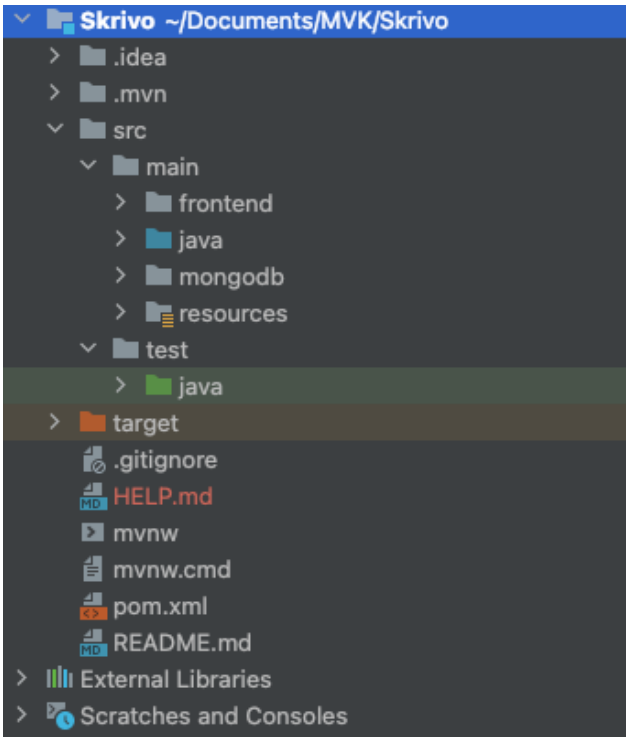
How to contribute

March 10, 2022

1 Folder structure



The project follows a standard folder structure for java projects, with a main folder and test folder inside of the src folder. The java code is written in src/main/java, and the tests for the java code is written in src/test/java. Database files are placed in src/main/mongodb, and after looking at various Spring Boot projects using React on the frontend, we decided to follow common practice and place the frontend code in src/main/frontend.

2 Best practices

GitHub Issues

- Assign yourself to an issue before working on it.

- Create a new issue for every new feature, bug fix, etc..
- Create clear and concise titles.
- Label all issues (e.g. enhancement for new features, bug for bug fixes, etc.).
- Keep issues small; they should be related to only one problem.

Branches

- Create a new branch for each issue. If the branch is working on issue#36, it should be named issue#36.

Commits

- Commit regularly and often. A commit should be related to one change that has been made.
- Write commit messages for each file that has been changed, try to avoid git add ..
- Use present or active tense.
- Write clear and concise commit messages that describe what has been done.
- Prefix each commit message with feat: for new features, refactor: for refactoring, doc: for documentation, bug: for bug fixes, test: for new tests, etc..

Pull Requests

- Always resolve your own merge conflicts.

- The pull request should clearly describe what has been changed.
- Assign yourself to a PR if you are reviewing it. Try to only review PRs within your area of competence. So, if you work on the frontend, review PRs that are related to the frontend and so on.
- Always end a PR with the keyword closes, so for instance "Closes #8" which would automatically close issue #8.
- Close PRs with the comment LGTM if you are merging it. Otherwise describe the problem that has to be fixed before you can merge the PR as a comment, and reach out to the person that created the PR on discord and ask them to have a look at the comment.
- Prioritize PRs to coding so that they don't pile up.
- Use squash and merge when merging. The merge commit should have the following format: "X (#Num): Concise description" where X is a prefix (e.g. feat, bug or test) and #Num is the issue that's being closed (e.g. #10).

3 Maintainability of the code

Skrivo will continue to develop the project when the course is over. Therefore, it is important to write clean code, so that the code is easier to maintain and enhance. Below we describe the best practices we will follow when writing our code. First, we describe the most important general rules to follow. Then we include other sections describing these rules in more detail.

General rules

- The code should run all tests.
- Minimize duplication. If more than one function is using a piece of code that is doing the same thing, use the Extract Method refactoring tool to create a helper function. When appropriate, use inheritance.

- Write descriptive names.
- Functions and classes should do one thing, and hence be small.
- Classes and functions should have concise and descriptive comments that describe the intent of that class or function. Smaller helper functions do not need to be commented.
- Adhere to the MVC design pattern, described in the section "Software Architecture".
- Be consistent. If one thing is done in a certain way, similar things should be done in the same way.

Names

- Replace numbers with named constants (variables).
- Choose descriptive and unambiguous names.
- Prefer longer to shorter names if it makes the code more understandable.

Functions

- Keep functions small.
- Functions should do one thing. If it is doing more than one thing and is becoming big, use the Extract Method refactoring tool to create one method that more clearly describes what it does, and submethods saying how it is done.

Comments

- Always try to explain yourself in code. Prefer writing clean code to commenting the code to explain what it does.

- When commenting functions and classes, use comments as explanation of intent and clarification of code.
- Keep comments short and concise, so don't be redundant and add noise.

Code structure

- Dependent functions should be close to each other.
- If a function has submethods, place the functions in the downward direction, with the method on top of the submethods.
- Declare variables close to their usage.
- Use white space to associate related code and disassociate weakly related code.

Tests

- A test should only have one assert!
- Make them readable.
- Make them independent of other tests.
- Write them in a way that makes them repeatable (they can be copy-pasted to create new tests).