Coding Guidelines

Table of Contents

Front-end

For the front end, it will be developed using React. The testing will be done using cypress. For the testing we will use Test-Driven Development, which means writing tests before creating any pages, functions, or components. This is to prevent last-minute fixes and allow for quick bug-catching.

Base Rules

- Only have one component for each React file.
- Use the JSX coding language.
- For variables, always use “const” or “let” to declare a variable. Make ‘const’ as a default, when you reassign a variable think of using ‘let’.

Naming

- Each file name needs to be identical to the component name.
- PascalCase naming convention for file names and component names, for example, “FooBar”.

JSX alignment

Follow the following styles for JSX syntax:

```
“”
// bad
<Foo superLongParam=’bar’
    anotherSuperLongParam=’baz’ />

// good
<Foo
  superLongParam=’bar’
  anotherSuperLongParam=’baz’
/>

// if props fit in one line, then keep it on the same line
<Foo bar=’bar’ />

// children get indented normally
<Foo
  superLongParam=’bar’
  anotherSuperLongParam=’baz’
>
   <Spazz />
</Foo>
“”
```

Quotes

Use double quotes (“) for JSX attributes. JS only single quotes (‘).

```
// good
<Foo bar='bar' />

// bad
<Foo bar="bar" />

// bad
<Foo style={{ left: "20px" }} />

// good
<Foo style={{ left: '20px' }} />
```

Props

- Always use camelCase for prop names.

```javascript
// bad
<Foo
  UserName=’hello’
  phone_number={12345678}
/>

// good
<Foo
  userName=’hello’
  phoneNumber={12345678}
/>
```

Tags

When there are no children, self-close it.

```
// bad
<Foo className=’stuff’></Foo>

// good
<Foo className=’stuff’ />
```

When the component has multiple lines, close it on a new line.

```
// bad
<Foo
    bar="bar"
    baz="baz" />

// good
<Foo
    bar="bar"
    baz="baz"
/>
```

(Schwanen, n.d.)

Naming conventions

Identifier	Description	Example
Event handlers	Use the prefix “handle”, for event occurring functions.	
CSS classes	Use a lower case and hyphens for CSS class names	
Constants	Use uppercase letters with an underscore (constant case) representing constants in JavaScript.	“API_URL”, “MAX_RESULTS”
Utility functions	camelCase for utility and helper functions	“”” const formatDate (date) => { return formattedDate; };
State variables	Prefix these state variables with **is**, **has,** or **should**, to show that it is a Boolean.	const [isActive, setIsActive] const [hasError, setHasError] const [shouldRender, setShouldRender]
Properties	Have descriptive names for properties, that indicate a purpose.	Bad: usr Good: user
Files	PascalCase	UserCard
Components	PascalCase. Meaningful name for react components.	const ToDoItem = () => {…}

(Velkov, 2023)

Back-End

The backend of the project will be developed in C#, using ASP. Net Core Web API. We will be using MS Test to test our logic. Testing will be approached in a Test-Driven Development. This will mean that we will first write our methods and classes, write failing tests, and try to make them pass. When dealing with complex designs and infrastructure, it's best to create tests beforehand to prevent bugs and issues later.

Coding style

We want to have certain standards when it comes to coding consistency, keeping everything well-maintained and easy to read.

Naming rules

See the following guidelines regarding the naming rules for our project.

Identifier	Description	Example
Interface	Interface will start with a capital `I`. After that PascalCase	"public interface IMyInterface"
Attribute types	End with the word "Attribute"	"public class SimpleAttribute : Attribute"
Enum types	PascalCase.	"public Enum UserTypes"
Enum Member	PascalCase	"TestDomain"
Class	PascalCase	"public class MyClass"
Variables	camelCase	"string employeeName="Test"
Constructor	PascalCase	"Employee(){}"
Method	PascalCase	"void AddEmployee(){}
Parameters in Method	camelCase	"void AddEmployee(string employeeName){}"
Constants	PascalCase	"const int EmployeeNumber=1"
Field	PascalCase	"public string EmplyeeName;"
Struct	PascalCase	"public struct ValueItem"
Property	PascalCase	"public string EmployeeName {get; set;}"
Delegate	PascalCase	"public delegate void DelegateType(string message);
Event	PascalCase	"public event Action EventProcessing;"
Local Functions	PascalCase. A function inside a method.	"static int CountQueueItems() => …;"
Record plus parameters	PascalCase.	"public record Employee(string Name, string City);

Private global variables	Prefixes with an underscore _. camelCasing when naming local variables, including instances of a delegate type.	public class DataService
		{
		  private IWorkerQueue _workerQueue;
		}

Static	All uppercase. Each name needs to be separated with an underscore _.	"public static int WORD_COUNTER=0;"

Tuple element names	PascalCase	(string First, string Last)
Variables declared using tuple syntax	CamelCase	(string first, string last) = (“John”, “Doe”);

(C# identifier naming rules and conventions, 2023)

Avoid single-letter names, except for easy loop counters such as Foreach(var u in users){}. But also for the use of well-known accepted terms in the working domain. Think of writing UI instead of UserInterface, or Id instead of Indentity.

Language

All the identifiers listed in the table above should be named using words from the American English language. This has the following requirements:

- Choosing easily-to-read language, with correct grammar. To give an example: **UserService** sounds better than **ServiceUser** or **HorizontalAllignment** is better to read than **AllignmentHorizontal**.
- Favor reading over brevity: It is better to have an easy-to-understand readable name such as CanScrollHorizontally is better than ScrollableX (not direct/hard-to-understand reference to the X-axis)
- Avoid words that conflict with keywords of widely used programming languages.

General Do’s and Don’ts

The following guides are some general behaviors to properly follow our coding guidelines.

Never use numbers in variables, parameters, and type members.

Do not use prefix fields.
An example of this is when you have a class called “CarEngine” and all the related fields regarding the car will get a prefix, such as “eng_serialNumber” or “eng_model” is not allowed to do.

Don’t repeat the name of a class or enumeration in its members.
See the following example:

```
class Employee
{
        // Wrong!
        static GetEmployee() {...}
        DeleteEmployee() {...}

        // Right.
        static Get() {...}
        Delete() {...}

        // Also correct.
        AddNewJob() {...}
        RegisterForMeeting() {...}
}
```

Figure 1: example naming members inside class

Name your members, parameters, and variables according to their meaning and not their type.
Do not use terms like Enum or Class in a name.
Do not add types in variable names, except to avoid conflicts with other variables.

Have similar member names related to the .NET framework classes.
Help with the same pattern to find for your classes. For example, when having a service with CRUD, use words like Add, Remove, and Count instead of AddItem, Delete, or NumberOfItems.

Properly name properties
Use nouns, Noun Phrases, or occasionally adjective phrases: Choose a descriptive name that clearly indicates the purpose of the property.
Name Boolean Properties that show agreement: For example, use "IsVisible" instead of "NotVisible". Prefixes with terms like 'Has", "Is", "Can", "Allows", or "Supports" to give more clarity to the boolean meaning.

Name namespaces using names, layers, verbs and features
Namespace cannot contain a name of a type.

Ordering in a code file
Proper structure of the file, based on types with what needs to be at the top:

- Constants
- Statics
- Events
- Publics
- Privates

Prefix using "On"
An example can be when you have a button that will close a pop-up, there will be an event, do not call it "CloseButton", instead name it OnCloseButton.
(Naming Guidelines, n.d.)

Comment styling
See the following rules:

- Short, single-line comments use "//".
- Try to avoid multi-line comments "/* *\"
- Describing the function of methods, classes, fields, and public methods will be described with XML comments.
- Comments on a separate line.
- Start a comment with an upper case.
- End comments with a point.
- Insert one space between the comment delimiter "//" and the comment text.

See the example for clarification:
```
""
// The following declaration creates a query. It does not run
// the query.
""
```

Composition over inheritance
Try to use interfaces over an inheritance structure.

Entity Framework Practices

Git

Properly writing a message when making a commit

Here are 7 rules to write a proper message to make a great commit.

1. **Separate the subject from the body with a blank**: keep in mind that some commits are so simple that they just can be committed with one line.
2. **Subject line max 50 characters**: Extra information in the description. This rule is to have a readable subject, so the limit for 50, but 72 is a hard limit.
3. **Capitalize the subject line**: Start the subject with a capital letter.
4. **Do not end the subject with a period**: Not needed, especially when 50 characters or less.
5. **In the subject line, write it as if you are giving a command**, such as “Clean your room”, “Close the door”
6. **Wrap the body at 72 characters**: Git then has plenty of room to add text, keeping the limit at 80 characters overall.
7. **Make clear why changes [not applicable at all commits]:** Focus on why you made changes, the way it worked before, how it works now, and why you decided to do it like this.

(CBEAMS, 2014)

Branch strategy

This branching strategy has multiple branches, consisting of both long-lived and short-lived. All code will be first reviewed and the tests need to pass before approving a merge request. When done with the feature, a review of the branch will be done by someone who didn’t have anything to do with the feature.

Long-lived

Main: The Branch will be only for stable production, used for releases. Only merged from development.
Development: This is used for all feature branches to build from.

Short-lived

Feature: Used for different types of features we are working on. When done, can be merged into development.

Diagram

To give further clarification, see the following diagram to get more clear in-depth information on how the strategy works.

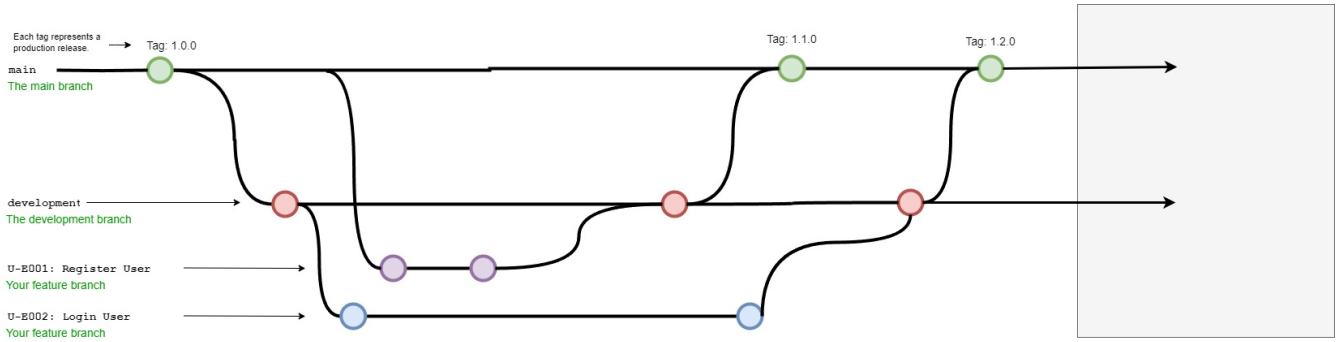

Figure 2: Branching Strategy

Reference

C# identifier naming rules and conventions. (2023, 12 15). Retrieved from learn.microsoft: https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/identifier-names

CBEAMS. (2014, 08 31). *How to Write a Git Commit Message*. Retrieved from cbea: https://cbea.ms/git-commit/

Naming Guidelines. (n.d.). Retrieved from csharpcodingguidelines: https://csharpcodingguidelines.com/naming-guidelines/#:~:text=Always%20prefix%20type%20parameter%20names,type%20parameter%20in%20that%20case.

Schwanen, O. (n.d.). *Coding Guidelines - ReactJS*. Retrieved from github: https://github.com/pillarstudio/standards/blob/master/reactjs-guidelines.md

Velkov, K. (2023, 06 18). *React JS - Naming convention*. Retrieved from linkedin: https://www.linkedin.com/pulse/react-js-naming-convention-kristiyan-velkov/