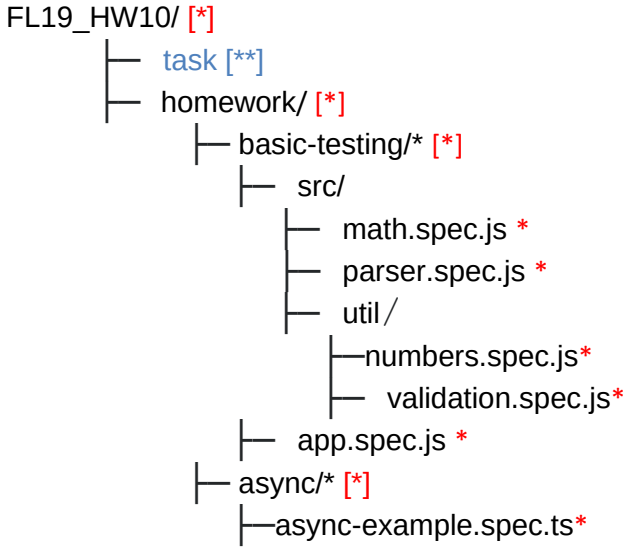
JS Unit Testing

DEADLINE: 04/07/2022

FOLDER STRUCTURE



[*] - required
[**] - restricted (no need to submit)

TASK

1. Go to the task folder and see two packages - basic-testing and async (testing of async functionality), in both these packages are package.json, there are some dependencies that we would need to write our tests. Each package are unrelated simple apps, and for both you would need to do same things to able to write and run the tests. So, for running each of the application write ***npm install*** in your console. After all the dependencies are installed you are able to write and run the tests.
   NOTE: We are going to use a *Vitest* framework (to read the documentation see *Useful links*). It is similar to *Jest* but has a better support of ES6 JS Modules.

2. Let's start from the basic testing. If you open *basic-testing/app.js* file you will see a function which handles a submit button and few other functions that are imported as a separate JS modules from different files such as *math.js*, *numbers.js* and *validation.js.*

3. Create a test files for each file like *math.spec.js*.

4. Start from these functions and try to test not only the positive scenarios but something that could make you function fail. You could change a bit an existing functions to be more safe (add some checks if needed).

5. Go to the *app.js* and see that *formSubmitHandler* function is not a good function because does a lot of things. Try to split it into another functions and test it in *app.spec.js.*
6. Now go to async folder and figure out how to test an asynchronous code.
   NOTE: If you see that test is green and you do testing in same way as for synchronous code - it does not mean you did everything well. It means that test is not even waiting for the *expect* statement at all. And if test does not have an expected assertion it looks like green.
7. In async folder is one more folder which is hooks, it is not a separate app and you are able to use already installed dependencies. Learn about hooks in testing and thing how you would write the tests for simple User class and optimize testing code using hooks.

FYI

1. You might ask why functions separated to so small peaces but the idea behind unit testing is not only to test a functionality but write the code that would be easy to cover by tests. Otherwise how would you cover by test a huge function with a lot of dependencies and inner functions if one of the principle of testing sound like «*Test should cover only one thing*».

BEFORE SUBMIT

- Code should be clean, readable, and tested
- Verify that all functionality is implemented according to the requirements.
- Run the linter and fix all the warnings and errors.
- Make sure your GitLab folder structure meets folder structure from this document (without **task** folder)

SUBMIT

- The **FL19_HW10** folder without **task** folder should be uploaded to GitLab repository "**FL-19**" into **main** branch.

USEFUL LINKS

- https://vitest.dev/