

# Contributing to Visual Music

## GitHub and PR Guide



### Difference between cloning and forking:

*Forking* a repository is creating a copy of the project repository you want to work on, under the GitHub Id:

- Any changes made to the original repository will be reflected back to your forked repository.
- Any changes made to your forked repository will require that you explicitly create a pull request to the original repository. Once your pull request is approved by the administrator of the original repository, your changes will be merged with the existing original code-base. Until then, your changes will be reflected only in the copy you forked.

*Cloning* a repository means having a proper duplication on your own machine.

- When you clone a GitHub repository on your local workstation, you cannot contribute back to the upstream repository, unless you are explicitly declared as "contributor". That's because your clone is a separate instance of that project.
  - If you want to keep a link with the original repository (also called *upstream*), you need to add a remote referring that original repository.
-

# Contributor's Workflow



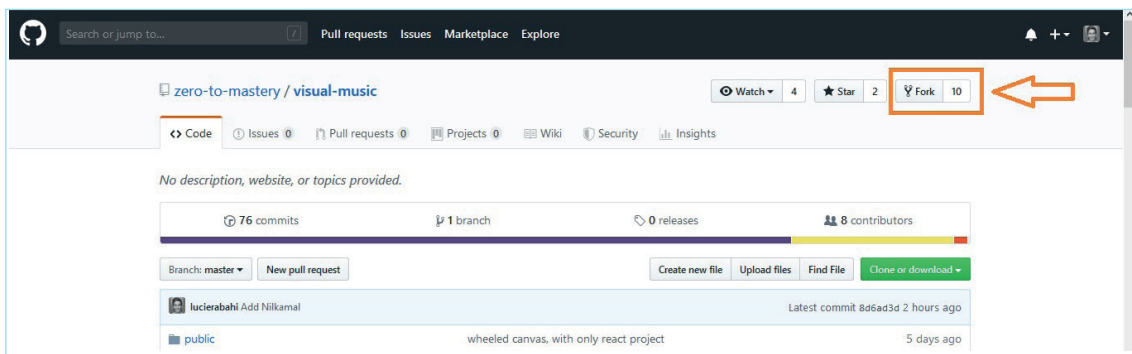
## Getting started:

### Fork

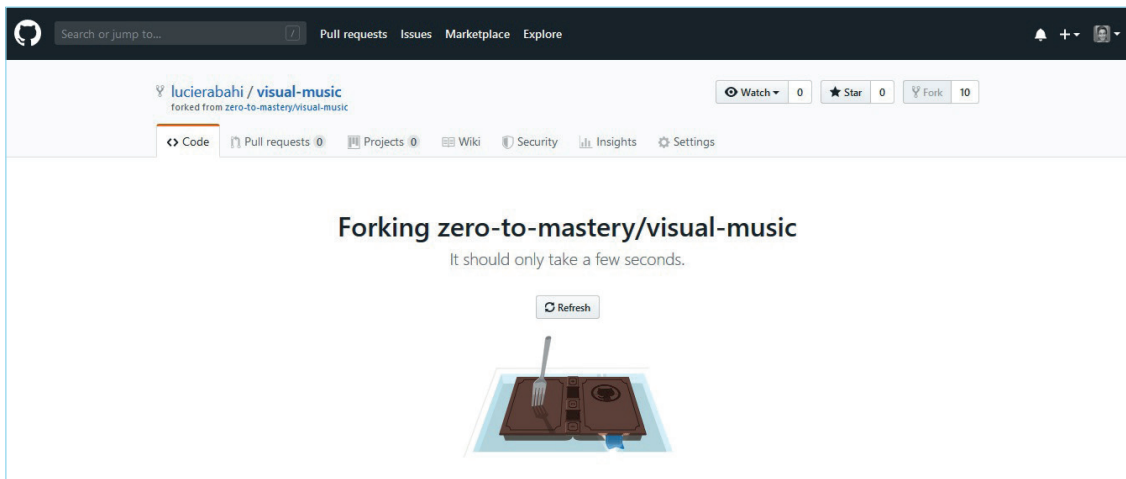
Go to Visual Music GitHub repository on zero-to-mastery:

**<https://github.com/zero-to-mastery/visual-music>**

In the top-right corner of the page, click *Fork*:

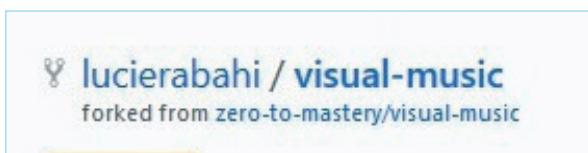


You should see this screen:



That's it!

You have a fork of the original **zero-to-mastery/visual-music** repository:



Now you should have your own public repository which contains exactly the same history as the main repository at the time you forked. You will later push your contributions into this repository and the maintainers of the main repository will pull your commits into the main branch.



## Keep your forked synced:

It's good practice to regularly sync your fork with the upstream repository. To do this, you'll need to use Git on the command line.

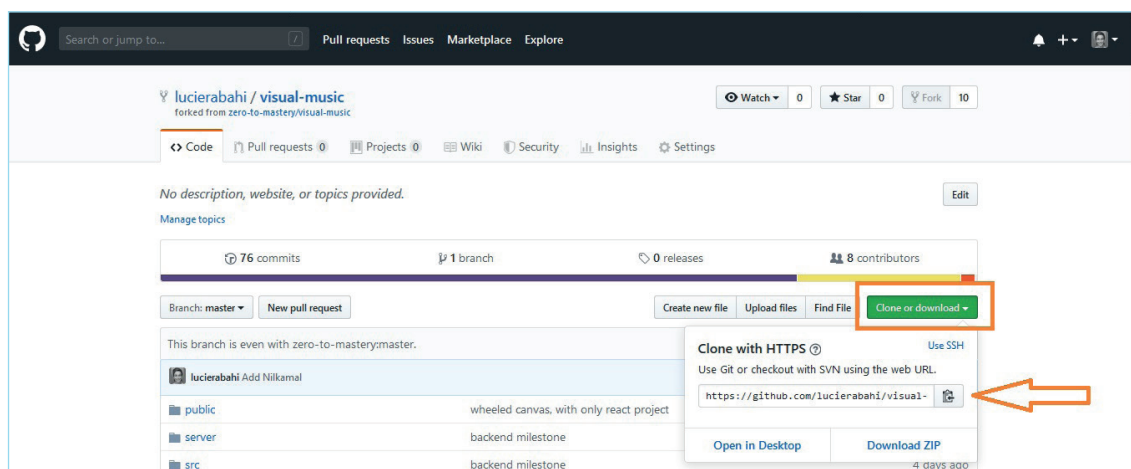
### Step 1: Set up Git

If you haven't yet, you should first set up Git. Don't forget to set up authentication to GitHub from Git as well.

### Step 2: Create a local clone of your fork

Right now, you have a fork of the visual-music repository, but you don't have the files in that repository on your computer. Let's create a clone of your fork locally on your computer.

- On GitHub, navigate to your fork of the visual-music repository.
- Under the repository name, click *Clone or download*.
- In the *Clone with HTTPs* section, click the *icon* to copy the clone URL for the repository.



- Open your Terminal and navigate to the location where you want to have the cloned repository.
- Type `git clone`, and then paste the URL you copied earlier. It will look like this, with your GitHub username instead of YOUR-USERNAME:

`git clone https://github.com/YOUR-USERNAME/visual-music`

```
Lucie Rabahi@LAPTOP-OQ0CONPV MINGW64 ~
$ cd documents

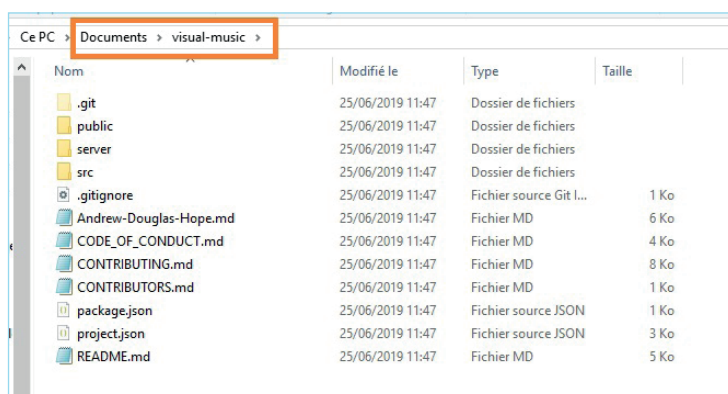
Lucie Rabahi@LAPTOP-OQ0CONPV MINGW64 ~/documents
$ git clone https://github.com/lucierabahi/visual-music.git
```

- Press Enter. Your local clone will be created:

```
Lucie Rabahi@LAPTOP-OQ0CONPV MINGW64 ~/documents
$ git clone https://github.com/lucierabahi/visual-music.git
Cloning into 'visual-music'...
remote: Enumerating objects: 153, done.
remote: Counting objects: 100% (153/153), done.
remote: Compressing objects: 100% (99/99), done.
remote: Total 419 (delta 68), reused 128 (delta 54), pack-reused 266R
Receiving objects: 100% (419/419), 8.59 MiB | 5.48 MiB/s, done.
Resolving deltas: 100% (171/171), done.

Lucie Rabahi@LAPTOP-OQ0CONPV MINGW64 ~/documents
$
```

Now, you have a local copy of your fork of the visual-music repository:

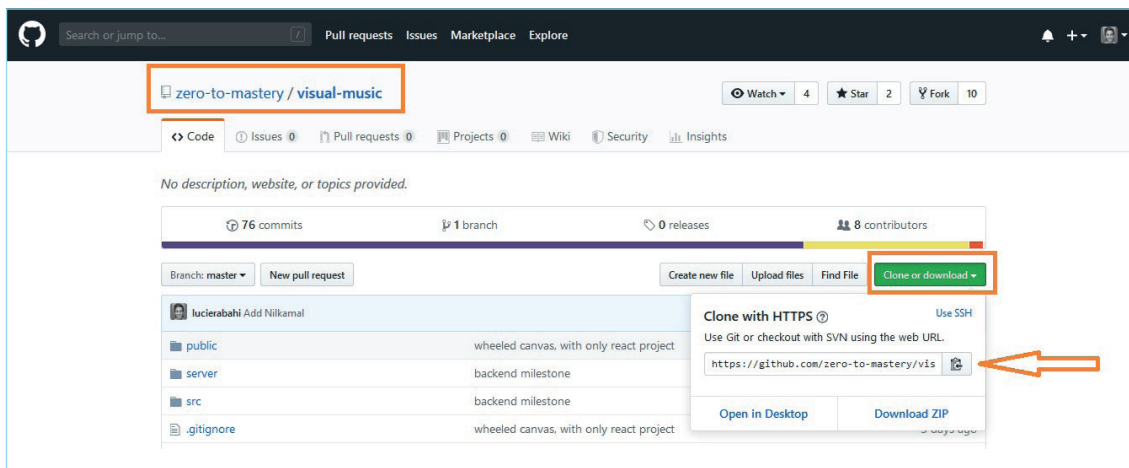




## Configure Git to sync your fork with the original repository

When you fork a project in order to propose changes to the original repository, you can configure Git to pull changes from the original, or upstream, repository into the local clone of your fork.

- On GitHub, navigate to the **zero-to-mastery/visual-music** repository.
- Under the repository name, click *Clone or download*.
- In the *Clone with HTTPs* section, click to copy the clone URL for the repository:



- Open your Terminal.
- Change directories to the location of the fork you cloned in previously:

```
Lucie Rabahi@LAPTOP-OQ0CONPV MINGW64 ~/documents
$ cd visual-music

Lucie Rabahi@LAPTOP-OQ0CONPV MINGW64 ~/documents/visual-music (master)
$
```

- Type `git remote -v` and press Enter. You'll see the current configured remote repository for your fork:

```
Lucie Rabahi@LAPTOP-OQ0CONPV MINGW64 ~/documents/visual-music (master)
$ git remote -v
origin https://github.com/lucierabahi/visual-music.git (fetch)
origin https://github.com/lucierabahi/visual-music.git (push)

Lucie Rabahi@LAPTOP-OQ0CONPV MINGW64 ~/documents/visual-music (master)
$
```

- Type `git remote add upstream`, and then paste the URL of the original GitHub repository you copied previously, and press Enter.
- To verify the new upstream repository you've specified for your fork, type `git remote -v` again. You should see the URL for your fork as origin, and the URL for the original repository as upstream:

```
Lucie Rabahi@LAPTOP-OQ0CONPV MINGW64 ~/documents/visual-music (master)
$ git remote add upstream https://github.com/zero-to-mastery/visual-music.git

Lucie Rabahi@LAPTOP-OQ0CONPV MINGW64 ~/documents/visual-music (master)
$ git remote -v
origin https://github.com/lucierabahi/visual-music.git (fetch)
origin https://github.com/lucierabahi/visual-music.git (push)
upstream https://github.com/zero-to-mastery/visual-music.git (fetch)
upstream https://github.com/zero-to-mastery/visual-music.git (push)

Lucie Rabahi@LAPTOP-OQ0CONPV MINGW64 ~/documents/visual-music (master)
$
```

- Now, you can keep your fork synced with the upstream repository with a few Git commands.



## Keep your fork repository up-to-date with the upstream repository:

If you have followed the setup instructions, the upstream is setup. To get updates from the upstream:

- Navigate to your project folder (the local copy of your fork) and run the command:

*git fetch upstream*

It fetches the branches and their respective commits from the upstream repository. Commits to *master* will be stored in a local branch, *upstream/master*.

```
Lucie Rabahi@LAPTOP-OQ8CONPV MINGW64 ~/documents/visual-music (master)
$ git fetch upstream
From https://github.com/zero-to-mastery/visual-music
* [new branch]      master      -> upstream/master

Lucie Rabahi@LAPTOP-OQ8CONPV MINGW64 ~/documents/visual-music (master)
$
```

## Bring your fork's *master* up-do-date:

- Check out your fork's local *master* branch.

*git checkout master*

- Merge the changes from *upstream/master* into your local *master* branch:

*git merge upstream/master*

This brings your fork's *master* branch into sync with the upstream repository, without losing your local changes.

### **Bring a specific branch up-to-date:**

- Checkout to a local branch that you want to be updated.

For example: *dev*

```
git checkout dev
```

- If you do not have this branch yet, you can create it by doing:

```
git checkout -b dev
```

- Now merge with the branch you want to be updated from:

```
git merge upstream/dev
```

This brings your fork's *dev* branch into sync with the same one on the upstream repository.

### **Tip:**

For a list of all the branches you can do:

```
git branch -r
```

---

### **Update your forked GitHub repo:**

Now your local repository is updated. This does not update your forked GitHub repository. To update it, you have to push your changes:

```
git push origin dev
```

or, if encountering a problem with the previous upstream setup:

```
git push --set-upstream origin dev
```

---





## Start coding and commit:

### Commit

When you are doing coding, keep in mind that other's are also modifying the same code base and that conflicts may arise. In order to make it easier for you and the others to resolve such conflicts on a per-commit basis try to commit as often as possible (when it makes sense!) and commit only small changes. Provide a meaningful message for every commit so that anyone reviewing the commit later knows from reading the message what the commit changes.

- Making a commit can be done on the command line interface. Type:

```
git add FILE1 [FILE2] ...      or
```

```
git add .                      (adds all the files)
```

- Then check if everything has been added or if you missed something:

```
git status
```

- Add a message for the commit:

```
git commit -m "YOUR TEXT HERE "
```

Always keep in mind that errors happen all the time. If it is later necessary to revert certain changes because they broke something, reverting a small commit is easy and not much of a problem.

If the change to be reverted is mixed up with lots of other changes the reverting of such a commit will be a painful nightmare...



## Create a pull request from a fork:

If you've forked a repository and made changes to the fork, you can ask that the upstream repository accept your changes by creating a *pull request*.

You can open a *pull request* to the *upstream* repository from any *branch* or *commit* in your fork.

However, we recommend that you **always start a new branch** when working on a task, **so that you can push followup commits** if you receive feedback on your pull request.

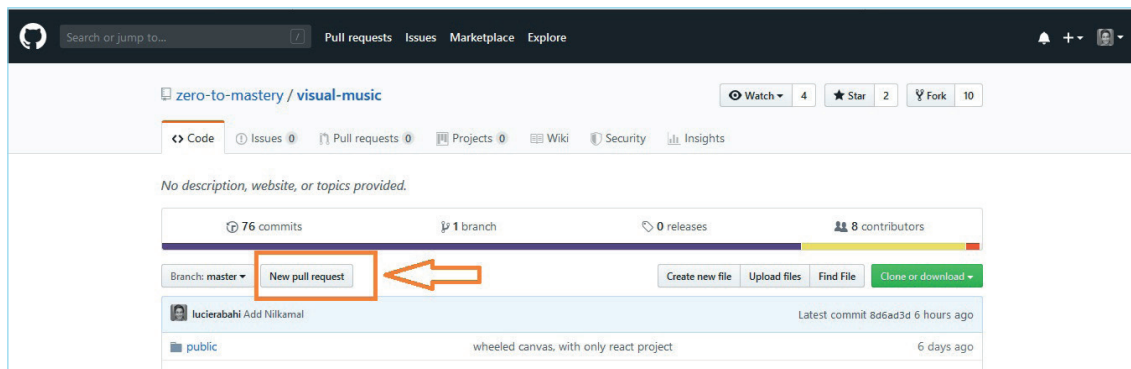
### Step 1:

- Navigate to the original repository you created your fork from, i.e.:

<https://github.com/zero-to-mastery/visual-music>

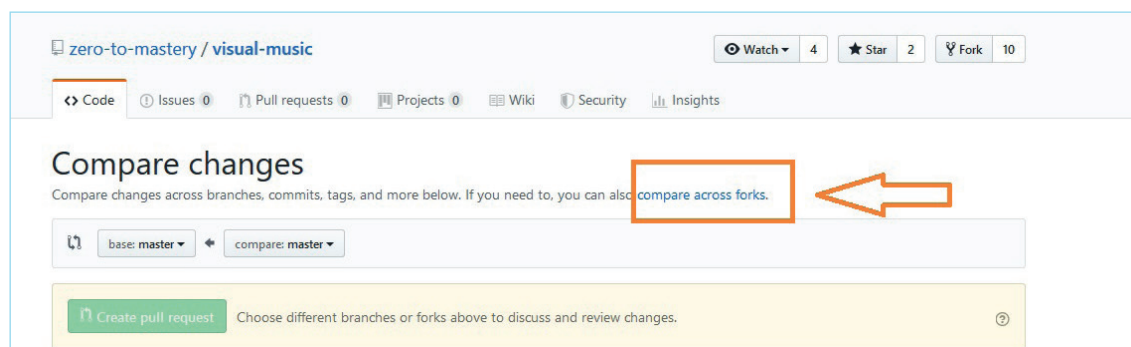
### Step 2:

- To the right of the *Branch* menu, click *New pull request*.



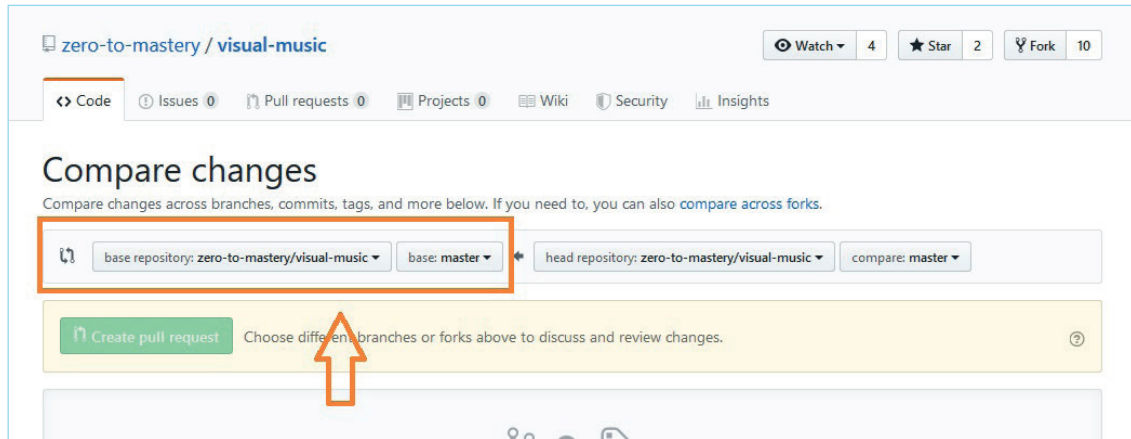
### Step 3:

- On the Compare page, click *compare across forks*.



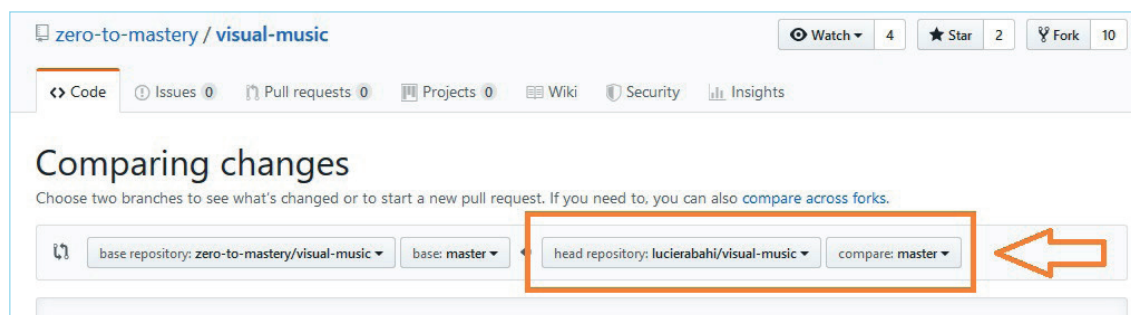
#### Step 4:

- Confirm that the *base fork* is the repository you'd like to merge changes into. Use the *base branch* drop-down menu to select the branch of the upstream repository you'd like to merge changes into:



#### Step 5:

- Use the *head fork* drop-down menu to select your fork, then use the *compare branch* drop-down menu to select the branch you made your changes in.



#### Step 6:

- Type a title and description for your pull request.

Please make it something useful, explain what you've worked on and/or if there are any issues you need help with.

**Useless message example:** pull request prguide

**Useful message example:**


*Title:* Created a PR Guide for making contributing process easier.

*Description:* 1. Explanation on how to install the project on local machine.  
2. Explanation on workflow and PR Guide

## Open a pull request

The change you just made was written to a new branch named `lucierabahi-patch-1-1`. Create a pull request below to propose these changes.

base: `master` ← compare: `lucierabahi-patch-1-1` ✓ **Able to merge.** These branches can be automatically merged.



Write Preview

This is where you write a description of the changes made. Please make it something useful, explain what you've worked on or/and if there are any issues you need help with.

**Useless message example:** pull request prguide

**Useful message example:**  
**Title:** Created a pr guide for everybody to collaborate.  
**Description:** 1. Updated to include explanation of forking and cloning.  
 2. Changed the flow chart to reflect the new branches

Create pull request

Reviews

No reviews

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestones

No milestones

Helpful links

Contributors

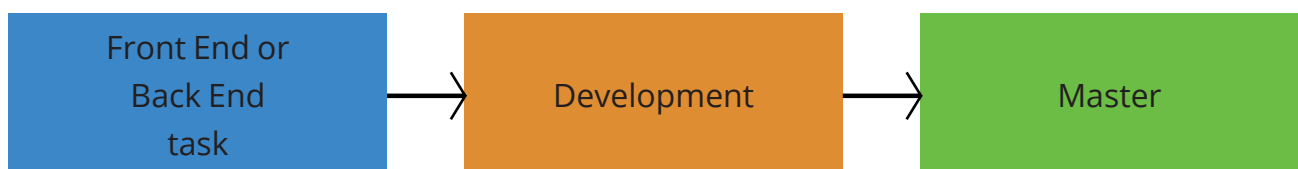
### Step 7:

- Click on *Create pull request*.

A project admin will review it, merge it or discuss it with you :)



### Sequence



Push your completed local tasks to the development branch ( or specific branch in case of a big feature )

Here we can fix possible bugs from merging before pushing a stable version to master.

This is the stable production branch.



## Rules for PRs:

- Nobody is allowed to merge their own pull requests, unless it is a very minor change. Why are we doing so?
  - \* It's deemed good practice to not merge your own PR,
  - \* It allows other people a chance to read/understand your code and learn from you,
  - \* It ensures the change is bug free or is the best approach,
  - \* It's good practice for when you become employed.
- We have a *dev branch* to work on and all PR will be added to that *dev branch*. There we can fix possible bugs from merging **before pushing a stable version to master**.
- No large PR without description will be merged: 1 PR = 1 Trello task.
- PRs won't be accepted if the changes made are not clearly explained (this includes, when relevant, details of changes to the interface, new environment variables, exposed ports, useful file locations...).
- PRs are not merged blindly without being reviewed and the reviewer confirms in a comment that everything's ok, and don't hesitate to make suggestions if necessary: this will improve the quality of the app and everybody's skills. If a doubt on the PR, there should be a discussion in the GitHub Pull request.
- Bugs and issues have to be referenced: screenshot if possible, and explanation on how the problem occurred (OS, browser, when did it occur during the thread...).
- A PR cannot be merged without correcting failures to build or code conflicts.