Auto Car Classifier
Coding Standards and Quality Documentation

Edited by:

Fiwa Lekhulani
Abhinav Thakur
Vincent Soweto
Andrew Jordaan
Keorapetse Shiko

Contents

1 System Overview

This system is a progressive web app that can detect cars in images, extract plates from the images and classify the car according to make model and year

2 Introduction

The aim of this document is to describe our conventions and styles to ensure a uniform style, clarity, flexibility, reliability and efficiency of your code. this documents outlines coding practices that we should adhere to for the project and organisation.

3 Definitions

Definitions:-

Style: is a set of rules or guidelines used when writing the source code for a computer program.
Clarity: clearness or lucidity as to perception or understanding of written code
Flexibility: defined by the ease with which you can modify it to fulfill some purpose you hadn't conceived possible at the time you wrote it
Reliability: is when programmers have developed a series of best practices when programming to ensure longevity and maintainability of developed code
Efficiency: describes the reliability, speed and programming methodology used in developing code for an application.

4 Coding Standards

4.1 File Names

The file names are supposed to be verbs so as to ensure that whoever reads the filename understand the functionality implemented in the file. for javascript the file have a js file extension. In the image below the filename predict.js has function preprocessImage which is self-explanatory with respect to tensorflow js. For input it takes and image and modelName. File names must be implemented in camelCase ie predict.js instead of Predict.js.

4.2 Description of Classes

Classes - class names should be camelcased with the first letter starting with a uppercase letter.
Purpose- they are used for object oriented programming in which the

class is used for creating instances of objects.
Description of methods is a method which is bound to the class and not the object of the class. They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
Description of fields
In-code comments were used for tasks for tasks that were outstanding and still had to be finished. Different software vendors use different coding standards and automated documentation tools. As a result some files had different coding stands.
We also stated that if the code is not simple and self-explanatory it should have comments so that whoever is reading it can understand. For example complex code that is related to machine learning is accompanied with comments

For JavaScript since there are functions so the use of classes is rarely used because——

Coding conventions include the following:

4.2.1 Naming conventions

- Reserverd words - when using naming conventions for javascript one cannot use the following key words for break , case , catch , continue , debugger , default , delete , do , else , finally , for , function , if , in , instanceof , new , return , switch , this , throw , try , typeof , var , void , while , and with. Separate any reserved word (such as if, for, or catch) from an open parenthesis (() that follows it on that line.Separating any reserved word (such as else or catch) from a closing curly brace that precedes it on that line.

- variable names that look similar should be avoided to avoid confusion ie
  Good:
  firstName = "John";
  lastName = "Doe";
  price = 19.90;
  tax = 0.20;

  Bad:
  var nam = 'liz';
  var nom = 'loitze'

- imports in js tend to conflict if they are not used in the correct programming language ie import * as tf from '@tensorflow/tfjs' instead of

- use let instead of var because var exists globally and let exists within block scope. This increases efficiency by saving on memory.
  Good:
  function find()

  let num = 0;
  Bad:
  function find()

  var num = 0;

- software version - to ensure that all the code that is written it has to adhere to particular standards for example in tensorflow the tensorflowjs loadModel function was changed to loadLayersmodel in a later version.

- code versions - branching is used to develop a new feature that could be introduced to make sure that there is one at least one working branch that can be used.

- Models - each models names is related to what it is supposed to classify. This these are the carDectectorModel.
  Good: objectDetectorModel
  Bad: model

- Paths - file paths are maintained a format specific to each languages API
  Good: const notes = '/users/joe/notes.txt'
  path.dirname(notes) // /users/joe
  path.basename(notes) // notes.txt
  path.extname(notes) // .txt
  Bad:
  path.dirname('/users/joe/notes.txt') // /users/joe
  path.basename('/users/joe/notes.txt') // notes.txt
  path.extname('/users/joe/notes.txt') // .txt

- Files - those that are of similar data are group in a folder related to the code that will use it.

- Attributes - attributes that use instead of magic numbers, string, etc. This also allows programmers to backtrack the origin of a variables definition.
  Good:
  var numOne = 1, numTwo = 2;
  var total = numeOne + numTwo;

Bad:
var total = 1 + 2;

- Functions function names are supposed to be verbs since they encapsulate blocks of code that perform a specific task
  Good:
  function addition()
  Bad: function funky()

- Constants const variable are used within the scope they are used in. This allows for efficient memory usage.
  Good:
  function add()
  const numOne = 1, numTwo = 2;
  return total = numeOne + numTwo;

  Bad:
  const numOne = 1, numTwo = 2;
  function add()
  return total = numeOne + numTwo;

- Special case characters - these are the escape sequence (; ;
  , , , , ; ,)
  Good:
  Bad:

4.2.2 Formatting conventions

- Ordering of program statements - the logic of statement execution should be a particular order to ensure that implementation is correct.

- Line breaks - After each semicolon and the end of each statement. We break a line an start inline with the previous statement unless it is a logic statement, braces.
  Good: let num = 0;
  let op = '*';
  Bad:

- Indentation - depending on the statement or function being executed. Most spacing is done by using a tab space. Tab spaces are equals to four spaces
  Good:
  function toCelsius(fahrenheit)

return (5 / 9) * (fahrenheit - 32);

NB: Different editors interpret tabs differently.

- Alignment - all blocks of code that are related should be grouped together.
  Good:
  let num = 0;
  let stringNum = ";
  let zeroBin = '000000';
  Bad:
  let num = 0;

  let stringNum = ";

  let zeroBin = '000000';

- Spacing a single space is used to separate operation in a statement and make it easier to read.
  Good: let total += three;
  Bad: let total+=three;

- Camel case- they ensure that long car variable name are easy to understand.
  Good:
  carBooleanDetector
  Bad:
  CarBooleanDetector

- Directories - all files related to the same tasks should be in the same folder. For example test files and images will be in files with similar data.Backend and documentation will have sub folders such as assets and tex files subfolders.

- Spaces Around Operators - Always put spaces around operators (= + - * /), and after commas:

  Good:

  var x = y + z;
  var values = ["Volvo", "Saab", "Fiat"];
  Bad: var x=y+z;

- Semi colon - as soon as the statement is done it should be ended with a semicolon with no spaces.
  Good: let num = 0;

Bad: let num = 0 ;

- Braces are required for all control structures (i.e. if, else, for, do, while, as well as any others), even if the body contains only a single statement. The first statement of a non-empty block must begin on its own line.
  Good:
  if (someVeryLongCondition())

  doSomething();

  Bad:
  if (someVeryLongCondition())
  doSomething();

4.2.3 In-code comment conventions

- Single-line comments are placed before functions. These comments detail use of the function, what arguments it uses and what it returns if specified to do so.
  Good:
  // single line comment

- Depending on the programming languages a particular syntax is used for javascript block statement /* */
  Good:
  /*
  *
  * This is a multi-line comment.
  */

5 Code Quality

5.0.1 Coding Standards

By the following rules that have been set in out coding standard we ensure that there is uniform structure to all the code that has been written in a manner that adheres to the coding styles.

5.0.2 Linting Software

Various tools are used to ensure that coding standards such as code syntax is maintained. Some online platforms include:

- JSONLint - The JSON Validator that is found at the website https://jsonlint.com/
- HTML Code Editor - Instant Preview that is found at the website https://htmlcodeeditor.com/

- JavaScript Tester online that is the found on the website https://www.webtoolkitonline.com/javascript-tester.html

6 Tree Structure

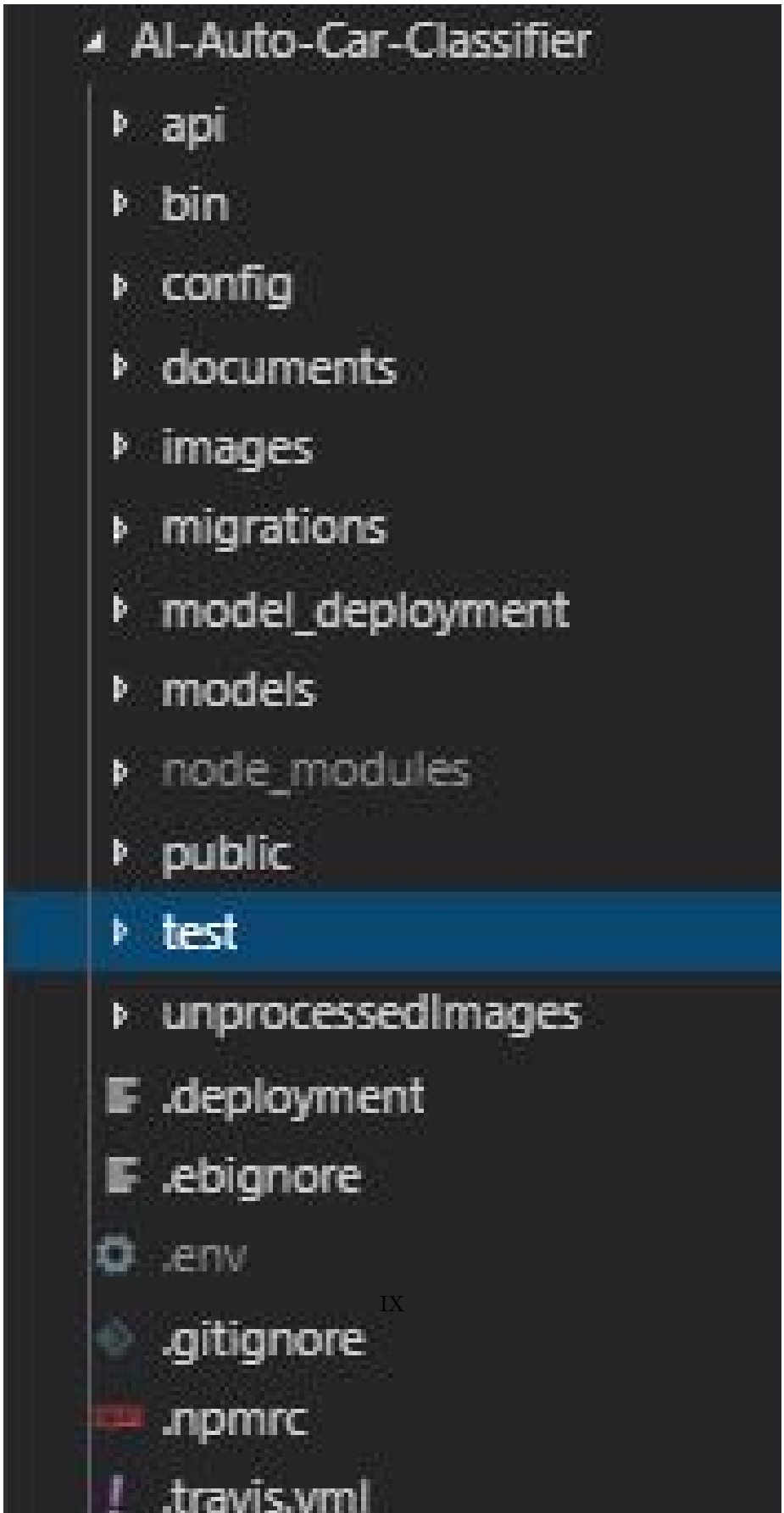
6.0.1 Repository rule

- Do not commit to master branch with out a pull request
- Peer reviews should be done for pull requests before branches are merged.
- All open branched should eventually be closed.
- When working on a new feature in dev branch, a new branch should be create and merge when the code is working and stable.
- Folder names should be descriptive verbs and east to understand

7 File Structure

- api - has api javascript files.
- config - has database consfiguration javascript files.
- documents - contains files for documentation of the system.
- migrations - to keep track of changes to the database.
- $node_modules - contains all the libraries that were used to build the project public - has all the files related to the website html, css and javascript pages.$

- test - it contains the test files for the project.


AI-Auto-Car-Classifier
api
bin
config
documents
images
migrations
model_deployment
models
node_modules
public
test
unprocessedImages
.deployment
.ebignore
.env
.gitignore
.npmrc
.travis.yml