

Pool - carpooling app

SW Engineering CSC648-848-05 Fall 2023

Milestone 4 | 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](#) (page 3)
2. [Usability Test Plan](#) (page 5)
 - a. [Usability Test Results](#) (page 5)
 - b. [Function 1: Create a crew](#) (page 7)
 - c. [Function 2: Post a private pool to your crew](#) (page 10)
 - d. [Function 3: Join a pool posted privately to your crew](#) (page 13)
 - e. [Function 4: View your crews](#) (page 16)
 - f. [Function 5: Leave a crew](#) (page 19)
3. [QA Test Plan](#) (page 22)
 - a. [Nonfunctional Requirement 2.2: Pool shall use user input validation to ensure the integrity of data stored and retrieved.](#) (page 22)
 - b. [Nonfunctional Requirement 17.1: Application will have an average response time of 2 seconds or less.](#) (page 26)
 - c. [Nonfunctional Requirement 19.2: Data shall be sent over https protocol.](#) (page 30)
 - d. [Nonfunctional Requirement 6.1: Web Application will have search functionality to assist users.](#) (page 33)
 - e. [Nonfunctional Requirement 8.1: Passwords shall be encrypted.](#) (page 36)
4. [Code Review](#) (page 36)
 - a. [Coding Style](#) (page 36)
 - b. [Choose the Code for Review](#) (page 36)
 - c. [Internal Code Review](#) (page 47)
 - d. [External Code Review](#) (page 48)
5. [Self Check on Best Practices for Security](#) (page 51)
 - a. [Major Protected Assets](#) (page 49)
 - b. [Password Encryption](#) (page 49)
 - c. [Input Validation](#) (page 51)
 - i. [Home \(carpoolwith.me\)](#) (page 51)
 1. [Frontend Home Screen - search bar input validation for zip code and city](#) (page 51)
 2. [Backend Home screen - search bar input validations for zip code and city](#) (page 56)
 - ii. [Sign up screen \(carpoolwith.me/signup\)](#) (page 57)
 1. [Frontend validations \(first name, last name, phone number, email, password\)](#) (page 57)
 2. [Backend validation for signup API](#) (page 63)
 - iii. [Sign in screen \(carpoolwith.me/signin\)](#) (page 65)

-
1. [Backend validations - sign in \(check if user exists\)](#) (page 65)
 2. [Frontend Validations - Sign in \(validate email and password\)](#) (page 66)
 - iv. [Post a Pool \(carpoolwith.me/create-pool\)](#) (page 66)
 1. [Backend validations for pool description, address, and privacy settings](#) (page 66)
 2. [Frontend Validations: Pool Description, address, and privacy settings](#) (page 69)
 3. [Frontend validation for pool date and time](#) (page 88)
 4. [Backend validation for createpool API \(calling from validation service\)](#) (page 90)
 6. [Self-check: Adherence to original Non-functional specs](#) (page 91)
 7. [List of Contributions](#) (page 99)

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 <i>minimal</i> 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.					<input checked="" type="checkbox"/>
I like the look and feel of the user interface for creating a crew.			<input checked="" type="checkbox"/>		
I would create a crew if I was a Pool user.					<input checked="" type="checkbox"/>

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 <i>minimal</i> 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.					<input checked="" type="checkbox"/>
I like the look and feel of the user interface for posting a private pool to my crew.		<input checked="" type="checkbox"/>			
I would post a private pool to my crew if I was a Pool user.					<input checked="" type="checkbox"/>

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 belonging 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 <i>minimal</i> 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.	<input checked="" type="checkbox"/>				
I like the look and feel of the user interface for joining a pool posted privately to my crew.		<input checked="" type="checkbox"/>			
I would join a pool posted privately to my crew if I was a Pool user.					<input checked="" type="checkbox"/>

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 <i>minimal</i> 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.					<input checked="" type="checkbox"/>
I like the look and feel of the user interface for viewing my crews.		<input checked="" type="checkbox"/>			
I would view my crews if I was a Pool user.					<input checked="" type="checkbox"/>

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 <i>minimal</i> 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.					<input checked="" type="checkbox"/>
I like the look and feel of the user interface for leaving a crew.				<input checked="" type="checkbox"/> (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.			<input checked="" type="checkbox"/> (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: Firefox	P/F: Duck	P/F: Duck	
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 message showing below every input other than										

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

	click “Post a Pool” in the head er 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.

T e s t N u m b e r	S et u p	D e s c r i p t i o n	T e s t I n p u t	E x p e c t e d O u t p u t	P/F: Chro me 	P/F: Chro me 	P/F : Saf ari 	P/F : Saf ari 	P/F: Firef ox 	P/F: Firef ox 	P/F: Firef ox 	P/ F: Du ck 	P/ F: Du ck
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 zipcode in the dropdown, enter a zipcode 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	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	P

	<u>with. me/</u>	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: <u>https ://car pool with. me/</u>	Test to see if the response for signing in is 2 seconds or less	In the home page, 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

			(JJo nthe trac k@g mail. com ,											

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	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	P

			site has https.									
3	Go to: https://carpoolwithme/	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 and select the security tab and then should validate that it is secure	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 <u>carpool with.me</u>	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	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	<u>Password Hash:</u> \$2a\$10\$3QhEsBRh3f8LupWS0HqQkOclSy/DYJTgx6YKMoTLsPybqRDIwJEiq	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	<u>Hash #1:</u> \$2a\$10\$1LWorToGCqqmjGWQZIpuYuD7.BhB9XMdKt0Tj2GTVLaFL6pIB.Lpy <u>Hash #2:</u> \$2a\$10\$ZCosaPzS7n4M/lscry8Xey9NEG W9tMyE8A.lqDkASgg NIJ9Q1lzm	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.

```
application > poolapp > src > main > java > com > fantasticfour > poolapp > controller > J CrewController.java > CrewController.java
1  /**
2   * CrewController: is part of the pool REST api that handles logic for CREWS.
3   * FYI: A crew is a group of people that can schedule rides with one another.
4   */
5  package com.fantasticfour.poolapp.controller;
6
7  import com.fantasticfour.poolapp.CustomResponse.CrewListResponse;
8  import com.fantasticfour.poolapp.CustomResponse.CrewResponse;
9  import com.fantasticfour.poolapp.domain.Crew;
10 import com.fantasticfour.poolapp.domain.Pool;
11 import com.fantasticfour.poolapp.domain.Profile;
12 import com.fantasticfour.poolapp.repository.PoolRepository;
13 import com.fantasticfour.poolapp.repository.ProfileRepository;
14 import com.fantasticfour.poolapp.services.CrewService;
15 import com.fantasticfour.poolapp.repository.CrewRepository;
16 import com.fasterxml.jackson.databind.ObjectMapper;
17 import com.fasterxml.jackson.databind.SerializationFeature;
18 import org.springframework.beans.factory.annotation.Autowired;
19 import org.springframework.data.jpa.repository.support.SimpleJpaRepository;
20 import org.springframework.http.HttpStatus;
21 import org.springframework.http.ResponseEntity;
22 import org.springframework.web.bind.annotation.*;
23
24 import java.util.ArrayList;
25 import java.util.List;
26 import java.util.Map;
27 import java.util.Optional;
28 import java.util.stream.Collectors;
29
30
31 @RestController
32 @RequestMapping("/api/crew")
33 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>> get CrewById(@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.get CrewByProfileId(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.getCreator());
```

```

175     //set pool created to 0 after creator is deleted. because crew is deleted when creator deletes a crew.
176     Optional<Pool> optionalPool = poolRepository.findById(crew.getOriginPoolId());
177     if (optionalPool.isPresent()) {
178         Pool originalPool = optionalPool.get();
179         originalPool.setCrewCreated(false);
180         poolRepository.save(originalPool);
181     }else {
182         System.out.println("origin pool id does not exist to toggle crew_created field");
183     }
184
185     return new ResponseEntity<>("Crew deleted", HttpStatus.OK);
186 }
187
188 /**
189 * Deleting crew members based on profile ID
190 */
191 if(crew.getMember1() != null && crew.getMember1().getProfileId() == profileId){
192     crew.setMember1(null);
193     isDeleted = true;
194 }
195 if(crew.getMember2() != null && crew.getMember2().getProfileId() == profileId){
196     crew.setMember2(null);
197     isDeleted = true;
198 }
199 if(crew.getMember3() != null && crew.getMember3().getProfileId() == profileId){
200     crew.setMember3(null);
201     isDeleted = true;
202 }
203 if(crew.getCreator() != null && crew.getCreator().getProfileId() == profileId){
204     crew.setCreator(null);
205     isDeleted = true;
206 }
207
208 if(isDeleted){
209     crewRepository.save(crew);

```

```
213     }  
214     return new ResponseEntity<>("Profile with id" + profileId + "not found", HttpStatus.NOT_FOUND);  
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(carpool 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
285         profileExists = true;
286     }
287     if(pool.getMember3() != null){
288         member3_id = pool.getMember1();
289         crew.setMember3(member3_id);
290         profileExists = true;
291     }
292     crew.setDescription(pool.getDescription());
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:



Sign up

First Name*

Last Name*

Phone Number*

Email Address*

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](#)

```

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
717	duyanh12015tm@gmail.com	\$2a\$10\$UqOvvT79mwcIu5qzvovj/zn.rQKtKnmpdaowZM/tQINXU/IMZnqZs8xy
718	duyanh120115tm@gmail.com	\$2a\$10\$2HiisGvhVFB8/oU8mgyXz.RfZksqbJXFeg9L/KA40ELcRzojkpe
719	lyla@yopmail.com	\$2a\$10\$izD1LknnO.6cP4KTejmAjucbfAmcJ62gOn.US3.gw0PoLuXdADLDS
720	lala@yonmail.com	\$2a\$10\$OHI KvluGoN8DR5K.IH1uPn..lvfA5e.M3vkeDOHV/Ev0wwv0sC4EoqG
721	ENCtest1@email.com	\$2a\$10\$1LWorToGCqqmjGWQZpuYuD7.BhB9XMdKt0Tj2GTVLaFL6pIB.Lpy
722	ENCtest2@email.com	\$2a\$10\$zC0saPZS/n4M/lsdcry6A8y9iEGWV9imyE8A.iqDKASggNIJ9QnIZm

Result 6

Input Validation

[Home \(carpoolwith.me\)](#)

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



The screenshot shows the top navigation bar with "JOIN A POOL", "Pool", and "HOME SIGNUP LOGIN". Below the navigation is a large image of a smiling man in a car, fastening his seat belt. To his right, the text "Get there, together." is displayed. Below the image is a search form with two input fields: "Enter an end location zip code" containing "tthfas" and "Ending Zipcode". A blue "SEARCH" button with a magnifying glass icon is positioned below the fields. A red error message at the bottom states: "Invalid zip code. The valid zip code should only be 5 valid numbers."



The screenshot shows the same layout as the first one, but the search form has been modified. The "Enter an end location zip code" field now contains "santa bar6535" and the "City" dropdown menu is open, showing a list of suggestions. The "SEARCH" button remains blue and visible. A red error message at the bottom states: "Enter a city name consisting only of letters with 85 or fewer characters".

[JOIN A POOL](#)
Pool
HOME SIGNUP LOGIN



Get there, together.

Find a carpool in your area.

Starting Zipcode ▾

SEARCH 🔍

Invalid zip code. The valid zip code should only be 5 valid numbers.

```

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 validation, 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;
    }

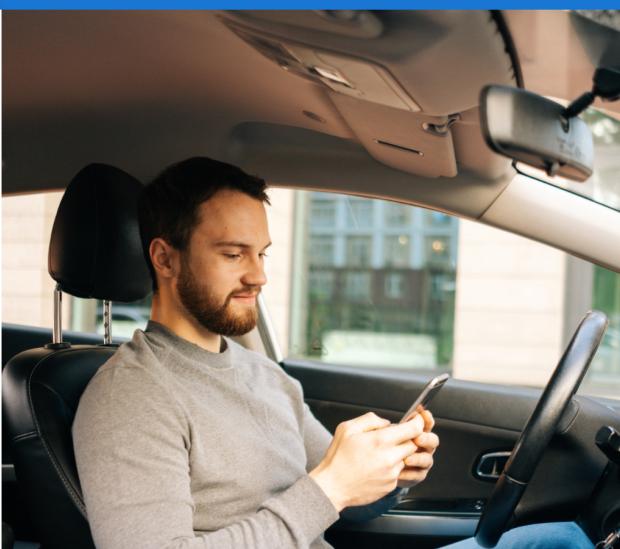
    return ZipCodeValidationEnum.VALID_ZIP;
}
```

Sign up screen (carpoolwith.me/signup)

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



The screenshot shows the 'Sign up' page of the Pool app. At the top, there are tabs for 'JOIN A POOL' and 'Pool'. On the right, there are links for 'HOME', 'SIGNUP', and 'LOGIN'. The main form has fields for 'First Name*', 'Last Name*', 'Phone Number*', 'Email Address*', and 'Password*'. The 'Phone Number*' field contains '9203948595965' and has a red border, indicating an error. Below the form is a checkbox for 'Would you like to register as a driver?' and a blue 'SIGN UP' button. A note below the button states: 'By signing up, you agree to Pool's [terms and conditions](#).'



This screenshot shows the same 'Sign up' page as the previous one, but with different validation errors. The 'Email Address*' field now contains 'k.com' and has a red border, indicating an error. A red note below the field states: 'Email must be valid, i.e. 'example@email.com''. The rest of the form and background are identical to the first screenshot.



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 *

888888888888888888888888

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 *

XXXXXXXXXXXXXX

Enter a name less than 51 characters long.

Last Name *

XXXXXXXXXXXXXX

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 (!isValidPhoneNumber(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);
}
```

```
1  export function isPhoneNumberValid(input) {
2    //check if it has 10 characters
3    console.log("aaaa", !/\^\\d{10}$.test(input));
4    if (!/\^\\d{10}$.test(input)) {
5      return "Phone number must be ten digits.";
6    }
7
8    //check if input isn't null
9    if (input === null) {
10      return;
11    }
12    You, yesterday • Uncommitted changes
13
14    //checking for list validation
15    if (input === "1234567890") {
16      return "Phone number can't be 1234567890";
17    }
18
19    //checking for repeated characters
20    if (/^(\\d)\\1{9}$.test(input)) {
21      return "Phone number can not consist of ten identical numbers.";
22    }
23
24    //first or 4th cant be 0 or 1
25    if (
26      input[0] === "0" ||
27      input[0] === "1" ||
28      input[3] === "0" ||
29      input[3] === "1"
30    ) {
31      return "First or fourth digit can not be 0 or 1";
32    }
33    return;
34  }
35 }
```

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

```
const emailRegex = /^[^\w\.-]+\.( [^\w-]+\.)+[\w-]{2,4}$/; You, i
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_OF_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)

```
75     //check if user exists by email
76     User user;
77     if (optionalUser.isPresent()){
78         user = optionalUser.get();
79     } else {
80         return new ResponseEntity<>(body: "Incorrect username or password", HttpStatus.UNAUTHORIZED);
81     }
82
83     Optional<Account> optionalAccount = accountRepository.findAccountByUserId(user.getUserId());
84
85     //Check if account exists for the user
86     Account account;
87     if (optionalAccount.isPresent()) {
88         account = optionalAccount.get();
89     } else {
90         return new ResponseEntity<>(body: "Could not find account", HttpStatus.UNAUTHORIZED);
91     }
92
93     //check if password matches
94     boolean validPassword = isPasswordNull(jsonMap.get("password"));
95     if(!validPassword){
96         return new ResponseEntity<>(body: "Password is empty or null", HttpStatus.UNAUTHORIZED);
97     }
98
99     String inputPassword = jsonMap.get("password");
100    String storedPassword = account.getPassword().getPassword(); //hashed password
101    boolean isPasswordMatching = passwordEncoder.matches(inputPassword, storedPassword);
102
103    if (isPasswordMatching) {
104        // Check if user is a driver
105        boolean isDriver = driverRepository.findByUser_UserId(user.getUserId()).isPresent();
106        user.setIsDriver(isDriver);
107
108        Profile profile= profileRepository.findById(user);
```

Frontend Validations - Sign in (validate email and password)



Pool

HOME SIGNUP LOGIN

Sign in

Email Address *

Please Enter a Valid Email

Password *

SIGN IN

Don't have an account? [Sign Up](#)



Pool

HOME SIGNUP LOGIN

Sign in

Email Address *

ashleyJefferson199@gmail.com

Password *

Please Enter a Valid Password

SIGN IN

Don't have an account? [Sign Up](#)

```
appfrontend > src > components > JS SignIn.js >  SignIn
1o
19  const defaultTheme = createTheme();
20
21  export default function SignIn() {
22    const history = useNavigate();
23    const [error, setError] = useState(null);
24    const [emailError, setEmailError] = useState(null);
25    const [passwordError, setPasswordError] = useState(null);
26    const userContext = useContext(UserContext);
27    const emailRegex = /^[^\w-\.]+\@([^\w-]+\.)+[\w-]{2,4}\$/;
28
29    const handleSubmit = async (event) => {
30      event.preventDefault();
31      const data = new FormData(event.currentTarget);
32      const email = data.get("email");
33      const password = data.get("password");
34
35      const requestBody = { email, password };
36
37
38      if (!email || !emailRegex.test(email)) {
39        setEmailError("Please Enter a Valid Email");
40        return;
41      } else {
42        setEmailError(false);
43      }
44      if(!password){
45        setPasswordError("Please Enter a Valid Password");
46        return;
47      }
48      else{
49        setPasswordError(false);
50      }
51      try {
52        const response = await axiosInstance.post("/signin", requestBody);
53        if (response.status === 200) {
54          localStorage.setItem("userInfo", JSON.stringify(response.data));
55          userContext.setUserInfo(response.data);
56          history("/", { replace: true });
57        }
58      } catch (error) {
59        setError(error.response.data);
60      }
61    };
62  }
```

```
<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

```
Usage: mvn test -Dtest=TestValidation

34     public ResponseEntity<String> createPoolValidation(Map<String, Object> poolData) {
35         StringBuilder errors = new StringBuilder();
36
37         // Length validation for street address and city
38         validateLength( fieldName: "startStreet", poolData, maxLength: 85, errorMessage: "invalidEntry.streetAddressMustConsistOfFewerThan86Characters", errors);
39         validateLength( fieldName: "endStreet", poolData, maxLength: 85, errorMessage: "invalidEntry.streetAddressMustConsistOfFewerThan86Characters", errors);
40         validateLength( fieldName: "startCity", poolData, maxLength: 85, errorMessage: "invalidEntry.cityMustConsistOfFewerThan86Characters", errors);
41         validateLength( fieldName: "endCity", poolData, maxLength: 85, errorMessage: "invalidEntry.cityMustConsistOfFewerThan86Characters", errors);
42
43         // Length validation for zip code
44         validateLength( fieldName: "startZip", poolData, maxLength: 5, errorMessage: "invalidEntry.zipMustBeFiveCharacters", errors);
45         validateLength( fieldName: "endZip", poolData, maxLength: 5, errorMessage: "invalidEntry.zipMustBeFiveCharacters", errors);
46
47         // Length validation for state
48         validateLength( fieldName: "startState", poolData, maxLength: 2, errorMessage: "invalidEntry.stateMustBeTwoCharacters", errors);
49         validateLength( fieldName: "endState", poolData, maxLength: 2, errorMessage: "invalidEntry.stateMustBeTwoCharacters", errors);
50
51         // Character validation for street address
52         validateCharacters( fieldName: "startStreet", poolData, regex: "[a-zA-Z0-9-. ]+", errorMessage: "invalidEntry.streetAddressMustContainOnlyLettersHyphensPeriods", errors);
53         validateCharacters( fieldName: "endStreet", poolData, regex: "[a-zA-Z0-9-. ]+", errorMessage: "invalidEntry.streetAddressMustContainOnlyLettersHyphensPeriods", errors);
54
55
56         // Character validation for city
57         validateCharacters( fieldName: "startCity", poolData, regex: "[a-zA-Z-. ]+$", errorMessage: "invalidEntry.cityMustContainOnlyLettersHyphensPeriods", errors);
58         validateCharacters( fieldName: "endCity", poolData, regex: "[a-zA-Z-. ]+$", errorMessage: "invalidEntry.cityMustContainOnlyLettersHyphensPeriods", errors);
59
60         // Character validation for zip code
61         validateCharacters( fieldName: "startZip", poolData, regex: "[0-9]+$", errorMessage: "invalidEntry.zipMustBeNumeric", errors);
62         validateCharacters( fieldName: "endZip", poolData, regex: "[0-9]+$", errorMessage: "invalidEntry.zipMustBeNumeric", errors);
63
64         // Character validation for state
65         validateCharacters( fieldName: "startState", poolData, regex: "[a-zA-Z]+$", errorMessage: "invalidEntry.stateMustBeAlphabetic", errors);
66         validateCharacters( fieldName: "endState", poolData, regex: "[a-zA-Z]+$", errorMessage: "invalidEntry.stateMustBeAlphabetic", errors);
67
68         // Name or description validation
69         if (poolData.get("name") == null || ((String) poolData.get("name")).isBlank()) {
70             errors.append("error.descriptionCannotBeNull ");
71         }
72     }
```

```
73     // startStreet validation
74     if (poolData.get("startStreet") == null || ((String) poolData.get("startStreet")).isBlank()) {
75         errors.append("error.startStreetCannotBeNull ");
76     }
77
78     // startCity validation
79     if (poolData.get("startCity") == null || ((String) poolData.get("startCity")).isBlank()) {
80         errors.append("error.startCityCannotBeNull ");
81     }
82
83     // startZip validation
84     if (poolData.get("startZip") == null || ((String) poolData.get("startZip")).isBlank()) {
85         errors.append("error.startZipCannotBeNull ");
86     }
87
88     // startState validation
89     if (poolData.get("startState") == null || ((String) poolData.get("startState")).isBlank()) {
90         errors.append("error.startStateCannotBeNull ");
91     }
92
93     // endStreet validation
94     if (poolData.get("endStreet") == null || ((String) poolData.get("endStreet")).isBlank()) {
95         errors.append("error.endStreetCannotBeNull ");
96     }
97
98     // endCity validation
99     if (poolData.get("endCity") == null || ((String) poolData.get("endCity")).isBlank()) {
100        errors.append("error.endCityCannotBeNull ");
101    }
102
103    // endZip validation
104    if (poolData.get("endZip") == null || ((String) poolData.get("endZip")).isBlank()) {
105        errors.append("error.endZipCannotBeNull ");
106    }
107
108    // endState validation
109    if (poolData.get("endState") == null || ((String) poolData.get("endState")).isBlank()) {
110        errors.append("error.endStateCannotBeNull ");
111    }
112
113    // Privacy validation
114    if (poolData.get("privacy") == null) {
115        errors.append("error.privacyCannotBeNull ");
116    }
```

```

118     // Character validation for street address to contain at least one letter
119     if (poolData.get("startStreet") != null && !containsLetter((String) poolData.get("startStreet"))) {
120         errors.append("error.startStreetMustContainAtLeastOneLetter ");
121     }
122     if (poolData.get("endStreet") != null && !containsLetter((String) poolData.get("endStreet"))) {
123         errors.append("error.endStreetMustContainAtLeastOneLetter ");
124     }
125
126     // Character validation for city to contain at least one letter
127     if (poolData.get("startCity") != null && !containsLetter((String) poolData.get("startCity"))) {
128         errors.append("error.startCityMustContainAtLeastOneLetter ");
129     }
130     if (poolData.get("endCity") != null && !containsLetter((String) poolData.get("endCity"))) {
131         errors.append("error.endCityMustContainAtLeastOneLetter ");
132     }
133
134     // Check for startDateTime and validate it's in the future
135     String startDateTimeStr = (String) poolData.get("startTime");
136     if (startDateTimeStr != null && !startDateTimeStr.isBlank()) {
137         try {
138             LocalDateTime startDateTime = LocalDateTime.parse(startDateTimeStr);
139             if (startDateTime.isBefore(LocalDateTime.now())) {
140                 errors.append("invalidEntry.dateTimeMustBeInTheFuture ");
141             }
142         } catch (DateTimeParseException e) {
143             errors.append("invalid.startTimeFormat ");
144         }
145     }
146
147     // Return any errors here
148     if (!errors.isEmpty()) {
149         return ResponseEntity.badRequest().body(errors.toString().trim());
150     }
151
152     // If all fields are valid
153     return ResponseEntity.ok( body: "All fields are valid");
154 }
```

```

156     // Check if a string contains at least one letter
157     @ > 4 usages  ± nexusstar12
158     private boolean containsLetter(String input) { return input.matches(regex ".*[a-zA-Z]+.*"); }
159
160     8 usages  ± nexusstar12
161     @ > private void validateLength(String fieldName, Map<String, Object> poolData, int maxLength, String errorMessage, StringBuilder errors) {
162         String fieldValue = (String) poolData.get(fieldName);
163         if (fieldValue != null && fieldValue.length() > maxLength) {
164             errors.append(errorMessage).append(" ");
165         }
166     }
167
168     8 usages  ± nexusstar12
169     @ > private void validateCharacters(String fieldName, Map<String, Object> poolData, String regex, String errorMessage, StringBuilder errors) {
170         String fieldValue = (String) poolData.get(fieldName);
171         if (fieldValue != null && !fieldValue.matches(regex)) {
172             errors.append(errorMessage).append(" ");
173         }
174     }
```

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
167     setEndStateError("Required");
168   } else {
169     setEndStateError(null);
170   }
171
172   return;
173 };
174   function validateName(name) {
175     let nameInput = name;
176     const namePattern = /[a-zA-Z0-9_ ]/;
177     return namePattern.test(nameInput);
178   }
179
180   function validateState(state) {
181     let stateInput = state;
182     const allStates = [
183       "AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DE", "FL", "GA", "HI", "ID", "IL", "IN", "IA", "KS", "KY",
184       "LA", "ME", "MD", "MA", "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ", "NM", "NY", "NC", "ND", "OH",
185       "OK", "OR", "PA", "RI", "SC", "SD", "TN", "TX", "UT", "VT", "VA", "WA", "WV", "WI", "WY",
186     ];
187
188     return allStates.includes(stateInput, 0);
189   }
190
191   function validateZipCode(zipCode) {
192     let zipCodeInput = zipCode;
193     if (zipCodeInput.length < 1 || zipCodeInput.length > 5) {
194       return false;
195     } else {
196       return true;
197     }
198   }
```

```
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>
```

```
/* 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

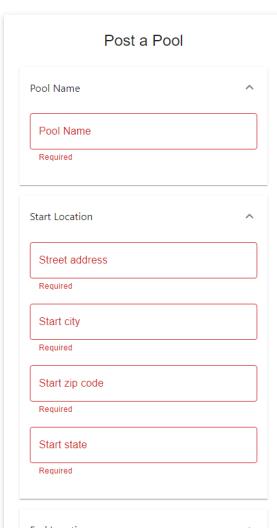
Start city
Required

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

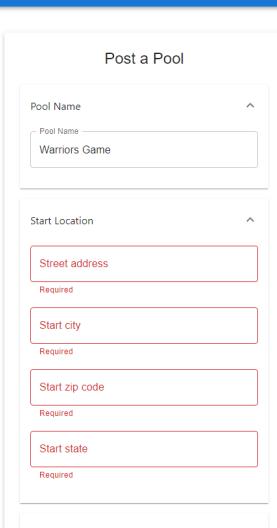
Start city
Required

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
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
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
94067

Start state
CA

End Location

Street address
1 Warriors Way

End city
San Francisco

End zip
94158

End state
CA

Date & Time

Privacy Settings

Pool

Post a Pool

Pool Name

Post Name
Isiah Isiah Paul-McGlothin

Start Location

Street address

Enter a location less than 85 characters long.

Start city

Enter a location less than 85 characters long.

Start zip code

Enter a zip code consisting only of numbers.

Start state

Enter a two-letter state abbreviation, e.g., CA for California

End Location

Street address

Enter a location less than 85 characters long.

End city

Enter a location less than 85 characters long.

California

End Location ^

Street address
ffff

Enter a location less than 85 characters long.

End city
ffff

Enter a location less than 85 characters long.

End zip
ffff

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 — 6000000000
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

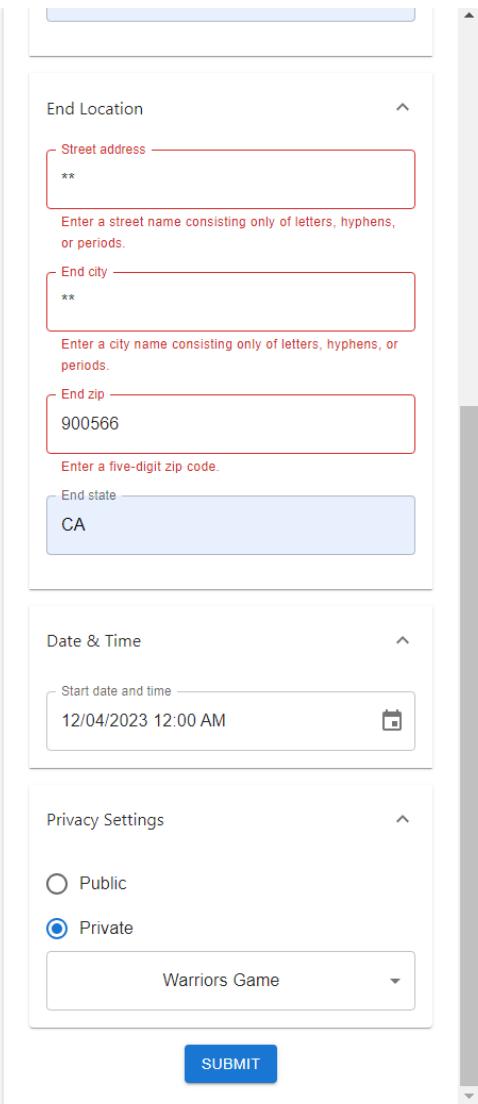
Date & Time

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

Date & Time

Start date and time

11/29/2023 12:00 AM

Privacy Settings

Public

Private

Please select a crew

SUBMIT

Post a Pool

November 2023							12	00	AM
S	M	T	W	T	F	S	01	05	PM
			1	2	3	4	02	10	
5	6	7	8	9	10	11	03	15	
12	13	14	15	16	17	18	04	20	
19	20	21	22	23	24	25	05	25	
26	27	28	29	30			06	30	
							07	35	

OK

Start date and time

MM/DD/YYYY hh:mm aa

Privacy Settings

```
    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()}
        />
      </DemoContainer>
    </LocalizationProvider>
  </AccordionDetails>
</Accordion>
```

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

```
191     @PostMapping(value="/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 local project

Project display name *

pool



Up to 255 characters. Some scanners might override the value you provide.

Project key *

pool



The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.

Main branch name *

master



The name of your project's default branch [Learn More](#)

Next

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.xml 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_a66fb4c752de8e70a23fe68717dce895589e2c8
```

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

SonarQube

Projects Issues Rules Quality Profiles Quality Gates Administration More Q

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	Maintainability
2 Bugs (E)	210 Code Smells (A)
Security	Security Review
0 Vulnerabilities (A)	5 Security Hotspots (E)
Coverage	Duplications
0.0% Coverage (C)	3.0% Duplications (C)
Coverage on 11k Lines to cover	Duplications on 2.8k Lines
1 Unit Tests	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