projecto Documentation

Release 0.0.1

Projecto Team

September 08, 2014

Contents

This is primarily a developer documentation. Most materials here will not apply to your average users.

CHAPTER 1

Part I: Projecto Overview

1.1 Features

This article loosely covers all the features that we have/plan to have in projecto.

1.1.1 Overview

Project management systems nowadays are too complicated for small-medium sized projects. Good ones costs money and offers the product as a service rather than a software package that you can host on your own server.

Projecto offers four main features:

- Feed: for project updates
- Todos: managing tasks
- Files: file sharing
- Schedule: time scheduling with people

1.1.2 Feed

Pretty much like twitter. Short messages are made here. Tasks done in other parts of the systems can be posted here and so forth. People can also comment on each feed items.

Features that need work:

- Integration with other systems: such as tasks, files, and so forth
- RSS feed

1.1.3 Todos

Sort of like github issues. Has tags, assign, due date, and so forth. This is a place where people can use Markdown for description, as well.

Features that need work:

- Assignment of tasks, duedates
- Milestone
- Single todo view/commenting

1.1.4 Files

A simulated file systems for file sharing. Straight forward and simple.

This system is currently minimally viable. More features will be added soonish.

Wishlist:

- WebDAV
- Integration with things like Dropbox

1.1.5 Schedule

A meeting calendar for the team.

This system needs to be designed.

Wishlist:

- CalDAV integration
- Commenting/voting on meeting times. - Integration with doodle?

1.1.6 Management

Management of projects. Adding/deleting contributors, admins, and so forth.

Features that need work:

- Another class of users with readonly permission
- An owner that cannot be deleted by other admins
- Changing project name, and maybe have other attributes.
- Automatically add all users with a certain domain name for email address.

1.1.7 Users

Login system uses Mozilla Persona for easier integration. This means people can feel free to integrate LDAP by using a Mozilla Persona LDAP bridge for authentication.

Gravatar for avatar, and other user profile features are currently minimal, but could be worked on.

1.1.8 Planned features

- Email integration: Sending emails to update the team. Let team use email to respond to the system. - There are security concerns here but there are ways to mitigate this.
- Standards, standards, standards. - Things like WebDAV and whatnot is essential in my view. Although it would be too much work to get an MVP out for now.
- Mobile clients - responsive site may not be possible. A mobile webapp seem to be the way to go as we do not want to support all the platforms.

1.2 Project Layout

This section is currently omitted as the project layout needs some work...

1.3 Team

Projecto is currently maintained by:

- Shuhao Wu [@shuhaowu] <shuhao@shuhaowu.com>

Contributors:

- Erick Cardenas-Mendez <developer@playlistparty.net>

1.4 Getting Help

You can email the maintainers or hit us at #projecto on irc.freenode.net

CHAPTER 2

Part II: New Developer Guide

2.1 Dev System Requirements

For development, it is recommended that you use Linux. The maintainer(s) of the project uses Linux as their primary development platform.

It is also possible to setup vagrant on Mac and Windows. On Mac it should be fairly straight forward (confirmed on Mavericks). Windows does not have NFS and therefore will take some more work.

Again, it is recommended that you use the *Vagrant Setup (Recommended)*. This cuts down the number of dependencies and should be much easier to work with.

Projecto's requirements are fairly light. The vagrantbox only has about 400MB of RAM and it runs just fine.

If you find that setup, especially the Vagrant box, is broken. Notify the maintainer(s) as soon as possible. This is a top priority as new contributors will be blocked and lose interest if the setup instructions do not work.

2.1.1 Vagrant Setup

The requirements for the vagrant setup is fairly light. You need to install the followings:

- Vagrant
- NFS - For Windows, ignore this and follow the instructions in *Vagrant Setup (Recommended)*.
- NodeJS
- Firefox and Chromium for now.
- After you get all of this, go to the project root and run *npm install*.
- Now remember to add *node_modules/.bin* to your *$PATH*.

Instructions for those are available online. So follow them.

Continue to *Vagrant Setup (Recommended)*.

2.2 Vagrant Setup (Recommended)

After you get everything from *Dev System Requirements*, clone the git repository and cd into it and create a file named settings_local.py with the following content:

```
DEBUG = True
SECRET_KEY = "<some random string here>"
```

Save the file. In the project directory, type:

```
$ vagrant up
```

Wait about 10 minutes while everything setup. You must have a working internet connection.

You should see that all tests has passed. After this point, you can do `vagrant ssh` to get into the vagrant box. After that you should be placed in a directory called /projecto. This is synced with your project dir via NFS.

Some convenience aliases has been set in the VM. Type `s` to start the dev server and `t` to server side unittest.

Note that unittests will fail if `DEBUG = False`. This is an issue currently under investigation.

The ip of the box is set to 192.168.33.10. `dev.getprojecto.ml` has an A record pointing to it. Going to http://dev.getproject.ml after you start the server should allow you to access your local instance of projecto.

It is best if you add the following to your host file:

    192.168.33.10 dev.getprojecto.ml

2.2.1 Windows Notes

(This may or may not be broken with the new files feature).

The lack of NFS on Windows is a big deal. So what we can do is redirect the `DATABASE_FOLDER` to be something different. The recommended place is your home directory, which is /home/vagrant.

So, create a database folder in /home/vagrant:

```
$ mkdir /home/vagrant/databases
```

Your settings_local.py file should have the extra line:

```
DATABASES_FOLDER = "/home/vagrant/databases"
```

After this, your unittests should pass. You can also remove the databases folder inside your directory.

2.3 Merging Code

Merging code into Projecto's master is not a very complicated process. You will need to use Github, so make sure you have an account there first.

After you get an account, fork the projecto repository.

2.3.1 Git Workflow

When authoring a patch for a feature or to fix a bug, use git's branching feature. Always branch out to a branch with the issue number in it. Let's say if you're working on issue 42 and it is about refactoring the feed controller, before you start your patch, you should branch out like this:

```
$ git branch refactor-feed-42
```

Of course, the name of the branch is up to you. Try to reference the issue number in the branch name.

Next, you should author your patch. Try to have some descriptive commit messages.

If master receives additional commits after you started your patch, have no fear! If master's change is relatively insignificant, just keep going. Otherwise you should rebase your changes against master:

```
$ git fetch upstream # Assuming that upstream points to the projecto repo.
$ git rebase upstream/master
```

Once you're done your patch, commit and push it to your own repository:

```
$ git commit # commits
$ git push origin refactor-feed-42 # or other branch names
```

Now go to the Github UI and hit pull request. Have a descriptive title and some descriptions of what this patch does and why it is useful. Put "Fixes #<issuenumber>" (example: Fixes #42) at the end of the pull request description.

Wait for review and once you get a thumbs up, your code will be merged.

2.3.2 Unittests

Testing is a very important part of Projecto. Not everything was tested when the project first began. However, all **new** code must have unittests unless it is an extremely trivial change (typo, or things that doesn't affect the core logic).

Additionally, all unittests must pass for code to be merged. So please run all tests.

The clientside tests runs by typing the following command on **host** machine:

```
$ karma start karma.config.js
```

This will test the clientside code in both Chrome, Firefox, and PhantomJS.

The server side tests runs by typing the following command on the **Vagrant box**:

```
$ t
```

CHAPTER 3

Part III: Production setup

3.1 Production System Requirements

The target server is Debian Wheezy. Other environments may work as well but is not supported.

This is yet to be written as the details have not finalized.

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*