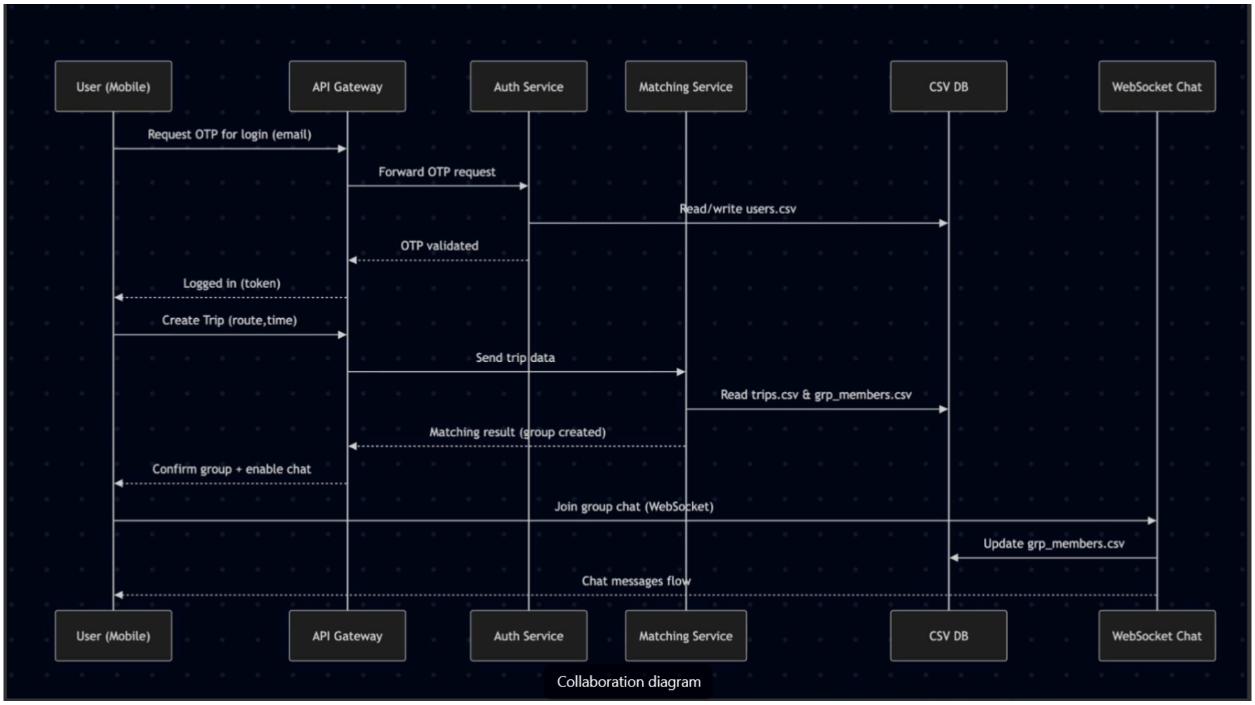
3.2 Deployment Diagram

Deployment targets for MVP:
- Single region cloud: Load Balancer -> App Servers -> Auth/Match/Chat -> Managed DB -> Object Store
- Mobile clients connect over HTTPS and WebSocket. Offline retry/backoff strategies are recommended for unstable campus networks.

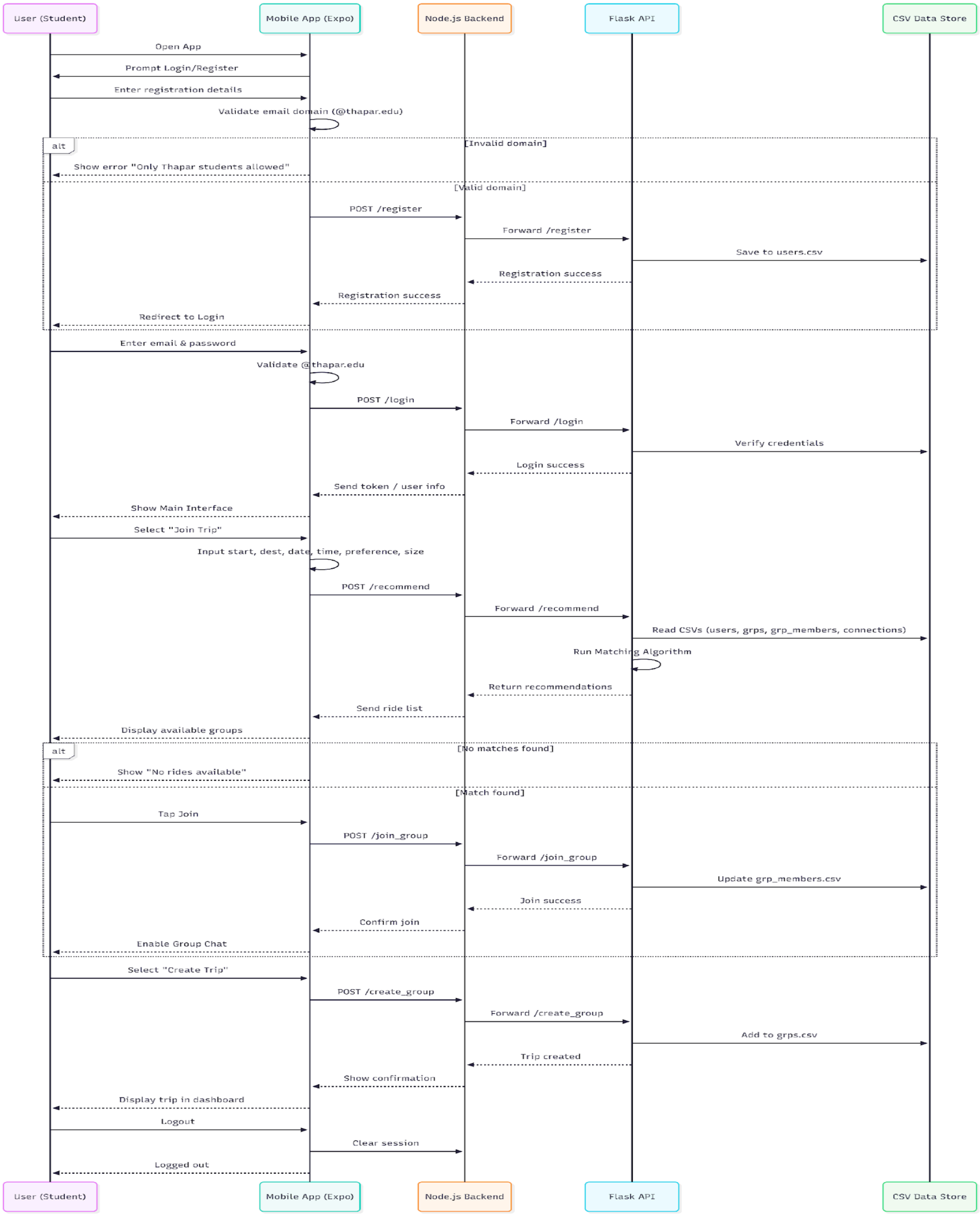


3.3 Sequence & Collaboration Diagrams

Key sequence: Login (App -> API -> Auth -> DB -> API -> App), Create Trip (App -> API -> Match -> DB -> API), Chat (App <-> ChatSrv via WebSocket). See Appendix for Mermaid code.

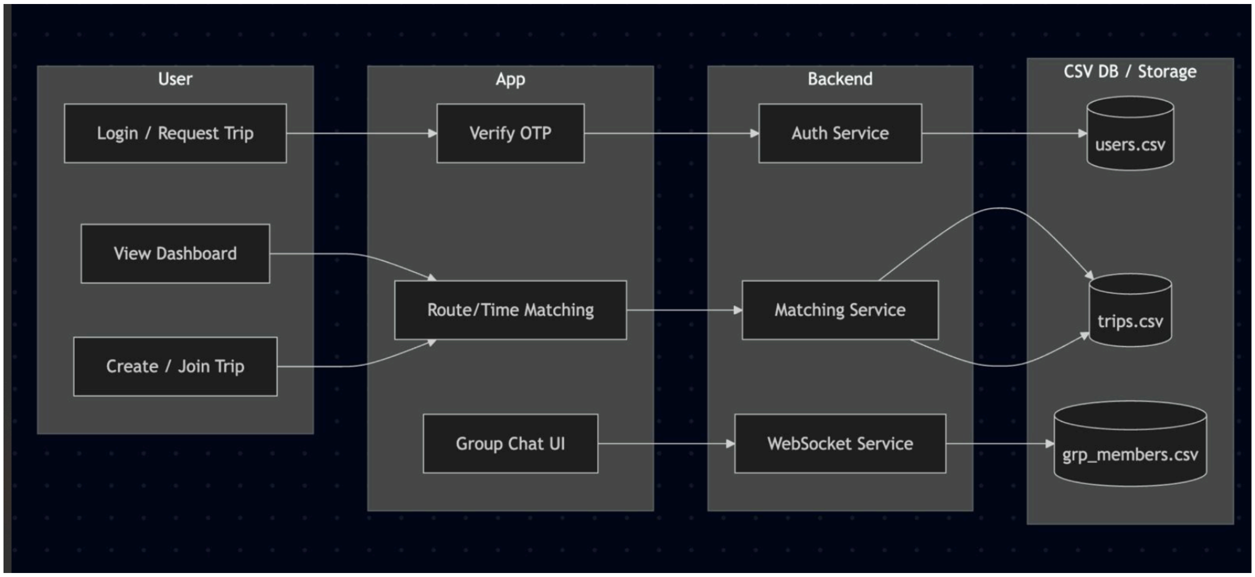


Collaboration diagram



Sequence Diagram

Swimlane Diagram



4. Implementation & Key Screens

MVP Tech Stack:
- Frontend: Expo (React Native)
- Backend: Node.js API Gateway, Flask microservices for Chat/Match
- Storage: Managed SQL for structured data, Blob storage for media
- Auth: OTP via campus email, JWT for sessions

Key screens (descriptions):
- Login/Onboarding (OTP verification)
- Dashboard (Recommended Trips, Create Trip)
- Create Trip (route entry, preferences, trust-level toggles)
- Trip Detail & Group Chat (WebSocket-powered chat)
- Profile & Trust Network (view 1°, 2°, 3° connections)

5. Testing Phase

Test Plan Overview: Unit tests for MatchEngine and Contact Mining, Integration tests for Auth and Chat, End-to-end tests for core flows (Login -> Create/Join Trip -> Chat -> Archive).

5.1 Sample Test Cases

Test ID: TC01
Title: OTP Login with valid @thapar.edu
Precondition: Student has a valid Thapar email
Steps: 1) Enter email 2) Receive OTP 3) Submit OTP
Expected: Login succeeds and dashboard opens

Test ID: TC02
Title: Create Trip and Match with 1° contacts
Precondition: User has existing 1° connections
Steps: 1) Create trip with source/dest/time 2) Observe recommended groups
Expected: Matching engine prioritizes groups with 1° connections

Test ID: TC03
Title: Group Chat Activation and Archival
Precondition: At least 2 members joined a trip
Steps: 1) Join group 2) Send messages 3) Mark trip as complete
Expected: Chat active during trip, archived after completion

6. Software Requirements Specification (SRS)

Purpose: Provide a complete functional and non-functional specification for CampusRide.
Scope: Mobile-first rideshare app limited to Thapar students (verified via email domain).
Functional Requirements (selection):
- FR1: User registration and OTP verification (Thapar email domain enforced).
- FR2: Trip creation, search, join, leave.
- FR3: Match engine with contact-mining producing 1°,2°,3° trust scores.
- FR4: Real-time group chat for trip members.

Non-Functional Requirements:
- NFR1: Response time for search results < 2s for typical campus dataset (<= 10k users).
- NFR2: Data at rest encrypted; chats stored with access controls.

Constraints:
- C1: MVP uses CSV/managed SQL for data; later migrate to graph DB for relationship queries.

7. Conclusion & Recommendations

CampusRide is a high-impact, low-cost student project with strong technical feasibility. The Circle of Trust feature improves safety and adoption. Recommended next steps: implement MVP, run pilot with 1–2 departments, and iterate on MatchEngine using pilot data.