CODE OF CONDUCT

Interactive Lessons Code of Conduct

Elise

In this document you will find the rules of coding in this project
Also former known as the code of conduct

Elise

Inhoud

Introduction

First words

This document contains the code of conduct. This coding rules will be applied by everyone to keep the code fresh and clean. Also by applying this conduct we have a code that is easy and quick to read by everyone. This prevents confusion and lost time by people who want to contribute to the project.

What happened if I don’t apply to the conduct?

Everyone can make a branch with a name in the project. This also means that everyone can code and contribute to the project. Only what happens if you don’t apply. Depends on how badly the ignore treatment is. So lets explain how it works

I changed a few names in regards of the conduct

As a person who might have dyslexia never researched. I can understand that it`s sometimes hard to check every word you spell. So if it happens that you miss spell a few names that it wont be punished. As long you keep check to the ruling names and people can understand it.

I still applied to the ruling but named it different.

I wont make it that hard. For consistency I just ask you to rename the functions and names to the conduct rules. Just to make it more easy to read and understandable for everyone. I know it sometimes hard to keep an eye on but, try to read after your names. Once your done coding your part. There wont be an allowance where we see it through the finger. Just to prevent people from going like. He/She did it why can’t we do it then

None of the names match the conduct.

In this case is it simple. The merge wont be allowed unless you change it. This to keep intact with the same reason as above. To keep it simple and easy for everyone. This make it also easier to read and understandable for people who are new and want to conduct to the project.

Repeated offense of the conduct can lead to a permanent ban on the project. So keep in mind to read the conduct carefully and always double check your code.

Readers guide

No special readers guide present for this document

Code of conduct

Naming of the branches rules

As first I want to start with how branches should be named if you want to contribute. This with the primary reason that we prevent people from working on the same thing. Now comes the question how do we name new branches.

First is useful to know that if you want a stable working build, that its an idea to keep in mind. I highly recommend making branches with names like: **what_your_making_Name.** So for example if I would make a lesson overview for the teacher then it would be: **getAllLesson_forTheTeacher_Elise** Keep in mind to keep your names as clear as possible. This to prevent confusion amongst the contributors. Keep in mind to use a name like: whatYourMaking_Name or what_your_making_Name

Pushing to branches rules

Done with coding something

I'm also going to apply some rules to pushing to branches. If you made something in your own named branch you can first push it to the Testing branch. From here a CI pipeline will run. After the CI pipeline is done one of the 2 scenarios can occur:

1. The ci pipeline will fail. In this case recheck the code and make sure all the given test are running without any issues. Make sure to fix these issues and then **again commit to this branch**
2. The ci pipeline succeeds. In this case your code can be pushed to the Testing branch. Please keep in mind that before you push to this branch to check for merge error`s and solve them. **In case your code causes a merge error with code from another contributor then contact the person and work it out. Never just use your version. Always contact each other and talk what is the right version to use**
3. This part only doesn't apply if your making something new or reworking a specific part. In that case the rule drops since your upgrading a specific part. For example if I upgraded the login with JWT tokens then the rule doesn't apply. Make sure to use the naming rule to avoid conflicts.

Pushing from Testing -> Main

Pushing from the testing branch to the "Main" branch is done from time to time. Were working with the X.Y.Z release notes. There comes an point where we done coding for a release or a patch in the future. From the patch to minor release all the codes need to best tested all the way through. This when we decided to push the "Beta/Alpaha" Branch or called Main.

The main branch holds several CI/CD pipelines for each separate planned project. The idea is that first when a push is made to this branch that the CI pipelines. These pipelines will check to see if the application can build and the test are running. Once these passed the CD pipelines will run

The idea for the CD pipelines is to build the application and deployed it to an test environment. From here a test group can test the project in an online environment. From here we have a beta environment. The test group can here do what they want and find bugs and issues. They can submit them by using the bug/issues reports templates

Its good to know that if one of the pipelines fails that the code will be send back to the Testing branch. In this case it prior to look back at the errors and fix them for once again pushing it back to the Main beta branch. Also this branch holds several security measures as a side note.

Pushing to the Master branch
Strictly prohibited and not possible. Since this branch is a protected branch.

Naming Tests rules
For naming test I want to apply the following rules in naming. **MethodName_StateUnderTest_ExpectedBehaviour** This will result that everyone can understand tests more easily and knows what is to be expected of them. As example this could be a few test names in the applications:

1. CanUploadALesson_AsStudent_False
2. StartALesson_AsStudent_True
3. CreateANewLesson_AsAssitingTeacher_ExpectionThrown
4. TryingToAccesAUnfinsihedMethod_AsAAdmin_ExpectionThrown

Running test rules
Make sure to always check and run your test after adding or updating something in the project. What I mean with running is that your test pass. Passing in the sentence of the test lits up green. This can also happens with test that fail or expect a behavior

Keep in mind that a failed test means no push towards the testing branch or main branch. This so we can keep a clean and working test environment on these branches.

Coding
Coding is going to be divided into smaller different sections. Reason for this is that there is planned for 2 different projects at least. One Front-end and One back-end. These will be written in a different coding languages. So there will be 2 different rules set for the planned project.

Language in coding
In this project we use the English language in coding. This to make it easier read for everyone and understand it better. Also as said earlier I want to keep this project consistent across the entire development cycle. So that primarily the reason why I choose for the English language in the project.

Front-end
The front-end is planned to be build in a React project. React is a front-end driven tool that primarily uses JavaScript in its environment. This is why most the rules for the front-end are based on coding conventions for react front-end framework. The version used for the project: *TODO: Enter version here*

https://www.jondjones.com/frontend/react/react-tutorials/react-coding-standards-and-practices-to-level-up-your-code/

The link is an example on what the conventions are. I'm not going to name them all but if you want to read through them. Its given the idea for our conventions.

Naming conventions

Component`s naming
Component`s names go in pascal styled casing. This mean each word in your naming starts with a capital letter. A few examples are given here below:

1. CookieBanner.js

2. Header.js
3. LessonViewer.js
4. BannerHeaderTemplate.js

Non-Component`s naming

For the non-component`s file were using camel casing. This means that the first word is written complete in lower case. Every different word after starts with a capital letter. A few examples here are:

1. dashboardLogger.js
2. fetchUserData.js
3. doLogin.js
4. cookieHelper.js

Attribute Naming

Attributes are considered to be named also as camel casing. This means as said earlier: you start with the first word complete lower cased. After this each new word starts a capital letter. A few examples are:

1. className
2. onClick
3. onMouseHover
4. multipleWordAttributeId

Inline styling

First and foremost I want to discourage to use inline styling as much as possible. This with the reason to keep the code as clean as possible. In case its needed to put styling in the html directly I still want to advise you too look if its possible to move it to the CSS file.

In case its only fits in the inline html. Then there will be a naming rule to apply. Inline styling is written in camel casing once again. Here are a few examples to be given:

1. style=”font-size: 12px;”
2. style=”display: inline-block;”

Variables naming

Variables the buckets of information in the code. Variables are also named with camel casing style. Be mindful variables are accepted with numbers and specials characters. Let’s give you a few examples to start with:

1. const testVariable = ‘test’;
2. let booleanVar = true;
3. int numberCounter = 5;
4. string username = “Klaas123”

CSS file naming

CSS files are also going to have a naming rule. CSS files should always be named as their component counterpart. So lets give you a few examples:

1. CookieBanner.js and CookieBanner.css
2. Header.js and Header.css
3. Footer.js and Footer.css

4. StickeyMenu.js and StickeyMenu.css

Unit testing naming

Unit testing should always be named after their corresponding files that they are testing. To give you a few examples:

1. CookieBanner.js and CookieBanner.test.js
2. fetchAPIData.js and fetchAPIData.test.js
3. loginUserThroughAPI.js and loginUserThroughAPI.test.js
4. Header.js and Header.test.js

Last few rules.

There are 2 more rules too keep track of in the project. This is for naming conventions. The 2 rules are as follows:

1. If a component requires multiple files (test,css) then locate all these files with in the same component folder. So for example a folder called Header with Header.js, Header.css and Header.test.js
2. File extensions for components are .jsx and .tsx heading down for javascript or typescript.

Coding conventions

Coming soon

Comments in coding

Coming soon

Back-end

https://www.freecodecamp.org/news/backend-for-react-projects/

https://console.firebase.google.com/

Coming soon

Naming conventions

Coming Soon

Coding conventions

Coming soon

Comments in coding

Test