PAWDOPT

Coding Standards

Table of contents

Table of contents 2

Introduction 3

2. File Structure 3
2.1 Code Structure for Frontend 3
2.2 Code Structure for Backend 4
2.3 Code Structure for Angular App Base 5
2.4 Code Structure for ML python script 6

3. Function Descriptions 6
3.1 Typescript functions 6
3.2 GraphQL Functions (API calls) 7

4. Naming Conventions 7
4.1 File Names 7
TypeScript 7
Python 7
4.2 Classes 8
Typescript 8
Python 8
4.3 Variables 8
Typescript 8
Python 8
4.4 Constants 8
Typescript 8
Python 8
4.5 Functions 9
Typescript 9
Python 9

5. Formatting 9
5.1 Indentation 9
TypeScript 9
Python 9
5.2 Spacing 10
TypeScript 10
Python 10

6. Commenting 10
6.1 Single line Comments 10
TypeScript 10
Python 10
6.2 Multi-line Comments 11
TypeScript 11
Python 11

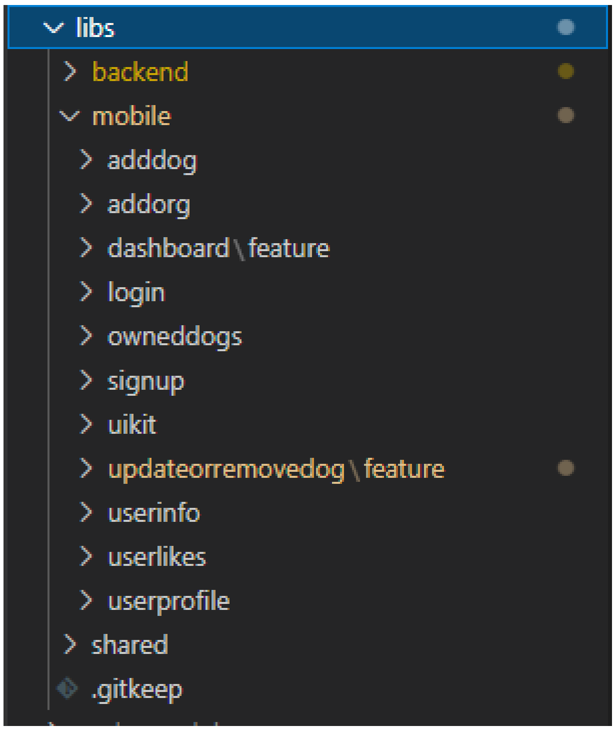
1. Introduction

This document contains the coding standards that the team will employ for the development of the PaWdopt application.

2. File Structure

2.1 Code Structure for Frontend

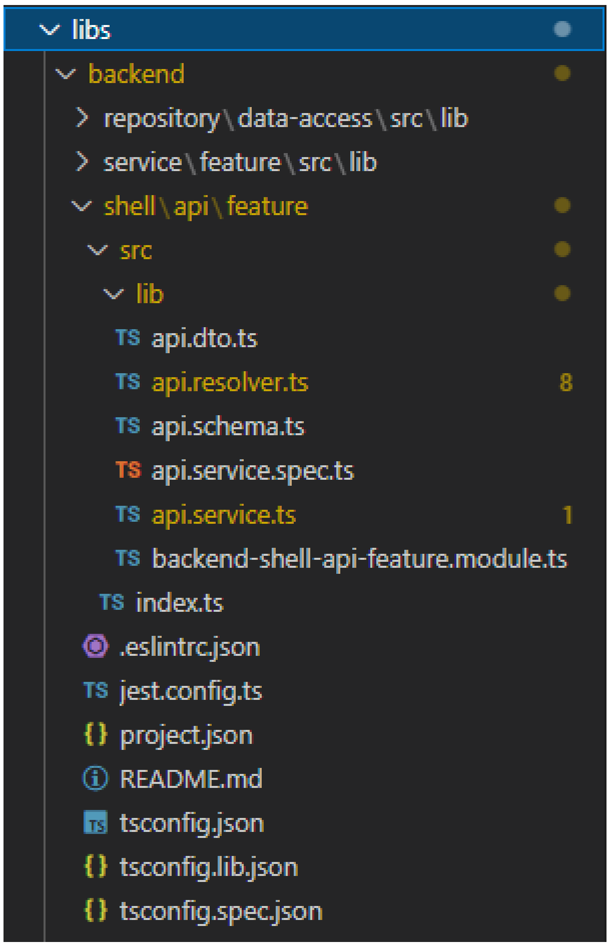
Since the Project uses Angular, the decided file structure was to set the project up with a main module and then every page would be a feature in the libs folder.



```
libs
  backend
  mobile
    adddog
    addorg
    dashboard\feature
    login
    owneddogs
    signup
    uikit
    updateorremovedog\feature
    userinfo
    userlikes
    userprofile
  shared
  .gitkeep
```

2.2 Code Structure for Backend

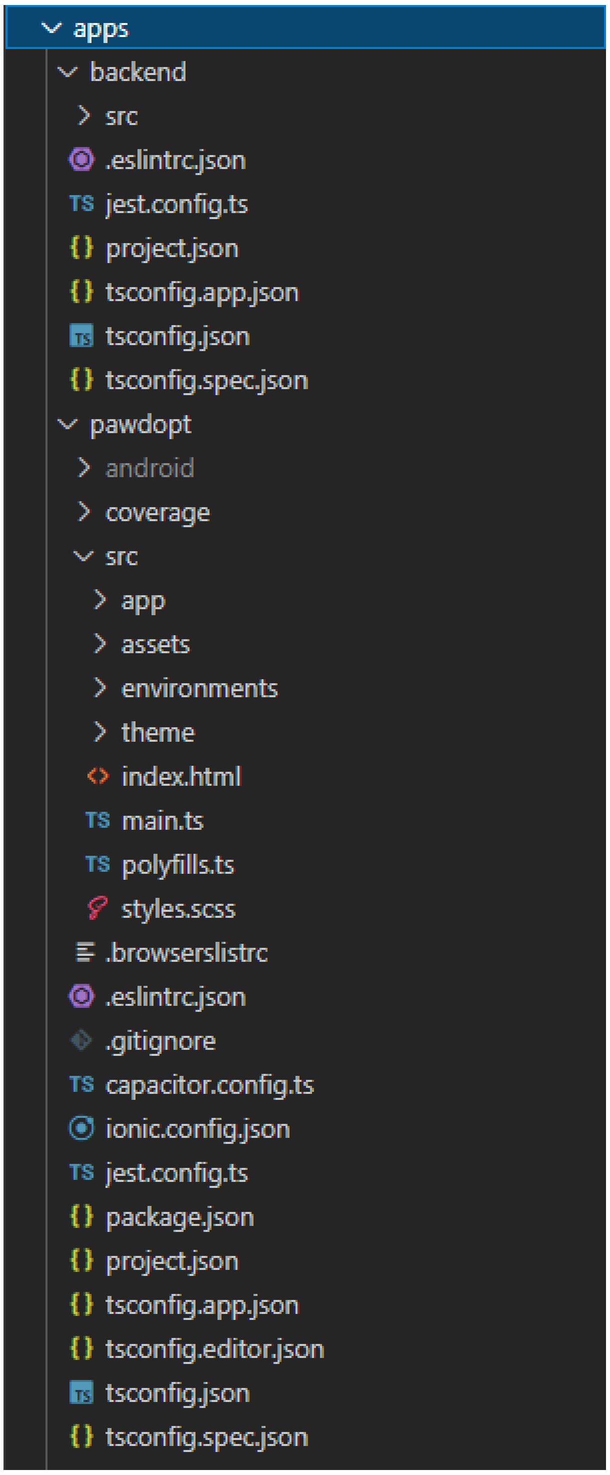
The backend is split into a shell in a backend folder contained in libs. This is also due to the Angular nature of the project.



```
libs
  backend
    repository\data-access\src\lib
    service\feature\src\lib
    shell\api\feature
      src
        lib
          api.dto.ts
          api.resolver.ts                8
          api.schema.ts
          api.service.spec.ts
          api.service.ts                 1
          backend-shell-api-feature.module.ts
        index.ts
      .eslintrc.json
      jest.config.ts
      project.json
      README.md
      tsconfig.json
      tsconfig.lib.json
      tsconfig.spec.json
```

2.3 Code Structure for Angular App Base

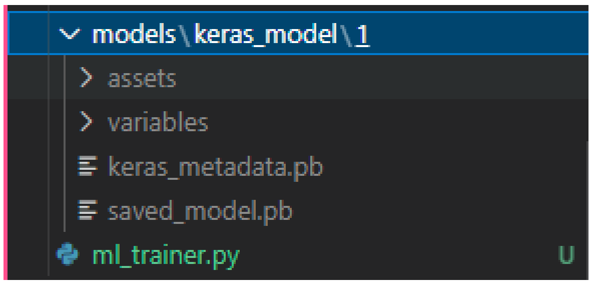
The main modules are contained in the apps folder in their respective headings.



```
apps
  backend
    src
    .eslintrc.json
    jest.config.ts
    project.json
    tsconfig.app.json
    tsconfig.json
    tsconfig.spec.json
  pawdopt
    android
    coverage
    src
      app
      assets
      environments
      theme
      index.html
      main.ts
      polyfills.ts
      styles.scss
    .browserslistrc
    .eslintrc.json
    .gitignore
    capacitor.config.ts
    ionic.config.json
    jest.config.ts
    package.json
    project.json
    tsconfig.app.json
    tsconfig.editor.json
    tsconfig.json
    tsconfig.spec.json
```

2.4 Code Structure for ML python script

The code structure for the Machine learning script is a single python script in the ml folder which generates models into a folder with the structure: models/{model_name}/{version}/

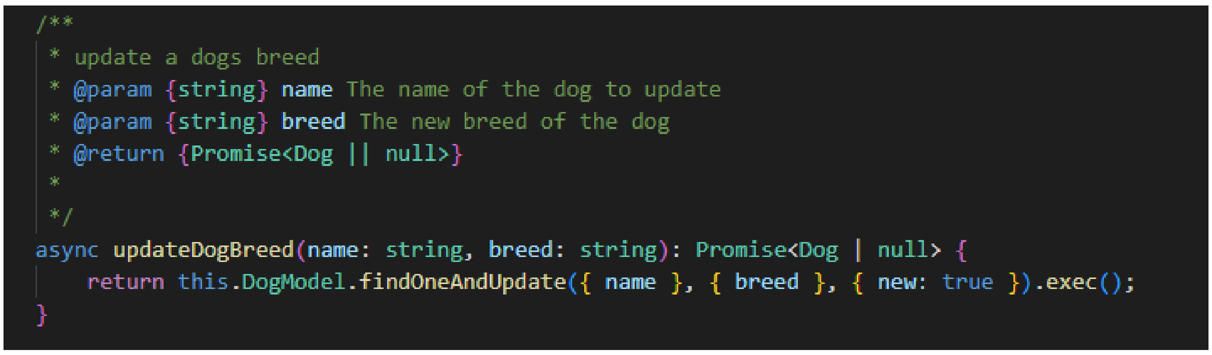


```
models\keras_model\1
  > assets
  > variables
  keras_metadata.pb
  saved_model.pb
ml_trainer.py                U
```

3. Function Descriptions

3.1 Typescript functions

Example of a function description:

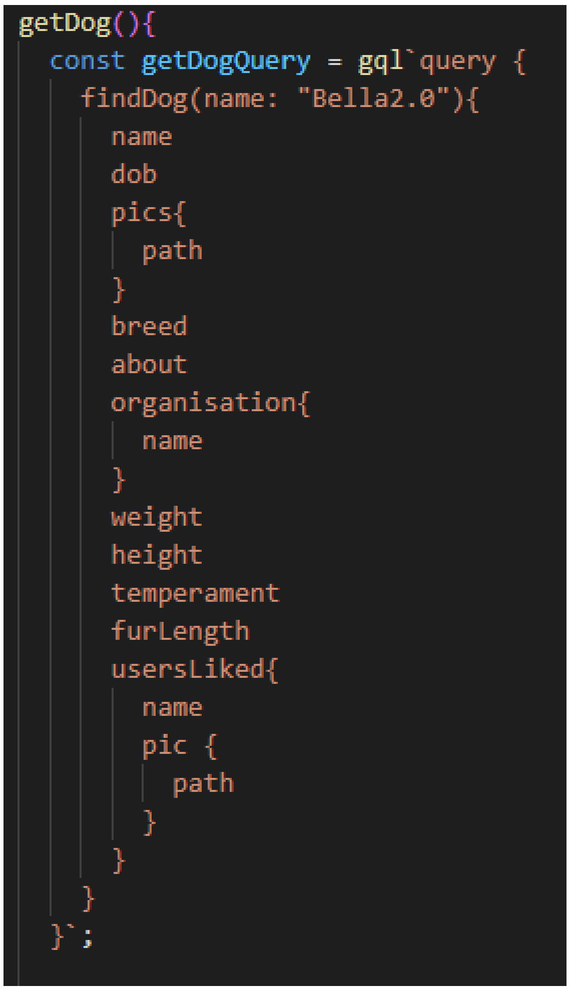


```
/**
 * update a dogs breed
 * @param {string} name The name of the dog to update
 * @param {string} breed The new breed of the dog
 * @return {Promise<Dog || null>}
 *
 */
async updateDogBreed(name: string, breed: string): Promise<Dog | null> {
    return this.DogModel.findOneAndUpdate({ name }, { breed }, { new: true }).exec();
}
```

Figure 1: Function description example

3.2 GraphQL Functions (API calls)

A GraphQL function format is shown in figure 2 below. This is the standard for all instances of the same nature.



```
getDog(){
  const getDogQuery = gql`query {
    findDog(name: "Bella2.0"){
      name
      dob
      pics{
        path
      }
      breed
      about
      organisation{
        name
      }
      weight
      height
      temperament
      furLength
      usersLiked{
        name
        pic {
          path
        }
      }
    }
  }`;
```

Figure 2: Function description example

4. Naming Conventions

4.1 File Names

TypeScript

For naming source code files, the team will use the lower dot case. All the letters are lowercase and words are separated with dots. Example:

api.service.ts

Python

The filenames for Python files (such as the ML script) should follow snake_case formatting.

4.2 Classes

Typescript

The name of classes will be the same as the name of the file containing the class. We will use the pascal case. Words are delimited by capital letters. Example:

```
export class ApiService{...}
```

Python

The names of classes need to be descriptive of the function and use lower snake_case.
Example

```
class ml_trainer():
    …
    …
```

4.3 Variables

Typescript

Variables will be named using Camel Case. Words are delimited by capitals except the initial word. Example:

```
let getDogQuery = …;
```

Python

Variables will be named using snake_case. Example:

```
image_size
```

4.4 Constants

Typescript

Constants will be named using Camel Case. Words are delimited by capitals except the initial word. Example:

```
const userInfo = …;
```

Python

Constants will be named using SCREAMING_SNAKE_CASE. Example:

```
FINAL_MODEL
```

4.5 Functions

Typescript

For functions we implement using Camel Case. Words are delimited by capitals except the initial word. Example:

```
getDogs{...}
```

Python

Functions will be named using snake_case. Example:

```
def make_ml_model():
```

5. Formatting

5.1 Indentation

TypeScript

Indentation will be used to ensure readability of the source code to keep the app easily maintainable. Examples of where indentation will be used closely are: conditional statements, loops and scope blocks. Indentation will make use of tab spaces of 2.

Examples of indentation:

- Conditional statements:
```
if(condition) {
  instruction1;
}
else {
  instruction2;
}
```
- Looping statements:
```
for(i = 0; i < j; i++) {
  instruction1;
}
```
- Scope blocks:
```
exampleFunction(param1: type, param2: type) {
  intruction1;
  instruction2;
}
```

Python

Standard Python indentation will be used.

5.2 Spacing

TypeScript

- There will be a space if an operator is followed by a parenthesis.
- There will be no space if a parenthesis is followed by an operator.
- There will be a space between operators.
- A space will always be followed after a comma.

Examples of spacing:

- functionCall(a, b, c) {

Python

- There will be a space if an operator is followed by a parenthesis.
- There will be no space if a parenthesis is followed by an operator.
- There will be a space between operators.
- A space will always be followed after a comma.

6. Commenting

6.1 Single line Comments

TypeScript

Single line comments will be implemented using the standard double backslash. They will be used to explain any code that requires an explanation. An example of this is:

```
//This is a comment
```

Python

Single line comments will be implemented using the standard pound. They will be used to explain any code that requires an explanation. An example of this is:

```
#This is a python comment
```

6.2 Multi-line Comments

TypeScript

Multi-line comments will be indicated using the “/* */” notation and used to indicate comments that extend across multiple lines. An example of this is:

```
/* This is a comment
This is part of the comment
This ends the comment */
```

Python

Multiline comments will use the standard pound notation in python except when defining the purpose of a function. In which case they will use docstring notation using three double quotes before and after the comments in new lines.
Examples:

-
  ```
   # This
   # is a multi-line
  # comment.
  ```

-
  ```
  def some_function():
  ”””
  This is a docstring comment
  defining some_function
  ”””
  ```