
Pool - carpooling app

SW Engineering CSC648-848-05 Fall 2023

Milestone 2 | 10/12/2023

1. **Team Lead:** Zoe Rivka Panagopoulos | zpanagopoulos@sfsu.edu
2. **UX Expert:** Isiah Alfonso Paul-McGlothin | ipaulmcglothin@mail.sfsu.edu
3. **Frontend Lead:** Kristian Goranov | kgoranov@sfsu.edu
4. **GitHub Guru:** Kendrick Alexis Rivas | krivas2@sfsu.edu
5. **Backend Baron:** Phillip Diec | gdiec@sfsu.edu
6. **Database Duke:** Jonathan Sum | jsum@sfsu.edu
7. **Devops Officer:** Xuan Duy Anh Nguyen | anguyen96@sfsu.edu

History Table

Milestone	Date
M2V1	10/12/23

Table of Contents

- 1. Data Definitions** page 2
- 2. Prioritized Functional Requirements** page 3
- 3. UI Mockups and Storyboards (High Level Only)** page 8
- 4. High Level Database Architecture and Organization** page 12
- 5. High Level APIs and Main Algorithms** page 16
- 6. High Level UML Diagram** page 18
- 7. High Level Application Network Diagram** page 19
- 8. Identify Actual Key Risks for Your Project at this Time** page 20
- 9. Project Management** page 22
- 10. Detailed List of Contributions** page 23

1. Data Definitions

User: browsing capabilities only, doesn't require an account

Account: can be created by user, required for drivers and passengers

Driver: user with an account, registers vehicles and submits trips

Passenger: user with an account, must be ID verified, requests rides in many trips

Profile: associated with a user who has an account, visible to drivers and passengers within a common pool, and after a trip for a limited time

Crew: a common group of people who pool together

Car: registered to one driver, associated with many pools

Pool: created by a driver, can have passengers, can be converted into a crew

Ratings: for drivers and passengers to rate each other

Ride request: how passengers request to be admitted into a pool

Payment: preferred methods of payment which allow passengers and drivers to exchange compensation for pool services off platform

Notifications: the way through which users are notified of updates pertaining to on-platform actions they have taken

2. Prioritized Functional Requirements

Priority 1

1. User

- 2.1 A user shall create an account
- 2.2 A user shall register as a driver
- 2.3 A user shall register as a passenger
- 2.4 A user shall search for pools.

2. Profile

- 3.1 A profile shall belong to a passenger
- 3.2 A profile shall belong to a driver
- 3.3 A profile shall contain user details
- 3.4 A profile shall display ride history

3. Driver

- 4.1 A driver shall be Fastrak verified
- 4.2 A driver shall have one or more cars
- 4.3 A driver shall create pools
- 4.4 A driver shall have many crews
- 4.5 A driver shall send ride requests for their pool(s)

4. Passenger

- 5.1 A passenger shall join pools
- 5.2 A passenger shall have many crews.
- 5.3 A passenger shall send invites for crews.

5. Crew

- 6.1 A crew shall be created by one user
- 6.2 A crew member shall invite users to their crew
- 6.3 A crew invite shall be accepted

6.4 A crew invite shall be declined

6. Car

7.1 A car shall be associated with a driver

7.2 A car shall be registered

7. Pool

8.1 A pool shall have a description

8.2 A pool shall have a start time.

8.3 A pool shall have an end time.

8.4 A pool shall be one-time or recurring.

8.5 A pool shall have passengers.

8.6 A pool shall have ride requests.

8.7 A pool shall have a car

8.8 A pool shall have an start location

8.9 A pool shall have an end location

8.10 A pool shall be viewable by its passengers or driver

Ride Requests

10.1 A ride request shall be made by or for passengers

10.2 A ride request shall be made for a Crew

10.3 A ride request shall be made for a Pool

10.4 A ride request shall have seats

10.5 A ride request shall contain the Profile associated with the requesting Passenger(s)

10.6 A ride request shall contain the name associated with the requesting Crew

10.7 A ride request shall contain the description associated with the requesting Crew

10.8 A ride request shall have zero or one pickup location(s)

10.9 A ride request shall have zero or one dropoff location(s)

10.10 A ride request shall be accepted

10.11 A ride request shall be declined

Notifications

- 12.1 Notifications may be triggered by many actions.

Priority 2

1. User

- 2.5 A user shall be able to view their passenger ratings received
- 2.6 A user shall be able to view their driver ratings received
- 2.7 A user shall be able to view their passenger profile
- 2.8 A user shall be able to view their driver profile
- 2.9 A user shall be able to view their passenger feedbacks received
- 2.10 A user shall be able to view their driver feedbacks received

2. Profile

- 3.5 A profile shall contain payment details
- 3.6 A profile shall display the average rating

3. Car

- 7.3 A car shall have ratings

3. Driver

- 4.6 A driver shall have ratings

4. Passenger

- 5.4 A passenger shall have ratings

5. Crew

- 6.5 A crew creator shall be able to remove users from their crew
- 6.6 A user shall be able to leave a crew

6. Pool

- 8.11 A pool shall have an occurrence rate.
- 8.12 A pool shall have a total distance in miles

9. Ratings

- 9.1 A rating shall have a numerical value of 1-5
- 9.2 A rating for a driver shall be provided by a passenger belonging to a common pool
- 9.3 A rating for a passenger shall be provided by a driver belonging to a common pool
- 9.4 A rating for a car shall be provided by a passenger belonging to a common pool

10. Ride Requests

- 10.12 A ride request shall have seats
- 10.13 A ride request shall contain the name associated with the requesting Crew

Priority 3

User

- 2.11 A user shall have an emergency contact
- 2.12 A user shall be able to file a report ticket
- 2.13 A user shall be able to chat with support
- 2.14 A user shall be able to report any feedbacks
- 2.15 A user shall be able to delete their account

Profile

- 3.7 A profile shall display total distance traveled
- 3.8 A profile shall display total environmental impact
- 3.9 A profile shall display total number of crews with membership

Crew

- 6.7 A crew shall have a description

Ride Request

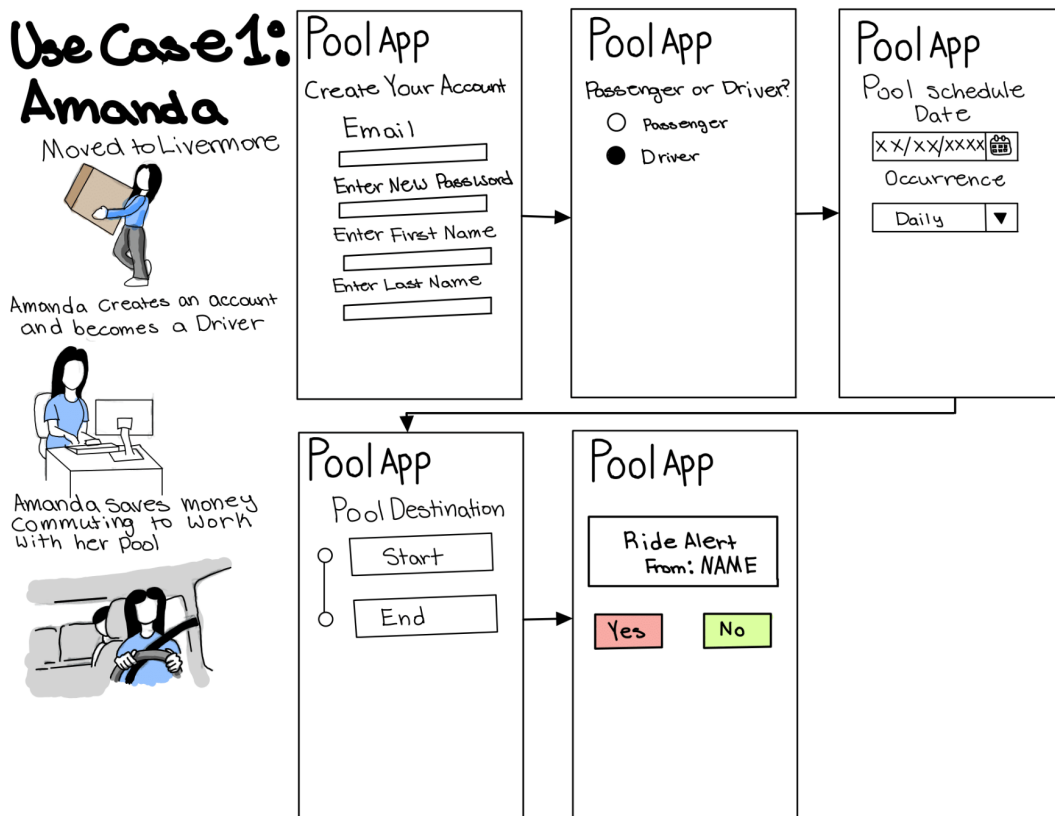
10.14 A ride request shall contain the description associated with the requesting Crew

12. Notifications

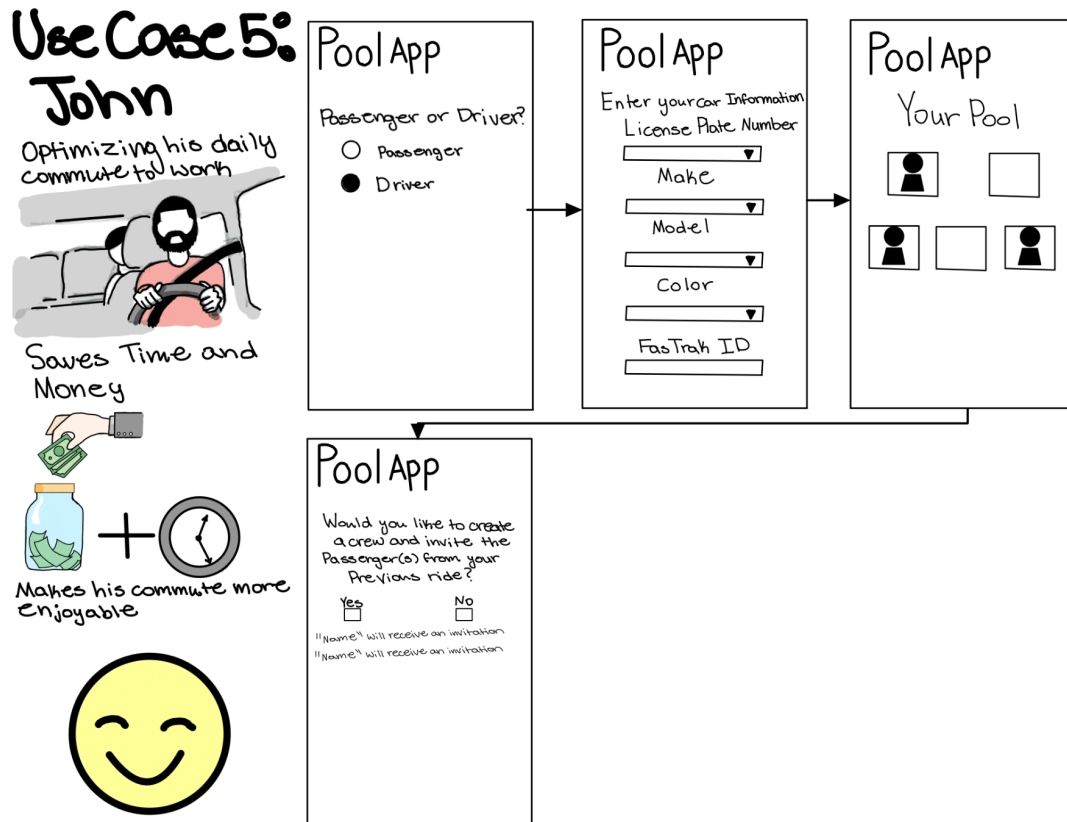
12.2 Notifications may be toggled on or off per account.

3. UI Mockups and Storyboards (high level only)

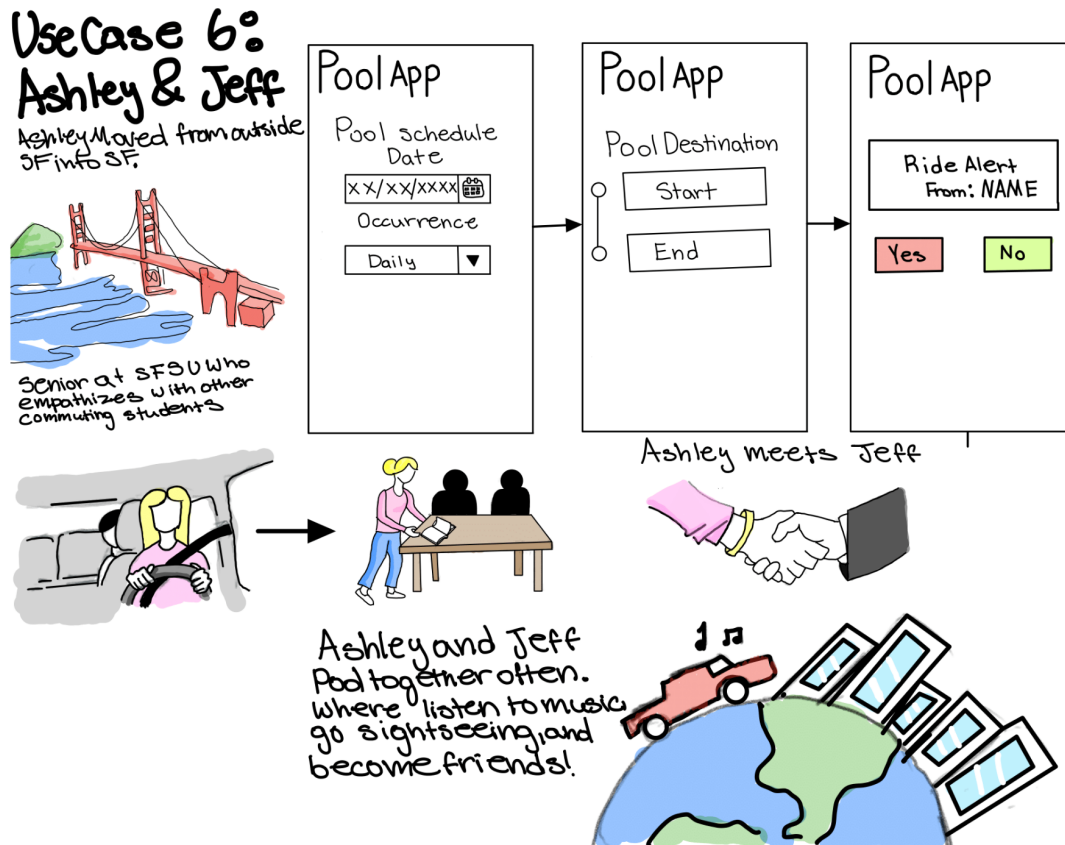
Use Case 1 - Recent college graduate Amanda moved to Livermore, CA for her job and chose to drive to avoid relying on public transportation. She discovered the Pool app through her friends, started offering daily rides to San Francisco, and received rider requests, allowing her to qualify for discounts on the highway and Bay Bridge tolls due to carpooling.



Use Case 5- San Francisco engineer John uses Fastrak to streamline his daily commute from San Jose, Saving time and money by funding his account for Bay Bridge tolls and express lanes. He has become a Fastrak verified driver on the Pool app, where he carpools with his coworkers in order to reduce toll costs and enhance his commute experience. They even formed a crew for easy coordination!



Use Case 6 - Ashley, a senior at SF State, uses the Pool app as a driver to offer rides to fellow students, alleviating expenses and helping newcomers navigate the city. She formed a two person crew with Jeff, a transfer student, who joined her daily pool route in the Outer Sunset, turning their once solitary commutes into a shared experience fostering friendship and community.



Use Case 12 - Tom, a Business professional, has recently moved to San Mateo, CA, and has found that commuting during peak hours of traffic to be too stressful. He then signed up to the Pool app as a passenger, and after searching for recurring pools, he was able to find a highly rated driver who accepts Venmo as a form of payment for the toll. Tom's daily commute now has gotten significantly better, and can now travel comfortably with a crew in a nice car.

Use Case 12: Tom

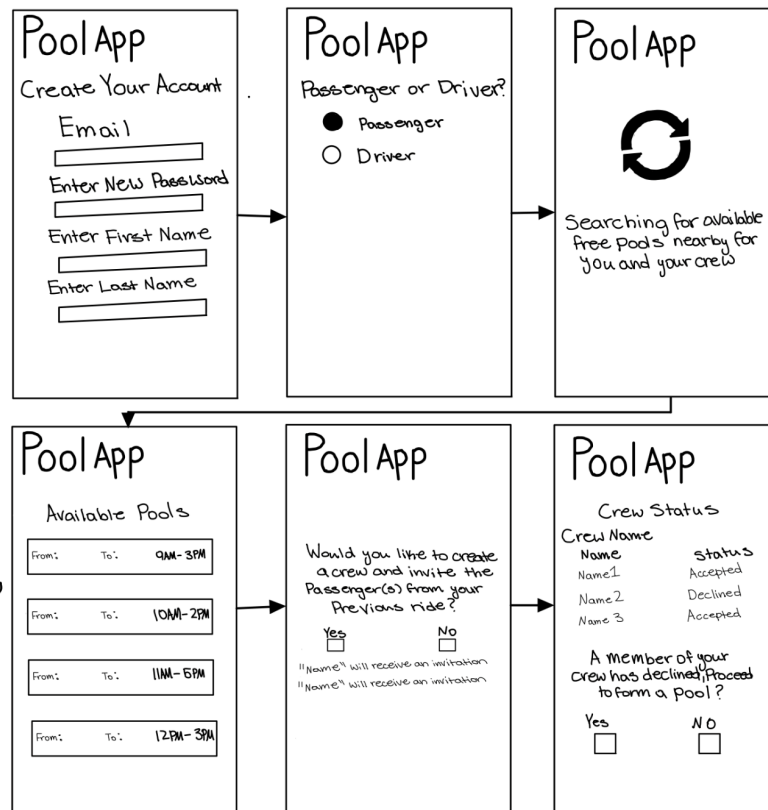
Tom is a working Professional who has recently relocated.



Hates Driving peak Hours and can never find parking



Prefers to be a Passenger, and wants to find a crew to commute to work with.



4. High level database architecture and organization

Database Requirements

User

- A user shall have a unique ID
- A user shall have a unique email address
- A user shall have a first name.
- A user shall have a last name.

Account

- An account shall be associated with one and only one user.
- An account shall have one unique ID.
- An account shall have notification preferences.

Passwords

- A password shall be tied to one and only one account
- A password shall have a unique ID
- A password shall have a timestamp associated with its last update
- A password shall have 0 or 1 previous password

Driver A driver represents an individual who offers Pool service to passengers

- A driver is one and only one user.
- A driver shall have a unique identifier
- A driver shall have a unique drivers license number
- A driver shall be associated with one or more cars
- A driver shall have a Fastrak ID

Passenger A passenger represents individual who rides with driver

- A passenger is one and only one user.
- A passenger shall have a unique identifier

Profile: A profile represents the information associated with each driver or passenger

A profile shall be associated with one and only one passenger or driver.

A profile shall have zero or one photo

Crew: A crew represents a group of passengers and driver of a recurring pool

A crew shall have one unique id.

A crew shall have a description.

A crew shall be associated with one driver.

A crew shall have as many passengers as car seats.

A crew shall have one and only one car.

Car: A car represents the vehicle a driver is registered to

- A car shall be associated with one and only one driver
- A car shall have a unique license plate number
- A car shall have a make.
- A car shall have a model.
- A car shall have a year.
- A car shall have a color.
- A car shall have zero or one photo
- A car shall have one or more passenger seats
- A car shall have wifi
- A car shall have air conditioning
- A car shall have zero or many child seats
- A car shall have wheelchair accommodations

Pool: A pool represents one driver and a zero to many passengers

- A pool shall have one unique ID.
- A pool shall have one description.
- A pool shall have one start time.
- A pool shall have one end time.
- A pool shall be one-time or recurring.
- A pool shall be created by one and only one driver.
- A pool shall have zero or many passengers.
- A pool shall have zero or one crew.
- A pool shall have many ride requests.

-
- A pool shall have one and only one car
 - A pool shall have one start location
 - A pool shall have one end location

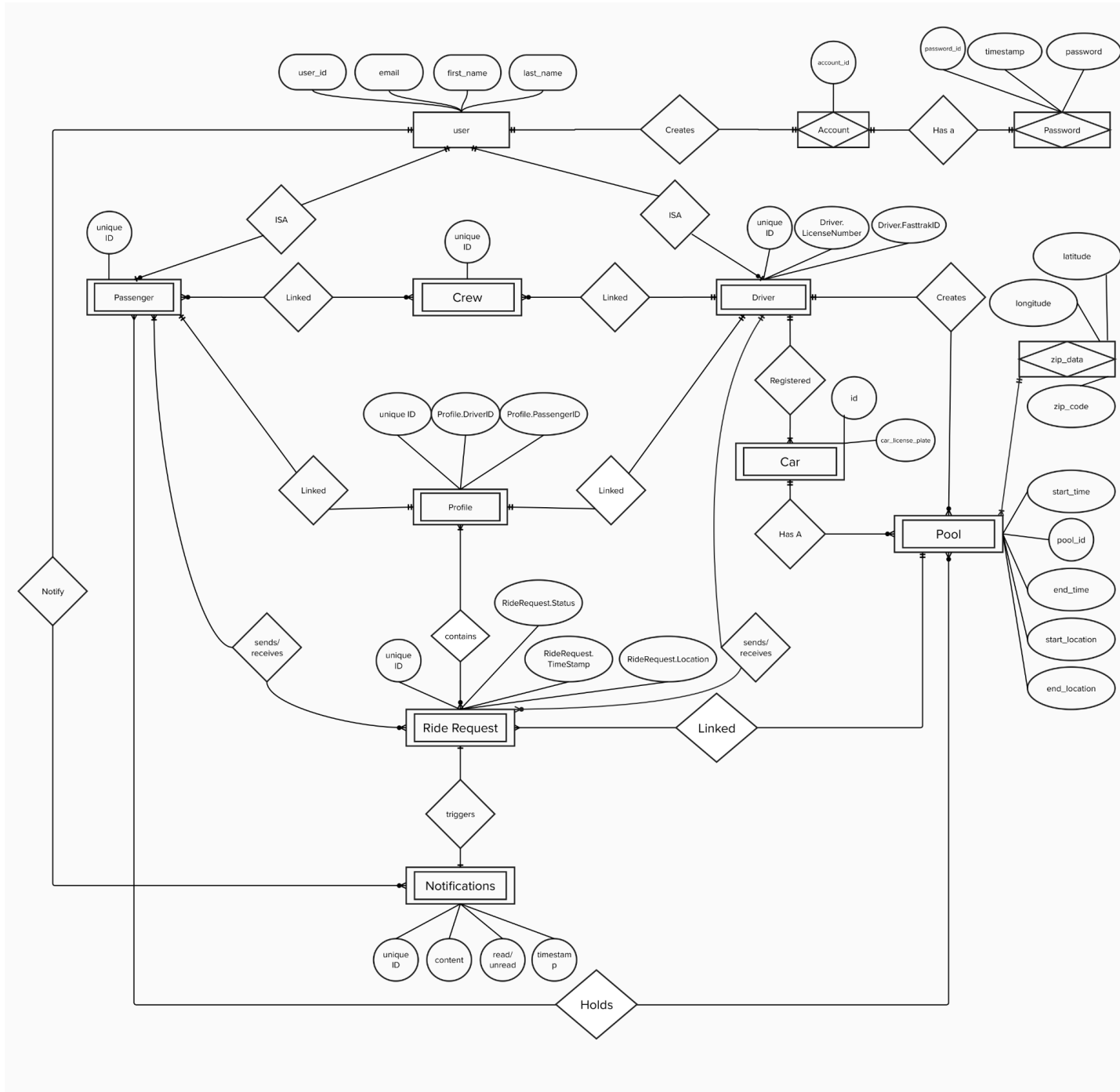
Ride request: Ride requests represent the request from users for a ride in the carpooling app

- A ride request shall have only one unique ID
- A ride request shall be made by one and only one passenger, crew, or driver
- A ride request shall be made for as many passengers as the pool allows OR one and only one crew
- A ride request shall be linked to one and only one Pool
- A ride request shall have zero or one pickup location
- A ride request shall have zero or one dropoff location
- A ride request shall have a dynamic status indicating if it is pending, accepted, declined, or canceled
- A ride request shall have only one timestamp indicating when the request was made

Notifications: Notifications represent the information or alerts triggered by various actions within the application

- A notification shall have only one unique ID
- A notification shall be associated with one and only one ride request
- A notification shall have only one timestamp indicating when it was generated

Entity Relationship Diagram (ERD)



Database Management System Choice (DBMS)

The choice for using MySQLWorkbench as our DBMS is due to the fact that our group has familiarity with the application due to having past experience from our database class.

5. High Level APIs and Main Algorithms

Hashing Methods

The system will deploy a SHA-256 algorithm to hash passwords so that each time one is entered, a new hash value is derived. The application will then check to see if the value entered matches that stored at the database level.

This will be implemented using the Maven commons-codec package (version 1.16.0), utilizing the simple encoder and decoder.

React Material UI

The Pool application will leverage select components from React's implementation of the Google Material UI (version 5), an open source component API.

Google maps JavaScript API or competitor

The Places API will allow the Pool user to search for a pool within a specific radius. Specifically, Pool will use Nearby Search for users to input their start location or destination and search for pool rides near their start or end point.

Axios

Axios is a popular Javascript library for making HTTP/HTTPS requests. It works both in-browser and in a Node.js environment. Axios is able to connect the frontend and backend by allowing our React application to retrieve or send data to the backend via HTTP/HTTPS requests, allowing the frontend to access databases, perform operations, and construct necessary response data to send back as an HTTP/HTTPS response. We've decided to go with Axios because the client simplifies the dynamic process of fetching and requesting data from the back end to the frontend.

Carpool Proximity Algorithm

Note: this algorithm may be replaced by Google Maps API in the medium term. To return carpools that are within a five mile radius of a user's supplied zip code, we created a table in the database that has all the zipcodes in California along with their respective longitude and latitude coordinates courtesy of <https://gist.github.com/erichurst/7882666>. The Haversine formula is then applied to this data to calculate the shortest distance between two points on the surface of a sphere:

$$d = 2r \sin^{-1} \left(\sqrt{\sin^2 \left(\frac{\Phi_2 - \Phi_1}{2} \right) + \cos(\Phi_1) \cos(\Phi_2) \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right)} \right)$$

Where r = radius of the earth, d = the distance between two points, λ_1, λ_2 is the latitude of each point respectively, and ϕ_1, ϕ_2 is the longitude of the two points respectively. First, the coordinates of the user provided zip code are fetched from the zipcode coordinate table and using an SQL query, the Haversine formula is applied to the coordinates of every pool (carpool) in the pool table to find those within a five mile radius.

6. High Level UML Diagram

```
classDiagram
    class User {
        +userId: Int
        +firstName: String
        +lastName: String
        +phoneNumber: String
        +email: String
        +setName(firstName: String, lastName: String) void
        +getName() String
        +setPhoneNumber(phoneNumber: String) void
        +getEmail() String
        +getUserByEmail(email: String) String
        +getUserByUserId(userId: Int) Int
    }
    class Account {
        +accountId: Int
        +getUser() User
        +setUser(user: User) User
        +getPassword() Password
        +setPassword(password: Password) Password
    }
    class Password {
        +passwordId: Int
        +password: String
        +previousPassword: String
        +timestamp: String
        +resetPassword(newPassword: String) boolean
        +setPassword(newPassword: String) void
    }
    class Passenger {
        +isPassenger: Boolean
        +rideRequest(poolId: Int, request: RideRequest) void
        +rideLookup(zipcode: Int) Pool
        +createCrew(userId: Int) Crew
    }
    class Driver {
        +isDriver: Boolean
        +fastrakId: Int
        +driversLicense: String
        +createCrew(userId: Int) Crew
        +registerCar(carId: Int, licenseNumber: String) void
        +createPool(userId: Int) Pool
        +verifyDriverLicense(licenseNum: String) Boolean
        +verifyFastrakID(fastrakId: Int) Boolean
    }
    class Car {
        +carId: Int
        +licenseNumber: String
        +make: String
        +model: String
        +year: Int
        +color: String
        +photo: Image
        +seatCount: Int
        +wifi: Boolean
        +airConditioning: Boolean
        +wheelchairAccessible: Boolean
        +childSeats: Int
    }
    class Pool {
        +poolId: Int
        +description: String
        +startTime: String
        +endTime: String
        +startLocation: String
        +endLocation: String
        +recurring: Boolean
        +numberOfUsers: Int
        +poolList: String
        +addPool(userId: String) void
        +sendRideRequest(userId: Int) void
        +acceptRideRequest(userId: Int) void
        +cancelRideRequest(userId: Int) void
        +declineRideRequest(userId: Int) void
        +assignCar(car: Car) void
        +createCrew(pool: Pool) void
        +inviteCrew(crew: Crew) void
        +invitePool(profileId: Int) void
    }
    class Notification {
        +notificationId: Int
        +timestamp: String
        +sendNotification() void
    }
    class Profile {
        +profileId: Int
        +picture: Image
        +uploadPicture(picture: Image) void
        +seeRating(profileId: Int) Int
    }
    class Crew {
        +crewId: Int
        +description: String
        +numberOfUsers: Int
        +inviteUser(userId: String) String
        +displayUsers() String
    }
    class RideRequest {
        +rideRequestId: Int
        +pickupLocation: String
        +dropoffLocation: String
        +requestStatus: Boolean
        +timestamp: String
        +setPickupLocation(string: String) void
        +getPickupLocation(string: String) String
        +setDropoffLocation(string: String) void
        +getDropoffLocation(string: String) String
        +changeStatus(status: Boolean) Boolean
    }
    class Ratings {
        +ratingId: Int
        +setRating(rating: Int) void
    }
    User --> Account
    User --> Password
    User --> Passenger
    User --> Driver
    User --> Profile
    Account --> Password
    Passenger --> Pool
    Driver --> Car
    Driver --> Pool
    Driver --> Crew
    Car --> Pool
    Pool --> RideRequest
    Pool --> Profile
    Profile --> RideRequest
    Profile --> Ratings
    RideRequest --> Ratings
    Notification --> Profile
    Ratings --> Profile
```

7. High Level Application Network Diagram



8. Identify actual key risks for your project at this time

1. **Risk:** Team Lead being abroad from the evening of 10/6-10/14, and team lead's absence putting the delivery team off schedule, resulting in falling behind on key deadlines.

a. **Mitigation Tactic:**

- i. Trying to get ahead of schedule so that key deliverables are complete or very close to completion prior to 10/6.
- ii. Identifying windows of time during which time zones will overlap so that team lead can continue supporting the team from afar – even if in a reduced capacity.
- iii. Designating a temporary lead who has been briefed with directives and can provide adequate coverage in Team Lead's absence.

2. **Risk:** Midterms might impact schedules and capacity by introducing competing priorities.

a. **Mitigation Tactic:**

- i. Find alternative times to meet which may work better for people during this time of the semester.
- ii. Setting tighter internal deadlines aimed at keeping the team not only on track but ahead of deadlines.

3. **Risk:**

- a. Relying on one individual who specializes in a certain area creates a dependency on one person who could have an emergency or suddenly become unavailable. This could put us behind schedule, should such an incident occur.
- b. Switching team leads is expensive (it takes time to ramp up to a new part of the codebase) and when a specialized team member is put on an area where they are less specialized, another team member may need to be over utilized in order to help fill that knowledge gap.

c. **Mitigation tactics:**

- i. Make sure everyone is comfortable with all the technology being used to minimize the likelihood of dependencies on people with specialized skills.
- ii. Share knowledge so that everyone becomes familiar with different corners of the codebase.
- iii. Introduce a process for briefing each other on work completed, like an internal sprint demo at the end of each milestone.

4. **Risk:** Challenges associated with initial integrations of all our frameworks pose a risk to timeline if we end up needing to go back on our decisions regarding frameworks and services and either troubleshoot, reconfigure, or start from scratch.

a. **Mitigation tactics:**

-
- i. Scale back as needed if challenges become too large to overcome. Find ways to simplify our systems approach to accommodate the same or a lighter weight outcome.
 - ii. Proactively create a fallback plan to reduce time needed to recalibrate in the event this happens.
 - 5. Risk:** Another entity already exists called Pool. This could result in us being presented with a cease and desist order.
 - a. Mitigation tactic:**
 - i. Make it known that this is for a school project, not a financial venture to reduce our threat factor.
 - ii. If no other options are viable, change the application name.
 - 6. Risk:** We inadvertently populate our database with images or other content we don't have the right to use.
 - a. Make it known that this is for a school project, not a financial venture to reduce as our threat factor.
 - b. Using our own photos and files.

9. Project Management

Team04 utilizes [Jira Cloud](#) for project management and documenting requirements. It is where team members are assigned tasks, and where they may reference the specifications for said tasks. It is also a progress tracking tool which may be utilized by anyone who wishes to see what the team is working on during any given moment in time.

Our Jira project (POOL) is configured using the agile model of time-boxed sprints. Most tickets are related to a larger initiative documented as epics. While scrum defines epics as being so large that they may not be completed within a single sprint, we do not use this definition, and epics in the POOL project may be completed within one sprint.

Jira is used in tandem with a suite of other tools, including Google docs and Discord. The “[Working doc](#)” is where forthcoming (and past) agendas for team co-working sessions and meetings may be found. Forthcoming agendas are also pinned in the team Discord channel for easy access, along with all milestone docs, the team [Zoom](#) (for co-working sessions and meetings), the team [Mural](#) (for virtual collaborative work), and other important team specific information and resources.







The team meets daily most weeks, and minimally once a week (there have only been two weeks this semester where the team only met once). Team availability is typically collected via Zoom poll, and asynchronous assignments are assigned on group calls and in Discord.

10. Detailed List of Contributions







Team Contribution Scores

- Isiah: 10
- Jonathan: 10
- Kendrick: 7
- Kristian: 10
- Phillip: 10
- Xuan: 10

Database Requirements

-  Isiah: asynchronously wrote draft db requirements for the user and account entities
-  Jonathan: asynchronously wrote draft db requirements for the car and pool entities, revised, refined, and finalized db requirements
-  Kendrick: asynchronously wrote draft db requirements for the passenger and driver entities
-  Kristian: asynchronously wrote draft db requirements for the profile and crew entities
-  Phillip: asynchronously wrote draft db requirements for the ride request and notification entities
-  Zoe: assigned db requirement categories to team by entity and supported Jonathan in the revision process

Data Definitions

-  Isiah: worked collaboratively to refine data definitions on a team group call
-  Jonathan: worked collaboratively to refine data definitions on a team group call
-  Kendrick: worked collaboratively to refine data definitions on a team group call
-  Kristian: worked collaboratively to refine data definitions on a team group call
-  Phillip: worked collaboratively to refine data definitions on a team group call
-  Zoe: organized and facilitated team group call during which this was worked on

Prioritized Functional Requirements

- ☒ Isiah: worked collaboratively to prioritize functional requirements on a team group call
- ☒ Jonathan: worked collaboratively to prioritize functional requirements on a team group call
- ☒ Kendrick: worked collaboratively to prioritize functional requirements on a team group call
- ☒ Kristian: worked collaboratively to prioritize functional requirements on a team group call
- ☒ Phillip: worked collaboratively to prioritize functional requirements on a team group call
- ☒ Zoe: organized and facilitated group working call during which functional requirements were prioritized

Prioritized Use Cases

- ☒ Jonathan: read through half of use cases to identify P1, P2, and P3 functional requirements referenced in them and marked use case as P1 if it only contained P1 requirements
- ☒ Kendrick: read through half of use cases to identify P1, P2, and P3 functional requirements referenced in them and marked use case as P1 if it only contained P1 requirements
- ☒ Zoe: organized and facilitated group working call during which use cases were prioritized

UI Mockups & Storyboards

- ☒ Isiah: worked collaboratively to outline storyboards by identifying actions (related to functional requirements) and motivators from prioritized use cases, created all mockups and storyboards based on early artifacts, owned final delivery of these materials
- ☒ Phillip: worked collaboratively to outline storyboards by identifying actions (related to functional requirements) and motivators from prioritized use cases, drafted wireframes for Isiah to reference

-
- ☒ Zoe: organized and facilitated group working calls during which these activities occurred

Entity Relationship Diagrams

- ☒ Jonathan: owned the ERD from draft to final phase with support from Zoe
- ☒ Zoe: supported Jonathan as needed

High Level APIs and Main Algorithms

- ☒ Kendrick: collaboratively wrote the “Google maps JavaScript API or competitor” section with Zoe
- ☒ Kristian: collaboratively wrote the “Hashing Methods” and “React Material UI” sections with Zoe and independently wrote the “Location Based Search” section
- ☒ Phillip: independently wrote the “Axios” section
- ☒ Zoe: supported Kristian and Kendrick in writing their respective sections

UML Diagram

- ☒ Isaiah: worked collaboratively on first draft
- ☒ Jonathan: worked collaboratively on first draft and fully owned all future states of the diagram, including final delivery
- ☒ Phillip: worked collaboratively on first draft
- ☒ Zoe: facilitated coworking sessions and supported Jonathan as needed

Network Diagram

- ☒ Kristian: worked collaboratively on first draft
- ☒ Jonathan: worked collaboratively on first draft
- ☒ Phillip: worked collaboratively on first draft
- ☒ Kendrick: owned all future revisions with exception to final version
- ☒ Zoe: facilitated coworking sessions, supported Kendrick as needed, made final round of revisions

Identified Risks

- ☒ Isaiah: worked collaboratively on risks on a team coworking call

-
- ☒ Kendrick: worked collaboratively on risks on a team coworking call
 - ☒ Zoe: facilitated coworking call during which this was worked on

JPA Integration

- ☒ Kristian: did preliminary research on JPA vs JDBC and supported Phillip as needed
- ☒ Phillip: owned and implemented JPA integration
- ☒ Zoe: facilitated coworking calls during which this was worked on

React Integration and First Frontend Iteration

- ☒ Kristian: integrated existing code base with new React app and owned the first iteration of Pool's frontend and delivered draft pages inclusive of early working components
- ☒ Zoe: wrote associated Jira tickets and supported as needed

Classes and Methods

- ☒ Jonathan: played a consulting role during the creation of classes, leveraging his db knowledge to help guide Phillip in ensuring alignment between the db and backend
- ☒ Kristian: supported and contributed to the creation of methods, wrote all queries required by methods
- ☒ Phillip: owned and implemented Classes and methods
- ☒ Zoe: facilitated coworking calls during which this was worked on

Connecting Frontend and Backend

- ☒ Kristian: owned the implementation and delivery of endpoints
- ☒ Phillip: set up initial connection and worked collaboratively with Kristian throughout endpoint creation process
- ☒ Xuan: integrated app with Axios
- ☒ Zoe: facilitated coworking calls during which this was worked on, wrote associated tickets

Documentation

-
- ☒ Kendrick: drafted documentation for MacOS local setup in GitHub
 - ☒ Zoe: provided guidance and supported Kendrick as needed

Populating the Database

- ☒ Phillip: worked collaboratively with Kristian to populate the db's first tables (user, account, password, pool), ensured tables may be created, updated, and populated using JPA
- ☒ Kristian: worked collaboratively with Phillip to populate the db's first tables (user, account, password, pool, plus zip codes)

Final Frontend Iteration

- ☒ Isiah: provided images for use by various site pages
- ☒ Xuan: fully owned frontend revisions and refinement including delivery of final product
- ☒ Zoe: wrote tickets and played a consulting role for requirements

Deployment

- ☒ Xuan: improved deployment process by moving the image tagging and updating job from a local Docker process to a cloud based process, shifted app to a new virtual machine to accomplish this and ensured the VM referenced both the back and frontend
- ☒ Zoe: provided guidance and supported Kendrick as needed