# Chapter 1 – Introduction

The purpose of this chapter is to describe the different aspects of our testing plan, describe how every requirement we have is going to be tested and describe multiple scenarios for each test. Our test plan is as follows:

1. **Unit Testing** -We test each individual component in isolation from the other components in the system. We will write multiple test cases for each function of our API. We will use Mocks for objects that the component we are testing is dependent on.
2. **Integration Testing** – We test how multiple components of our system are working together. We’ll use the Bottom-Up approach where we will start with smaller components, and we’ll iteratively connect more components until the complete system will be tested.
3. **Acceptance Testing** – These tests will be conducted after the system has been developed and before it is release to the client. The primary goal of those tests is to ensure that the system meets the business requirements of the client and is ready for deployment.
4. **System Testing** – Our system tests will focus mostly on the Raspberry Pi before we release it. We will test the use cases where the Rider is the actor.

In these tests we will use the real system components (except the external systems) to make sure the Raspberry Pi application is working as expected.

The company should provide us with a road that contains all kinds of hazards, and we are expecting to get an alert for each one of them.

We will use all the different options for the Raspberry Pi’s Configuration.

We will also check the backend of our Admin and Rider API’s.

1. **Regression Testing** – We will use the GitHub CI/CD in order to verify new changes are not breaking the correct behavior of our system. We will run Integration Tests and Unit tests on every commit.
2. **GUI Testing** – We will perform tests that emulate the behavior of our users. Those tests are meant to make sure the system is working as expected End to End. We will use an external library to test our GUI (Cypress?)
3. **Load Testing** – We will perform load tests to make sure our system is able to withstand the number of concurrent operations that our client is expecting.
4. **Robustness Testing** – We will perform tests to see how well the system is recovering from errors such as: DB failures, External systems failures, GPS Failures etc…

# Chapter 2 – Functional Requirements Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Req  Number | Test Type | Test Purpose | Input | Expected Output | Status |
| 1 | Unit | Make sure our system can deserialize a JSON file properly to Configuration class | Valid JSON file that describes the configuration | Successful Deserialization to Configuration class |  |
| 2 | System + GUI | Make sure our system is working as expected with all available external systems we are working with | This test will be performed on all available External Systems | Our System Should continue to work as expected with all kinds of external services |  |
| 3 | Unit + System + GUI | Make sure our Registration is working as expected | Valid user information | A New user has been created |  |
|  | X | X | Invalid Email address | The operation wasn’t successful |  |
|  | X | X | Invalid password | The operation wasn’t successful |  |
|  | X | X | Invalid raspberry pi serial number | The operation wasn’t successful |  |
|  | X | X | Invalid birth date | The operation wasn’t successful |  |
|  | X | X | Invalid license issue date | The operation wasn’t successful |  |
|  | X | X | Invalid scooter type | The operation wasn’t successful |  |
| 4 | Unit + System + GUI | Make sure the user profile can be updated as expected | Valid user information | User profile is changed to display the new information |  |
|  | X | X | Invalid age | The operation wasn’t successful |  |
|  | X | X | Invalid scooter type | The operation wasn’t successful |  |
|  | X | X | Invalid driver’s license issue date | The operation wasn’t successful |  |
| 5 | Unit + System + GUI | Make sure users can login to the system | Valid user credentials | User is logged in |  |
|  | X | X | Incorrect email | The operation wasn’t successful |  |
|  | X | X | Incorrect password | The operation wasn’t successful |  |
| 6 | Unit + System + GUI | Make sure users can logout out of the system | User is logged in | User is logged out |  |
|  | X | X | User is not logged in | The operation wasn’t successful |  |
| 7 | Unit + System + GUI | Make sure users can send a message to the admins | information in the message is valid | A message was created and sent to the admins |  |
|  | X | X | Invalid title | The operation wasn’t successful |  |
|  | X | X | Invalid message body | The operation wasn’t successful |  |
| 8 | Integration + GUI | Make sure our system can fetch routes from the external maps service | Valid destination | A list of possible routes has been returned to the user |  |
|  | X | X | Invalid destination address | The operation wasn’t successful |  |
| 9 | GUI + Manual | Make sure our system can display instructions during the ride | A route from source to destination that we received from google maps | A map with instructions has been loaded |  |
| 10 | Unit | Make sure our Raspberry Pi can detect hazards | A photo without hazards (We will test it in multiple scenarios – Rainy, Sunny, Cloudy, etc.…) | Raspberry pi didn’t detect any hazards |  |
|  | X | X | A photo with hazards (We will test it in multiple scenarios – Rainy, Sunny, Cloudy, etc.…) | Raspberry pi detected the hazard |  |
|  | System | X | We will use a route with multiple hazards and drive through it | Raspberry pi detected the hazards |  |
|  | X | X | We will use a route without hazards and drive through it | Raspberry pi didn’t detect any hazards |  |
| 11 | System | Make sure raspberry pi can detect mobile hazards | We will drive in a road and insert mobile hazard mid-drive | Raspberry pi detected the mobile hazards |  |
| 12 | Unit | Make sure raspberry pi can detect the riding surface | A photo with some riding surface (sidewalk, highway, etc.…) | Raspberry pi correctly detected the riding surface |  |
|  | System | X | We will drive on some riding surface | Raspberry pi correctly detected the riding surface |  |
| 13 | System | Make sure that the system is storing information only about new hazards | We will go on a test drive and ride past the same hazard twice | A new hazard information has been recorded only for the first occurrence of the hazard |  |
| 14 | Unit + GUI + System | Make sure admins are notified when a user filled a riding experience questionnaire | Valid description of the experience | The information is stored and admins are notified about it |  |
|  | X | X | Invalid description | The operation wasn’t successful |  |
| 15 | Unit + Integration + System | Make sure the system can store information about new rides | Valid ride information | The information was stored in the system |  |
|  | X | X | Invalid rider id in the ride description | The information wasn’t saved |  |
|  | X | X | Invalid ride date | The information wasn’t saved |  |
| 16 | Unit + GUI + System | Make sure the system displays the rides history for the user and admins | None | The system should show the user all of his rides information (if the user did not perform any rides the system will display a message that says that) |  |
| 17 | Unit + System | Make sure the system updates the rating of the user based on a recent ride according to the specification we received from the client | The rider ignored hazard alerts and didn’t drove in a safely manner | The rider’s rating is lowered using the formulas provided in the ARD document |  |
|  | X | X | The rider reacted appropriately to alerts and drove in a safely manner | The rider’s rating is increased using the same formulas from above |  |
| 18 | GUI + System | Make sure our system’s delayed notifications mechanism is working properly | We will send multiple messages to a user (admin or rider) that is logged out and then the user is logged in | The user should receive a notification about each message that was sent to him while he was logged out |  |
| 19 | Unit + System + GUI | Make sure the system allows the **admins** to view the information of all users | An admin is trying to view an existing user information | Information is displayed to the admin |  |
|  | X | X | An admin is trying to view information of a user that doesn’t exit | The information is not showed to the admin |  |
|  | X | X | A regular rider tried to view some else’s information | The information is not showed to the rider |  |
| 20 | Unit + GUI + System | Make sure the system allows **admins** to delete users from the system | An admin is trying to delete an existing user | The user account is deleted |  |
|  | X | X | An admin is trying to delete a user that doesn’t exist | The operation wasn’t successful |  |
|  | X | X | A regular user tried to delete another account | The operation wasn’t successful |  |
| 21 | Unit + GUI + System + Integration | Make sure the system allows **admins** to add awards to outstanding users and the users are getting notified about it | An admin is trying to add an award to an existing user | The award was added to the user’s account and he was notified about it |  |
|  | X | X | An admin is trying to add an award to a user that doesn’t exist | The operation wasn’t successful |  |
| 22 | Unit + GUI + System | Make sure the **admins** can modify the different configurations of the system | An admin tried to modify a configuration with valid data | The new configuration is set |  |
|  | X | X | An admin tried to modify a non-existing configuration | The operation wasn’t successful |  |
|  | X | X | An admin tried to modify a configuration with invalid data | The operation wasn’t successful |  |
|  | X | X | A regular user tried to modify a configuration | The operation wasn’t successful |  |
| 23 | Unit + GUI + System | Make sure only **master admin** can remove **regular admin** appointments | A master admin is trying to remove a regular admin | The regular admin appointment is removed |  |
|  | X | X | A master admin is trying to remove another master admin appointment | The operation wasn’t successful |  |
|  | X | X | A master admin is trying to remove an admin that does not exist | The operation wasn’t successful |  |
|  | X | X | A regular admin is trying to remove admin appointment | The operation wasn’t successful |  |
|  | X | X | A regular user is trying to remove admin appointment | The operation wasn’t successful |  |
| 24 | Unit + GUI + System | Make sure only **master admin** can add **regular admin** appointments | A master admin is trying to add a regular admin | A new regular admin appointment is created |  |
|  | X | X | A regular admin is trying to add admin appointment | The operation wasn’t successful |  |
|  | X | X | A regular user is trying to add admin appointment | The operation wasn’t successful |  |
| 25 | Unit + GUI + System + Integration | Make sure it is possible for **admins** to filter all rides in the system by certain criteria | An admin entered fast travel threshold | Only rides with maximum riding velocity above the threshold are being shown |  |
|  | X | X | An admin entered change of speed threshold | Only rides with amount of speed changes after alert above the threshold are being shown |  |
|  | X | X | Admin entered amount of breaking threshold | Only rides with number of breaks above the threshold are being shown |  |
| 26 | Unit + GUI | Make sure the users receive informative messages about the operations that were being performed. | This will be tested by checking the messages in the response in the Unit tests and by checking the alerts in GUI testing | X |  |
| 27 | Unit + GUI + System + Integration | Make sure **admins** can see statistics about the system in specific dates range | Admin has entered valid **From** and **To** dates | Statistics about the system between the dates provided are returned |  |
|  | X | X | Admin has entered **To** date that is before the **From** date | The operation wasn’t successful |  |
|  | X | X | Admin has entered just a **From** date | Statistics about the system between the **From** and the current date are returned |  |
|  | X | X | Admin didn’t enter any dates | Statistics about the system from all time are returned |  |
| 28 | Integration + GUI | Make sure the information is being stored in the DB | This will be tested by performing certain operation on the system that suppose to store information in the DB such as: Register, question sending, ending a ride, admin addition, hazard addition etc.…  Then we will Shut down the system to cause a call to the DB to store the information. then we will make sure the information is stored in the DB by performing another operation that requires the fetching of that data from the DB such as: Login, Login as admin and see question, see information about new hazards and admins etc.… | X |  |

# Chapter 3 – Non-Functional Requirements Testing

|  |  |  |
| --- | --- | --- |
| **Requirement** | **How we are going to test** | **Status** |
| 27 | We will activate the object detection system 4 times (for each type of alert) on a video of a route containing a dangerous object (which has been tested as such).  For each session, we will change the configuration settings so that the system creates a different type of alert.  The expected result is  that the type of alert that the system generates corresponds to the type of alert that it was supposed to generate |  |
| 28 | We will run the hazard detection system on all videos containing hazards and measure the distance between the hazard and the scooter rider (RP) while the alert is activated.  A distance greater than the ‘Minimum Distance To Alert’ is an unwanted result, another result is a result we would like and expect to receive |  |
| 29 | We will run the hazard detection system on all the videos that contain hazards and measure the duration of time when the alert is activated, we would like this time to correspond to the value set by the admin in seconds |  |
| 30 | We will activate the ‘fetch\_safe\_routes’ function from the ‘RoutesRetriever’ object that will be activated during the ‘start\_ride’ function from the Facade, we will expect that the ‘fetch\_safe\_routes’ function will return a list  containing routes of size equal to  ‘Number Of Routes’ value (Appendix A) |  |
| 31 | We will sort our tagged videos into 2 groups, one will contain videos that contain a large amount of hazards (to simulate peak time), the other group will contain videos that contain an amount of hazards that corresponds to normal activity time. The amount of danger required for each group is calculated based on previous data we have collected.  We will run the object detection system on both groups, for the simulated peak time group we will expect a response time (from the moment the hazard is detected until the alert is activated) of 1 second.  For the group simulating regular activity time,we will expect a response time of 0.1 seconds .  Also, we will run the system on all the videos we have, calculate the average response time and expect a value of 0.5 second. |  |
| 32 | we will run the application on web, ios, android and expect that the application’s color is green in all platforms |  |
| 33 | We will upload a new advertisement to the application, go to the advertisement page of the new advertisement we uploaded, and check if the details we entered, including the URL of the advertisement are correct.  In addition, we will enter the advertisements page and count (manually) the amount of advertisements and we will expect the value to be the same as the amount of advertisements in the the Database |  |
| 34 | We will activate the start\_ride function from a scooter containing RP and drive a certain way  We will expect that the ride screen will show us the riding speed at any given moment. |  |
| 35 | We will activate the system and after an extended period of time (several days or weeks) during which many test operations will be performed except from turning off the system by the administrators, we will expect that the system will work properly |  |
| 36 | We will create initiated multiple failures in the system, and we will expect normal operation of the system (no crashing) and that the system will display a detailed message about the problem |  |
| 37 | We will run a loop 10000 times, in each iteration a new thread simulating a client will be created, the client will login to the system and activate the main function’strat\_ride’.  we will expect normal operation of the system, that is, without crashes and malfunctions |  |
| 38 | The registration process in our system will be done manually by a technician in order to link the RP to a specific user, so we will not check this requirement |  |
| 39 | We will manually check this requirement by looking at the DB in the ‘USERS’ table, we will observe that the ‘PASSWORD‘ column will only have hash values, we will also make sure that when creating a new user the password is not saved at any stage in a variable but only its hash value |  |
| 40 | We will manually check this requirement by making sure that no buttons appear on the application client screens that enable admin functionality |  |
| 41 | We will initialize the system, after the initialization we will (automatically) compare the data included in the configuration file with the configuration file itself. A complete match of the data is expected |  |
| 42 | We will run the server on a Linux operating system. We will expect to valid operation of the system |  |
| 43 | We will run the administrator application on a  Windows OS(version 10 or higher). We will expect to valid operation of the system and clear visibility of all the user interface elements |  |
| 44 | we will run the rider application on web, ios, android and expect that the application windows will fit the device on which they run in terms of display and position in relation to the margins, thus all the user interface elements are visible and clear |  |
| 45 | We will create initiated failures in the communication between the system components, and we will expect normal operation of the system (no crashing) and that the system will display a detailed message about the problem |  |
| 46 | We will create initiated failures in the external services (using a proxy ) , and we will expect normal operation of the system (no crashing) and that the system will display a detailed message about the problem |  |
| 47 | We will activate the camera on a scooter in a route that contains certain objects that are less than 15 meters away with the shooting direction, we will expect to see all these objects in the video transmitted to the RP |  |
| 48 | We will activate several functions in the system, including functions with incorrect arguments that cause an error, after the activation we will observe at the error log and the event log and check whether all the actions we performed are mentioned in the event log and whether all the actions that generated an error are mentioned in the error log |  |
| 49 | We will login to the system as an admin and try to view the error log and event log, we will expect to be able to view them |  |
| 50 | we will run the application on web, ios, android and expect that the application’s language is English in all platforms |  |

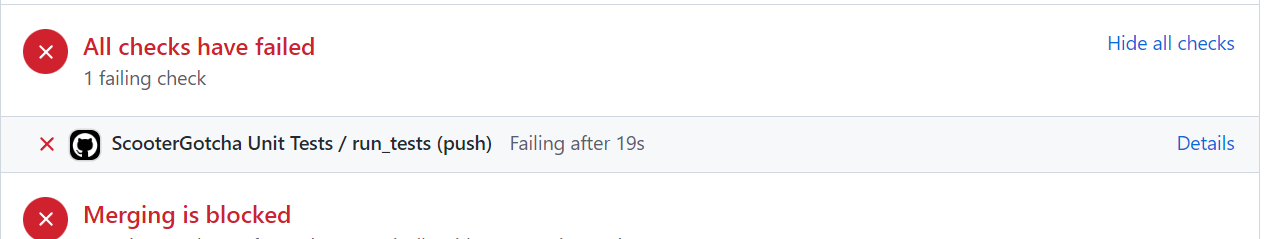
# Chapter 4 – Regression Testing

The purpose of this chapter is to describe the ways we will perform Regression testing on our project.

We will use GitHub Actions in order to run our Unit tests and later our Integration tests on all commits.

Each Pull Request will have to pass the unit tests successfully for it to be merged into main.

By using this method we will make sure that new changes don’t break the desired behavior of our system and we will protect the Main branch against bad commits.



As we can see the checks have failed and the merging is blocked. We can view the details of the failing tests in the details section.

# Chapter 5 – Robustness Testing

The purpose of this chapter is to describe the way we will perform robustness tests on our system.

We will perform those tests by manually shutting down the external services we are working with while running the system.

We will try to do the following:

1. Stop the DB
2. Disable our External Services
3. Disable GPS on our Raspberry Pi
4. Disable internet connection on our Raspberry Pi

We will manually make sure that the system continues to operate as expected

# Chapter 6 – Load Testing

The purpose of this chapter is to describe how are we going to test that our system can withstand the number of multiple connections we are required to by our client.

For this purpose we will JMeter which is a free tool that allows us to simulate any load we want on our system by creating multiple HTTP Requests to our server.

The results of the tests will be showed in a graph and in XML to represent the way our system works.

The general test scenario is as follows:

1. We will choose a number of request to perform X
2. We will choose an operation to perform such as: Register, Login, Send message, Fetch routes from external service, etc….
3. We will perform X HTTP request for the desired operation.
4. The JMeter will show us the latency and the Availability of our service when performing multiple concurrent HTTP requests.

General Note: The results of those tests are highly dependent on the computer that is running the tests, so different computers will give different results.

# Chapter 7 – GUI Testing

We will use Cypress to test our GUI performance. Cypress is a free JavaScript library which allows you to test the performance of your system end-to-end.

We will be able to test the entire flow of the system by using our real Backend and creating HTTP requests to our server and verify that the behavior is as expected.

We will also be able to solely test our GUI and see it is working as expected by using a stub for our Backend and intercepting the outgoing HTTP requests from the GUI to the Backend and return the result we want for each test and verify that the GUI is working as expected in multiple scenarios.

# Chapter 8 – Integration testing

**Server -** we will use bottom up methodology to test our integration,

We will use Mocks for testing each unit, and incrementally connect the separate units until we get complete integration of the server.

Diagram

Description automatically generatedFigure 9.2 - Server bottom up integration tests design

**Raspberry Pi -** we will use bottom up methodology to test our RP integration, our main challenge is to test Camera & Alert modules automatically.

In addition, our video process module will test with our tagged images hoard.

Diagram

Description automatically generated

Figure 9.3 - RP bottom up integration tests design

# Chapter 9 – Acceptance testing

The acceptance testing plan for the project will be performed per the usage scenarios using the data tables containing happy, sad and bad scenarios.

In accordance with the actors that activate the action, we will separate the tests into tests that are activated by each actor when the admin application, the rider application and the Raspberry Pi will activate the usage scenarios.

As part of our testing program, we will abstractly describe the Facade class on the server that implements the "complete functions" (that is, functions that are activated from another component and activate a set of sub-functions), as an endpoint that initiates the use case scenario.

For this tests, we intended to implement the following design patterns combination for separate system tests & implementation:

Diagram

Description automatically generated