AI Auto Car Classifier Testing Policy

Edited by:

Fiwa Lekhuleni
Abhinav Thakur
Vincent Soweto
Andrew Jordaan
Keorapetse Shiko

Contents

1 Introduction

1.1 Purpose

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 Definition

- To validate and improve end user experience of the product, ensuring it is fit for its intended purpose.
- To prevent the migration of defective code throughout the system software development.
- Ensuring that the system requirements are met. These may be test requirements according to according to the documentation.

3 Approach to testing

We will measure, review and improve the quality of testing by making sure that we test as my cases as possible the different probable outcomes in a manner that is clearly defined, robust according to testing process and coding standards.

4 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.

5 Testing Benefits

- Ensuring that existing functions are validated during an upgrade/ development of the project.
- Testing that is directly linked back to regression testing and to the higher level functional and non-functional requirements.
- Testing is a consistent and repeatable activity rather than a bespoke activity for each change in functionality.
- Effective team-working through the adoption of standardised principles, processes and deliverables across the organisation.
- A software development process under which testing can take place from planning, design, execution to closure of project features.
- A test resource that documents and capture the key elements of the testing process

6 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.

6.1 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.

6.2 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.

6.3 Test Levels

Use of test levels ensures reduction of quality risk as early as possible and to ensure accuracy to the greatest degree. The table below describes the various test levels for the project

Level	Owner	Objective	Key areas of testing
Unit	Development	• Detect defective code in units • Reduce risk of unit failure in Live Service	• Functionality • System response time
Integration	Development	• Detect defects in end- to-end scenarios concentrating on system validation for each function performing as expect	• Functionality • Performance (Reliability) • Usability • Maintenance

Figure 1: Test Levels

7 Testing Methods and Techniques

7.1 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

8 Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	28.73	4.72	14.89	29.51	
AI-Auto-Car-Classifier	100	50	100	100	
app.js	100	50	100	100	41
server.js	100	100	100	100	
AI-Auto-Car-Classifier/api	15.56	0	1.28	16.08	
authentication.js	41.94	0	0	41.94	... 34,35,36,38,42
classification.js	10.99	0	1.41	11.41	... 48,949,959,960
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
AI-Auto-Car-Classifier/config	70.83	0	0	70.83	
config.js	100	100	100	100	
passport.js	66.67	0	0	66.67	... 21,23,25,26,28
AI-Auto-Car-Classifier/models	96.88	80	83.33	96.88	
car.js	100	100	100	100	
index.js	95	80	66.67	95	24
inventory.js	100	100	100	100	
user.js	100	100	100	100	

Figure 2: Use Case Diagram

Check out the Travis logs here https://travis-ci.org/cos301-2019-se/AI-Auto-Car-Classifier/builds

9 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.