Constructor University Bremen

Frontend Requirements

Spring Semester 2025

Author:

Nikolay Lachezarov Tsonev

Contents

1 Motive

The Purpose of this document is to be the foundation for the frontend team. This document will include: requirements of our software, The objective, productivity criteria, standard and more.

2 Productivity

Each member is expected to work on average of 10 hours a week. That does not mean that every week it will be exactly 10 hours but that by the end of this project, each team member will have contributed 1200 hours.
These 10 hours will mainly consist of programming, but also will include: meetings, designing the User Interface (UI) and the User Experience (UX). The delegation of tasks will be decided on the weekly meetings and will be kept track of on a **Google Sheet** to make it easier for reporting later on. Another criteria will be the activity on github via **commits**, **pull requests** (PR) and **reviews**.

2.1 PR Approval Criteria

Pull requests must be peer reviewed, meaning that anyone from team can approve a pull request on the **dev** branch. Any direct commits to master apart from exceptions are forbidden and must be done by the team lead.

Criteria the PR must adhere to in order to be accepted

- No Merge conflicts.
- The PR must be reviewed and approved by at least one other team member.
- The PR should include a clear description of the changes made.
- Cannot have more than one **component** per file.
- Any new features must include appropriate documentation.
- Must adhere to the font, color pallet and website style that has been chosen.

2.2 Commit message prefix

This is so that github actions can automatically close issues.

Commit Type	**Issue Type**	**Github Action**	**Use Case**
feat	<feature>	close <issue nr>	Introduction of a new feature.
refactor	<feature>	close <issue nr>	Changing the implementation of an existing feature.
fix	<feature> / <bug>	close <issue nr>	Fixing a feature or a bug.
tests	<feature>	close <issue nr>	Adding / removing / changing tests.
docs	x	x	Documentation.
misc	x	x	Miscellaneous (renaming of files, removing junk, etc.).

3 Software Requirements

3.1 Accesability

The final result should be viable on The following web browser:

1. Google Chrome
2. Mozilla Firefox
3. Safari
4. Microsoft Edge
5. Opera
6. Mobile web browser (**Optional**)

This does not mean that the final result will not be able to run on other web browser, however it ensures that we give the best UX for the most popular web browsers as all of them have their own formatting and layouts.

3.2 UX

The final product should be intuitive, simple and available for groups of all ages and different backgrounds. The means that the team should consider users with conditions that make it more challenging for them to use our software and create solutions that accommodate their needs.

3.3 Security and Privacy

The end result should encrypt sensitive data sent to the backend to prevent sniffing attacks and should not show sensitive information to unauthorised users.

4 Software Architecture

4.1 Programming Language and Frameworks

The main programming language for this product is going to be **Typescript** which is a version of javascript with statically inferred data types. The framework for the frontend shall be **React** because of its modularity and allowing the team to write modular components that can be reused and modifying saving a lot of time writing boiler plate code.

For styling the website, the team will use CSS with the specific framework being decided based on the web style the team chooses.

4.2 Naming convention

Component files

Component files should have the same name as the component itself. For example if you are creating a button, the name of the file would be **Button.tsx**, or if you are making a signing form **Signing-form.tsx**. Note that all component files must start with upper case and if it has more than one word, it must be separated with a hyphen (-).

Page files

main pages such as index, main, login, dashboard, etc... follow the same naming convention as for component files.

Utility files

These are files with helper functions, or other helper attributes like a colour enum, etc... have a utils suffix. For example if you need utils for the homepage the file name would be **home.utils.ts**. Note that these files are all lower case, unless it is an excpetion the like the word ID.

Styles files

Style file with describe how the components or pages will look follow the same convention where the prefix is the component name or page followed by styles. So if there is a styles page for **Home.txs**, the styles file is going to be called **Home.styles.ts**.

4.3 Functions and variables

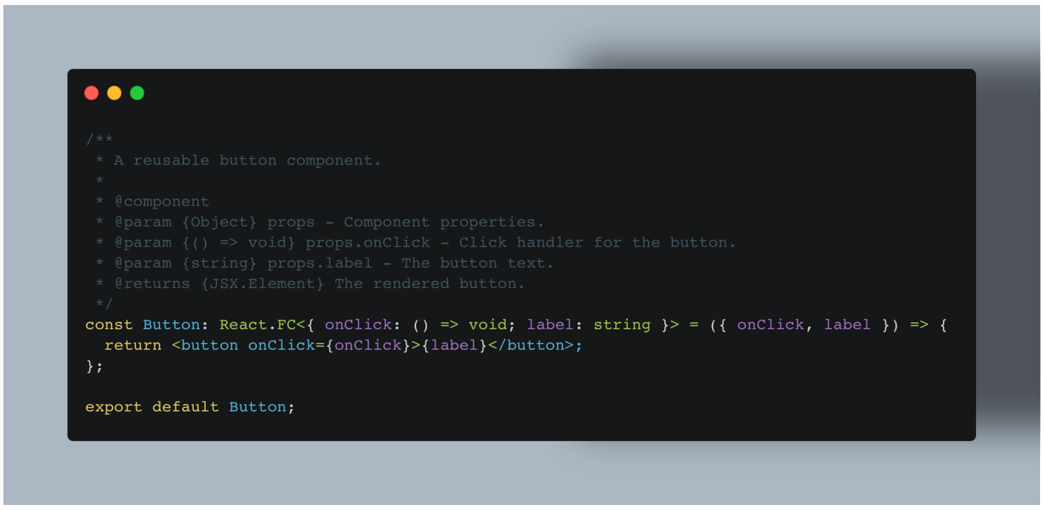
Functions and variables are all going to use a snake case. For example the would be called ”const user_login: number = (user: string, user_password: string) =¿ { ... }”.
Note that global variables must be written in all caps and in snake case. So for example the width of the website would be declared as ”const WEBSITE_WIDTH: number = 1280;”.

5 Commenting and Documentation

For documentation, the **TypeDoc** library will be used that is based on **JSDoc** our commenting standard. This library will check the comments of each function and output an appropriate HTML file.

5.1 Commenting

Comments should be on top of functions giving a brief description of what the function does. Afterwards it should include the what the function is, parameters and the return type as seen below.



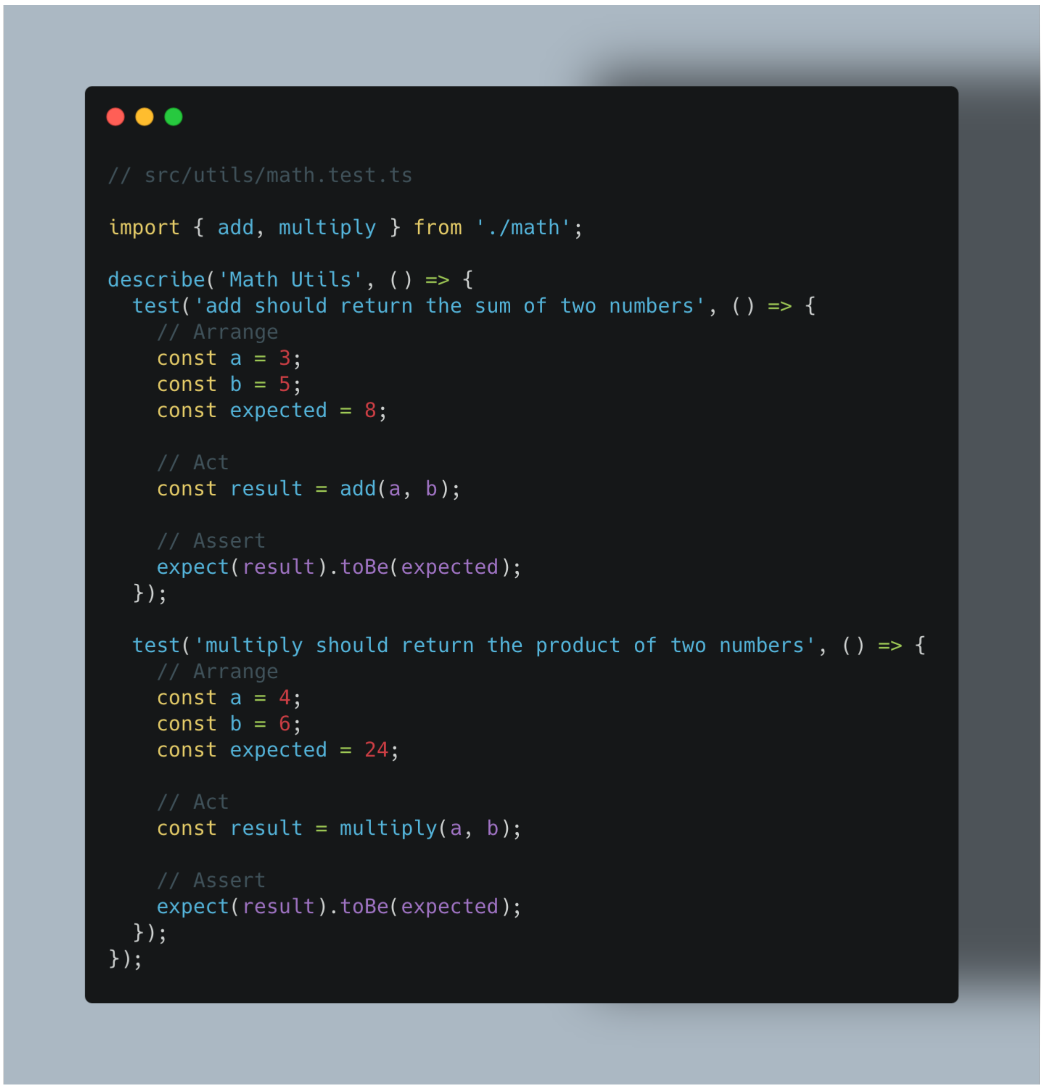
```tsx
/**
 * A reusable button component.
 *
 * @component
 * @param {Object} props - Component properties.
 * @param {() => void} props.onClick - Click handler for the button.
 * @param {string} props.label - The button text.
 * @returns {JSX.Element} The rendered button.
 */
const Button: React.FC<{ onClick: () => void; label: string }> = ({ onClick, label }) => {
  return <button onClick={onClick}>{label}</button>;
};

export default Button;
```

6 Testing

Because Components cannot be directly tested. The team will keep a spreadsheet with each component and on which platform it works and if it does not work, what the issue is. This is so that the team can create new github issues and bug fixes.

Any member writing utils must also provide an appropriate test file with unit test for each new util implemented. Ideally the unit test has been written before the util. Unit tests will be done using the **jest** library. If you have a util name **Button.utils.test.ts** and should have code similar to the figure below.



```typescript
// src/utils/math.test.ts

import { add, multiply } from './math';

describe('Math Utils', () => {
  test('add should return the sum of two numbers', () => {
    // Arrange
    const a = 3;
    const b = 5;
    const expected = 8;

    // Act
    const result = add(a, b);

    // Assert
    expect(result).toBe(expected);
  });

  test('multiply should return the product of two numbers', () => {
    // Arrange
    const a = 4;
    const b = 6;
    const expected = 24;

    // Act
    const result = multiply(a, b);

    // Assert
    expect(result).toBe(expected);
  });
});
```