Test Report

There are two types of testing used for the application: manual end-to-end testing and API testing for the backend.

Manual End-to-end Testing

For the Manual End to end testing, various use cases were gone over to see if they meet the expected cases to see if they succeed. This is the most likely features to be used and are fully tested.

Use Cases		
Name	**Expected**	**What It is**
Scan QR	Scanning the QR code will bring the user to a page where they can click to join the queue of the charger linked to the QR code that they have scanned.	As expected.
Login	You go to the login page, and you type in your email, and it will send an email with verification.	As expected
Email Verification	An email is sent to the email address that was entered and you can then press sign in and reach the dashboard homepage	As expected
Reserve/Queue	You can do this in multiple ways. You get to the map view and click the specific charger on the map, or you can press the location on the dashboard. If there is no one in the queue you can reserve the space immidiately, otherwise you are placed in the queue.	As expected.
Leave Queue	To leave the queue you go to the queue on the menu bar which will tell you where you are in the queue. If you want to leave all the queues you press leave all. If you press leave, then you can select	As expected.

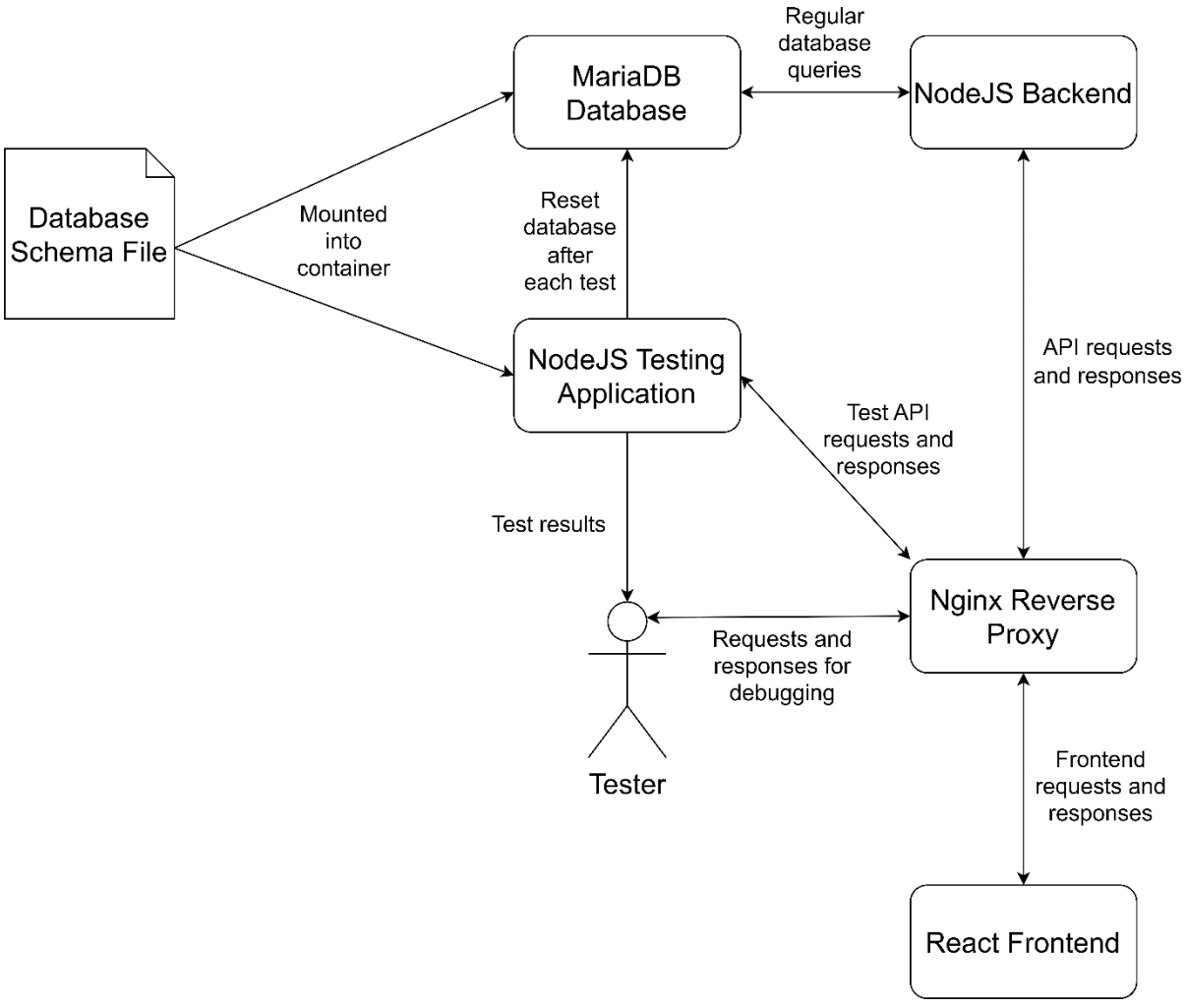
	the specific queues that you will leave.	
Check-in	You can press check in by pressing the queues and then pressing check in	As expected.
Check-out	You can press check out through the dashboard or the queues tab on the left.	As expected.
Log Out	You press profile and then select logout.	As expected.
Delete Account	You press profile and then select delete account and then press confirm.	As expected.
Report Charger	Similar to reserve or queuing you find a charger on the dashboard or map, select update charger status. You then select the given charger and select its new status. The admin will then go to the report section of the admin site and will see the report that they can read and accept or deny.	As expected.
Add Charger/ Delete Charger/ Edit Chargers	You go to Chargers and press add. You can give the location a name, the number of chargers located at it and its lattitude and longitude. You can also change the details of the location as well or delete individual chargers or the entire location (which will require confirmation).	As expected.
Add Admin	You go to the Admin menu and then add New Admin and type in the email address which will create and new email address.	As expected.

Backend API Testing

Methodology

API testing was done using a custom testing application to make API requests and verify the subsequent state of the application. The testing configuration for the application uses a Dockerized version of the system, with a replica MariaDB database in a Docker container. The custom testing application is another NodeJS server that runs inside of its own Docker container. Although the automated tests do not test the frontend React application, this is also configured to run with the test system, since it can be useful to have when debugging failed tests. All five containers required for testing can be run from a single Docker compose file, located in /Testing/UnitTesting/docker-compose.yaml.

When the test application is started, it attempts to make a connection to the backend, and will keep retrying until it does so successfully. After making an initial connection, the system automatically imports all the exported functions in all Javascript files in the "tests" directory, each of which it considers to be a separate test. It then resets the database using SQL commands that stored in a file which are mounted into the container. Once the database has been reset, it starts iterating through the tests, running them one by one, resetting the database after each of them. After all of the tests are complete, it compiles a text-based summary of the test results and makes these available at an exposed endpoint (by default, this is port 3003).



Each test consists of multiple sub-tests, each of which contains a request that changes the state of the application (e.g. join-queue, leave-queue, check-in) and an assertion. An assertion allows the testing application to evaluate whether the application is in the expected state or not. Assertions can be chained together using a "compound assertion" so that multiple things can be checked. For example, we may run a join-queue request to have user 1 join a queue for location 5, then assert that user 1 has status "WAITING", and that they are in position 1 in queue for location 5. The system then automatically evaluates both assertions that have been chained together (one for user data and another for queue data) to check if the sub-test has passed. Then we may have another sub-test for leaving the queue, which also must be successful for the overall test to pass. A summary of the result is also generated, including a title for each sub-test and the specific result of each assertion (expected data and actual data).

The testing application also takes a JSON web token secret. This is the same secret that is passed to the API backend, as this allows the testing application to create valid JSON web tokens, so that we can test the security of our backend. For example, one of the tests is to ensure that the backend will not verify an expired JSON web token, which is done by generating a JSON web token that expires in one second, waiting one second then asserting that the HTTP response status is 401: Unauthorized.

The tests implemented verify the functionality of most of the API calls (primarily excluding the email system). The tests could be expanded on in future to involve more complex queuing scenarios, such as more combinations of joining and leaving queues, checking in and out of charging points, and cancelling reservations.

Test Results

The test results are shown below in tables, with each table being a test and each row being a sub-test. Each table has three columns:

1. Action: The action that the testing application makes for a sub-test. Generally, one action corresponds to one API call.
2. Assertions: The list of assertions for the sub-test. All assertions must be successful for the sub-test to pass. This column is part of the output of the testing application.
3. Description: A short description of each of the sub-tests.

Tests Passed: 19/19

Test joining and leaving queue: PASS

Action	Assertions	Description
Test user 1 joining queue 5: PASS	**Assert user 1 data:** Success Expected user data: {"status":"WAITING"} Actual user data: {"status":"WAITING"} **Assert queue data of user 1:** Success	User with ID 1 joins queue 5, then the test asserts that the user has the "WAITING" status, and that the queue data returned shows user 1 in position 1 of the queue

	Expected queue data: [{"position":"1","locationID":5,"name":"Test Location 5","wattage":"50"}] Actual queue data: [{"position":"1","locationID":5,"name":"Test Location 5","wattage":"50"}]	
Test user 1 leaving queue 5: PASS	**Assert user 1 data:** Success Expected user data: {"status":"IDLE"} Actual user data: {"status":"IDLE"} **Assert queue data of user 1:** Success Expected queue data: Empty Actual queue data: Empty	User with ID 1 leaves queue 5, then the test asserts that the user has the "IDLE" status, and that the queue is now empty

Test filling a location: PASS

Action	Assertions	Description
Test user 1 joining queue 4: PASS	**Assert user 1 data:** Success Expected user data: {"status":" PENDING"} Actual user data: {"status":" PENDING"} **Assert charger 13 data:** Success Expected charger data: {"locationID":4,"status":"RESERVED"} Actual charger data: {"locationID":4,"status":"RESERVED"} **Assert queue data of user 1:** Success Expected queue data: Empty Actual queue data: Empty	User with ID 1 joins queue 4, then the test asserts that the user goes directly into the "PENDING" status, since there are charging points available. It then asserts that the corresponding charging point is set to "RESERVED" and that the queue is empty.
Test user 2 joining queue 4: PASS	**Assert user 2 data:** Success Expected user data: {"status":"PENDING"} Actual user data: {"status":"PENDING"} **Assert charger 14 data:** Success Expected charger data: {"locationID":4,"status":"RESERVED"} Actual charger data: {"locationID":4,"status":"RESERVED"}	User with ID 2 joins queue 4, then the test asserts that the user goes directly into the "PENDING" status, since there are charging points available. It then asserts that the corresponding charging point is set to "RESERVED" and that the queue is empty.

	Assert queue data of user 2: Success Expected queue data: Empty Actual queue data: Empty	
Test user 3 joining queue 4: PASS	**Assert user 3 data:** Success Expected user data: {"status":"WAITING"} Actual user data: {"status":"WAITING"} **Assert queue data of user 3:** Success Expected queue data: [{"position":"1","locationID":4,"name":"Test Location 4","wattage":"20"}] Actual queue data: [{"position":"1","locationID":4,"name":"Test Location 4","wattage":"20"}]	User with ID 3 joins queue 4, then the test asserts that the user has the "WAITING" status, and that the queue data returned shows user 3 in position 1 of the queue
Test user 4 joining queue 4: PASS	**Assert user 4 data:** Success Expected user data: {"status":"WAITING"} Actual user data: {"status":"WAITING"} **Assert queue data of user 4:** Success Expected queue data: [{"position":"2","locationID":4,"name":"Test Location 4","wattage":"20"}] Actual queue data: [{"position":"2","locationID":4,"name":"Test Location 4","wattage":"20"}]	User with ID 4 joins queue 4, then the test asserts that the user has the "WAITING" status, and that the queue data returned shows user 4 in position 2 of the queue

Test checking in and out: PASS

Action	**Assertions**	**Description**
Test user 1 joining queue 4: PASS	**Assert user 1 data:** Success Expected user data: {"status":" PENDING"} Actual user data: {"status":" PENDING"} **Assert charger 13 data:** Success Expected charger data: {"locationID":4,"status":"RESERVED"} Actual charger data: {"locationID":4,"status":"RESERVED"}	User with ID 1 joins queue 4, then the test asserts that the user goes directly into the "PENDING" status, since there are charging points available. It then asserts that the corresponding charging point is set to "RESERVED" and that the queue is empty.

	Assert queue data of user 1: Success Expected queue data: Empty Actual queue data: Empty	
Test user 1 checking in: PASS	**Assert user 1 data:** Success Expected user data: {"status":"CHARGING"} Actual user data: {"status":"CHARGING"} **Assert charger 13 data:** Success Expected charger data: {"locationID":4,"status":"CHARGING"} Actual charger data: {"locationID":4,"status":"CHARGING"} **Assert queue data of user 1:** Success Expected queue data: Empty Actual queue data: Empty	User with ID 1 checks into the reserved charging point, then the test asserts that the user and the corresponding charger both enter the "CHARGING" state. It also asserts that the queue is empty.
Test user 1 checking out: PASS	**Assert user 1 data:** Success Expected user data: {"status":"IDLE"} Actual user data: {"status":"IDLE"} **Assert charger 13 data:** Success Expected charger data: {"locationID":4,"status":"IDLE"} Actual charger data: {"locationID":4,"status":"IDLE"} **Assert queue data of user 1:** Success Expected queue data: Empty Actual queue data: Empty	User with ID 1 checks out of the charging point, then the test asserts that the user and the corresponding charger both return to the "IDLE" state. It also asserts that the queue is empty.

Action	**Expected Data**	**Actual Data**	**Result**
User 1 joining queue 4	user data: {"status":"PENDING"} charger data: {"locationID":4,"status":"RESERVED"} queue data: Empty	user data: {"status":"PENDING"} charger data: {"locationID":4,"status":"RESERVED"} queue data: Empty	PASS

User 2 joining queue 4	user data: {"status":"PENDING"} charger data: {"locationID":4,"status":"RESERVED"} queue data: Empty	user data: {"status":"PENDING"} charger data: {"locationID":4,"status":"RESERVED"} queue data: Empty	PASS
User 3 joining queue 4	user data: {"status":"WAITING"} queue data: [{"position":"1","locationID":4,"name":"Test Location 4","wattage":"20"}]	user data: {"status":"WAITING"} queue data: [{"position":"1","locationID":4,"name":"Test Location 4","wattage":"20"}]	PASS
User 4 joining queue 4	user data: {"status":"WAITING"} queue data: [{"position":"2","locationID":4,"name":"Test Location 4","wattage":"20"}]	user data: {"status":"WAITING"} queue data: [{"position":"2","locationID":4,"name":"Test Location 4","wattage":"20"}]	PASS

Test cancelling reservations: PASS

Action	Assertions	Description
Test user 1 joining queue 4: PASS	**Assert user 1 data:** Success Expected user data: {"status":" PENDING"} Actual user data: {"status":" PENDING"} **Assert charger 13 data:** Success Expected charger data: {"locationID":4,"status":"RESERVED"} Actual charger data: {"locationID":4,"status":"RESERVED"} **Assert queue data of user 1:** Success Expected queue data: Empty Actual queue data: Empty	User with ID 1 joins queue 4, then the test asserts that the user goes directly into the "PENDING" status, since there are charging points available. It then asserts that the corresponding charging point is set to "RESERVED" and that the queue is empty.
Test user 1 cancelling reservation: PASS	**Assert user 1 data:** Success Expected user data: {"status":"WAITING"}	User with ID 1 cancels their reservation, then the test

	Actual user data: {"status":"WAITING"} **Assert charger 13 data:** Success Expected charger data: {"locationID":4,"status":"IDLE"} Actual charger data: {"locationID":4,"status":"IDLE"} **Assert queue data of user 1:** Success Expected queue data: Empty Actual queue data: Empty	asserts that the user has returned to the "WAITING" state and that the charger has returned to the "IDLE" state. It also asserts that the queue is empty

Test queue ordering when someone leaves: PASS

Action	Assertions	Description
Test user 1 joining queue 4: PASS	**Assert user 1 data:** Success Expected user data: {"status":" PENDING"} Actual user data: {"status":" PENDING"} **Assert charger 13 data:** Success Expected charger data: {"locationID":4,"status":"RESERVED"} Actual charger data: {"locationID":4,"status":"RESERVED"} **Assert queue data of user 1:** Success Expected queue data: Empty Actual queue data: Empty	User with ID 1 joins queue 4, then the test asserts that the user goes directly into the "PENDING" status, since there are charging points available. It then asserts that the corresponding charging point is set to "RESERVED" and that the queue is empty.
Test user 2 joining queue 4: PASS	**Assert user 2 data:** Success Expected user data: {"status":"PENDING"} Actual user data: {"status":"PENDING"} **Assert charger 14 data:** Success Expected charger data: {"locationID":4,"status":"RESERVED"} Actual charger data: {"locationID":4,"status":"RESERVED"} **Assert queue data of user 2:** Success Expected queue data: Empty	User with ID 2 joins queue 4, then the test asserts that the user goes directly into the "PENDING" status, since there are charging points available. It then asserts that the corresponding charging point is set to "RESERVED" and that the queue is empty.

	Actual queue data: Empty	
Test user 3 joining queue 4: PASS	**Assert user 3 data:** Success Expected user data: {"status":"WAITING"} Actual user data: {"status":"WAITING"} **Assert queue data of user 3:** Success Expected queue data: [{"position":"1","locationID":4,"name":"Test Location 4","wattage":"20"}] Actual queue data: [{"position":"1","locationID":4,"name":"Test Location 4","wattage":"20"}]	User with ID 3 joins queue 4, then the test asserts that the user has the "WAITING" status, and that the queue data returned shows user 3 in position 1 of the queue
Test user 4 joining queue 4: PASS	**Assert user 4 data:** Success Expected user data: {"status":"WAITING"} Actual user data: {"status":"WAITING"} **Assert queue data of user 4:** Success Expected queue data: [{"position":"2","locationID":4,"name":"Test Location 4","wattage":"20"}] Actual queue data: [{"position":"2","locationID":4,"name":"Test Location 4","wattage":"20"}]	User with ID 4 joins queue 4, then the test asserts that the user has the "WAITING" status, and that the queue data returned shows user 4 in position 2 of the queue
Test user 3 leaving queue 4: PASS	**Assert user 3 data:** Success Expected user data: {"status":"IDLE"} Actual user data: {"status":"IDLE"} **Assert queue data of user 3:** Success Expected queue data: Empty Actual queue data: Empty **Assert user 4 data:** Success Expected user data: {"status":"WAITING"} Actual user data: {"status":"WAITING"} **Assert queue data of user 4:** Success Expected queue data: [{"position":"1","locationID":4,"name":"Test Location 4","wattage":"20"}]	User with ID 3 leaves the queue for location 4, then the test asserts that the user has the "IDLE" status and that user 3 is not in any queues. It also asserts that user 4 is still in the "WAITING" state and that they have moved into position 1 in queue.

	Actual queue data: [{"position":"1","locationID":4,"name":"Test Location 4","wattage":"20"}]	
Test user 3 joining queue 4 again: PASS	**Assert user 3 data:** Success Expected user data: {"status":"WAITING"} Actual user data: {"status":"WAITING"} **Assert queue data of user 3:** Success Expected queue data: [{"position":"2","locationID":4,"name":"Test Location 4","wattage":"20"}] Actual queue data: [{"position":"2","locationID":4,"name":"Test Location 4","wattage":"20"}]	User with ID 3 joins queue 4 again, then the test asserts that the user has the "WAITING" status, and that the queue data returned shows user 3 in position 2 of the queue

User Status Tests: PASS

Action	Assertions	Description
Test user 1 status: PASS	**Assert user 1 data:** Success Expected user data: {"status":"IDLE"} Actual user data: {"status":"IDLE"}	Test that the user with ID 1 is in the "IDLE" state. This ensures that the get user data API call is functioning correctly.

Location Data Access Tests: PASS

Action	Assertions	Description
Test location 1 data access: PASS	**Assert location 1 data:** Success Expected location data: {"locationID":1,"name":"Test Location 1","wattage":"10","lat":"1","lng":"2"} Actual location data: {"locationID":1,"name":"Test Location 1","wattage":"10","lat":"1","lng":"2"}	Test that the get location data API call is functioning correctly.

Charger Data Access Tests: PASS

Action	Assertions	Description
Test charger 1 data access: PASS	**Assert charger 1 data:** Success Expected charger data: {"chargingPointID":1,"status":"IDLE","locationID":1} Actual charger data: {"chargingPointID":1,"status":"IDLE","locationID":1}	Test that the get charger data API call is functioning correctly.

Test updating a charger's status from a report: PASS

Action	Assertions	Description
Test reporting a charger as broken, which should log a report: PASS	**Assert report data:** Success Expected report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car!"}] Actual report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car!"}]	A user sends a report that a charger is broken, and the test asserts that the report has been logged successfully.
Test reporting a charger as in use, which should automatically update it's status: PASS	**Assert report data:** Success Expected report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car!"}] Actual report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car!"}] **Assert charger 2 data:** Success Expected charger data: {"chargingPointID":2,"locationID":1,"status":"CHARGING"} Actual charger data: {"chargingPointID":2,"locationID":1,"status":"CHARGING"}	A user sends a report that a charger is in use, and the test asserts that the report does not get logged (since the list of reports only returns the first report), and that the charger has been automatically updated to the “CHARGING” state.

Test deleting a report: PASS

Action	Assertions	Description
Test reporting a charger as broken, which should log a report: PASS	**Assert report data:** Success Expected report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car!"}]	A user sends a report that a charger is broken, and the test asserts that the

	Actual report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car!"}]	report has been logged successfully.
Test deleting a report: PASS	**Assert report data:** Success Expected report data: Empty Actual report data: Empty	An admin user deletes the report, and the test asserts that the list of reports is now empty.

Test getting report count: PASS

Action	Assertions	Description
Test reporting a charger as broken, which should log a report: PASS	**Assert report data:** Success Expected report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car 1!"}] Actual report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car 1!"}]	A user sends a report that a charger is broken, and the test asserts that the report has been logged successfully.
Test reporting a charger as broken, which should log a report: PASS	**Assert report data:** Success Expected report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car 1!"},{"reportID":2,"chargingPointID":3,"message":"It is not charging my car 2!"}] Actual report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car 1!"},{"reportID":2,"chargingPointID":3,"message":"It is not charging my car 2!"}]	An admin user deletes the report, and the test asserts that both reports have been logged successfully.
Test reporting a charger as broken, which should log a report: PASS	**Assert report data:** Success Expected report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car 1!"},{"reportID":2,"chargingPointID":3,"message":"It is not charging my car 2!"},{"reportID":3,"chargingPointID":5,"message":"It is not charging my car 3!"}] Actual report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car 1!"},{"reportID":2,"chargingPointID":3,"message":"It is not charging my car	An admin user deletes the report, and the test asserts that all three reports have been logged successfully.

	2!"},{"reportID":3,"chargingPointID":5,"message":"It is not charging my car 3!"}]	
Test that the report count is 3: PASS	**Assert report count:** Success Expected report count: 3 Actual report count: 3	Test the get report count API call successfully returns 3.

Test validating a report: PASS

Action	Assertions	Description
Test reporting a charger as broken, which should log a report: PASS	**Assert report data:** Success Expected report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car!"}] Actual report data: [{"reportID":1,"chargingPointID":1,"message":"It is not charging my car!"}] **Assert charger 1 data:** Success Expected charger data: {"locationID":1,"status":"IDLE"} Actual charger data: {"locationID":1,"status":"IDLE"}	A user sends a report that a charger is broken, and the test asserts that the report has been logged successfully.
Test validating the report, which should update the charger's state to broken: PASS	**Assert report data:** Success Expected report data: Empty Actual report data: Empty **Assert charger 1 data:** Success Expected charger data: {"locationID":1,"status":"BROKEN"} Actual charger data: {"locationID":1,"status":"BROKEN"}	An admin user validates the report, and then asserts that the charger status is “BROKEN” and that the report has been deleted.

Test setting user permission level: PASS

Action	Assertions	Description
Test setting user 1 to admin: PASS	**Assert admin user data:** Success Expected admin data: [{"email":"test-admin-1"},{"email":"test-admin-	An admin user sets test-user-1's permission level to “ADMIN”, then the test

	2"},{"email":"test-superadmin-1"},{"email":"test-user-1"}] Actual admin data: [{"email":"test-admin-1"},{"email":"test-admin-2"},{"email":"test-superadmin-1"},{"email":"test-user-1"}]	asserts that they are now in the list of admin users.
Test setting user 1 to user: PASS	**Assert admin user data:** Success Expected admin data: [{"email":"test-admin-1"},{"email":"test-admin-2"},{"email":"test-superadmin-1"}] Actual admin data: [{"email":"test-admin-1"},{"email":"test-admin-2"},{"email":"test-superadmin-1"}]	An admin user sets test-user-1's permission level to back to “USER", then the test asserts that they are no longer in the list of admin users.

Test clearing queue: PASS

Action	Assertions	Description
Test user 1 joining queue 5: PASS	**Assert user 1 data:** Success Expected user data: {"status":"WAITING"} Actual user data: {"status":"WAITING"} **Assert queue data of user 1:** Success Expected queue data: [{"position":"1","locationID":5,"name":"Test Location 5","wattage":"50"}] Actual queue data: [{"position":"1","locationID":5,"name":"Test Location 5","wattage":"50"}]	User with ID 1 joins queue 5, then the test asserts that the user goes into the “WAITING” status, and that they are in position 1 in the queue.
Test user 2 joining queue 5: PASS	**Assert user 2 data:** Success Expected user data: {"status":"WAITING"} Actual user data: {"status":"WAITING"} **Assert queue data of user 2:** Success Expected queue data: [{"position":"2","locationID":5,"name":"Test Location 5","wattage":"50"}] Actual queue data: [{"position":"2","locationID":5,"name":"Test Location 5","wattage":"50"}]	User with ID 2 joins queue 5, then the test asserts that the user goes into the “WAITING” status, and that they are in position 2 in the queue.

Test user 3 joining queue 5: PASS	**Assert user 3 data:** Success Expected user data: {"status":"WAITING"} Actual user data: {"status":"WAITING"} **Assert queue data of user 3:** Success Expected queue data: [{"position":"3","locationID":5,"name":"Test Location 5","wattage":"50"}] Actual queue data: [{"position":"3","locationID":5,"name":"Test Location 5","wattage":"50"}]	User with ID 3 joins queue 5, then the test asserts that the user goes into the "WAITING" status, and that they are in position 3 in the queue.
Test clearing queue: PASS	**Assert user 1 data:** Success Expected user data: {"status":"IDLE"} Actual user data: {"status":"IDLE"} **Assert queue data of user 1:** Success Expected queue data: Empty Actual queue data: Empty **Assert user 2 data:** Success Expected user data: {"status":"IDLE"} Actual user data: {"status":"IDLE"} **Assert queue data of user 2:** Success Expected queue data: Empty Actual queue data: Empty **Assert user 3 data:** Success Expected user data: {"status":"IDLE"} Actual user data: {"status":"IDLE"} **Assert queue data of user 3:** Success Expected queue data: Empty Actual queue data: Empty	An admin user clears the queue, then the test asserts that all three of the users are reset to the "IDLE" state and that the queue is now empty.

Test updating location: PASS

Action	Assertions	Description

Test adding new location: PASS	**Assert location 6 data:** Success Expected location data: {"locationID":6,"name":"Newly added location","wattage":"30","lat":"11","lng":"12"} Actual location data: {"locationID":6,"name":"Newly added location","wattage":"30","lat":"11","lng":"12"}	An admin user adds a new location, then the test asserts that the new location has been added correctly.
Test updating the new location: PASS	**Assert location 6 data:** Success Expected location data: {"locationID":6,"name":"Newly added location updated","wattage":"31","lat":"13","lng":"14"} Actual location data: {"locationID":6,"name":"Newly added location updated","wattage":"31","lat":"13","lng":"14"}	An admin user updates the location, then the test asserts that the location has been updated correctly.

Test updating charger: PASS

Action	Assertions	Description
Test adding new charger to location 5: PASS	**Assert charger 15 data:** Success Expected charger data: {"chargingPointID":15,"status":"IDLE","locationID":5} Actual charger data: {"chargingPointID":15,"status":"IDLE","locationID":5}	An admin user adds a new charger to location 5, then the test asserts that the new charger has been added correctly.
Test updating the new charger to be broken: PASS	**Assert charger 15 data:** Success Expected charger data: {"chargingPointID":15,"status":"BROKEN","locationID":5} Actual charger data: {"chargingPointID":15,"status":"BROKEN","locationID":5}	An admin user updates the charger to the “BROKEN” state, then the test asserts that the charger has been updated correctly.

Test deleting location: PASS

Action	Assertions	Description
Test adding new location: PASS	**Assert location 6 data:** Success Expected location data: {"locationID":6,"name":"Newly added location","wattage":"30","lat":"11","lng":"12"} Actual location data: {"locationID":6,"name":"Newly added location","wattage":"30","lat":"11","lng":"12"}	An admin user adds a new location, then the test asserts that the new location has been added correctly.
Test deleting the new location, then trying to access it: PASS	**Assert HTTP status:** Success Expected HTTP status: 404 Actual HTTP status: 404	An admin user deletes the new location, then the test asserts that the backend returns a "404: Not found" HTTP status, when trying to access it.

Test deleting charger: PASS

Action	Assertions	Description
Test adding new charger to location 5: PASS	**Assert charger 15 data:** Success Expected charger data: {"chargingPointID":15,"status":"IDLE","locationID":5} Actual charger data: {"chargingPointID":15,"status":"IDLE","locationID":5}	An admin user adds a new charger to location 5, then the test asserts that the new charger has been added correctly.
Test deleting the new charger, then trying to access it: PASS	**Assert charger 15 data:** Success Expected charger data: undefined Actual charger data: undefined	An admin user deletes the new location, then the test asserts that when the backend returns an array of chargers, it is not in the array (this means that when searching for the new charger in the array it returns undefined)

Test Login Verification: PASS

Action	Assertions	Description
Test user verification with valid JWT: PASS	**Assert HTTP status:** Success Expected HTTP status: 302 Actual HTTP status: 302	The testing application generates and sends a valid JWT, then the test asserts that the backend returns a “302: Found” HTTP status.
Test user verification with invalid JWT: PASS	**Assert HTTP status:** Success Expected HTTP status: 401 Actual HTTP status: 401	The testing application generates and sends an invalid JWT (the string “123”), then the test asserts that the backend returns a “401: Unauthorized” HTTP status.
Test user verification with expired JWT: PASS	**Assert HTTP status:** Success Expected HTTP status: 401 Actual HTTP status: 401	The testing application generates a JWT that expires in 1 second, waits for 1 second, then sends it to the backend. The then test asserts that the backend returns a “401: Unauthorized” HTTP status.

Usability Report

Meeting with first client

When we showed an earlier version of our prototype to our first client they stated that they thought the email verification made the system overly complicated. This was in large part down to the fact they assumed that they would have to use their university email, which was not on their phone and therefore they would need to run into the campus building to sign in on a computer. This is something which we could potentially make clearer in the UI, as since it has 'Heriot Watt University' on the login page some users may incorrectly assume that they have to use a university account.

The client also stated he would prefer if users could just enter their email address receive a notification when a charger is free rather than having to go through email verification or passwords. Although this would streamline the login process, as we are storing which chargers users are currently parked at it means that anyone with the users email could find the location of their car. Therefore, we must have a more rigorous log in process in order to protect users and in order to follow the UK governments 'Design System' guidance which states that the email confirmation loop should be used if "accidentally using the wrong email address would give someone else access to sensitive information about the user" [1].

This does still highlight one potential flaw in our assumptions for the project, in that it was assumed that all users would have an email address on their phone that they would be happy to provide to the site. In future we could make use of nodemailer's SMS service, to give users an option to sign up using either their email or phone number.

Meeting with second client

After completing the final prototype implementation, we interviewed our second client where the prototype was demonstrated to collect feedback on usability and the site as a whole. The clients feedback was overwhelmingly positive, and felt the user interface was 'clean and minimalistic' and stated that the doughnut charts on the dashboard helped clearly show charger status' at a glance. He also stated that the prototype provided a 'vast improvement' compared to the existing process of getting a charger.

After using the prototype we asked the client to fill out the System Usability Scale questionnaire in order to gather standardised feedback which we can use to compare the usability of our system to industry standards. The answers to this questionnaire can be seen below:

Question	**Answer (Strongly Disagree-Strongly Agree)**
I think that I would like to use this system frequently	Strongly Agree
I found the system unnecessarily complex	Strongly Disagree
I thought the system was easy to use	Strongly Agree
I think that I would need the support of a technical person to be able to use this system	Strongly Disagree
I found the various functions in this system were well integrated	Strongly Agree
I thought there was too much inconsistency in this system	Strongly Disagree
I would imagine that most people would learn to use this system very quickly	Agree
I found the system very cumbersome to use	Strongly Disagree
I felt very confident using the system	Agree
I needed to learn a lot of things before I could get going with this system	Strongly Disagree

The answers from our client gave us an overall system usability score of 95/100, which is very positive. The industry standard for the system usability score is that any score above 68 is considered above average and is a system with good usability. As our score of 95 greatly outperforms this, we can conclude that overall we have successfully created a application with good usability.

They suggested that it could be beneficial to add a time for when the status of a charger was last updated, which might help users feel more confidence regarding the reliability of the system as it is a self-reporting system. This could be a useful and easy to implement feature in future versions of the system.

They also suggested that they felt the map view could potentially be moved to be moved from the dashboard so it was on the main page, this contradicts the opinions of our first client who felt the map view was not a feature that many frequent users would use/need and therefore would be better on another page. As both our clients have different opinions on this matter, future research could be done in usability studies to evaluate this further.

He also highlighted a flaw in the site, that if you were already in a queue you could still click 'join queue'. Although this action had no effect on the site, instead it should stop users from being able to view the 'join queue' button.

References

[1] G. D. System, “Confirm an email address,” [Online]. Available: https://design-system.service.gov.uk/patterns/confirm-an-email-address/. [Accessed 20 03 2024].