Rules and conventions

This document will describe rules that MUST be followed by contributors to this project during development and coding. The document Code of Conduct focuses more on rules regarding the behavior of Contributors.

Code contribution workflow

Contributions should follow the correct order of steps. This way the project development will remain of consistent structure and workflow.

1. List out tasks from the backlog as issues.
2. Add any newly appeared issues (missing features, bugs, performance related etc.)
3. Branch out from the **correct branch** and use the branch naming convention specified below.
4. Open a work in progress **Merge Request** and request reviews from other developers.
5. Commit your changes in a controlled and well-manner. Use the convention for naming your commits.
6. After reviewing the changes, merge your request

GitHub projects integration (for repository members): it is of great importance to utilize the GitHub Projects tool when working in this project. Use the provided columns and set assignees to the cards added in the Project. Cards are added via issues or created manually to encapsulate a specific requirement.

Issues conventions

Issues are a great way to contribute to the project and to keep track of its future updates.

1. The issues must follow the naming convention `[<TYPE>] <Title>`:
    - The `[<TYPE>]` can be one of the following:
        - `[FEATURE]` - Feature that should be implemented
        - `[BUG]` - Bug that occurs
        - `[DOCS]` - Documentation issue in this repository
        - `[STRUCTURE]` - Issue related to the file/logic structure of the project
        - `[DESIGN]` - Design issue with the website

        - `[REPO]` - Repository fixes
    - The `<Title>` should begin with an uppercase letter and briefly describe the issue
2. The body of an issue must follow the provided templates or should adhere to their structure iff there is no template available for this type of issue.

Upon creating an issue, the author must assign it to a repository developer unless the author is an outside contributor and label the issue with the appropriate labels. Spamming issues or in any other way breaking the Code of Conduct may result in a ban or mute.

Branch conventions

1. **Branches follow the below-specified logic:**

Instance	Branch	Branches from	Accepts from	Description
Stable	`stable`	-	Working & Hotfixes	Branch for stable code, deployed to production. Interacted with only on well-tested updates.
Working	`main`	-	Features, Issues & Hotfixes	The main working branch that encapsulates current development. Accepts `features` and `hotfix` merges. Issues on this branch must be solved before publishing to `stable`
Features Issues	`topic-*`	main	-	Used when developing a new feature or an enhancement OR a bug fix that could be implemented on a later stage
Hotfixes	`hotfix-*`	stable	-	Fixes that must be implemented immediately upon

   Most of the branch conventions are taken from the branching standards & conventions gist.

2. **Branches must follow the following naming conventions:**

    - `stable` - always represents the latest code deployed to production

- `main` - always reflects a state with the latest delivered development changes for the next release
  - a developer branches and merges from `main`
- `<issue number>-feature-<name>` - logically isolated changes related to a specific feature or user story
- `<issue number>-bugfix-<name>` - changes related to fixing a specific bug
- `<(optional)issue_number>-hotfix-<name>` - changes related to fixing a critical bug in the production environment

Commit conventions

Commits must abide the following set of rules:

1. **Commit messages must follow the Conventional Commits standards**.

```
<type>[optional scope]: <description>

[optional body]

[optional footer(s)]
```

Here are some additional rules that abide to the original ones but are focused on this repository:

- The `<type>` keyword could be one of the following **nouns**:
  - `feat` - a new feature is introduced in the changes
  - `fix` - a bug fix has occurred
  - `chore` - changes that do not relate to a fix or feature and do not modify files
  - `refactor` - the changes refactor code (neither fixes a bug nor adds a feature)
  - `docs` - changes the documentation (both in `.md` files and inline code documentation)
  - `style` - changes that do not affect the meaning or behavior of the code (formatting)
  - `test` - including new or correcting old tests
  - `ci` - continuous integration related
  - `build` - changes that affect the build system or external dependencies
  - `revert` - reverts a previous commit

- The `[optional scope]` can be one of the following (or if reasonable a new one). Keep in mind that points to what the commit is focused on and not all changes must be in this specific scope (you may also do small changes in other files too):
  - `routes` - general changes specifically in the `/src/routes` folder
  - `tests` - general changes specifically in the `/tests` folder
  - `lib` - general changes specifically in `/src/lib` folder (usually introducing new components)
  - `website` - changes that are related both to `routes` and to `lib` (usually general website additions)
  - `<component_name>` - changes that are specifically focused on a certain component
  - `root` - changes made at the `/` folder (usually documentation updates)
- The `<description>` must be in lower-case and in imperative

2. **Commits must be**
   - **focused** - a coherent change to the system is aggregated by commits that only affect a small number of files.
   - **isolated** - each commit should ideally contain changes that are related to a single feature, bug fix, or refactoring task

Merge request conventions

1. Merge requests MUST adhere to the branch merging conventions specified above.
2. Merge requests MUST follow the `pull_request_template.md` and provide descriptive change-log
3. Branches may only be merged into `stable` iff they do not break it.

Comments & Reviews

Comments

1. **Comments must follow the rules written in the Code of Conduct.**
2. Comments must not be spam messages.
3. Comments may include meme references.

Reviews

1. Merge requests must be reviewed by at least one developer before being merged
2. Reviews MUST be descriptive and **MUST NOT criticize the developer in any nonconstructive or derogatory way.**
3. Critics should be well-argued and provoke discussion rather than insulting the developer.