Pool - carpoolwith.me

Final Project for SW Engineering CSC648-848-05 Fall 2023

Milestone 5 | 12/07/2023

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

History Table

Milestone	Date
M5	**12/07/23**
M4V2	**12/07/23**
M4V1	**11/30/23**
M3V2	**11/25/23**
M3V1	**11/02/23**
M2V2	**11/02/23**
M2V1	**10/12/23**
M1V2	**10/8/23**
M1V1	**9/21/23**

Table of Contents

1. Product Summary

Pool: carpoolwith.me

Pool is a carpooling website which provides a platform for drivers to post a “pool” and for passengers to join them.

Unlike other carpooling platforms, Pool helps drivers highlight their FasTrak account status. This ensures that prospective passengers know their driver will never have to avoid toll routes in their pool and can make use of time-saving fast lanes during peak traffic hours. Passengers looking for a guarantee that their pool is FasTrak equipped need only to check the driver’s FasTrak verification status on their profile.

What further sets Pool apart from its competitors is the unique ability for drivers and passengers who have “pooled” together in the past to form a crew. Drivers may choose to post a pool publicly – available for anyone to join – or privately, only available to members of their crew. This feature allows users to pool together with familiar faces, adding stability and predictability to their commute, wherever it takes them.

Prioritized Functional Requirements

Priority 1

1. User

- 1.1 A user shall create an account
- 1.2 A user shall register as a driver
- 1.3 A user shall register as a passenger
- 1.4 A user shall sign in
- 1.5 A user shall log out

2. Profile

- 2.1 A profile shall belong to a passenger
- 2.2 A profile shall belong to a driver
- 2.3 A profile shall contain user details
- 2.4 A profile shall be viewed by the user it is associated with

3. Driver

3.1 A driver shall be Fastrak verified
3.2 A driver shall create zero or many pools
3.3 A driver shall have zero or many crews
3.4 A driver shall have a driver's license
3.5 A driver shall view the pools they posted
3.6 A driver shall view the crews they are a member of
3.7 A driver shall leave the crews they are a member of

4. Passenger

4.1 A passenger shall join many pools
4.2 A passenger shall have many crews
4.3 A passenger shall view the pools they have joined
4.4 A passenger shall view the crews they are a member of
4.5 A passenger shall leave the crews they are a member of
4.6 A passenger shall leave the pools they are a member of

5. Crew

5.1 A crew shall be created by one user
5.2 A crew shall have a description
5.3 A crew shall have members

6. Pool

6.1 A pool shall have a description
6.2 A pool shall have a start time.
6.3 A pool shall have passengers.
6.4 A pool shall have one driver
6.5 A pool shall have a start location
6.6 A pool shall have an end location
6.7 A pool shall be deleted by its driver

Pool - carpooling app

SW Engineering CSC648-848-05 Fall 2023

Milestone 1 V2 | 10/08/2023

1. **Team Lead:** Zoe Rivka Panagopoulos | zpanagopoulos@sfsu.edu
2. **GitHub Guru:** Isiah Alfonso Paul-McGlothin | ipaulmcglothin@mail.sfsu.edu
3. **Release Manager:** Kristian Goranov | kgoranov@sfsu.edu
4. **Database Duke:** Kendrick Alexis Rivas | krivas2@sfsu.edu
5. **Backend Baron:** Phillip Diec | gdiec@sfsu.edu
6. **Frontend Lead:** Jonathan Sum | jsum@sfsu.edu

History Table

Milestones	Date
M1V2	**10/8/23**
M1V1	**9/21/23**
M1V1	**9/21/23**

Executive Summary

We're all students and work, some of us even do both. Having to drive from San Jose or the East Bay all the way to San Francisco can be less burdensome with an app like Pool. We all commute and are all looking for an alternative to the pricey big name apps like Uber and Lyft. Even in places where public transit is relatively connected, a more private transit experience is strongly preferred to buses and trains. For drivers, Pool would be much easier on our wallets by alleviating gas and toll prices. Uber, Lyft, and public transportation also lack community building elements that we all crave as busy people with little time for socializing.

We're a team of full stack web developers ready to revolutionize the commuting experience. As CS students and workers, we've all experienced the pain points of commuting using apps like Lyft and Uber, of driving during peak traffic, and of navigating the complex and sometimes unpredictable web of public transit. These experiences are at the forefront of the conception, design, and development of Pool.

Pool makes it convenient for people to commute to work – even during peak traffic, without relying on public transit or traditional ridesharing apps. By using Pool to carpool with fellow community members, passengers and drivers alike reduces commuter costs while limiting environmental impact and building community in the process. Plus, it's just nice to have someone else drive sometimes; Pool allows users to relax in the passenger's seat without the price tag of a Lyft or Uber.

Other ridesharing apps don't guarantee access to HOV and low-traffic lanes requiring FasTrak. Pool verifies all drivers for FasTrak subscriptions. Most carpooling apps don't break down the cost for you, but Pool will break down the costs of tolls and fuel per passenger. Pool is laser focused on the commuting experience and empowers users to ride to work with a trusted and familiar group of passengers, and a verified FasTrak driver able to cruise through traffic using HOV lanes. After work, expand your transit experience with your carpooling crew; go grocery shopping, get dropped off at the gym, or head home from sports practice together – all while paying less, having fun, and reducing your carbon footprint.

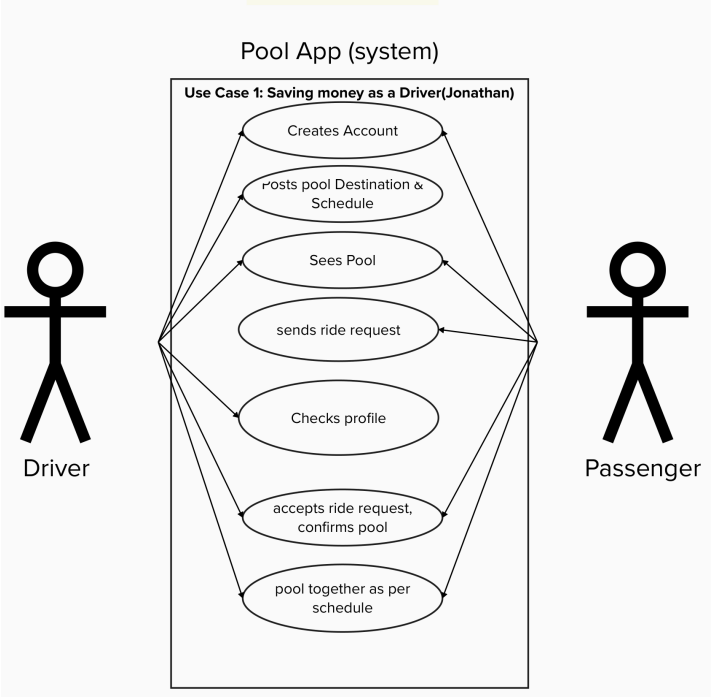
Main Use Cases

Use Cases: Drivers

Use Case 1: Saving money as a driver

Actors: Amanda

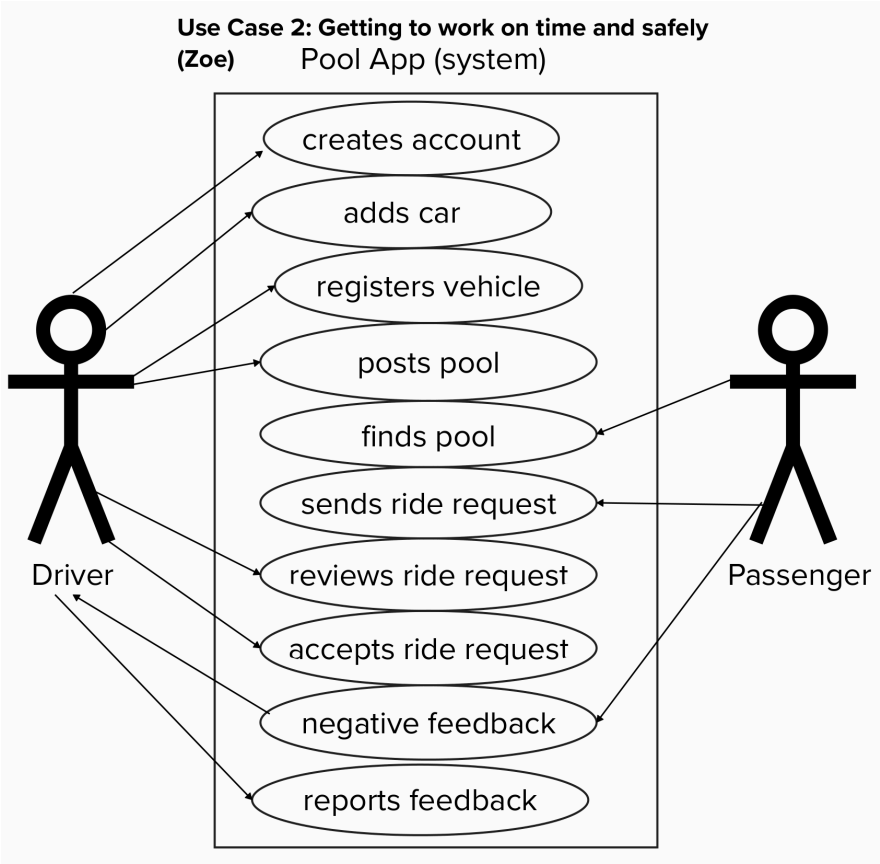
Amanda is a recent college graduate who has moved to Livermore, CA for her new job. Amanda prefers not to have to rely on public transportation to move around the bay, so she drove her car to California. Her commute requires her to cross the Bay Bridge into San Francisco which costs money from the toll system. Additionally, there are toll lanes on the highway down from Livermore that require additional fees. Amanda discovers Pool by talking to some of her friends that consistently use Pool for their daily commute. Just like her coworkers, Amanda is able to find passengers that also live in Livermore that commute daily to San Francisco in a "pool". Amanda creates an account and posts pools for each week day from Livermore to San Francisco. Amanda makes some small adjustments to pickup time and end location, and soon receives rider requests from passengers who wish to join her pool. Since there are multiple passengers in Amanda's vehicle, she qualifies for discounts not only on the highway, but for crossing the Bay Bridge as well.



Use Case 2: Getting to work on time and safely

Actors: Alex

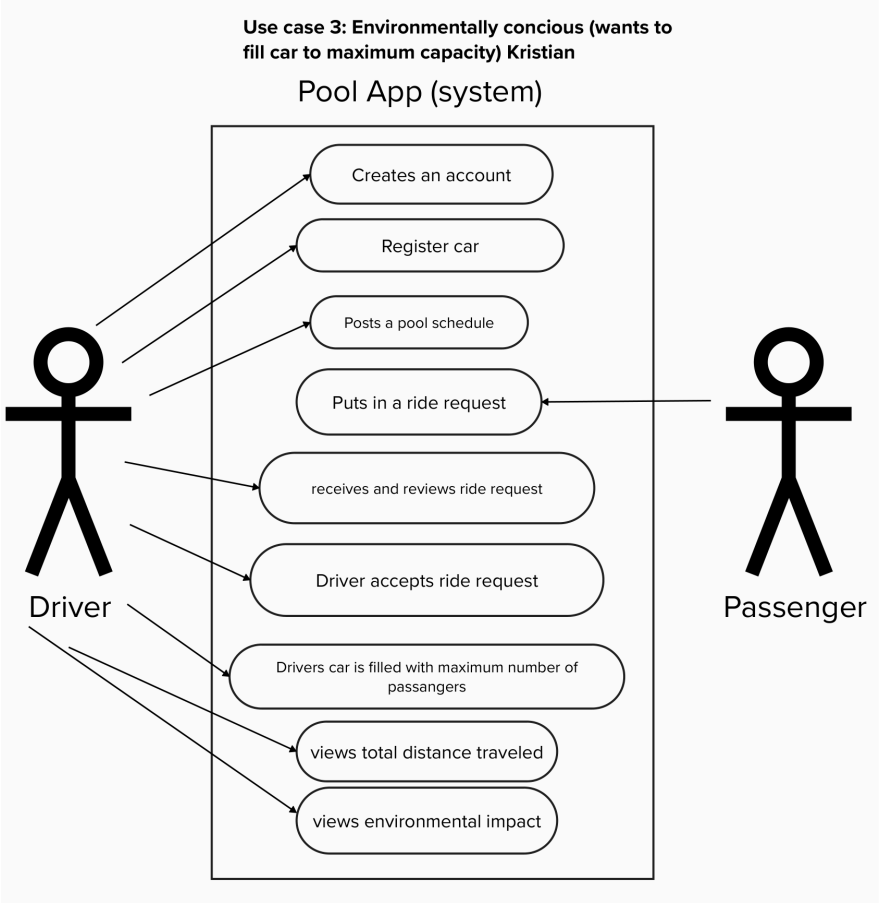
Alex loves driving and has faced harassment on public transit, so choosing to drive to work is a no brainer – she'll even pay a premium for this choice (tolls, gas, parking, etc). But getting to work on time is a major problem as a commuter traveling from Marin County into San Francisco, over the Golden Gate. Alex has tried carpooling before, so she decides to try out Pool as a driver in her own car, where she can better control her environment and the passengers she commutes with by reviewing the profiles of prospective passengers in ride requests before accepting or declining them. And when she receives transphobic or otherwise inappropriate feedback from passengers, she can view it before it becomes public and easily report it. On days where Alex doesn't feel like dealing with people, it's easy to toggle off notifications so she can review ride requests when she's ready.



Use Case 3: Environmentally Conscious

Actors: Carol

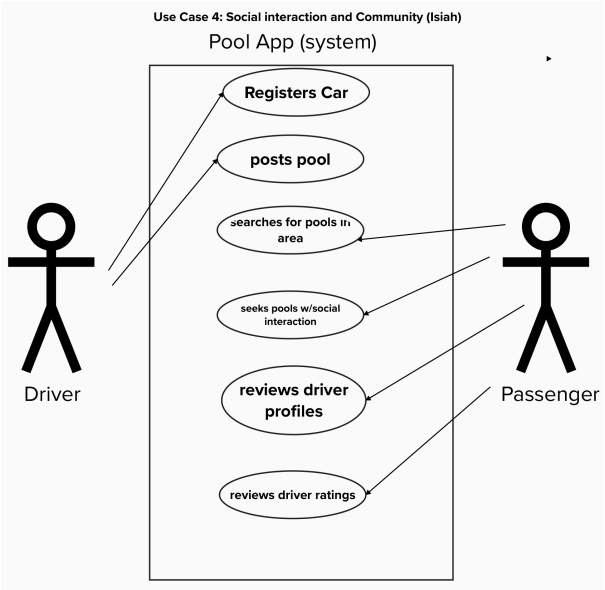
Carol has been seeing a lot of news articles about the acceleration of climate change and is wondering what she can do to limit her environmental footprint. But she likes her car and enjoys driving to work. So she creates a password protected account on Pool. There, she can post pools that align with her commuting schedule and receive ride requests from prospective passengers who are going in the same general direction. Sometimes, Carol drives a little out of her way to pick up and drop passengers off as per their ride requests, but it's worth it because Carol prefers to have a full car to reduce her impact on the environment. Plus, she loves seeing her total distance traveled and environmental impact from all of her past rides; it reaffirms her choice to be a driver on the Pool app. When Carol thinks of other features that could create more in-app incentives for lessening environmental impact, she files a report ticket for the Pool customer support team with her suggestions.



Use Case 4: Social interaction and community

Actors: Sarah

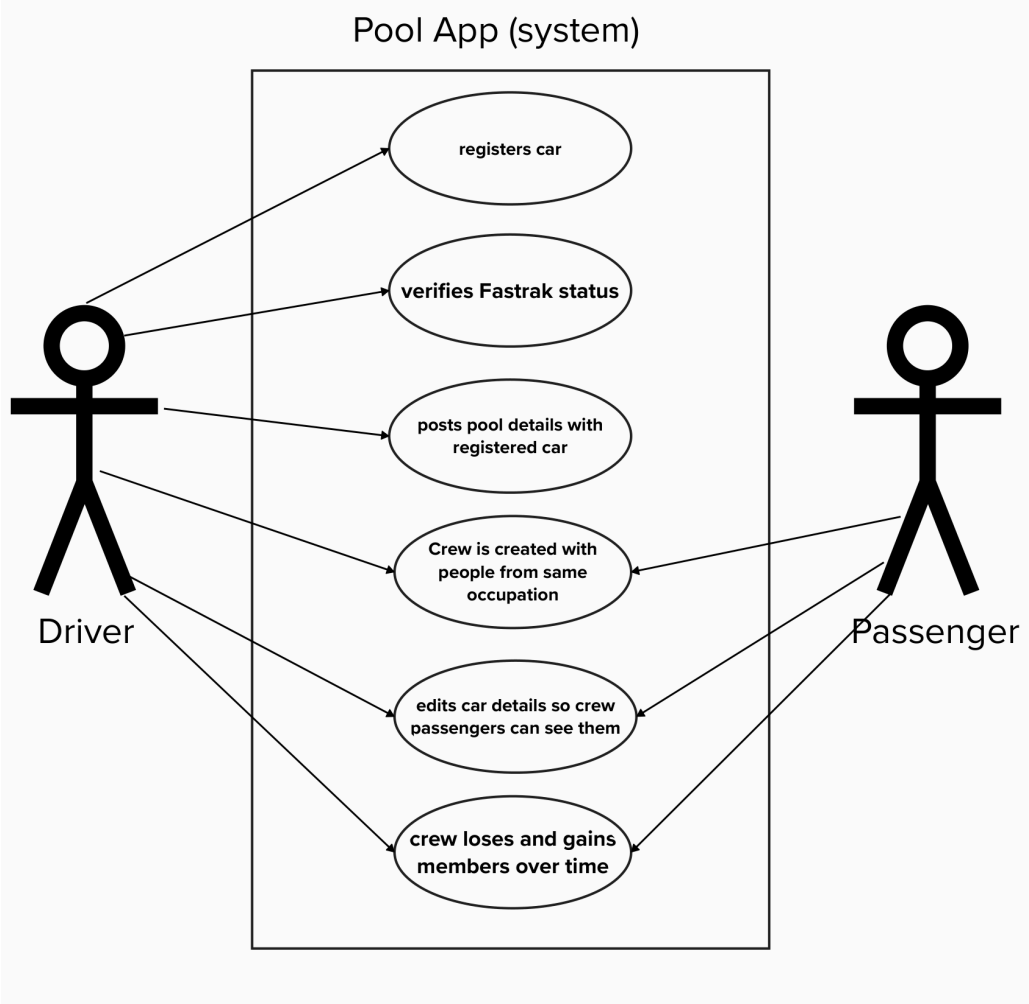
Sarah is a retired teacher in her early 60's, living in a quiet suburban neighborhood in the Bay Area. She no longer commutes to work and enjoys her retirement, but she frequently needs to cross the Bay Bridge and navigate the city for her errands, social outings, and medical appointments. Sarah's primary concern is the hassle and stress of her regular commutes. Sarah is not looking to save money but to simplify her errands and improve her overall quality of life. Sarah is seeking a carpooling app that can help her find companions for her trips into the city. She hopes to connect with individuals who share her interests or have similar destinations. This way, she can have enjoyable conversations and company during her commutes, making her trips more pleasant and less stressful. Safety and reliability are vital to her, ensuring that she shares rides with trustworthy individuals. In her research, she learns that the Pool app has passenger and driver ratings, which entices her to try it out. Because Pool does not require her to create an account to search for available pools in her area, she starts by doing this. When Sarah tells her daughter that Pool has an emergency contact feature, her daughter feels much better about the prospect of her creating an account as a passenger, knowing that she'll be contacted by phone and email in the event of any unforeseen circumstances. And in the event Sarah deems the app too difficult to navigate, she can try chatting with support or can always just delete the account.



Use Case 5: Reduce commuting time and having Fastrak

Actors: John

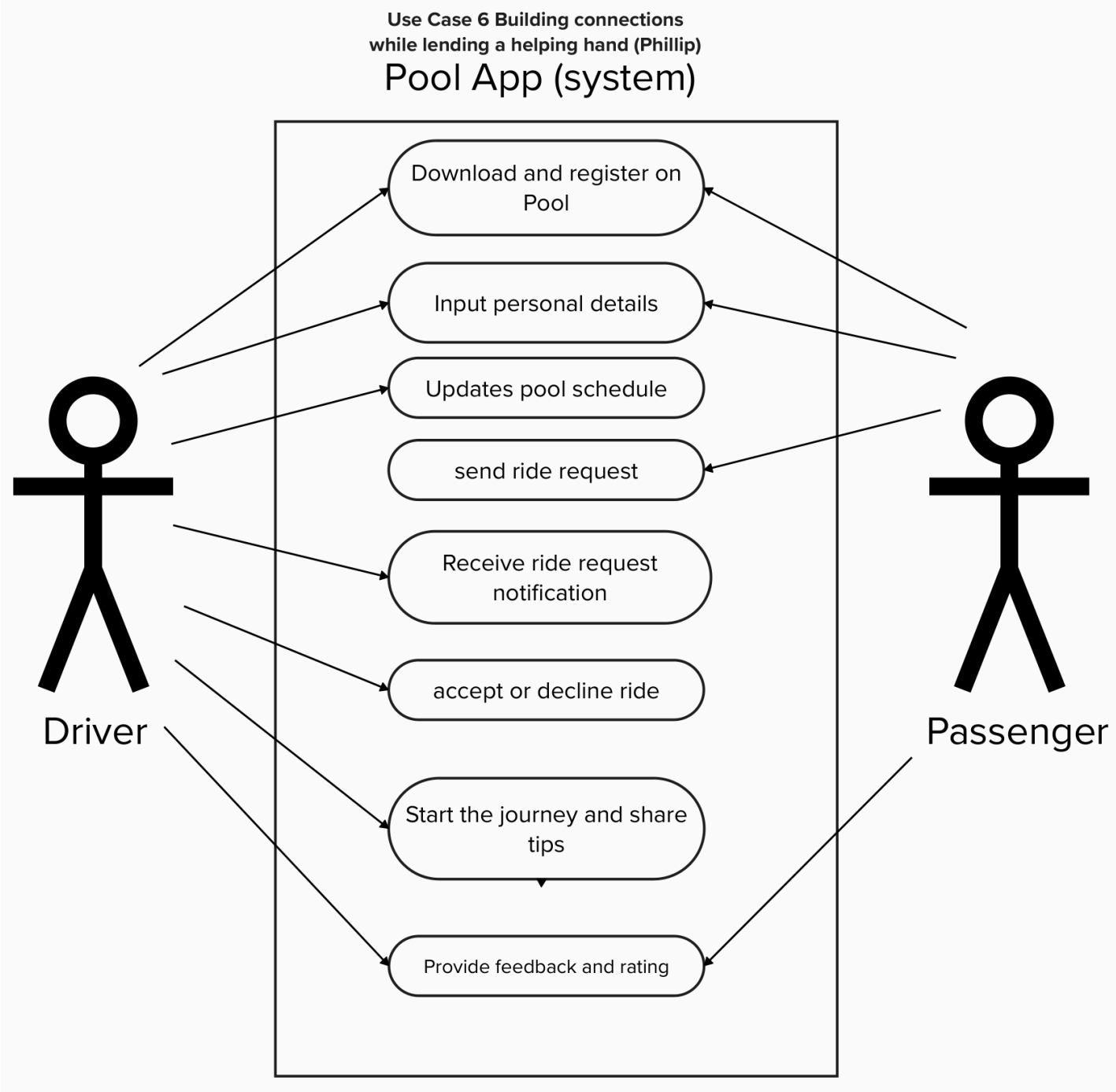
John, a software engineer in San Francisco, uses Fastrak to optimize his daily commute from San Jose. To save time and money, he ensures his Fastrak account is funded, allowing for smooth toll payments on the Bay Bridge and express lanes. John participates in carpooling through the Pool app as a Fastrak verified driver, sharing his registered vehicle with co-workers from the same area in a pool, which not only reduces his toll expenses but also makes his commute more enjoyable. John and his co-workers recently created a crew from their regular pool commute to make traveling together using the Pool app even easier.



Use Case 6: Building connections while lending a helping hand

Actors: Ashley, Jeff

Ashley is a senior at San Francisco State who has been living in the city for over three years. Originally from a suburban area, Ashley became acquainted with San Francisco's dynamics with its unique neighborhoods and city life. Her residence in the Outer Richmond area allows her to quietly live outside of the busier parts of the city and her commute time to the university isn't half bad. Ashley has been using Pool for over a year now as a driver, and the app not only helps her offer rides to fellow students which makes their commutes easier, but it also helps offset some of her expenses with contributions from her passengers. Ashley is aware of the challenges new students face, especially those who are new to the city, so each semester, Ashley updates her pool schedule on the app, sharing pool routes with start and end locations she frequently takes, marking which pools are recurring, and indicating any available seats in her car. One evening, a notification from Pool pops up, a new passenger by the name of Jeff, is requesting a ride that aligns perfectly with her pool. Noting that he's a transfer student living in the Outer Sunset, Ashley realizes that his location is right along her daily pool route. Recalling how confused she was when she first came to the city, Ashley empathizes with Jeff, from one student to another, so she accepts the ride request for the next morning. Ashley pulls up outside Jeff's apartment the following day since he requested a pickup location different than Ashley's pool start location, they exchange greetings, and their pool begins. Ashley shares her experiences and gives Jeff tips about university life in San Francisco State and how to get around San Francisco, although the pool was short, the engagement made the once mundane drive to school an opportunity for connection and friendship. As the weeks went by, Ashley and Jeff's rides became more lively, with discussions, shared playlists and mutual appreciation of the city landmarks. Pool has turned Ashley's daily drive from a solitary task into a shared experience that fosters friendship and a sense of community, leading to the creation of this two person crew.

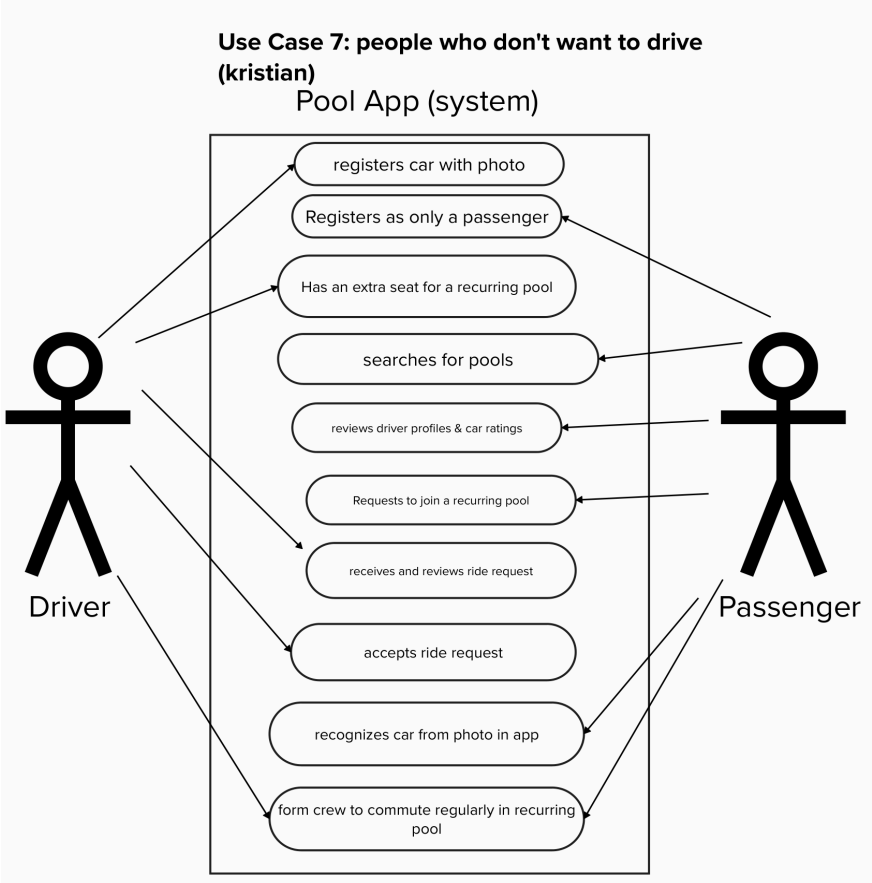

Use Case 6 Building connections
while lending a helping hand (Phillip)
Pool App (system)
Download and register on Pool
Input personal details
Updates pool schedule
send ride request
Receive ride request notification
accept or decline ride
Start the journey and share tips
Provide feedback and rating
Driver
Passenger

Use Cases: Passengers

Use Case 7: People who do not want to drive

Actors: Tom (carpool passenger), Driver

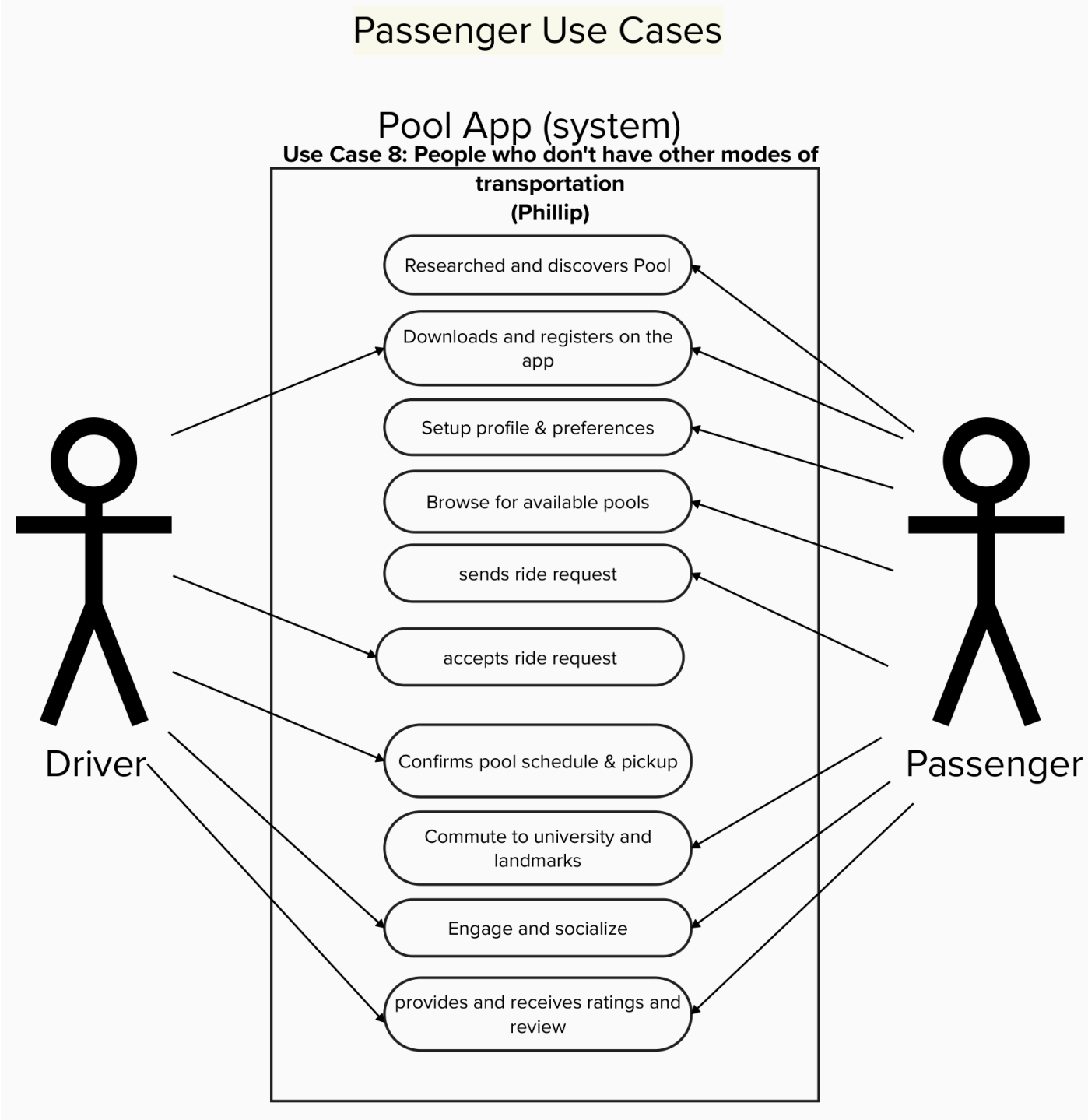
Tom is a business professional who works in the San Francisco financial district. Tom and his wife recently moved away from their downtown apartment to a home in San Mateo, California. When Tom Lived downtown he was conveniently located next to an underground Muni train station and only needed to go a few stops to get to work. After moving to San Mateo he tried driving, but quickly realized he did not like driving during peak traffic hours, it caused him a great deal of distress. Because Tom does not like driving during peak hours, he creates an account on the Pool carpooling website, he is able to register as only a passenger without the need to be a driver.. Then Tom searches the Pool website for recurring Pools he can regularly join to share a car to work. Tom finds an available pool that has space in the car; both the driver and car have great ratings, so he sends a ride request to the driver. Plus, the driver's profile shows that he accepts Venmo, making it easy for Tom to pay the very reasonable toll contribution of $1 using his posted account details. The driver reviews Tom's profile and approves his ride request. Tom easily recognizes the car at the pickup location of his first pool based on the make, model, year and color, which matches its picture in the app. Tom now commutes to work regularly with a crew and he no longer dreads his daily commute to work – and because the car has air conditioning and wifi, he can even get a head start on work during his commute, in comfort.



Use Case 8: People who don't have other modes of transportation

Actors: Jeff

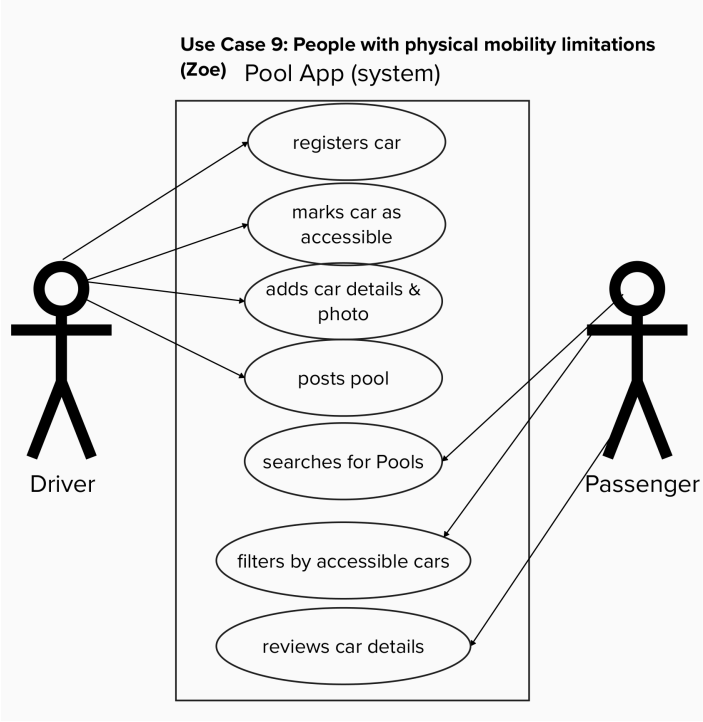
Jeff is a university student who recently transferred to San Francisco from his community college in a relatively smaller town where he only had to walk to where he needed to be, so a car was unnecessary. Excited to start attending San Francisco State and a new life in a different city, Jeff forgot how much bigger San Francisco was compared to where he was from. Initially intimidated by the city's overwhelming and bustling transportation system, Jeff began to stress out, although there were public transport options available, things such as unfamiliar routes and packed schedules made his commute to the university stressful. While attending a university orientation event, Jeff hears about Pool, an app specifically designed for students and residents of San Francisco looking to share rides, intrigued by the idea, he downloads the app right away. After getting the app, Jeff quickly set up his profile, detailing his commute from his apartment in the Outer Sunset to SF State, he inputs his class times to allow the app to suggest optimal pick-up times and check preferences such as fellow student passengers or quiet rides to study during the journey. The app identifies several drivers with overlapping routes, but a particular driver was also a senior student from his university and has been carpooling for the past year, they have been driving by his apartment every morning and have space in their car. The next morning, Jeff waits outside his apartment and, right on time, the senior student from his university arrives, the ride to campus is smooth and quick, with the driver sharing tips about the city and university life, Jeff feels more connected, having made a new friend along with an efficient way to commute to school. As days turn into weeks, Jeff regularly uses Pool as a mode of transportation, not just for school but also for exploring the city and visiting iconic places such as the Golden Gate Bridge and Fisherman's Wharf. The app allows him to travel with locals, making his adaptation to San Francisco life cost-effective and socially enriching. Jeff provides positive feedback for his driver and gives her a great rating.


Passenger Use Cases
Pool App (system)
Use Case 8: People who don't have other modes of transportation
(Phillip)
Researched and discovers Pool
Downloads and registers on the app
Setup profile & preferences
Browse for available pools
sends ride request
accepts ride request
Confirms pool schedule & pickup
Commute to university and landmarks
Engage and socialize
provides and receives ratings and review
Driver
Passenger

Use Case 9: People with physical mobility limitations

Actors: Ty

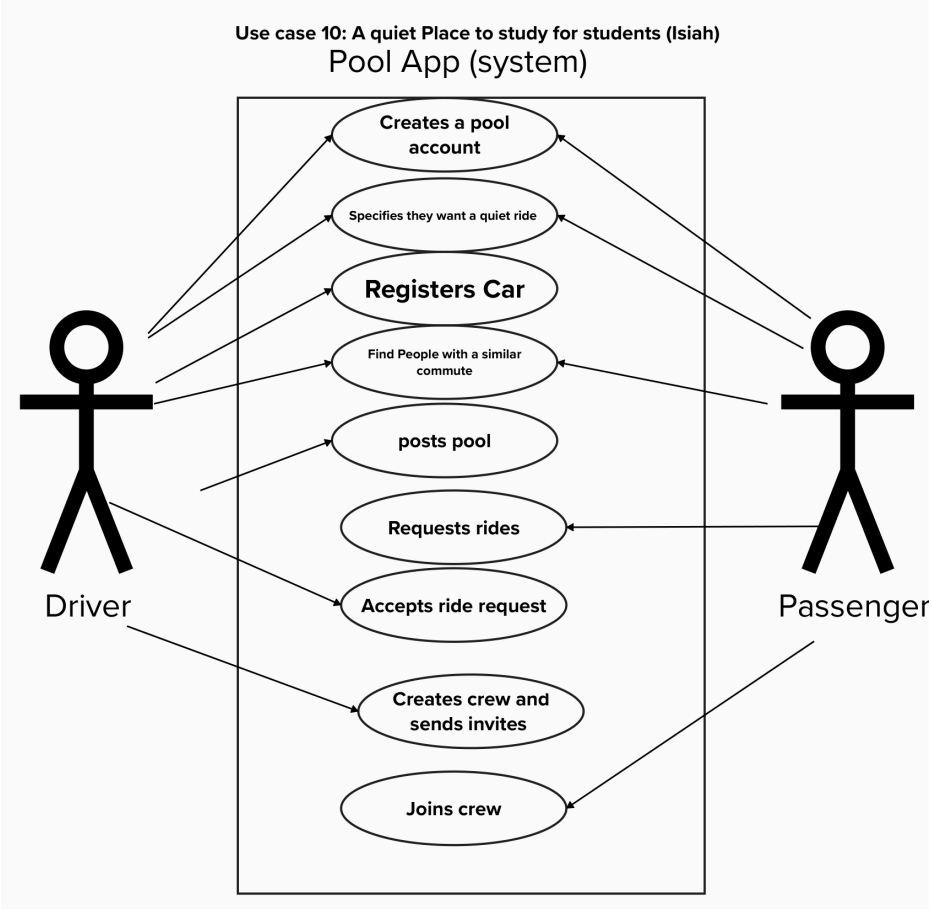
Ty has navigated their life in a wheelchair since they were a toddler, and has always been incredibly independent. This has left Ty with remarkable upper body strength and the ability to self transport around home as well as within their community. But even as an advanced wheelchair user, Ty struggles with public transit – at no fault of their own; many times, elevators are out of service at Bart stations, and Ty has no way of finding this out prior to their arrival, at which point they must completely renavigate and often end up arriving late for work. A friend recently told Ty about Pool, a community carpooling app. But Ty is skeptical of Pool, because their experiences with Lyft and Uber haven't been consistent; sometimes drivers who say their vehicle can accommodate wheelchairs actually can't, and the process of even communicating about this with drivers has been arduous. With no better options, Ty decides to try Pool since it allows you to filter by pools with accessible vehicles, but plans to set aside a whole extra hour for the experience in case it doesn't work out and they need to make alternative plans to get to work. Before creating an account, Ty simply browses to see if any pools in their area actually have wheelchair accommodations.



Use Case 10: A quiet place to study for students

Actors: Emily

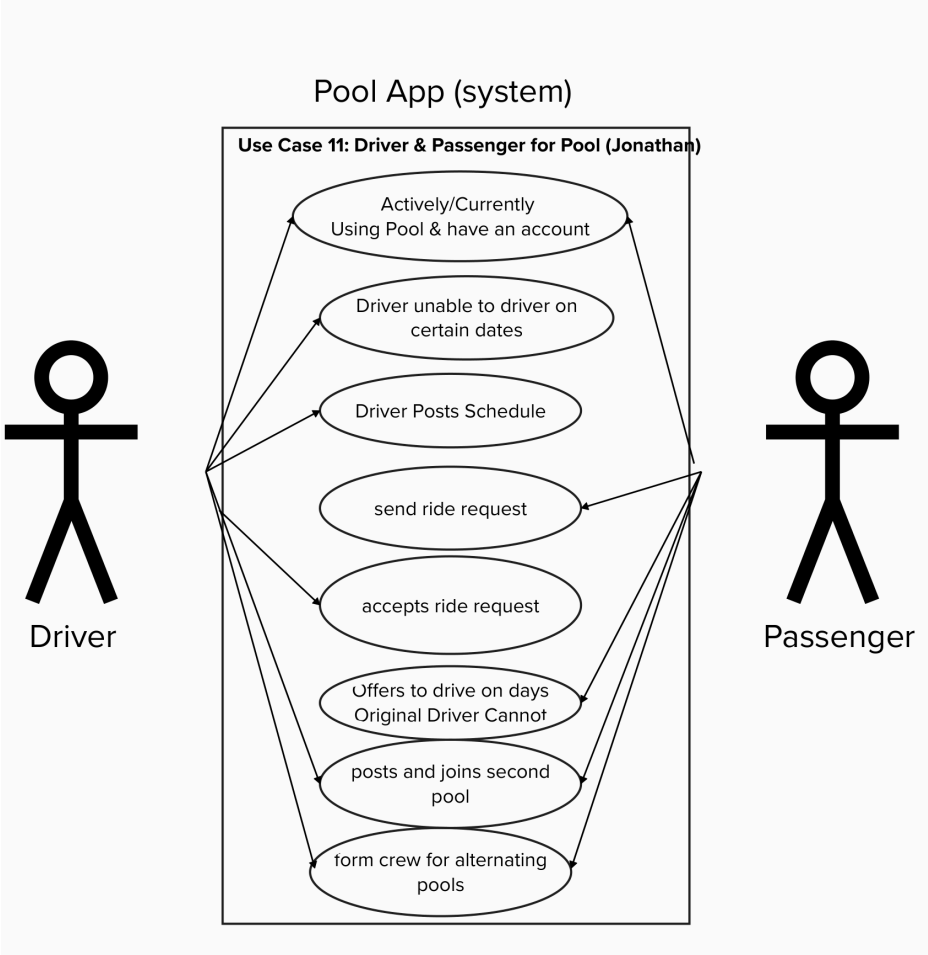
Emily is a University Student attending San Francisco State University but lives in Milpitas. She has a long commute to campus, and she's determined to make the most of her time. Emily uses the Pool app to find pools to school with wifi where she can study during the journey, so she searches for pool names and descriptions containing keywords like "study" and "students". For Emily, the pool isn't just a mode of transportation; it's her mobile study space. She is actively looking for a crew of fellow passengers who also value a quiet and focused environment during their pool. By carpooling with others who share her goal, Emily can make the most of her daily commute, optimize her study time, and arrive at school prepared for her classes. After her fourth pool with a consistent group of passengers who also see their commute as an opportunity to enhance their productivity and learning experience, Emily invites the members of her pool to create a crew. The members all accept and they proceed to regularly pool together.



Use Case 11: Driver & Passenger for Pool

Actors: Jason, Marissa

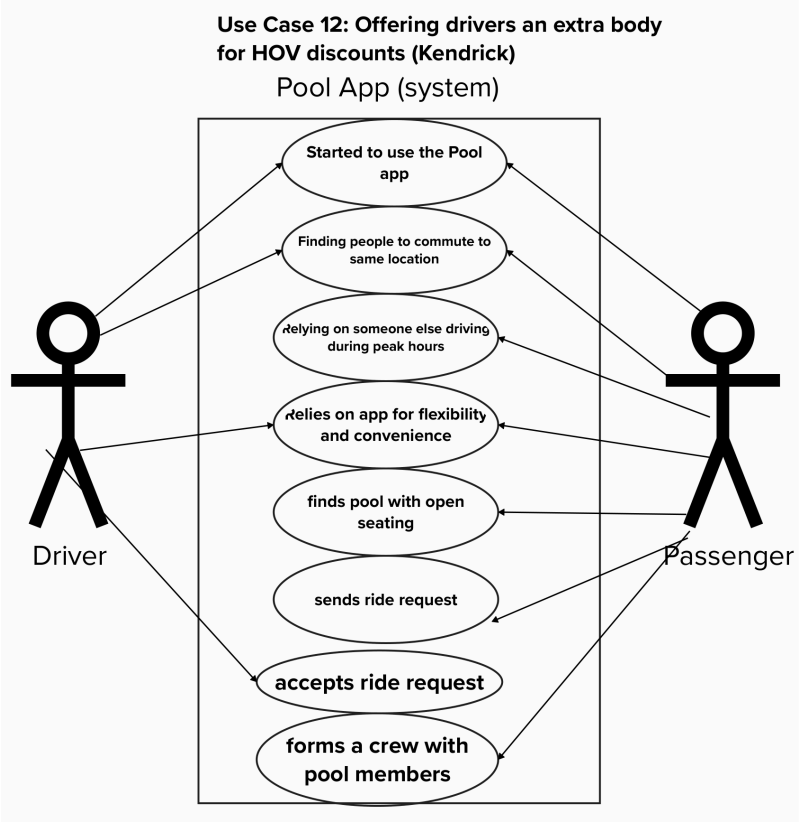
Jason recently moved to the East Bay for work and has been living there for the past 3 years. During his time there, he has met coworkers that also live in East Bay and have agreed to form a crew and commute together regularly through the Pool app. Jason doesn't mind driving, but he sleeps later than usual every other Tuesday night to help his kids study for their school's biweekly math competitions. Because of the lack of sleep Jason gets, Jason doesn't feel comfortable driving on less sleep than usual – especially when transporting his youngest to daycare en route. Jason has informed his coworkers of his situation beforehand so they are aware that he can't drive their crew on this day of the week. One of his coworkers, Marissa, agrees to sign up as a driver and register her vehicle – which has child seats – and create a recurring pool to drive their crew to work every other Wednesday. When Jason's schedule later changes, it's easy for him to leave his Wednesday crew and add this day back to his own recurring pool's schedule.



Use Case 12: Offering drivers an extra body for HOV discounts

Actors: Tom

Tom, a business professional working in San Francisco SoMa, recently relocated to the East Bay and faced the challenge of commuting during peak traffic hours, which caused him distress. To alleviate this, Tom turned to the Pool app. He wanted to use the app to start commuting with a regular pool who he could form a crew with to regularly share a car with to work, so he started by seeking out pools. After Tom sent his first crew invites to members of a pool he'd ridden with twice, one member declined, but the group was still big enough to form a new crew without him. Tom preferred being a passenger rather than a driver during peak hours, and the app made it easy to find free pools posted by drivers seeking HOV discounts. Now, Tom no longer dreads his daily commute, where he had to find his luck for parking, as he enjoys a reliable and stress-free mode of transportation, thanks to his Pool crew. As a passenger, Tom benefits from the app's flexibility and convenience, transforming his daily commute into a more pleasant experience.



Main Data Items and Entities

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

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

Password: tied to one user, stored separately from other user details

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

Notifications: how users are informed that an action has been taken in regards to a ride request they are associated with

Functional Requirements

1. User
    - 1.1. A user shall search for pools.
    - 1.2. A user shall create an account
    - 1.3. A user shall register as a driver
    - 1.4. A user shall register as a passenger
    - 1.5. A user shall be able to view their passenger profile
    - 1.6. A user shall be able to view their driver profile
    - 1.7. A user shall be able to view their passenger ratings received
    - 1.8. A user shall be able to view their driver ratings received
    - 1.9. A user shall be able to view their passenger feedbacks received
    - 1.10. A user shall be able to view their driver feedbacks received
    - 1.11. A user shall be able to report any feedbacks
    - 1.12. A user shall have an emergency contact
    - 1.13. A user shall be able to delete their account
    - 1.14. A user shall be able to file a report ticket
    - 1.15. A user shall be able to chat with support

2. Profile
    - 2.1. A profile shall belong to a passenger
    - 2.2. A profile shall belong to a driver
    - 2.3. A profile shall contain user details
    - 2.4. A profile shall display ride history
    - 2.5. A profile shall display the average rating
    - 2.6. A profile shall contain payment details
    - 2.7. A profile shall display total distance traveled
    - 2.8. A profile shall display total environmental impact
    - 2.9. A profile shall display total number of crews with membership

3. Driver
    - 3.1. A driver shall be FasTrak verified

3.2. A driver shall create pools
3.3. A driver shall have ratings
3.4. A driver shall have cars
3.5. A driver shall have crews
3.6. A driver shall be associated with an account
3.7. A driver shall send ride requests for their pool(s)

4. Passenger
   4.1. A passenger shall join pools
   4.2. A passenger shall join crews.
   4.3. A passenger shall send invites for crews.
   4.4. A passenger shall be associated with one and only one profile
   4.5. A passenger shall be associated with one and only one account
   4.6. A passenger shall have ratings

5. Crew
   5.1. A crew shall be created by a user
   5.2. A crew shall have a name
   5.3. A crew have a description
   5.4. A crew member shall be able to invite users to their crew
   5.5. A crew invite shall be accepted
   5.6. A crew invite shall be declined
   5.7. A crew creator shall be able to remove users from their crew
   5.8. A user shall be able to leave a crew

6. Car
   6.1. A car shall be associated with a driver
   6.2. A car shall be registered
   6.3. A car shall have ratings

7. Pool:
   7.1. A pool shall have a description
   7.2. A pool shall have a start time.

7.3. A pool shall have an end time.
7.4. A pool shall have an occurrence rate.
7.5. A pool shall have passengers.
7.6. A pool shall have ride requests.
7.7. A pool shall have a car
7.8. A pool shall have an start location
7.9. A pool shall have an end location
7.10. A pool shall be viewable by its passengers or driver
7.11. A pool shall have a total distance in miles

8. Ratings
   8.1. A rating shall have a numerical value of 1-5
   8.2. A rating for a driver shall be provided by a passenger belonging to a common pool
   8.3. A rating for a passenger shall be provided by a driver belonging to a common pool
   8.4. A rating for a car shall be provided by a passenger belonging to a common pool

9. Ride request
   9.1. A ride request shall be made by or for passengers
   9.2. A ride request shall be made for a Crew
   9.3. A ride request shall be made for a Pool
   9.4. A ride request shall have seats
   9.5. A ride request shall contain the Profile associated with the requesting Passenger(s)
   9.6. A ride request shall contain the name associated with the requesting Crew
   9.7. A ride request shall contain the description associated with the requesting Crew
   9.8. A ride request shall have zero or one pickup location(s)
   9.9. A ride request shall have zero or one dropoff location(s)
   9.10. A ride request shall be accepted

9.11. A ride request shall be declined

10. Notifications

10.1. Notifications may be triggered by many actions.
10.2. Notifications may be toggled on or off per account.

Non Functional Requirements

1. Scalability
   - 1.1. Pool shall use scalable storage in Google Cloud Platform.
   - 1.2. Pool shall use load balancing on GCP as demand increases.
   - 1.3. Pool team shall monitor CPU usage and traffic on GCP and allocate resources as necessary.

2. Reliability
   - 2.1. Pool shall be available to users 24/7 with minimal downtime.
   - 2.2. Pool shall use user input validation to ensure the integrity of data stored and retrieved.
   - 2.3. Pool shall perform regular data backups in GCP to ensure no data loss.

3. Regulatory
   - 3.1. Pool shall be in compliance with cookie and tracking regulations.
   - 3.2. Pool shall comply with Web Content Accessibility Guidelines (WCAG) to ensure accessibility to individuals with disabilities.

4. Maintainability
   - 4.1. Pool shall implement version control using GitHub to track changes to the codebase.
   - 4.2. Pool shall use a GitHub flow branching strategy.
   - 4.3. Pool shall use Apache Maven version 3.9.4 for backend dependency management.
   - 4.4. Pool shall use the latest npm version 7.0 (Node package Manager) for the react.js front end.

5. Serviceability
   - 5.1. Application backend is capable of switching between different data sources when it is required to call a backup data source
   - 5.2. Application has an intentional splash page for unexpected downtime.
   - 5.3. Code deployments and maintenance will not require downtime.

6. Utility
   - 6.1. Web Application will have search functionality to assist users.
   - 6.2. The system will have a user-friendly interface for new and existing users.

7. Manageability
   - 7.1. Alerting mechanisms should allow administrators to easily & quickly understand problems during critical events
   - 7.2. Administrators will have control of Pool's system through dev portals of Google Cloud and MySQL Workbench v 8.0.34

8. Data Integrity
   - 8.1. Passwords shall be encrypted.
   - 8.2. Encryption and decryption shall be tested for accuracy.

9. Capacity
   - 9.1. Pool shall specify amount of data the application can handle
   - 9.2. Pool shall specify maximum number of concurrent user sessions to support seamless user experience when in high traffic periods.

10. Availability
    - 10.1. Application shall have a high level of system uptime
    - 10.2. Application shall be available through various platforms and devices

11. Usability
    - 11.1. Users shall be able to easily navigate through Pool
    - 11.2. Pool shall give error messages that are clear and helpful when wanting to resolve issue

12. Interoperability
    - 12.1. Pool shall be able to integrate with map services to provide directions
    - 12.2. Pool shall be compatible for different range of vehicles

13. Privacy
    - 13.1. Passwords shall be stored as hashes, ie. SHA256 hashing function.
    - 13.2. Pool shall have a response plan in the event of a data breach.
    - 13.3. User accounts shall require authentication for access.
    - 13.4. PII shall be handled with an added layer of care and protection.
    - 13.5. Requesting systems must have the required credentials in order to access application data.

14. Coding Standards
    14.1. Testing coverage tooling such as SonarQube will be implemented to expose areas of the codebase requiring test coverage.
    14.2. Code shall be peer reviewed prior to being merged into the master branch.

15. Networks
    15.1. Pool shall be performant on low bandwidth networks.
    15.2. Pool shall leverage caching to increase performance during periods of reduced network availability.

16. Databases
    16.1. Pool shall use database optimization techniques to increase performance.
    16.2. User actions and associated data shall be preserved at the database level.

17. Performance
    17.1. Application will have an average response time of 2 seconds or less.
    17.2. Application utilizes efficient data storage techniques to reduce load time.
    17.3. Pool shall minimize device CPU usage.
    17.4. Pool shall minimize device memory usage.

18. Navigation & Wayfinding
    18.1. All open fields shall have sufficient validations.
    18.2. All open validations shall have helpful error handling that help the user course correct.
    18.3. Open fields shall auto suggest and when possible, pre-fill data to reduce the cognitive burden required to complete an action.

19. Security
    19.1. Data shall be backed up regularly.
    19.2. Data shall be sent over https protocol.

20. Portability
    20.1. Application shall have cross platform compatibility across all actively supported iOS, MacOS, Windows OS, Linux OS, and Android OS devices.
    20.2. Application shall function on the latest version of most common web browsers, i.e. Safari, Chrome, Firefox, DuckDuckGo

21. Cost & Budgeting

21.1. Application operating entity shall maintain sufficient financial resources capable of maintaining and scaling core services including the database, servers, third party applications and software, etc.
21.2. Google Cloud should not have to incur any charges based on memory usage

22. Storage

22.1. Tables should not exceed 400 bytes
22.2. Values in tables should not exceed 50 bytes

23. Accessibility

23.1. Application frontend can accommodate users who may require a larger font size.
23.2. Application frontend can be easily translated to any language.
23.3. Application frontend can be easily navigated by users without sight.
23.4. Application frontend is optimized for readability.
23.5. Application signup should take no more than 10 clicks

24. Environmental Impact

24.1. The application shall incentivize making full use of car's capacity to optimize environmental impacts
24.2. The application shall have a feature that will remind drivers to turn off their engines if they're waiting for long periods of time, like waiting for carpool participants
24.3. The application shall give priority or recognition to drivers with hybrid or electric vehicles to promote environmental sustainability
24.4. The application shall provide periodic emission saving reports to users to keep them motivated and informed

Competitive Analysis

Table 1

Features	Uber https://www.uber.com/	Lyft https://www.lyft.com/	Merge https://merge.511.org/#/	SF Casual Carpool https://sfcasualcarpool.com/	Public Transit https://www.bart.gov/	Taxis/Flywheel https://www.flywheel.com/
UI/UX	Intuitive, each category has concisely what one needs.	User friendly , intuitive UI/Ux	Simple, Compact, not overly flashy, easy to navigate	UI design is simplistic and easy to navigate	Varies by service and transit agency, not consistent	User friendly, easy to navigate and accessible to many tabs
Accessibility	Covers all of the major cities spanning multiple countries as well areas with smaller populations.	Robust coverage in metropolitan areas. Sparser coverage farther from large cities. Mobile version.	Nationwide, dependent on user population for rider options	There are no accessibility features, things like dark mode is absent on the site, and there are no instructions for non abled people, and the drop off point(s) are very limited.	Accessibility varies for web experiences. In person accessibility varies greatly by vehicle in use (light rail vs bus when it comes to Muni), and whether or not elevators are in service for BART.	Covers four different cities
Community	Based on one's location, uber drivers and riders are paired	Riders are paired with drivers based on their	Small community, but enough carpooling options. Communit	The stops across the bay are condensed , so drivers and passenger	BART has had some recent efforts to build community on	Not many carpooling options and the rider asks for a pickup. If one needs to learn more

	together.	vicinity to one another.	y based on how the demand for carpoolin g in a given area	s are most likely going to be going the same way, so familiar drivers/pas sengers is a possibility. However, ride etiquette is random and based on the people you carpool with.	board, none for other public transit agencies .	about the company they have to reach out.
Privacy	An in depth policy that labels what the drivers can and cannot see from the rider and vice versa, including but not limited to ratings, phone numbers, profile picture, last name etc.	Robust data privacy. Issues with drivers posting dash cams without rider permissi on have occurred .	User profile shows only what user wants to publicly show, no profile picture, cookies used to share with third parties, location data tracking the moment app is download ed	SF Casual Carpool doesn't really ask for your phone number or details, so what you decide to share is up to you when meeting drivers or other passenger s.	Standar d cookie policies on most sites, none in IRL experien ces.	Do not gather info of visitors under 13 of age. Collects personal that is listed and it is all accessed once the user gives consent. Only name, phone # and email can be inputted. No profile pic or pronouns.
Pricing	Pricing is mainly based on	Pricing varies because of the	Pricing varies depende	Pricing is dependent on the driver, the	Due to a lack of connectiv ity	Pricing is based on the base fare,

	the base rate, the operating fee, and where one is in a busy time or area. There is a pricing estimate on their website. For major cities and long commutes, expect higher pricings.	surge pricing model, but is generally expensive for daily commuting.	nt on location A to B and driver. From first glance, no money required except for tip at the end	driver is responsible for communicating with the passengers if they would like a small contribution to cover expenses (usually $1).	between services, expensive.	distance, time, additional charges, tips. No pricing estimate is located in their data
Sustainability	In 2020, Uber had announced a global commitment to becoming a zero-emission mobility platform. Their goal by 2025 is that a large percentage of drivers transition to EVs. (Electric Vehicles.)	Lyft pool service has been suspended. Uses a one driver one request ride request model contributing to added traffic congestion and greater environmental impact.	As of 2022, Merge has been announced as an official carpool collaborator for the Bay Area, extremely community driven as passengers are only expected to tip at the end	The casual carpool system comprises and relies on common sense, as passengers and drivers are expected to act responsibly. And the entire system is community driven, so it's cost effective and adaptive, but a challenge would definitely	BART is electric, so low environmental impact. BART also utilizes sheep and goat herds for natural land maintenance. Muni is a mixed gas/electric fleet (50% electric).	Rideshare hasn't announced much for the sustainability. Although, in 2022 Flywheel had partnered up with Uber.

				be monetization and funding.		
Personalization	Very Barebones personalization, one can add their name, pronouns, profile picture, and their home and work addresses.	Account links to social media and remembers most frequently traveled destinations.	Only can post short description and commute preferences	No personalization since there's no need to nor ability to put any personal information on the website.	None	Can favor their favorite location, set preferred vehicle type, only schedule rides to airport
Trust & Safety	Uber has highlighted a lot of safety measures. They have a multi-step background process for uber drivers. As well as policies such as no front seat use for riders, 911 assistance built into the app,auto insurance, and ways to report any issues	Have only an initial background check. Many incidents have made the news questioning safety standards.	Trust solely based on driver and passenger relationship. No authorization required. Driver may ask for authorization after messaging about commute	Random carpool forming, no system to really allow riders or drivers to know who they're meeting.	BART launched a "citizen oversight board" to increase account ability of BART police	Taxi drivers require a higher level of licensing and permits than rideshare drivers.

	on their app.					

Table 2

Features	Uber	Lyft	Merge	SF Casual Carpool	Public Transit	Taxis	Pool
Finding a ride	++	++	++	+	+	-	+
Ride w/your "Crew"	-	-	-	-	-	-	++
Ride ratings	++	++	+	-	-	-	++
profiles	+	-	+	-	-	-	++
carpooling	-	-	+	++	+	+	++
Reducing carbon emissions	+	-	+	++	++	+	++

Legend: **+** = has this feature, **++** = does this really well, **–** = doesn't have this feature

Pool's laser focus on carpooling, with an emphasis on people and planet, sets it apart from its competition in the ridesharing market. Competitors focus on putting profit over people, and this shows in the general lack of customization in apps like Uber and Lyft, a general disregard for the environment (demonstrated by the discontinuation of both apps' pooling feature), and lack of regard for passengers who may get stuck in traffic due to not all drivers having Fastrak. Pool uniquely allows passengers to ride with their own "Crew", a feature that enhances community and safety by allowing passengers and drivers to ride with familiar faces. Crews offer a safer, consistent, and predictable riding experience, which is often absent from shared transit experiences provided by SF Casual Carpool, public transit, and taxis. Pool makes rides cost-effective and environmentally friendly by requiring drivers to provide an active Fastrak ID so that all pools are HOV and fast lane eligible by default. Pool takes into account user preferences, community-building, and environmental considerations and puts them all together to offer a diverse and user-driven ride sharing platform. This is all made possible through partnerships with city and county governments, as well as other public and private entities interested in offering ride sharing services to its constituencies, residents, and employees. Funded through these partnerships as well as governmental

grants, Pool makes ride sharing cost effective for everyone involved. This model allows Pool to put people over profit and focus on valuable user improvements without needing to focus on expanding its bottom line.

High-level System Architecture & Technologies Used

Server Host: Google Cloud
Operating System: Windows and Mac
Database: MySQL v 8.0.34
Web Server: Tomcat v 8.5 (via Spring Boot)
Server-Side Language: Java
Additional Technologies:
- Web Framework: Spring Boot v 3.1.3 and React.js v 18.2
- IDE: Visual Studio Code 1.81 and IntelliJ v 2023.2.1
- SSL Cert: Google-managed SSL

Checklist

1. **Done** ✅Team found a time slot to meet outside of the class.
2. **Done** ✅Github Master chosen.
3. **Done** ✅Team decided and agreed together on using the listed SW tools and deployment server.
4. **Done** ✅Team ready and able to use the chosen back and front end frameworks and those who need to learn are working on learning and practicing.
5. **Done** ✅Team lead ensured that all team members read the final M1 and agree/understand it before submission.
6. **Done** ✅GitHub is organized as discussed in class (e.g. master branch, development branch, folder for milestone documents etc.)

List of Team Contributions

Contribution Scores

- Isiah: 10
- Jonathan: 10
- Kendrick: 8
- Kristian: 10
- Phillip: 10

Team Contributions: Checkpoint 1

Executive Summary

Required roles: full team, collaborative work on a Zoom

- ✅Isiah: (V1) verbally contributed
- ✅Jonathan: (V1) verbally contributed, (V2) collaboratively worked through revisions (adding team motivation + info about team) to executive summary
- ✅Kendrick: (V1) verbally contributed, (V2) collaboratively worked through revisions (adding team motivation + info about team) to executive summary
- ✅Kristian: (V1) verbally contributed, (V2) collaboratively worked through revisions (adding team motivation + info about team) to executive summary
- ✅Phillip: (V1) verbally contributed
- ✅Zoe: (V1) typed and screenshared while team verbally contributed, (V2) facilitated collaborative revision session

Main Use Cases

Required roles: full team, each member writes one passenger use case and one driver use case and creates corresponding visual diagrams (asynchronously)

- ✅Isiah: (V1) Use cases 4 & 10
- ✅Jonathan: (V1) Use cases 1 & 11, (V2) edited Use case 1
- ✅Kendrick: (V1) Use cases 5 & 12, (V2) edited Use cases 2 & 5
- ✅Kristian: (V1) Use cases 3 & 7, (V2) edited Use cases 3 & 7
- ✅Phillip: (V1) Use cases 6 & 8
- ✅Zoe: (V1) Use cases 2 & 9

Main Data Items & Entities

Required roles: full team, collaborative work on two Zooms (two sessions split between team members)

- ✅Isiah: (V1) wrote two data items and their entities
- ✅Jonathan: (V1) wrote two data items and their entities
- ✅Kendrick: (V1) wrote two data items and their entities
- ✅Kristian: (V1) wrote two data items and their entities
- ✅Phillip: (V1) wrote two data items and their entities
- ✅Zoe: (V1) facilitated co-working session during which we all worked on this section

Functional Requirements

Required roles: full team, collaborative work on two Zooms (two sessions split between team members)

- ✅Isiah: (V1) requirements for Profile and Crew
- ✅Jonathan: (V1) requirements for Ratings
- ✅Kendrick: (V1) requirements for Ride Request
- ✅Kristian: (V1) requirements for Contact Information (now deprecated) and Passenger
- ✅Phillip: (V1) requirements for User and Car
- ✅Zoe: (V1) requirements for Driver and Pool

Nonfunctional Requirements

Required roles: full team, collaborative work on Zoom plus asynchronous contributions (five additional nonfunctional requirements each)

- ✅Isiah: (V1) contributed collaboratively on Zoom, plus added five nonfunctional requirements across several categories asynchronously
- ✅Jonathan: (V1) contributed collaboratively on Zoom, plus added five nonfunctional requirements across several categories asynchronously, (V2) wrote new functional requirements for serviceability, utility, manageability, and data integrity categories
- ✅Kendrick: (V1) contributed collaboratively on Zoom, plus added five nonfunctional requirements across several categories asynchronously, (V2) wrote

new nonfunctional requirements for capacity, availability, usability and interoperability categories

- ✅Kristian: (V1) contributed collaboratively on Zoom, plus added five nonfunctional requirements across several categories asynchronously, (V2) wrote new functional requirements for scalability, reliability, regulatory, and maintainability categories
- ✅Phillip: : (V1) contributed collaboratively on Zoom, plus added five nonfunctional requirements across several categories asynchronously
- ✅Zoe: (V1) facilitated collaborative work session on Zoom, assigned our asynchronous work, reviewed work, wrote new nonfunctional requirements for privacy, coding standards, networks, and database categories

Competitive Analysis

Required roles: full team, collaborative work on Zoom plus asynchronous contributions (team worked together on column 1 in both tables and ranked our product together, each team member was also assigned a competitor to rank against column 1 in each table asynchronously)

- ✅Isiah: (V1) “Uber” row in Table 1 and Table 2
- ✅Jonathan: (V1) “Merge” row in Table 1 and Table 2, (V2) added URLs to Table headers (Merge and SF Casual Carpool) and collaborative work on summary
- ✅Kendrick: (V1) “Taxis” row in Table 1 and Table 2, (V2) added URLs to Table headers (public transit and taxis) and collaborative work on summary
- ✅Kristian: (V1) “Lyft” row in Table 1 and Table 2, (V2) added URLs to Table headers (Uber and Lyft) and collaborative work on summary
- ✅Phillip: (V1) “SF Casual Carpool” row in Table 1 and Table 2
- ✅Zoe: (V1) “Public Transit” row in Table 1 and Table 2, (V2) facilitated collaborative revision session

High-level System Architecture & Tech Used

Required roles: full team, collaborative work on Zoom

- ✅Isiah: participated in full-team discussion and working session where decisions were made for software stack, architecture, etc
- ✅Jonathan: participated in full-team discussion and working session where decisions were made for software stack, architecture, etc

- ✅Kendrick: participated in full-team discussion and working session where decisions were made for software stack, architecture, etc
- ✅Kristian: participated in full-team discussion and working session where decisions were made for software stack, architecture, etc
- ✅Phillip: participated in full-team discussion and working session where decisions were made for software stack, architecture, etc
- ✅Zoe: facilitated in full-team discussion and working session where decisions were made for software stack, architecture, etc

Team Contributions: Checkpoint 2

Creating the application

Required roles: GitHub Guru, Backend Baron, Team Lead (collaborative work on two Zoom sessions with Backend Baron driving)

- ✅Isiah: played an “active passenger” role while pairing with Phillip
- ✅Phillip: played the “driver” role while pairing with Isiah, created the folder and file structure for the app using Spring initializer with all required dependencies (web, MySQL, etc), committed changes
- ✅Zoe: facilitated coworking session, provided support as needed

Configuring Cloud Instance and Setting Up VM & CloudSQL DB

Required roles: Release Manager and Team Lead (collaborative work on three Zoom sessions with Release Manager driving, and later, Team Lead driving due to device incompatibility issue (ARM Mac).

- ✅Kristian: set up our instance in Google Cloud Console: created a Cloud SQL db, created a virtual machine instance, configured settings to ensure credits won’t run out. Tested the deploy process using a Docker image, paired with Zoe while she completed the deploy process.
- ✅Zoe: supported Kristian in navigating Google Cloud Console, created a docker image, added to Artifact Registry, added image to vm, test ran live site to ensure it was successful.

Create About Page

Required roles: Frontend Lead (driving), DB Lead, Release Manager, Team Lead

- ✅Jonathan: created main About Page and created a team member page template for everyone to use for their own pages.

Create Team Member Pages

Required roles: Full team, everyone creates and contributes to their own page, Frontend Lead troubleshoots as needed and ensures consistency between pages.

- ✅Isiah: created his team member page using the template Jonathan provided
- ✅Jonathan: created his team member page using the template Jonathan provided
- ✅Kendrick: created his team member page using the template Jonathan provided
- ✅Kristian: created his team member page using the template Jonathan provided
- ✅Phillip: created his team member page using the template Jonathan provided
- ✅Zoe: created her team member page using the template Jonathan provided

Gathering and Documenting Credentials

- ✅Zoe: created credentials .pdf provided by Milestone 1 guiding doc, added a copy of this table plus access instructions to our credentials folder in GitHub.

Pool - carpooling app

SW Engineering CSC648-848-05 Fall 2023

Milestone 2 V2 | 11/02/2023

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
M2V2	**11/02/23**
M2V1	**10/12/23**
M1V2	**10/8/23**
M1V1	**9/21/23**

Table of Contents

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

- 1.1 A user shall create an account
- 1.2 A user shall register as a driver
- 1.3 A user shall register as a passenger
- 1.4 A user shall sign in
- 1.5 A user shall log out

2. Profile

- 2.1 A profile shall belong to a passenger
- 2.2 A profile shall belong to a driver
- 2.3 A profile shall contain user details
- 2.4 A profile shall be viewed by the user it is associated with

3. Driver

- 3.1 A driver shall be Fastrak verified
- 3.2 A driver shall create zero or many pools
- 3.3 A driver shall have zero or many crews
- 3.4 A driver shall have a driver's license
- 3.5 A driver shall view the pools they posted
- 3.6 A driver shall view the crews they are a member of
- 3.7 A driver shall leave the crews they are a member of

4. Passenger

- 4.1 A passenger shall join many pools
- 4.2 A passenger shall have many crews
- 4.3 A passenger shall view the pools they have joined
- 4.4 A passenger shall view the crews they are a member of
- 4.5 A passenger shall leave the crews they are a member of
- 4.6 A passenger shall leave the pools they are a member of

5. Crew

5.1 A crew shall be created by one user

5.2 A crew shall have a description

5.3 A crew shall have members

6. Pool

6.1 A pool shall have a description

6.2 A pool shall have a start time.

6.3 A pool shall have passengers.

6.4 A pool shall have one driver

6.5 A pool shall have a start location

6.6 A pool shall have an end location

6.7 A pool shall be deleted by its driver

Priority 2

1. User
   - 1.1. A user shall be able to view their passenger ratings received
   - 1.2. A user shall be able to view their driver ratings received
   - 1.3. A user shall be able to view their passenger feedbacks received
   - 1.4. A user shall be able to view their driver feedbacks received

2. Profile
   - 2.1. A profile shall contain payment details
   - 2.2. A profile shall display the average rating
   - 2.3. A profile shall display ride history

3. Driver
   - 3.1. A driver shall have ratings
   - 3.2. A driver shall send ride requests for their pool(s)

4. Passenger
   - 4.1. A passenger shall have ratings
   - 4.2. A passenger shall send invites for crews.

5. Crew
    5.1. A crew creator shall be able to remove users from their crew
    5.2. A crew invite shall be accepted
    5.3. A crew invite shall be declined

6. Pool
    6.1. A pool shall have an occurrence rate.
    6.2. A pool shall have a total distance in miles
    6.3. A pool shall have an estimated trip duration
    6.4. A pool shall have ride requests.

7. Ratings
    7.1. A rating shall have a numerical value of 1-5
    7.2. A rating for a driver shall be provided by a passenger belonging to a common pool
    7.3. A rating for a passenger shall be provided by a driver belonging to a common pool
    7.4. A rating for a car shall be provided by a passenger belonging to a common pool

8. Ride Requests
    8.1. A ride request shall have seats
    8.2. A ride request shall contain the description associated with the requesting Crew
    8.3. A ride request shall contain a clickable name of the profile associated with the requesting Passenger(s) or Crew
    8.4. A ride request shall contain the description associated with the requesting Crew
    8.5. A ride request shall have zero or one pickup location(s)
    8.6. A ride request shall have zero or one dropoff location(s)
    8.7. A ride request shall be made by passengers
    8.8. A ride request shall be made for a Crew
    8.9. A ride request shall be made for a Pool

8.10. A ride request shall be accepted

8.11. A ride request shall be declined

9. Notifications

9.1. Notifications may be triggered by many actions.

Priority 3

1. User

1.1. A user shall have an emergency contact

1.2. A user shall be able to file a report ticket

1.3. A user shall be able to chat with support

1.4. A user shall be able to report any feedbacks

1.5. A user shall be able to delete their account

2. Driver

2.1. A driver shall have one or more cars

3. Car

3.1. A car shall be associated with a driver

3.2. A car shall be registered

3.3. A car shall have ratings

4. Profile

4.1. A profile shall display total distance traveled

4.2. A profile shall display total environmental impact

4.3. A profile shall display total number of crews with membership

5. Pool

5.1. A pool shall have a car

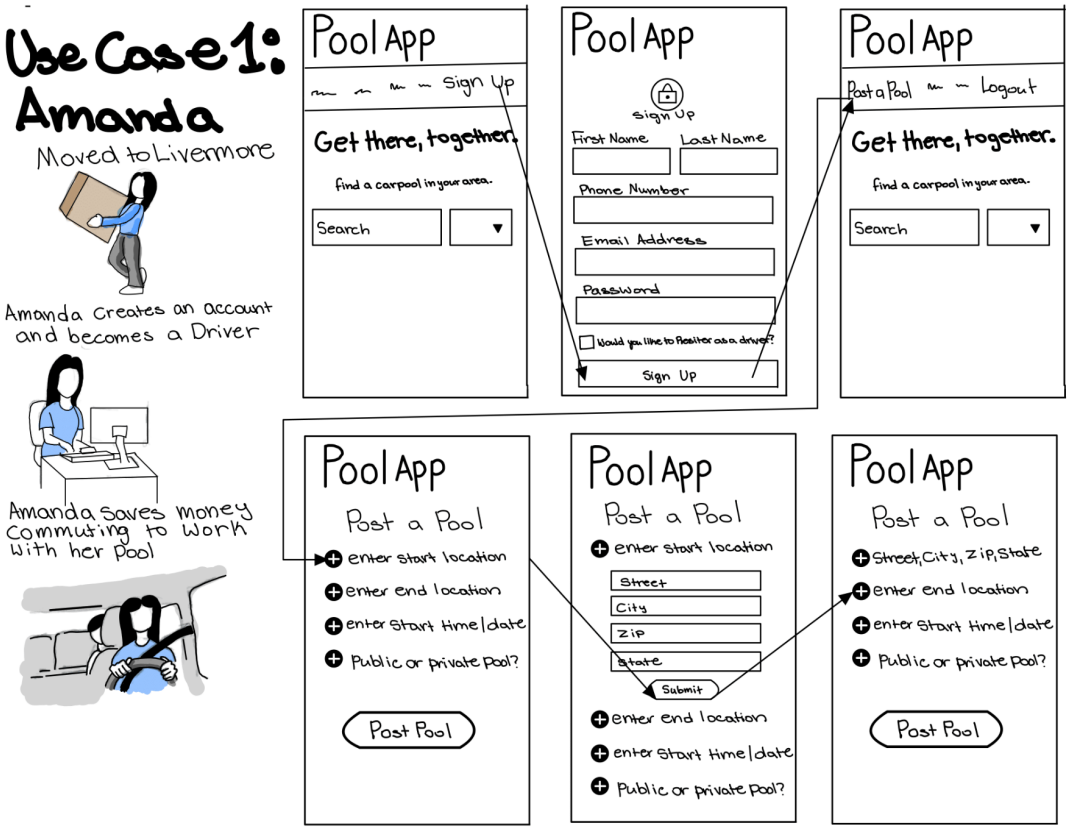
5.2. A pool shall be one-time or recurring.

6. Notifications

6.1. Notifications may be toggled on or off per account.

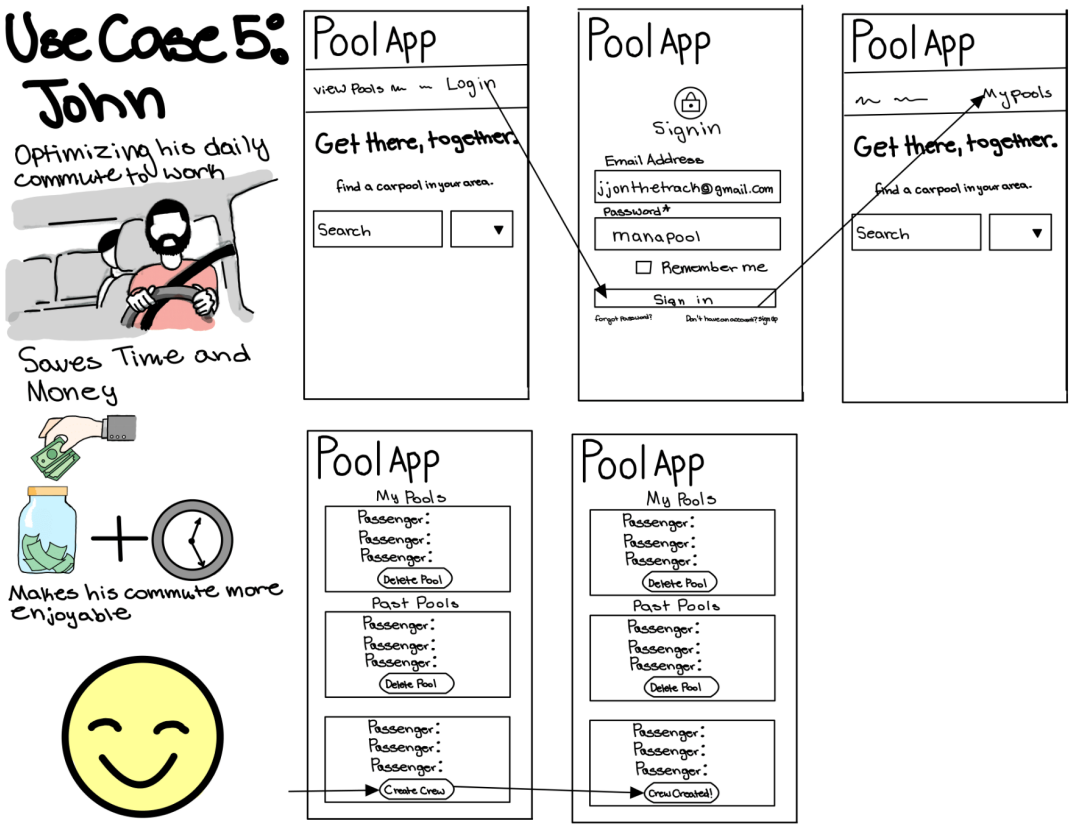
3. UI Mockups and Storyboards (high level only)

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, signed up as a driver and started offering daily rides to San Francisco. She Posts pools and lists them as public in order for others to join her, allowing her to qualify for discounts on the highway and Bay Bridge tolls due to carpooling.

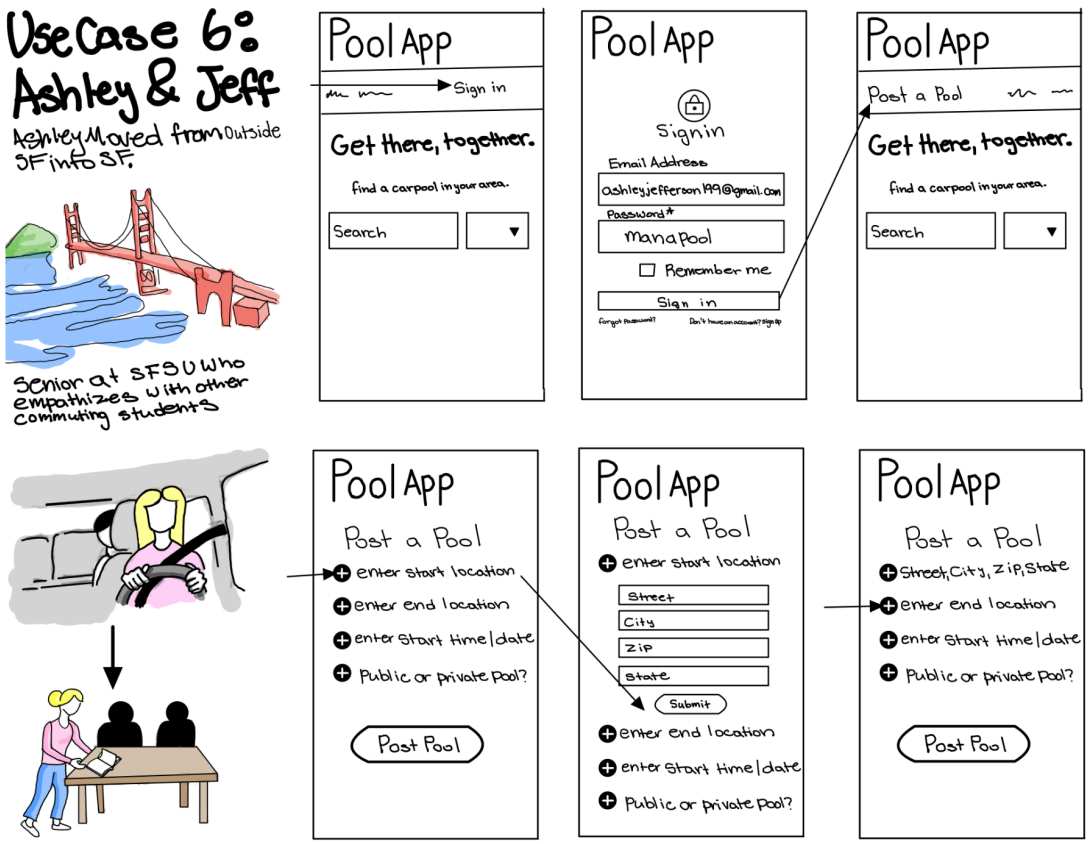


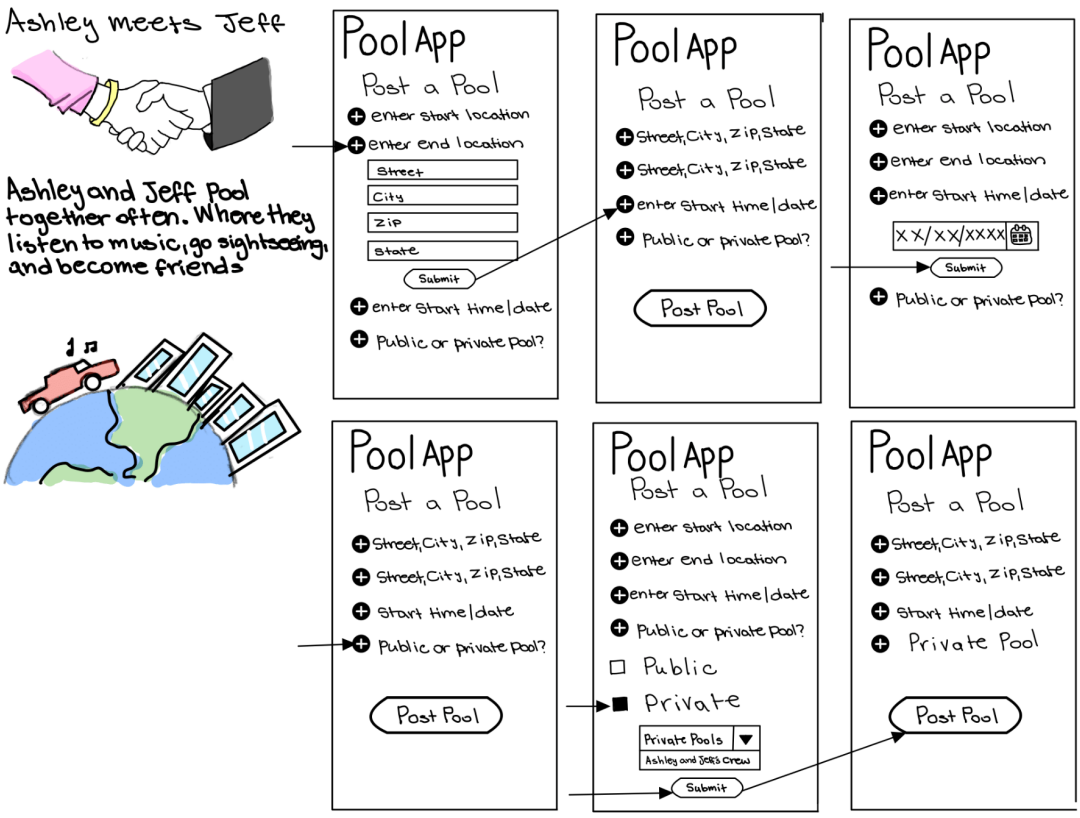

Pool App
Post a Pool
enter start location
enter end location
Street
City
Zip
State
Submit
enter start time|date
Public or private pool?
Pool App
Post a Pool
Street,City, zip,State
Street,City, zip,State
enter start time|date
Public or private pool?
Post Pool
Pool App
Post a Pool
enter start location
enter end location
enter start time|date
XX/XX/XXXX
Submit
Public or private pool?
Pool App
Post a Pool
Street,City, zip,State
Street,City, zip,State
Start time|date
Public or private pool?
Post Pool
Pool App
Post a Pool
enter start location
enter end location
enter start time|date
Public or private pool?
Public
Private
Private Pools
Submit
Pool App
Post a Pool
Street,City, zip,State
Street,City, zip,State
Start time|date
Public Pool
Post Pool

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 signs into his account under jjonthetrack@gmail.com and enters his password under manapool. 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!

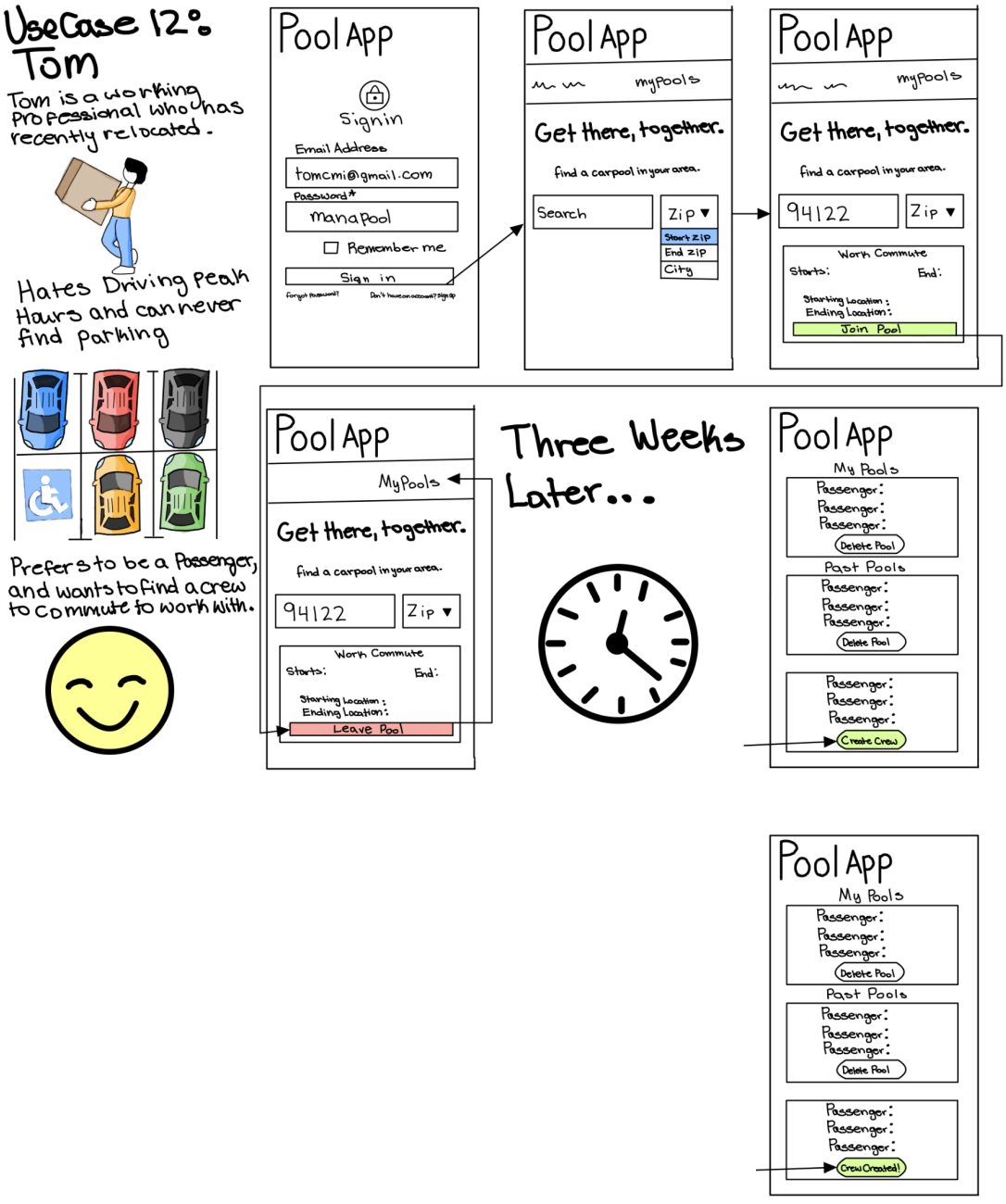


Use Case 6 - Ashley, a senior at SF State, uses the Pool app as a driver to offer rides to fellow students, alleviating expenses and helping newcomers navigate the city. She signs in under her account ashleyjefferson199@gmail.com and enters her password under manapool. She formed a two person crew with 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.




Ashley meets Jeff
Ashley and Jeff Pool together often. Where they listen to music, go sightseeing, and become friends
Pool App
Post a Pool
enter start location
enter end location
Street
City
Zip
State
Submit
enter start time/date
Public or private Pool?
Pool App
Post a Pool
Street, City, Zip, State
Street, City, Zip, State
enter start time/date
Public or private Pool?
Post Pool
Pool App
Post a Pool
enter start location
enter end location
enter start time/date
XX/XX/XXXX
Submit
Public or private Pool?
Pool App
Post a Pool
Street, City, Zip, State
Street, City, Zip, State
start time/date
Public or private Pool?
Post Pool
Pool App
Post a Pool
enter start location
enter end location
enter start time/date
Public or private Pool?
Public
Private
Private Pools
Ashley and Jeff's Crew
Submit
Pool App
Post a Pool
Street, City, Zip, State
Street, City, Zip, State
start time/date
Private Pool
Post Pool

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, so for the past month, has been using Pool in order to commute to work as a passenger.He signs in under tomcmi@gmail.com and enters his password under manapool and find a pool, 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.



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 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

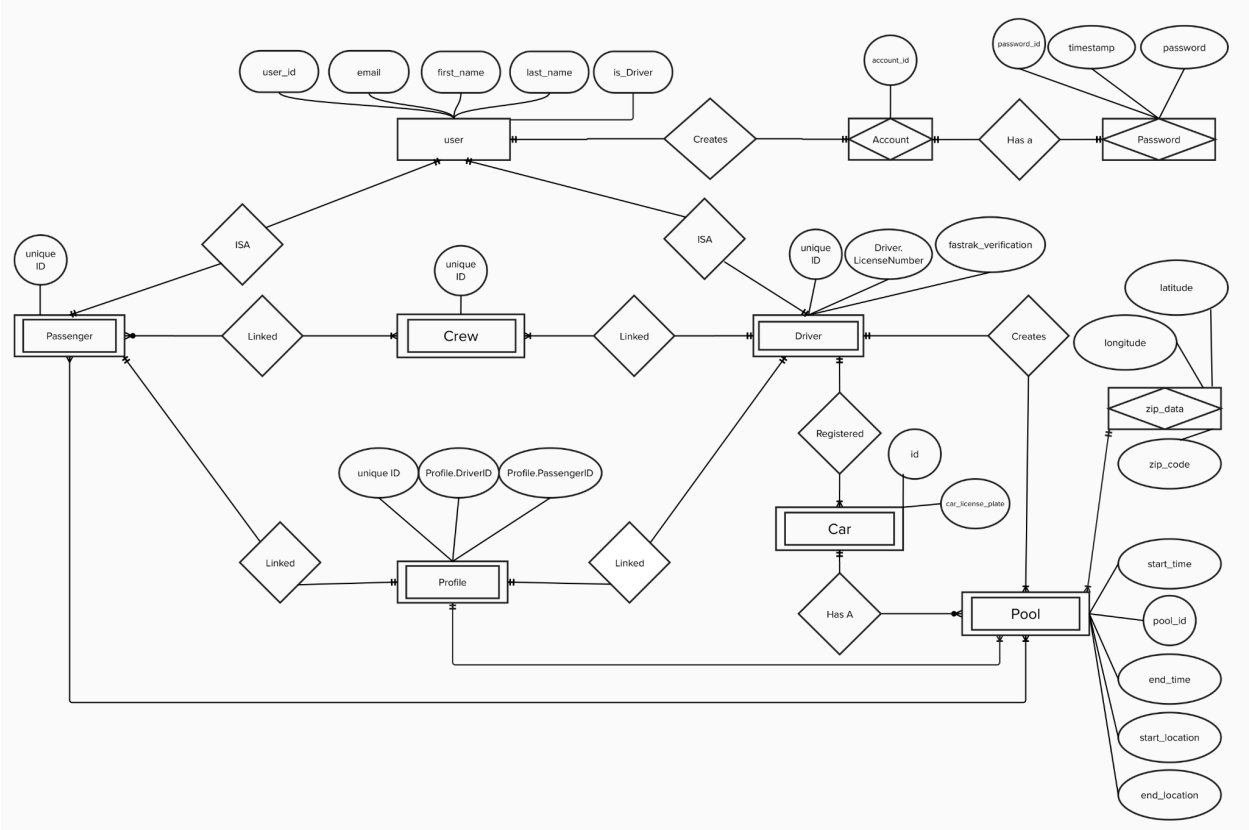
Ride request: Ride requests represent the request from users for a ride in the carpooling app

- 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.

Media Storage

Pool contains no user generated media.

Search Filter/Architecture and Implementation

Spring JPA is used to persist data between java objects and the database. The searches below are encapsulated within methods. Below is a list of supported search terms:

Search by start location zip code

Users search for pools by providing a zip code correlating to the pool's start location. The table **pool** is then searched on attribute **start_zip.** This is matched against the **zip_data** table by the **zip_code** attribute. (See "Pool Proximity Algorithm" in the section below for more details.)

Returned results are sorted by proximity using the “Pool Proximity Algorithm” when displayed on the frontend.

Search by end location zip code

Users search for pools by providing a zip code correlating to the pool’s end location. The table **pool** is then searched on attribute **end_zip.** This is matched against the **zip_data** table by the **zip_code** attribute. (See “Pool Proximity Algorithm” in the section below for more details.)

Returned results are sorted by proximity using the “Pool Proximity Algorithm” when displayed on the frontend.

Search by city

Users search for pools by providing a city correlating to the pool’s start or end location. The table **pool** is then searched on attributes **start_city** or **end_city** for a direct match. Results returned on the frontend are sorted by start time with the pool starting closest to now displaying at the top.

Search for user at login

The system searches for users when login is attempted by referencing the unique attribute **email** in the **user** table, in addition to **password** in the **password** table.

Search for user at account creation

The system searches for users when account creation is attempted by referencing the unique attribute **email** in the **user** table.

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.

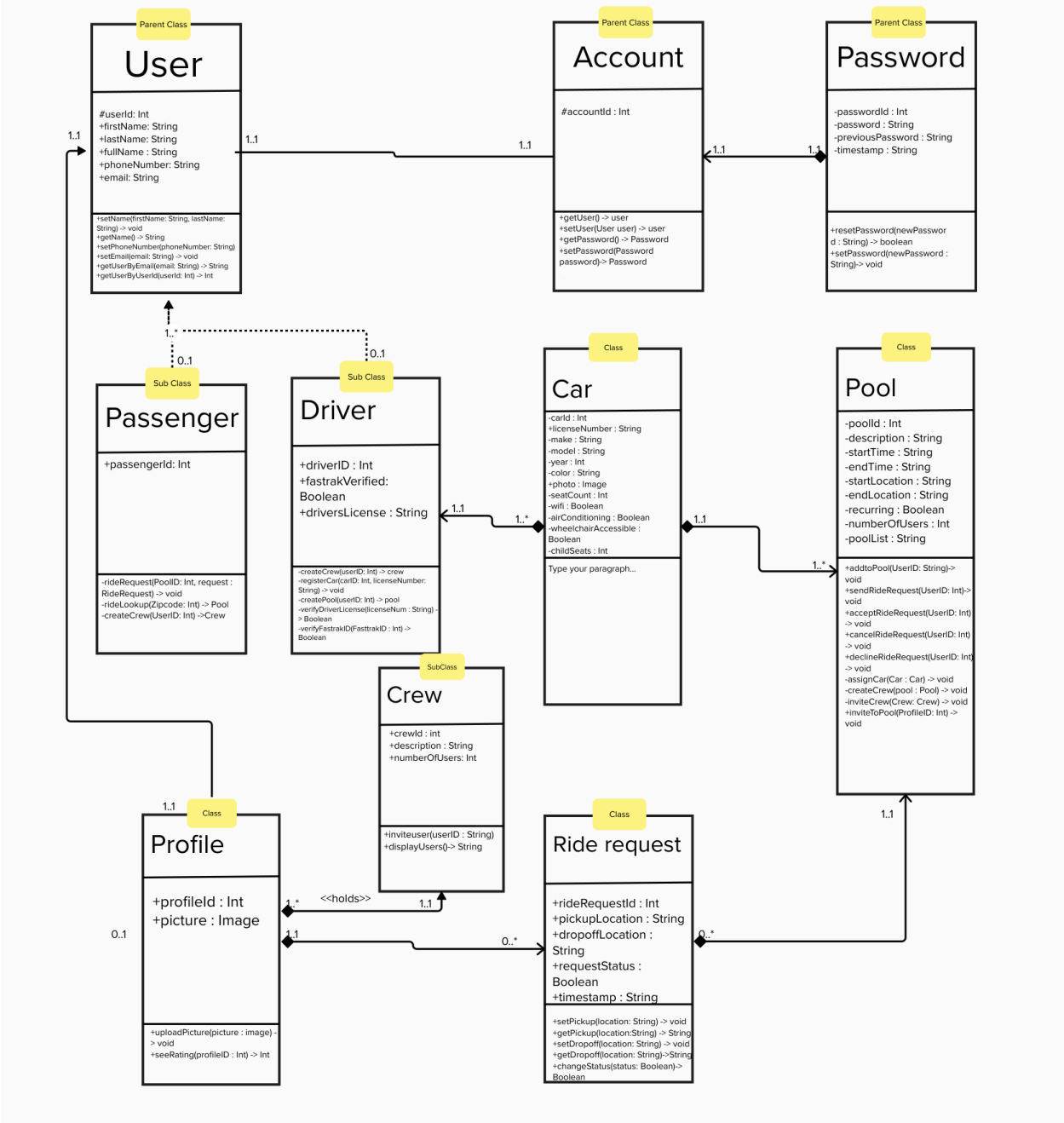
Pool Proximity Algorithm

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:

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

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 coordinate of every pool (carpool) in the pool table to find those within a five mile radius.

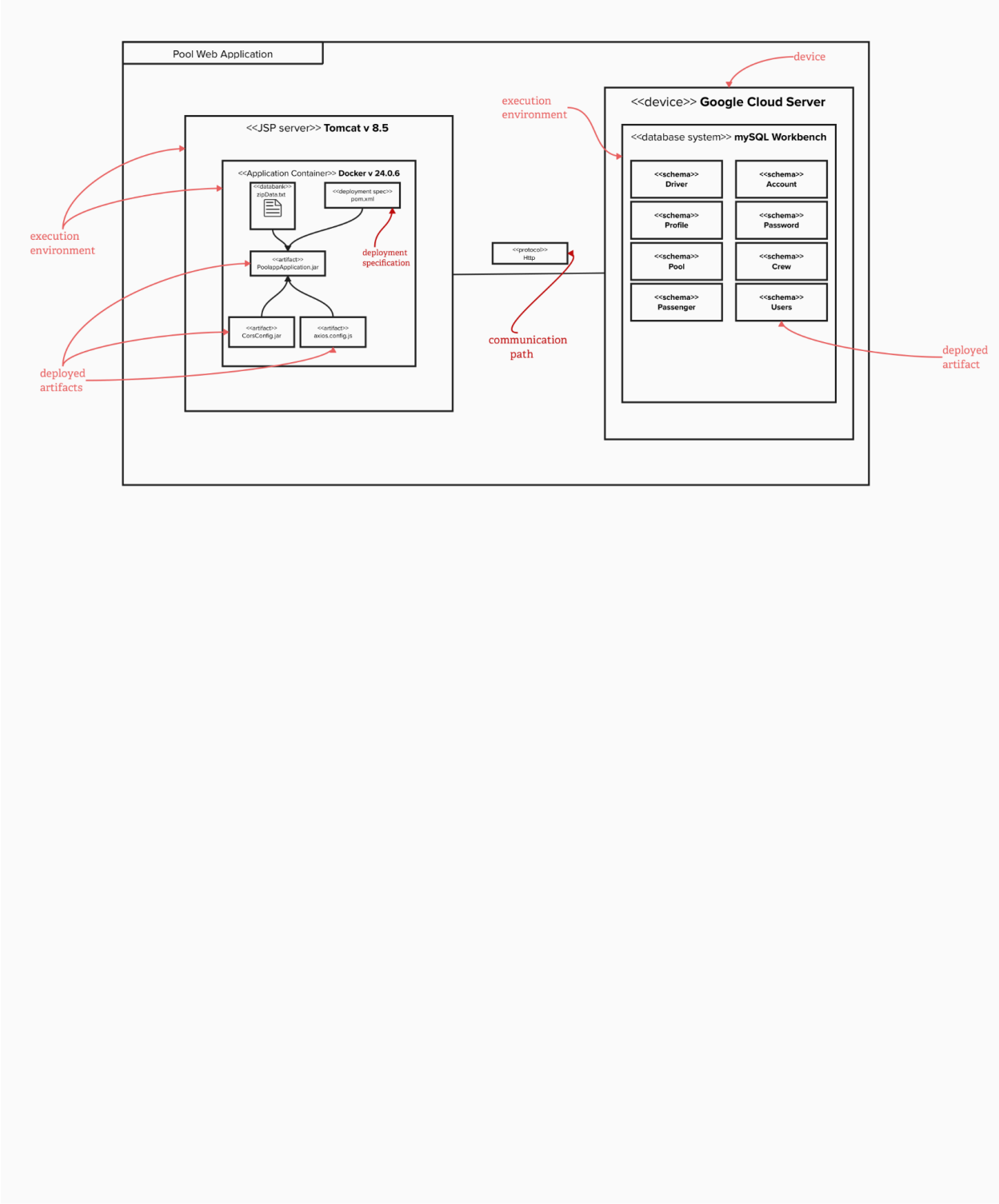
6. High Level UML Diagram



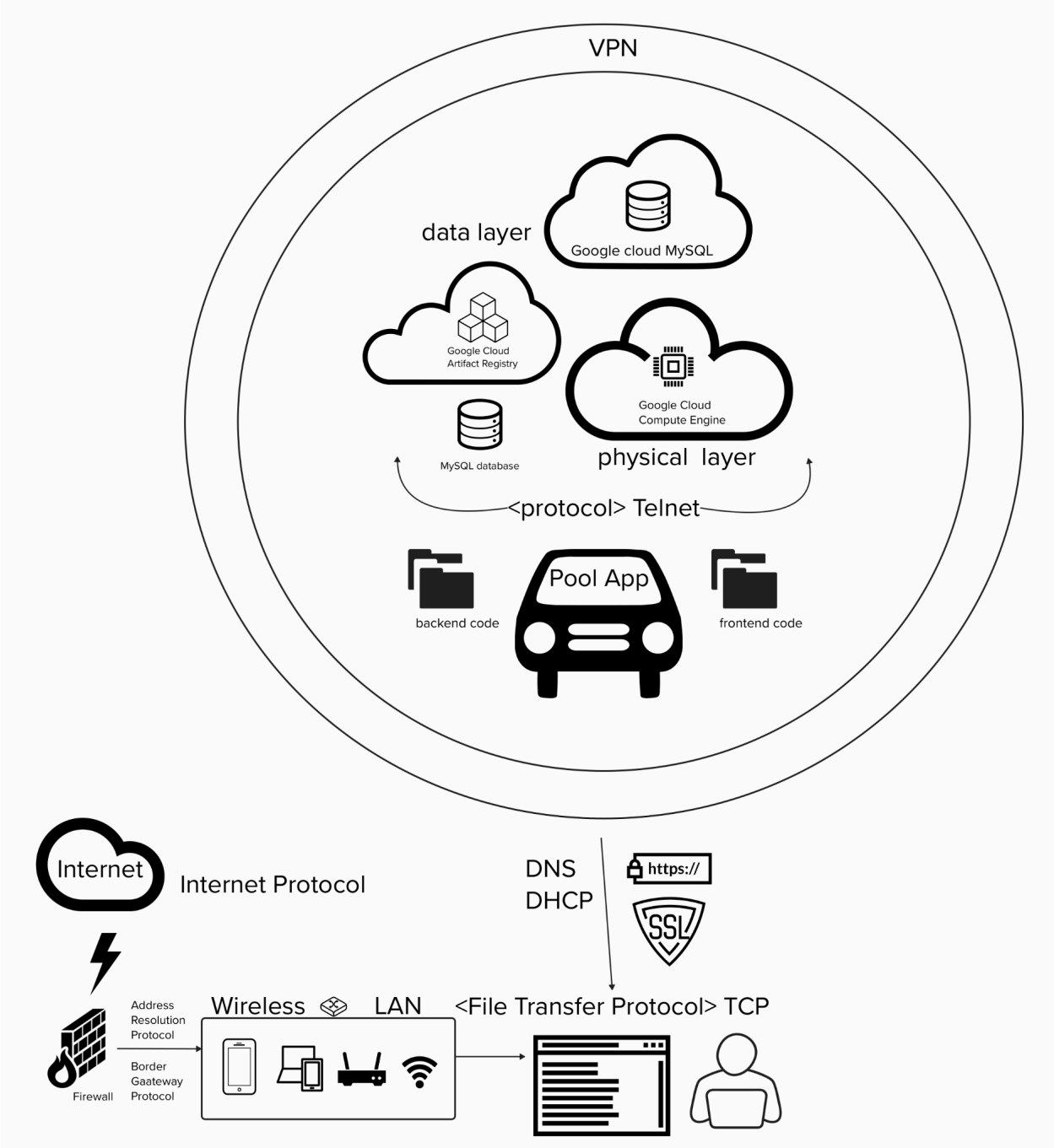
(Relationship from Profile to Crew is inheritance)

7. High Level Application Network & Deployment Diagrams

Deployment Diagram



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: (V1) worked collaboratively to prioritize functional requirements on a team group call, (V2) worked collaboratively to re-prioritize functional requirements on a team group call
- ✅Jonathan: (V1) worked collaboratively to prioritize functional requirements on a team group call, (V2) worked collaboratively to re-prioritize functional requirements on a team group call
- ✅Kendrick: (V1) worked collaboratively to prioritize functional requirements on a team group call, (V2) worked collaboratively to re-prioritize functional requirements on a team group call
- ✅Kristian: (V1) worked collaboratively to prioritize functional requirements on a team group call, (V2) worked collaboratively to re-prioritize functional requirements on a team group call
- ✅Phillip: (V1) worked collaboratively to prioritize functional requirements on a team group call, (V2) worked collaboratively to re-prioritize functional requirements on a team group call
- ✅Xuan: (V2) worked collaboratively to re-prioritize functional requirements on a team group call
- ✅Zoe: (V1) organized and facilitated group working call during which functional requirements were prioritized

Prioritized Use Cases

- ✅Jonathan: (V1) 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: (V1) 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: (V1) organized and facilitated group working call during which use cases were prioritized

UI Mockups & Storyboards

- ✅Isiah: (V1) 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, (V2) with input from Zoe, fully revised all storyboard style mockups to better match reprioritized functional requirements
- ✅Phillip: (V1) 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: (V1) organized and facilitated group working calls during which these activities occurred

Entity Relationship Diagrams

- ✅Jonathan: (V1) owned the ERD from draft to final phase with support from Zoe, (V2) fully owned ERD revisions as per system updates and feedback
- ✅Zoe: (V1) supported Jonathan as needed

High Level APIs and Main Algorithms

- ✅Kendrick: (V1) collaboratively wrote the “Google maps JavaScript API or competitor” section with Zoe
- ✅Kristian: (V1) collaboratively wrote the “Hashing Methods” and “React Material UI” sections with Zoe and independently wrote the “Location Based Search” section
- ✅Phillip: (V1) independently wrote the “Axios” section
- ✅Zoe: (V1) supported Kristian and Kendrick in writing their respective sections

UML Diagram

- ✅Isiah: (V1) worked collaboratively on first draft
- ✅Jonathan: (V1) worked collaboratively on first draft and fully owned all future states of the diagram, including final delivery, (V2)
- ✅Phillip: (V1) worked collaboratively on first draft
- ✅Zoe: (V1) facilitated coworking sessions and supported Jonathan as needed

Network Diagram

- ✅Kristian: (V1) worked collaboratively on first draft
- ✅Jonathan: (V1) worked collaboratively on first draft
- ✅Phillip: (V1) worked collaboratively on first draft
- ✅Kendrick: (V1) owned all future revisions with exception to final version
- ✅Zoe: (V1) facilitated coworking sessions, supported Kendrick as needed, made final round of revisions, (V2) fully owned revisions as per feedback

Identified Risks

- ✅Isiah: (V1) worked collaboratively on risks on a team coworking call
- ✅Kendrick: (V1) worked collaboratively on risks on a team coworking call
- ✅Zoe: (V1) facilitated coworking call during which this was worked on

JPA Integration

- ✅Kristian: (V1) did preliminary research on JPA vs JDBC and supported Phillip as needed
- ✅Phillip: (V1) owned and implemented JPA integration
- ✅Zoe: (V1) facilitated coworking calls during which this was worked on

React Integration and First Frontend Iteration

- ✅Kristian: (V1) 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: (V1) wrote associated Jira tickets and supported as needed

Classes and Methods

- ✅Jonathan: (V1) 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: (V1) supported and contributed to the creation of methods, wrote all queries required by methods
- ✅Phillip: (V1) owned and implemented Classes and methods
- ✅Zoe: (V1) facilitated coworking calls during which this was worked on

Connecting Frontend and Backend

- ✅Kristian: (V1) owned the implementation and delivery of endpoints
- ✅Phillip: (V1) set up initial connection and worked collaboratively with Kristian throughout endpoint creation process
- ✅Xuan: (V1) integrated app with Axios
- ✅Zoe: (V1) facilitated coworking calls during which this was worked on, wrote associated tickets

Documentation

- ✅Kendrick: (V1) drafted documentation for MacOS local setup in GitHub
- ✅Zoe: (V1) provided guidance and supported Kendrick as needed

Populating the Database

- ✅Phillip: (V1) 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: (V1) worked collaboratively with Phillip to populate the db's first tables (user, account, password, pool, plus zip codes)

Final Frontend Iteration

- ✅Isiah: (V1) provided images for use by various site pages
- ✅Xuan: (V1) fully owned frontend revisions and refinement including delivery of final product
- ✅Zoe: (V1) wrote tickets and played a consulting role for requirements

Deployment

- ✅Xuan: (V1) 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: (V1) provided guidance and supported Kendrick as needed

Pool - carpooling app

SW Engineering CSC648-848-05 Fall 2023

Milestone 3 V2 | 11/25/2023

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

History Table

Milestone	Date
M3V2	**11/25/23**
M3V1	**11/02/23**
M2V2	**11/02/23**
M2V1	**10/12/23**
M1V2	**10/8/23**
M1V1	**9/21/23**

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. Reprioritized Functional Requirements

Priority 1

1. User

1.1 A user shall create an account
1.2 A user shall register as a driver
1.3 A user shall register as a passenger
1.4 A user shall sign in
1.5 A user shall log out

2. Profile

2.1 A profile shall belong to a passenger
2.2 A profile shall belong to a driver
2.3 A profile shall contain user details
2.4 A profile shall be viewed by the user it is associated with

3. Driver

3.1 A driver shall be Fastrak verified
3.2 A driver shall create zero or many pools
3.3 A driver shall have zero or many crews
3.4 A driver shall have a driver's license
3.5 A driver shall view the pools they posted
3.6 A driver shall view the crews they are a member of
3.7 A driver shall leave the crews they are a member of

4. Passenger

4.1 A passenger shall join many pools
4.2 A passenger shall have many crews
4.3 A passenger shall view the pools they have joined
4.4 A passenger shall view the crews they are a member of
4.5 A passenger shall leave the crews they are a member of
4.6 A passenger shall leave the pools they are a member of

5. Crew

5.1 A crew shall be created by one user

5.2 A crew shall have a description

5.3 A crew shall have members

6. Pool

6.1 A pool shall have a description

6.2 A pool shall have a start time.

6.3 A pool shall have passengers.

6.4 A pool shall have one driver

6.5 A pool shall have a start location

6.6 A pool shall have an end location

6.7 A pool shall be deleted by its driver

Priority 2

1. User
   - 1.1. A user shall be able to view their passenger ratings received
   - 1.2. A user shall be able to view their driver ratings received
   - 1.3. A user shall be able to view their passenger feedbacks received
   - 1.4. A user shall be able to view their driver feedbacks received

2. Profile
   - 2.1. A profile shall contain payment details
   - 2.2. A profile shall display the average rating
   - 2.3. A profile shall display ride history

3. Driver
   - 3.1. A driver shall have ratings
   - 3.2. A driver shall send ride requests for their pool(s)

4. Passenger
   - 4.1. A passenger shall have ratings
   - 4.2. A passenger shall send invites for crews.

5. Crew
    5.1. A crew creator shall be able to remove users from their crew
    5.2. A crew invite shall be accepted
    5.3. A crew invite shall be declined

6. Pool
    6.1. A pool shall have an occurrence rate.
    6.2. A pool shall have a total distance in miles
    6.3. A pool shall have an estimated trip duration
    6.4. A pool shall have ride requests.

7. Ratings
    7.1. A rating shall have a numerical value of 1-5
    7.2. A rating for a driver shall be provided by a passenger belonging to a common pool
    7.3. A rating for a passenger shall be provided by a driver belonging to a common pool
    7.4. A rating for a car shall be provided by a passenger belonging to a common pool

8. Ride Requests
    8.1. A ride request shall have seats
    8.2. A ride request shall contain the description associated with the requesting Crew
    8.3. A ride request shall contain a clickable name of the profile associated with the requesting Passenger(s) or Crew
    8.4. A ride request shall contain the description associated with the requesting Crew
    8.5. A ride request shall have zero or one pickup location(s)
    8.6. A ride request shall have zero or one dropoff location(s)
    8.7. A ride request shall be made by passengers
    8.8. A ride request shall be made for a Crew
    8.9. A ride request shall be made for a Pool

8.10. A ride request shall be accepted

8.11. A ride request shall be declined

9. Notifications

9.1. Notifications may be triggered by many actions.

Priority 3

1. User

1.1. A user shall have an emergency contact

1.2. A user shall be able to file a report ticket

1.3. A user shall be able to chat with support

1.4. A user shall be able to report any feedbacks

1.5. A user shall be able to delete their account

2. Driver

2.1. A driver shall have one or more cars

3. Car

3.1. A car shall be associated with a driver

3.2. A car shall be registered

3.3. A car shall have ratings

4. Profile

4.1. A profile shall display total distance traveled

4.2. A profile shall display total environmental impact

4.3. A profile shall display total number of crews with membership

5. Pool

5.1. A pool shall have a car

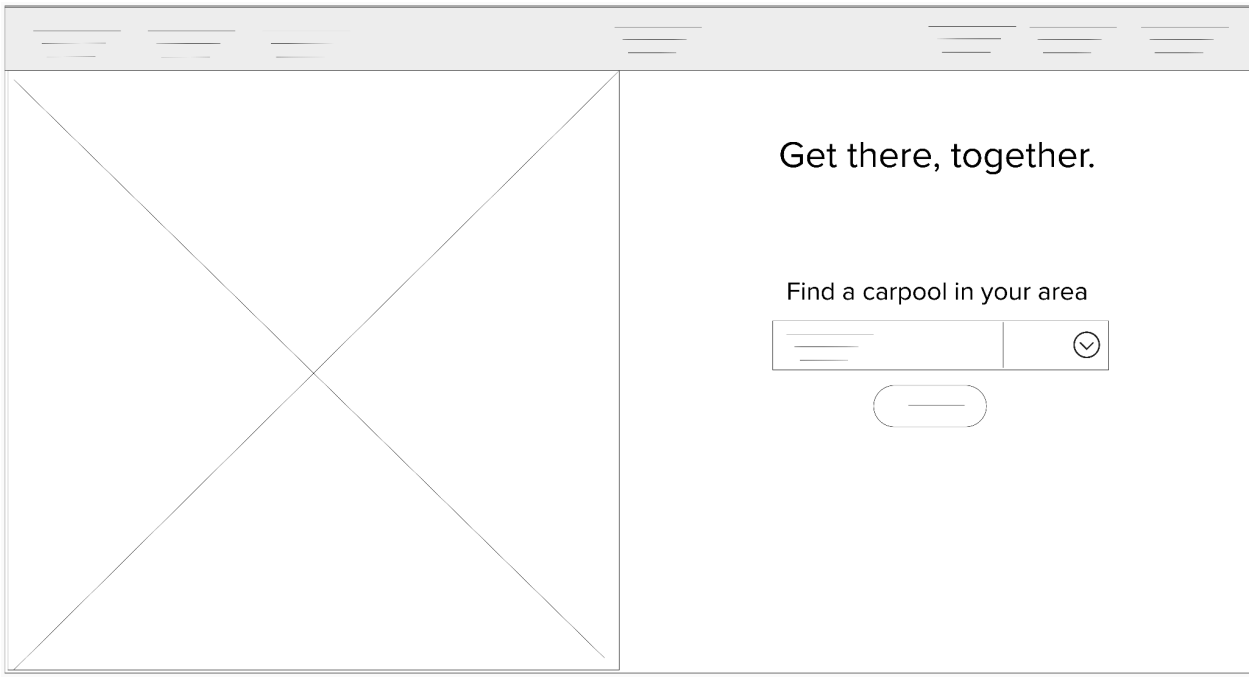
5.2. A pool shall be one-time or recurring.

6. Notifications

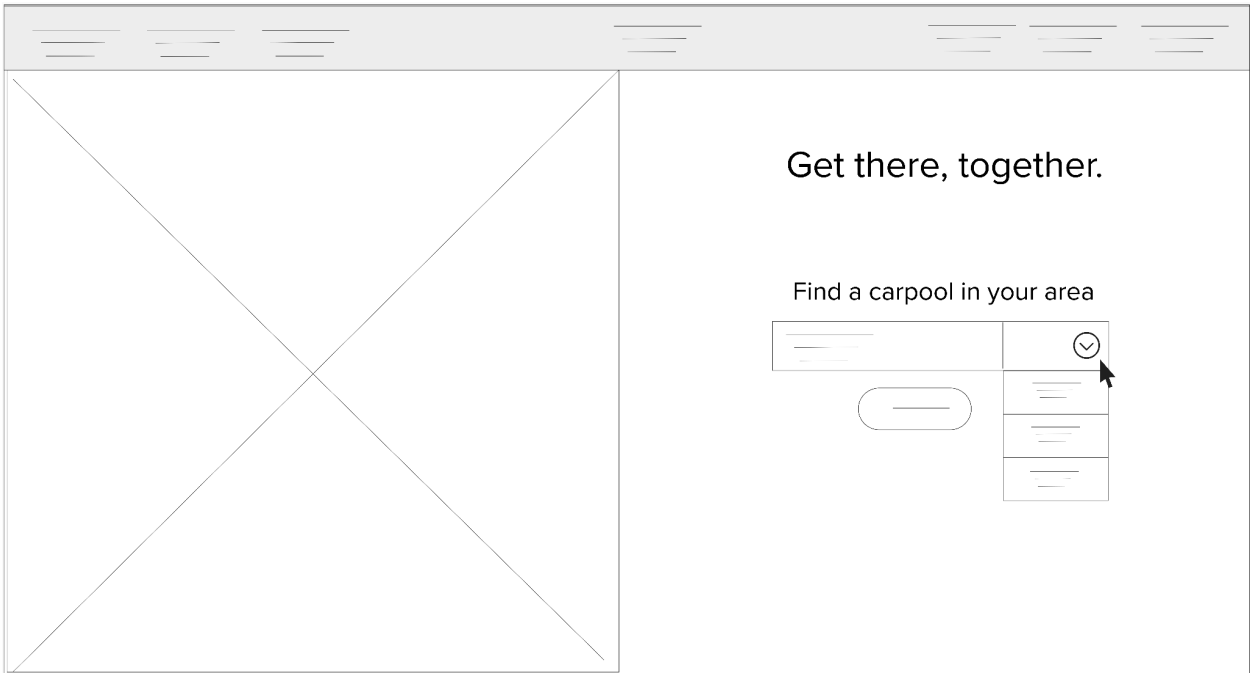
6.1. Notifications may be toggled on or off per account.

3. Wireframes

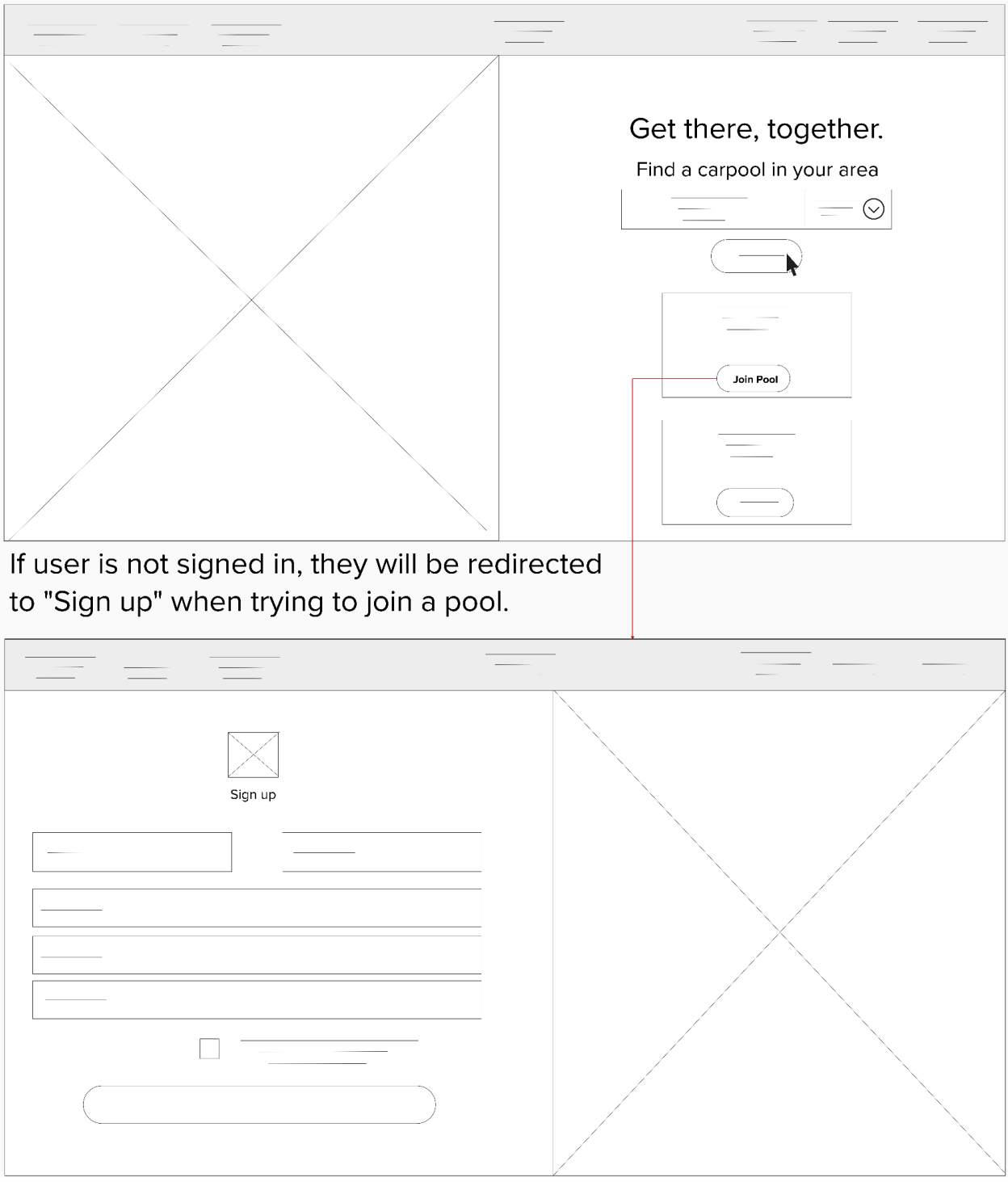
Home page



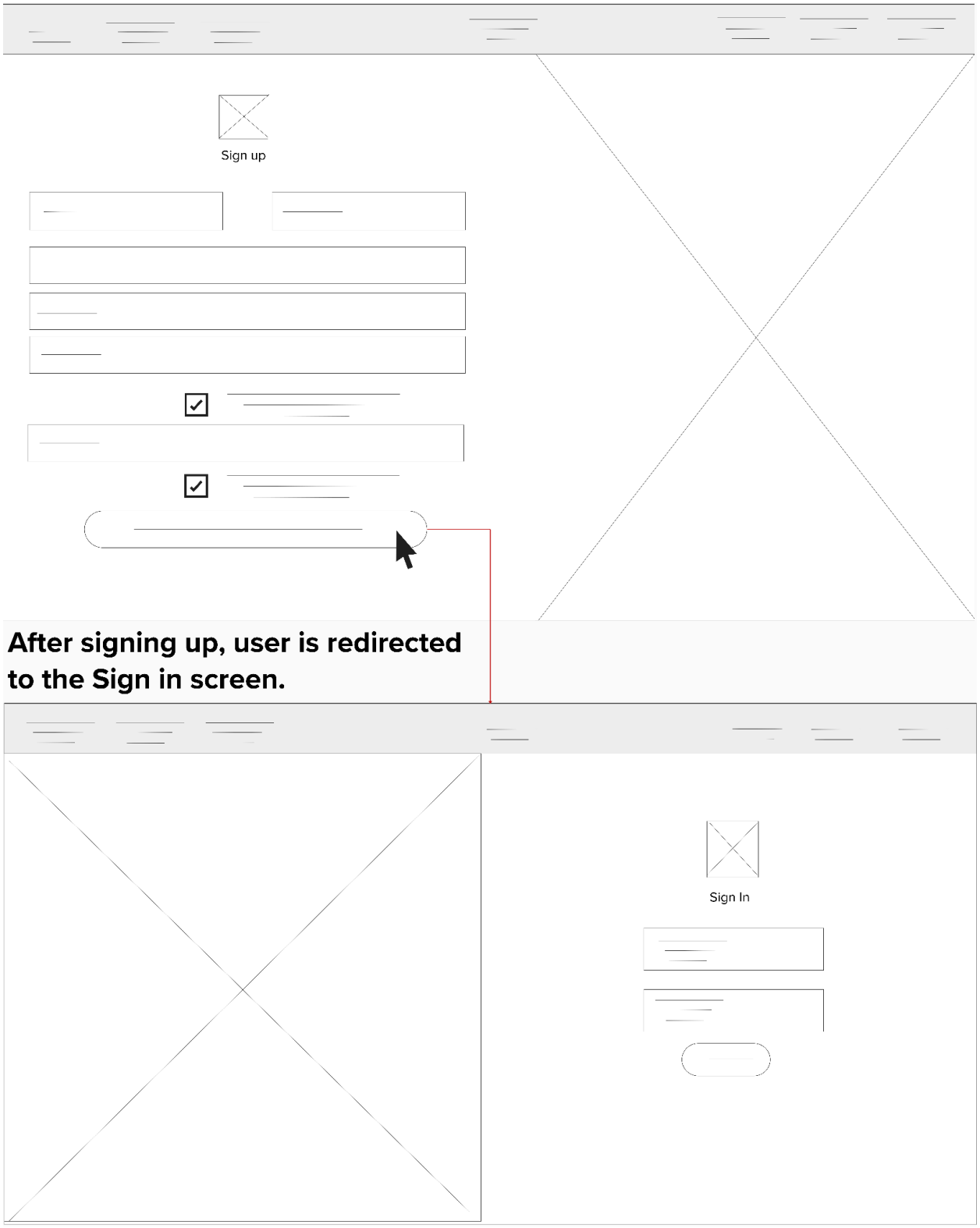
Home page with pulldown open (same screen)



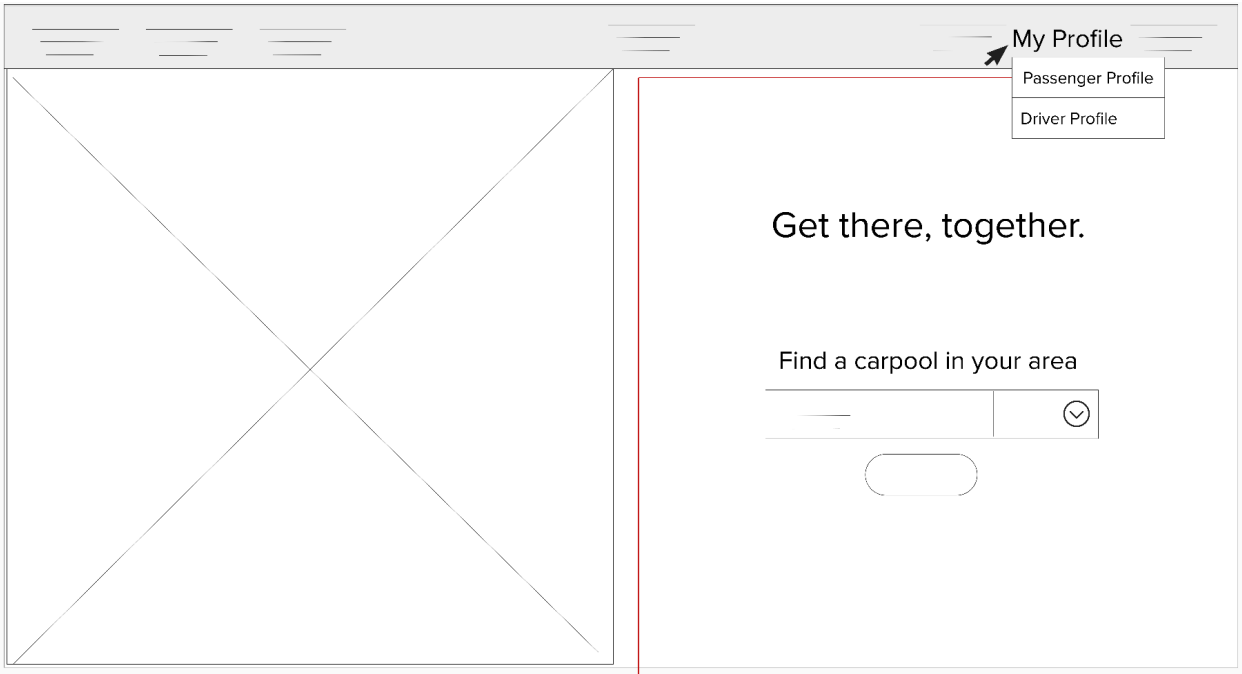
Home page with search results (same screen)



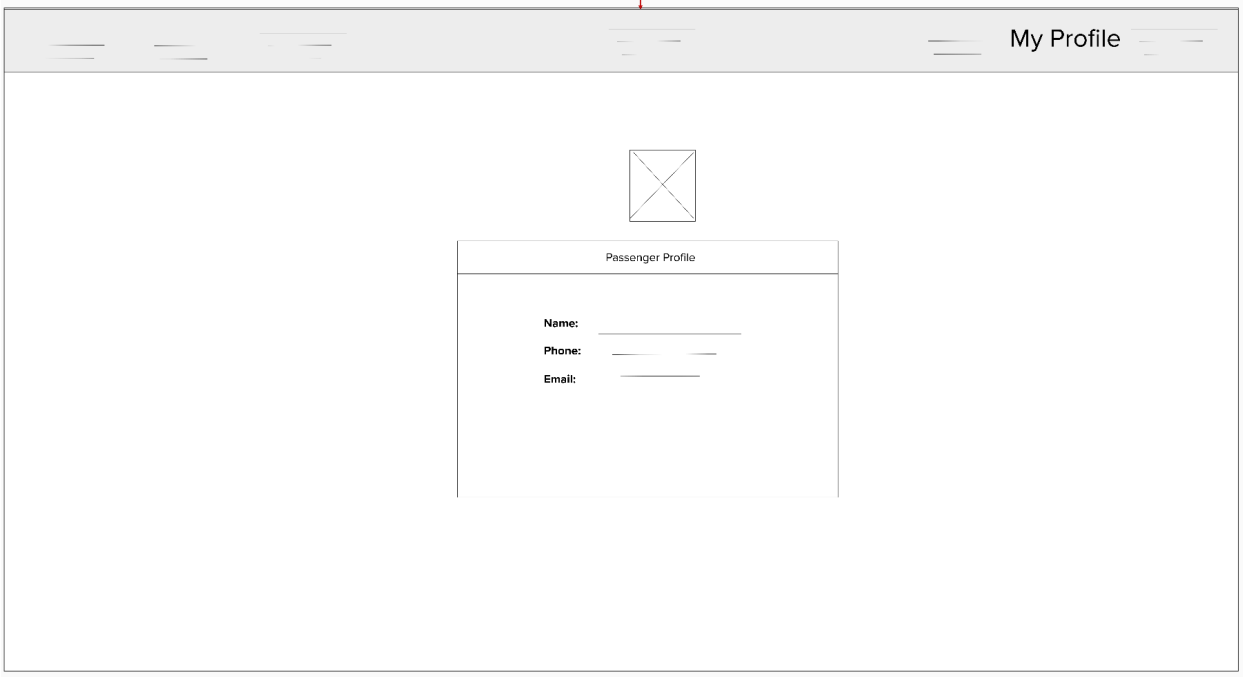
Sign up as a driver (same screen, additional fields displaying)



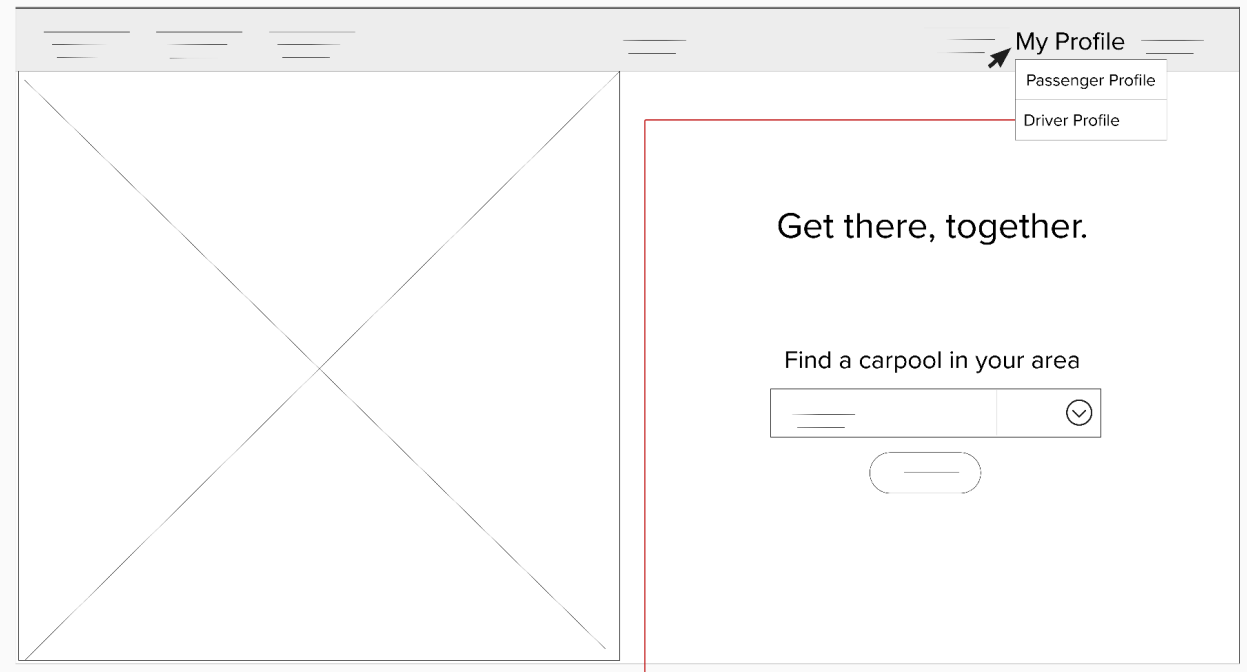
View passenger profile (new screen): once signed in, user is redirected to the home page where “My Profile” now displays in header menu



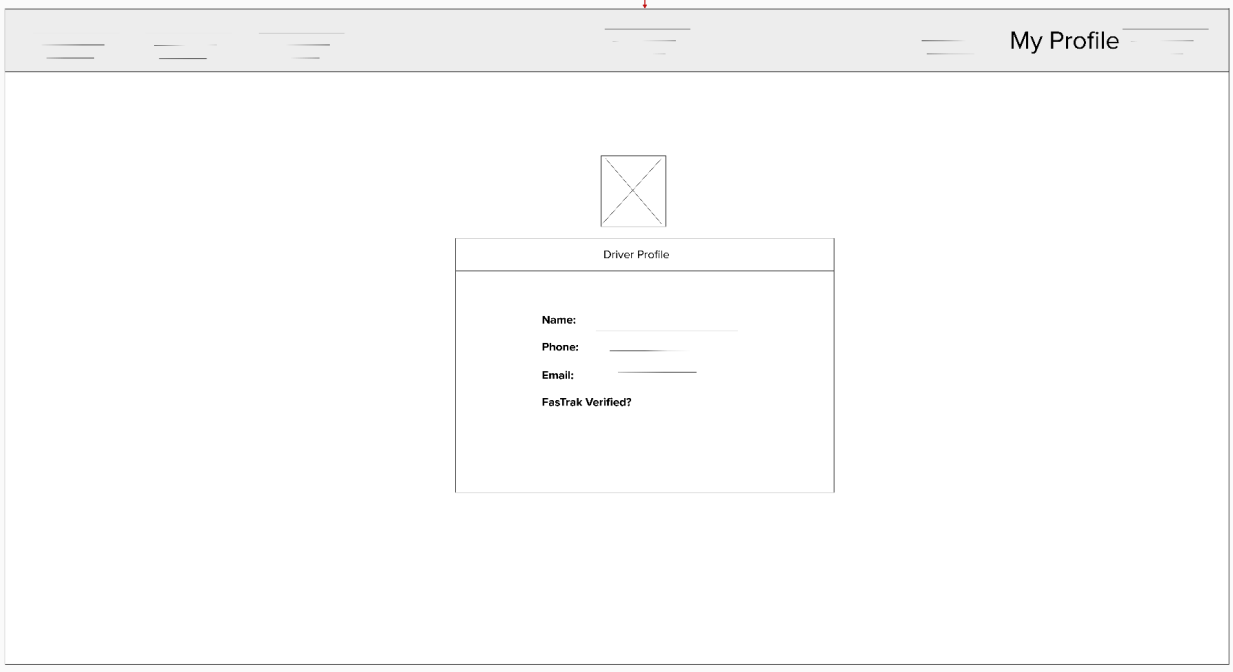
Passenger Profile



View driver profile (new screen): once signed in, user is redirected to the home page where “My Profile” now displays in header menu

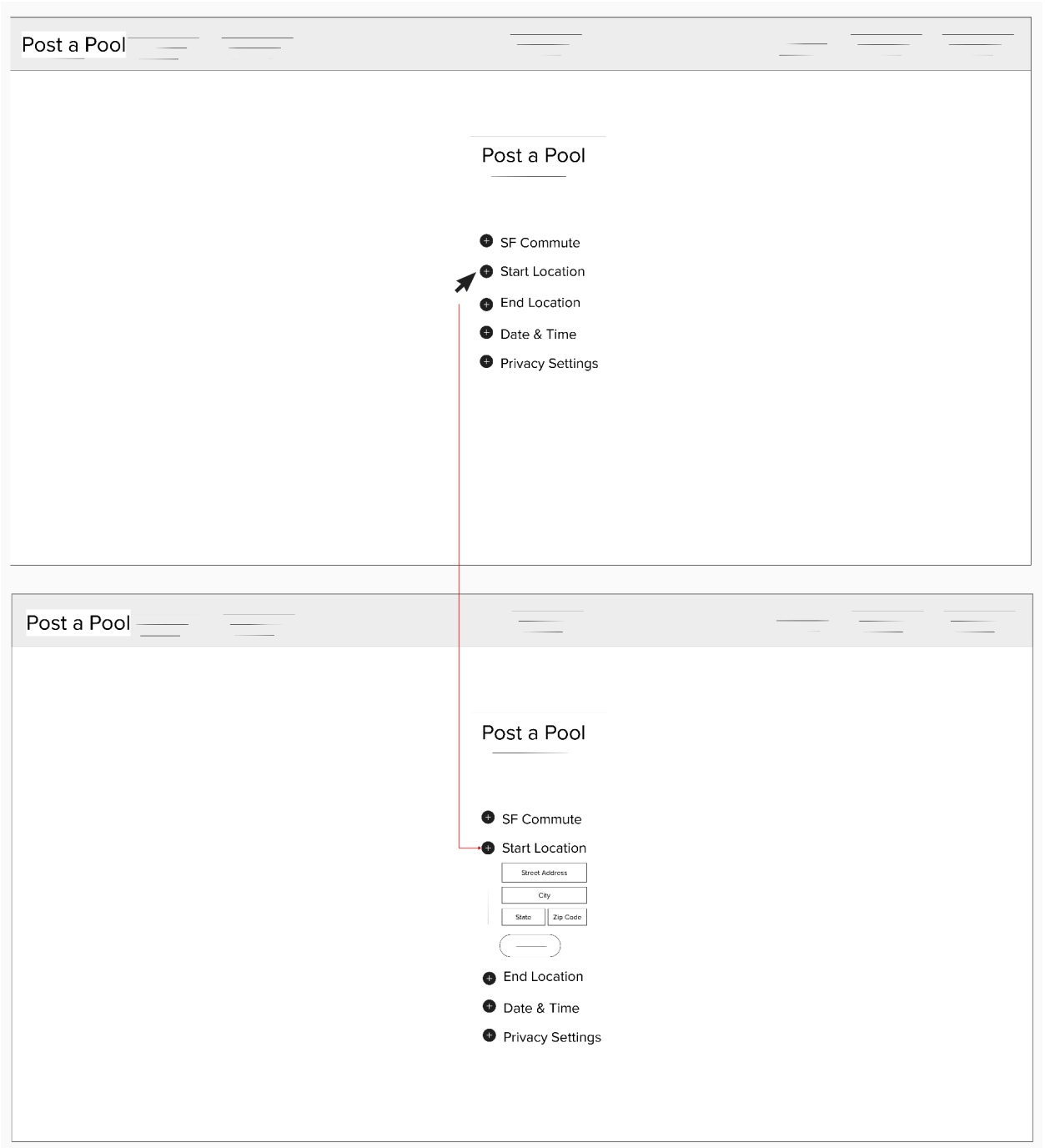


Driver Profile

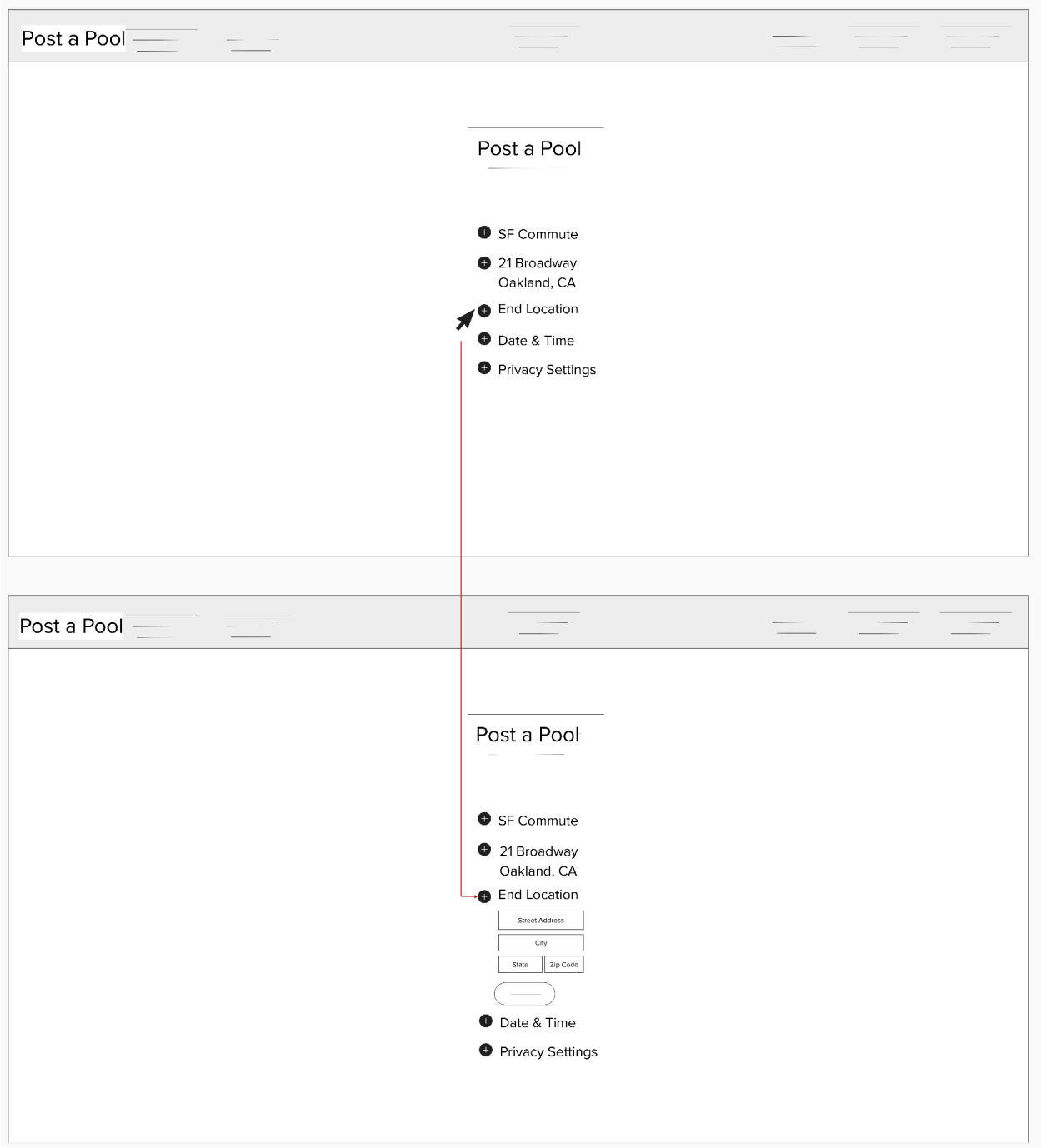


Post a pool: blank state and pool name (accessible from any page via header menu for authenticated users registered as a driver)

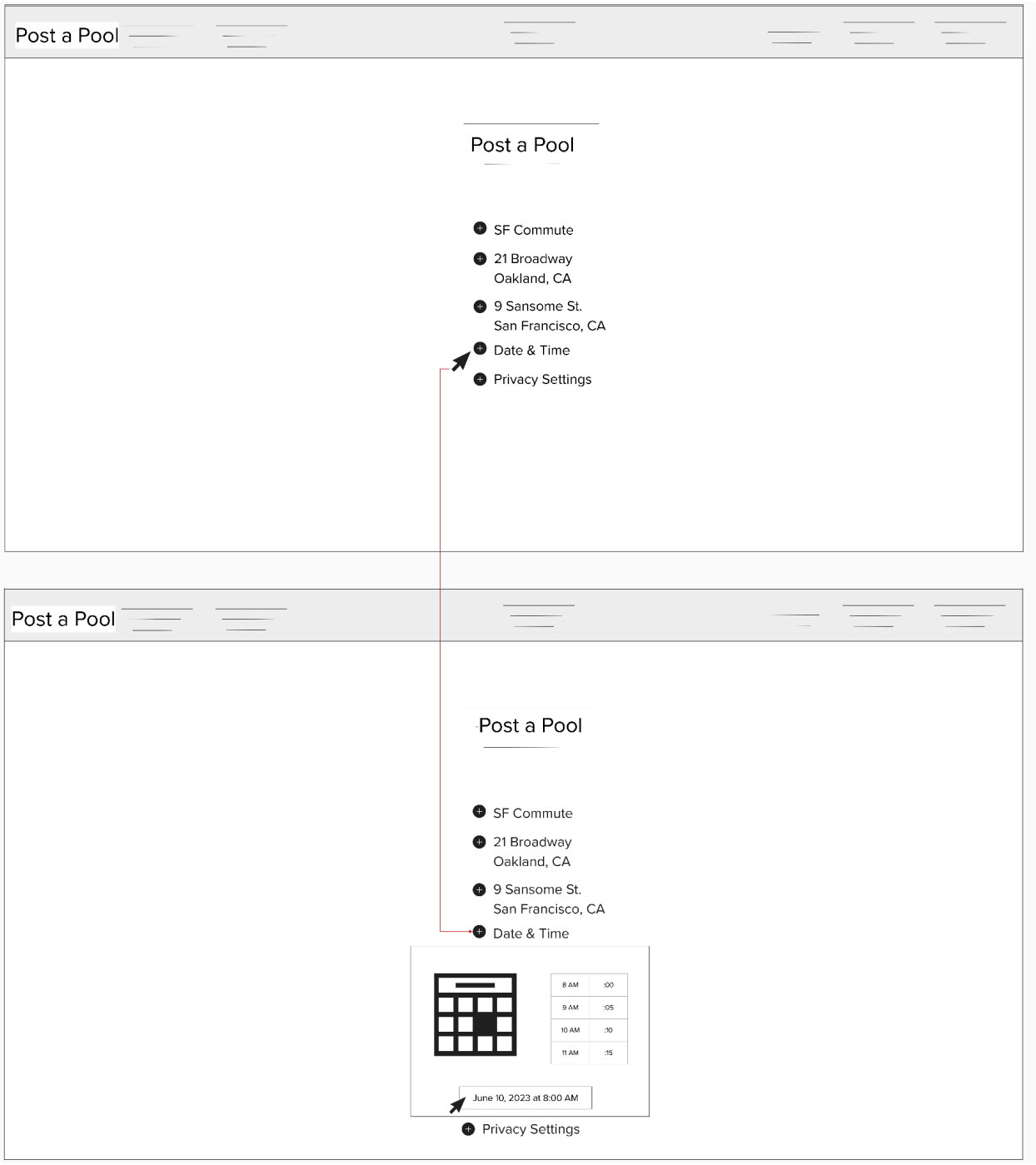
Post a pool: pool name (filled) and selecting & filling start location (same screen as previous page)



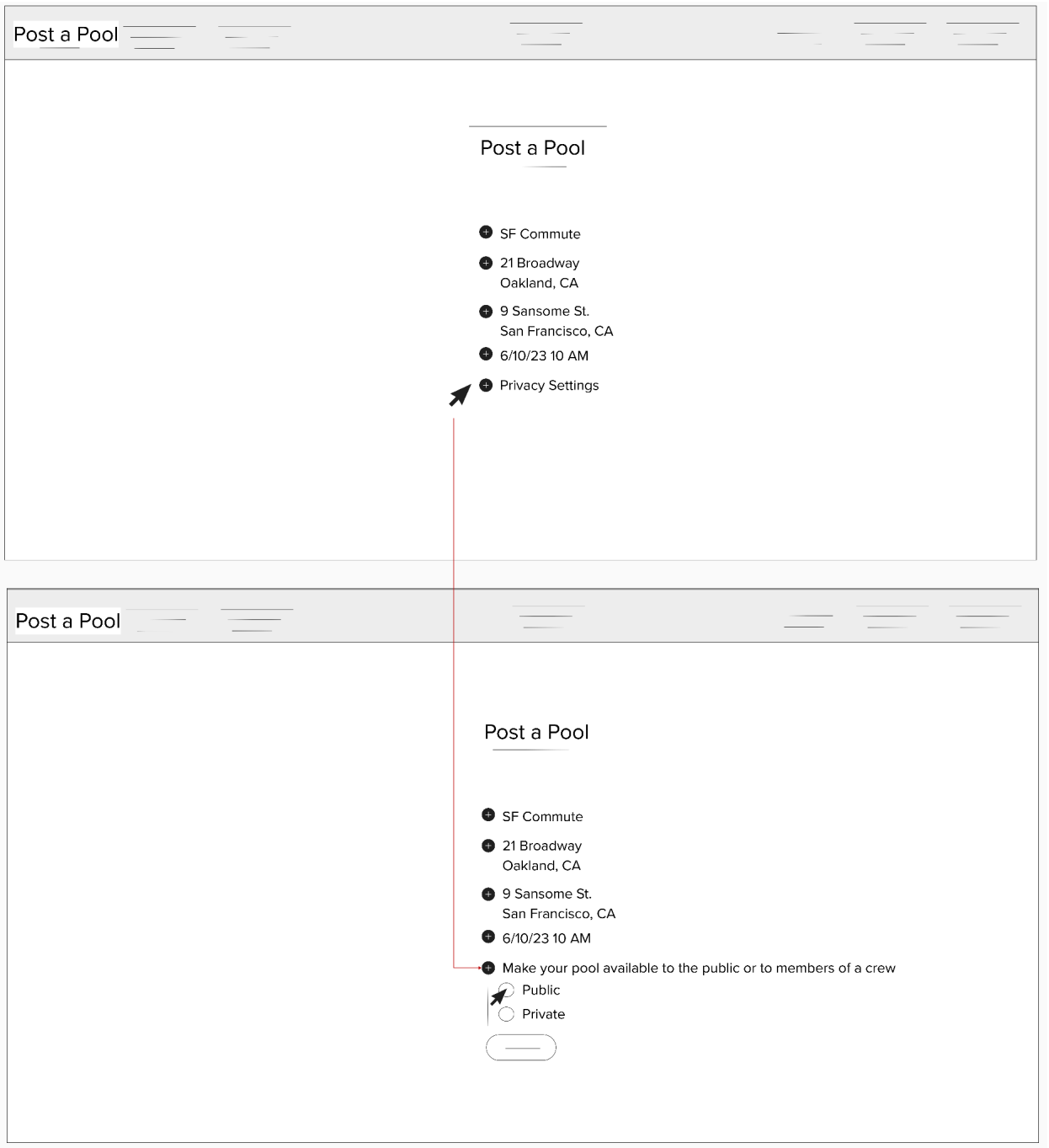
Post a pool: start location (filled) and selecting & filling end location (same screen as previous page)



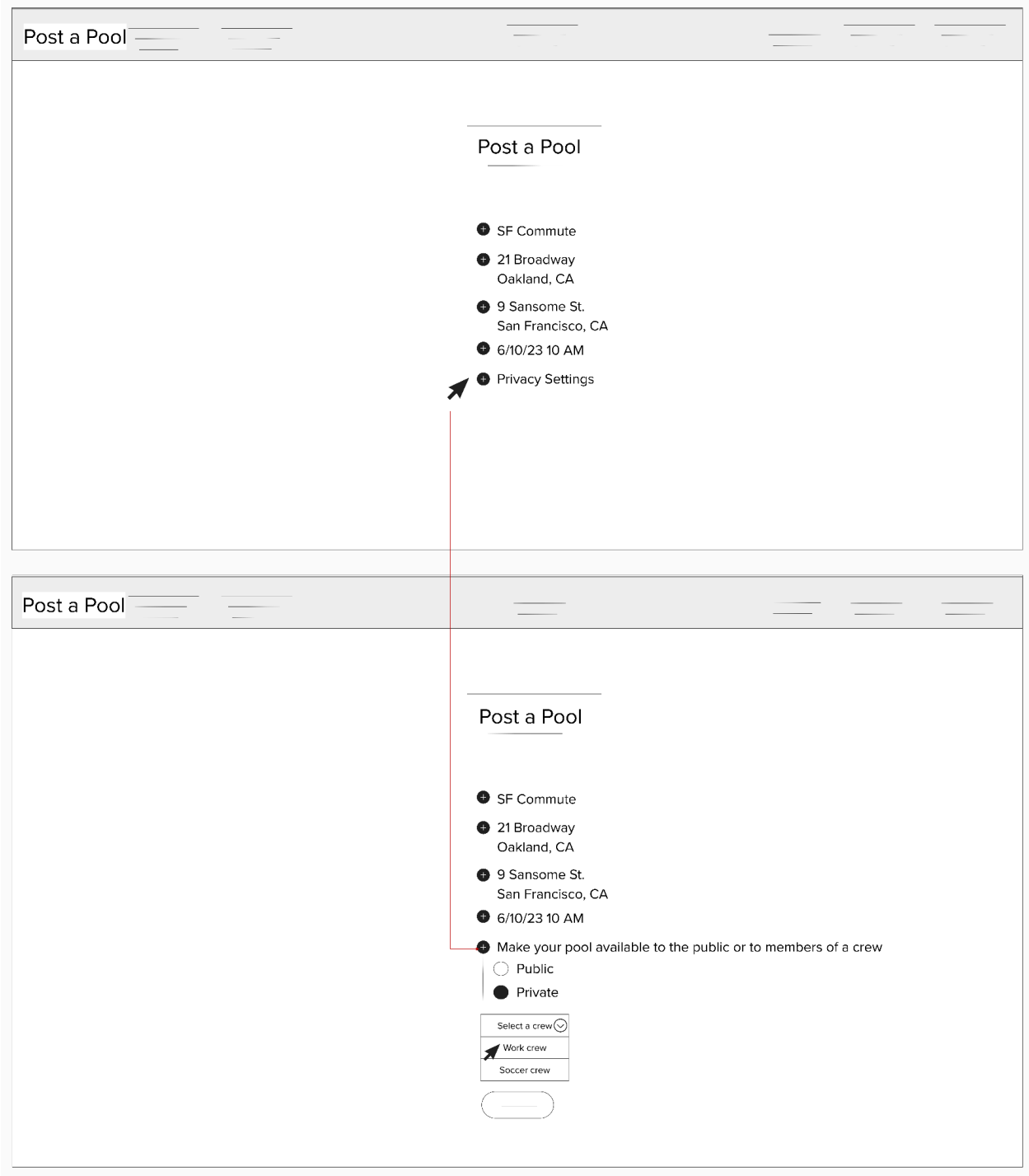
Post a pool: end location (filled) and selecting and filling date/time (same screen as previous page)



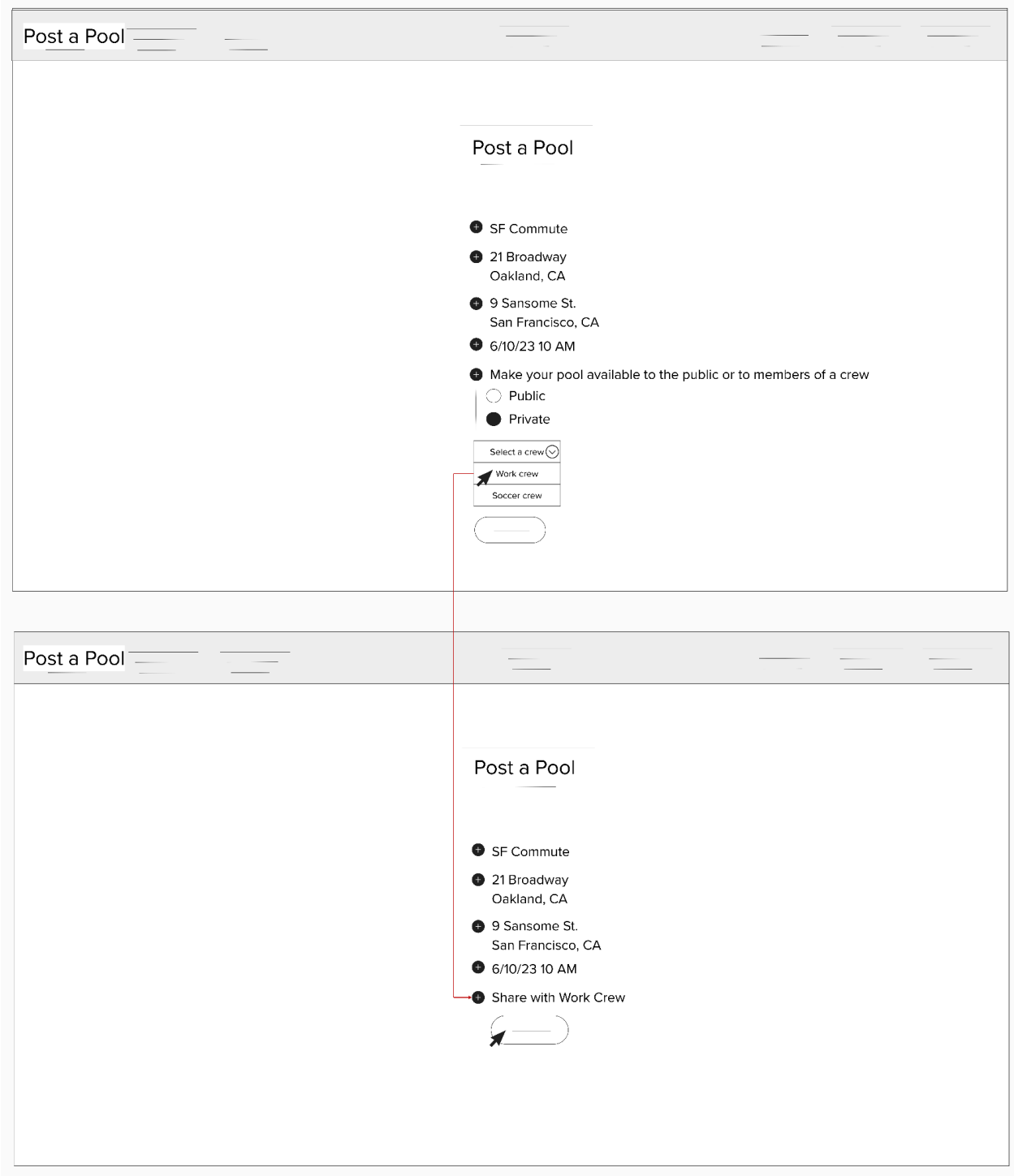
Post a pool: date/time (filled) and selecting “public” under privacy settings (same screen as previous page)



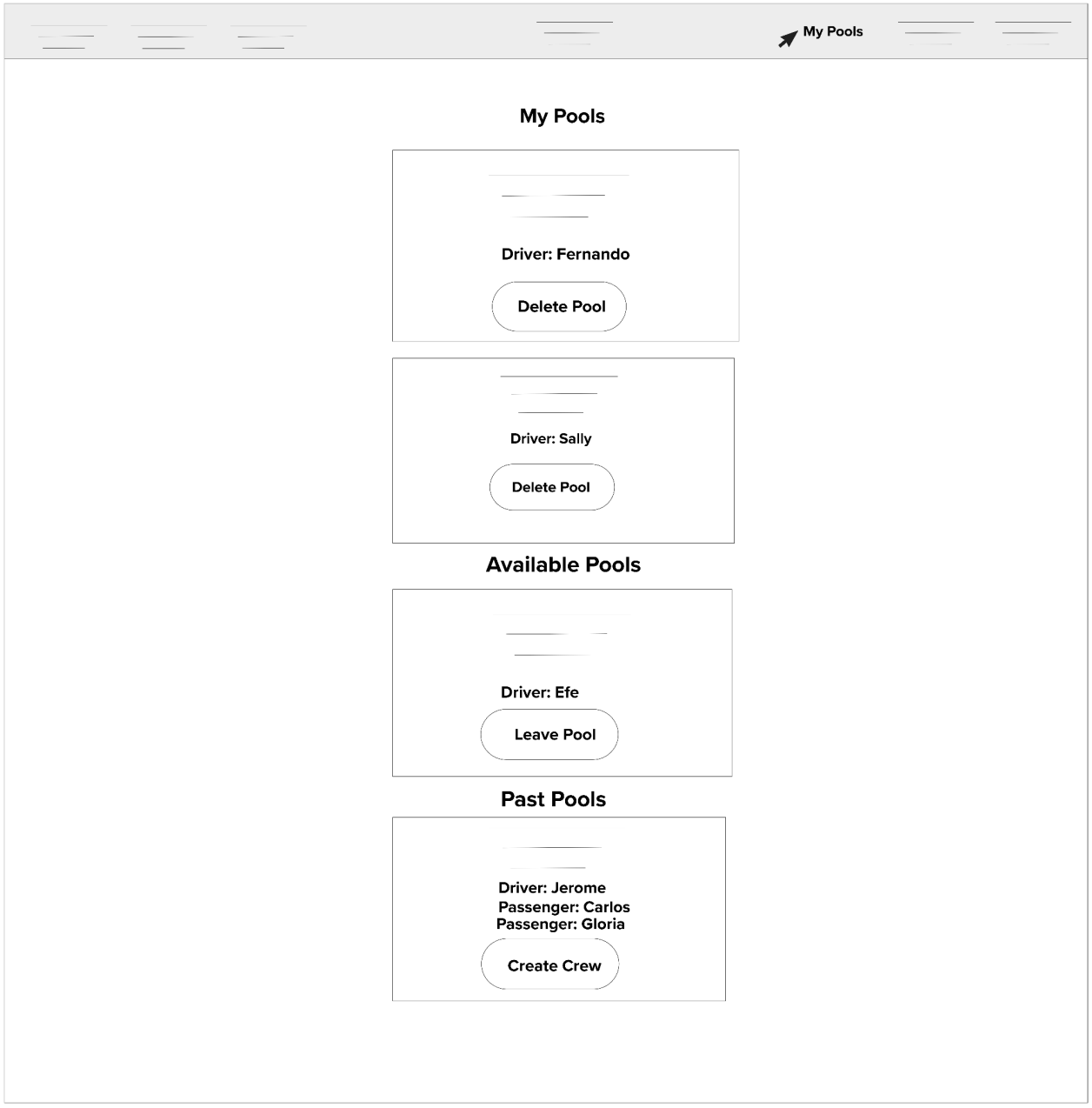
Post a pool: date/time (filled) and selecting & filling “private” under privacy settings (same screen as previous page)



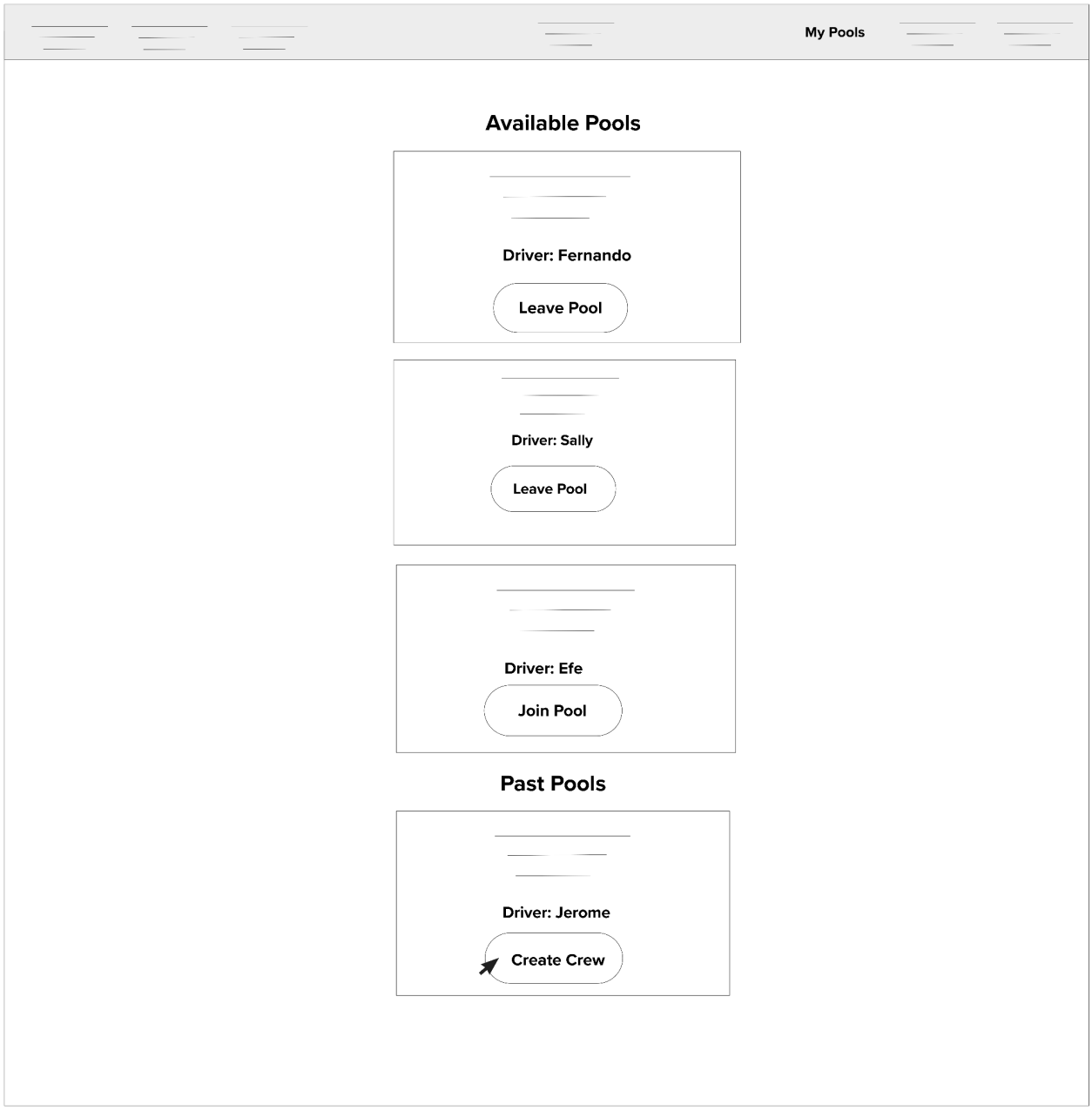
Post a pool: crew selected under privacy settings and filled (same screen as previous page)



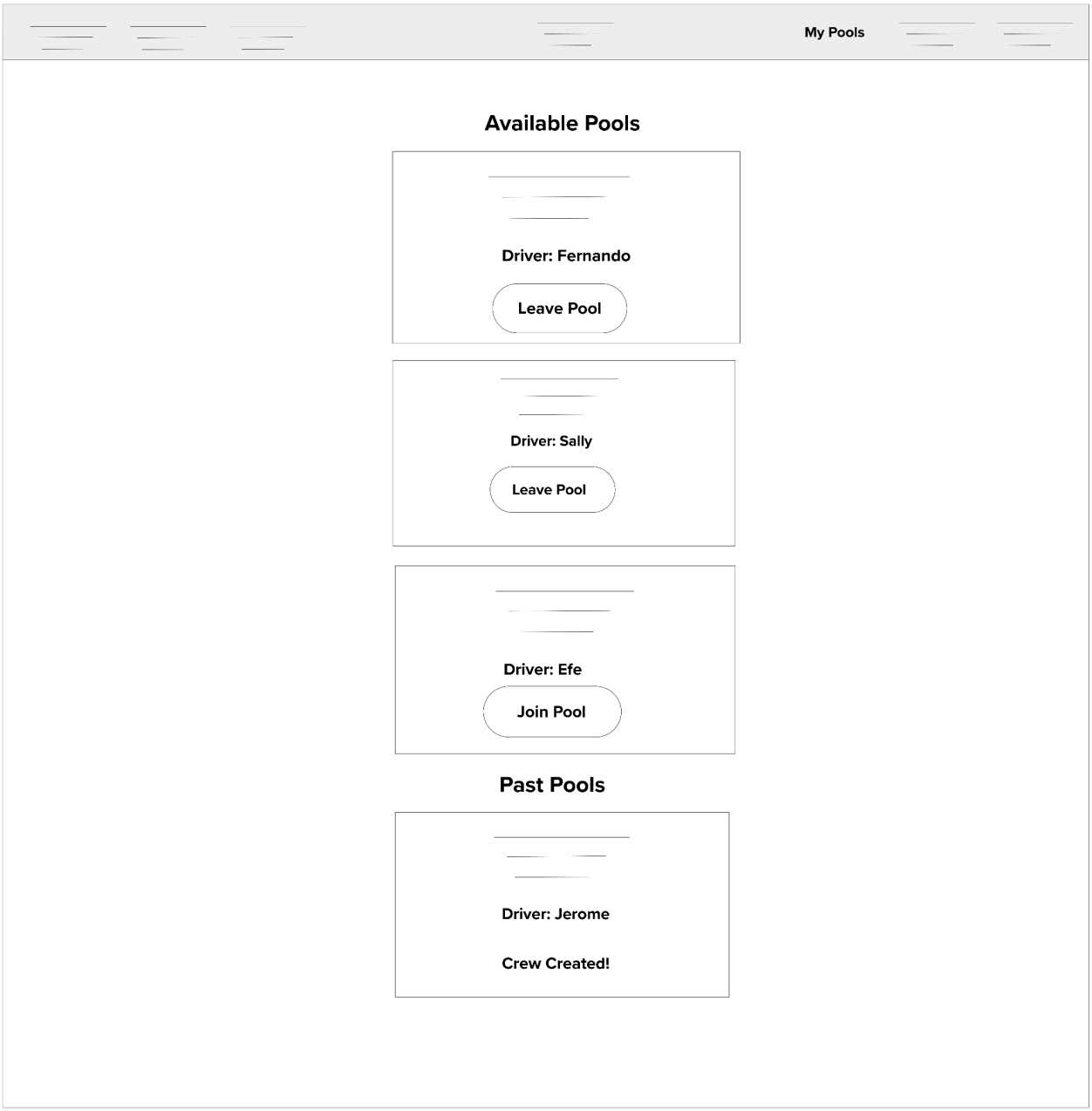
View pools as a driver: created pools display here, accessible to authenticated users from any page via header menu



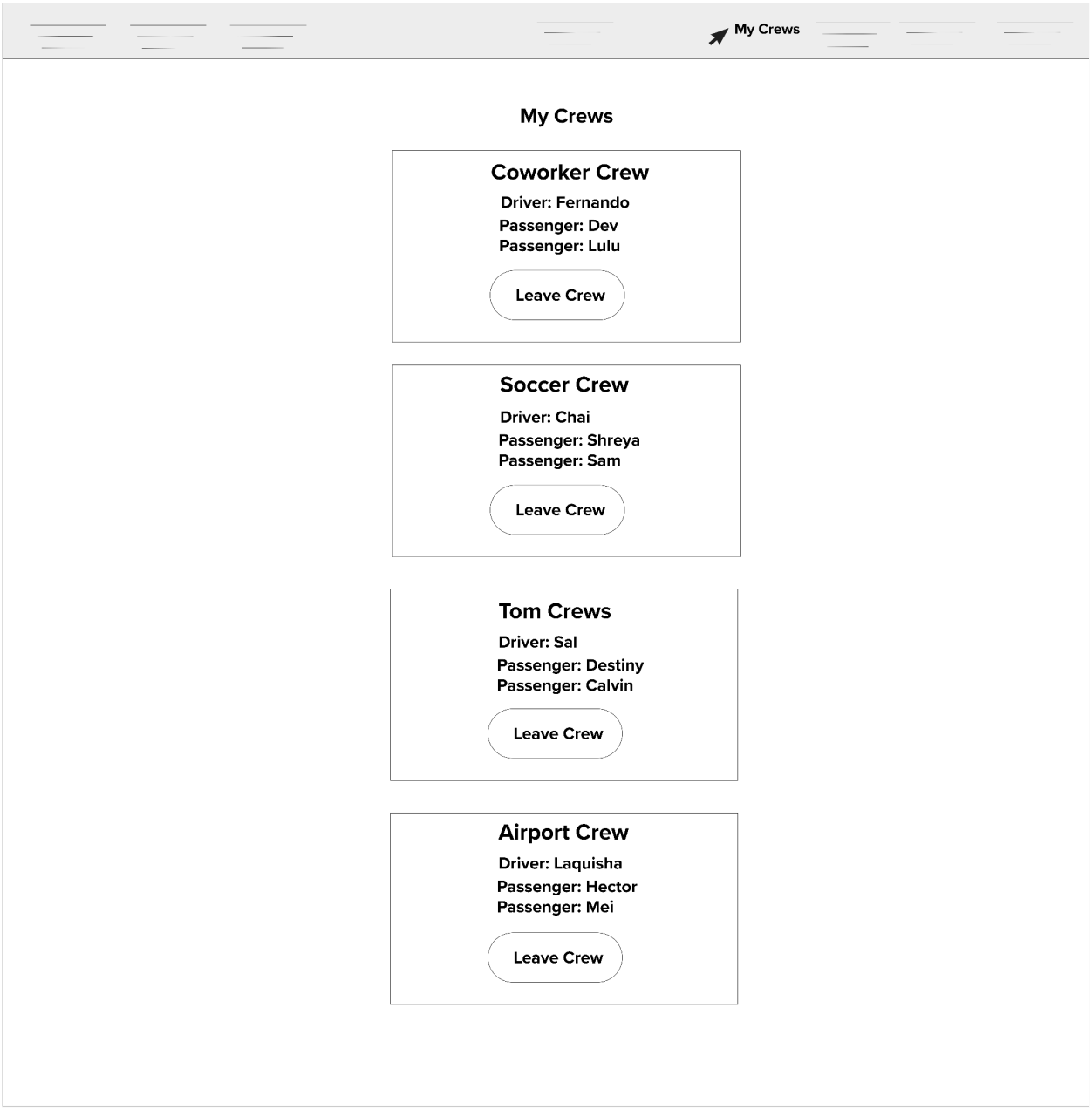
View pools as a passenger: create crew with members of a past pool (same screen as previous page)



View pools as passenger: crew created (same screen as previous page)

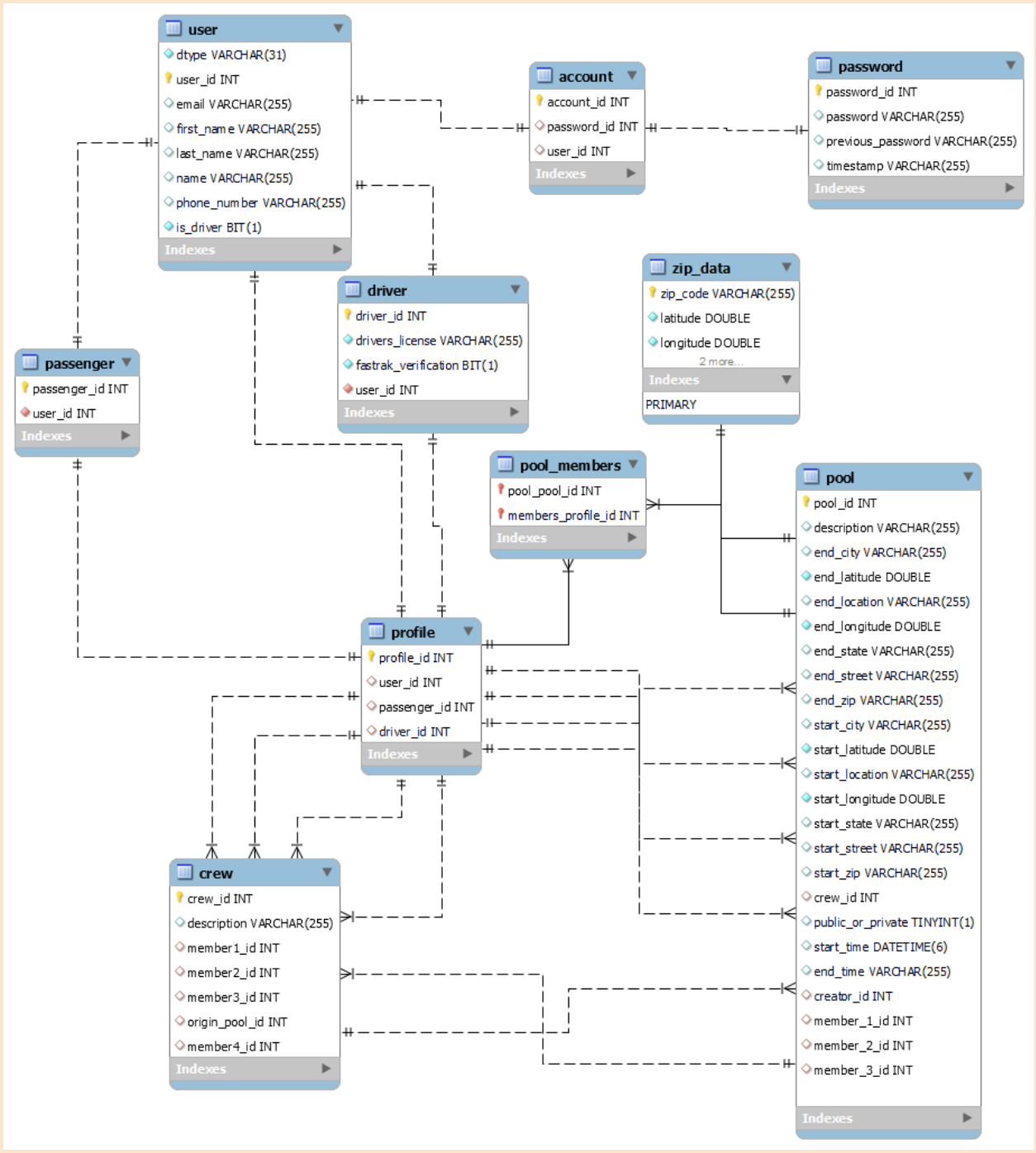


View crews: created crews are viewable here, this screen is accessible to authenticated users from any page via header menu

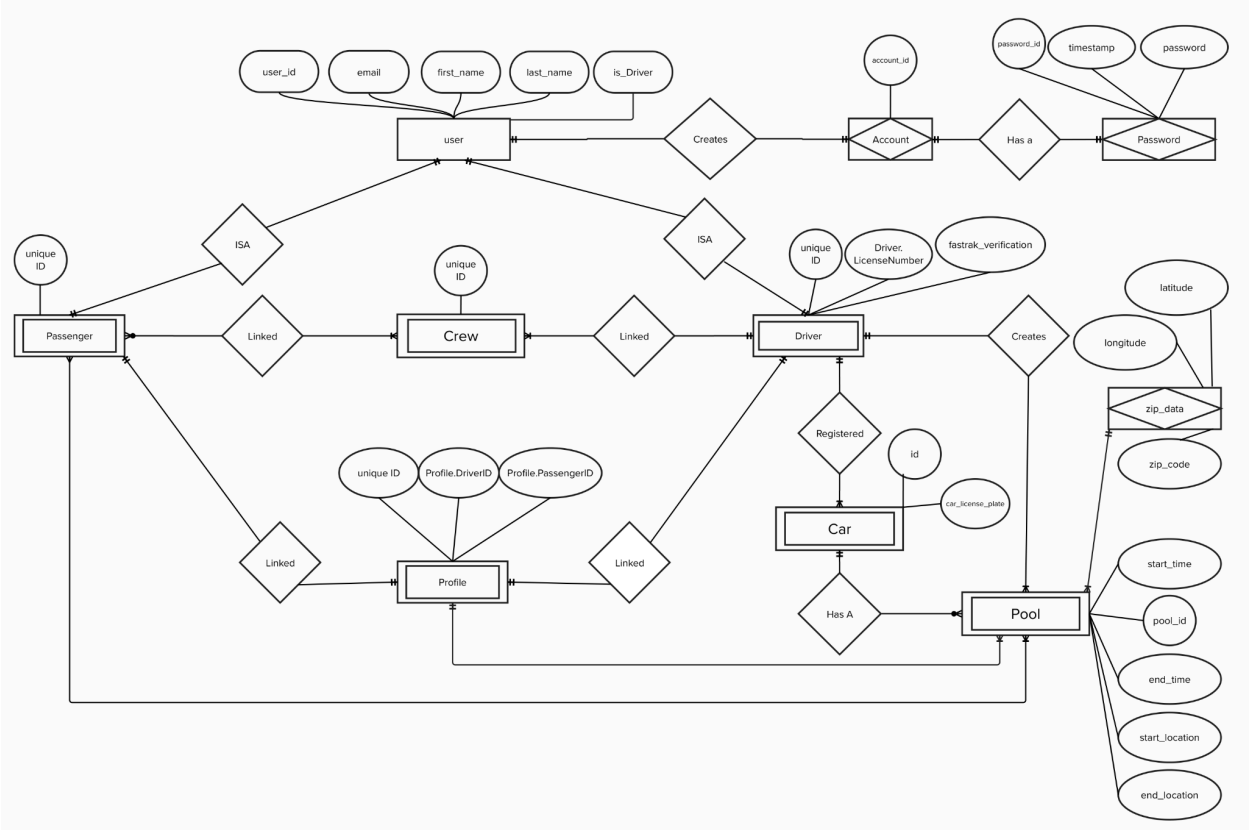


4. High level database architecture and organization

Entity Establishment Relationship Diagram (EER)

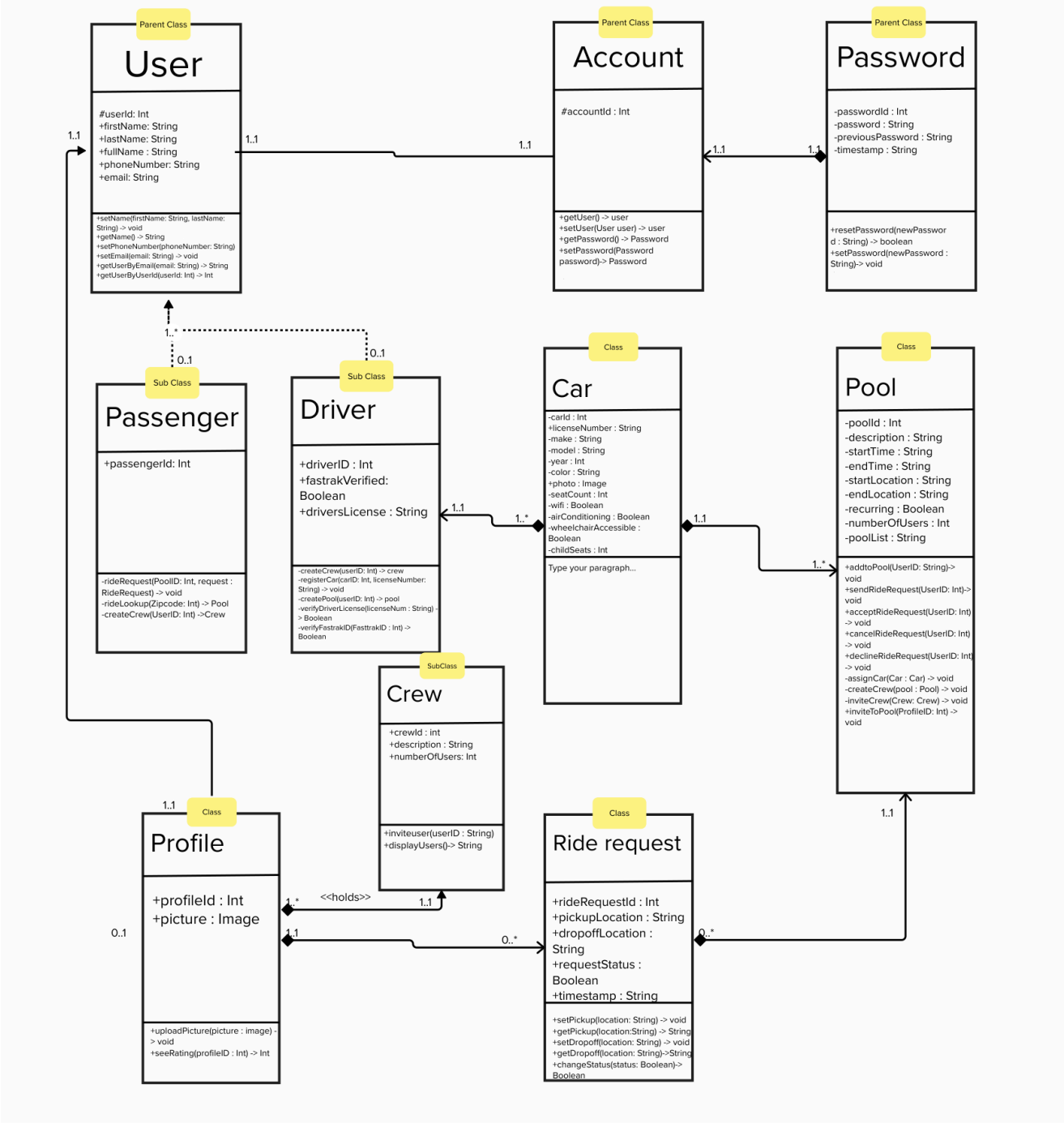


Entity Relationship Diagram (ERD)

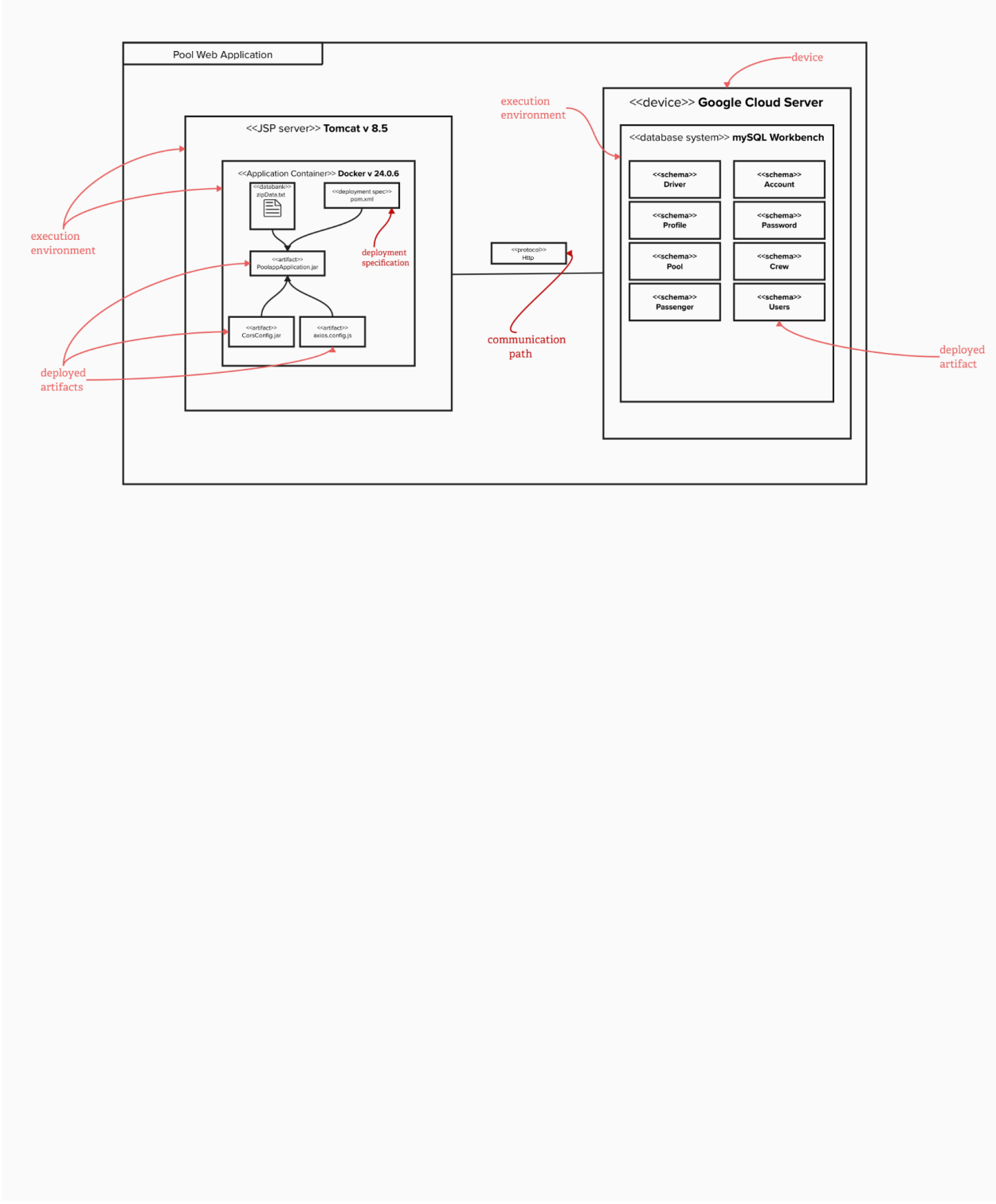


5. High Level Diagrams

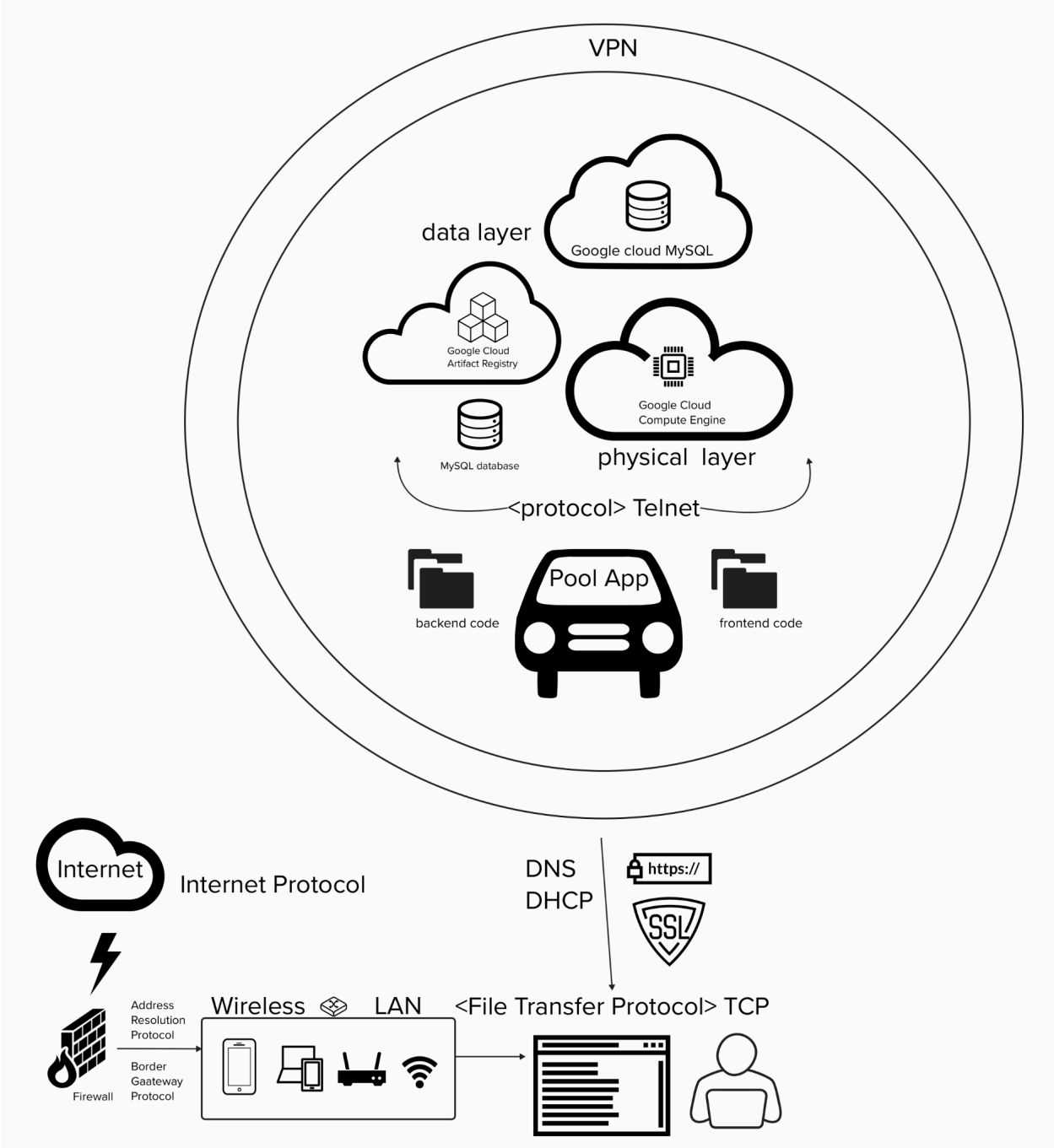
UML Diagram



Deployment Diagram



Network Diagram



6. List of Contributions

Team Contribution Scores

- Isiah: 10
- Jonathan: 10
- Kendrick: 8
- Kristian: 10
- Phillip: 10
- Xuan: 10

Prioritized Functional Requirements

- ✅Isiah: verbal contribution to full reprioritization during two collaborative sessions with full team
- ✅Jonathan: verbal contribution to full reprioritization during two collaborative sessions with full team
- ✅Kendrick: verbal contribution to full reprioritization during two collaborative sessions with full team
- ✅Kristian: verbal contribution to full reprioritization during two collaborative sessions with full team
- ✅Phillip: verbal contribution to full reprioritization during two collaborative sessions with full team
- ✅Xuan: verbal contribution to full reprioritization during two collaborative sessions with full team
- ✅Zoe: facilitated two collaborative sessions with full team

Wireframes

- ✅Isiah: Home page with pulldown open, Home page with search results
- ✅Jonathan: Home page, Home page with pulldown open
- ✅Kendrick: Signup
- ✅Phillip: Login
- ✅Zoe: Signup as driver, View passengerpProfile, View driver profile, Pool creation, Create crew, View pools, View crews (V1), all revisions to make wireframes more clearly connected (V2)

High Level Database Architecture and Organization

- ✅Isiah: Worked collaboratively deployment diagram on a team call using Mural
- ✅Jonathan: Worked collaboratively deployment diagram on a team call using Mural, created EER
- ✅Kendrick:Worked collaboratively deployment diagram on a team call using Mural

Frontend Implementation

- ✅Isiah:
  - View Profile
- ✅Kendrick
  - Add phone number to account creation screen
  - View Crews
- ✅Xuan
  - Driver registration
  - Pool creation
  - Join pool
  - Create a crew
  - View Pools
  - User can switch from logged in to logged out state
  - Show “Select” in search pulldown by default
  - Add search button to home page
  - Update default values in create pool input fields
  - Shift search bar up to make room for results

Backend Implementation

- ✅Jonathan:
  - Passenger becomes pool member on “join pool”
  - Create passenger class
  - Create passenger endpoint and implement method
  - Create profile endpoint and implement method
  - Update pool class to accommodate public vs private pool

    - Update data type of pool.start_time and delete pool.end_time
    - Insert user, pool, and crew data
    - Clean up profile table and class
    - Create endpoint to delete user from crew
    - Create endpoint to add user to crew
    - Update crew to include a fourth member
    - Update passenger to remove profileId
    - Clean up pool table and class
    - Update profile to remove user_type
- ✅Kristian:
    - Optimized proximity based search method for pool search
    - Supported frontend and backend teams, ensured both systems were working together throughout the development lifecycle
    - Update HTTP method returned in backend response
    - Create endpoint for “View pools”
    - Create pool endpoint
    - Update Sign-in endpoint to include profileId in response to frontend
- ✅Phillip:
    - Create driver class
    - Create crew class
    - Create profile class
    - Create driver endpoint and implement method
    - Create crew endpoint and implement method
    - Create passenger or driver + profile on user creation
    - Create endpoint to add user to pool
    - Create endpoint for pool creation

Deployment

- ✅Xuan: with input from Zoe purchased a domain with Google managed DNS and SSL, updated build process, fully owned deployment for delivery

Pool - carpooling app

SW Engineering CSC648-848-05 Fall 2023

Milestone 4 V2 | 12/07/2023

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

History Table

Milestone	Date
M4V2	**12/07/23**
M4V1	**11/30/23**
M3V2	**11/25/23**
M3V1	**11/02/23**
M2V2	**11/02/23**
M2V1	**10/12/23**
M1V2	**10/8/23**
M1V1	**9/21/23**

Table of Contents

1. Product Summary

Pool: carpoolwith.me

Pool is a carpooling website which provides a platform for drivers to post a “pool” and for passengers to join them.

Unlike other carpooling platforms, Pool helps drivers highlight their FasTrak account status. This ensures that prospective passengers know their driver will never have to avoid toll routes in their pool and can make use of time-saving fast lanes during peak traffic hours. Passengers looking for a guarantee that their pool is FasTrak equipped need only to check the driver’s FasTrak verification status on their profile.

What further sets Pool apart from its competitors is the unique ability for drivers and passengers who have “pooled” together in the past to form a crew. Drivers may choose to post a pool publicly – available for anyone to join – or privately, only available to members of their crew. This feature allows users to pool together with familiar faces, adding stability and predictability to their commute, wherever it takes them.

Prioritized Functional Requirements

Priority 1

1. User

1.1 A user shall create an account
1.2 A user shall register as a driver
1.3 A user shall register as a passenger
1.4 A user shall sign in
1.5 A user shall log out

2. Profile

2.1 A profile shall belong to a passenger
2.2 A profile shall belong to a driver
2.3 A profile shall contain user details
2.4 A profile shall be viewed by the user it is associated with

3. Driver

- 3.1 A driver shall be Fastrak verified
- 3.2 A driver shall create zero or many pools
- 3.3 A driver shall have zero or many crews
- 3.4 A driver shall have a driver's license
- 3.5 A driver shall view the pools they posted
- 3.6 A driver shall view the crews they are a member of
- 3.7 A driver shall leave the crews they are a member of

4. Passenger

- 4.1 A passenger shall join many pools
- 4.2 A passenger shall have many crews
- 4.3 A passenger shall view the pools they have joined
- 4.4 A passenger shall view the crews they are a member of
- 4.5 A passenger shall leave the crews they are a member of
- 4.6 A passenger shall leave the pools they are a member of

5. Crew

- 5.1 A crew shall be created by one user
- 5.2 A crew shall have a description
- 5.3 A crew shall have members

6. Pool

- 6.1 A pool shall have a description
- 6.2 A pool shall have a start time.
- 6.3 A pool shall have passengers.
- 6.4 A pool shall have one driver
- 6.5 A pool shall have a start location
- 6.6 A pool shall have an end location
- 6.7 A pool shall be deleted by its driver

2. Usability Test Plan

Superior Feature: Ride with your crew

Usability Test Results

Test use case	Environment	Task	Effectiveness (% Completed)	Errors	Comments	Efficiency (Time)	Recording (screen capture only)
5	Mobile web (Chrome), iOS	Create a crew	100%	0		1 min 56 seconds	Click to watch session
6	Mobile web (Chrome), iOS	Post a private pool to your crew	100%	3 inline errors hit	Defect encountered: submit button inaccessible at bottom of screen when all dropdowns are open	3 min 17 seconds	Click to watch session
12	Mobile web (Chrome), iOS	Join a pool posted privately to your crew	90%	1 blocking defect (pool posted to crew did not display under “Available pools”)	User got all the way to “My Pools”, which is where the pool should have shown, and is where the	59 seconds	Click to watch session

					final action would have been taken.		
12	Mobile web (Chrome), iOS	View your crews	100%	0	User vocalized that they remembered where to navigate to from the hamburger menu.	11 seconds	Click to watch session
11	Mobile web (Chrome), iOS	Leave a crew	100%	0	Same as above, user knew right to go. User got a glimpse of the task and almost left the crew during the previous test.	8 seconds	Click to watch session

Function 1: Create a crew

Objective

A “crew” is a group of familiar people who choose to pool (or carpool) together. After a user has ridden in a pool, they may create a crew composed of the members of that pool, including themselves. Crew creation is the first step Pool users can take to regularize their commute with familiar faces to shared destinations.

It is imperative that users can easily navigate to where they may create a crew – from a past pool. If a user can not accomplish this, or if it takes a user a long time to accomplish this, that would indicate that a better/different UX is required to assist in navigation and wayfinding.

Test Description

System Setup

This test requires logging into an existing Pool account using credentials belonging to a user with *past pools* in their account. The test monitor will assist in this process since it is not a part of the test. This test may be conducted on any device in any browser.

Starting Point

User will start on the logged in state of the home page: carpoolwith.me, and will be explained what a crew is. Then, user will be instructed to “create a crew with members of a past pool you rode with.”

Intended Users

Creating a crew is a feature which may be utilized by Pool users looking to regularize their commute with familiar faces commuting to a shared destination.

URL of the system or start screen to be tested

carpoolwith.me/my-pools

What is to be measured

This test measures:

- Whether or not the user is able to complete the task
- What percentage of the task the user is able to complete
- How long it takes the user to complete the task
- Whether or not creating a crew is a feature the user would utilize

Problem Statement & Task Descriptions

Problem Statement	User wants to pool with familiar faces.
Objectives	Ensure user can easily navigate the authenticated experience to create a crew.
User Profile	Carlos: Carpool curious, likes driving (current commuting method), middle aged, male identifying
Method	Qualitative: live observation with minimal intervention
Test Environment	In-person: user is in their home with live observation
Test Monitor Role	Test monitor sets up user on carpoolwith.me, provides test prompt, provides *minimal* assistance as needed, asks questions at end of session.
Evaluation Method	Observation of completion criteria checked against benchmark, Likert Questionnaire.

Task	**Description**
Task	Create a crew
Default State	Homepage, logged in
Successful completion criteria	New crew displays under “My crews”
Benchmark	Completed in under three minutes

Likert Questionnaire

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Creating a crew is easy.					✅
I like the look and feel of the user interface for creating a crew.			✅		
I would create a crew if I was a Pool user.					✅

Function 2: Post a private pool to your crew

Objective

Drivers may post a pool either publicly (so that it is returned in search results by all users and available for anyone to join) or privately (so that it is only surfaced to members of a crew the driver belongs to). In order for drivers to maintain control of who they share their vehicles with through the Pool platform, they must understand how to post a pool privately to a crew they are a member of, should they choose to.

Test Description

System Setup

This test will be conducted immediately after Test 1, so that the user is already logged into the account they may use for the test – and may utilize the crew they created in the previous test.

Starting Point

The test monitor will direct the user to return to the home page once Test 1 is complete, where they will begin Test 2 from.

Intended Users

Posting a pool privately to a crew is a feature which may be utilized by Pool users who have registered as a driver looking to maintain better control and predictability over their carpooling experience – particularly as it pertains to passengers of their shared vehicle.

URL of the system or start screen to be tested

carpoolwith.me/create-pool

What is to be measured

This test measures:

- Whether or not the user is able to complete the task

- What percentage of the task the user is able to complete
- How long it takes the user to complete the task
- Whether or not posting a private pool is a feature the user would utilize

Problem Statement & Task Descriptions

Problem Statement	User wants to share their vehicle with familiar faces.
Objectives	Ensure user can easily navigate the authenticated experience to post a pool privately to a crew they just created.
User Profile	Carlos: Carpool curious, likes driving (current commuting method), middle aged, male identifying
Method	Qualitative: live observation with minimal intervention
Test Environment	In-person: user is in their home with live observation
Test Monitor Role	Test monitor sets up user on carpoolwith.me, provides test prompt, provides *minimal* assistance as needed, asks questions at end of session.
Evaluation Method	Observation of completion criteria checked against benchmark, Likert Questionnaire.

Task	**Description**
Task	Post a private pool to your crew
Default State	Homepage, logged in
Successful completion criteria	Pool displays under “My pools” with Crew name that it was posted to
Benchmark	Completed in under five minutes

Likert Questionnaire

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Posting a private pool to my crew was easy.					✅
I like the look and feel of the user interface for posting a private pool to my crew.		✅			
I would post a private pool to my crew if I was a Pool user.					✅

Function 3: Join a pool posted privately to your crew

Objective

Members of a crew are in a specialized user group – they get access to private pools only they can join. In order for this exclusive feature to be utilized, it must be tested to ensure that crew members understand it is available to them, and where and how to access it.

Test Description

System Setup

This test will require logging out of the account used for Test 1 and Test 2 and logging into a new account. (This will help get the user into the mindset of transitioning from being a driver to a passenger.) This account must belong to a member of the crew the pool was posted privately to in the prior test. If required, the test moderator will assist with login since it is not a part of the test.

Starting Point

The test monitor will explain that the user has now logged into the account belong to a member of the crew they just posted a pool privately to. The monitor will then instruct the user to find and join that pool.

Intended Users

Any member of a crew may join pools posted privately to them. This feature targets recurring users with multiple carpooling destinations who have different crews they commute to different places with.

URL of the system or start screen to be tested

carpoolwith.me/my-pools

What is to be measured

This test measures:

- Whether or not the user is able to complete the task

- What percentage of the task the user is able to complete
- How long it takes the user to complete the task
- Whether or not joining a private pool is a feature the user would utilize

Problem Statement & Task Descriptions

Problem Statement	User wants an alternative commuting method that is consistent and familiar.
Objectives	Ensure user can easily navigate the authenticated experience to create a crew.
User Profile	Carla: Currently commutes on Bart and Muni, dislikes driving, student, female identifying
Method	Qualitative: live observation with minimal intervention
Test Environment	In-person: user is in their home with live observation
Test Monitor Role	Test monitor sets up user on carpoolwith.me, provides test prompt, provides *minimal* assistance as needed, asks questions at end of session.
Evaluation Method	Observation of completion criteria checked against benchmark, Likert Questionnaire.

Task	**Description**
Task	Join a pool posted privately to your crew
Default State	Homepage, logged in
Successful completion criteria	Joined pool now displays under “My pools”
Benchmark	Completed in under three minutes

Likert Questionnaire

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Joining a pool posted privately to my crew was easy.	✅				
I like the look and feel of the user interface for joining a pool posted privately to my crew.		✅			
I would join a pool posted privately to my crew if I was a Pool user.					✅

Function 4: View your crews

Objective

For Pool users that are members of many crews, crew management is essential. It is important for Pool users to feel in control of their commuting experience, and to understand the tools available for doing so. This test evaluates whether or not users understand where they may manage the crews they are a member of, and how to navigate to this area of the site.

Test Description

System Setup

This test will utilize the same account credentials as the previous test, so the user may remain logged in.

Starting Point

The test monitor will direct the user back to the home page, and ask them to navigate to where they may view all of the crews they are a member of.

Intended Users

This feature is particularly useful for recurring users belonging to multiple crews they commute to different places with.

URL of the system or start screen to be tested

carpoolwith.me/my-crews

What is to be measured

This test measures:

- Whether or not the user is able to complete the task
- What percentage of the task the user is able to complete
- How long it takes the user to complete the task
- Whether or not viewing crews is a feature the user would utilize

Problem Statement & Task Descriptions

Problem Statement	User wants to view information about the crews they are a member of.
Objectives	Ensure user can easily navigate the authenticated experience to view the crews they are a member of.
User Profile	Carla: Currently commutes on Bart and Muni, dislikes driving, student, female identifying
Method	Qualitative: live observation with minimal intervention
Test Environment	In-person: user is in their home with live observation
Test Monitor Role	Test monitor sets up user on carpoolwith.me, provides test prompt, provides *minimal* assistance as needed, asks questions at end of session.
Evaluation Method	Observation of completion criteria checked against benchmark, Likert Questionnaire.

Task	**Description**
Task	View your crews
Default State	Homepage, logged in
Successful completion criteria	Crew(s) display under “My crews”
Benchmark	Completed in under two minutes

Likert Questionnaire

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Viewing my crews was easy.					✅
I like the look and feel of the user interface for viewing my crews.		✅			
I would view my crews if I was a Pool user.					✅

Function 5: Leave a crew

Objective

A user's commute changing may require finding a new crew to join. Leaving a crew signals to other members that the user no longer shares their commute, and requires another crew to commute with. If the time does come to leave a crew, knowing how to do this within the UI is essential.

Test Description

System Setup

This test will utilize the same account credentials as the previous test, so the user may remain logged in.

Starting Point

The test monitor will direct the user back to the home page, and ask them to navigate back to where they may view all of the crews they are a member of. Once there, the user will be asked to leave a crew.

Intended Users

This feature is particularly useful for users with a changing commute who no longer require membership to a crew they are currently a member of.

URL of the system or start screen to be tested

carpoolwith.me/my-crews

What is to be measured

This test measures:

- Whether or not the user is able to complete the task
- What percentage of the task the user is able to complete
- How long it takes the user to complete the task
- Whether or not leaving a crew is a feature the user would utilize

Problem Statement & Task Descriptions

Problem Statement	User wants to leave a crew they are a member of.
Objectives	Ensure user can easily navigate the authenticated experience to view and leave a crew they are a member of.
User Profile	Carla: Currently commutes on Bart and Muni, dislikes driving, student, female identifying
Method	Qualitative: live observation with minimal intervention
Test Environment	In-person: user is in their home with live observation
Test Monitor Role	Test monitor sets up user on carpoolwith.me, provides test prompt, provides *minimal* assistance as needed, asks questions at end of session.
Evaluation Method	Observation of completion criteria checked against benchmark, Likert Questionnaire.

Task	**Description**
Task	Leave a crew
Default State	Homepage, logged in
Successful completion criteria	Crew which has been left no longer displays under “My crews”
Benchmark	Completed in under three minutes

Likert Questionnaire

	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Leaving a crew was easy.					✅
I like the look and feel of the user interface for leaving a crew.				✅(with feedback that this part of the UI was better due to the responsive color change of the “leave crew” button and the change state)	
I would leave a crew if I was a Pool user.			✅(with feedback “why would I leave a crew?... I would use it I guess if I needed it.”		

3. QA Test Plan

Nonfunctional Requirement 2.2: Pool shall use user input validation to ensure the integrity of data stored and retrieved.

Objective

This QA test puts a special focus on frontend validations to ensure that users are:

- Hitting a validation error when an invalid input is entered
- Seeing the validation in the right place
- Being presented with the correct inline error message

Backend validations are the final safeguard against bad or malicious data, but it should not function as the only protection against this – which is why it's important that the app's first line of protection, the frontend, is effectively catching bad data and preventing it from being sent in request bodies to the backend.

Setup

URLs of the system (or start screen) to be tested

carpoolwith.me/signup

carpoolwith.me/signin

carpoolwith.me/create-pool

System Setup

Test 1.

Navigate to carpoolwith.me/signup.

Test 2.

Navigate to carpoolwith.me/signin.

Test 3.

Navigate to: carpoolwith.me/signin and enter the following credentials:

- Email: JJonthetrack@gmail.com
- Password: manapool

Click on “Post a Pool” in the header menu (or from the hamburger menu, on mobile).

Test Environment

This will be tested live in production on carpoolwith.me on desktop web in all supported browsers.

Feature to be tested

This tests the inline error messaging on required fields of the “Sign Up”, “Sign In”, and “Post a Pool” screens.

Test Number	Setup	Description	Test Input	Expected Output	P/F: Chrome 🖥️	P/F: Chrome 📱	P/F: Safari 🖥️	P/F: Safari 📱	P/F: Firefox 🖥️	P/F: Firefox 📱	P/F: DuckDuckGo 🖥️	P/F: DuckDuckGo 📱
1	Navigate to carpoolwith.me/signup	Required validation should trigger when fields are left blank on Sign Up	Click “Sign Up” without entering any inputs.	“Required” inline error messaging shows beneath every input other than	P	P	P	P	P	P	P	P

				the driver registration checkbox.								
2	Navigate to carpoolwith.me/signin	Required validation should trigger when fields are left blank on Sign In	Click “Sign In” without entering any inputs.	“Required” inline error messaging shows beneath every input.	P	P	P	P	P	P	P	P
3	Navigate to carpoolwith.me, sign in with the credentials: JJonthetrack@gmail.com (password: manapool), and	Required validation should trigger when fields are left blank on Post a Pool	Click “Submit” without entering any inputs.	“Required” error messaging shows alongside every missing input.	P	P	P	P	P	P	P	P

	click “Post a Pool” in the header menu											

Nonfunctional Requirement 17.1: Application will have an average response time of 2 seconds or less.

Objective

The objective of testing this nonfunctional requirement is to ensure that users may take actions on Pool in an expectantly quick fashion, without encountering slow load times. To achieve this, this series of QA tests focuses on different user driven actions critical to the key functionality of Pool, i.e. sign up, log in, crew creation, etc. The tests put a special focus on time between triggering an action on the frontend and the end result, or output. It is important for users to have an experience on Pool which matches their expectations for highly functioning web services, and fast load time is a big part of this evaluation.

Setup

URLs of the system (or start screen) to be tested

carpoolwith.me

System Setup

Navigate to carpoolwith.me.

For Test 3, the following credentials will be required:

Email: JJonthetrack@gmail.com

Password: manapool

Test Environment

This will be tested live in production on carpoolwith.me on desktop web in all supported browsers.

Feature to be tested

This tests the application response time to ensure that the system can handle requests in two seconds or less.

Test Number	Setup	Description	Test Input	Expected Output	P/F: Chrome 🖥️	P/F: Chrome 📱	P/F: Safari 🖥️	P/F: Safari 📱	P/F: Firefox 🖥️	P/F: Firefox 📱	P/F: DuckDuckGo 🖥️	P/F: DuckDuckGo 📱
1	Go to: https://carpoolwith.me/	Test that the search pools by starting zipcode returns results within 2 seconds using a stopwatch application.	On the Pool homepage, select the starting zip code in the dropdown, enter a zip code and press enter or search.	Display pools near the starting zip code within 2 seconds of searching.	P	P	P	P	P	P	P	P
2	Go to: https://carpool	Test to see if the response to	In the homepage,	The user is redirected to the	P	P	P	P	P	P	P	P

	with.me/	redirect users to the signup page from the “login” takes 2 seconds or less.	first click on “Log in” in the top right, once there, click on “Don't have an account? Sign Up”.	signup page within 2 seconds.								
3	Go to: https://carpoolwith.me/	Test to see if the response for signing in is 2 seconds or less	In the homepage, select the “login” option on the top right corner and enter email and password	Once the user enters their information, and clicks on “sign in” button, it will take 2 seconds or less to get redirected to the home page	P	P	P	P	P	P	P	P

			(JJonthetrack@gmail.com, manapool) before clicking to sign in									

Nonfunctional Requirement 19.2: Data shall be sent over https protocol.

Objective

The objective of testing this nonfunctional requirement is to ensure that https protocol is always being used for the transmission of data. This is important for data security and to protect user assets. This also verifies that carpoolwith.me has an active SSL certificate recognized at the browser and network level.

Setup

URLs of the system (or start screen) to be tested

carpoolwith.me

System Setup

Navigate to carpoolwith.me on desktop web in the browser being tested.

Test Environment

This will be tested live in production on carpoolwith.me on desktop web in all supported browsers.

Feature to be tested

This test confirms that all data and requests are being transmitted using https protocol.

Test Number	Setup	Description	Test Input	Expected Output	P/F: Chrome 🖥️	P/F: Chrome 📱	P/F: Safari 🖥️	P/F: Safari 📱	P/F: Firefox 🖥️	P/F: Firefox 📱	P/F: DuckDuckGo 🖥️	P/F: DuckDuckGo 📱
1	Go to https://www.carpoolwith.me	When you get to the homepage click on the padlock and view the valid ssl certificate	Click the padlock and view ssl certificate.	A valid SSL certificate.	P	P	P	P	P	P	P	P
2	Go to: https://carpoolwith.me/	When on the homepage, click the link to check for the “https” that appears before the url to see that data is being sent over https.	Go to the site, click on the link, might have to click it twice, and it should reveal if the	“Https” should appear before the url when the url is clicked once or twice.	P	P	P	P	P	P	P	P

			site has https.									
3	Go to: https://carpoolwith.me/	When on the homepage, right click on the page and select the security tab and then should validate that it is secure	On the link, right click the page the view the page source	Click on the security tab and it should validate that the page is secured	P	P	P	P	P	P	P	P

Nonfunctional Requirement 6.1: Web Application will have search functionality to assist users.

Objective

This test seeks to confirm that search functionality is currently operational and behaving as expected. Search is a core component of the site, and this feature not working would be considered a critical bug preventing users from utilizing its main service (for both drivers and passengers).

Setup

URLs of the system (or start screen) to be tested

carpoolwith.me

System Setup

Navigate to carpoolwith.me on desktop web in the browser being tested.

Test Environment

This will be tested live in production on carpoolwith.me on desktop web in all supported browsers.

Feature to be tested

This test confirms that searching for pools using the following filters works as expected:

- Search for pools by start zip code
- Search for pools by end zip code
- Search for pools by city

Test Number	Setup	Description	Test Input	Expected Output	P/F: Chrome 🖥️	P/F: Chrome 📱	P/F: Safari 🖥️	P/F: Safari 📱	P/F: Firefox 🖥️	P/F: Firefox 📱	P/F: DuckDuckGo 🖥️	P/F: DuckDuckGo 📱
1	Navigate to carpoolwith.me	Tests functionality of searching for pools by start zip code.	Select “Starting Zipcode” from the pulldown and enter “94133” in the search bar. Click “Search”.	Pools are returned	P	P	P	P	P	P	P	P
2	Navigate to carpoolwith.me	Tests functionality of searching for pools by end zip code.	Select “Ending Zipcode” from the pulldown and enter “94103” in the	Pools are returned	P	P	P	P	P	P	P	P

			search bar. Click “Search”.									
3	Navigate to carpoolwith.me	Tests functionality of searching for pools by city.	Select “City” from the pulldown and enter “San Francisco” in the search bar. Click “Search”.	Pools are returned	P	P	P	P	P	P	P	P

Nonfunctional Requirement 8.1: Passwords shall be encrypted.

Objective

This test seeks to confirm that password encryption is functioning as expected to ensure the security of passwords stored at the database level. Specifically, this set of tests manually replicates the BCrypt hashing algorithm to hash a password and ensure that the end result is the same as the one the system produces with the same password input (with exception to the salt, which is randomly generated for every unique password generated – confirming this random generation also functions is a core part of this test suite).

Setup

URLs of the system (or start screen) to be tested

carpoolwith.me/signup

System Setup

Navigate to carpoolwith.me on desktop web in the browser being tested.

Test Environment

This will be tested live in production on carpoolwith.me on desktop web in all supported browsers.

Feature to be tested

This test confirms that the following aspects of password encryption work as expected:

- Password hashing
- Secure password storage
- Random salt generation

Test #	Setup	Description	Test Input:	Expected Output	P/F:
1	Go to: https://www.carpoolwith.me/signup	When a user signs up as a Pool user, confirm that a hash is stored in the password table in the database and not the characters of the password.	Password entered: manapool	Password Hash: $2a$10$3QhEsBRh3f8LupWS0HqQkOclSy/DYJTgx6YKMoTLsPybqRDlwJEiq	Pass
2	Go to: https://www.carpoolwith.me/signup	Confirm that a user can sign in with their text password: so the password can be matched to the hash for user sign in.	Password entered: manapool	User is able to sign in with the correct password and is unable to sign in with the incorrect password.	Pass
3	Go to: https://www.carpoolwith.me/signup	Create two new accounts with the same password. Each password hash stored in the database should be different ,although the same password has been used, because we are using a different salt for the hash each time a password is encrypted.	Password entered: abc123	Hash #1: $2a$10$1LWorToGCqqmjGWQZlpuYuD7.BhB9XMdKt0Tj2GTVLaFL6plB.Lpy Hash #2: $2a$10$ZCosaPzS7n4M/lsbcry8Xey9NEGW9tMyE8A.lqDkASggNlJ9Q1lzm	Pass

4. Code Review

Coding Style

We pair program very regularly as a team, it is how we share knowledge across different corners of our application. Our code base leverages a microservices architecture, with a separate backend and frontend app. The coding style is defined by the following patterns within each app and framework:

Backend Code

- Java / Spring
    - Camel case
    - Indent using spaces
    - Tab size = four spaces

Frontend Code

- React.js
    - Camel case
    - Indent using spaces
    - Tab size = two spaces

Database

- MySQL (JPA)

Choose the Code for Review

The crew controller was chosen for review because it plays an essential role in the crew feature, which is the focus of our usability test. From crew creation, posting a pool privately to a crew, to leaving a crew, the crew controller makes it all possible. The crew controller can be thought of as the glue between the database, backend, and frontend systems related to the crew entity and function.

[Preview] README.md | CrewController.java

application > poolapp > src > main > java > com > fantasticfour > poolapp > controller > CrewController.java > CrewController

```java
/**
 * CrewController: is part of the pool REST api that handles logic for CREWS.
 *  FYI: A crew is a group of people that can schedule rides with one another.
 */
package com.fantasticfour.poolapp.controller;

import com.fantasticfour.poolapp.CustomResponse.CrewListResponse;
import com.fantasticfour.poolapp.CustomResponse.CrewResponse;
import com.fantasticfour.poolapp.domain.Crew;
import com.fantasticfour.poolapp.domain.Pool;
import com.fantasticfour.poolapp.domain.Profile;
import com.fantasticfour.poolapp.repository.PoolRepository;
import com.fantasticfour.poolapp.repository.ProfileRepository;
import com.fantasticfour.poolapp.services.CrewService;
import com.fantasticfour.poolapp.repository.CrewRepository;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.jpa.repository.support.SimpleJpaRepository;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Optional;
import java.util.stream.Collectors;


@RestController
@RequestMapping("/api/crew")
public class CrewController {

```

```
34
35      @Autowired
36      private CrewService crewService;
37
38      @Autowired
39      private CrewRepository crewRepository;
40
41      @Autowired
42      private ProfileRepository profileRepository;
43
44      @Autowired
45      private PoolRepository poolRepository;
46
47      /**
48       * Create a crew entity.
49       * @param crew from json request body.
50       * @return ResponseEntity<Crew>
51       */
52      @PostMapping({"", "/"})
53      public ResponseEntity<Crew> addCrew(@RequestBody Crew crew) {
54          Crew newCrew = crewService.addCrew(crew);
55          return new ResponseEntity<>(newCrew, HttpStatus.CREATED);
56      }
57
58      /**
59       * Retrieves crews certain profiles are a member of.
60       * @param profileId
61       * @return a list of crew entities the user is a member of.
62       */
63      @GetMapping("/{id}")
64      public ResponseEntity<List<CrewResponse>> getCrewById(@PathVariable("id") int profileId) {
65
66          CrewListResponse crewListResponse = new CrewListResponse();
67          List<CrewResponse> crewResponselist = new ArrayList<>();
68
```

```
69          //get crews by profile id
70          List<Crew> crews = crewService.getCrewByProfileId(profileId).stream()
71                                        .filter(Optional::isPresent)
72                                        .map(Optional::get)
73                                        .collect(Collectors.toList());
74
75          //build a custom http response body
76          if(!crews.isEmpty()){
77              for (Crew crew: crews
78                   ) {
79                  CrewResponse crewResponse = new CrewResponse();
80                  crewResponse.setCrewId(crew.getCrewId());
81                  crewResponse.setDescription(crew.getDescription());
82                  if ((crew.getMember1() != null)) {
83                      crewResponse.setOneMember(crew.getMember1());
84                  } else {
85                      System.out.println("User does not Exist");
86                  }
87                  if ((crew.getMember2() != null)) {
88                      crewResponse.setOneMember(crew.getMember2());
89                  } else {
90                      System.out.println("User does not Exist");
91                  }
92                  if ((crew.getMember3() != null)) {
93                      crewResponse.setOneMember(crew.getMember3());
94                  } else {
95                      System.out.println("User does not Exist");
96                  }
97                  if((crew.getCreator() != null)) {
98                      crewResponse.setOneMember(crew.getCreator());
99                  } else {
100                     System.out.println("User does not Exist");
101                 }
102                 crewResponselist.add(crewResponse);
103             }
```

```
104             return new ResponseEntity<>(crewResponselist,HttpStatus.OK);
105         }
106         else{
107             return new ResponseEntity<>(null,HttpStatus.OK);
108         }
109     }
110 
111     /**
112      * Get all crews in the database
113      * @return A list of all the crews in the DB.
114      */
115     @GetMapping({"", "/"})
116     public ResponseEntity<List<Crew>> getAllCrews() {
117         List<Crew> crews = crewService.getAllCrews();
118         if (crews.isEmpty()) {
119             return new ResponseEntity<>(null, HttpStatus.OK);
120         }
121         return new ResponseEntity<>(crews, HttpStatus.OK);
122     }
123 
124     /**
125      * Updates members of an existing crew.
126      * @param id (crew id)
127      * @param crew
128      * @return A json response body of the updated crew.
129      */
130     @PutMapping("/{id}")
131     public ResponseEntity<Crew> updateCrew(@PathVariable("id") int id, @RequestBody Crew crew) {
132         Optional<Crew> currentCrew = crewService.getCrewById(id);
133         if (currentCrew.isPresent()) {
134             Crew updatedCrew = crewService.updateCrew(crew);
135             return new ResponseEntity<>(updatedCrew, HttpStatus.OK);
136         } else {
137             return new ResponseEntity<>(HttpStatus.NOT_FOUND);
138         }
```

```
139     }
140
141
142     /**
143      * Removes a single member of a crew, if the member of the crew removed is also the
144      * creator then the entire crew entity is deleted.
145      * @param jsonMap : json request body with profileID and crewID.
146      * @return String in json response body whether removal is successful or not.
147      */
148     @DeleteMapping("/remove/member")
149     public ResponseEntity<?> deleteUser(@RequestBody Map<String, Object> jsonMap) {
150         jsonMap.forEach((key, value) -> System.out.println("Key: " + key + ", Value: " + value));
151         System.out.println("we are in the remove member function");
152         if(jsonMap.isEmpty()){
153             return new ResponseEntity<>(HttpStatus.NO_CONTENT);
154         }
155
156         int profileId = (int)jsonMap.get("profileId");
157         int crewId = (int)jsonMap.get("crew_id");
158
159
160         Optional<Crew> optionalCrew = crewRepository.findById(crewId);
161
162         if(!optionalCrew.isPresent()){
163             return new ResponseEntity<>("Crew not found", HttpStatus.NOT_FOUND);
164         }
165
166         Crew crew = optionalCrew.get();
167
168         //if profileId called is the creator, delete the entire crew
169         //if not, remove profile from crew
170
171         boolean isDeleted = false;
172         if(crew.getCreator() != null && profileId == crew.getCreator().getProfileId()){
173             deleteCrew(crew.getCrewId());
```

```
            //set pool created to 0 after creator is deleted. because crew is deleted when creator deletes a crew.
            Optional<Pool> optionalPool = poolRepository.findById(crew.getOriginPoolId());
            if (optionalPool.isPresent()) {
                Pool originalPool = optionalPool.get();
                originalPool.setCrewCreated(false);
                poolRepository.save(originalPool);
            }else {
                System.out.println("origin pool id does not exist to toggle crew_created field");
            }

            return new ResponseEntity<>("Crew deleted", HttpStatus.OK);
        }

        /**
         * Deleting crew members based on profile ID
         */
        if(crew.getMember1() != null && crew.getMember1().getProfileId() == profileId){
            crew.setMember1(null);
            isDeleted = true;
        }
        if(crew.getMember2() != null && crew.getMember2().getProfileId() == profileId){
            crew.setMember2(null);
            isDeleted = true;
        }
        if(crew.getMember3() != null && crew.getMember3().getProfileId() == profileId){
            crew.setMember3(null);
            isDeleted = true;
        }
        if(crew.getCreator() != null && crew.getCreator().getProfileId() == profileId){
            crew.setCreator(null);
            isDeleted = true;
        }

        if(isDeleted){
            crewRepository.save(crew);
```

```
213             return new ResponseEntity<>("Profile with id" + profileId + "not found", HttpStatus.NOT_FOUND);
214         }
215     }
216
217     /**
218      * Delete a crew entity.
219      * @param id (Crew ID)
220      * @return no content returned on deletion.
221      */
222     @DeleteMapping("/{id:[\\d]+}") //"/{id}" regex to make pathing more specific, only accepts integers.
223     public ResponseEntity<Void> deleteCrew(@PathVariable("id") int id) {
224         crewService.deleteCrew(id);
225         return new ResponseEntity<>(HttpStatus.NO_CONTENT);
226     }
227
228     /**
229      * Create a crew from from a previous pool id(carpoool id). Member of the pool will be the
230      * new members of the crew.
231      * @param jsonMap keys including the pool_id and creator of the crew.
232      * @return a json return body confirming if new crew is created or not.
233      */
234     @PostMapping("/createcrew")
235     public ResponseEntity<?> createCrew(@RequestBody Map<String,Object> jsonMap){
236         jsonMap.forEach((key, value) -> System.out.println("Key: " + key + ", Value: " + value));
237
238         // Checking is request body is empty
239         if(jsonMap.isEmpty()){
240             return new ResponseEntity<>("Nothing in Json Body", HttpStatus.NO_CONTENT);
241         }
242         boolean profileExists = false;
243         Profile creator_id;
244         Profile member1_id;
245         Profile member2_id;
246         Profile member3_id;
247         int originPoolId;
```

```
249             Crew crew = new Crew();
250             Pool pool = new Pool();
251 
252             // Create crew from original pool
253             if(jsonMap.get("origin_pool_id") != null){
254                 originPoolId = (int)jsonMap.get("origin_pool_id");
255                 Optional<Pool> optionalPool = poolRepository.findPoolByPoolId(originPoolId);
256                 if(optionalPool.isPresent()){
257                     pool = optionalPool.get();
258                     pool.setCrewCreated(true);
259                     poolRepository.save(pool);
260                     crew.setOriginPoolId((int)jsonMap.get("origin_pool_id"));
261                 }
262             }
263 
264             //checks for existing profileId
265             if(jsonMap.get("creator_id") != null){
266                 int creatorid = (int)jsonMap.get("creator_id");
267                 Optional<Profile> creator = profileRepository.findProfileByProfileId(creatorid);
268                 if(creator.isPresent()){
269                     creator_id = creator.get();
270                     crew.setCreator(creator_id);
271                     profileExists = true;
272                 }
273             }
274 
275             // Setting members in a crew
276             if(pool.getMember1() != null){
277                 member1_id = pool.getMember1();
278                 crew.setMember1(member1_id);
279                 profileExists = true;
280             }
281             if(pool.getMember2() != null){
282                 member2_id = pool.getMember1();
283                 crew.setMember2(member2_id);
```

```
284                 profileExists = true;
285             }
286             if(pool.getMember3() != null){
287                 member3_id = pool.getMember1();
288                 crew.setMember3(member3_id);
289                 profileExists = true;
290             }
291             crew.setDescription(pool.getDescription());
292 
293 
294             if(profileExists){
295                 crewRepository.save(crew);
296                 return new ResponseEntity<>("Crew created", HttpStatus.OK);
297             }
298 
299             else{
300                 return new ResponseEntity<>("Profile(s) do not exist, crew cannot be created", HttpStatus.NOT_FOUND);
301             }
302 
303     }
304 }
```

Internal Code Review

- Removing unused imports, such as those on lines 16, 17, and 19, would be good practice.
- The controller correctly uses field injection with @Autowired annotations (lines 36-45), but we should consider using constructor injection for better testability to follow Spring's framework practices.
- addCrew (lines 48-56) properly uses an HTTP status of 'CREATED' upon successfully adding a new crew (line 55).
- getCrewById (lines 58-109) is well-structured, but it should be able to handle potential 'null' values better. Instead of printing to the console (lines 85, 90, 95, 100), we could use a logger and return meaningful response to the client.
- getAllCrews (lines 115-121) correctly handles cases where no crews are found by returning an empty list rather than a 'null.'
- updateCrew (lines 130-139) properly checks if the crew exists before attempting an update, if the crew is not found, it correctly responds with an HTTP status of 'NOT_FOUND.'
- deleteUser (lines 142-214) provides logic for removing a crew member or an entire crew, depending on the request. However, lines 152 and 183 show that there are direct console outputs that are used for debugging purposes, which should be replaced with a proper logger to return actual response to the client.
- deleteCrew (lines 222-226) uses the correct HTTP status response of 'NO_CONTENT' when a deletion is successful (line 225).
- createCrew (lines 234-303) checks if the request body is empty and returns early if it is (lines 238-241), this is good in practice.
- On lines 282 and 287, there seems to be a bug where member2_id and member3_id are set to pool.getMember1() instead of pool.getMember2() and pool.getMember3() respectively.
- On line 296, consider using HTTP status 'CREATED' instead of 'OK' to state that a new resource is being created.

External Code Review *Conducted by Jeremy Tran's team*

"CrewController" class looks to be well-organized and does well at setting up endpoints for managing crews. I thought that all the methods are clear and well-commented. Overall, it was very easy to follow along and understand what was going on. There are a few areas that I think could be improved on.

- The error handling I think could be better such as on updateCrew where it assume the requested resource exists. It should handle cases where the resource isn't there.
- Response from the server has a very general type 'ResponseEntity<?>'. I think it could be more clear what kind of data type you're handling.
- There's repeated code in createCrew, maybe make it into a function so it's not duplicated.
- API paths are mostly consistent but ones like '/remove/member' moves away from your pattern.
- There are use of 'System.out.println' which shouldn't be there but I assume these will be removed during final submission.

5. Self Check on Best Practices for Security

Major Protected Assets

Pool secures major protected assets such as user information including but not limited to emails, phone numbers and email addresses using several methods:

- HTTPS (SSL/TLS) provided by “Let's Encrypt” is used to encrypt data transmitted by the client and server.
- Spring Security Module in conjunction with JSON Web Tokens (JWT) is used to secure Pool application API’s from being accessed without proper authentication.
- User input is validated on the front-end and back-end to avoid injection attacks.
- Pool does not log sensitive user data in application error logs or return user information in error messages.

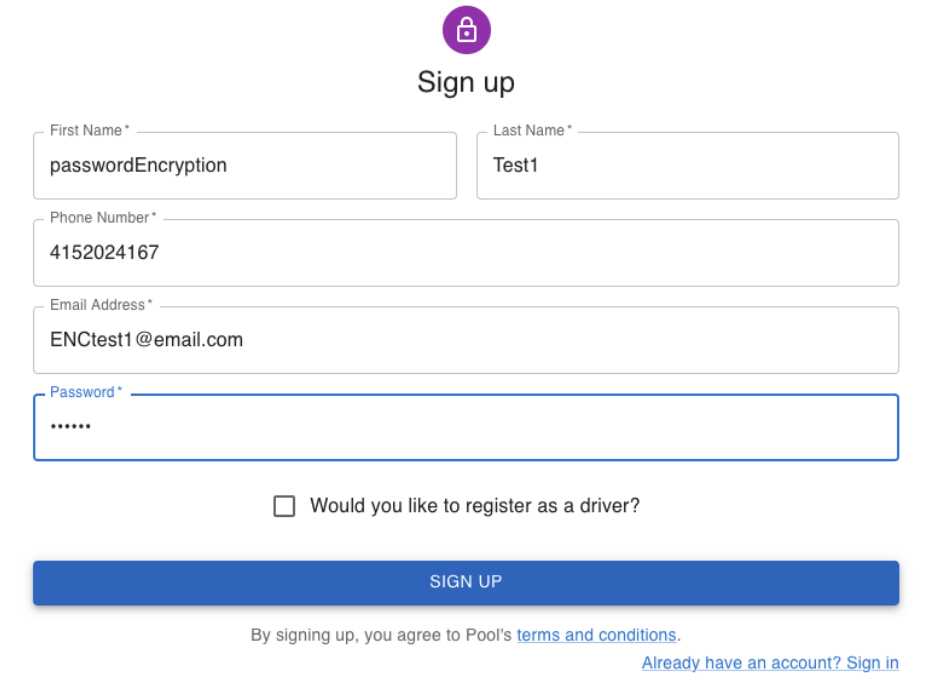
Password Encryption

The Pool application uses the BCrypt hashing algorithm provided by the Spring Security Cryptography Module to secure user passwords by hashing plain text passwords and only storing the resulting hash in the password table in the database. BCrypt generates a random salt each time a password is hashed so even two identical passwords should have different hashes stored in the database.

Example of user signing up and the resulting database entry:

Password: abc123

See next page for screenshots:




```
10  SELECT
11      u.user_id, u.email,
12      p.password
13  FROM
14      account a
15      JOIN user u ON a.user_id = u.user_id
16      JOIN password p ON a.password_id = p.password_id;
```

0% 25:11

Result Grid Filter Rows: Search Export:

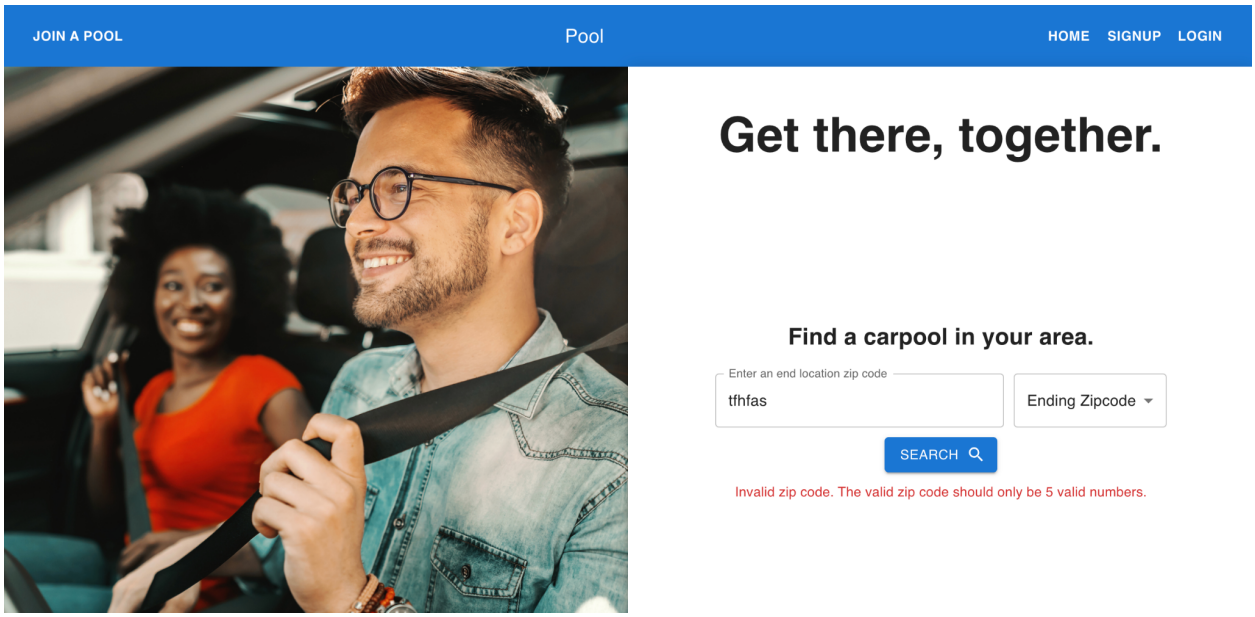
user_id	email	password
718	duyanh120115tm@gmail.com	$2a$10$2HiisGvhVFB8/oU8mgyXz.RfZksqbJXFeig9L/fKA40ELcRzojKpe
719	lyla@yopmail.com	$2a$10$lzD1LknnO.6oP4KTejmAjucbfAmcJ62gOn.US3.gw0PoLuXdADLDS
721	ENCtest1@email.com	$2a$10$1LWorToGCqqmjGWQZlpuYuD7.BhB9XMdKt0Tj2GTVLaFL6plB.Lpy

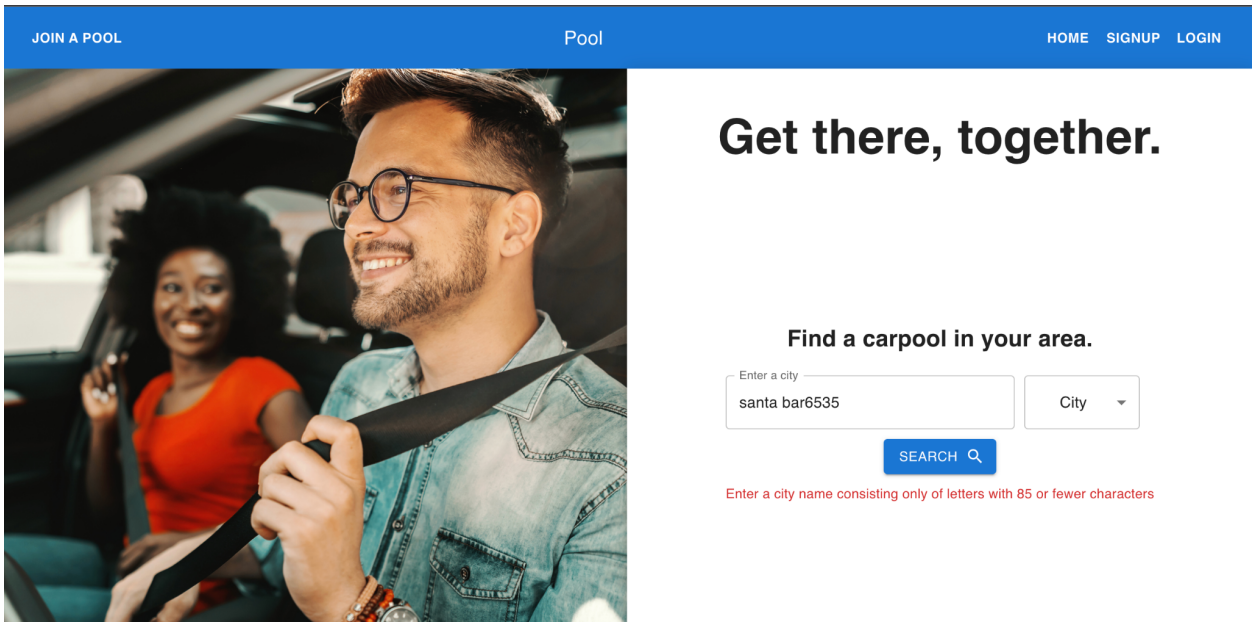
Result 6

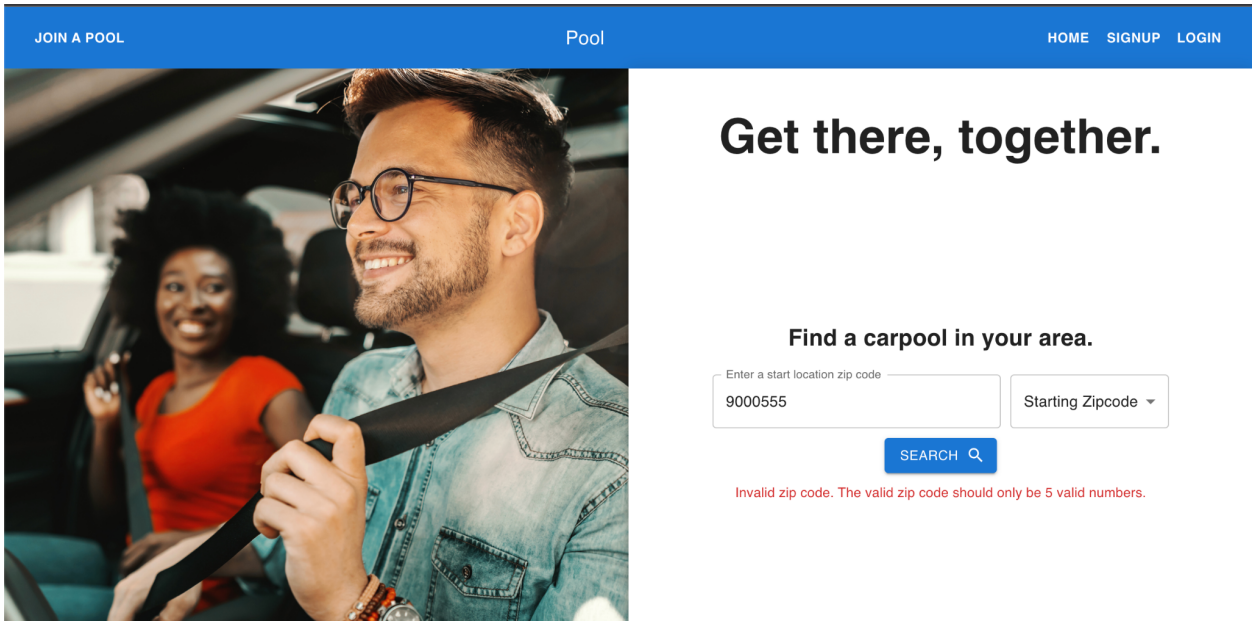
Input Validation

Home (carpoolwith.me)

Frontend Home screen - search bar input validation for zip code and city:







Nguyen Xuan Duy Anh, last week | 2 authors (Nguyen Xuan Duy Anh and others)

```
1  export function isZipCodeValid(input) {
2    const regex =
3      /^(?!0{5}|1{5}|2{5}|3{5}|4{5}|5{5}|6{5}|7{5}|8{5}|9{5})\d{5}(-\d{4})?$/;
4
5    if (input.length !== 5) {
6      return false;
7    }
8
9    return regex.test(input);
10 }
11
```

```
1  export function isCityNameValid(input) {
2    var regex = /^[A-Za-z ]+$/;
3
4    if (input.length > 85) {
5      return false;
6    }
7
8    return regex.test(input);
9  }
10
```

```
const handleKeyDown = (event) => {
  if (event.key === "Enter") {
    if (!filterOption) {
      setError(true);
      return;
    }
    setError(false);

    let hasError = false;

    if (
      (filterOption === "startZip" || filterOption === "endZip") &&
      !isZipCodeValid(searchQuery)
    ) {
      setZipCodeError(true);
      hasError = true;
    } else {
      setZipCodeError(false);
    }

    if (filterOption === "city" && !isCityNameValid(searchQuery)) {
      setCityNameError(true);
      hasError = true;
    } else {
      setCityNameError(false);
    }

    if (!hasError) {
      setHasSearched(true);
      onSearch({ searchQuery, filterOption });
    }
  }
};
```

```
const handleClick = () => {
  if (!filterOption) {
    setError(true);
    return;
  }
  setError(false);

  let hasError = false;

  if (
    (filterOption === "startZip" || filterOption === "endZip") &&
    !isZipCodeValid(searchQuery)
  ) {
    setZipCodeError(true);
    hasError = true;
  } else {
    setZipCodeError(false);
  }

  if (filterOption === "city" && !isCityNameValid(searchQuery)) {
    setCityNameError(true);
    hasError = true;
  } else {
    setCityNameError(false);
  }

  if (!hasError) {
    setHasSearched(true);
    onSearch({ searchQuery, filterOption });
  }
};
```

```
            <MenuItem value="" disabled>
              Select
            </MenuItem>
            <MenuItem value="startZip">Starting Zipcode</MenuItem>
            <MenuItem value="endZip">Ending Zipcode</MenuItem>
            <MenuItem value="city">City</MenuItem>
          </Select>
        </div>
        <Button
          variant="contained"
          color="primary"
          endIcon={<SearchIcon />}
          onClick={handleClick}
          style={{ marginTop: "10px", marginBottom: "10px" }}
        >
          Search
        </Button>
        {(error || zipCodeError || cityNameError) && (
          <Typography color="error" variant="body2">
            {error && "Please select an option before searching."}
            {zipCodeError &&
              "Invalid zip code. The valid zip code should only be 5 valid numbers."}
            {cityNameError &&
              "Enter a city name consisting only of letters with 85 or fewer characters"}
          </Typography>
        )}
      </div>
    );
```

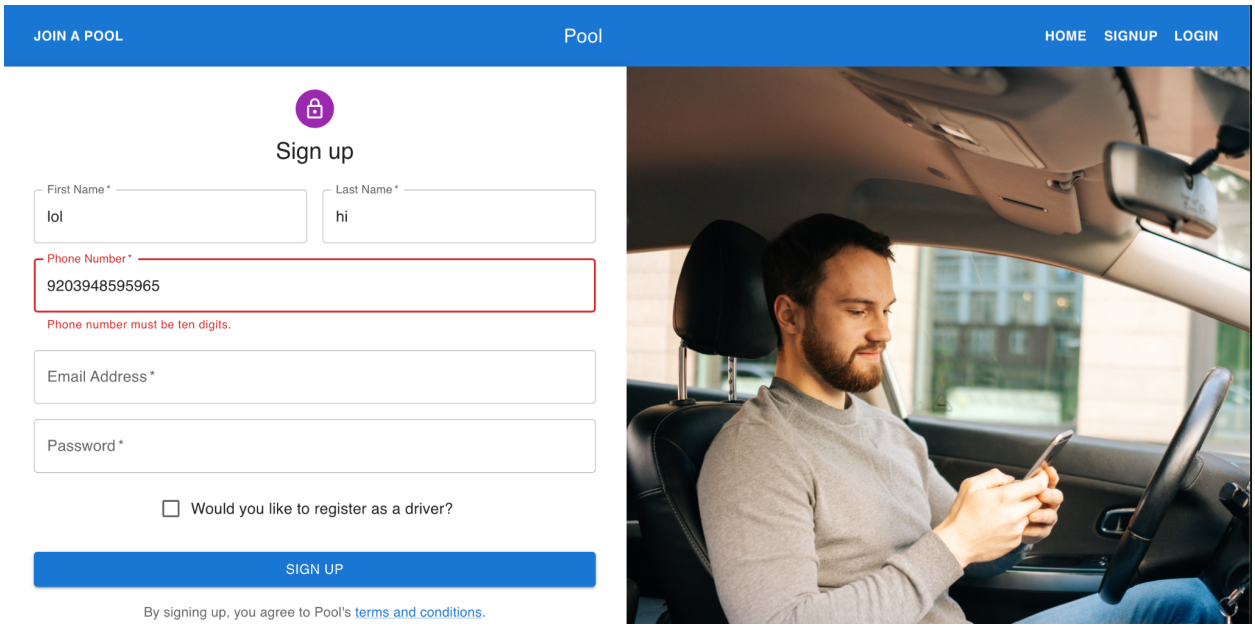
Backend Home screen - search bar input validations for zip code and city:

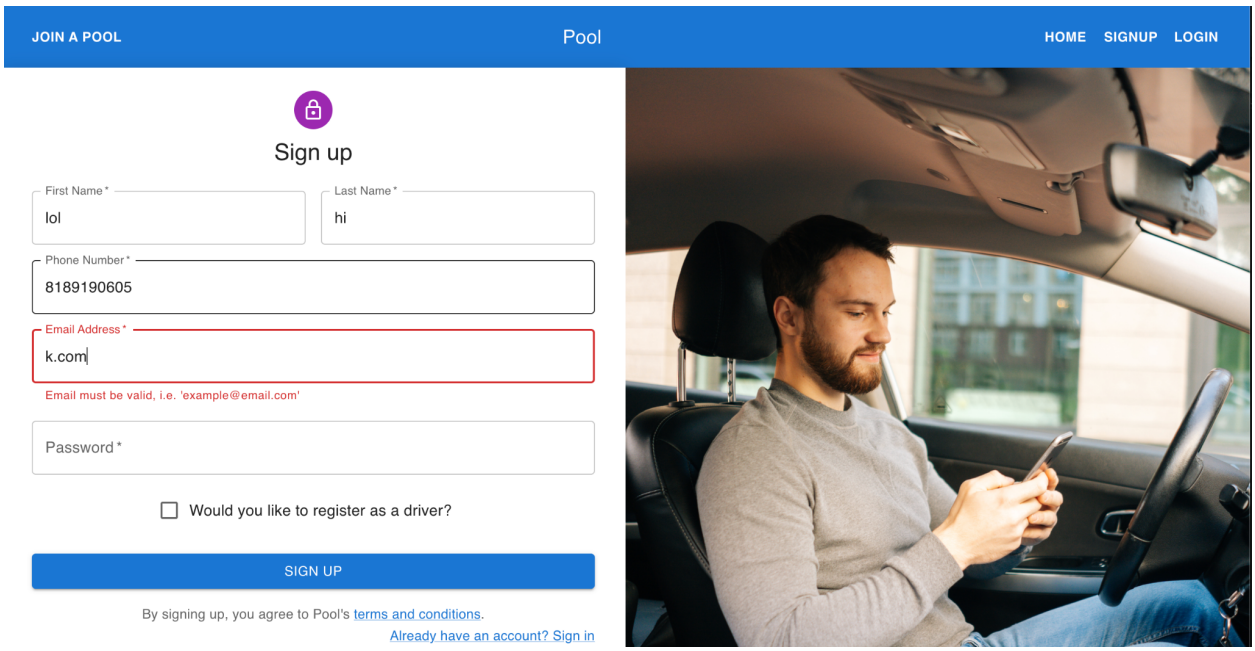
```
1 usage   kkrstchn +1
public CityValidationEnum cityInputValidation(String city) {
    if(city == null || city.isBlank()) {
        return CityValidationEnum.CITY_HAS_BLANK_NAME;
    } else if (!city.matches( regex: "^[a-zA-Z\\- ]+$")) { // Allows alphabetic chars, hyphens, and spaces
        return CityValidationEnum.CITY_NAME_CONTAINS_NON_ALPHA_CHARS;
    } else if (city.length() > 85) {
        return CityValidationEnum.CITY_NAME_TOO_LONG;
    }
    return CityValidationEnum.CITY_IS_VALID;
}
```

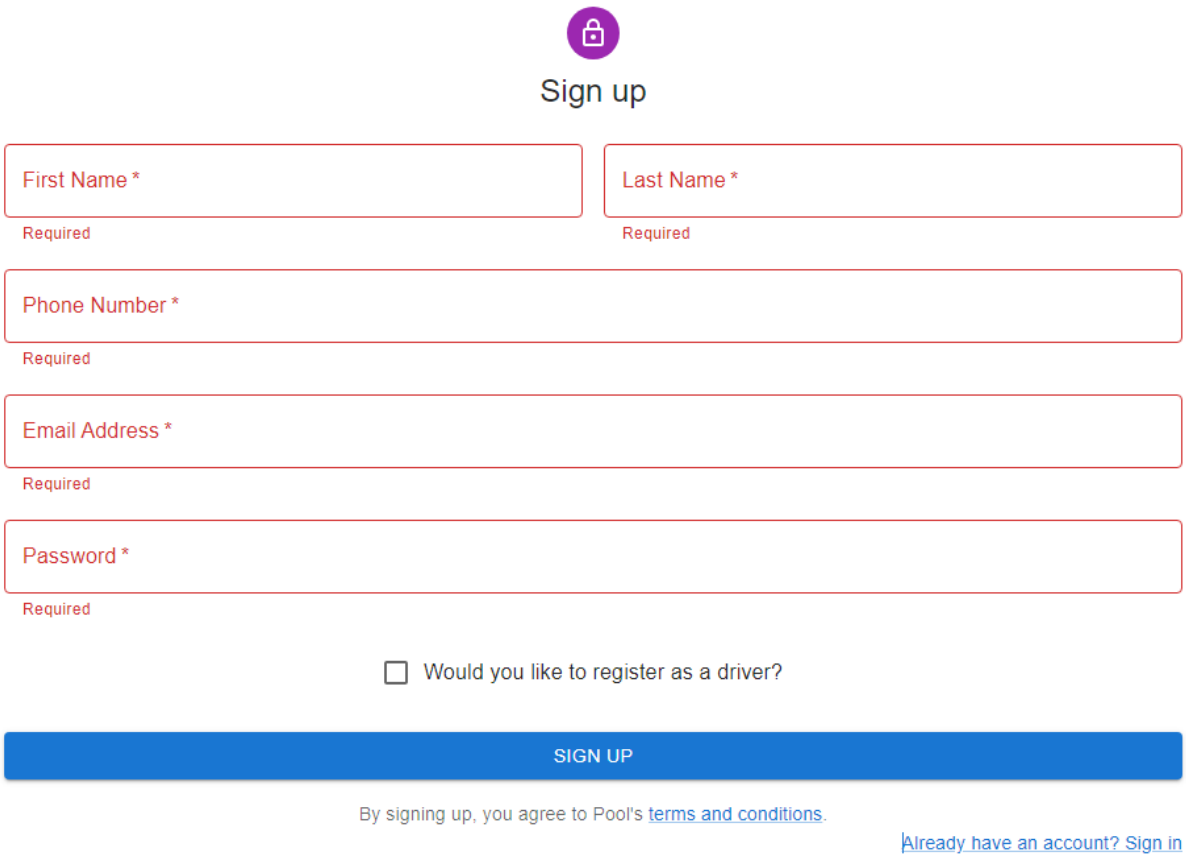
```
2 usages   kkrstchn
public ZipCodeValidationEnum zipcodeValidation (String zipcode) {
    System.out.println("zip validaztion, in zip service: " + zipcode);
    if (zipcode == null || zipcode.isBlank()) {
        return ZipCodeValidationEnum.BLANK_ZIP;
    }else if (!zipcode.matches( regex: "^[0-9]+$")) {
        return ZipCodeValidationEnum.HAS_NON_NUMERIC_CHAR;
    } else if (zipcode.length() != 5) {
        return ZipCodeValidationEnum.ZIP_DOES_NOT_HAVE_CORRECT_LENGTH;
    }

    return ZipCodeValidationEnum.VALID_ZIP;
}
```

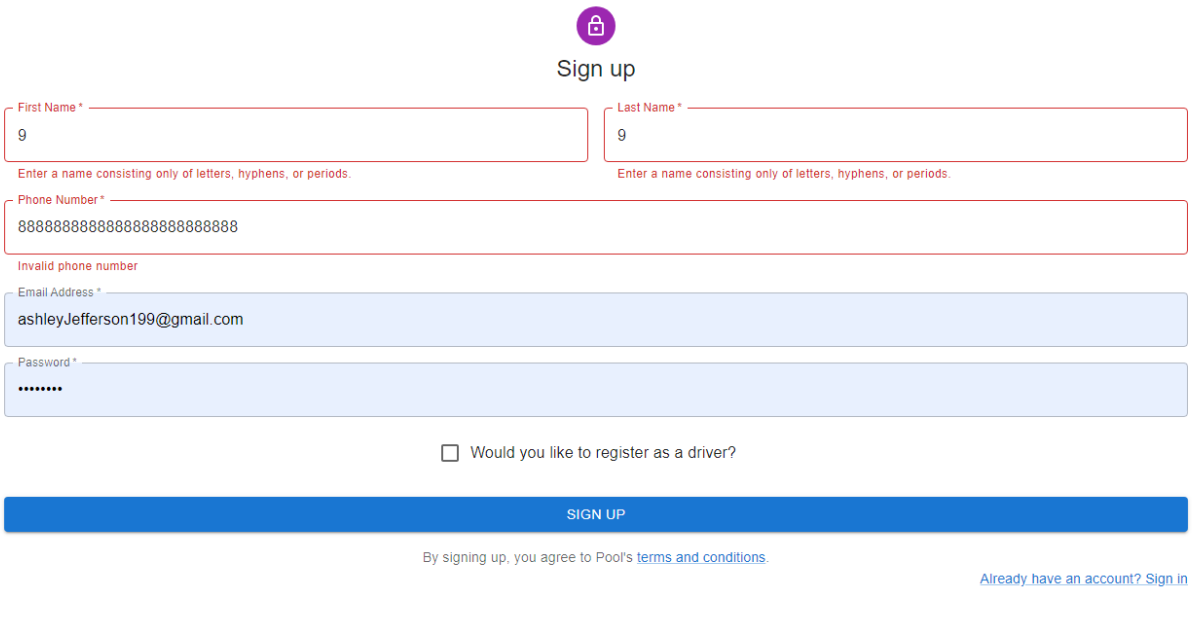
Sign up screen (carpoolwith.me/signup)

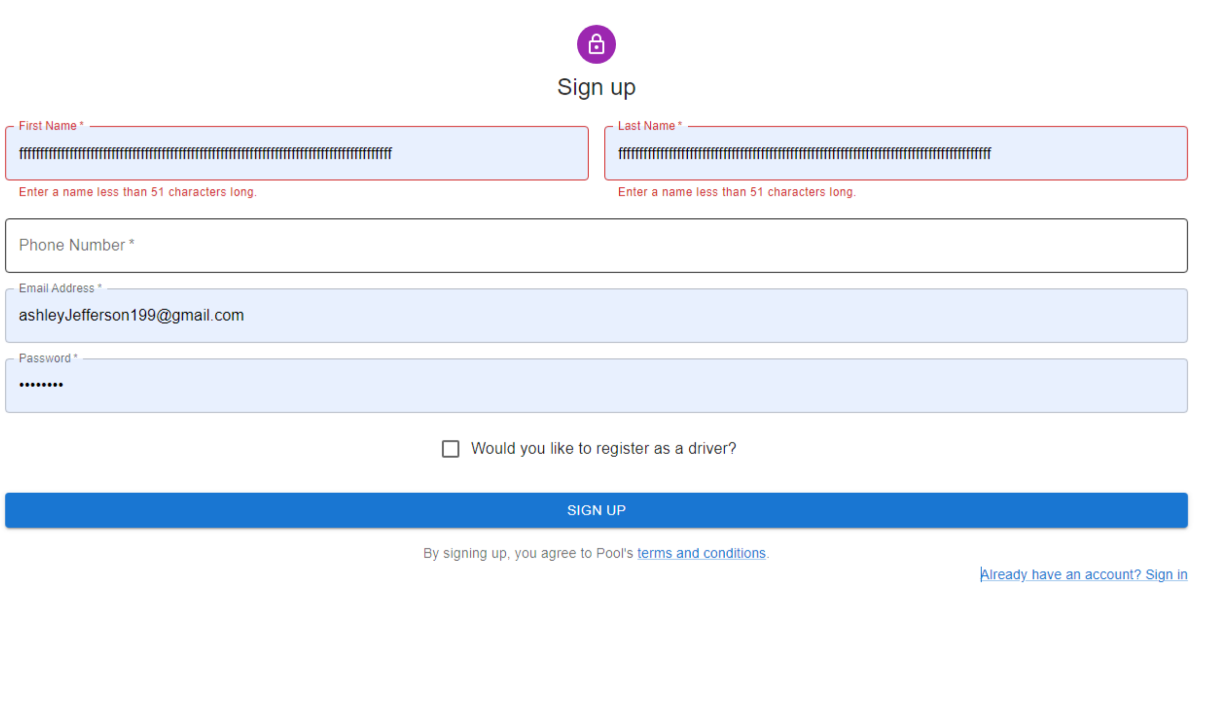
Frontend validations (first name, last name, phone number, email, password):






Sign up
First Name *
Required
Last Name *
Required
Phone Number *
Required
Email Address *
Required
Password *
Required
Would you like to register as a driver?
SIGN UP
By signing up, you agree to Pool's terms and conditions.
Already have an account? Sign in


Sign up
First Name *
9
Enter a name consisting only of letters, hyphens, or periods.
Last Name *
9
Enter a name consisting only of letters, hyphens, or periods.
Phone Number *
88888888888888888888888
Invalid phone number
Email Address *
ashleyJefferson199@gmail.com
Password *
••••••••
Would you like to register as a driver?
SIGN UP
By signing up, you agree to Pool's terms and conditions.
Already have an account? Sign in


Sign up
First Name *
Last Name *
Enter a name less than 51 characters long.
Enter a name less than 51 characters long.
Phone Number *
Email Address *
ashleyJefferson199@gmail.com
Password *
Would you like to register as a driver?
SIGN UP
By signing up, you agree to Pool's terms and conditions.
Already have an account? Sign in

```
if (firstName.length > 50) {
  setFirstNameError("Enter a name less than 51 characters long.");
  return;
} else {
  setFirstNameError(null);
}
if (lastName.length > 50) {
  setLastNameError("Enter a name less than 51 characters long.");
  return;
} else {
  setLastNameError(null);
}
const firstNameRegex = /^[a-zA-Z.-]+$/;
if (!firstNameRegex.test(firstName)) {
  setFirstNameCharError("Enter a name consisting only of letters, hyphens, or periods.");
  return;
} else {
  setFirstNameCharError(null);
}
const lastNameRegex = /^[a-zA-Z.-]+$/;
if (!lastNameRegex.test(lastName)) {
  setLastNameCharError("Enter a name consisting only of letters, hyphens, or periods.");
  return;
} else {
  setLastNameCharError(null);
}
const requestBody = {
  firstName,
  lastName,
  phoneNumber,
  email,
  password,
  fasTrakVerification,
  driversLicense,
  role: driversLicense ? "driver" : "passenger",
};
```

```
if (!isPhoneNumberValid(phoneNumber)) {
  setPhoneError("Invalid phone number");
  return;
} else {
  setPhoneError(false);
}

if (driversLicense) {
  if (!licenseRegex.test(driversLicense)) {
    setLicenseError("Invalid license");
    return;
  } else {
    setLicenseError(false);
  }
}

if (!email || !emailRegex.test(email)) {
  setEmailError("Invalid email");
  return;
} else {
  setEmailError(false);
}

if (!agreeTerms) {
  setAgreeTermsError(
    "You must agree to the Terms and Conditions to continue."
  );
  return;
} else {
  setAgreeTermsError(null);
}
```

```
export function isPhoneNumberValid(input) {
  //check if it has 10 characters
  console.log("aaaa", !/^\d{10}$/.test(input));
  if (!/^\d{10}$/.test(input)) {
    return "Phone number must be ten digits.";
  }

  //check if input isnt null
  if (input === null) {
    return;
  }
      You, yesterday • Uncommitted changes

  //checking for list validation
  if (input === "1234567890") {
    return "Phone number can't be 1234567890";
  }

  //checking for repeated characters
  if (/^(\d)\1{9}$/.test(input)) {
    return "Phone number can not consist of ten identical numbers.";
  }

  //first or 4th cant be 0 or 1
  if (
    input[0] === "0" ||
    input[0] === "1" ||
    input[3] === "0" ||
    input[3] === "1"
  ) {
    return "First or fourth digit can not be 0 or 1";
  }
  return;
}

```

```
if (isPhoneNumberValid(phoneNumber)) {
  setPhoneError(isPhoneNumberValid(phoneNumber));
  return;
} else {
  setPhoneError(false);
}
```

```
const emailRegex = /^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/;
if (!emailRegex.test(email)) {
  setEmailError("Email must be valid, i.e. 'example@email.com'");
  return;
} else {
  setEmailError(null);
}
```

Backend validation for signup api:

```
1 usage  kkrstchn
public SignUpValidationEnum signUpValidation (String signUpKey, String signUpValue) {
    if (signUpValue == null || signUpValue.isBlank()) {
        return SignUpValidationEnum.IS_EMPTY;
    }
    if (signUpKey.equals("firstName") || signUpKey.equals("lastName")) {
        SignUpValidationEnum signUpValidationEnum = validateUserName(signUpValue);
        if (signUpValidationEnum != SignUpValidationEnum.IS_VALID) {
            return signUpValidationEnum;
        }
    }
    if (signUpKey.equals("phoneNumber")) {
        SignUpValidationEnum signUpValidationEnum = validatePhoneNumber(signUpValue);
        if (signUpValidationEnum != SignUpValidationEnum.IS_VALID) {
            return signUpValidationEnum;
        }
    }
    return SignUpValidationEnum.IS_VALID;
}
```

```
1 usage  kkrstchn
private SignUpValidationEnum validateUserName  (String name) {
    if (name == null || name.isBlank()) {
        return SignUpValidationEnum.IS_EMPTY;
    } else if (name.length() > 50) {
        return SignUpValidationEnum.IS_LONGER_THAN_50_CHAR;
    } else if (!name.matches( regex: "^[a-zA-Z-.]+$")) { //matches letters, hyphens and periods in names
        return SignUpValidationEnum.INVALID_CHARACTERS_OTHER_THAN_HYPHEN_OR_PERIOD;
    }
    return SignUpValidationEnum.IS_VALID;
}
```

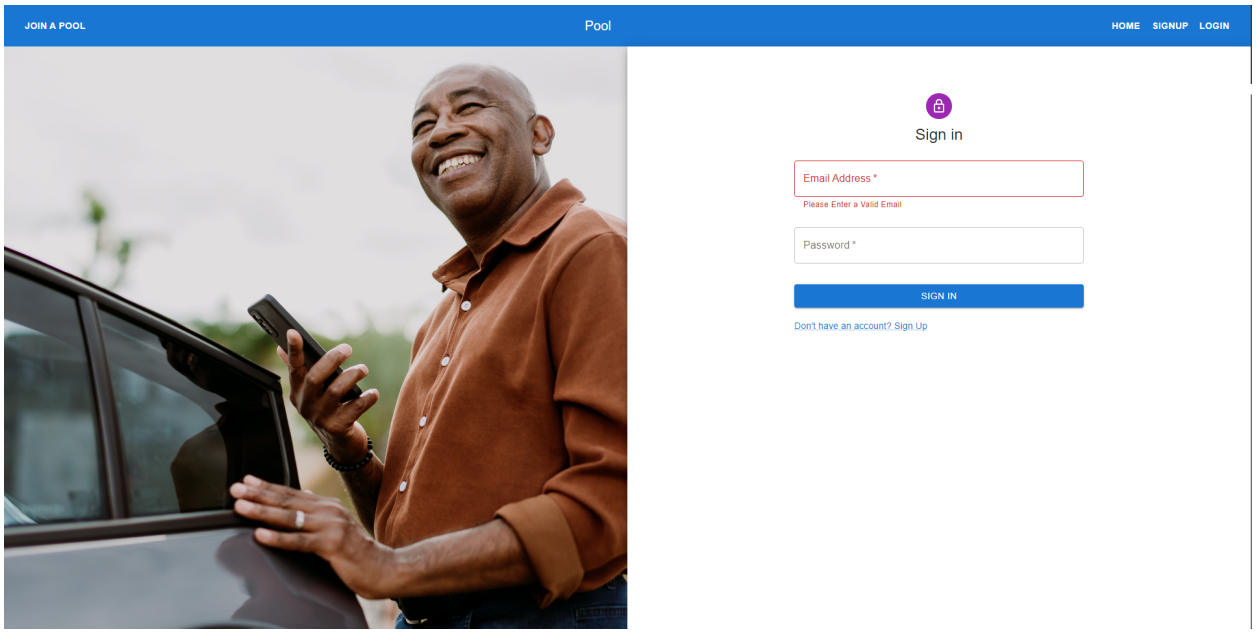
```
1 usage   kkrstchn *
private SignUpValidationEnum validatePhoneNumber(String phoneNumber) {
    if (phoneNumber == null || phoneNumber.isBlank()) {
        return SignUpValidationEnum.IS_EMPTY;
    } else if (phoneNumber.length() != 10) {
        return SignUpValidationEnum.PHONE_NUMBER_MUST_BE_LENGTH_0F_10_CHARACTERS;
    } else if (!phoneNumber.matches( regex: "^[0-9]+$")) {
        return SignUpValidationEnum.PHONE_NUMBER_MUST_CONSIST_OF_ONLY_NUMBERS;
    } else if (phoneNumber.matches( regex: "^(.)\\1+$")) {
        //phone number is one duplicate character repeating
        return SignUpValidationEnum.PHONE_NUMBER_COMPOSED_OF_DUPLICATE_CHARACTERS;
    } else if (isSequential(phoneNumber)) {
        return SignUpValidationEnum.NON_SEQUENTIAL_PHONE_NUMBER_REQUIRED;
    } else if (phoneNumber.matches( regex: "^(?:[01].{2}.|...[01]).*$")) {
        //area code validation 1st and 4th digit are 0 or 1 is invalid
        return SignUpValidationEnum.AREA_CODE_INVALID;
    }
    return SignUpValidationEnum.IS_VALID;
}
```

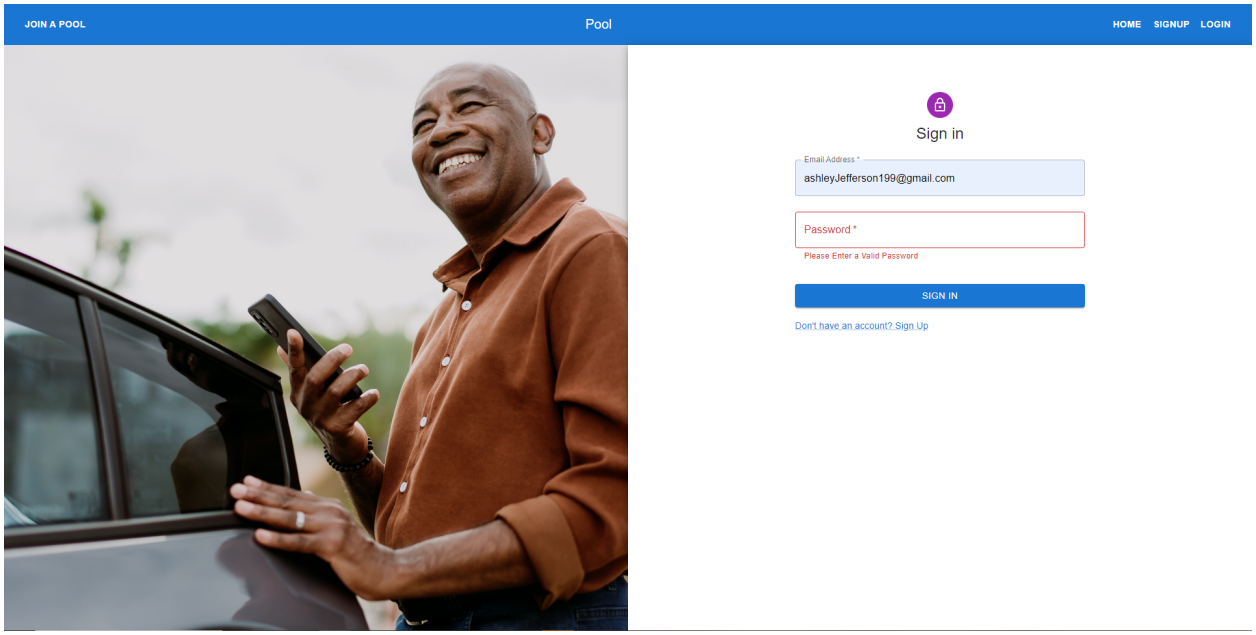
Sign in screen (carpoolwith.me/signin)

Backend validations - sign in (check if user exists)

```
        //check if user exists by email
        User user;
        if (optionalUser.isPresent()){
            user = optionalUser.get();
        } else {
            return new ResponseEntity<>( body: "Incorrect username or password", HttpStatus.UNAUTHORIZED);
        }

        Optional<Account> optionalAccount = accountRepository.findAccountByUserId(user.getUserId());

        //Check if account exists for the user
        Account account;
        if (optionalAccount.isPresent()) {
            account = optionalAccount.get();
        } else {
            return new ResponseEntity<>( body: "Could not find account", HttpStatus.UNAUTHORIZED);
        }

        //check if password matches
        boolean validPassword = isPasswordNull(jsonMap.get("password"));
        if(!validPassword){
            return new ResponseEntity<>( body: "Password is empty or null", HttpStatus.UNAUTHORIZED);
        }

        String inputPassword = jsonMap.get("password");
        String storedPassword = account.getPassword().getPassword(); //hashed password
        boolean isPasswordMatching = passwordEncoder.matches(inputPassword, storedPassword);

        if (isPasswordMatching) {
            // Check if user is a driver
            boolean isDriver = driverRepository.findByUser_UserId(user.getUserId()).isPresent();
            user.setIsDriver(isDriver);

           Profile profile= profileRepository.findByUserId(user);
```

Frontend Validations - Sign in (validate email and password)





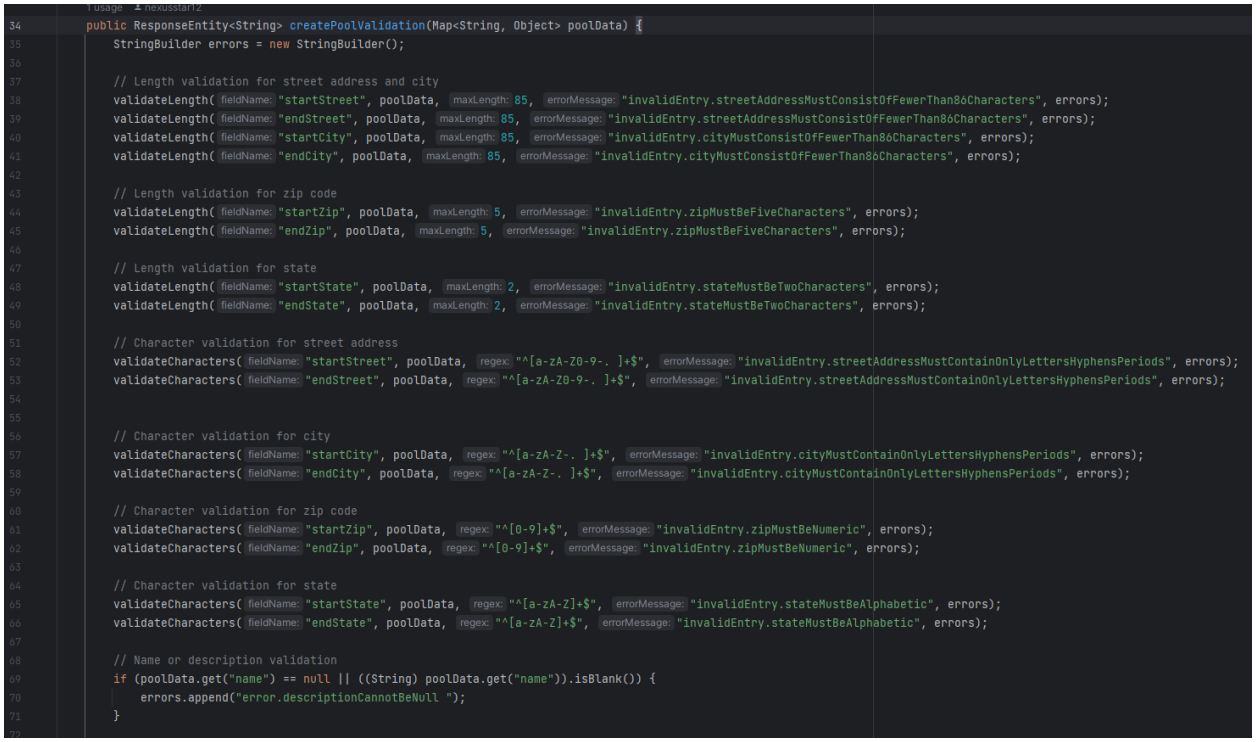
appfrontend > src > components > JS SignIn.js > SignIn

```
const defaultTheme = createTheme();

export default function SignIn() {
  const history = useNavigate();
  const [error, setError] = useState(null);
  const [emailError, setEmailError] = useState(null);
  const [passwordError, setPasswordError] = useState(null);
  const userContext = useContext(UserContext);
  const emailRegex = /^[\w-\.]+@([\w-]+\.)+[\w-]{2,4}$/;

  const handleSubmit = async (event) => {
    event.preventDefault();
    const data = new FormData(event.currentTarget);
    const email = data.get("email");
    const password = data.get("password");

    const requestBody = { email, password };


    if (!email || !emailRegex.test(email)) {
      setEmailError("Please Enter a Valid Email");
      return;
    } else {
      setEmailError(false);
    }
    if(!password){
      setPasswordError("Please Enter a Valid Password");
      return;
    }
    else{
      setPasswordError(false);
    }
    try {
      const response = await axiosInstance.post("/signin", requestBody);
      if (response.status === 200) {
        localStorage.setItem("userInfo", JSON.stringify(response.data));
        userContext.setUserInfo(response.data);
        history("/", { replace: true });
      }
    } catch (error) {
      setError(error.response.data);
    }
  };

```

```
<Box
  component="form"
  noValidate
  onSubmit={handleSubmit}
  sx={{ mt: 1 }}
>
  <TextField
    margin="normal"
    required
    fullWidth
    id="email"
    label="Email Address"
    name="email"
    autoComplete="email"
    autoFocus
    error = {!!emailError}
    helperText={emailError}
  />
  <TextField
    margin="normal"
    required
    fullWidth
    name="password"
    label="Password"
    type="password"
    id="password"
    autoComplete="current-password"
    error = {!!passwordError}
    helperText={passwordError}
  />
  <Typography color="error" variant="body2">
    {error}
  </Typography>
```

Post a Pool (carpoowith.me/create-pool)

Backend validations for pool description, address, and privacy settings



```
1 usage  1 nexusstar12
public ResponseEntity<String> createPoolValidation(Map<String, Object> poolData) {
    StringBuilder errors = new StringBuilder();

    // Length validation for street address and city
    validateLength( fieldName: "startStreet", poolData,  maxLength: 85,  errorMessage: "invalidEntry.streetAddressMustConsistOfFewerThan86Characters", errors);
    validateLength( fieldName: "endStreet", poolData,  maxLength: 85,  errorMessage: "invalidEntry.streetAddressMustConsistOfFewerThan86Characters", errors);
    validateLength( fieldName: "startCity", poolData,  maxLength: 85,  errorMessage: "invalidEntry.cityMustConsistOfFewerThan86Characters", errors);
    validateLength( fieldName: "endCity", poolData,  maxLength: 85,  errorMessage: "invalidEntry.cityMustConsistOfFewerThan86Characters", errors);

    // Length validation for zip code
    validateLength( fieldName: "startZip", poolData,  maxLength: 5,  errorMessage: "invalidEntry.zipMustBeFiveCharacters", errors);
    validateLength( fieldName: "endZip", poolData,  maxLength: 5,  errorMessage: "invalidEntry.zipMustBeFiveCharacters", errors);

    // Length validation for state
    validateLength( fieldName: "startState", poolData,  maxLength: 2,  errorMessage: "invalidEntry.stateMustBeTwoCharacters", errors);
    validateLength( fieldName: "endState", poolData,  maxLength: 2,  errorMessage: "invalidEntry.stateMustBeTwoCharacters", errors);

    // Character validation for street address
    validateCharacters( fieldName: "startStreet", poolData,  regex: "^[a-zA-Z0-9-. ]+$",  errorMessage: "invalidEntry.streetAddressMustContainOnlyLettersHyphensPeriods", errors);
    validateCharacters( fieldName: "endStreet", poolData,  regex: "^[a-zA-Z0-9-. ]+$",  errorMessage: "invalidEntry.streetAddressMustContainOnlyLettersHyphensPeriods", errors);


    // Character validation for city
    validateCharacters( fieldName: "startCity", poolData,  regex: "^[a-zA-Z-. ]+$",  errorMessage: "invalidEntry.cityMustContainOnlyLettersHyphensPeriods", errors);
    validateCharacters( fieldName: "endCity", poolData,  regex: "^[a-zA-Z-. ]+$",  errorMessage: "invalidEntry.cityMustContainOnlyLettersHyphensPeriods", errors);

    // Character validation for zip code
    validateCharacters( fieldName: "startZip", poolData,  regex: "^[0-9]+$",  errorMessage: "invalidEntry.zipMustBeNumeric", errors);
    validateCharacters( fieldName: "endZip", poolData,  regex: "^[0-9]+$",  errorMessage: "invalidEntry.zipMustBeNumeric", errors);

    // Character validation for state
    validateCharacters( fieldName: "startState", poolData,  regex: "^[a-zA-Z]+$",  errorMessage: "invalidEntry.stateMustBeAlphabetic", errors);
    validateCharacters( fieldName: "endState", poolData,  regex: "^[a-zA-Z]+$",  errorMessage: "invalidEntry.stateMustBeAlphabetic", errors);

    // Name or description validation
    if (poolData.get("name") == null || ((String) poolData.get("name")).isBlank()) {
        errors.append("error.descriptionCannotBeNull ");
    }
```

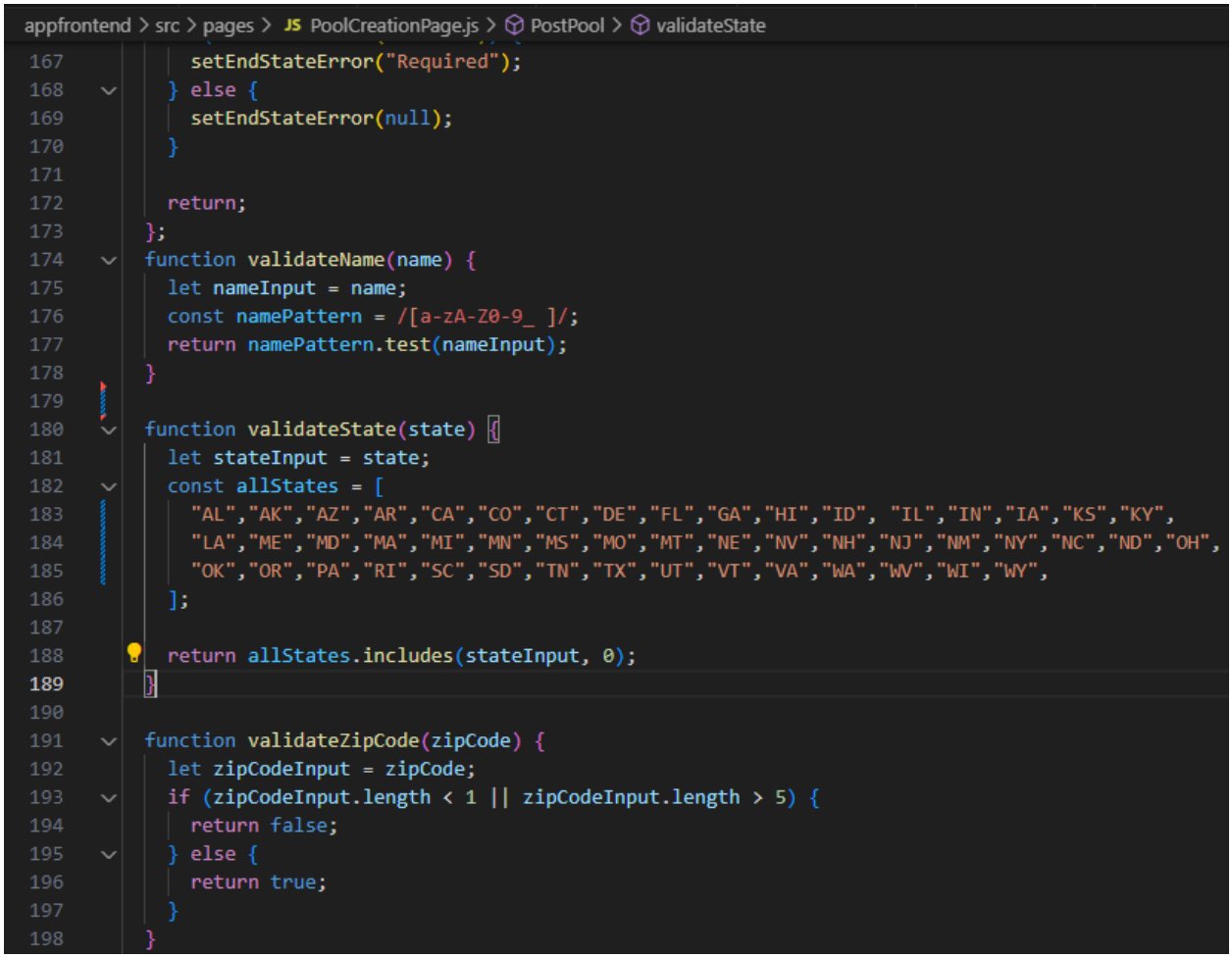
```
        // startStreet validation
        if (poolData.get("startStreet") == null || ((String) poolData.get("startStreet")).isBlank()) {
            errors.append("error.startStreetCannotBeNull ");
        }

        // startCity validation
        if (poolData.get("startCity") == null || ((String) poolData.get("startCity")).isBlank()) {
            errors.append("error.startCityCannotBeNull ");
        }

        // startZip validation
        if (poolData.get("startZip") == null || ((String) poolData.get("startZip")).isBlank()) {
            errors.append("error.startZipCannotBeNull ");
        }

        // startState validation
        if (poolData.get("startState") == null || ((String) poolData.get("startState")).isBlank()) {
            errors.append("error.startStateCannotBeNull ");
        }

        // endStreet validation
        if (poolData.get("endStreet") == null || ((String) poolData.get("endStreet")).isBlank()) {
            errors.append("error.endStreetCannotBeNull ");
        }

        // endCity validation
        if (poolData.get("endCity") == null || ((String) poolData.get("endCity")).isBlank()) {
            errors.append("error.endCityCannotBeNull ");
        }

        // endZip validation
        if (poolData.get("endZip") == null || ((String) poolData.get("endZip")).isBlank()) {
            errors.append("error.endZipCannotBeNull ");
        }

        // endState validation
        if (poolData.get("endState") == null || ((String) poolData.get("endState")).isBlank()) {
            errors.append("error.endStateCannotBeNull ");
        }

        // Privacy validation
        if (poolData.get("privacy") == null) {
            errors.append("error.privacyCannotBeNull ");
        }
```

```
        // Character validation for street address to contain at least one letter
        if (poolData.get("startStreet") != null && !containsLetter((String) poolData.get("startStreet"))) {
            errors.append("error.startStreetMustContainAtLeastOneLetter ");
        }
        if (poolData.get("endStreet") != null && !containsLetter((String) poolData.get("endStreet"))) {
            errors.append("error.endStreetMustContainAtLeastOneLetter ");
        }

        // Character validation for city to contain at least one letter
        if (poolData.get("startCity") != null && !containsLetter((String) poolData.get("startCity"))) {
            errors.append("error.startCityMustContainAtLeastOneLetter ");
        }
        if (poolData.get("endCity") != null && !containsLetter((String) poolData.get("endCity"))) {
            errors.append("error.endCityMustContainAtLeastOneLetter ");
        }

        // Check for startDateTime and validate it's in the future
        String startDateTimeStr = (String) poolData.get("startTime");
        if (startDateTimeStr != null && !startDateTimeStr.isBlank()) {
            try {
                LocalDateTime startDateTime = LocalDateTime.parse(startDateTimeStr);
                if (startDateTime.isBefore(LocalDateTime.now())) {
                    errors.append("invalidEntry.dateTimeMustBeInTheFuture ");
                }
            } catch (DateTimeParseException e) {
                errors.append("invalid.startTimeFormat ");
            }
        }

        // Return any errors here
        if (!errors.isEmpty()) {
            return ResponseEntity.badRequest().body(errors.toString().trim());
        }

        // If all fields are valid
        return ResponseEntity.ok( body: "All fields are valid");
    }
```

```
    // Check if a string contains at least one letter
    4 usages  nexusstar12
    private boolean containsLetter(String input) { return input.matches( regex: ".*[a-zA-Z]+.*"); }

    8 usages  nexusstar12
    private void validateLength(String fieldName, Map<String, Object> poolData, int maxLength, String errorMessage, StringBuilder errors) {
        String fieldValue = (String) poolData.get(fieldName);
        if (fieldValue != null && fieldValue.length() > maxLength) {
            errors.append(errorMessage).append(" ");
        }
    }

    8 usages  nexusstar12
    private void validateCharacters(String fieldName, Map<String, Object> poolData, String regex, String errorMessage, StringBuilder errors) {
        String fieldValue = (String) poolData.get(fieldName);
        if (fieldValue != null && !fieldValue.matches(regex)) {
            errors.append(errorMessage).append(" ");
        }
    }
```

Frontend Validations: Pool Description, address, and privacy settings

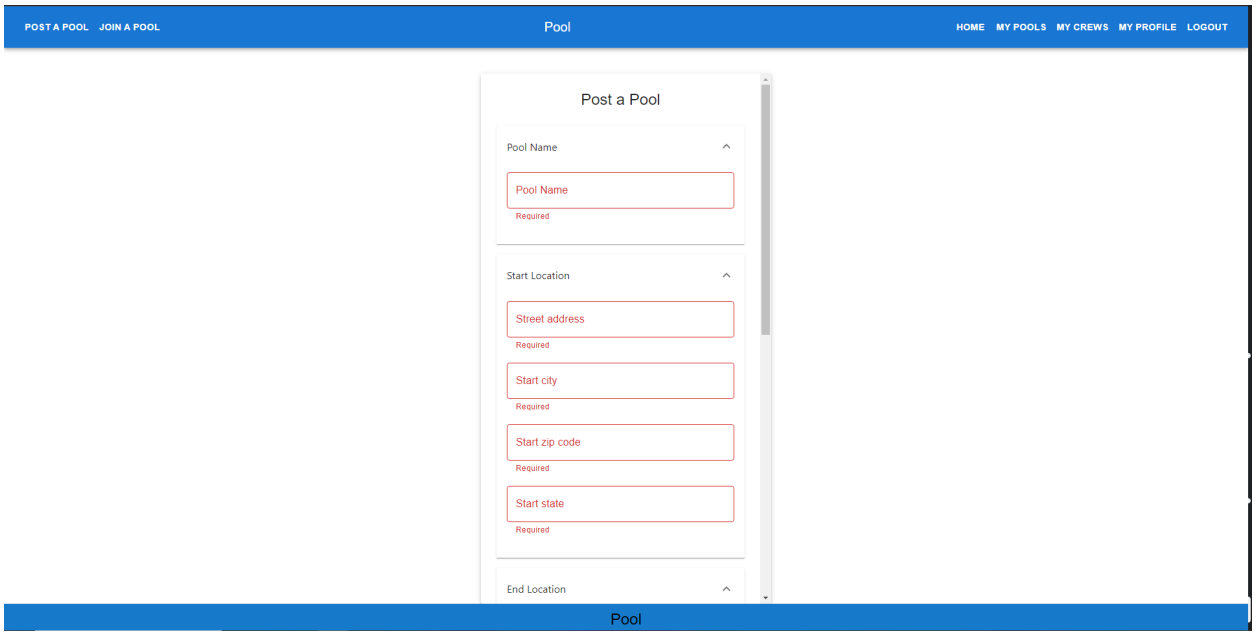
```
//field validations
if (!validateName(name)) {
  setNameError("Required");
} else {
  setNameError(null);
}
if (!startStreet) {
  setStartStreetError("Required");
} else {
  setStartStreetError(null);
}
if (!startCity) {
  setStartCityError("Required");
} else {
  setStartCityError(null);
}
if (!validateZipCode(startZip)) {
  setStartZipError("Required");
} else {
  setStartZipError(null);
}
if (!validateState(startState)) {
  setStartStateError("Required");
} else {
  setStartStateError(null);
}

if (!endStreet) {
  setEndStreetError("Required");
} else {
  setEndStreetError(null);
}
if (!endCity) {
  setEndCityError("Required");
} else {
  setEndCityError(null);
}
if (!validateZipCode(endZip)) {
  setEndZipError("Required");
} else {
  setEndZipError(null);
}
```

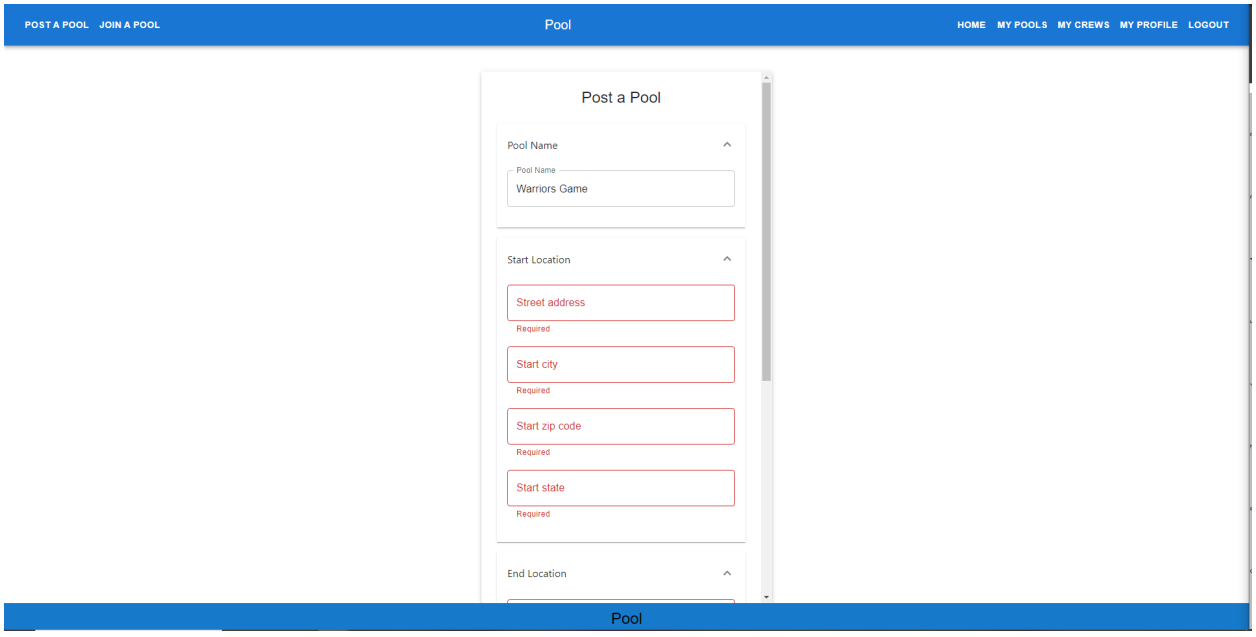

appfrontend > src > pages > JS PoolCreationPage.js > PostPool > validateState
setEndStateError("Required");
} else {
setEndStateError(null);
}
return;
};
function validateName(name) {
let nameInput = name;
const namePattern = /[a-zA-Z0-9_]/;
return namePattern.test(nameInput);
}
function validateState(state) {
let stateInput = state;
const allStates = [
"AL","AK","AZ","AR","CA","CO","CT","DE","FL","GA","HI","ID", "IL","IN","IA","KS","KY",
"LA","ME","MD","MA","MI","MN","MS","MO","MT","NE","NV","NH","NJ","NM","NY","NC","ND","OH",
"OK","OR","PA","RI","SC","SD","TN","TX","UT","VT","VA","WA","WV","WI","WY",
];
return allStates.includes(stateInput, 0);
}
function validateZipCode(zipCode) {
let zipCodeInput = zipCode;
if (zipCodeInput.length < 1 || zipCodeInput.length > 5) {
return false;
} else {
return true;
}
}

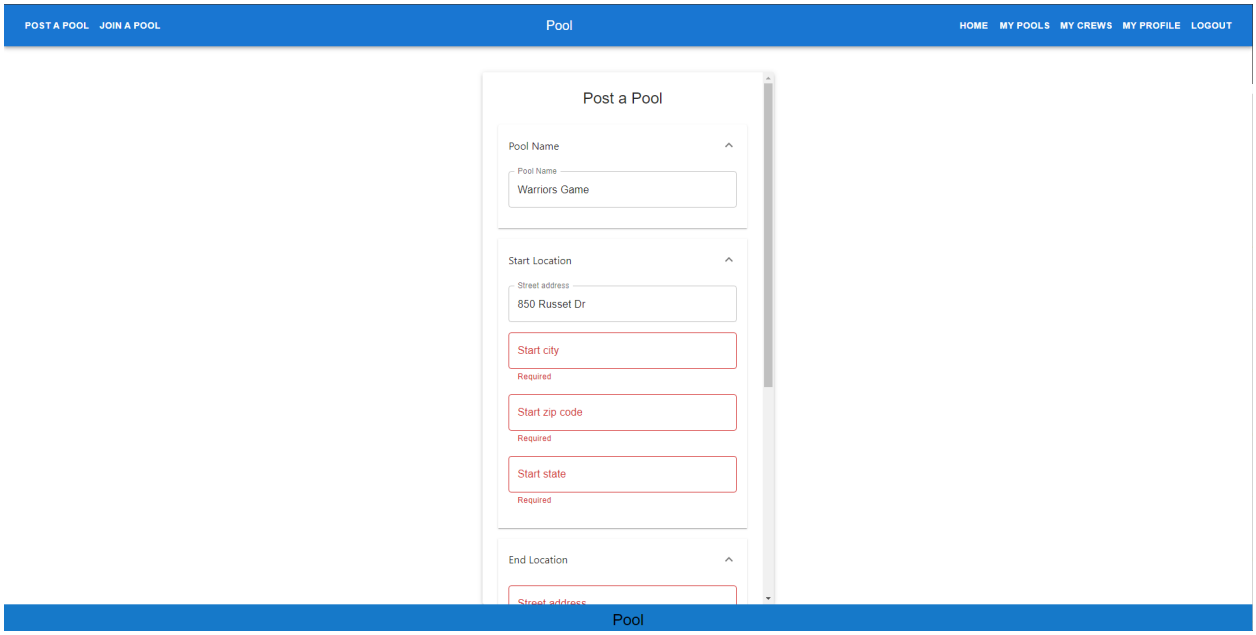

appfrontend > src > pages > JS PoolCreationPage.js > PostPool > validateState
225
226 return (
227 <Box
228 component="form"
229 noValidate
230 onSubmit={handleSubmit}
231 sx={{
232 mt: 5,
233 display: "flex",
234 flexDirection: "column",
235 alignItems: "center",
236 }}
237 >
238 <Paper
239 elevation={3}
240 sx={{
241 p: 3,
242 width: "80%",
243 maxWidth: 400,
244 overflowY: "auto",
245 maxHeight: "80vh",
246 }}
247 >
248 <Typography variant="h5" align="center" mb={3}>
249 Post a Pool
250 </Typography>
251 <Accordion>
252 <AccordionSummary expandIcon={<ExpandMoreIcon />}>
253 Pool Name
254 </AccordionSummary>
255 <AccordionDetails>
256 <TextField
257 fullWidth
258 name="name"
259 label="Pool Name"
260 sx={{ mb: 2 }}
261 error={!!nameError}
262 helperText={nameError}
263 />
264 </AccordionDetails>
265 </Accordion>

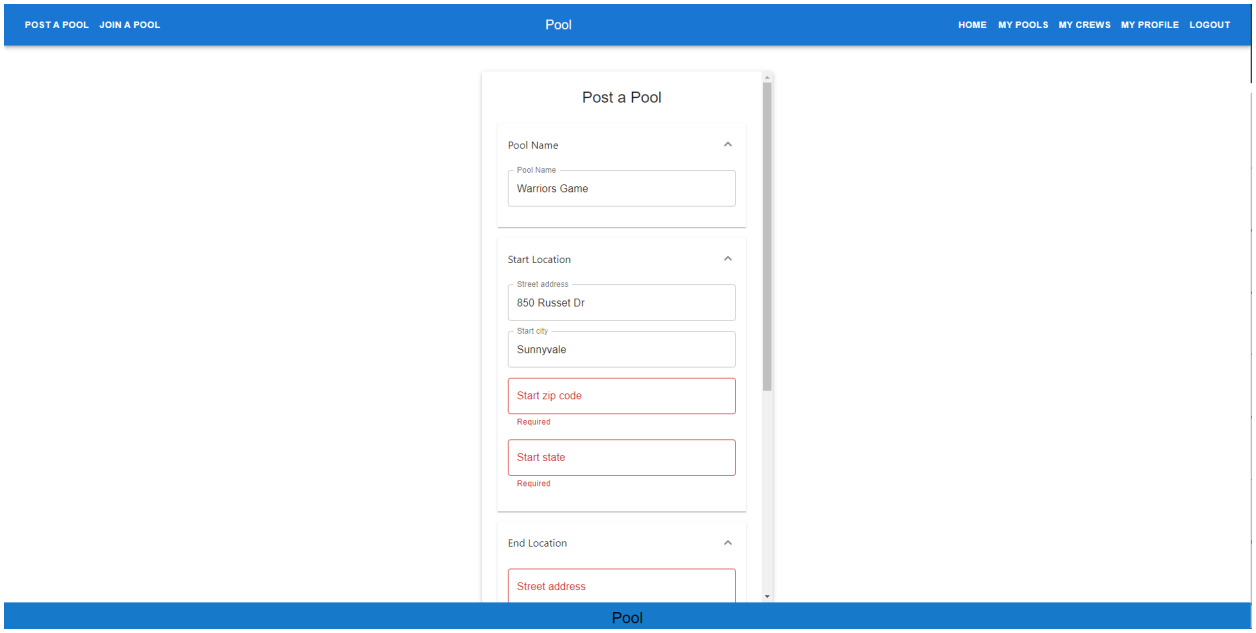
```
<Accordion>
  <AccordionSummary expandIcon={<ExpandMoreIcon />}>
    Start Location
  </AccordionSummary>
  <AccordionDetails>
    <TextField
      fullWidth
      name="startStreet"
      label="Street address"
      error={!!startStreetError}
      helperText={startStreetError}
      sx={{ mb: 2 }}
    />
    <TextField
      fullWidth
      name="startCity"
      label="Start city"
      error={!!startCityError}
      helperText={startCityError}
      sx={{ mb: 2 }}
    />
    <TextField
      fullWidth
      name="startZip"
      label="Start zip code"
      error={!!startZipError}
      helperText={startZipError}
      sx={{ mb: 2 }}
    />
    <TextField
      fullWidth
      name="startState"
      label="Start state"
      error={!!startStateError}
      helperText={startStateError}
      sx={{ mb: 2 }}
    />
  </AccordionDetails>
</Accordion>
```

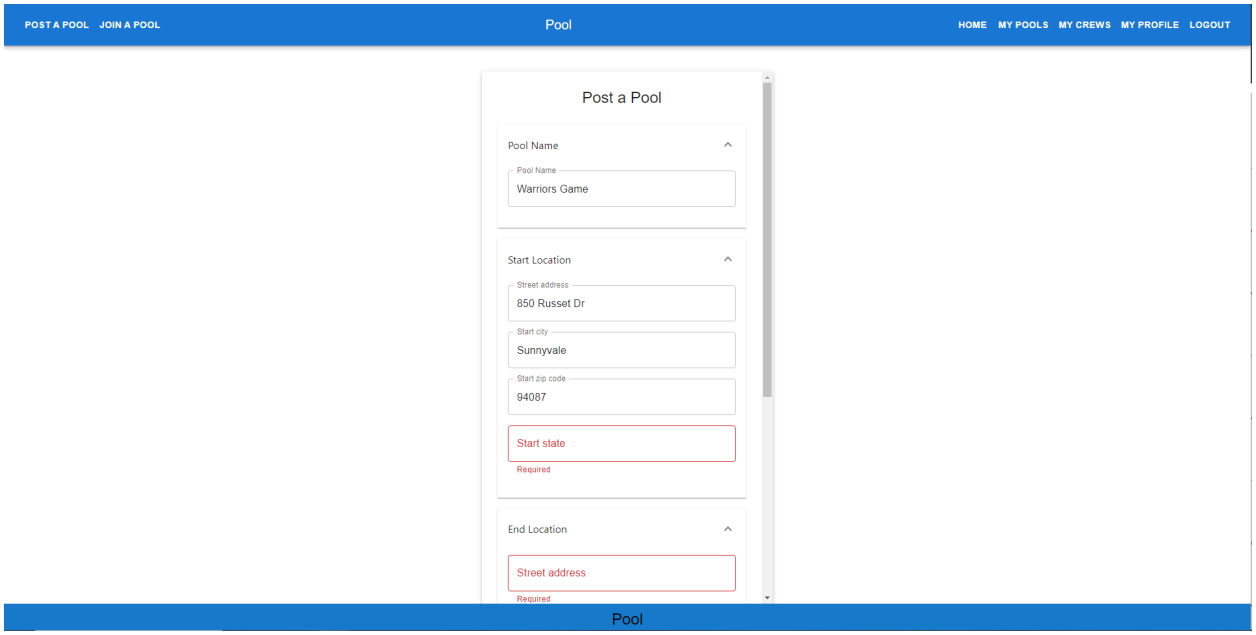
```jsx
{/* Enter end location section */}
<Accordion>
  <AccordionSummary expandIcon={<ExpandMoreIcon />}>
    End Location
  </AccordionSummary>
  <AccordionDetails>
    <TextField
      fullWidth
      name="endStreet"
      label="Street address"
      error={!!endStreetError}
      helperText={endStreetError}
      sx={{ mb: 2 }}
    />
    <TextField
      fullWidth
      name="endCity"
      label="End city"
      error={!!endCityError}
      helperText={endCityError}
      sx={{ mb: 2 }}
    />
    <TextField
      fullWidth
      name="endZip"
      label="End zip"
      sx={{ mb: 2 }}
      error={!!endZipError}
      helperText={endZipError}
    />
    <TextField
      fullWidth
      name="endState"
      label="End state"
      error={!!endStateError}
      helperText={endStateError}
      sx={{ mb: 2 }}
    />
  </AccordionDetails>
</Accordion>
```

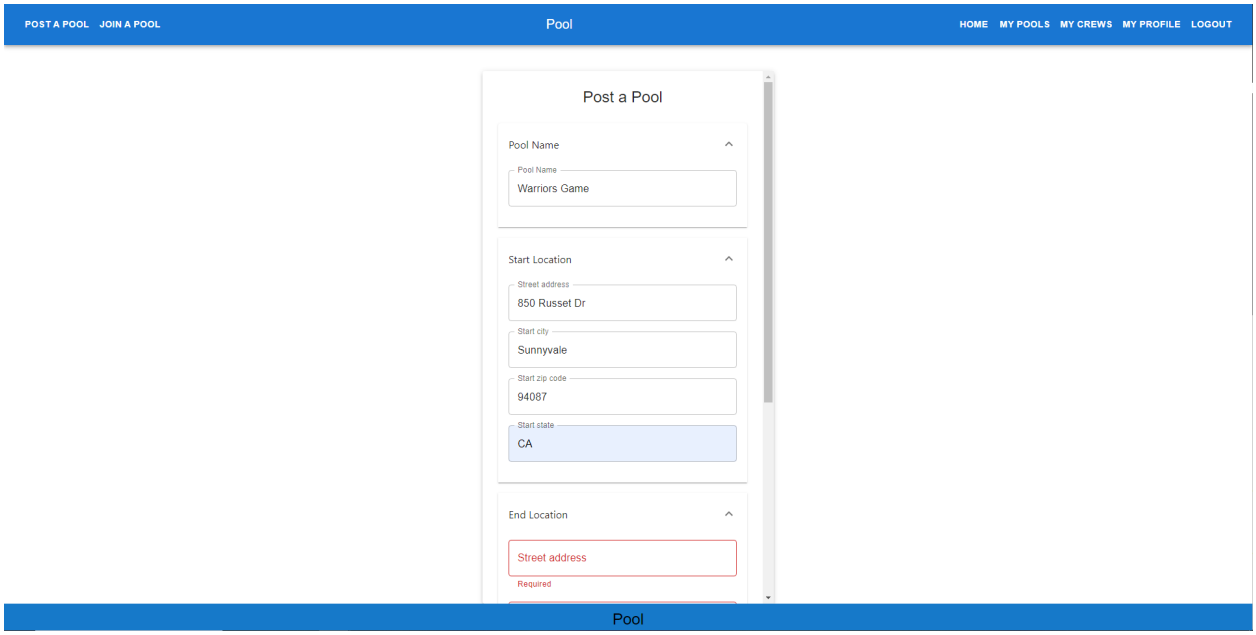

POST A POOL
JOIN A POOL
Pool
HOME
MY POOLS
MY CREWS
MY PROFILE
LOGOUT
Post a Pool
Pool Name
Pool Name
Required
Start Location
Street address
Required
Start city
Required
Start zip code
Required
Start state
Required
End Location
Pool

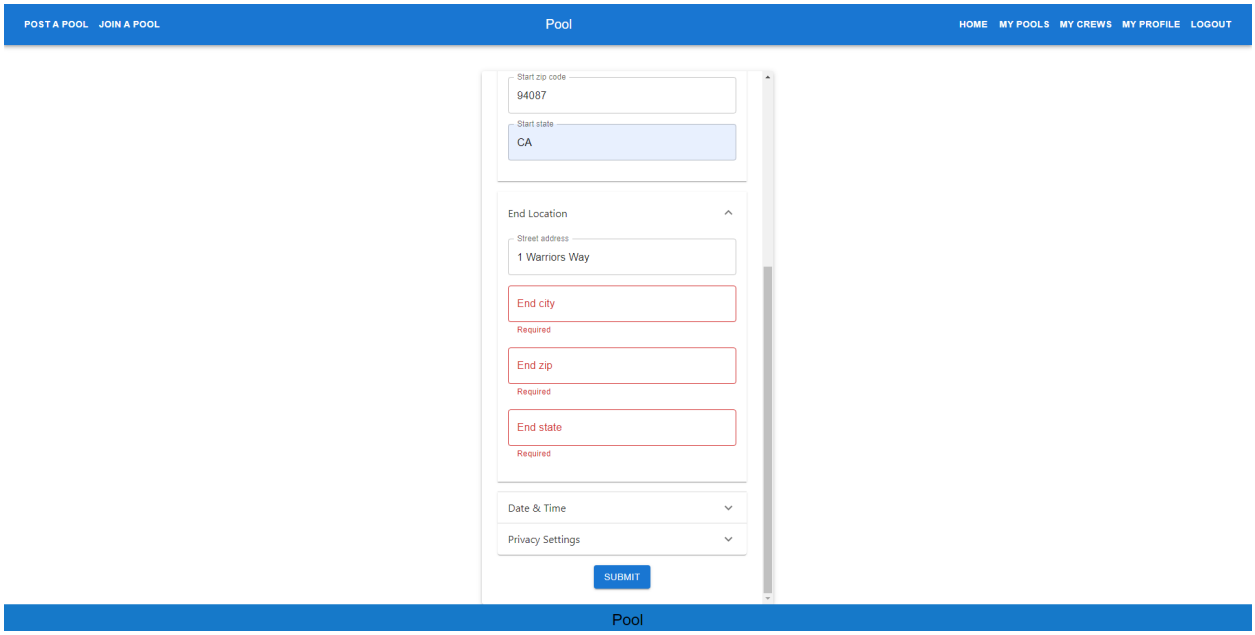

POST A POOL
JOIN A POOL
Pool
HOME
MY POOLS
MY CREWS
MY PROFILE
LOGOUT
Post a Pool
Pool Name
Pool Name
Warriors Game
Start Location
Street address
Required
Start city
Required
Start zip code
Required
Start state
Required
End Location
Pool

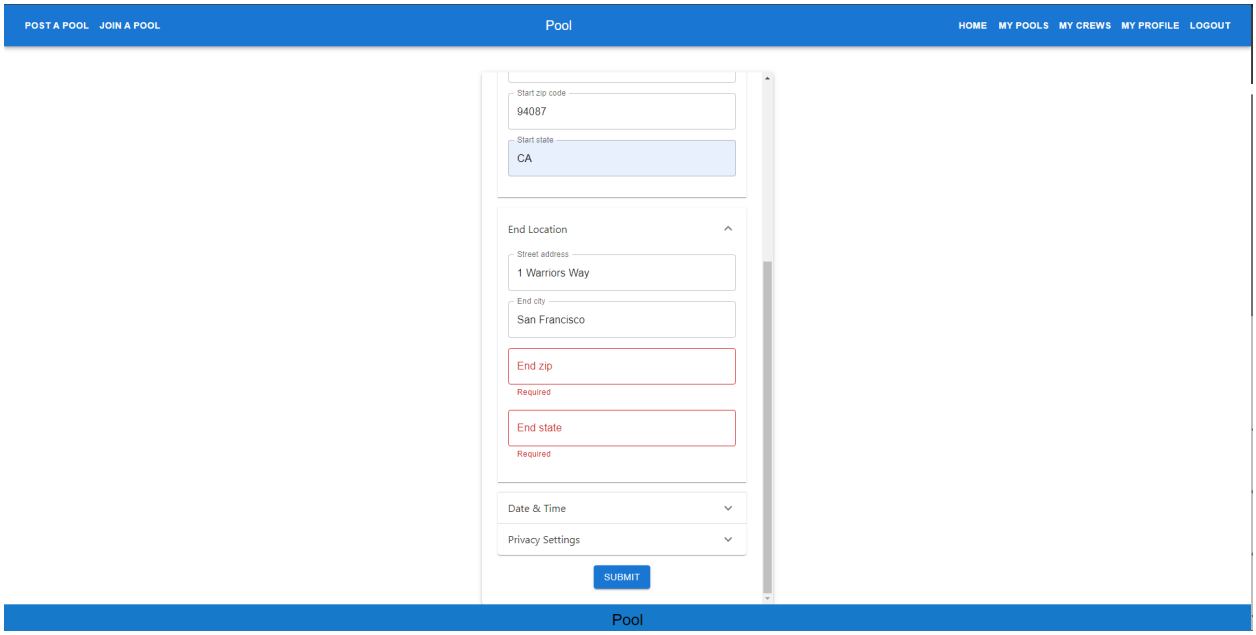

POST A POOL
JOIN A POOL
Pool
HOME
MY POOLS
MY CREWS
MY PROFILE
LOGOUT
Post a Pool
Pool Name
Pool Name
Warriors Game
Start Location
Street address
850 Russet Dr
Start city
Required
Start zip code
Required
Start state
Required
End Location
Pool

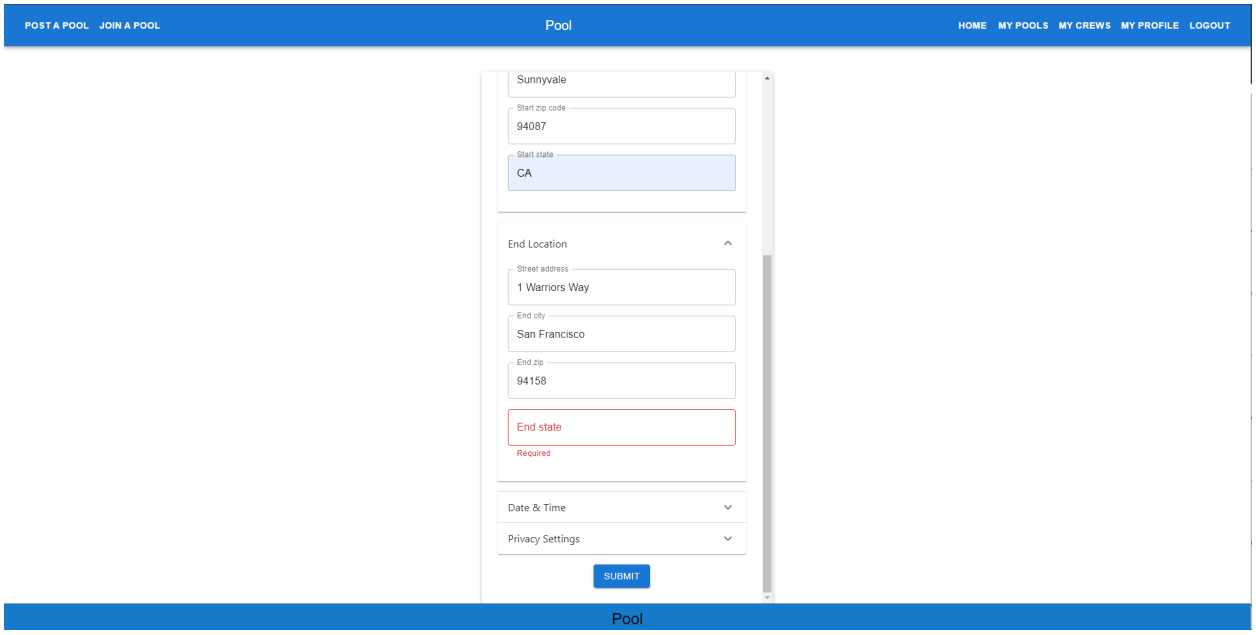

POST A POOL
JOIN A POOL
Pool
HOME
MY POOLS
MY CREWS
MY PROFILE
LOGOUT
Post a Pool
Pool Name
Pool Name
Warriors Game
Start Location
Street address
850 Russet Dr
Start city
Sunnyvale
Start zip code
Required
Start state
Required
End Location
Street address
Pool

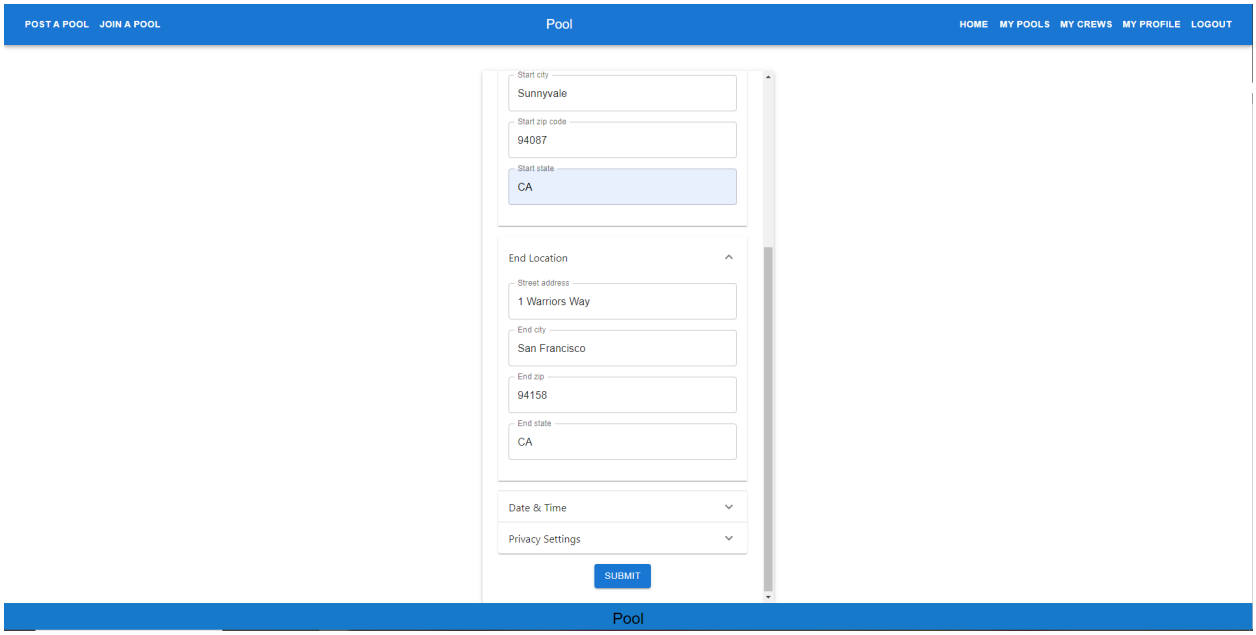

POST A POOL
JOIN A POOL
Pool
HOME
MY POOLS
MY CREWS
MY PROFILE
LOGOUT
Post a Pool
Pool Name
Pool Name
Warriors Game
Start Location
Street address
850 Russet Dr
Start city
Sunnyvale
Start zip code
94087
Start state
Required
End Location
Street address
Required
Pool

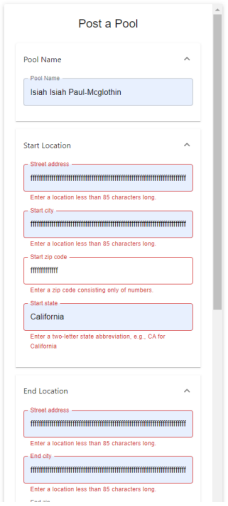

POST A POOL
JOIN A POOL
Pool
HOME
MY POOLS
MY CREWS
MY PROFILE
LOGOUT
Post a Pool
Pool Name
Pool Name
Warriors Game
Start Location
Street address
850 Russet Dr
Start city
Sunnyvale
Start zip code
94087
Start state
CA
End Location
Street address
Required
Pool

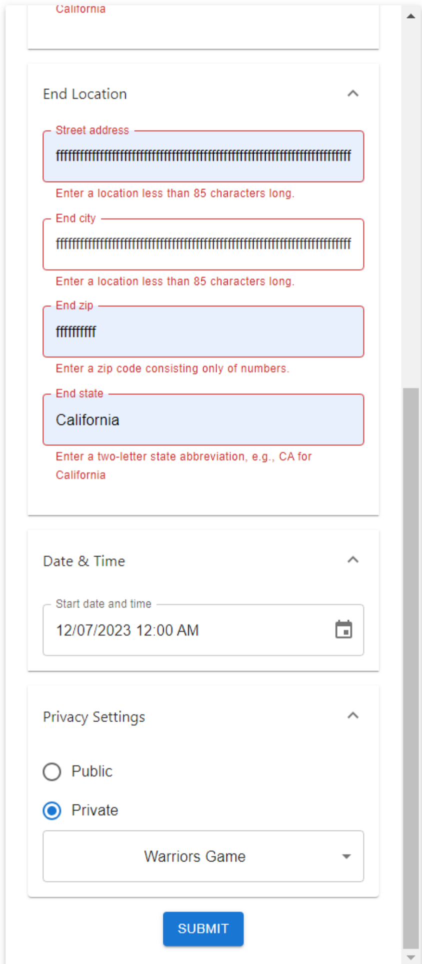

POST A POOL
JOIN A POOL
Pool
HOME
MY POOLS
MY CREWS
MY PROFILE
LOGOUT
Start zip code
94087
Start state
CA
End Location
Street address
1 Warriors Way
End city
Required
End zip
Required
End state
Required
Date & Time
Privacy Settings
SUBMIT
Pool

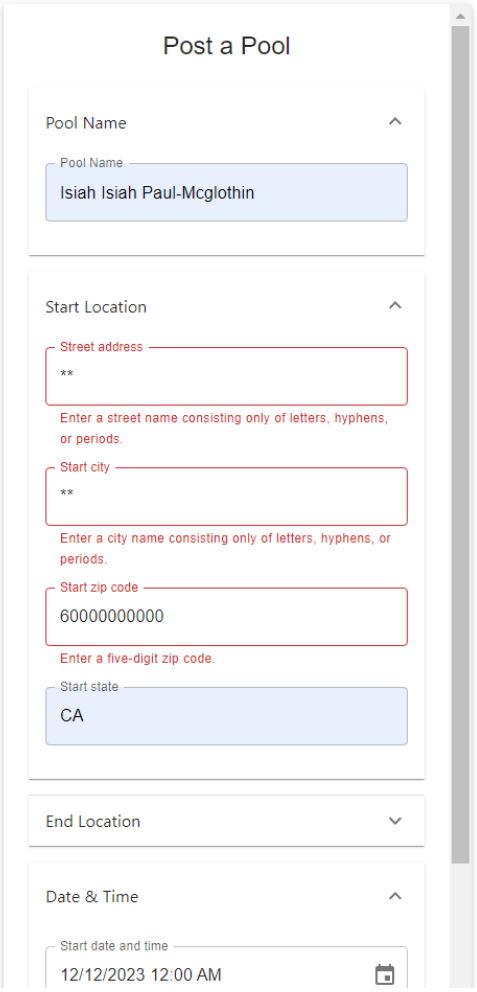

POST A POOL
JOIN A POOL
Pool
HOME
MY POOLS
MY CREWS
MY PROFILE
LOGOUT
Start zip code
94087
Start state
CA
End Location
Street address
1 Warriors Way
End city
San Francisco
End zip
Required
End state
Required
Date & Time
Privacy Settings
SUBMIT
Pool

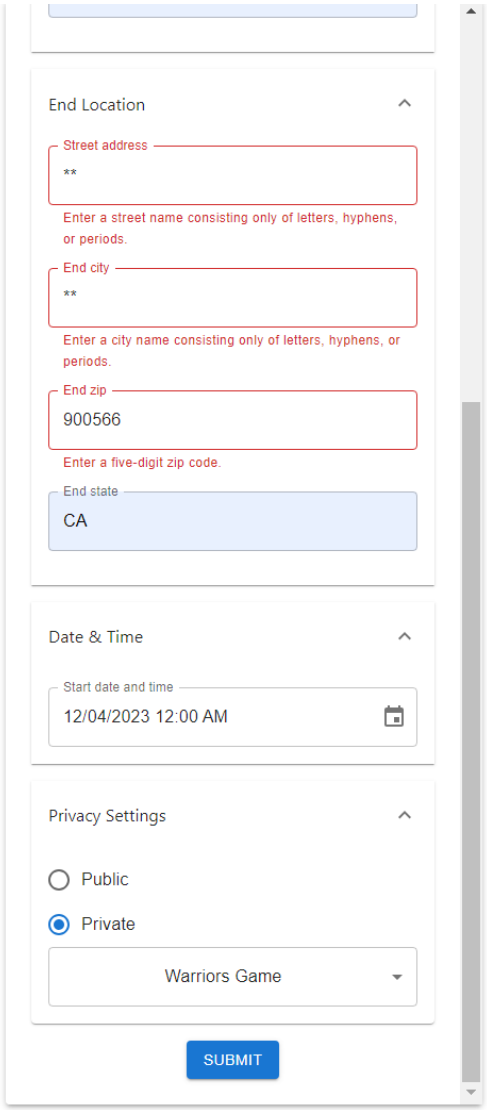

POST A POOL
JOIN A POOL
Pool
HOME
MY POOLS
MY CREWS
MY PROFILE
LOGOUT
Sunnyvale
Start zip code
94087
Start state
CA
End Location
Street address
1 Warriors Way
End city
San Francisco
End zip
94158
End state
Required
Date & Time
Privacy Settings
SUBMIT
Pool


POST A POOL
JOIN A POOL
Pool
HOME
MY POOLS
MY CREWS
MY PROFILE
LOGOUT
Start city
Sunnyvale
Start zip code
94087
Start state
CA
End Location
Street address
1 Warriors Way
End city
San Francisco
End zip
94158
End state
CA
Date & Time
Privacy Settings
SUBMIT
Pool


Post a Pool
Pool Name
Isiah Isiah Paul-Mcglothin
Start Location
Enter a location less than 85 characters long.
Enter a location less than 85 characters long.
Enter a zip code consisting only of numbers.
California
Enter a two-letter state abbreviation, e.g., CA for California
End Location
Enter a location less than 85 characters long.
Enter a location less than 85 characters long.


California
End Location
Street address
Enter a location less than 85 characters long.
End city
Enter a location less than 85 characters long.
End zip
Enter a zip code consisting only of numbers.
End state
California
Enter a two-letter state abbreviation, e.g., CA for California
Date & Time
Start date and time
12/07/2023 12:00 AM
Privacy Settings
Public
Private
Warriors Game
SUBMIT

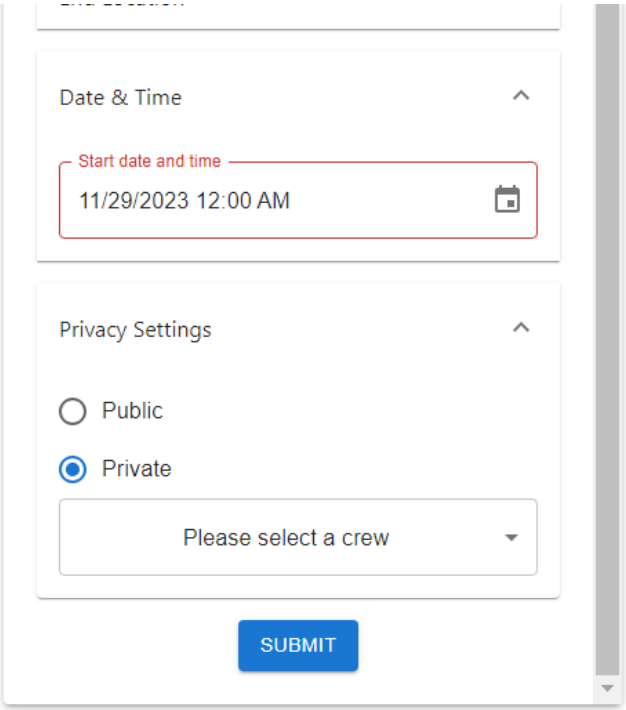

Post a Pool
Pool Name
Pool Name
Isiah Isiah Paul-Mcglothin
Start Location
Street address
**
Enter a street name consisting only of letters, hyphens, or periods.
Start city
**
Enter a city name consisting only of letters, hyphens, or periods.
Start zip code
60000000000
Enter a five-digit zip code.
Start state
CA
End Location
Date & Time
Start date and time
12/12/2023 12:00 AM

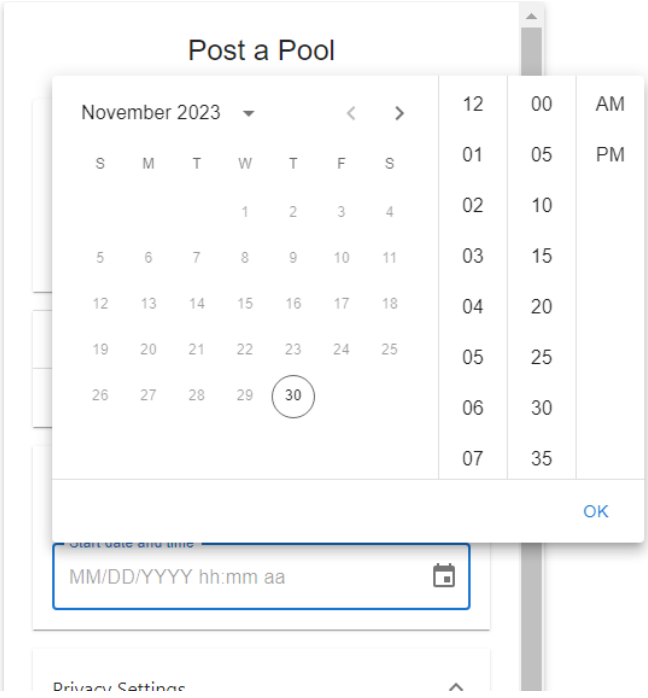

End Location
Street address
**
Enter a street name consisting only of letters, hyphens, or periods.
End city
**
Enter a city name consisting only of letters, hyphens, or periods.
End zip
900566
Enter a five-digit zip code.
End state
CA
Date & Time
Start date and time
12/04/2023 12:00 AM
Privacy Settings
Public
Private
Warriors Game
SUBMIT

```
//field validations
if (!validateName(name)) {
  setNameError("Required");
} else {
  setNameError(null);
}

if (!startStreet) {
  setStartStreetError("Required");
} else if (!validateStreetName(startStreet)) {
  setStartStreetError(
    "Enter a street name consisting only of letters, hyphens, or periods."
  );
} else if (!validateStreetAddress(startStreet)) {
  setStartStreetError("Enter a location less than 85 characters long.");
} else {
  setStartStreetError(null);
}
// Start City Validation

if (!startCity) {
  setStartCityError("Required");
} else if (!validateCityName(startCity)) {
  setStartCityError(
    "Enter a city name consisting only of letters, hyphens, or periods."
  );
} else if (!validateLength(startCity, maxLength)) {
  setStartCityError("Enter a location less than 85 characters long.");
} else {
  setStartCityError(null);
}

if (!startState) {
  setStartStateError("Required");
} else if (!validateState(startState)) {
  setStartStateError(
    "Enter a two-letter state abbreviation, e.g., CA for California"
  );
} else {
  setStartStateError(null);
}

if (!endStreet) {
  setEndStreetError("Required");
} else if (!validateStreetName(endStreet)) {
  setEndStreetError(
    "Enter a street name consisting only of letters, hyphens, or periods."
  );
} else if (!validateStreetAddress(endStreet)) {
  setEndStreetError("Enter a location less than 85 characters long.");
} else {
  setEndStreetError(null);
}

// End City Validation
if (!endCity) {
  setEndCityError("Required");
} else if (!validateCityName(endCity)) {
  setEndCityError(
    "Enter a city name consisting only of letters, hyphens, or periods."
  );
} else if (!validateLength(endCity, maxLength)) {
  setEndCityError("Enter a location less than 85 characters long.");
} else {
  setEndCityError(null);
}
```

```
  if (!endState) {
    setEndStateError("Required");
  } else if (!validateState(endState)) {
    setEndStateError(
      "Enter a two-letter state abbreviation, e.g., CA for California"
    );
    return;
  } else {
    setEndStateError(null);
  }

  // Start Zip Validation
  if (!startZip) {
    setStartZipError("Required");
  } else if (!/^\d+$/.test(startZip)) {
    setStartZipError("Enter a zip code consisting only of numbers.");
  } else if (startZip.length !== 5) {
    setStartZipError("Enter a five-digit zip code.");
  } else {
    setStartZipError(null);
  }

  // End Zip Validation
  if (!endZip) {
    setEndZipError("Required");
  } else if (!/^\d+$/.test(endZip)) {
    setEndZipError("Enter a zip code consisting only of numbers.");
  } else if (endZip.length !== 5) {
    setEndZipError("Enter a five-digit zip code.");
  } else {
    setEndZipError(null);
  }

  return;
};
```

Frontend validation for pool date and time

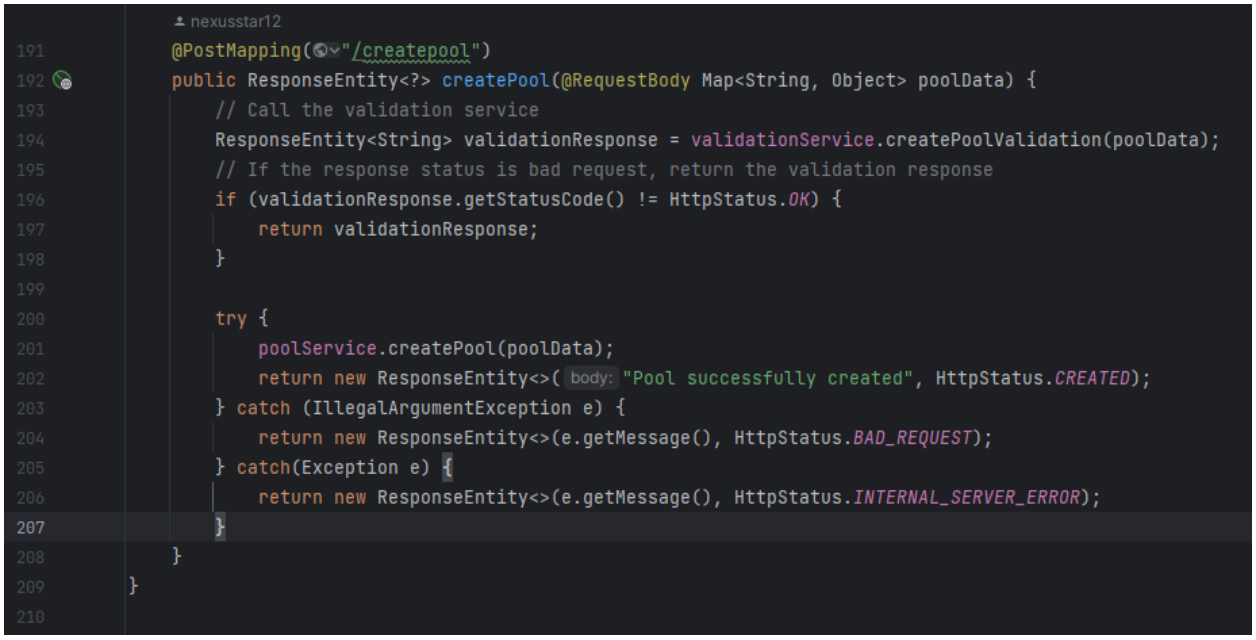




```
    if (newValue && dayjs().isAfter(newValue)) {
      setDateError("Time can not be in the past.");
    } else {
      setDateError(null);
    }
  };
```

```
        </Accordion>
        {/* Start time/date section */}
        <Accordion>
          <AccordionSummary expandIcon={<ExpandMoreIcon />}>
            Date & Time
          </AccordionSummary>
          <AccordionDetails>
            <LocalizationProvider dateAdapter={AdapterDayjs}>
              <DemoContainer components={["DateTimePicker"]}>
              <DateTimePicker
                  required
                  label="Start date and time"
                  value={selectedDate}
                  onChange={handleDateChange}
                  minDate={dayjs()}
                />
```

Backend validation for createpool api (calling from validation service):



```
    nexusstar12
191     @PostMapping("/createpool")
192     public ResponseEntity<?> createPool(@RequestBody Map<String, Object> poolData) {
193         // Call the validation service
194         ResponseEntity<String> validationResponse = validationService.createPoolValidation(poolData);
195         // If the response status is bad request, return the validation response
196         if (validationResponse.getStatusCode() != HttpStatus.OK) {
197             return validationResponse;
198         }
199
200         try {
201             poolService.createPool(poolData);
202             return new ResponseEntity<>( body: "Pool successfully created", HttpStatus.CREATED);
203         } catch (IllegalArgumentException e) {
204             return new ResponseEntity<>(e.getMessage(), HttpStatus.BAD_REQUEST);
205         } catch(Exception e) {
206             return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
207         }
208     }
209 }
210
```

6. Self-check: Adherence to original Non-functional specs

1. Scalability
   - 1.1. Pool shall use scalable storage in Google Cloud Platform. DONE
   - 1.2. Pool shall use load balancing on GCP as demand increases. DONE (approach and implementation plan documented here)
   - 1.3. Pool team shall monitor CPU usage and traffic on GCP and allocate resources as necessary. DONE

2. Reliability
   - 2.1. Pool shall be available to users 24/7 with minimal downtime. DONE (approach and implementation plan documented here)
   - 2.2. Pool shall use user input validation to ensure the integrity of data stored and retrieved. DONE
   - 2.3. Pool shall perform regular data backups in GCP to ensure no data loss. DONE

3. Regulatory
   - 3.1. Pool shall be in compliance with cookie and tracking regulations. DONE
   - 3.2. Pool shall comply with Web Content Accessibility Guidelines (WCAG) to ensure accessibility to individuals with disabilities. DONE (full compliance report here)

4. Maintainability
   - 4.1. Pool shall implement version control using GitHub to track changes to the codebase. DONE
   - 4.2. Pool shall use a GitHub flow branching strategy. DONE
   - 4.3. Pool shall use Apache Maven version 3.9.4 for backend dependency management. DONE
   - 4.4. Pool shall use the latest npm version 7.0 (Node package Manager) for the react.js front end. DONE

5. Serviceability
   - 5.1. Application backend is capable of switching between different data sources when it is required to call a backup data source. DONE
   - 5.2. Application has an intentional splash page for unexpected downtime. DONE
   - 5.3. Code deployments and maintenance will not require downtime. DONE

6. Utility
   - 6.1. Web Application will have search functionality to assist users. DONE
   - 6.2. The system will have a user-friendly interface for new and existing users. DONE

7. Manageability
   - 7.1. Alerting mechanisms should allow administrators to easily & quickly understand problems during critical events. DONE
   - 7.2. Administrators will have control of Pool's system through dev portals of Google Cloud and MySQL Workbench v 8.0.34. DONE

8. Data Integrity
   - 8.1. Passwords shall be encrypted. DONE
   - 8.2. Encryption and decryption shall be tested for accuracy. DONE

9. Capacity
   - 9.1. Pool shall specify the amount of data the application can handle DONE
     - 9.1.1. Auto Storage Increase is enabled in our Google Cloud MySQL database – meaning, if available storage falls below a specified threshold (calculated by Google based on storage currently available and defined upper limits, currently set to 3,054 GB), storage auto increases. More information may be found here.
   - 9.2. Pool shall specify maximum number of concurrent user sessions to support seamless user experience when in high traffic periods. DONE
     - 9.2.1. In its current pre launch phase, Pool does not expect (or support) more than 80 concurrent user sessions during its highest traffic periods (the lower limit default for GCP applications). If this upper limit is approached, this use case would justify the cost associated with implementing load balancing to better manage numbers at and above the current upper bounds. If high traffic periods persist, at this point Pool would begin conducting regular performance testing to better identify its limits and identify ways to increase performance during these periods.

10. Availability
    - 10.1. Application shall have a high level of system uptime DONE
    - 10.2. Application shall be available through various platforms and devices DONE

11. Usability

11.1. Users shall be able to easily navigate through Pool. DONE
11.2. Pool shall give error messages that are clear and helpful when wanting to resolve issue. DONE

12. Interoperability

12.1. Pool shall be able to integrate with map services to provide directions. ISSUE: related feature descoped to P3 functional requirement
12.2. Pool shall be compatible for different a range of vehicles. ISSUE: related feature descoped to P3 functional requirement

13. Privacy

13.1. Passwords shall be stored as hashes, ie. SHA256 hashing function. DONE
13.2. Pool shall have a response plan in the event of a data breach. DONE
13.3. User accounts shall require authentication for access. DONE
13.4. PII shall be handled with an added layer of care and protection. DONE
13.5. Requesting systems must have the required credentials in order to access application data. DONE

14. Coding Standards

14.1. Testing coverage tooling such as SonarQube will be implemented to expose areas of the codebase requiring test coverage. DONE
14.1.1. Read about our SonarQube implementation here.
14.2. Code shall be peer reviewed prior to being merged into the master branch. DONE

15. Networks

15.1. Pool shall be performant on low bandwidth networks. DONE
15.2. Pool shall leverage caching to increase performance during periods of reduced network availability. ISSUE: with permission from the Professor, caching was descoped due to the high level of complexity and cost associated with implementing it after initial analysis

16. Databases

16.1. Pool shall use database optimization techniques to increase performance. DONE (documented database optimization opportunities here)
16.2. User actions and associated data shall be preserved at the database level. DONE

17. Performance
    - 17.1. Application will have an average response time of 2 seconds or less. DONE
    - 17.2. Application utilizes efficient data storage techniques to reduce load time. DONE (documented database optimization opportunities here)
    - 17.3. Pool shall minimize device CPU usage. DONE
    - 17.4. Pool shall minimize device memory usage. DONE

18. Navigation & Wayfinding
    - 18.1. All open fields shall have sufficient validations. DONE
    - 18.2. All open validations shall have helpful error handling that help the user course correct. DONE
    - 18.3. Open fields shall auto suggest and when possible, pre-fill data to reduce the cognitive burden required to complete an action. ISSUE: Partially fulfilled by built-in browser pre-filling mechanisms, otherwise descoped due to cost associated with Google Maps API integration for address pre-filling.

19. Security
    - 19.1. Data shall be backed up regularly. DONE
    - 19.2. Data shall be sent over https protocol. DONE

20. Portability
    - 20.1. Application shall have cross platform compatibility across all actively supported iOS, MacOS, Windows OS, Linux OS, and Android OS devices. DONE
    - 20.2. Application shall function on the latest version of most common web browsers, i.e. Safari, Chrome, Firefox, DuckDuckGo. DONE

21. Cost & Budgeting
    - 21.1. Application operating entities shall maintain sufficient financial resources capable of maintaining and scaling core services including the database, servers, third party applications and software, etc. DONE
    - 21.2. Google Cloud should not have to incur any charges based on memory usage. DONE

22. Storage
    - 22.1. Tables should not exceed 400 bytes. DONE
    - 22.2. Values in tables should not exceed 50 bytes. DONE

23. Accessibility

23.1. Application frontend can accommodate users who may require a larger font size. DONE

23.2. Application frontend can be easily translated to any language. ISSUE: This nonfunctional requirement is partially fulfilled; while the app was tested using Google Translate and other browser-based translation tools, the app name “Pool” and commonly used in-app terms such as “carpool” do not translate well to other languages. (This was tested in Bulgarian, Spanish, and Vietnamese.)

23.3. Application frontend can be easily navigated by users without sight. DONE see screen reader test here

23.4. Application frontend is optimized for readability. DONE

23.5. Application signup should take no more than 10 clicks DONE

24. Environmental Impact

24.1. The application shall incentivize making full use of a car’s capacity to optimize environmental impacts. ISSUE: related feature (car entity) descoped to P3 functional requirement

24.2. The application shall have a feature that will remind drivers to turn off their engines if they’re waiting for long periods of time, like waiting for carpool participants. ISSUE: related feature (car entity) descoped to P3 functional requirement

24.3. The application shall give priority or recognition to drivers with hybrid or electric vehicles to promote environmental sustainability. ISSUE: related feature (car entity) descoped to P3 functional requirement

24.4. The application shall provide periodic emission saving reports to users to keep them motivated and informed. ISSUE: related feature (car entity) descoped to P3 functional requirement

Achieving High Availability for the Pool Application:

The Pool application is currently deployed on a single Ubuntu virtual machine on Google Compute Engine, where HTTPS is configured with SSL through “Let's Encrypt.” Having only one VM makes Pool vulnerable to outages and complicates upgrades and maintenance without causing downtime.

Our solution for making Pool more highly available involves adding a Google Load Balancer with HTTPS and Health Checks. These will direct traffic to two VMs on Compute

Engine. If one of the VMs goes down, the Health Checks will instruct the load balancer to route traffic only to the operational VM.

One challenge in making Pool highly available with this setup is transitioning the HTTPS logic from the single virtual machine to the load balancer without compromising our project. This may require extensive troubleshooting and configuration due to our team's limited expertise in this area. Additionally, financial constraints are a concern; the cost for services like extra virtual machines and a load balancer could potentially triple our current expenditure over our single VM setup. These factors make achieving high availability a future goal for the Pool team.

High-Level Implementation Steps:

1. **Create a Second VM:** Set up a second VM with settings identical to the original VM.
2. **Add Health Checks:** Implement Health Checks for the HTTP/HTTPS protocol, setting intervals and thresholds. This will inform the load balancer about any issues with the Pool VMs.
3. **Create an Instance Group:** Select the same region as the VMs and add both Pool VMs to this group.
4. **Configure the Load Balancer:**
    - Use HTTP(S) Load Balancing.
    - Create a backend service using the instance group and incorporate the earlier configured Health Checks.
    - Set up a frontend service to dictate how the load balancer handles incoming traffic. Select HTTPS as the protocol, add a Google-managed SSL certificate, and include Pool's domain (https://www.carpoolwith.me/).

SonarQube: Code Analysis Tool

Getting SonarQube Set up:

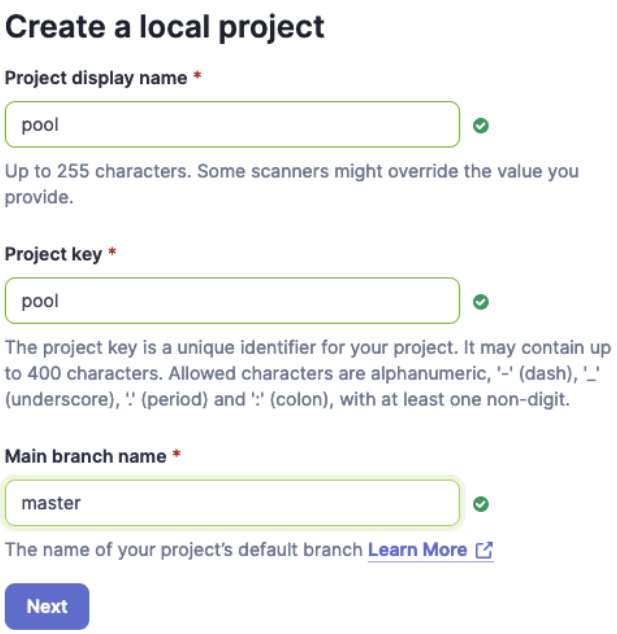
Get the official SonarQube Docker Image:
Run **docker pull sonarqube** the terminal.

Start the SonarQube docker container:
Run **docker run --name sonarqube --restart always -p 9000:9000 -d sonarqube**

Login:
Default username: admin password: admin

Change default username and password.

Create a local project (see next page for screenshot):

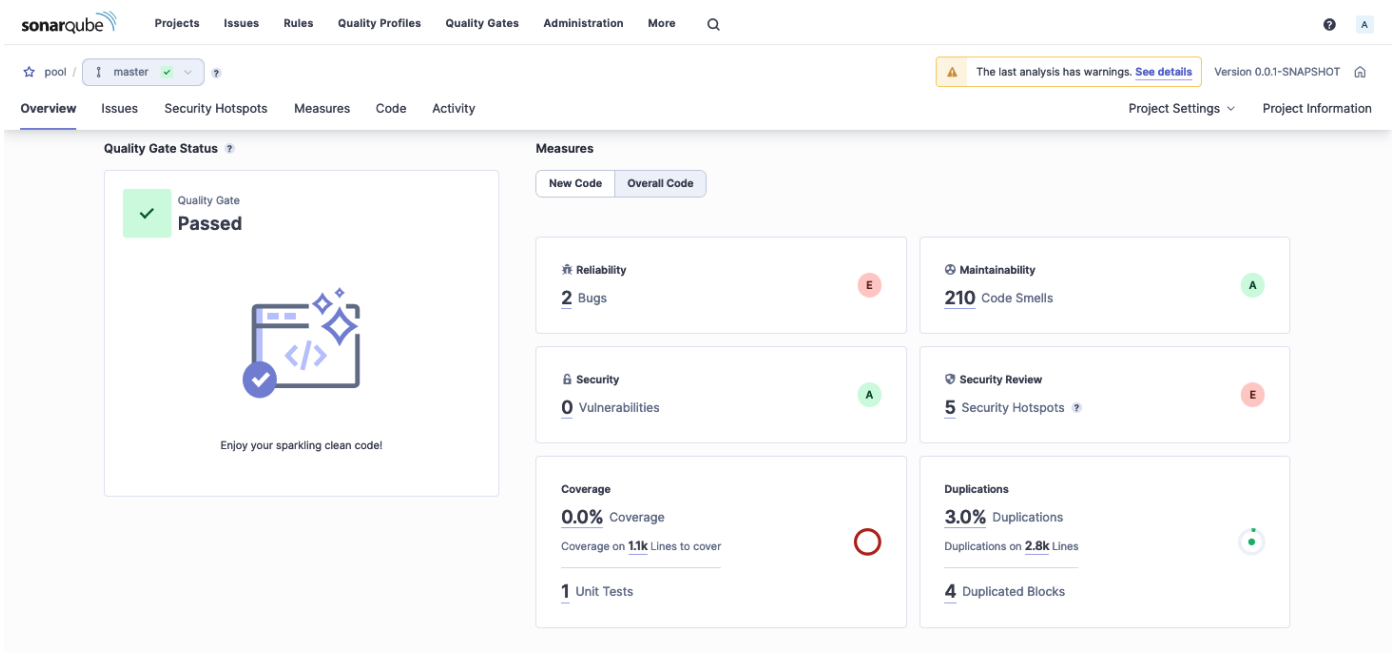


Create a SonarQube project token and store it somewhere safe.

Run analysis locally:
In the Intellij terminal navigate to the poolapp folder or the folder that contains the pom.xm and run: (SonarQube will provide you this code automatically when prompting the user, the SonarQube token will be different for each user).

```
mvn clean verify sonar:sonar \
  -Dsonar.projectKey=pool \
  -Dsonar.projectName='pool' \
  -Dsonar.host.url=http://localhost:9000 \
  -Dsonar.token=sqp_a66fbb4c752de8e70a23fe68717dce895589e2c8
```

Get analysis from the SonarQube application dashboard (see next page for screenshot):


sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
Administration
More
pool
master
The last analysis has warnings. See details
Version 0.0.1-SNAPSHOT
Overview
Issues
Security Hotspots
Measures
Code
Activity
Project Settings
Project Information
Quality Gate Status
Quality Gate
Passed
Enjoy your sparkling clean code!
Measures
New Code
Overall Code
Reliability
2 Bugs
E
Maintainability
210 Code Smells
A
Security
0 Vulnerabilities
A
Security Review
5 Security Hotspots
E
Coverage
0.0% Coverage
Coverage on 1.1k Lines to cover
1 Unit Tests
Duplications
3.0% Duplications
Duplications on 2.8k Lines
4 Duplicated Blocks

7. List of Contributions

Team Contribution Scores

- Isiah: 9
- Jonathan: 9
- Kendrick: 9
- Kristian: 10
- Phillip: 10
- Xuan: 10

Product Summary

- ✅Zoe: drafted initial versions and revised final version

Usability Test Plan & Test Moderation

- ✅Zoe: created and authored all elements of usability test plan and moderated usability test sessions

QA Test Plan & Testing

- ✅Kendrick: conducted QA testing for nonfunctional requirements 17.1 and 19.2
- ✅Kristian: developed test plan for nonfunctional requirement 8.1 and conducted QA testing for nonfunctional requirements 8.1, 17.1 and 19.2
- ✅Phillip: conducted QA testing for nonfunctional requirements 17.1 and 19.2
- ✅Zoe: created and authored all elements of usability test plan (with exception to that for nonfunctional requirement 8.1), moderated QA test sessions, and conducted QA for nonfunctional requirements 2.2 and 6.1

Code Review

- ✅Isiah: Conducted external code review for Jeremy Tran's team
- ✅Phillip: Conducted internal code review
- ✅Zoe: Analyzed and authored coding style section, coordination with Jeremy Tran's team for external code review

Self Check on Best Practices for Security

- ✅Kristian: Authored the entirety of this section

Input Validation

- ✅Isiah:
  - Required fields on “Sign up”
  - Length & Character validations on “Sign up”
  - Length & Character validations on “Post a pool”
  - Date & Time validations on “Post a pool”
- ✅Jonathan
  - Required fields on “Sign in”
  - Required fields on “Post a pool”
  - Length & Character validations on “Post a pool”
- ✅Kendrick
  - Zip code input in search bar
  - City input in search bar
  - Updated inline error messaging for phone and email on “Sign up”
- ✅Kristian
  - Backend validations for phone number received by pool/signup API
  - Backend validations for name fields received by pool/signup API
  - Backend validations for city received by pool/search API
  - Backend validations for zip code received by pool/search API
  - Backend validations for required fields received by pool/signup API
- ✅Phillip
  - Backend validations for required fields received by pool/signin API
  - Backend validations for required fields received by pool/createpool API
  - Backend length & character validations for inputs received by pool/createpool API
  - Backend validations for date and time received by pool/createpool API
- ✅Xuan
  - Assisted Kendrick with the above validations
- ✅Zoe
  - Wrote requirements for all validations and conducted testing

Self-check: Adherence to original Non-functional specs

- ✅Isiah: conducted self-check review on scalability, reliability, regulatory, and maintainability categories
- ✅Jonathan: conducted self-check review on serviceability, utility, manageability, and data integrity categories
- ✅Kendrick: conducted self-check review on capacity, availability, usability, and interoperability categories
- ✅Kristian: conducted self-check review on privacy, coding standards, networks, and databases categories
- ✅Phillip: conducted self-check review on performance, navigation & wayfinding, security, and portability categories
- ✅Xuan: conducted self-check review on cost & budgeting, storage, accessibility, and environmental impact categories

Nonfunctional Requirement Development & Implementation

- ✅Isiah:
  - created and implemented error page
- ✅Jonathan:
  - authored the entirety of the database optimization opportunities document following a thorough audit of our data types and usage
  - Tested application in DuckDuckGo and logged any found defects
- ✅Kendrick:
  - created and implemented terms and conditions page
  - Tested application in Safari and logged any found defects
  - Added terms and conditions agreement to signup
- ✅Kristian:
  - authored the entirety of the approach and implementation plan for load balancing and high availability
  - implemented SonarQube and documented the implementation and use
  - tested application in Firefox and logged any found defects
  - Set up event monitoring and alerts in Google Cloud Platform
  - Created and published internal data breach plan in GitHub repo
- ✅Phillip:

    - configured automated data backups
    - implemented ability for backend to switch to a backup CloudSQL db instance
    - ensure backend is sending proper error response to frontend for 400s and 500s
    - Test application in Chrome and logged any found defects
- ✅Xuan:
    - implemented cookie consent management
    - Made application fully screen size responsive
    - Convert header menu to hamburger menu on mobile web
- ✅Zoe:
    - Conducted accessibility compliance audit to produce final report using Accessibee
    - authored explanation of capacity nonfunctional requirement approaches

Bug Fixes

- ✅Isiah:
    - App name & tag line missing in mobile view
    - Date/time in My Pools still unreadable in military time
- ✅Jonathan:
    - Send poolId when calling create crew endpoint
- ✅Kendrick:
    - Bottom of terms & conditions screen cut off in full resolution
    - Display specific error messaging for invalid email
- ✅Kristian:
    - Supported all debugging requiring backend assistance
- ✅Phillip:
    - Supported all debugging requiring backend assistance
- ✅Xuan:
    - Nothing happens when user tries to create account in Safari
    - Search results displayed too close to bottom of screen
    - Search result display issue in select resolution
    - Fix pool creation
    - Resolve issue with frontend request to create crew endpoint
    - Pass crewId as null or INT when creating pool

    - Debug signup
    - Fix public/private post a pool feature
    - Debug create pool issue

Frontend Improvements & Updates

- ✅Isiah:
    - search result page card enhancements
- ✅Jonathan:
    - Create site footer
- ✅Xuan:
    - add confirmation for successful account creation
    - Display “crew created” on past pools when crewCreated=1
    - Convert date:time to UTC before sending to backend
    - Direct to sign up from “join pool” for unauthenticated users

Backend Improvements & Updates

- ✅Jonathan:
    - Update pools to be privacy=true or false

Deployment

- ✅Xuan: fully oversees deployment pipeline and activities

3 - Team Member Contributions

- **Isiah:** 10

Isiah is largely responsible for the look and feel of Pool; from the user journey through the pool and crew creation process to the images and logo, Isiah's various creative outputs played a big role in making Pool the site it is today. In the early stages, his vision helped inform the product and UX direction as we crafted user stories and mockups, and his frontend contributions implemented many of the features he brought to life through earlier artifacts.

Signature confirms that the team member is in agreement with their score.

- **Jonathan:** 10

From the database, to the backend to the frontend, Jonathan's code contributions span the entire Pool codebase (plus, he came up with the app name). Jonathan's role in architecting the database informed what grew into a class and function structure powering much of the app's core functionality. He has served not only as a critical and diverse contributor but also as a key consultant to both backend and frontend teams given his in depth knowledge of the database.

Signature confirms that the team member is in agreement with their score.

- **Kendrick**: 8

Kendrick's frontend contributions ensure that only valid user input is passed from the frontend to the backend (inline error messaging and validation triggers), that end users know their privacy rights (terms and conditions page), and that people can learn more about the team behind Pool (about page). Additionally, he helped develop early diagrams and artifacts and documented instructions for local app setup on Mac OS.

Signature confirms that the team member is in agreement with their score.

- **Kristian:** 10

Kristian has been essential to the success of our team from day one; he set up our initial server and deploy configurations, built our first iteration of the frontend, and then dived into the backend. Kristian has not only been a core contributor but also a fearless learner, which speaks to his various contributions in the greenfield phase. Kristian did much of the early research which informed our tech stack, filled in his own knowledge gaps, and then shared that knowledge with the team throughout the development process.

Signature confirms that the team member is in agreement with their score.

- **Phillip:** 10

Phillip has served as the team's primary backend contributor from Pool's conception to the present. He worked independently and collaboratively with Jonathan and Kristian to implement Spring Boot JDBC, build APIs, develop controllers and repositories corresponding to core functions, and coordinated integrated testing with the frontend team. He is a happy and cheerful collaborator and teammate, and has made himself available at all hours for debugging or helping – even in areas he is not the most familiar.

Signature confirms that the team member is in agreement with their score.

- **Xuan:** 10

Nearly all of the core frontend functionality was developed and implemented by Xuan, who served as our committed frontend lead. Xuan went from 0 to 100 on the frontend and is fully responsible for the site being screen responsive, for configuring all outbound API calls and inbound responses, for integrating with React Material UI which powers most of the site's frontend components, for debugging and resolving defects at an incredibly fast pace, and so much more. Additionally, Xuan serves as the Devops Manager and oversees the entire build and deploy process.

Signature confirms that the team member is in agreement with their score.

4 - Post Analysis - Lessons Learned

Team04 worked incredibly well together and had the fortunate composition of seven members, each compassionate, empathetic, eager to learn, unafraid of making mistakes, and wanting to do well in the course. Some key elements to our working style were: pair programming, staying connected on Discord, keeping front matter visible as pinned items, and meeting regularly (very regularly – sometimes twice a day, minimally every other day). The team got accustomed to highly collaborative work early on and really embraced digital tools for this such as Mural, Google docs, and Zoom.

Learning how to work with tickets (and Jira as a platform) was a big learning curve for most of the team; navigating the UI was challenging for first time users, and it took some team members way too long to learn that a ticket was clickable and contained more than a description. I deeply regret not spending more time facilitating hands-on tutorials for ticket creation and for following/using tickets as an engineer, which would have been a bigger investment up front but would have paid for itself very quickly.

The team was generally very good at meeting attendance but totally lacked tools for communicating when conflicts arose. Creating and sharing an open framework for communicating known and unforeseen schedule conflicts would have likely been very effective had I introduced it early on – the team takes very well to learning expected norms through role modeling and defined rules, and I think these communication skills could have benefited them in the industry, as well.

The top thing that would have made all development much smoother is creating better documentation and guides earlier on for best practices for branch hygiene (pulling down before developing) and commits and pull requests (especially cadence). The team also expressed that they think they would have benefited from this earlier on and been able to ramp up faster with it.