AI Auto Car Classifier Testing Policy

Edited by:

Fiwa Lekhuleni
Abhinav Thakur
Vincent Soweto
Andrew Jordaan
Keorapetse Shiko

Contents

1 Introduction

This document provides information on the various testing mechanisms that are implemented in the system. It specifies the technologies and the policy to be followed in terms of the testing. A testing policy provides a defined structure that encourages transparency and effective team work.

2 Testing Objectives

Through testing, the aim is to investigate how the system is executing the certain tasks. The testing objectives are to define a structure of testing each subsystem and to see how the subsystems interact with each other, facilitate consistent and continuous testing, early detection of defects and to decrease the complexity of debugging.

3 Types of Tests

Conventional black-box testing is used by functional testing. Software testing is a dynamic validation activity to detect defects and undesirable behaviour in the software being tested, and demonstrated that the software satisfies its requirements and constraints.

3.1 Unit Testing

Unit testing entails separating each individual component (unit) of the system and testing them individually to ensure that they all behave in the expected manner, in line with the specified requirements. The Mocha framework is used to conduct the unit tests.

3.2 Automated Testing and Continuous Integration

Travis CI is the tool chosen for our continuous integration. It executes automated testing by running the unit tests after each commit. Travis CI displays the test results and maintains logs for each test executed.

4 Testing Methods and Techniques

4.1 Test Cases

DESCRIPTION	***INPUT***	***OUTPUT***
Testing car recognition	Text file	Image not found (status=500)
Testing car recognition	Image with no vehicle	Car not found (status=500)
Testing car recognition	Image with vehicle	Car manufacturer (status=200)

Functional Testing

The test scripts will have the *.test.js extension and will be stored under the test folder. The "npm test" command runs the Mocha test scripts and displays the test results on the terminal.

An example of a test is given below:

```
describe('Ensure we can check if an image contains a car or not', function () {
  it('It should succeed and return the confidence percentage', function (done) {
      request(app)
          .post('/classify/car_detector')
          .send({imageID: 'Image1.jpg'})
          .set('Accept', 'application/json')
          .expect('Content-Type', /json/)
          .expect(200)
          done()
  });

  it('It should return a 404 because the endpoint has a typo', function (done) {
    request(app)
        .post('/classify/car_detector')
        .send({imageID: 'Image1.jpg'})
        .set('Accept', 'application/json')
        .expect('Content-Type', /json/)
        .expect(404)
        done()
  });
```

Unit Test

5 Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	41.53	12.5	15.63	42.46	
backend	100	100	100	100	
app.js	100	100	100	100	
server.js	100	100	100	100	
backend/api	29.61	0	6.9	30.41	
classification.js	21.67	0	8.7	22.41	... 98,299,300,301
displayPage.js	50	0	0	50	6,7,8,10,11
logging.js	57.14	100	0	57.14	15,17,28,30,41,43
notification.js	75	100	0	75	12,14

Test Coverage

6 Required Resources

A few resources are required to conduct the tests. Since the system is a Node application, "npm" is required to call the test command. The Mocha framework executes the unit tests, this can be done in less than a minute. Travis CI can be found on it's website, it runs the tests and deployment in less than 5 minutes.