CS/SE/CE 3354 Software Engineering
Comet Courier
Project Deliverable #2

Github: https://github.com/JoelRoy5/3354-Team3

Delegated Tasks by Member

1.1. Deliverable # 1 [Each member's tasks]

- **Joel Roy**: Making Github repo and adding collaborators, creating use case diagram
- **Itay Kadosh**: Sequence diagram (Figure 5.6), sequence diagram (Figure 5.7)
- **Ayaan Khan**: Case Diagram (Figure 5.5)
- **Dana Ibrahim**: Making the first commit (README file) to the repository, class diagram (Figure 5.9)
- **Patrick Sigler**: Functional requirements, non-functional requirements
- **Diego Villegas**: Software Process Model (ch. 2), 1.5: Project Scope.pdf
- **Pranav Pulickal**: Apply one Architectural Design pattern (ch. 6)

1.2. Deliverable # 2 [Each member's tasks]

- **Joel Roy**: Unit code/test plan, slides
- **Itay Kadosh**: Design Comparison, Sequence Diagrams, Slides
- **Ayaan Khan**: Sequence Diagrams, Slides
- **Dana Ibrahim**: Project Scheduling, Slides
- **Patrick Sigler**: Functional/non-functional requirements, conclusion, slides
- **Diego Villegas**: Test Plan
- **Pranav Pulickal**: Cost Estimation, Hardware/Software

Assumptions

For this project, we need to make a wide range of assumptions as a ride-sharing app in the real world can bring upon a range of issues, both legally and logistically. We make the assumptions:

- This ride sharing app will serve UTD students in the greater Dallas/Richardson area.
- Our users are current students with school verified emails.
- Trips are primarily to/from campus within common commute windows.
- Drivers are already en route to UTD and have
  - A safe Texas registered vehicle.
  - A valid drivers license.
  - Valid insurance.
- Location will be shared on a pickup/drop off basis, not a continuous tracking approach.

- Compensation occurs through the app with a simple in app payment flow.
- No need for background checks at the current moment.
- Legally it is feasible to create a ride sharing app without getting all the licenses of the real world.
    - Companies like Uber require a huge lawyer backbone as potential harms are detrimental, we make the assumption that this is no issue.

Addressing Feedback From Proposal

We received no feedback from the grader for the proposal, thus there is nothing to address. We will follow the original steps laid out in our proposal as well as this deliverable document.

Software Process Model Employed

We chose the Incremental Software Process Model for our app because it allows the system to be developed and delivered in small, functional increments. In each increment we can focus on implementing a set of related features, which can be tested and validated before moving onto the next phase. We chose this model because it supports continuous integration, customer feedback, and the ability to adapt to changing requirements. This is important for an app involving user safety and real-time coordination where reliability and usability should improve through testing.

Software Requirements

a. Functional requirements

1. UTD-verified accounts: The system shall allow users to register and sign in using a UTD email address.
2. Offer & request rides: The system shall let a user create either a ride offer (driver already en route to campus) or a ride request, each with origin, pickup window, destination (e.g., campus building/Parking lot), seat count, and optional recurring schedule.
3. Match & notify: Given a request or offer, the system shall compute candidates matches within a configurable radius and time window and shall push notifications to both sides; users shall be able to accept/decline a proposed match.
4. In-app chat: After a tentative match, the system shall provide an in-app chat thread for pickup coordination until the ride is marked complete or cancelled.
5. Ride state & history: The system shall support states {Proposed, Accepted, En-route, Completed, Cancelled} and maintain a ride history visible to the participants.
6. Reputation & reporting: After each completed ride, the system shall allow both parties to submit a 1-5 star rating and optional comments, and shall provide “block” and “report” actions that flag the account for moderation.
7. Cost note: The system shall allow passengers to record an agreed price note per ride, visible to the two participants in the ride thread.

b. Non-functional requirements

Product Requirements

- Usability Requirements
    - The app shall allow first-time users to post a ride request or offer within 5 minutes of registration.
    - The user interface shall be intuitive, requiring no more than 7 taps to create or accept a ride.
    - The system shall meet WCAG 2.1 AA standards for color contrast.
    - The system shall meet WCAG 2.1 AA standards for text scaling.
- Efficiency Requirements
    - Performance Requirements
        - Ride-match results shall be displayed within 2 seconds.
        - Chat messages shall be delivered within 1 second of sending.
        - Push notifications for matches and updates shall appear within 5 seconds.
    - Space Requirements
        - The app's download size shall not exceed 80 MB.
        - The app's installed size shall not exceed 150 MB.
- Dependability Requirements
    - The system shall maintain 99.5% uptime each month.
    - Backups shall occur every 24 hours.
    - The system shall maintain a Recovery Point Objective (RPO) of ≤ 24 hours.
    - The system shall maintain a Recovery Time Objective (RTO) of ≤ 4 hours.
    - User ride history loss shall not exceed 0.01% of stored records.
- Security Requirements
    - All network communication shall use TLS 1.2+ encryption.
    - Passwords shall be stored using salted hashing (bcrypt/scrypt).
    - Duo 2 Factor Authentication is required for access to verify ownership.
    - After 5 failed login attempts in 15 minutes, an account shall be temporarily locked.
    - Only authorized roles shall access personally identifiable information (PII).
    - All PII access shall be logged.

Organizational Requirements

- Environmental Requirements
    - Core functions shall operate under network speeds as low as 512 kbps.
    - Supported platforms include Android and iOS.

    - The app shall gracefully recover from temporary connection loss by resending queued requests.
- Operational Requirements
    - The admin dashboard shall allow moderators to view user reports.
    - The admin dashboard shall allow moderators to suspend users.
    - The admin dashboard shall allow moderators to manage abuse cases.
    - System health monitoring shall alert maintainers if downtime exceeds 5 minutes.
    - Logs of user actions and moderation activity shall be retained for at least 90 days.
- Development Requirements
    - Source code shall follow consistent style and naming conventions.
    - Automated tests shall cover at least 70% of critical modules.
    - Each build shall pass security linting and static analysis before deployment.

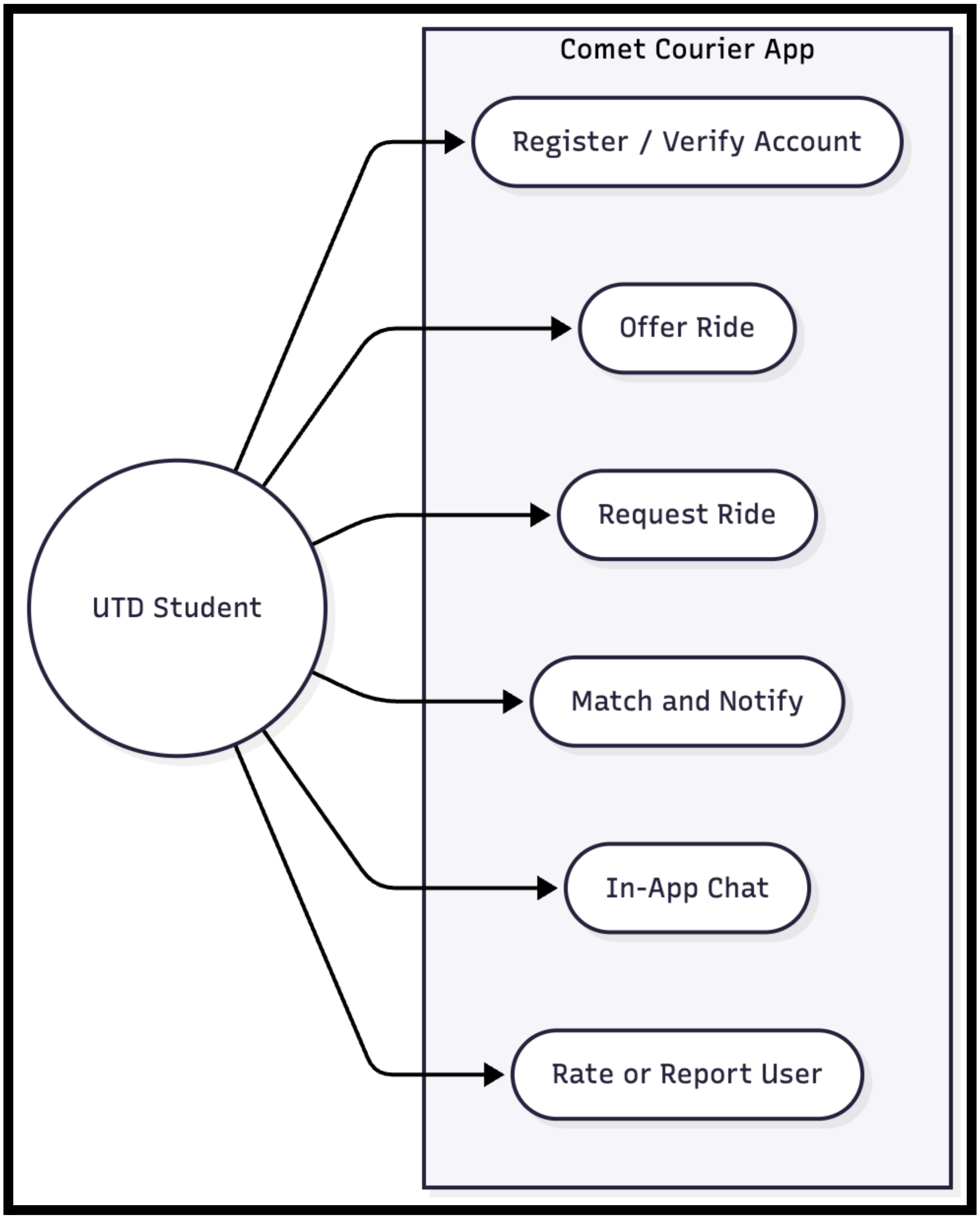
External Requirements

- Regulatory Requirements
    - The app shall include a privacy policy compliant with Texas and U.S. data-protection laws.
    - Only users aged 18 or older shall be eligible to register.
    - If online payments are implemented, processing shall occur through a PCI-DSS-compliant provider.
- Ethical Requirements
    - Ride matching shall not use protected personal attributes such as gender or ethnicity.
    - A code of conduct shall be shown and accepted upon first login.
    - The app shall provide "Report" and "Block" functions for inappropriate behavior.
- Legislative Requirements
    - Accounting Requirements
        - If fare notes or payments are used, the system shall record the amount, date, and ID.
        - Monthly summaries shall be available to users if digital payments are added.
    - Safety/Security Requirements
        - The app shall include an SOS button for immediate contact with 911 or campus police.
        - The app shall provide a Share Trip feature.
        - The Share Trip feature shall allow users to send their live location to a trusted contact.
        - If a passenger is not marked "picked up" after a defined window, the system shall prompt for confirmation.

- The system shall ensure all drivers are UTD-verified.
- The system shall display each driver’s name, rating, and vehicle information.

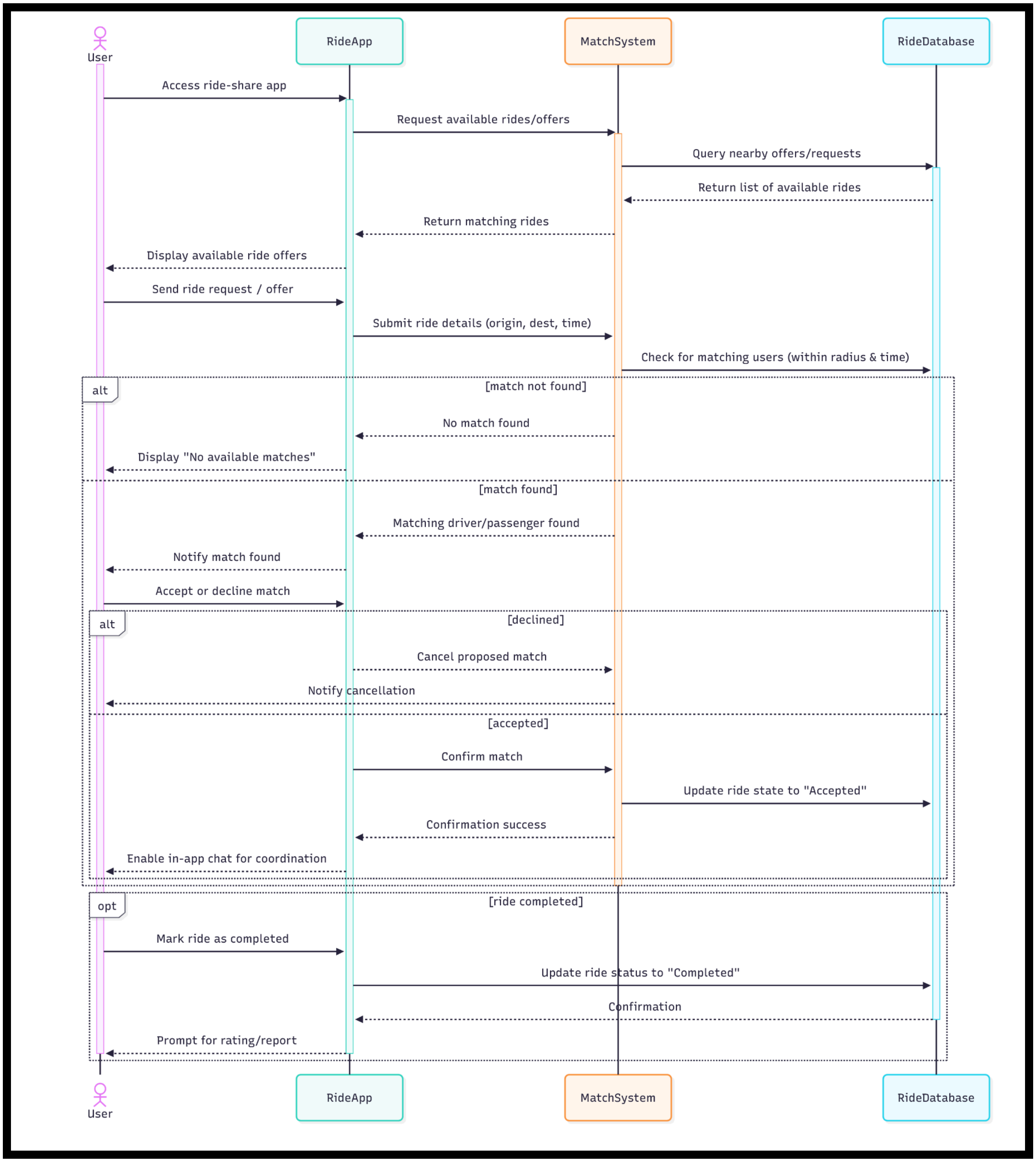
Use Case, Sequence, and Class Diagrams:

Use Case Diagram

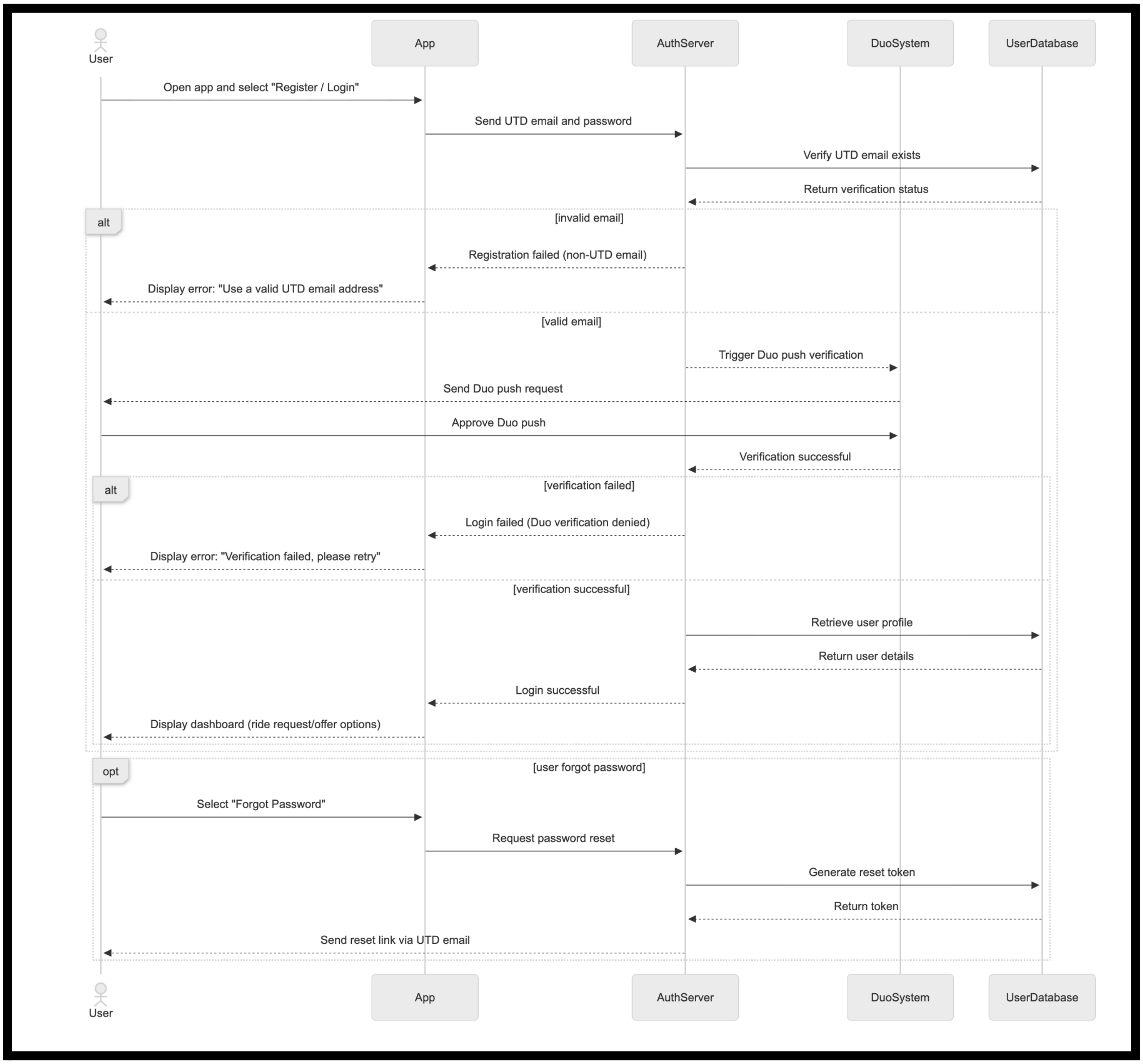


Sequence diagrams:

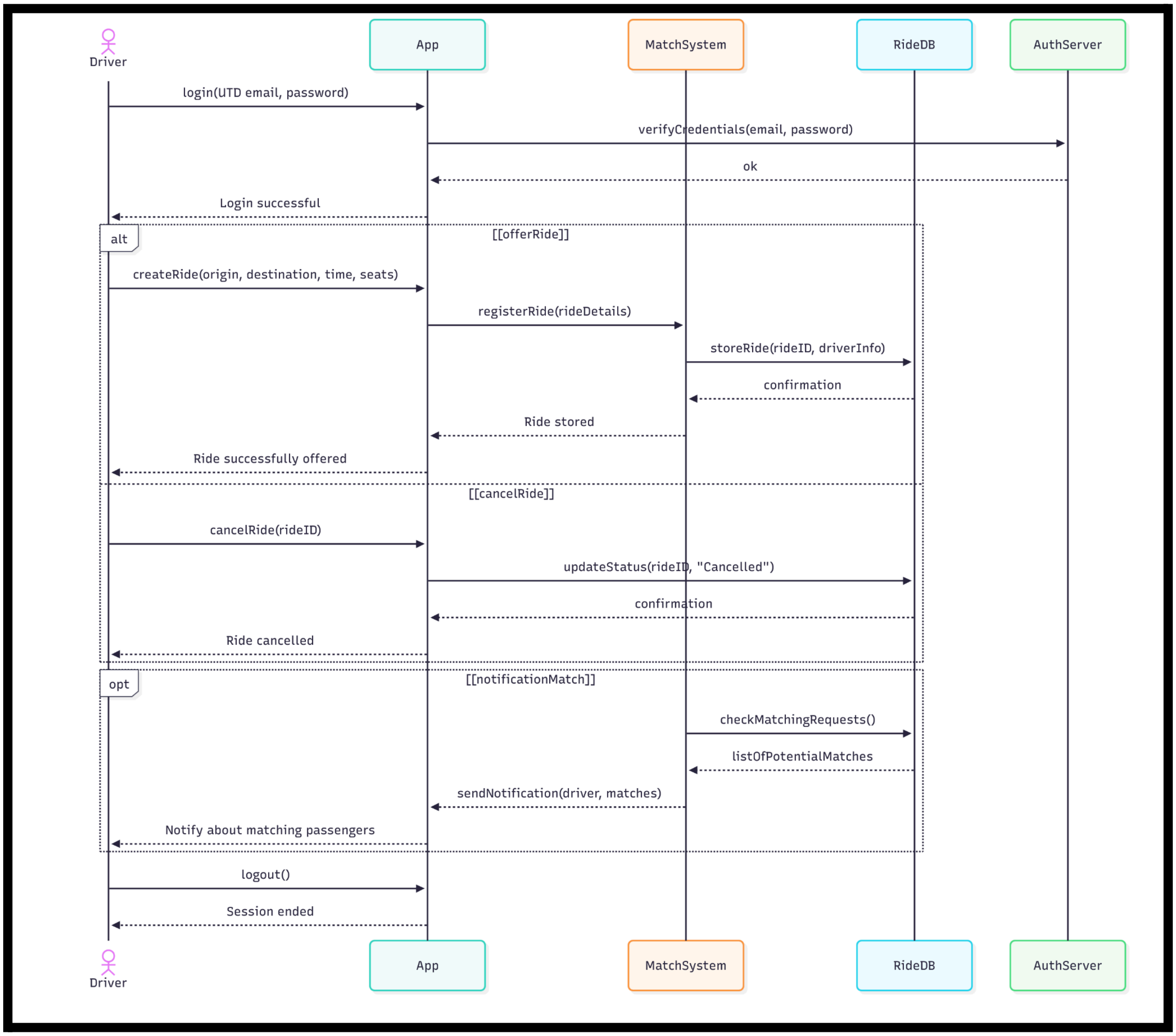
Main Functionality Sequence Diagram:


User
RideApp
MatchSystem
RideDatabase
Access ride-share app
Request available rides/offers
Query nearby offers/requests
Return list of available rides
Return matching rides
Display available ride offers
Send ride request / offer
Submit ride details (origin, dest, time)
Check for matching users (within radius & time)
alt
[match not found]
No match found
Display "No available matches"
[match found]
Matching driver/passenger found
Notify match found
Accept or decline match
alt
[declined]
Cancel proposed match
Notify cancellation
[accepted]
Confirm match
Update ride state to "Accepted"
Confirmation success
Enable in-app chat for coordination
opt
[ride completed]
Mark ride as completed
Update ride status to "Completed"
Confirmation
Prompt for rating/report
User
RideApp
MatchSystem
RideDatabase

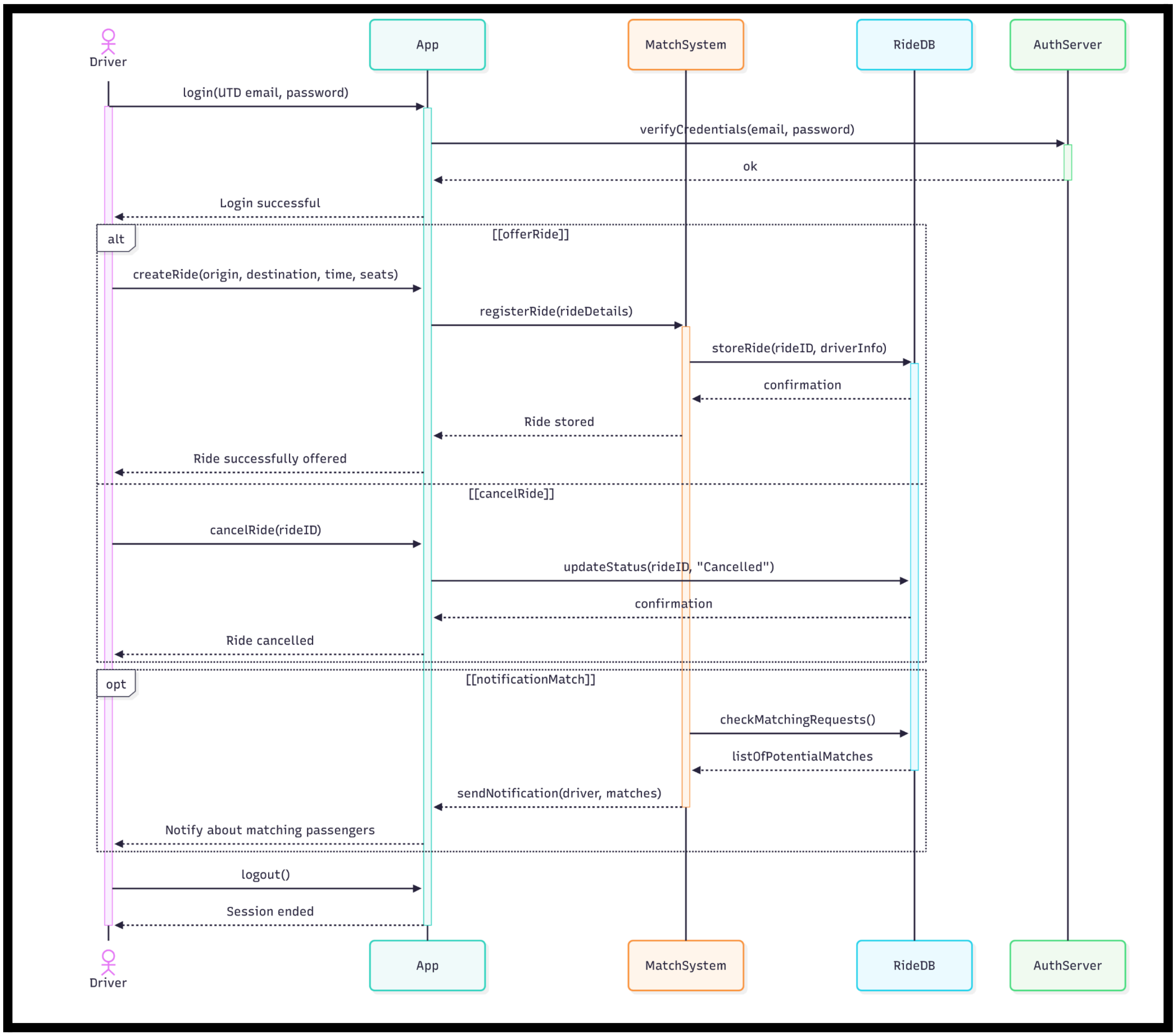
Verification and Login Sequence Diagram


User
App
AuthServer
DuoSystem
UserDatabase
Open app and select "Register / Login"
Send UTD email and password
Verify UTD email exists
Return verification status
alt
[invalid email]
Registration failed (non-UTD email)
Display error: "Use a valid UTD email address"
[valid email]
Trigger Duo push verification
Send Duo push request
Approve Duo push
Verification successful
alt
[verification failed]
Login failed (Duo verification denied)
Display error: "Verification failed, please retry"
[verification successful]
Retrieve user profile
Return user details
Login successful
Display dashboard (ride request/offer options)
opt
[user forgot password]
Select "Forgot Password"
Request password reset
Generate reset token
Return token
Send reset link via UTD email
User
App
AuthServer
DuoSystem
UserDatabase

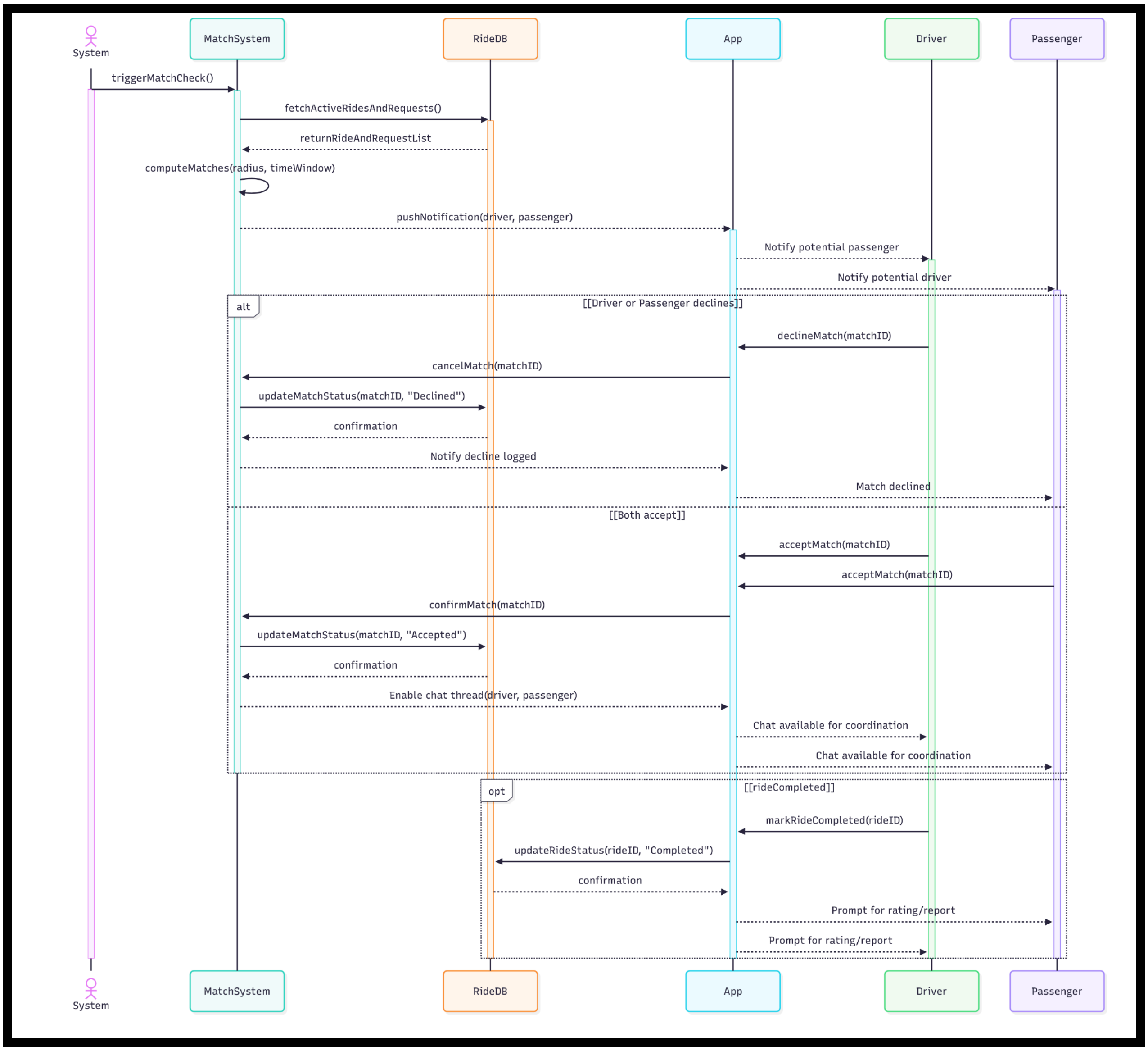
Offer Ride Sequence Diagram


Driver
App
MatchSystem
RideDB
AuthServer
login(UTD email, password)
verifyCredentials(email, password)
ok
Login successful
alt
[[offerRide]]
createRide(origin, destination, time, seats)
registerRide(rideDetails)
storeRide(rideID, driverInfo)
confirmation
Ride stored
Ride successfully offered
[[cancelRide]]
cancelRide(rideID)
updateStatus(rideID, "Cancelled")
confirmation
Ride cancelled
opt
[[notificationMatch]]
checkMatchingRequests()
listOfPotentialMatches
sendNotification(driver, matches)
Notify about matching passengers
logout()
Session ended
Driver
App
MatchSystem
RideDB
AuthServer

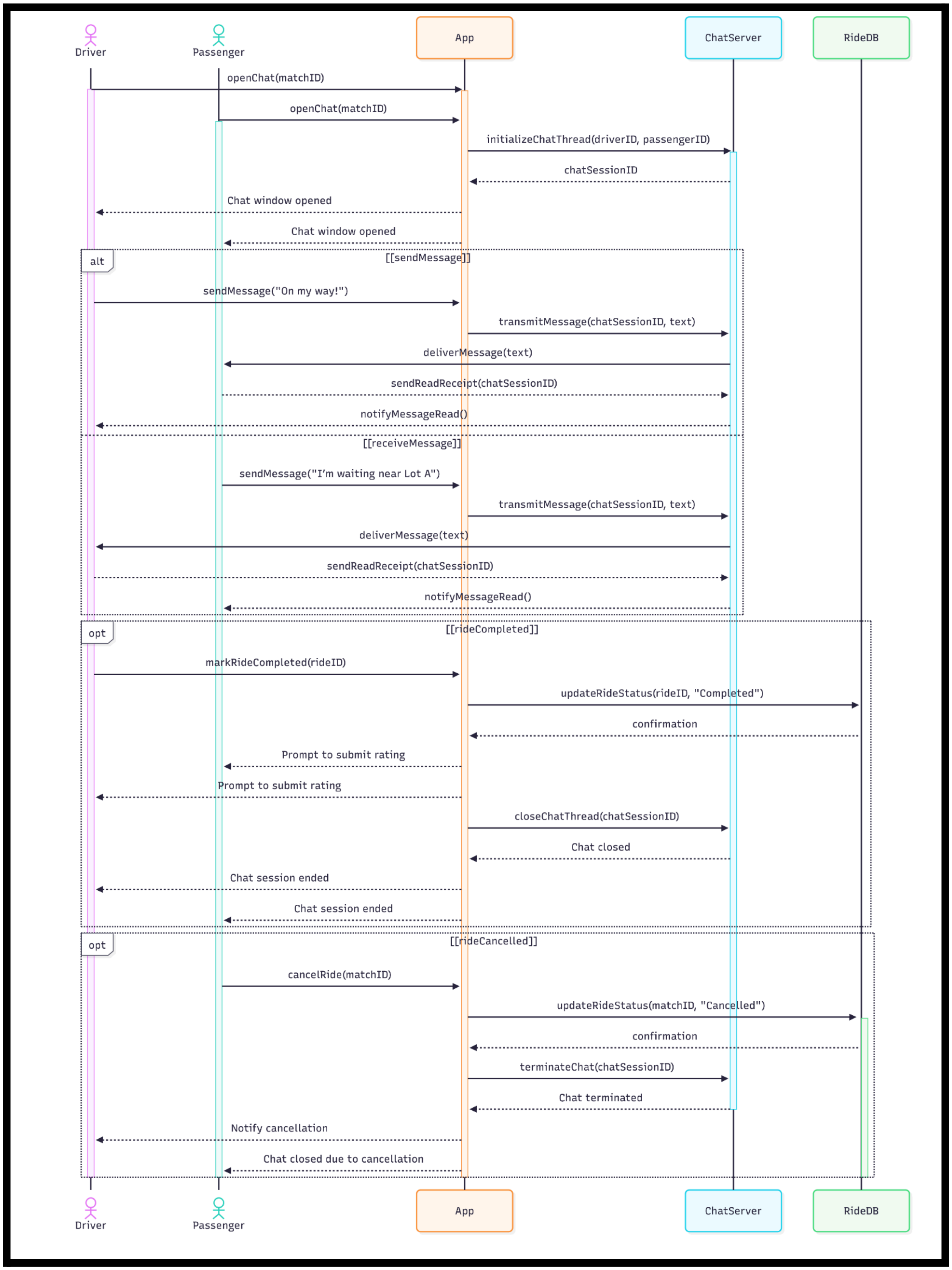
Request Ride Sequence Diagram


Driver
App
MatchSystem
RideDB
AuthServer
login(UTD email, password)
verifyCredentials(email, password)
ok
Login successful
alt
[[offerRide]]
createRide(origin, destination, time, seats)
registerRide(rideDetails)
storeRide(rideID, driverInfo)
confirmation
Ride stored
Ride successfully offered
[[cancelRide]]
cancelRide(rideID)
updateStatus(rideID, "Cancelled")
confirmation
Ride cancelled
opt
[[notificationMatch]]
checkMatchingRequests()
listOfPotentialMatches
sendNotification(driver, matches)
Notify about matching passengers
logout()
Session ended
Driver
App
MatchSystem
RideDB
AuthServer

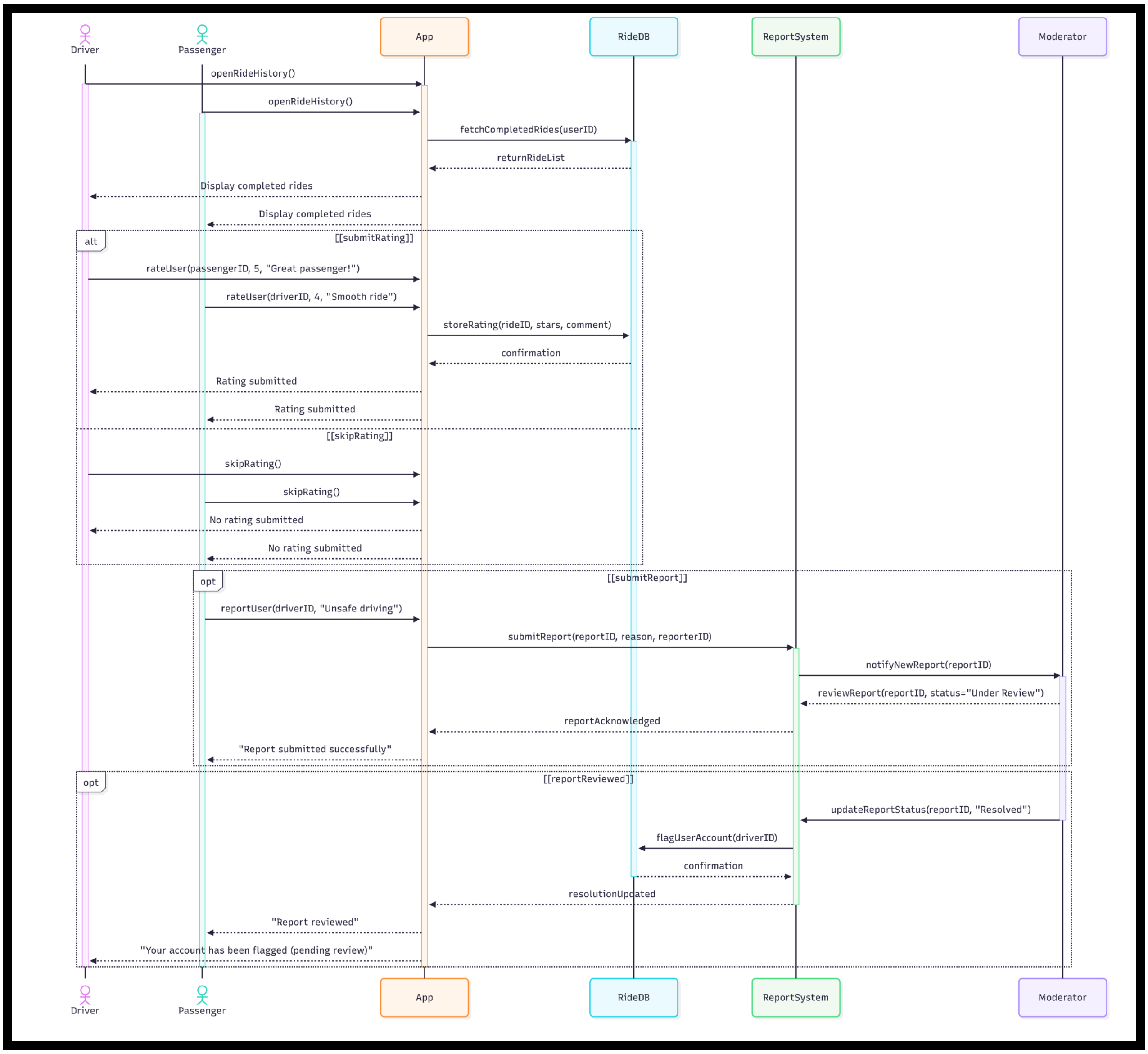
Match and Notify Sequence Diagram



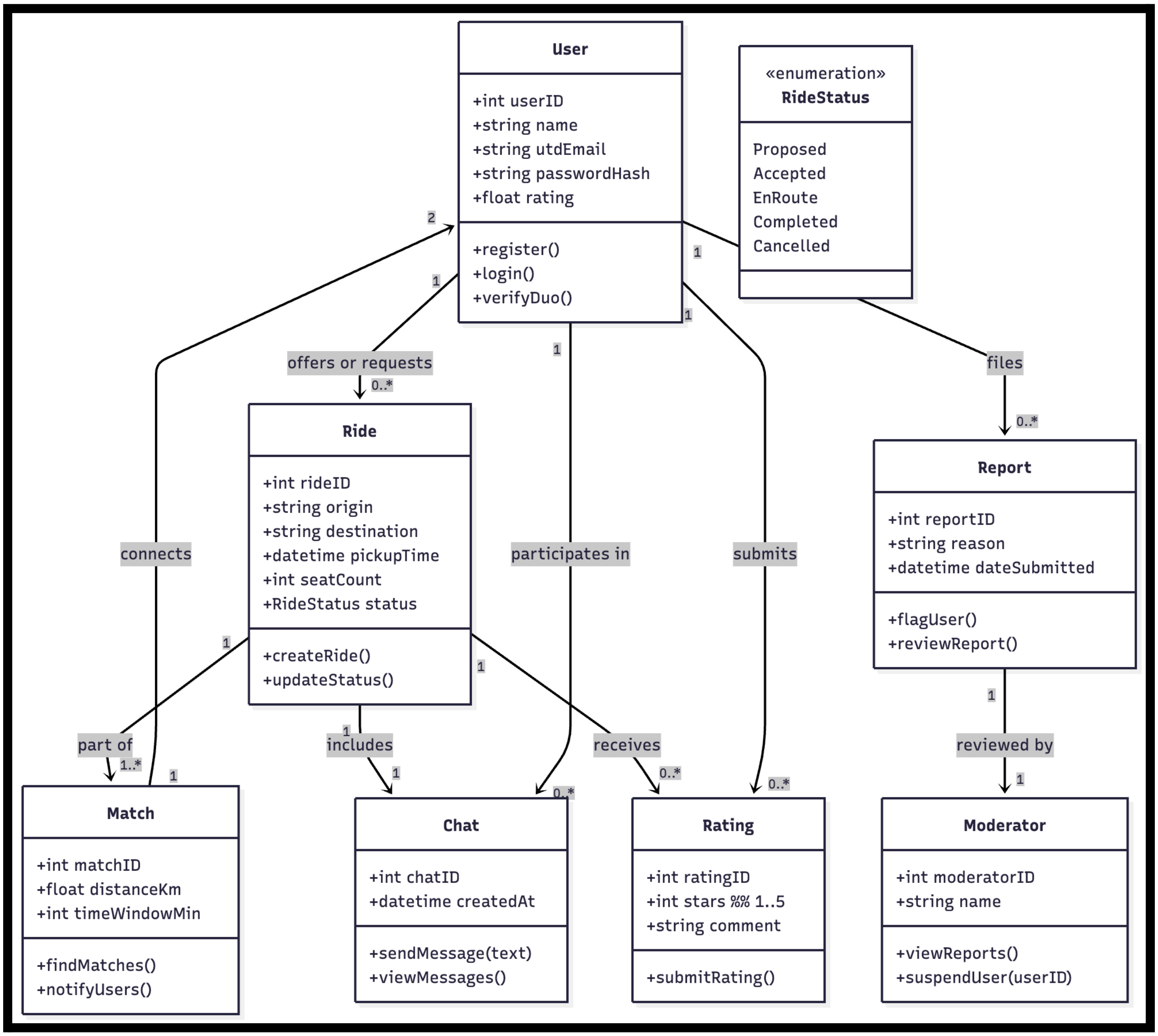
In App Chat Sequence Diagram



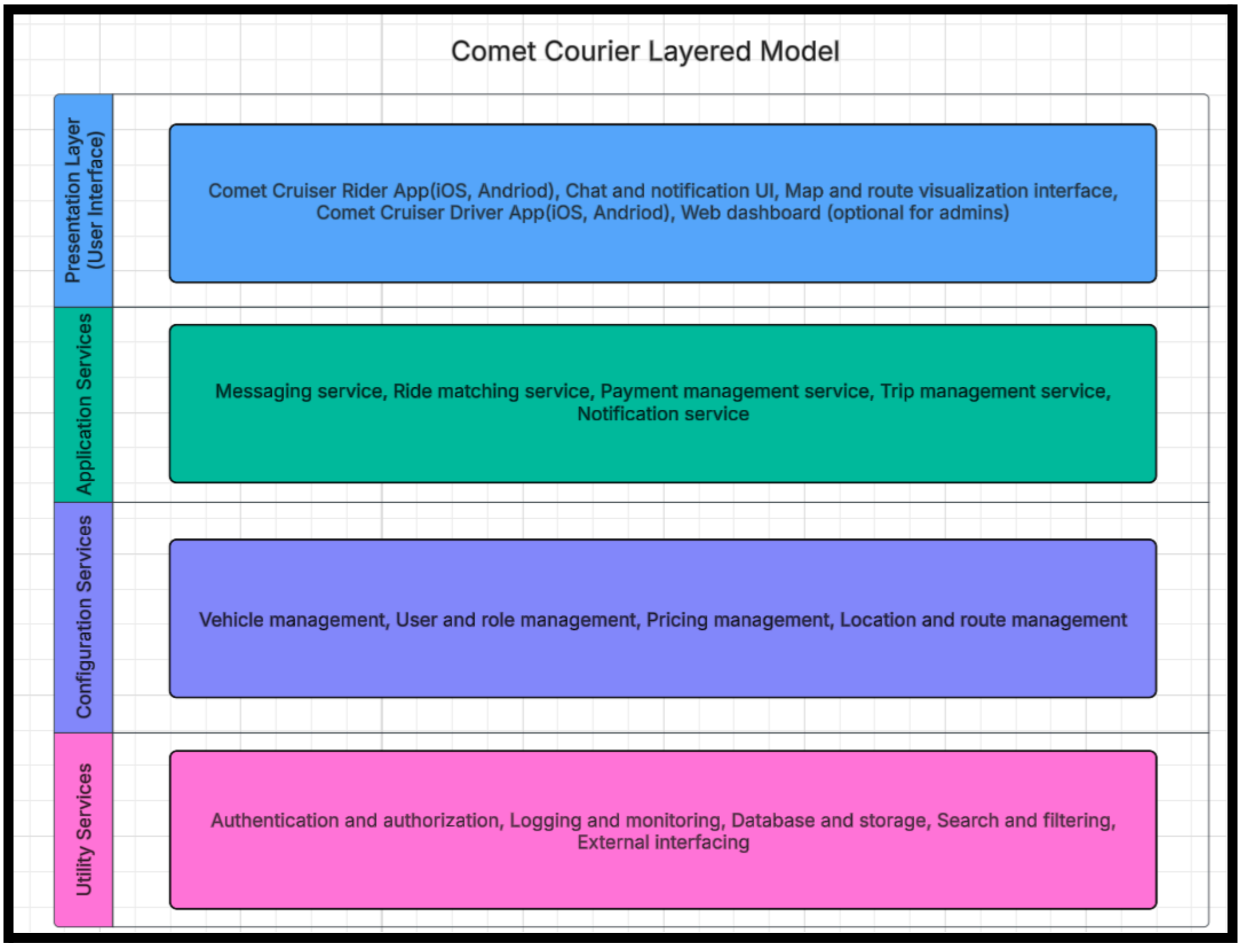
Rate and Report Users Sequence Diagram


Driver
Passenger
App
RideDB
ReportSystem
Moderator
openRideHistory()
openRideHistory()
fetchCompletedRides(userID)
returnRideList
Display completed rides
Display completed rides
alt
[[submitRating]]
rateUser(passengerID, 5, "Great passenger!")
rateUser(driverID, 4, "Smooth ride")
storeRating(rideID, stars, comment)
confirmation
Rating submitted
Rating submitted
[[skipRating]]
skipRating()
skipRating()
No rating submitted
No rating submitted
opt
[[submitReport]]
reportUser(driverID, "Unsafe driving")
submitReport(reportID, reason, reporterID)
notifyNewReport(reportID)
reviewReport(reportID, status="Under Review")
reportAcknowledged
"Report submitted successfully"
opt
[[reportReviewed]]
updateReportStatus(reportID, "Resolved")
flagUserAccount(driverID)
confirmation
resolutionUpdated
"Report reviewed"
"Your account has been flagged (pending review)"
Driver
Passenger
App
RideDB
ReportSystem
Moderator

Class diagram



Architectural design



Comet Courier Layered Model

Presentation Layer (User Interface)

Comet Cruiser Rider App(iOS, Andriod), Chat and notification UI, Map and route visualization interface, Comet Cruiser Driver App(iOS, Andriod), Web dashboard (optional for admins)

Application Services

Messaging service, Ride matching service, Payment management service, Trip management service, Notification service

Configuration Services

Vehicle management, User and role management, Pricing management, Location and route management

Utility Services

Authentication and authorization, Logging and monitoring, Database and storage, Search and filtering, External interfacing

Project Scheduling, Cost, Effort, and Pricing Estimation, Project duration and staffing:

Project Scheduling:

A mobile ridesharing app like Comet Courier requires user authentication, location services, driver–rider matching logic, messaging, database integration, and payment processing. We assume a development team of 6 full-time members (2 backend developers, 2 mobile developers, 1 QA engineer, and 1 product/requirements analyst). A realistic timeline for a professional software development team to complete these features with proper development, testing, and deployment is approximately 6 - 7 months. Project start date is January 7th, 2025 and project end date is July 25th, 2025. This includes backend development, mobile app development, user interface design, integration testing, and initial beta testing. Professional software development companies normally follow a Monday through Friday, 40-hour weekly schedule and weekend work is reserved only for emergencies or deployment days. The working hours per day will be 8 hours per day, this is because a full-time development team typically works 8-hour days. Because our team is using the Incremental Software Process Model, this overall timeline is divided into several increments allowing the system to be designed, implemented, and tested in stages throughout the 6–7 month schedule.

Cost, Effort and Pricing Estimation

Estimation Methodology: ***Application Composition Model***
Reasoning: The Application Composition Model is best suited for early-stage estimation when the system can be described in terms of **screens, reports, and 3GL components**. Comet Courier fits this pattern as a mobile app with a set of UI views, admin/reporting screens, and backend services (matching, SOS, notifications, etc.).

Step 1 - Object Point Identification: We classify the application into three object types:

- **Screens (UI views)** – mobile app pages used by riders, drivers, and admins
- **Reports** – views that summarize stored data
- **3GL Components** – backend services implemented in a general-purpose language

For Comet Courier, the team agreed on the following counts and complexity levels:

Object Type	Count	Complexity	OP / Item	Subtotal OP
Screens	10	mix of medium/difficult	5.2 (avg)	52
Reports	2	medium	4	8
3GL Components	8	complex	10	80
Total OP				**140**

Step 2 - Reuse*:* Although we leverage Firebase and AWS (Auth, DB, and Functions) and Google Maps, a student team still needs to write significant glue code, validation, and custom logic. We therefore assume only **10%** effective reuse.

Thus, using the formula **NOP = OP * (1 - Reuse),** we can calculate our new object points to be

NOP = 140 * 0.9 = 126 new object points

Step 3 - Productivity: Productivity in the Application Composition Model is expressed as new object points per person-month (NOP/PM) and depends on:

- Developer experience (student team, junior-level)
- Tool maturity (VS Code + SwiftUI + Firebase, but no heavy enterprise CASE tools)

To remain conservative, the team assumes

PROD = 3 NOP per person-month

This reflects a junior team composed of students that will spend extra time learning tools integrating services, and debugging.

Step 4 - Effort Calculation: Using the Application Composition effort formula:

PM = NOP/PROF = 126/3 = 42 person-months

Step 5 - Personnel Cost: We use the formula **Cost = Effort (PM) * Avg Monthly Salary.** Using the numbers that we derived above, and the agreed upon rate of $15/hour, equating roughly $2,500 dollars monthly salary, we calculate that our total personnel cost is

42 PM * $2,500 = $105,000

Our team will be composed of junior developers, who are computer science / software engineering students. This helped inform our decisions regarding pay, estimated person-months, object points, etc.

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 across the full 7-month development timeline.

Cloud & Hardware Cost Breakdown

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

Strategic Tech Stack

Tech Choice Rationale:
SwiftUI and Xcode were selected for their extensive built-in and open-source library support, which accelerates the *Application Composition* methodology.

Licensing Compliance:
All tools use MIT or Apache 2.0 licenses, allowing full legal commercial usage with no seat-based licensing fees.

Scalability Cost Approach:
Costs are isolated to *consumption-based services* (e.g., Google Maps API), ensuring spending only on value-add features rather than development tooling.

Software Cost Breakdown

Item	Est. Cost (7 Months)
IDE (Xcode)	**$0**
Framework (SwiftUI)	**$0**
OS (macOS / Linux / Windows)	**$0**
Licensed API (Google Maps)	**$500**

Total Estimated Cost

This represents the complete cost 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**

Test Plan

We will be testing the bookRide() method from our Ride class. This method allows a user to book a ride by sending their pickup and dropoff location as well as the time that they want to be picked up. The objective here is to ensure that the request is processed correctly and that a positive confirmation is returned.

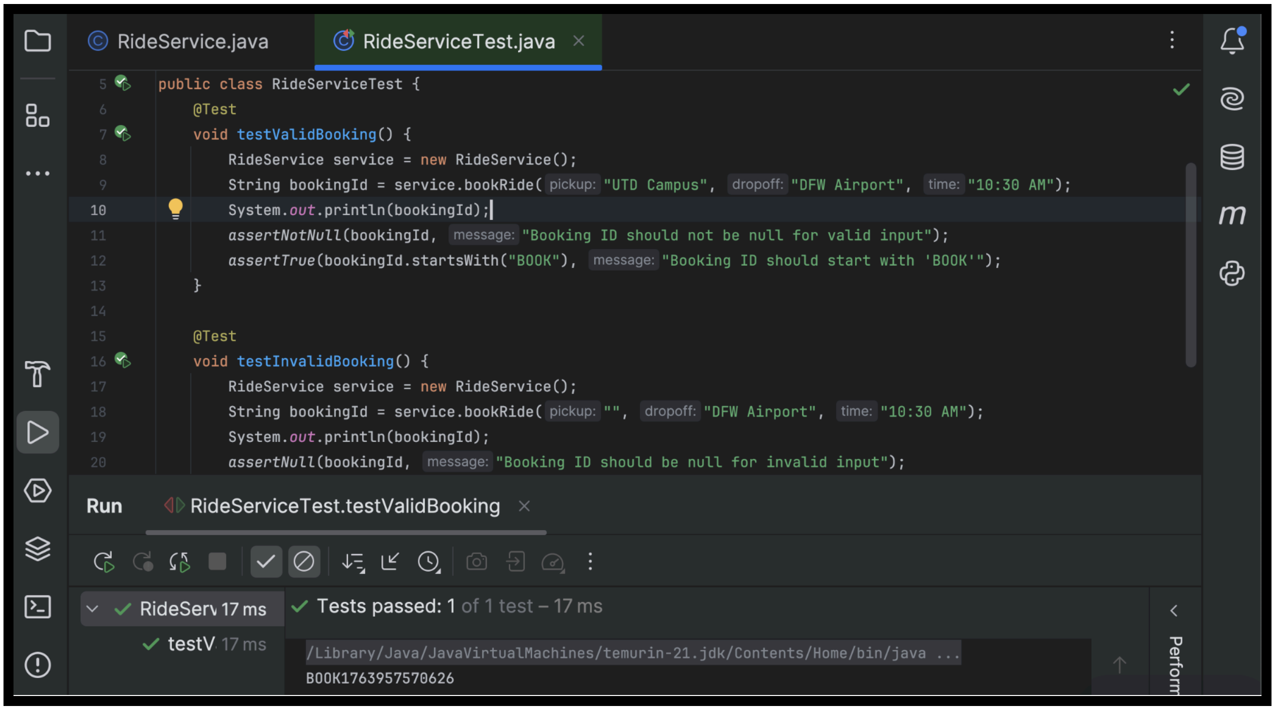
The test goals are to verify that the method successfully creates a booking when given correct inputs, verify that invalid inputs are rejected, and ensure that our response gives the correct booking ID. We will be using JUnit 5 and will test the backend API logic.

RideService.java

```java
public class RideService {
    public String bookRide(String pickup, String dropoff, String time) {
        if (pickup == null || pickup.isEmpty() ||
            dropoff == null || dropoff.isEmpty() ||
            time == null || time.isEmpty()) {
            return null; // invalid booking
        }

        return "BOOK" + System.currentTimeMillis();
    }
}
```

RideServiceTest.java

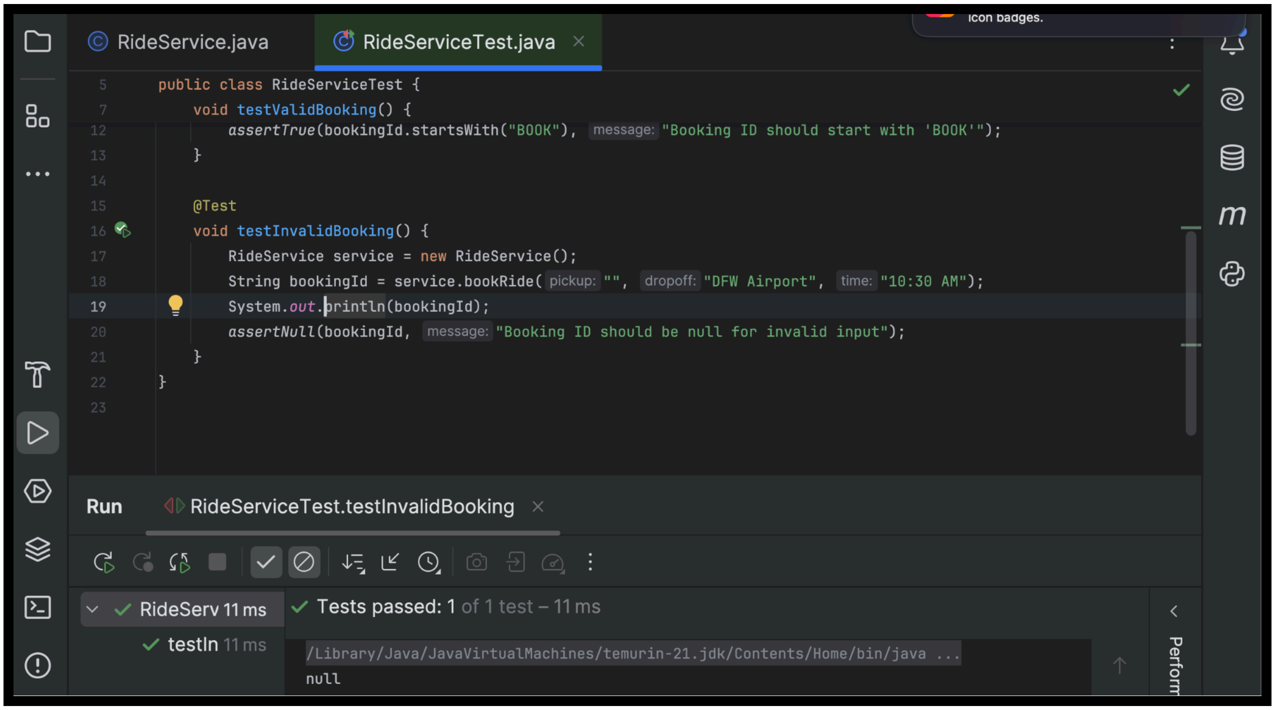
```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class RideServiceTest {
    @Test
    void testValidBooking() {
        RideService service = new RideService();
        String bookingId = service.bookRide("UTD
Campus", "DFW Airport", "10:30 AM");
        assertNotNull(bookingId, "Booking ID should not
be null for valid input");
        assertTrue(bookingId.startsWith("BOOK"),
"Booking ID should start with 'BOOK'");
    }

    @Test
    void testInvalidBooking() {
        RideService service = new RideService();
        String bookingId = service.bookRide("", "DFW
Airport", "10:30 AM");
        assertNull(bookingId, "Booking ID should be
null for invalid input");
    }
}
```



```
public class RideServiceTest {
    @Test
    void testValidBooking() {
        RideService service = new RideService();
        String bookingId = service.bookRide( pickup: "UTD Campus",  dropoff: "DFW Airport",  time: "10:30 AM");
        System.out.println(bookingId);
        assertNotNull(bookingId,  message: "Booking ID should not be null for valid input");
        assertTrue(bookingId.startsWith("BOOK"),  message: "Booking ID should start with 'BOOK'");
    }

    @Test
    void testInvalidBooking() {
        RideService service = new RideService();
        String bookingId = service.bookRide( pickup: "",  dropoff: "DFW Airport",  time: "10:30 AM");
        System.out.println(bookingId);
        assertNull(bookingId,  message: "Booking ID should be null for invalid input");
```

✓ Tests passed: 1 of 1 test – 17 ms

```
/Library/Java/JavaVirtualMachines/temurin-21.jdk/Contents/Home/bin/java ...
BOOK1763957570626
```



```
public class RideServiceTest {
    void testValidBooking() {
        assertTrue(bookingId.startsWith("BOOK"),  message: "Booking ID should start with 'BOOK'");
    }

    @Test
    void testInvalidBooking() {
        RideService service = new RideService();
        String bookingId = service.bookRide( pickup: "",  dropoff: "DFW Airport",  time: "10:30 AM");
        System.out.println(bookingId);
        assertNull(bookingId,  message: "Booking ID should be null for invalid input");
    }
}
```

✓ Tests passed: 1 of 1 test – 11 ms

```
/Library/Java/JavaVirtualMachines/temurin-21.jdk/Contents/Home/bin/java ...
null
```

As shown in the pictures, the tests tested to see whether the returned booking ID was valid meaning a successful call, or as null meaning an unsuccessful call.

Comparison of Work

Our work, Comet Courier, is a UTD focused ridesharing application designed *specifically* for students commuting to and from campus, many existing systems such as Uber, Lyft, and Waze Carpool target a broader audience with different regulations and business constraints. In this section, we will compare Comet Courier's requirements and design to several similar existing methods, namely Uber, Lyft, and Waze Carpool.

Target Audience and Scope: Uber and Lyft provide city-wide, on demand rides for the general public and operate as commercial transportation networks. Their apps are designed for a wide and diverse range of trip purposes (e.g. airport, restaurants, residential areas, etc.) and utilize dynamic pricing, based on supply and demand of rides [1]. Waze Carpool on the other hand, focuses explicitly on commuters and cost-sharing, emphasizing carpooling rather than professional drivers.

By contract, Comet Courier's scope is restricted only to UT Dallas students, verified via UTD email and Duo authentication. Trips are primarily between home and UTD, or between campus buildings and nearby locations, aligning more closely with campus-parking and commuting problems that Waze Carpool has also been targeting [2]. While the difference between Comet Courier and Uber and Lyft is clear, the important distinguishing factor between our project and Waze Carpool is the focus on UTD students, providing much more consistency in time of requests pooling around classes, ride paths always moving in or out of UTD, and the student body making up 100% of the customer basis.

Safety, Verification, and Reputation: In this project domain of transportation, safety is crucial. Uber requires multi-step driver screening, including background checks for impaired driving and violent offenses, and performs periodic re-screening [4]. Uber and Lyft both have rider safety toolkits that include emergency buttons, trip sharing, ETA sharing, and 24/7 incident support [5]. Similarly, both platforms rely on a two way rating system, where both the driver and rider can report their satisfaction with the other party based on a 1-5 star scale.

Comet Courier adopts some but not all of these mechanisms, it includes a rating and reporting system, a built in SOS button alerting both local and campus police, and a share trip feature. However, our system does not go into such depth with background checks as Uber and Lyft. This is allowed due to the key distinction in the user base. While Uber and Lyft allow for complete strangers whose profession is driving for these apps, our user base consists fully of fellow classmates, coworkers, and neighbors which campus identity as a primary safety filter. Our safety feature is community based, whereas Uber and Lyft are regulation and compliance based.

Matching and Pricing: Conceptually, our matching algorithm follows the same core idea as commercial ridesharing apps: given a set of ride offers and requests, we compute candidate matches based on distance and time-window constraints and then notify both parties so they can accept or decline. The key distinction is that Comet Courier operates on a campus-focused network where most trips share a common endpoint (UTD). This allows us to leverage simplified, near-optimal path planning and potential ride batching, since routes naturally converge to or from campus rather than being arbitrarily distributed across a city.

On the pricing side, we adopt a non-profit cost-sharing model rather than a commercial fare model. The app computes a per-mile cost derived from the current price of gas and an assumed fuel efficiency, then allocates this cost between rider and driver so that neither party earns a profit. In addition, the model can factor in extra time or distance incurred by non-optimal routing (e.g., small detours to pick up other students) and distribute that incremental cost proportionally. The result is a transparent, fair, and campus-friendly pricing scheme that focuses on sharing commuting expenses rather than generating income.

Overall, our method is a novel application of the ride-share framework, providing a safer and cheaper alternative to our competitors, and making the commuting experience more enjoyable.

Conclusion

Comet Courier represents a focused, campus-oriented approach to ridesharing that directly addresses the commuting challenges faced by UT Dallas students. Through careful requirements analysis, structured design artifacts, and a clear Incremental Process Model, our team established a foundation for a system that prioritizes usability, safety, and reliability while remaining lightweight and accessible for everyday student use. The functional and non-functional requirements, supported by detailed sequence diagrams, class diagrams, and architectural design, illustrate how the system maintains secure authentication, efficient ride matching, simple communication, and transparent cost-sharing.

Our comparison with existing platforms such as Uber, Lyft, and Waze Carpool highlights the unique advantages of a university-restricted user base, allowing Comet Courier to operate with simpler verification, community-driven safety, predictable commuter patterns, and a non-profit pricing structure. Additionally, the cost, effort, and scheduling estimates demonstrate that the project is feasible within a realistic development timeline and budget.

Overall, Deliverable #2 shows that Comet Courier is both technically achievable and well-designed for its intended environment. With clear requirements, structured modeling, and responsible planning, the system is positioned to evolve into a functional, secure, and student-focused transportation tool that enhances campus mobility and fosters a safer, more efficient commuting experience for the UTD community.

References

[1] Uber Technologies Inc., “How Uber’s dynamic pricing model works,” Uber Blog, 2025. [Online]. Available: https://www.uber.com/en-GB/blog/uber-dynamic-pricing/. [Accessed: 21-Nov-2025].

[2] “Waze carpooling app aims to fix campus parking crisis,” The Mossy Log, 08-Feb-2019. [Online]. Available: https://mossylog.org/2019/02/08/waze-carpooling-app-aims-to-fix-campus-parking-crisis. [Accessed: 21-Nov-2025].

[3] K. Leswing, “Google’s Waze is challenging Lyft and Uber with a new feature that enables people to share rides and split gas money,” Business Insider, 10-Oct-2018. [Online]. Available: https://www.businessinsider.com/google-waze-carpool-uber-lyft-rival-2018-10. [Accessed: 21-Nov-2025].

[4] Uber Technologies Inc., “Safety,” Uber, 2025. [Online]. Available: https://www.uber.com/us/en/safety/. [Accessed: 21-Nov-2025].

[5] Uber Technologies Inc., “In-app safety features for riders,” Uber, 2025. [Online]. Available: https://www.uber.com/br/en/ride/safety/rider-safety-features/. [Accessed: 21-Nov-2025].