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. The coding standards will be analysed on a per subsystem basis. We also define rules pertaining to the file headers as well as the description of classes.

1.1 File Headers

We require that our file headers have the following information:

Item	Description
File name	This is just the file name of the code file
Version number	The version number.
Author name	The name of the programmer who creates this file.
Project name	The name of the project.
Organization	The name of the team or organization that developed the project.
Related Use Cases	The shows the related use cases the resulted in the creation of this file.
Update history	List of updates, with the date of the update, the author of the changes, what was updated and why.
Reviewers	List of reviewers and what they reviewed.

Here is an example:

1.2 Class Descriptions

Each class will need to have a description of its purpose, a description of its methods i.e. the purpose of each, the parameters, return type, input and output, and if possible or necessary, the asymptotic complexity of the algorithm executed[1]. It should also have a description of each field, this includes the name of the field, the data type, and initialization requirement.

Here is an example:

1.3 Frontend: Angular7

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.

1.3.1 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.[2]

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.

The View will be bundled into varous components; thus the "components" directory will hold all components comprising of the view. The Controllers will be bundled into various services; thus the "services" directory will hold all the controllers. The Model will also be represented as "service" components. This is how we will map each of the Angular constructs to the architecture we have chosen.

In the Typescript files we will require that by default, camel case is used for both method names and variables. The type is indicated using Pascal case.

1.3.2 Formating Rules

Here is a snippet containing some of our coding standards and additional formating rules with are checked in our code using tsLint:

```
{
  "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.

1.3.3 Commenting Practices

These haven't been defined just yet.

1.3.4 Enforcement

1.4 Other Subsystems using TypeScript

The Persistence layer, as well as the Chatbot layer (both APIs and modules) will be using TypeScript which will be transpiled into JavaScript. Since we have already begun the use of Typescript in our Angular frontend, we can continue using it in our Node API. The benefits of this are getting a more "universal" language and standard for our project, including testing, having less errors because we now use a type safe language rather than JavaScript, Typescript can allow our IDEs to expose project modules easily and the more robust language offers more reliability[3]. To this end, the tuturial in [3] helped us use TypeScript instead of JavaScript in a RESTful API.

Like in the frontend, we will stick to the same formating rules, and the relevant naming conventions. Tslint will be soon be deprecated though, and migrations to Eslint will be emphasized [4]. We will migrate to Eslint, once we are confident that other high priority issues have been handled. The naming conventions for the Persistence API as well as Chatbot API will have to be checked manually however, for the time being. The rules meant to be enforced are:

```
"naming-convention": [
            true,
            {"type": "default", "format": "camelCase", "leadingUnderscore": "forbid", "
    trailingUnderscore": "forbid"},
            {"type": "method", "modifiers": "public", "format": "camelCase"},
            {"type": "member", "modifiers": "private", "leadingUnderscore": "require"},
            {"type": "type", "format": "PascalCase"}
        ],
```

1.5 Code Review Checklist

The code review checklist provided by [1] will be used to infer our code review checklist:

1. Does the program correctly implement the functionality and conform to the design spec?
2. Does the implementation comply to coding standards?
3. Are the programming constructs correctly used?
4. Are there any potential performance bottlenecks, or the inability to fulfill timing constraints?

These will be reviewed as the project continues.

2 References

[1] D. C. Kung, Object-Oriented Software Engineering An Agile Unified Methodology. McGraw-Hill, 2014.

[2] 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.

[3] A. Gardi, "How (and why) you should use typescript with node and express," https://medium.com/javascript-in-plain-english/typescript-with-node-and-express-js-why-when-and-how-eb6bc73edd5d, October 2018, accessed on 2019-08-06.

[4] adidahiya, "Roadmap: Tslint -> eslint," https://github.com/palantir/tslint/issues/4534, Feb 2019, accessed on 2019-08-07.