Alabama Liquid Snake

University of Pretoria

Epi-Use

Botic - Privacy aware chatbot
Coding Standards

Justin Grenfell - u16028440
Peter Msimanga - u13042352
Alicia Mulder - u14283124
Kyle Gaunt - u15330967
Lesego Mabe - u15055214

Contents

1 Introduction

This document is meant to contain all details regarding the coding standards used in the development of all subsystems.

1.1 Frontend: Angular

In Angular, each component makes use of a ts file, Typescript, which is a superset of JavaScript that enforces typing, a view which uses HTML (Hypertext Markup Language) as well as styled using CSS (Cascading Style Sheets). These are going to be the focus of the coding standards defined in this subsection, as well as the directories that contain the software artifacts written in the languages listed in the previous sentence.

1.1.1 Naming Conventions

Here we have decided to makes use of the package "tslint-consistent-codestyle." This packages add many useful coding rules that are not available when using vanilla TSLint, such as the ability to control the way declare a variable, function and et cetera.[1]

The Angular CLI will be used for the creation of each component and service. New components are created using the command "ng g c" followed by the directory "components", and the subdirectory according to the function of the component, and lastly the name of the component. Here is an example:

```
ng generate component components/layout/footer
```

Following this rule, services and models are seperated into their respective folders, and within those folders, they are separated and ordered according their the functions.

Models, components and views are created using lowercase names. Since Angular CLI adds identifying information to the names of each file, i.e. ".component.ts" for components, ".component.html" for view, and ".service.ts" for services, the name of the file in lower case according the name of the component, as well as it's positioning in an appropriate directory, is enough.

In the Typescript files we will require that by default, camel case is used for both method names and variables. Members that are modifiers, however will need to have a leading underscore and the type is indicated using Pascal case.

1.1.2 Layout Rules

Here is a snippet containing some of our coding standards:

```
{
  "rulesDirectory": [
    "node_modules/codelyzer",
    "tslint-consistent-codestyle"
  ],
  "rules": {
    "arrow-return-shorthand": true,
    "callable-types": true,
    "class-name": true,
    "comment-format": [
      true,
      "check-space"
    ],
    "curly": true,
    "deprecation": {
      "severity": "warn"
```

```
    },
    "eofline": true,
    "forin": true,
    "import-blacklist": [
      true,
      "rxjs",
      "rxjs/Rx"
    ],
    "import-spacing": true,
    "indent": [
      true,
      "spaces"
    ],
    "interface-over-type-literal": true,
    "label-position": true,
    "max-line-length": [
      true,
      140
    ],
    "member-access": true,
    "member-ordering": [
      true,
      {
        "order": [
          "static-field",
          "instance-field",
          "static-method",
          "instance-method"
        ]
      }
    ],
    "no-arg": true,
    "no-bitwise": true,
    "no-console": [
      true,
      "debug",
      "info",
      "time",
      "timeEnd",
      "trace"
    ],
    "no-construct": true,
    "no-debugger": true,
    "no-duplicate-super": true,
    "no-empty": false,
    "no-empty-interface": true,
    "no-eval": true,
    "no-inferrable-types": [
      true,
      "ignore-params"
    ],
    "no-misused-new": true,
    "no-non-null-assertion": true,
    "no-shadowed-variable": true,
    "no-string-literal": false,
    "no-string-throw": true,
    "no-switch-case-fall-through": true,
    "no-trailing-whitespace": true,
    "no-unnecessary-initializer": true,
```

```
    "no-unused-expression": true,
    "no-use-before-declare": true,
    "no-var-keyword": true,
    "object-literal-sort-keys": false,
  }
}
```

Using Visual Studio Code and having installed TSLint from the market place (`https://marketplace.visualstudio.com/items?itemName=eg2.tslint`), all the errors will be displayed whilst one is coding, however, running the command "ng lint" will display all liniting errors within the project. These rules will be enforced using Prettier.

1.1.3 Commenting Practices

These haven't been defined just yet.

2 File Structure

2.1 Frontend: allisn directory

The Angular CLI will be used for the creation of each component and service. New components are created using the command "ng g c" followed by the directory "components", and the subdirectory according to the function of the component, and lastly the name of the component. Here is an example:

```
ng generate component components/layout/footer
```

Following this rule, services and models are seperated into their respective folders, and within those folders, they are separated and ordered according their the functions.

2.2 Message Scrubber: ai-message-scrubber

All files are in the same directory for the time being.

3 References

[1] A. Giretti, "Getting started with tslint and tslint-consistent-codestyle in angular 5+ projects, part 1: Visual studio code," https://nexusinno.com/en/getting-started-with-tslint-and-tslint-consistent-codestyle-in-angular-5-projects-part-1-visual-studio-code/, May 2018, accessed on 2019-05-24.