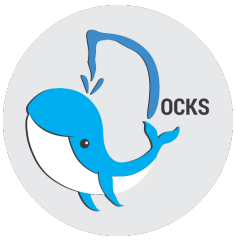


Coding Standards for Docks

Team: TripleParity
Client: Compiax

Team Members
Francois Mentz
Connor Armand du Plooy
Raymond De Vos
Evert Geldenhuys
Anna-MariÃľ Helberg
Paul Wood

1 Introduction

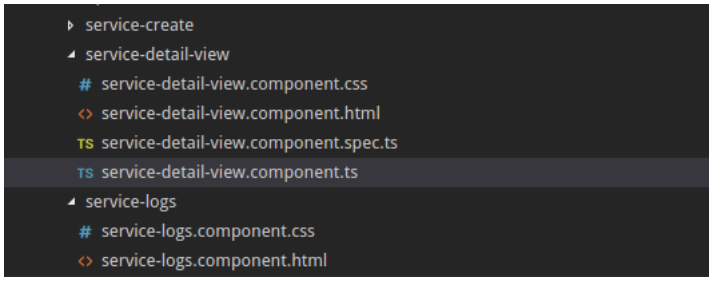
This document gives a brief overview of the coding standards we plan on using in our project. To ensure consistency throughout our project we will cover how we will be designing our coding documents written in Angular (front end), and JavaScript (back end). We will be using TypeScript in Angular and therefore will be keeping the same coding standards for both front and back end JavaScript.

2 Required and optional items

2.1 Naming Conventions

In Angular 5 we generate components inside the project. Each Component contains a single HTML, CSS, Spec and TypesScript file. To ensure consistency across all file names when creating a component Angular Components use only lower case letters and in any case where there are multiple words separate each with a "-" ie. "stack-view";
In Angular 5 we also generate our own Modules which we can use to represent a JSON object of variables we can pass too and receive from the docker API. To keep consistency, Modules should be created in the same way as Components. ie. "stack"
We also make use of Angular 5 Services which we use to help us make an asynchronous system. When creating a Service we should keep to the same naming conventions as Modules and Components. It is also required to append the word service ie. "stack-service" to a service. It is Import to Note that Angular generates Components, Modules, and Service objects with capital letters and camel casing but we would prefer file names to remain lower case and the actual Component being passed around by Angular to remain Capitalized with camel casing.

Function names inside TypeScript files will always start with a lower case letter, if the function requires more than one word it will make use of camel casing ie. "getStack()". Attributes and local variables will follow the same naming conventions as functions.
Objects inside each module must always start with a capital letter.



Above is a view of the file names of the service-detail-view component.

2.2 Layout rules

Our group members make use of the Visual Studio Code IDE. Every TypesScript file is put through a series of tests, one of which is a lint test. This is a test to see if the file's contents are maintained by coding standards set up by the lint document. By making use of the TSLint tool TSLint Website we can set up VScode to display linting errors as errors a compiler would normally show, thus we are able to test and ensure coding standards across our system.

```
//demo

if(this.isLoaded)
{
  console.log("hello world");
}
else
{
  console.log("goodbye world");
}
```

```
// demo

if (this.isLoaded) {
  console.log('hello world');
} else {
  console.log('goodbye world');
}
```

Above left is an example of what a developer will see if they develop using the wrong coding standards. Above right is an example of what the developer will see if they use the correct coding standards using VScode as the IDE.

A partial List of our coding standards:

- Enforce return statements in getters
- Disallow assignment operators in conditional expressions
- Disallow constant expressions in conditions
- Disallow duplicate arguments in function definitions
- Disallow empty block statements
- Disallow irregular whitespace outside of strings and comments
- Disallow unreachable code after return, throw, continue, and break statements
- Enforce the use of variables within the scope they are defined
- Enforce that class methods utilize this
- Require default cases in switch statements
- Require the use of === and !==
- Disallow the use of eval()
- Disallow fallthrough of case statements
- Disallow new operators with the Function object
- Disallow new operators with the String, Number, and Boolean objects

- Disallow variable redeclaration
- Disallow javascript: urls
- Disallow assignments where both sides are exactly the same
- Disallow comparisons where both sides are exactly the same
- Enforce consistent brace style for blocks
- Require newline at the end of files
- Require newline at the end of files
- Enforce the consistent use of either function declarations or expressions
- Disallow trailing whitespace at the end of lines
- Enforce consistent line breaks inside braces
- Require semicolons instead of ASI
- Require let or const instead of var
- Enforce max characters per line
- Enforce the consistent use of single quotes in JSX attributes

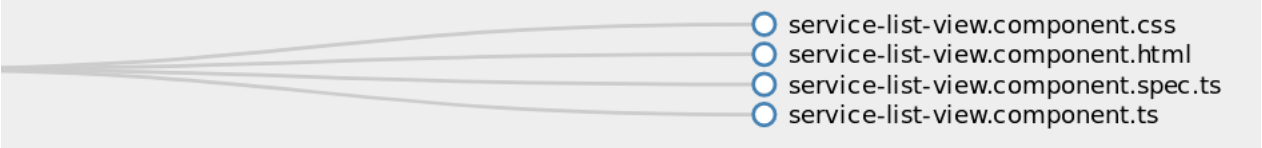
These are the coding standards for all the Typescript files in the front end and are enforced by lint testing. Any git commit is first run through a lint test to ensure that the code is written according to these standards layed out. There are more see ESlint rules.
For formatting we make use of the Prettier tool (view site) which helps remove blank wasted spaces and enforces a consistent level of indentation across all HTML, Typescript, and CSS pages.

2.3 Commenting practices

Comments will be placed above complex functions or above service return function's. Any function that returns data should have a comment describing what the data type is of the data returned, unless it is a simple get function. TODO:'s + (+ developers name +) must be placed by a developer who has knowledge in a specific area of code. This is not a promise that the developer know's everything but rather used as a temporary marker left by a developer to say that they will get back to or that they are stuck and require some help or possibly even to pass on the task to someone else. The developer should be able to provide other members with input about that piece of code at a later stage.
Enforce a space between comment expression and comment

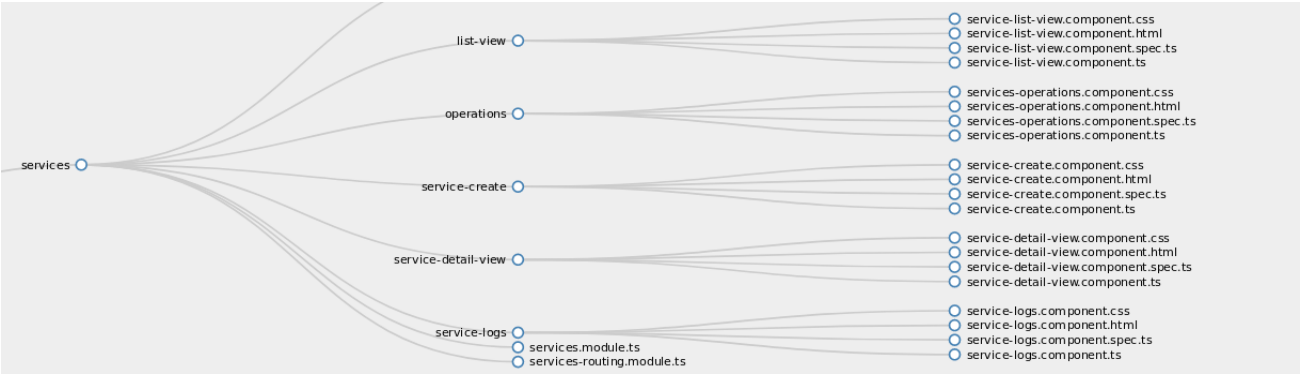
3 FIle Structure

As mentioned at an earlier stage in this document our project makes use of Angular and Components. A single Component is made up of 4 files

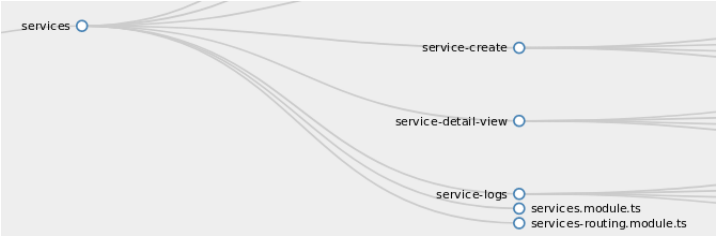


- A CSS file
- A HTML file
- A Spec file
- A Typescript file

We group all Components that share a type ie. Services together with one another in a directory of that type ie. Services



In each of these directories are two very specific Modules. The first is a module used to import all the components that the all the other components of type "Service" –for example– needs. This module has the same name as the directory. The second is a routing module, the module that sets up the links between the module and how to navigate the router to each component in that module.

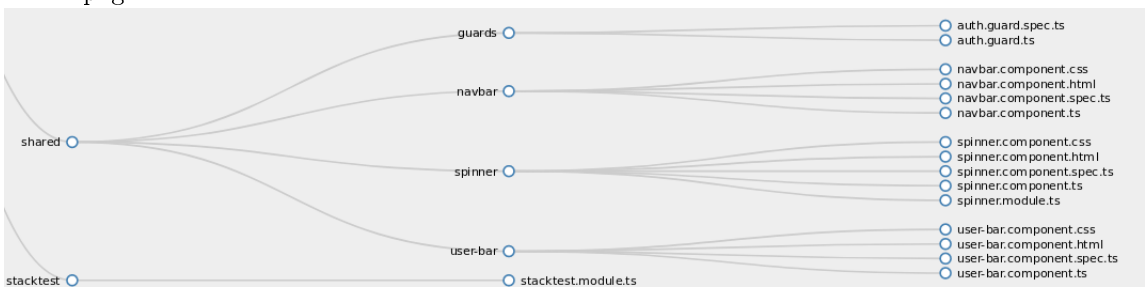


(this is a close up on the previous image)

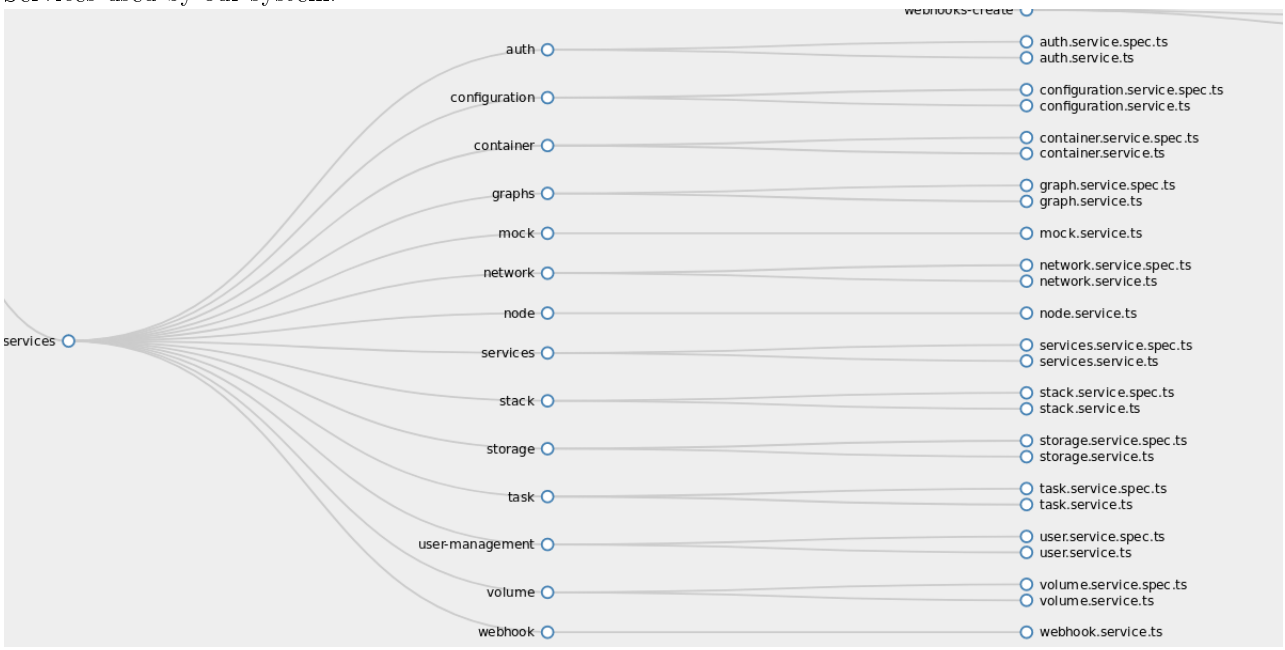
There are 13 directories in the "pages" directory. The pages directory houses all the components that are displayed as a page in the interface. These include:

- home
- login
- networks
- nodes
- page-not-found
- stacks
- services
- tasks
- volumes
- user-management
- webhooks

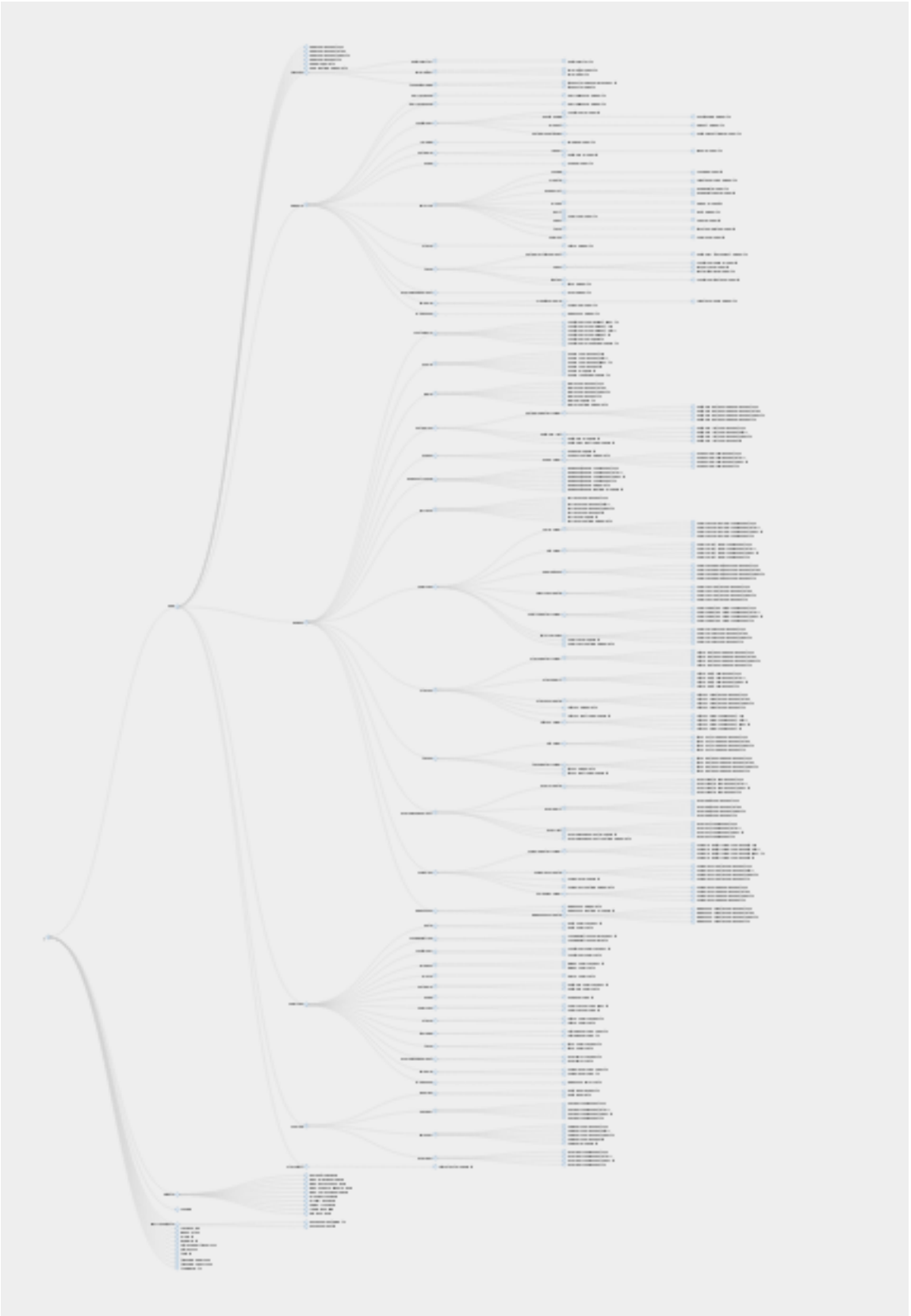
A sibling to the pages directory is a the shared directory which contains components that are likely to be used across pages ie. the navbar



Other siblings of the pages directory include the Module and Service (Not to get confused with the service component, services are used in Angular for requesting services) directories which contain all Modules and Services used by our system.



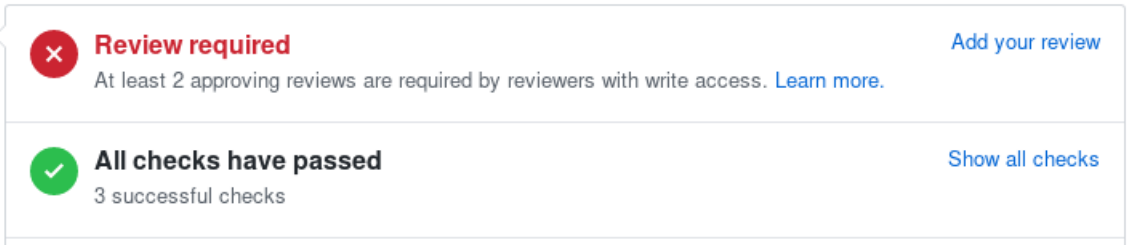
A complete view of our system can be seen on the next page



4 Code reviews

4.1 Peer Reviews

We make use of multiple peer reviews before merging any new code into git. A minimum of two group members must review code before it makes it's way into production. All new feature branches created by group members have to pass a series of travis build tests as well as linting tests mentioned earlier. After that they still have to be reviewed by peers before being merged into our develop branch.



Besides peer reviews we also make use of lint testing as mentioned above which regularly monitors the coding standards of each commit. Commits that fail lint do not pass the tests and cannot be merged into develop.

5 UML diagram

5.1 Docks UML

With a system this large there are not many tools we can use to create an accurate uml diagram. The following is the best we could find.