Table of Contents

General Rules

GitHub

- Prefer Git GUI's rather than git bash (GitHub Desktop, Source Tree, Fork, IDE VCS Controls).
- Local branches should not be deleted.
- Issues, commits and branches should be single purpose, replayable. This allows them to be easily reverted and cherry-picked.
- Do **NOT** forget to name your stashes (does not apply to GitHub desktop users).
- You may **NOT** rebase or force push changes to the remote without being in accordance with the whole development team.

Git Workflow

1. Create a new issue for the feature/hotfix. Follow the Issue Flow.
2. Create a new LOCAL branch from the develop branch, use the naming convention 'feature-' and then a description of the feature in development. Prefer small names.
3. Perform any code changes on your local branch and:
    a. if you are using the git cli you must add the files first before committing with the `git add .` command.
    b. Otherwise, select which file/line changes to pass to your next commit from your GUI of choice.
4. Commit your changes, use the Commit Guidelines.
5. Fetch and pull the main and develop branch locally.
6. Merge your local develop branch into your working branch. **TIP:** This is best performed using a GUI git client.
    a. If you are working on a hotfix branch, merge main into your working branch also
7. Resolve any conflicts if any, contact other team members if you are unsure when resolving. Commit the changes.
8. Push your working (feature) branch to the remote repo.
9. Make a new pull request from your working branch into develop. Follow the Pull Request Flow
    a. If you are working on a hotfix, create a pull request into main also.

10. Repeat

Git Commit Message Guidelines

- Subject from body must have an empty line separating them.
- The subject should be up to 50 characters, starting with a capital letter and without a period at the end. Make sure it uses the imperative mood, for example: ‘Add’ not ‘Added’, ‘Modify’ not ‘Changed’.
- Each line in the body should be wrapped at 72 characters. Mention what you did **briefly**. Ideal format below:

```
Add heroku auto-deploy automation
- Requires heroku email.
- Requires secret to be maintained
- Requires heroku app name
- Add setup-java to make it a runnable property
- Add procfile inline
```

GitHub Issues Flow

When creating a new issue:

1. Add yourself as assignee
2. Add any needed labels that are appropriate
3. Add the current project
4. Complete the provided issue template

GitHub Pull Request Flow

Creator

1. Create an appropriate pull request title based on the commits you have made. The title should be less than 44 characters. First letter is capitalized.
2. Assign one reviewer.
3. Assign yourself as an assignee.
4. Add any applicable labels.
5. Add to the “development” section the issue/s you are resolving
6. Complete the pull request template
7. Wait for reviewer feedback.
    a. If changes are accepted, continue working.
    b. If changes are rejected, try to resolve the identified issues by following the GitHub Workflow**, but make sure you add the bug label in the issue**. Go to step 7 again.

Reviewer

1. Start a review of the pull request you are responsible for.
2. If there are any conflicts, refuse the pull request. Force the other to create a new PR after resolving the conflicts locally (with a nice comment :)).
3. Review all files and leave any comments where needed for clarity in code (do **NOT** start a new review).
4. When submitting your review, either approve or request for changes (**do not submit as COMMENT**).
   a. Wait for corrections, if any, discuss with team members for clarity in person.
5. When merging the pull request, the new commit title should be ‘Merge ’ + {PR title}. The description must contain ‘fixes #{number}’ at the end, where the number is the issue being resolved.
6. Delete the branch which was just merged (from remote).

GitHub Releases Flow

1. When creating a release from GitHub, first add the tag following the below guidelines:

   Given a version number MAJOR.MINOR.PATCH (such as 1.0.0 or 2.1.1), increment the:

   1. MAJOR version for compatibility breaking changes.
   2. MINOR version when you add functionality without breaking changes.
   3. PATCH version when you make bug fixes without breaking changes.
   4. Pre-release versions must be denoted with a hyphen and the type of pre-release along with a number. For example, 1.0.0-alpha.1, 1.0.0-beta.1.

   When making a new release, reset all numbers except the new Major version number.

2. Release title should have this format: “Release 1.0.0 - Customer API”. In other words, release then the tag number dash a small description of the biggest change or the latest state of the project as a last resort (Iteration 1).
3. Add auto-generated release notes.
4. Publish release.

GitHub Labels

found bug → For newly discovered bugs
bug fix → for bug fix pull requests
enhancement → feature addition

documentation → change in documentation files

React.JS

React Rules/Principles

- Data management (with statefulness) is achieved with Redux library: Explanation
- All component data that is/must be stateful (for UI presentation) must be stored in the “state” variable.
- All components should be in a function-like form. Class components were not created to work with hooks and are considered legacy from the React developers.
- The “props” variable data should **NOT** be changed. It is discouraged by React.
- Each page is a component. These components should be used as organizers of other nested components.
- Create as many components as possible when practical, avoid making components that handle multiple unrelated responsibilities and data sets (SOLID).
- Do not create components for only displaying a particular set of data/information content with no functionality (especially when only used once). Emphasis on reusability and abstraction when possible.
- **NEVER** allow full browser page refreshes unless desperate.
- Do **NOT** use jQuery or any equivalent unless desperate. jQuery based user libraries that provide created functionalities may be acceptable.
- Mobile responsiveness is optimized for a minimum width at 350px.

React Tips

Hooks

Use react “hooks” (which are functions) as much as possible: Short Video

Component

Tips for identifying components:

- If you are repeating code functionality (especially JavaScript) and/or HTML (JSX), it is likely a component.
- If you are presenting a list in the UI, the list itself and the items are independent components. Both components are combined with data sharing.
- If HTML (JSX) must be hidden or revealed based on a condition, it may be a component.

Express.JS

- Do ***NOT*** use semicolons when programming with JavaScript (or any other JavaScript based framework like Express.JS or React)
- When programming with JavaScript, always use “double quotes” when creating strings and ‘single quotes’ for characters. In any unmentioned case, prefer “double quotes”.

Plugin Management

Updates

For updating plugins use the npm-check-update package (only when necessary):

1. Run `ncu` to see which dependencies have updates available.
    a. If there are updates in the frontend, do npm install
    b. If there are updates in the backend, manually write the new version in package.json for each dependency.
        i. Then close the terminal and re-open it.
        ii. Then `npm install` and make sure that mongoose version has not changed in your GitHub current changes.
2. Test if the dependency changes are stable, before pushing your branch to the remote.