MCI project Testing plan

Team number: 03
Project Title: A Map Management Platform for Self-Driving Cars

1. Introduction

1.1. Project Overview

Our project provides a solution to a problem faced by researchers in the autonomous driving field: the lack of a lifelong mapping platform for self-driving cars. The project goal is to develop a web-based application of map management for self-driving cars that can be continuously updated to represent a changing environment. The platform uses an SQLite database that contains geo-data and images downloaded from Mapillary, a crowdsourced image service. The data is processed using Quadtree based algorithms in the backend to identify all the nearest neighbour images having the same direction. The backend, which uses Django, renders the data to Django REST API from which the frontend using React consumes and processes the data. The frontend also uses Mapbox to render the map and Deck.gl to visualise layers.

1.2. Scope of the test plan document

The test plan document provides an overview of the project and its test plan scope as well as describes the testing approaches required for the project. All the test activities are carried out during milestone 2 of the project and explained in the Test Strategy section. Details of how the test cases are performed are listed in the Test Execution section. A testing plan schedule is provided, and risks that might occur during the testing process are identified in this document.

2. Test strategy

2.1. Test scope

The test scope is defined based on the application's main functions. There are three testing approaches used in the project: defect/unit testing, integration testing, and release testing. Defect/ Unit testing is to test individual components in isolation to discover defects in the application. Integration testing is performed to identify problems that might arise from component interactions. Release testing is carried out on the website release that will be distributed to the client to ensure that the application meets the client's requirements.

2.2. Testing report

The test results are recorded in a test report with key information as test type, test description, input, expected output, actual output, test date, person in charge, and reviewer. A summary table is provided in Appendix A to show a list of main test cases that have been conducted so far.

2.3. Test assumptions

- Source code has meaningful comments for all classes and functions.
- The set of inputs per test case is randomly chosen.

3. Test execution

3.1. Defect/ Unit testing

- Django views:
  - Quad Tree algorithm: testing utils/quadtree.py using sanity_check.py for randomly generated 1,000,000 points. Each point is generated from an array of np.random.randn(N, 2) where the 2 dimensions represent longitude and latitude values for any given coordinates. After that, all these points are inserted in the quadtree one by one for testing the accuracy of points stored to values of decimal precision and when queried the program returns correct values for longitude and latitude. Also, perform a stress test to check if the quadtree can handle up to 3 million values.

  To determine accuracy and testing for false positives or true negatives, a scatter plot is used to plot all coordinates in a set 2D plane of dimensions 500 X 500. Plot all coordinates in blue and domain in red as a rectangular bounding box. Plot queried coordinates in red and then after querying the algorithm for nearest neighbours, plot nearest neighbours in red. If on the boundary, the nearest neighbours shall be considered and any points outside shall be blue.
  - Nearest neighbours: create a test function in tests.py that checks the results of Quad Tree algorithm compared with KD Tree algorithm implemented from Scikit learn for validation of nearest neighbours using the same values in the test database.
  - Direction calculation: test the function that calculates the direction for a particular coordinate found in backend/views.py. The test consists of randomly generating 2 different coordinates 100 times, calculating the bearing (direction) between them using two different algorithms and validating if the difference (error) between them is less than 5%. The test is part of backend/tests.py.
- React components:
  - First test: when the element TopMenu is rendered, it should contain the text “Layers” and “Regions” in the menu bar.
  - Second test: suppose the object with id2 is the neighbour of object id1, the image key retrieved from the id2 in the neighbour list of id1 should be the same as the image key of object id2.
- Deck.GL layers: The set of utilities from Deck.gl (@deck.gl/test-utils) is used to test the layers. The utils allow testing the conformance and rendering of the layers. Four layers available in the frontend: Scatterplot, Path, Heatmap and Hexagon are generated with a small data sample created by the backend and checked using the testLayer from the utils.

3.2. Integration testing

- Mapillary API – Django integration:
  - Fetching sequences: The fetching function is tested to populate the database with information from Mapillary. To test it, a new region is created, after that, the fetching function is called and all the sequences are stored in the database. For the test to be considered successful, the data from Mapillary should be the same as the one stored in the backend.
  - Fetching users: Same test logic as fetching sequences but with the user data.
  - Downloading images: To test the downloading, a data sample is used with the information of image keys. The function should download all the images available in the data sample. To pass the test, the number of images should be the same as the number of coordinates stored in the database. Also, every image should be readable.
  - Scheduler: The scheduler orchestrates all other functions in the backend. To test the scheduler, a new region is generated in the Django administrator website, after that the job scheduler is run to fetch data from Mapillary and populate the database with the information for that region. To check that everything works correctly, all other functions should have been finalized as expected.

- Django - Django REST API - React integration:
  - Serializers - test if the 3 serializers CoordinatePropertySerializer, RegionSerializer, and SequenceSerializer under backend/serializers.py can properly serialize the 3 model instances from Coordinate_property, Region, and Sequence models, which is to convert the models to a JSON format that is served through the Django REST API: Firstly, create a set of attributes to initialize an object for Coordinate property, Region, and Sequence. Secondly, create a simple instance of the serializer initialized with these objects. Thirdly, verify if each serializer has the exact attributes it is expected to and produces the expected data to each field using assertEqual(). Finally, run the command *$ python manage.py test* in the terminal. All the test cases are recorded in backend/tests.py.
  - API views - test if the 3 API views under backend/api_views.py (CoordinatePropertyView, RegionView, SequenceView) show the correct total number of coordinates, regions, and sequences using ListAPITestCase subclass: Query all the objects from the database models equivalent to the API views and count them, then compare with the length of the response data from the GET method of each view's path. Run the command *$ python manage.py test* in the terminal. All the test cases are recorded in backend/tests.py.
  - Django REST API - React integration: check if a random value exists in the Coordinate API, Region API, and Sequence API when accessing from React. Using cypress, a Javascript end to end testing framework, we first visit the API URLs http://localhost:8000/api/coordinates, http://localhost:8000/api/regions, http://localhost:8000/api/sequences, and then check if the pages contain some certain values. Run the command *$ npm run e2e* in the terminal. All the test cases are recorded in frontend/cypress/integration.

3.3. Release testing

- Select a random region and expect that the map will be moved to the new destination.
- Select different layer types and expect that the layers will display data accordingly.
- Select an image point and expect that neighbour images are displayed, the point is highlighted and the sequence it belongs to is also highlighted.
- Select a certain date range and expect that coordinates and sequences are displayed according to the filtered date range.
- Access the published website using different web browsers to test browser compatibility.

4. Testing plan schedule

Refer to Appendix B - Testing plan schedule.

5. Risks

- Irrelevant test cases → to mitigate: team members need to discuss the test cases within the team and assign a reviewer for each test case.
- Tight timeline → to mitigate: team members have to strictly follow the testing plan schedule and give timely notification of potential problems during testing.
- High number of defects → to mitigate: the team should prioritize fixing the defects that are related to the main functionalities; other good-to-have functions will be fixed based on the extensibility plan.

Appendix A – Test report

Test type	Test description	Input	Expected output	Actual output	Test date	Person in charge	Reviewer
Defect/Unit Testing	Django views – Quad Tree algorithm	- Plot 500 coordinates on a 2D plane and highlights nearest neighbours in red. - 3 million randomly generated points	- A plot of all test points in blue and only nearest neighbours in red - Correct functioning of the algorithm under stress test as it should work with 500 points	Same as expected	11/05/2021	Aryaman Dhawan	Nhu Quynh Hoa
Defect/Unit Testing	Django views – Nearest neighbours	1,000,000 points randomly generated using np.random.randn(N,2) where the 2 dimensions represent longitude and latitude values	The same list of nearest neighbours generated from Quad Tree algorithm and KD Tree algorithm	Same as expected	12/05/2021	Aryaman Dhawan	Nhu Quynh Hoa
Defect/Unit Testing	Django views – Direction calculation	100 test cases with two coordinates each. Compared with another algorithm to calculate bearing between two points.	The difference (error) between the two algorithms is less than the tolerance error of 5%.	Same as expected	14/05/2021	Jonhatan Cotes Calderon	Nhu Quynh Hoa
Defect/Unit Testing	React components – Render TopMenu	TopMenu component	The TopMenu should be rendered successfully with the text "Layers" and "Regions" in the menu bar	Same as expected	15/05/2021	Huu Thanh Nguyen	Jonhatan Cotes Calderon
Defect/Unit Testing	React components – Check data in the neighbour list	Image key of object id2 and neighbour list of object id1	The image key retrieved from an object with id2 in the neighbour list of id1 should be the same as the image key of object id2	Same as expected	16/05/2021	Huu Thanh Nguyen	Jonhatan Cotes Calderon
Defect/Unit Testing	Deck.GL layers	Generate a JSON test data from the API to use as an input for the 4 layers available	Layers are generated properly after the validation of the testLayer of deckgl	Same as expected	18/05/2021	Jonhatan Cotes Calderon	Huu Thanh Nguyen

Test type	Test description	Input	Expected output	Actual output	Test date	Person in charge	Reviewer
Integration Testing	Mapillary API – Django integration: Fetching sequences and users	Generate a region and fetch sequences and users for that region	Function finished without any error, data populated in the database with all the sequences and users	Same as expected	12/05/2021	Jonhatan Cotes Calderon	Aryaman Dhawan
Integration Testing	Mapillary API – Django integration: Downloading images	Downloaded images from the database using the downloader	All images stored in the static folder	Same as expected	14/05/2021	Jonhatan Cotes Calderon	Aryaman Dhawan
Integration Testing	Mapillary API – Django integration: Scheduler	Use a sample database with a new region and run the scheduler from the Django admin site	Fetching and calculating functions finalized with all the data populated in the database.	Same as expected	15/05/2021	Jonhatan Cotes Calderon	Aryaman Dhawan
Integration Testing	Django - Django REST API - React integration: Serializers	- 3 random sets of attributes to initialize 3 objects for Coordinate property, Region, and Sequence - Serialized fields required for 3 models Coordinate_property, Region, and Sequence	Each serializer has the exact attributes it is expected to and produces the expected data to each field	Same as expected	17/05/2021	Nhu Quynh Hoa	Jonhatan Cotes Calderon
Integration Testing	Django - Django REST API - React integration: API views	- Objects from the database models equivalent to the API views, - Data responses from the 3 API views' paths	The total number of objects from the database models is equal to the length of data responses from the API	Same as expected	19/05/2021	Nhu Quynh Hoa	Jonhatan Cotes Calderon
Integration Testing	Django REST API - React integration	Data from the 3 API URLs of Coordinates, Regions, and Sequences	Certain test values indicated in the test cases exist in the API URLs when accessing from React	Same as expected	21/05/2021	Nhu Quynh Hoa	Huu Thanh Nguyen

Appendix B – Testing plan schedule

No.	Test description	Start	Finish
1	**Defect/ Unit testing**	**11/05/2021**	**19/05/2021**
1.1	***Django views***	***11/05/2021***	***15/05/2021***
1.1.1	Quad Tree algorithm	11/05/2021	12/05/2021
1.1.2	Nearest neighbours	12/05/2021	13/05/2021
1.1.3	Direction calculation	14/05/2021	15/05/2021
1.2	***React components***	***15/05/2021***	***19/05/2021***
1.2.1	Render TopMenu component	15/05/2021	16/05/2021
1.2.2	Check data in the neighbour list	16/05/2021	17/05/2021
1.3	***Deck.GL layers***	***18/05/2021***	***19/05/2021***
2	**Integration testing**	**11/05/2021**	**23/05/2021**
2.1	***Mapillary API – Django integration***	***11/05/2021***	***16/05/2021***
2.1.1	Fetching sequences	12/05/2021	13/05/2021
2.1.2	Fetching users	13/05/2021	14/05/2021
2.1.3	Downloading images	14/05/2021	15/05/2021
2.1.4	Scheduler	15/05/2021	16/05/2021
2.2	***Django - Django REST API - React integration***	***16/05/2021***	***23/05/2021***
2.2.1	Serializers	17/05/2021	19/05/2021
2.2.2	API views	19/05/2021	21/05/2021
2.2.3	Django REST API - React integration	21/05/2021	23/05/2021
3	**Release testing**	**24/05/2021**	**29/05/2021**
3.1	Select a random region	24/05/2021	25/05/2021
3.2	Select different layer types	25/05/2021	26/05/2021
3.3	Select an image point	26/05/2021	27/05/2021
3.4	Select a certain date range	27/05/2021	28/05/2021
3.5	Access the published website using different web browsers to test browser compatibility	28/05/2021	29/05/2021