COMET COURIER

Dana Ibrahim, Itay Kadosh, Ayaan Khan, Pranav Pulickal, Joel Roy, Patrick Sigler, Diego Villegas

Objective

- **Fun Fact**: **78%** of the **UTD student body** live off campus [1].
- Many students have a daily commute of **45-60** minutes just to get to their classes, from all around the **Dallas metroplex**.
- The goal is **simple**, to provide a **cheap** and **safe** alternative to commuting to campus, by **comets for comets**.
- We propose a **ride-share mobile application**, where commuters will **pick up** fellow students along the way, saving **gas**, **parking passes**, and provides company for the often long and boring commute.

Project Timeline Overview

- **Start Date**: January 7, 2025
- **End Date**: July 25, 2025
- **6 full-time members:**
  - *2 backend developers*
  - *2 mobile developers*
  - *1 QA engineer*
  - *1 product/requirements analyst*
- **Total Duration**: 6 - 7 Months
- **Work Schedule:**
  - *Monday – Friday (no weekends)*
  - *8 hours/day, 40 hours/week*
- Using **incremental Software Process Model**. Project divided into multiple increments. Each increment is **designed**, **implemented**, and **tested** separately

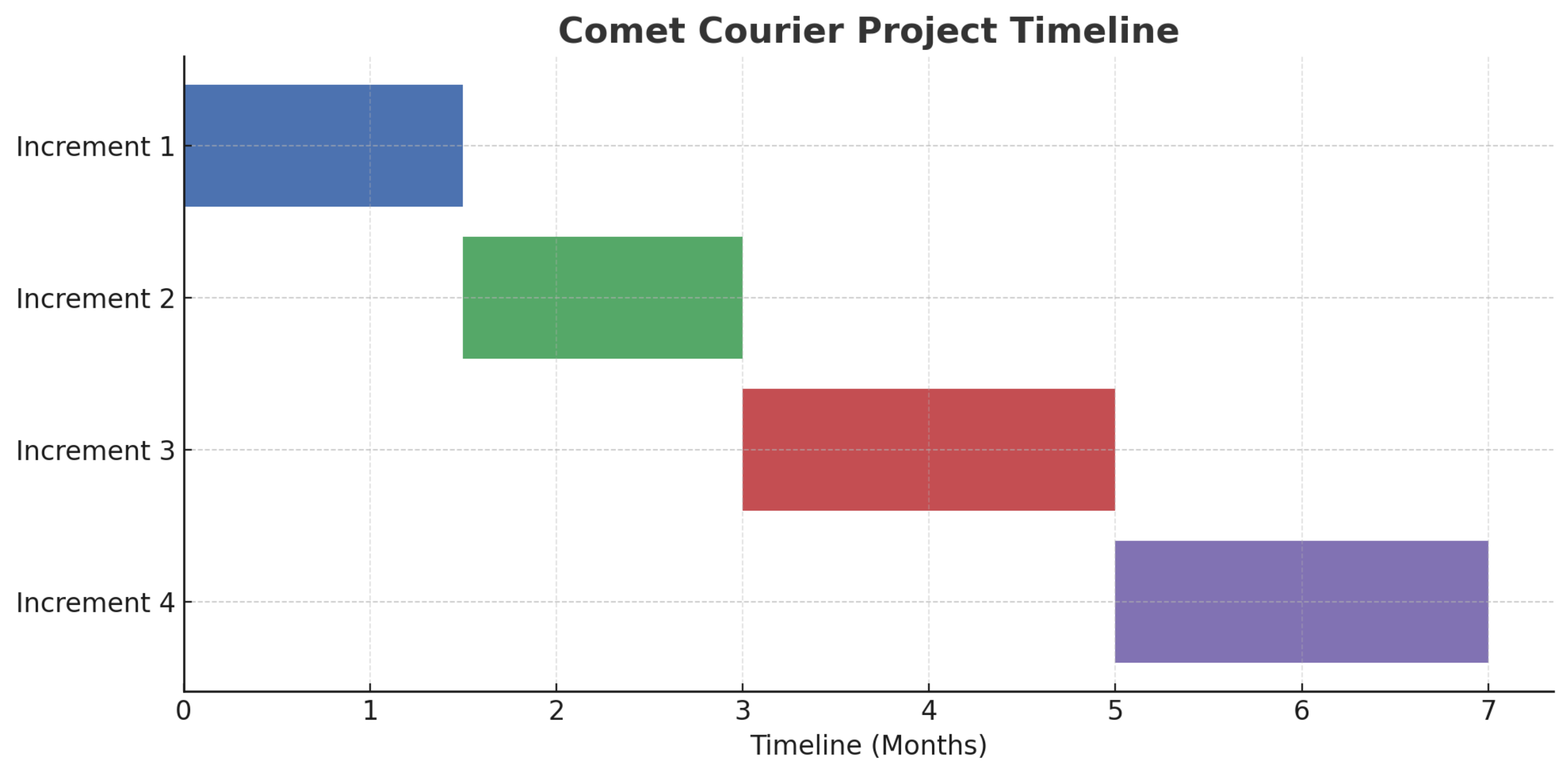
Project Timeline Increment Plans

- **Increment 1 – Jan 7 – Feb 15**
  - *High-Level architecture*
  - *Backend Skeleton + DB setup*
  - *Basic UI Skeleton*
  - *User Identification via UTD email setup*
- **Increment 2 – Feb 16 – Apr 1**
  - Ride Offer/ Ride Request features
  - Simple client matching algorithm (time + radius)
  - Mock notification implementation
  - In-app chat (early version)
  - Early QA testing

Project Timeline Increment Plans – Cont.

- **Increment 3 – Apr 2 – May 31**
  - *Ride State management*
  - *Ride History*
  - *Rating System*
  - *Safety Features (SOS + Share Trip Draft)*
  - *Improved Admin Support*
- **Increment 4 – Jun 1 – Jul 25**
  - Refined Final UI
  - Comprehensive Full QA
  - Load testing, bug fixes
  - Deployment Packaging
  - Releasing the build

Project Timeline
Increment Plans – Gantt Chart



Cost Estimation Labor

- **Estimation Methodology: Application Composition Model**
- **Reasoning:** Best suited for early-stage estimation based on object points (screens, reports, components).
- **Step 1: Effort Calculation**
  - ***Formula:*** *Effort (Person-Months) = Team Size x Duration*
  - ***Team:*** *6 Developers (Full-time)*
  - ***Duration:*** *7 Months*
  - ***Total Effort:*** *6 x 7 =* ***42 Person-Months***

Cost Estimation Labor

- **Step 2: Personnel Cost**
  - ***Formula:*** *Cost = Effort (PM) x Avg. Monthly Salary*
  - ***Rate: $15/hour*** *(approx.* ***$2,500/month****)*
  - ***Justification:*** *Rate assumes junior developers/interns*
  - ***Internal (Developers):*** *Testing effort is integrated into the* ***42 Person-Months*** *calculation. The QA Engineer's salary covers this function, so no separate "Testing Budget" is required.*
  - ***External (Clients/Users):*** *User Acceptance Testing (UAT) is conducted via a* ***Volunteer Beta Program*** *(using TestFlight/Play Console). Users are incentivized by early access, not monetary payments.*
  - ***Calculation:*** *42 PM x $2,500 =* ***$105,000***

Cost Estimation Hardware

- **Cloud Infrastructure Strategy (PaaS/SaaS):**
- **Usage Profile:** Consistent usage from Month 1 to Month 7 due to the **Incremental Process Model**.
- **Workflow:** As each increment is developed, it is immediately deployed to a cloud-based Staging Environment for QA testing.
- **Cost Basis:** Monthly cloud fees (AWS/Firebase) apply to the full **7-month development timeline**

Item	Est. Cost (7 mo)
Cloud Services (Firebase/AWS) for DB, Auth, Functions	**$2,000**
Developer Laptops	**$0 (BYOD)**
Testing Devices	**$0 (BYOD)**
Total Hardware Cost	**$2,000**

Cost Estimation Software

- **Strategic Tech Stack:** Selected **SwiftUI** and **VS Code** for their extensive open-source libraries, accelerating the "Application Composition" methodology.
- **Licensing Compliance:** All tools selected utilize **MIT or Apache 2.0 licenses**, ensuring full legal compliance for a for-profit commercial release without seat fees.
- **Scalability Costs:** The budget isolates costs strictly for **Consumption-Based Services** (Google Maps API), ensuring we only pay for value-add features (Geolocation) rather than development environments during development.

Item	Est. Cost (7 mo)
IDE (VS Code)	**$0**
Framework (Swift UI)	**$0**
OS (Linux, macOS, Windows)	**$0**
Licensed API (Google Maps)	**$500**

Total Project Cost & Pricing

- Total Estimated Cost
- This is the total cost for the business to build and launch the initial version of Comet Courier.

Cost Category	Total
Personnel	**$105,000**
Hardware	**$2,000**
Software	**$500**
Total Project Cost	**$107,500**

System Design & Comparison

- We developed the **system architecture** by **organizing** the **app** into **modules** such as **authentication**, **ride matching**, **chat**, and **moderation.**
- The team compared our design to similar systems like **Uber** and **Waze** Carpool to evaluate how our approach fit a university setting.
- All diagrams were refined to ensure they aligned with the finalized requirements and were suitable for implementation.

Functional Requirements

- **UTD Verification:** Register/sign in with UTD email
- **Ride Creation:** Users can post ride offers or requests (origin, time, destination, seats, optional recurring).
- **Matching & Notifications:** System finds matches within radius/time window and notifies both users; matches can be accepted or declined.
- **Chat:** In-app chat opens after a match and closes when ride ends or is cancelled.
- **Ride Tracking:** Supports states {Proposed, Accepted, En-route, Completed, Cancelled} and stores ride history.
- **Ratings & Reports:** After rides, users can rate (1–5 stars), comment, block, or report.
- **Cost Notes:** Riders can record an agreed fare visible only to participants.

Non-Functional Requirements

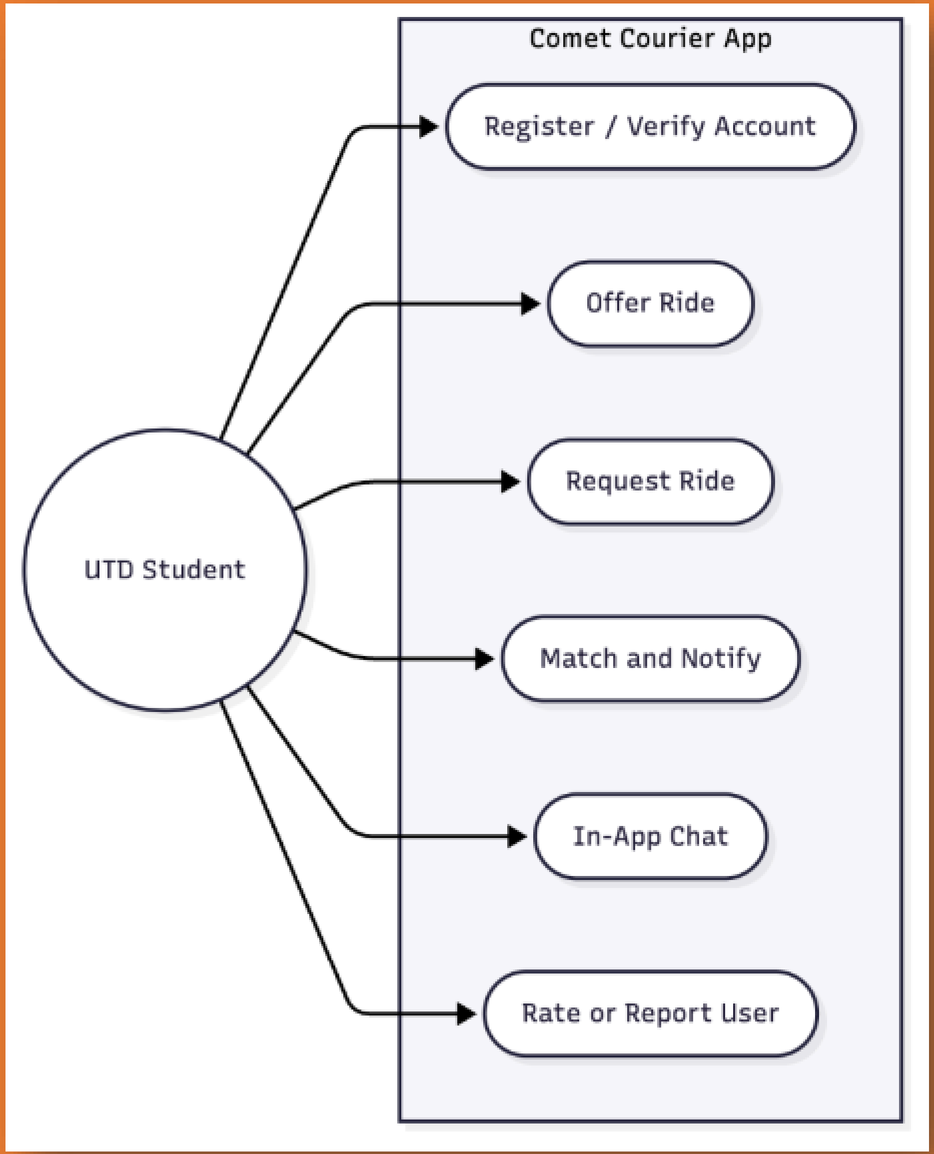
- **Usability:**
  - *First-time setup and posting ≤ 5 minutes.*
  - *≤ 7 taps to request/accept rides.*
  - *Meets WCAG 2.1 AA for contrast & text scaling.*
- **Performance:**
  - *Ride matches ≤ 2s, chat ≤ 1s, notifications ≤ 5s.*
- **Storage:**
  - *Download ≤ 80 MB, installed ≤ 150 MB.*
- **Dependability:**
  - *99.5% uptime; daily backups (RPO ≤ 24h, RTO ≤ 4h).*
  - *Ride history loss ≤ 0.01%.*
- **Security:**
  - *TLS 1.2+, hashed passwords, lockout after 5 failures, logged PII access.*
  - Duo 2 Factor Authentication required for access.

Non-Functional Requirements

- **Environment:** Works ≥ 512 kbps; Android + iOS; auto-resends on reconnect.
- **Operations:** Admin dashboard (view, suspend, manage reports); logs ≥ 90 days.
- **Development:** Code style enforced; ≥ 70% test coverage; static security checks.
- **Regulatory:** Texas / U.S. privacy-law compliant; users ≥ 18; PCI-DSS if payments.
- **Ethical:** No bias in matching; show code of conduct; enable report/block.
- **Accounting:** Track fare amount/date/ID; monthly summaries if digital pay.
- **Safety:** SOS to 911/campus police; live location sharing; pickup confirmation; verified drivers with visible info.

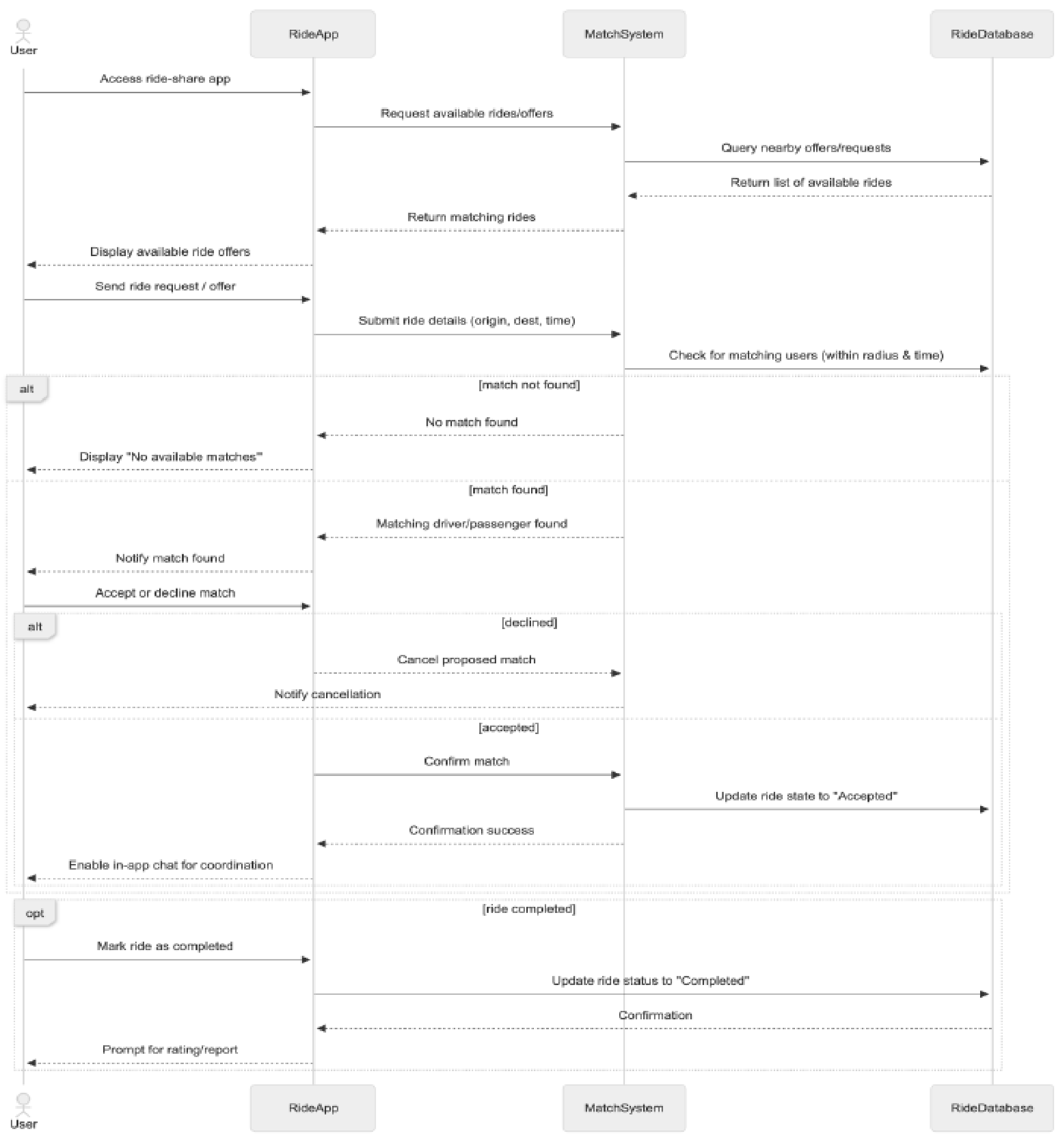
Use Case Diagram

- The student is the main user, all the interactions point to features in the app
- A student can register and verify their account with UTD to ensure it's for UTD students only
- A student can offer or request a ride
- When a ride is matched the student will get paired with someone in their time window and radius
- Once the ride is complete students can rate or report the user to ensure safety.



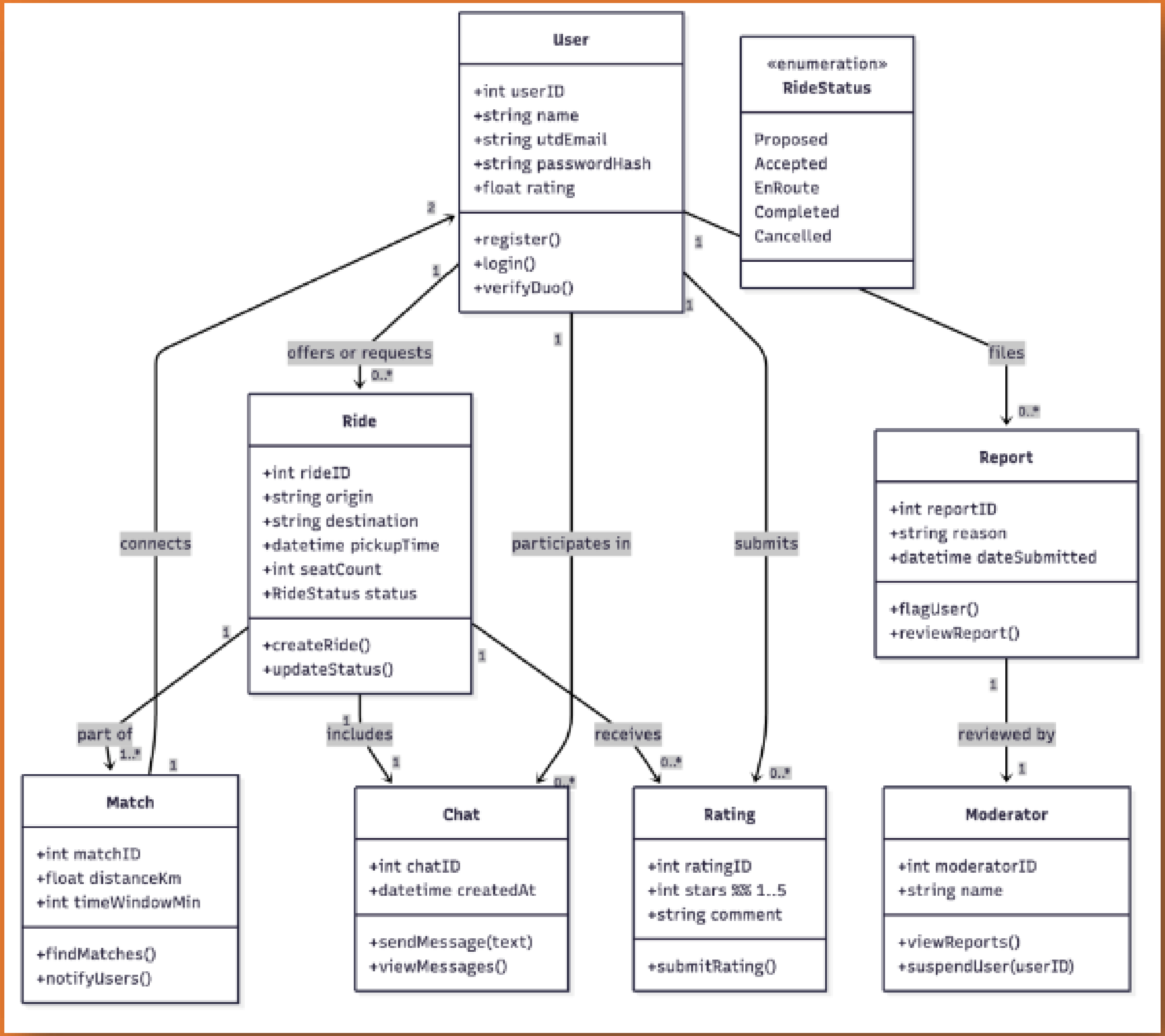
Sequence Diagram – Main Function

- Four main components are involved in the sequence diagram, Users, RideApp interface, MatchSystem, and RideDatabase
- The user will open the app which will allow the RideApp to request available rides or offers
- MatchSystem queries the ride database and will retrieve the list of available rides and send it back to the RideApp database
- MatchSystem checks for users nearby who fit the radius and time window, and there are two outcomes. If no match is found, the app will show display a message saying “No available matches.” however if the match is found it will notify the user and the driver. The user can accept or decline
- Once the ride is complete the user will mark the ride as completed and then the RideApp interface will prompt the user with a rating/report



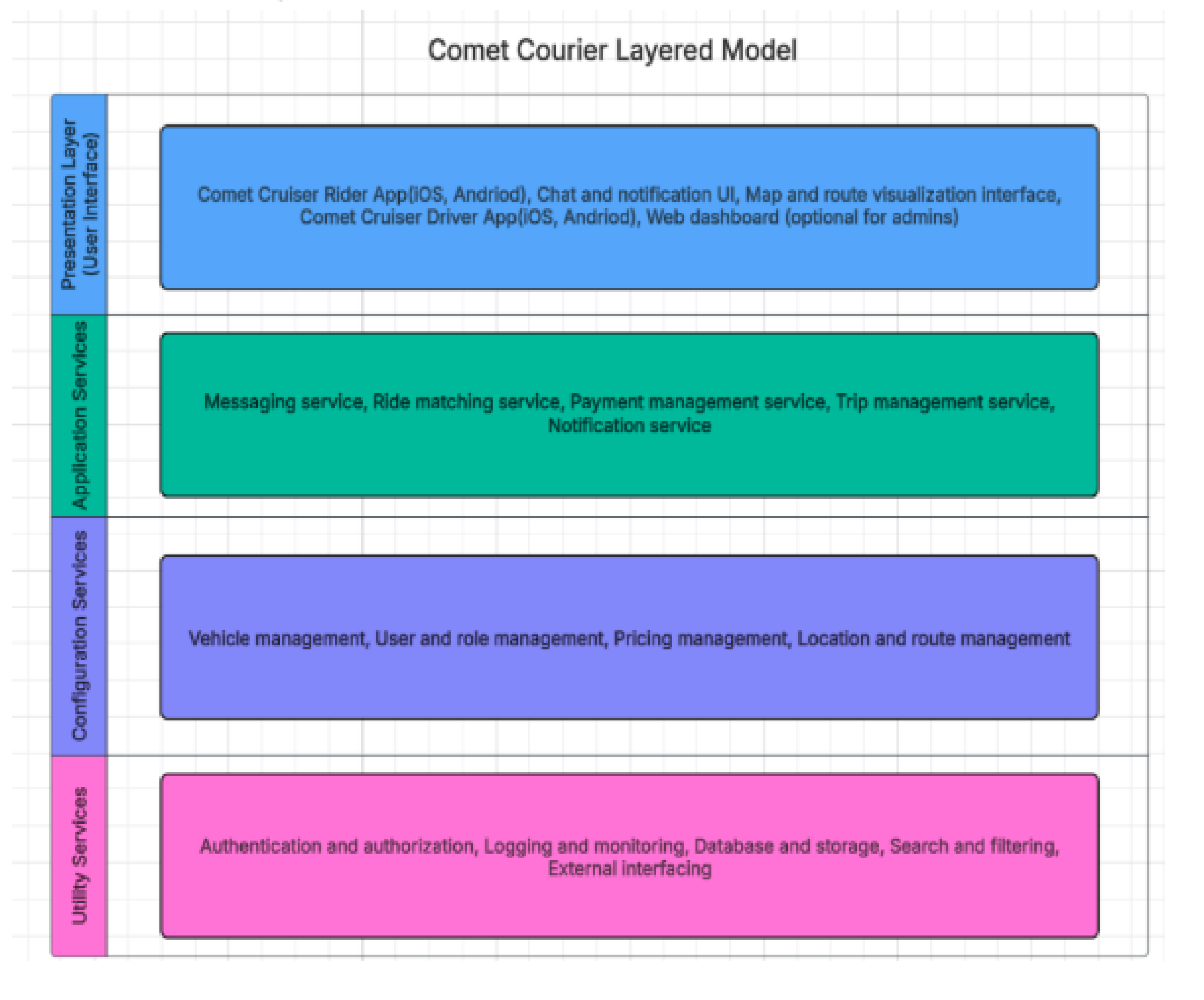
Class Diagram

- The class diagram shows the main objects in the system. The main class is the User class which stores the Users information. (ID, email, rating)
- Users can offer or request a ride which will connect them to the Ride class, the ride class will store information such as origin, destination, pickup time etc.
- The Ride class also connects to other classes such as Match class, which handles the match radius, time window and methods that find matches and notifies users
- After the ride is finished a user can submit a Rating (class) and if there is an issue they can submit a Report (class).



Architectural Design

- The system is modularly split into four layers that work together.
- At the top is the Presentation layer, which includes the actual user interfaces like the rider and driver apps
- The Application services layer shows the core functionality of the system, such as the messaging, ride matching, payments, and notification
- The Configuration services layer manages important data like vehicle information, user roles and pricing rules
- The Utility Services layer will handle the low-level operations, such as authentication and authorization, logging, searching, filtering, and database storage. This layer provides the foundational support for the higher layers.



References

- [1] “The University of Texas--Dallas student life - US news best colleges,” The University of Texas--Dallas Student Life, https://www.usnews.com/best-colleges/the-university-of-texas-dallas-9741/student-life (accessed Nov. 20, 2025).

Thank You!

Ayaan Khan, Dana Ibrahim, Joel Roy, Patrick Sigler, Pranav Pulickal, Diego Villegas, Itay Kadosh