Development Process

This document outlines the development process to be followed by the saints serving on the FTTA Attendance Project. We ask that all parties involved take grace and exercise patience and diligence to observe this process as closely as possible. In all our service, even our coding, we need to be exact, thorough, fine, patient, and proper. This is to ensure the continuation of the project in a good way; we want to maintain the project’s longevity. Our looseness in our coding damages the entire project and in turn, the training. As Paul wrote, “A little leaven leavens the whole lump” (Gal. 5:9).

# Feature Request/Ideation

The development process begins with an idea. This can come from anyone; oftentimes it will be a TA who requests that we implement or modify a feature, but any trainee, including the developers, are welcome to contribute ideas and suggest features or improvements. However, it is ultimately up to the TAs’ and overseers’ discretion as to what will be implemented.

The overseers should meet regularly with the TAs to determine priorities for the project and development effort and assign features to developers accordingly. The overseers should endeavor to lay out a roadmap for a term, possibly two terms, scheduling how long it will take to accomplish the necessary tasks.

Feature requests can be seen and submitted on the issue tracker at: [git.ftta.lan/](http://git.ftta.lan)

## Hotfixes

Development for hotfixes (bug fixes that go straight to production) is different from regular feature development. Depending on the urgency of the bug, they can either be reported on the issue tracker at: [git.ftta.lan/](http://git.ftta.lan/) or by talking directly to one of the developers.

For procedures on patching hotfixes, go directly to the *Development*-*>Hotfixes.*

# User Study

For some features, especially user-facing ones, it will be necessary to conduct a user study before writing a spec and building out the feature. The overseer will let you know whether or not a user study should be conducted and who should be responsible. For more information on user studies, see *How to Conduct a User Study*.

# Specification & Design

When a feature is assigned for implementation, the first thing to do will be to write and agree on a specification for that feature. Typically the overseer will write a rough spec for the developer(s) to build on and flesh out. The spec does not need to be extremely thorough or overly technical, but should be detailed enough that future developers can understand *what the module is supposed to do*.

After the spec is approved, a design document should be written. The responsibility of writing the design doc falls on the developer(s). As with the spec, the design doc needn’t be overly detailed, just enough so that future developers can understand *how the module works*, particularly how it is structured and how it’s different components interact.

Both the spec and design should be finished before any code is written. The design should be sent to the overseer for review, but it is not a hard and fast rule to wait for approval on the design before development begins (unless otherwise specified).

See *How to write spec and design docs* for more help on writing a specification and design documents.

# Development

## Coding Standards

In general, committed code should be:

* **Well formed:** properly indented, vertically spaced, following all the standards in the style guide.
* **Clear:** free from ambiguity, easy for others to read and follow
* **Logical:** well structured, organized clearly
* **Concise:** free of redundancies, done in simplest way possible
* **Well documented:** including functions, variables, and major code blocks

See *Developer Guidelines* for more details on Attendance Project standard development practices.

## Workflow

The Attendance Project workflow mostly follows [git-flow](http://nvie.com/posts/a-successful-git-branching-model/), a popular workflow using the Git VCS. In brief:

There are two main branches: *dev* and *master:*

* *Dev:* This branch can be considered a “beta” of the app and contains the latest deployable code that is being tested and not yet ready for production. This branch is deployed to the development server.
* *Master*: This is the production branch that is deployed to all the trainees and training staff. Code on this branch is either live or should be production-ready.

There are three kinds of branches you will create: *feature*, *release*, and *hotfix:*

* *Feature:* a feature branch is created whenever a new feature is started and exists for as long as the feature is in development. Some larger feature branches may require further branching, which is encouraged in git. This type of branch should be initially branched off of *dev* and then merged back to *dev* when it is complete and ready for testing.
* *Release:* a release branch is used for last minute fixes and cleanup before a major release is deployed. You should create a release branch off of *dev* when the state of *dev* is very close to production-ready. At that point, all last minute fixes and tuning should be done in the *release* branch and then merged into *master* and deployed. Afterwards, be sure to merge the *release* branch back into *dev*. All release branches should be named with the convention *release-\** (e.g. release-3.1 or release-1.3)
* *Hotfix:* a hotfix branch is made when there is a bug discovered *in production* and needs to patched immediately. This type of branch should be branched off *master* and merged back into *master* as soon as it is fixed. All hotfix branches should be named with the convention *hotfix-\** (where *\** is the current version number plus the hotfix number e.g. hotfix-3.1.1 or hotfix-3.1.77)

## General Development

Developers should follow the above workflow, setting up a *feature* branch as they begin development of a module. As development moves forward, developers should have the Attendance+ Project stack set up locally to test as they develop.

The developers should refer back to the spec and design documents to ensure they on the right track. Additionally, if changes are made to the design, these should be reflected in the design doc.

In the course of development, the overseer should check in with the developers to see how things are progressing, to help produce estimates and make adjustments in teams and workload if necessary. In some cases, it may also be necessary to produce a work plan and timeline to help development on track.

## Testing and Code Review

After working and testing locally, the feature should reach a point where the developer would consider it done and ready for testing.

Unit-testing will be done inside the feature branch. The overseer will assign a developer to do the testing (for more information, read *How to write tests*). The test engineer will be responsible for developing a battery of unit-tests by consulting with the spec, the design, the module developer(s), and the overseer. Then the developer(s) will be responsible for ensuring that the module passes all unit tests.

After the module passes all its unit tests, it needs to pass code review. Code reviews are conducted to check for code readability and conformance to Attendance Project coding standards. An overseer or someone the overseer assigns should perform the code review, briefly reading through the new code (for more information, read *How to do a code review*).

Once a module has passed both unit testing and code review, its entire *feature* branch should be merged into the *dev* branch. This is where integration testing takes place (again, see *How to write tests*). Before a major release, the entire *dev* branch must pass a battery of integration tests. Thus, we are assured that any code that gets pushed into production has passed through unit-testing, code review, and also integration testing.

## Production

Pushing a new feature into production is a privilege earned by following the above procedure. Developers ought to be aware that the code they push into production affects the entire training, which includes 300+ trainees and the various servings ones in the training office. A developer should have the assurance that the code being deployed is working, tested, well formed, well documented, and usable by future developers.

When merging *dev* into *master* remember to also tag youre release using *git tag* with the version number and release name. Once the code is pushed and live, the developer should do a quick sanity check to make sure everything works as expected and that nothing broke. The responsible developer should then be ready to address any bugs discovered in production.

## Hotfixes

When a bug is discovered in production (as opposed to on the dev server), a *hotfix* should be applied, that is, a fix that is patched immediately and live. Hotfixes should be branched off of the *Master* branch, fixed, tested locally, and merged back into master as quickly as possible.

If you do not have deployment privileges, however, instead of merging directly, submit a merge request with a detailed description of the problem and fix. Tag an overseer (and notify them in person as well). The overseer will review your change before deploying it to production for you.

For bugs that are too large for a straightforward hotfix (hopefully this never happens), the master branch should be rolled back to the most recent stable version instead of trying to apply a patch.