ZeldaDev Documentation

Carla Wilby, Mike Field

Jul 01, 2019

Contents

Here you can find an intro to our ways of working, our coding practices and repo documentation.

For documentation specific to:

Student Web App Docs. These are the docs specific to the internals of the Angular Student Web App.

Client Side Docs. These are the docs specific to the Angular Client Side Docs.

Machine Learning Docs These are the docs specific to the ML Component of the Zelda infrastructure.

Field Vision Docs <https://fieldvision.readthedocs.io/en/latest/>_. These are the docs specific to the MLKit implementation.

Internal Upload Content Page Docs.

CHAPTER 1

Contents

1.1 Welcome to Zelda!

We have four main areas of development:

1. The Android app
2. The student facing webapp
3. The client facing webportal
4. The machine learning backend

1.1.1 All the links you'll ever need:

Check out the following and make sure you have the permissions for the project components you work on.

Admin

Zelda Internal Home Page

App Dev Docs Drive

Slack

Repositories

Android app repository

Student Webapp repository

Client Webportal repository

Consoles

Firebase Dev console

Firebase Prod console

1.2 Getting Started

If you want to get started as quickly as possible, here is a rundown of how to set up our dependencies and get going.

1.2.1 Cloning a repository

1. Setup git

   ```
   sudo apt-get install git
   ```

2. Get added to the project as an Admin/Developer and find the repo you want to clone in the Zelda Project directory here: https://bitbucket.org/account/user/zelda_learning/projects/ZEL

3. Example cloning zeldav1 repo

   ```
   git clone https://USERNAME@bitbucket.org/zelda_learning/zeldav1.
   git cd <REPO> git checkout master
   ```

4. A problem you may encounter is when pulling a new repository, you only get and can update the current branch. In order to get all branches use the following:

```
``git branch -r | grep -v '\->' | while read remote; do git branch --track "$
↪{remote#origin/}" "$remote"; done``
``git fetch --all``
``git pull --all``
```

1.2.2 Adding code to a repo

Now that the repository is ready, you can now start adding code to it.

The steps are as follows:

1. Create an issue on BitBucket outlining the changes you want to make in a branch.

   ```
   git flow feature start <ISSUE NUMBER>-<BRIEF DESCRIPTION,
   SEPARATED BY DASH "-" > git branch git flow feature publish <ISSUE
   NUMBER>-<BRIEF DESCRIPTION, SEPARATED BY DASH "-" >
   ```

If you have problems with gitflow you can create an issue through the BitBucket UI.

2. Start a new branch with a name *<ISSUE NUMBER>-<SAME BRIEF DESCRIPTION AS ABOVE >*

   ```
   git checkout -b <ISSUE NUMBER>-<SAME BRIEF DESCRIPTION AS ABOVE >
   ```

3. Write code MAKE SURE YOU'RE DOING THIS ON THE RIGHT BRANCH!

This is where the actual magic happens.

4. Commit it

   ```
   git add . git commit -a -m "hey look, real work!"
   ```

Please commit between changes, even if small!

5. Push it back up to bitbucket

   ```
   git push
   ```

If this is the first time you're pushing you may need:

```
``git push --set-upstream origin <BRANCH-NAME>``
```

6. Open a pull request (PR)

   You can do this using the bitbucket UI. Make sure you specify and explain all the changes you have made. Add a reviewer you would like to look at your code. Doing this will automatically close the issue when the PR is merged.

7. Get it tested (we're still setting this up!), reviewed and +1'ed
8. Merge!

   Once you've got a +1 you can click Merge on the UI. Back in your command line:

   ```
   git checkout master git pull
   ```

If you have other unfinished branches make sure you catch them up with the current changes.

```
git checkout master git pull git checkout <UNFINISHED-BRANCH-NAME> git
merge master
```

And resolve any conflicts.

1.3 Ramping Up

Here are some tools, tips, links and courses to get you going as quickly as possible.

1.3.1 New to Unix?

https://www.csoft.net/docs/course.html

1.3.2 New to Git?

https://www.udacity.com/course/version-control-with-git–ud123

1.3.3 Angular2

How to learn Angular 2: http://www.angular2.com/ Design doc: https://docs.google.com/document/d/1JIlz8LTUBqNm_GCTihHLsmBmnC2ZBZCLFHhiJ2ZwhzU/edit Firebase JS SDK: https://github.com/firebase/firebase-js-sdk Firebase Webapp Init: https://firebase.google.com/docs/web/setup Spring MVC and Angular stack: https://dzone.com/articles/web-app-architecture-spring-0 Help: https://blog.angular-university.io/top-10-angular-2-tutorials-blogs-and-podcasts/

1.3.4 Android

Best Principles: https://docs.google.com/document/d/1sH9hy8qQOHgKDh-mno_WWZBwYz0wDFe4ss6gXU149i0/edit?usp=sharing Naming conventions: https://medium.com/@mikelimantara/overview-of-android-project-structure-and-naming-conventions-b08f6d0b7291

1.3.5 Machine Learning

TensorFlow: https://www.tensorflow.org/ Keras: https://keras.io/ Using Keras: https://elitedatascience.com/keras-tutorial-deep-learning-in-python Using Keras: https://machinelearningmastery.com/introduction-python-deep-learning-library-keras/ Firebase Python SDK: https://firebase.google.com/docs/reference/admin/python/ Firebase Python pip: https://pypi.python.org/pypi/python-firebase/1.2 Firebase Python API library: https://developers.google.com/api-client-library/python/apis/firebaserules/v1 Characteristics map: https://drive.google.com/file/d/1j6PsCi4gCRq-pFd6hm_n5hpzBJADle8r/view?usp=sharing Intro to Machine Learning Stanford course: https://www.coursera.org/learn/machine-learning?authMode=login Practical Machine Learning Coursera: https://www.coursera.org/learn/practical-machine-learning

1.4 Our Dev Process

1.4.1 Purpose

The purpose of our Dev Process is to ensure that we follow these essential team dynamics:

Communication

As the team grows and our components become more integrated, it is essential that we follow a structured process that allows us to keep up to date with team progress. Communication should be seamless and informed, with the goal of easing decision making that applies to multiple parties and facilitating easy collaboration where necessary.

Accountability

As a lot of the team works from home, it is essential to ensure that the rate of development is consistent. There should be adequate and reasonable expectation around work outputs on a weekly basis. This will allow us to build realistic timelines for projects.

Time Management

It is easy to get distracted or to lose momentum, especially if you are in charge of your own work output. For this reason it is essential that we establish and maintain a metric for work output expectations based on realistic use of time.

Code Quality and Best Practices

It is essential that we have a set of guidelines that we have agreed upon to ensure that code quality is maintained. This includes readable code, following language conventions and descriptive commit messages. Best practises is the manner in which we conduct the process of issue tracking, repository usage, Kanban usage and also includes testing and documentation.

1.4.2 The Code Process

We want to maintain consistency and naming conventions across branches, commit messages and pull requests. This allows us to easily find and understand where changes were made.

The steps are as follows: Create an issue on KanbanFlow outlining the changes you want to make in a branch. The issue title should follow this convention:

```
<ISSUE NUMBER>-<BRIEF DESCRIPTION, TEXT IN CAMEL CASE>
```

Then checkout a new branch from the correct parent feature branch with the same title as the issue on KanbanFlow

```
$ git checkout -b <SAME ISSUE NUMBER>-<SAME BRIEF DESCRIPTION>
```

Make one component/feature change, make a commit that adequately describes this change - please please commit between changes - even if they're small!

```
$ git add .
```

```
$ git commit -a -m "Something something something"
```

Push it back up to Bitbucket as often as you can. This will make sure the code is safe and also run the CI Pipeline tests to make sure it passes.

```
$ git push
```

If this is the first time you're pushing you may need:

```
git push --set-upstream origin <BRANCH-NAME>
```

Open a pull request (PR) through the Bitbucket UI. Make sure you specify and explain all the changes you have made, if you've made descriptive commits you won't have to add anything more as the commit messages are added by default.

Check the code content before you publish to make sure all the right code is included. Add a reviewer/s you would like to look at your code.

Once the request is accepted you can go ahead and merge, you can click this on the UI or through slack.

To make sure you're up to date locally, go back to your command line and pull from the feature branch you just merged to. Feature branches are described in more detail in the "Organising branches" section.

```
$ git checkout <FEATURE-BRANCH> $ git pull
```

Also make sure you keep master up to date if applicable

```
$ git checkout master $ git pull
```

If you have other unfinished branches on the same feature branch, make sure you catch them up with the current changes.

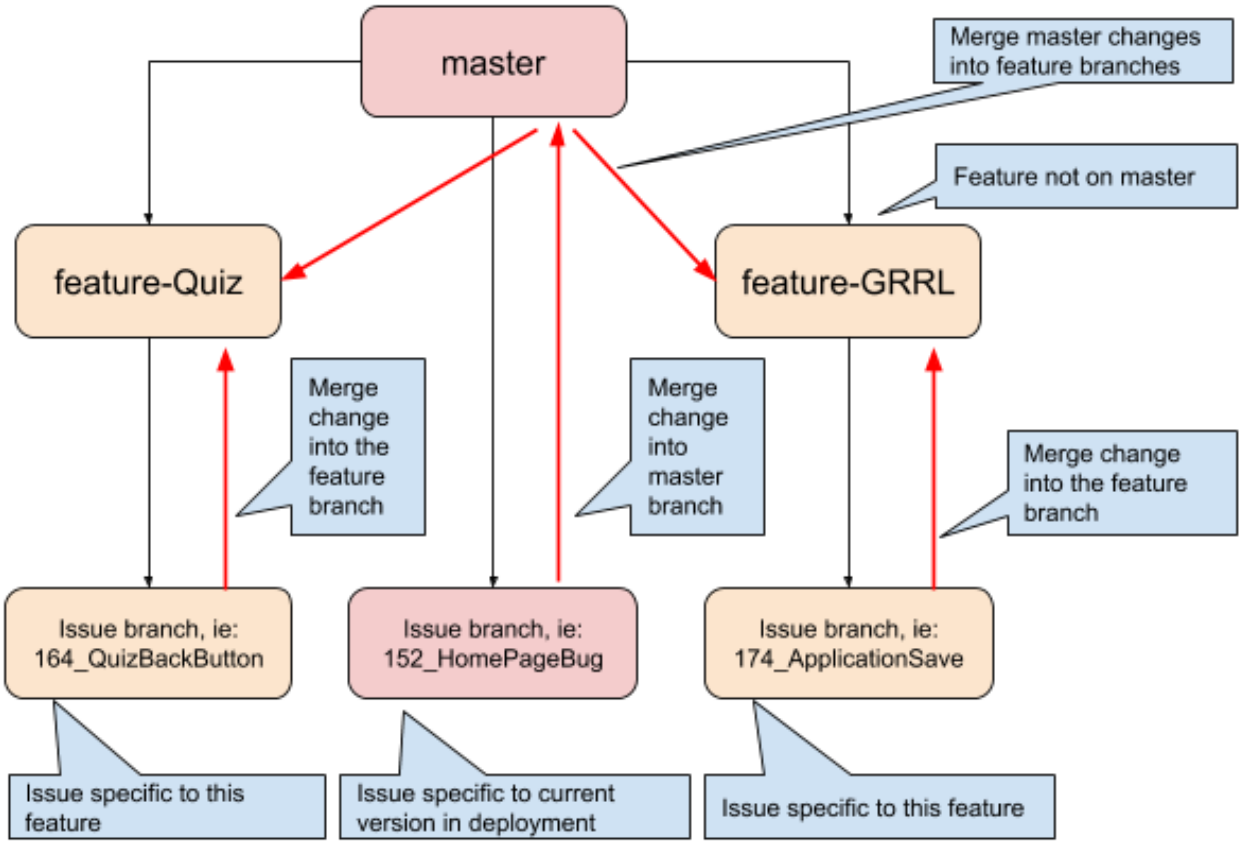
```
$ git checkout master git pull git checkout <UNFINISHED-BRANCH-NAME>
git merge <FEATURE-BRANCH>
```

And resolve any conflicts.

1.4.3 Organising Branches

Maintaining branch processes and order can be difficult, especially if you're working on your own. However, having feature branches is really important, especially if you want to keep different features that are in development away from each other and off master - which should just be for stuff that's being added to the current version in deployment.

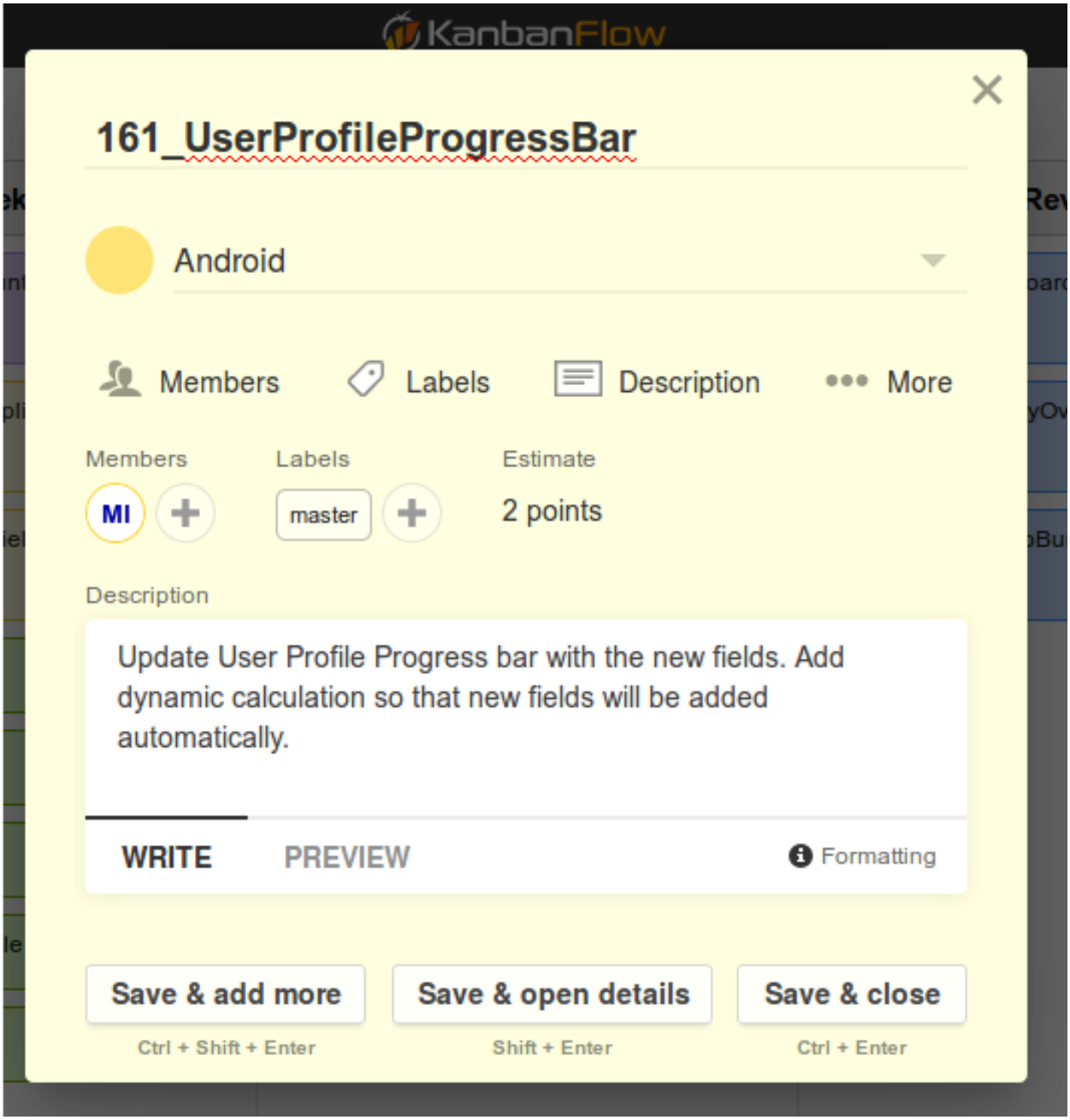
Here's a diagram illustrating how they should work:



1.4.4 Assigning Issues As Tickets

1. Add Task Tickets to To-do column

This column is the backlog, these tickets are necessary but not necessarily immediate. Once they are allocated for work they are moved into the "This Week" column.



The following key information that must be added to the ticket (as illustrated above) include:

- **Title**: Must correspond with the branch name and number (if applicable)
- **Assign component**: ie Android
- **Assign member**: this is the person/s responsible for completing the task
- **Assign label**: this corresponds to the parent branch (ie master, feature-GRRL)
- **Add description**: the core changes or details of the issue to be resolved
- **Add points estimate**: here I have given it 2 points which corresponds to 4 hours

2. Add to Review column once the task is completed. Add the PR url in the description

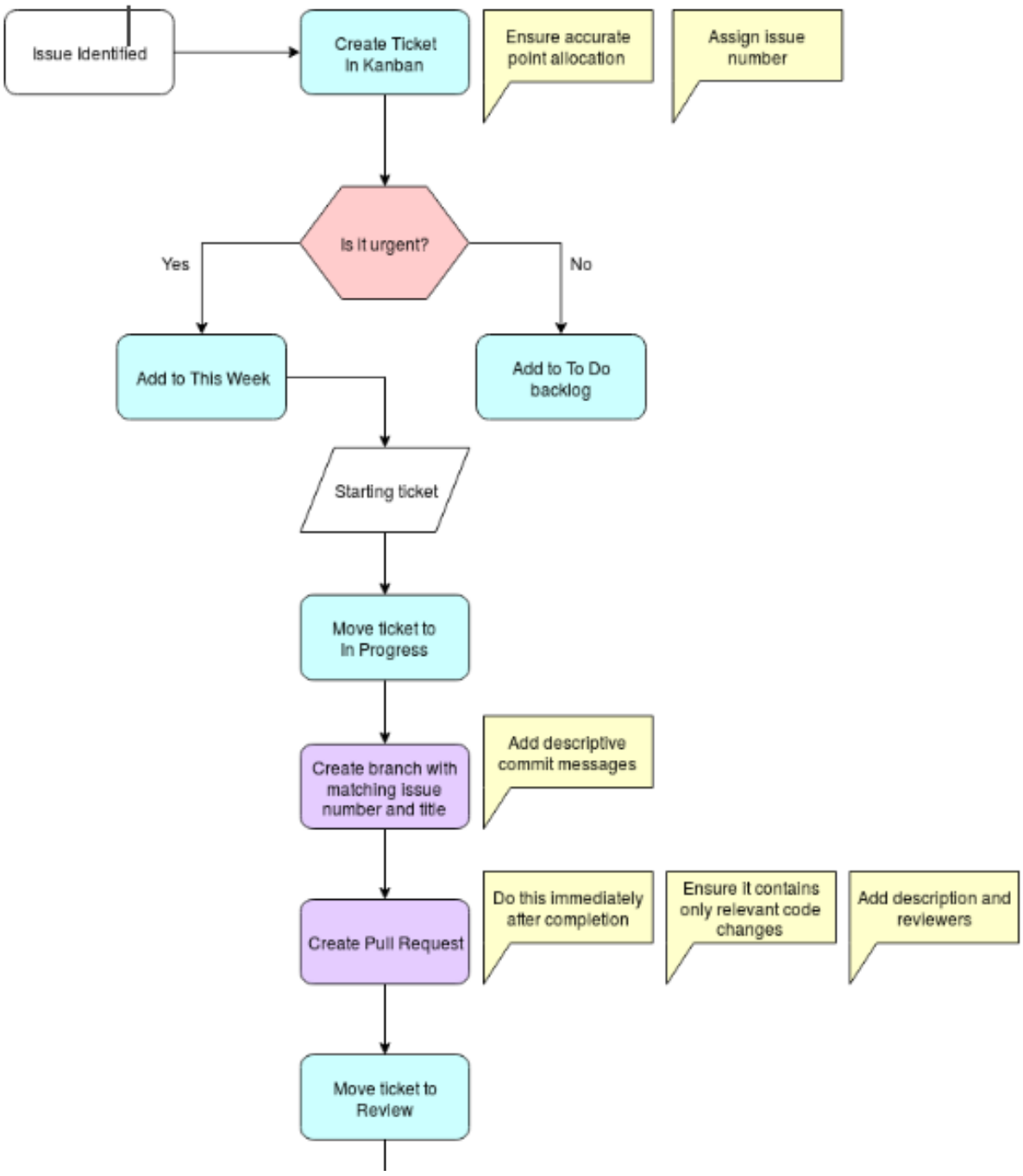
The Pull Request should have a reviewer, a description of what you changed and a title that matches the branch name and ticket name

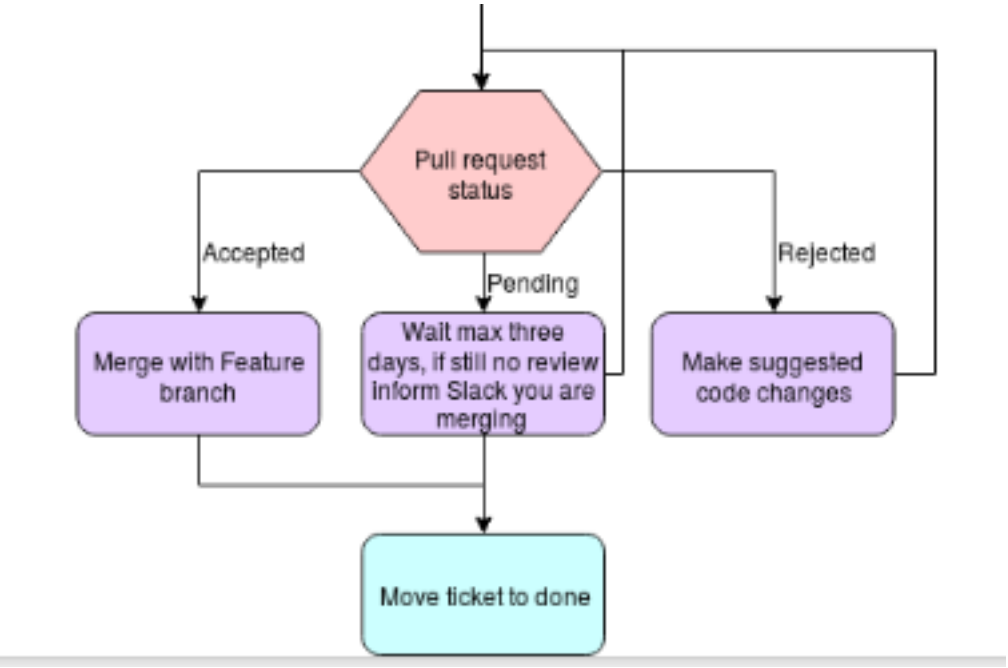
Don't forget to check the contents of the pull request to make sure it contains ONLY the changes relevant to this pull request and that you haven't made any mistakes.

Move to the Done column once the PR is accepted and merged.

1.4.5 Issue Tracking and Repository Process

Here's a diagram illustrating how they should work:


Issue Identified
Create Ticket In Kanban
Ensure accurate point allocation
Assign issue number
Is it urgent?
Yes
No
Add to This Week
Add to To Do backlog
Starting ticket
Move ticket to In Progress
Create branch with matching issue number and title
Add descriptive commit messages
Create Pull Request
Do this immediately after completion
Ensure it contains only relevant code changes
Add description and reviewers
Move ticket to Review



1.5 Setting up Documentation

1.5.1 Linking your Repo:

Cd into project directory, then:

```
mkdir docs sudo pip install sphinx
```

(call project name ZeldaDev) Add your name as author and just press enter for the other commands.

To create docs:

```
sphinx-quickstart
```

To render docs:

```
make html
```

To add ReadTheDocs theme:

```
pip install sphinx_rtd_theme
```

Edit your conf.py to add the sphinx_rtd_theme:

```
html_theme = "sphinx_rtd_theme"
```

Find your docs in the /docs folder, in the .rst files. Preview the docs in _build/html/index.html

Cheatsheet of documentation directives: http://dont-be-afraid-to-commit.readthedocs.io/en/latest/cheatsheet.html

1.6 Setting up Pipelines

Bitbucket Pipelines is an integrated CI/CD service, built into Bitbucket. It allows you to automatically build, test and even deploy your code, based on a configuration file in your repository.

To set up Pipelines you need to create and configure the *bitbucket-pipelines.yml* file in the root directory of your repository. This file is your build configuration, and using configuration-as-code means it is versioned and always in sync with the rest of your code.

1.6.1 Writing your YAML:

1. By selecting Pipelines in your repo toolbar and configure for your repo language, BitBucket will autogenerate and push a *bitbucket-pipelines.yml*
2. The bitbucket-pipelines.yml file holds all the build configurations for your repository. YAML is a file format that is easy to read, but writing it requires care. Indenting must use spaces, as tab characters are not allowed.
3. There is a lot you can configure in the bitbucket-pipelines.yml file, but at its most basic the required keywords are:

pipelines: contains all your pipeline definitions.

default: contains the steps that run on every push.

step: each step starts a new Docker container with a clone of your repository, then runs the contents of your script section.

script: a list of commands that are executed in sequence.

1.7 Deploying an app

Firebase supports build and deployment to an autogenerated domain. I'm looking into setting up a static redirect from this domain to eg student.zelda.org.za, but it seems to be harder than I thought it was.

1.7.1 Firebase Deployment:

1. **Build your prod version to `./dist` folder** `ng build --prod`

   Since this is a prod build please make sure it points to the prod Firestore and storage bucket.
2. **If you don't have it already, install firebase-tools** `npm install -g firebase-tools`
3. **Login to Firebase from CLI** `firebase login`
4. **From CLI launch firebase** `firebase init`

   This will give a number of configuration options:

   - When it asks if you want to overwrite files, enter No
   - When it asks to provide a public folder type 'dist'
   - When it asks if you want to configure as single page app enter Yes
5. **Deploy!** `firebase use --add`

   ```
   firebase deploy
   ```

1.8 Android

1.9 Machine Learning