Contents

Overview

Tribe is an application for efficient families.

Each member of the family uses Tribe. Family members complete daily tasks to earn points which unlock rewards.

The Problem

- Kids need motivation to do homework and do tasks like cleaning their bedrooms
- Families are busy and find it hard to keep track of what other family members are doing and when
- Family members spend their days in different locations

Aims

Motivation Create motivation for family members to work together on completing day-to-day tasks.

Schoolwork Remind and encourage kids to complete homework.

Accountability Allow parents to check that kids’ homework has been done. Make sure everybody knows the dog has already been walked and fed.

Family Interaction Encourage the family to spend time together and co-operate.

Fun Give a sense of satisfaction and joint achievement and add a fun element to day-to-day tasks.

Requirements Specification

Overview

Roles

Each member of the family who is signed up to Tribe becomes a “Tribe Member”.

A parent, (or both parents, or an older child or child minder) becomes the “Tribe Leader”.

Tasks

Tribe leaders create tasks.

Tasks can be once-off or recurring, eg. daily tasks (“do homework”) or weekly tasks (“take out bins”).

Tribe members complete tasks, and mark them as “done” using either the web interface or mobile app. Optionally a photo can be uploaded as proof that the task has been completed.

Points

A Tribe Leader confirms that the task has been done and the points are awarded to the Tribe.

The accumulated points are visible to all Tribe Members.

Rewards

If the Tribe scores enough points in a given week, Rewards are achieved.

Rewards are set by a Tribe Leader, and are specific to Tribe Members. For example, for parents a reward might be a bottle of wine or cinema trip. For the older kids, the equivalent reward might be pocket money or phone top-up. For the youngest kid, it might be sweets. All Tribe members can view what Rewards (for both themselves and other Tribe Members) are available.

Tribe has a selection of default rewards to choose from, however the Tribe leaders have the option to customise the rewards.

Sample rewards are

- Breakfast in bed
- 1 hour Playstation

- Trip to cinema
- Game of Golf with Dad
- Shopping with Mum

- Day off chores
- Swimming
- Trip to Zoo
- museum visit
- amounts of pocket money
- phone top-up credit
- presents (eg. a toy, new phone)

Categories

Tasks can fit into a category.

Information about how many points have been scored in a certain category are available, giving the Tribe an idea of where they are doing well and where improvements are needed.

Tribe provides several categories by default:

- Household
- Diet
- Fitness
- Pets
- School

Entity Relationship diagram

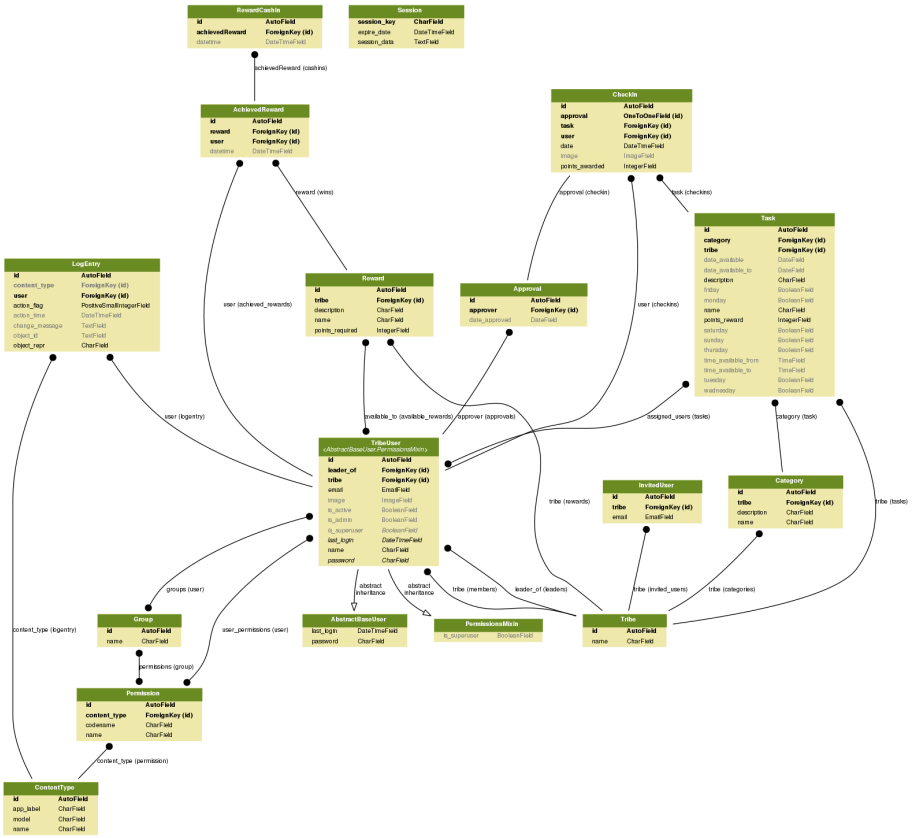

Figure 1: Entity Relationship diagram crated using Graphviz http://www.graphviz.org/

User Stories

As a new user I want to read information about Tribe as I am thinking about signing up

---

User Story: As a new user I want to sign up to Tribe so that I can organise my family or because I have been invited to an existing Tribe

Data required:

- email address
- name
- family name
- password

User Story: As a Tribe Leader I want to add my family members to the tribe because I want to have my whole family set up using Tribe

Data required:

- email address

User Story: As a Tribe Member I want to sign in so I can use features of Tribe

- email
- password

As a Tribe Member I want to edit my profile because I want to add details about myself and choose a picture.

- name
- image

As a Tribe Leader I want to add, edit, or delete a task because I have thought of a new task that needs to be done or one that needs to be changed

- task name
- description
- task category
- number of points rewarded for completing the task

- availability details
- assigned Tribe Members

As a Tribe Member I want to see a list of tasks currently avalable to me because I want to earn points

- task details
- points reward

As a Tribe Member I want to mark a task as done because I have completed a task and want to earn points for it.

As a Tribe Member I want to see the number of points my tribe has accumulated this week because I want to see how close we are to winning a reward.

As a Tribe Member I want to see the number of points my tribe has accumulated overall for motivation

---

As a Tribe Member I want to see how many points the tribe has accumulated on a per-day basis because I want to see which days we are better on

---

As a Tribe Leader I want to create, edit or delete a reward which can be won by Tribe Members because I want to create incentives

- reward name
- image
- optional description

As a Tribe Leader I want to create a new Category or edit existing Categories because I want points to control how points are tracked.

- name
- optional description

As a Tribe Member I want to view my or another Tribe Member's profile because I want to see how they are doing and see what Rewards they have gained

- name
- stats on points and tasks
- rewards gained

As a Tribe Member or Tribe Leader I want to be able to do all of the above using the REST API to allow for new clients and flexibility

---

As an Admin of Tribe I want to be able to modify any of the above entities using an administration interface

---

Use Case diagram

Tribe Leader: This is usually a parent but can be any tribe member (eg. babysitter)
Tribe Member: Any member of the family/tribe who has signed up
Admin: The administrator of a Tribe instance (server, database administrator etc.)

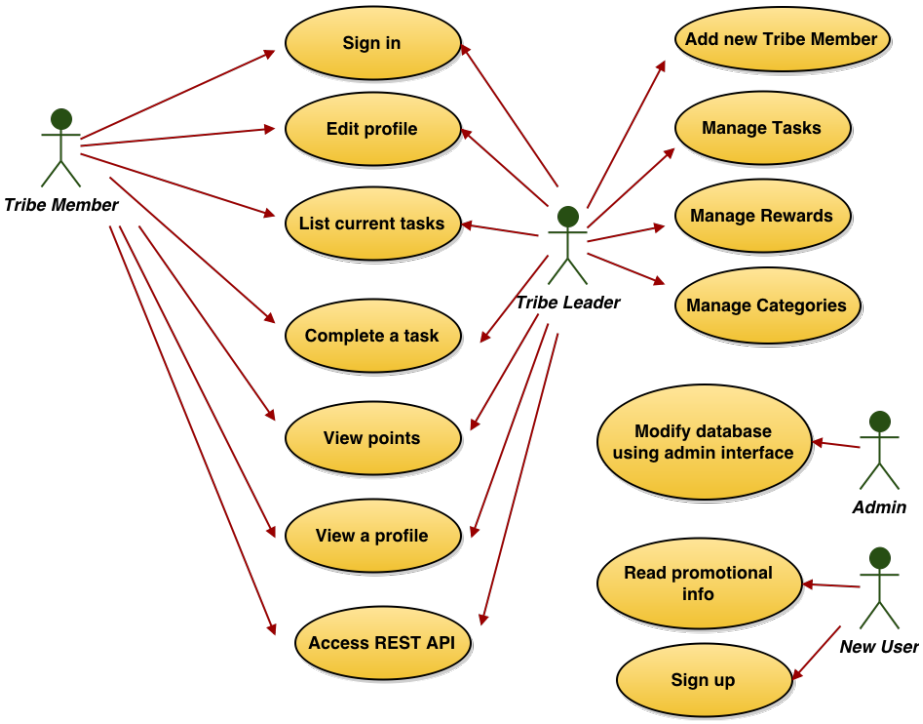

Figure 2: Use Case diagram from https://www.draw.io/

User Manual

Web interface

Registration

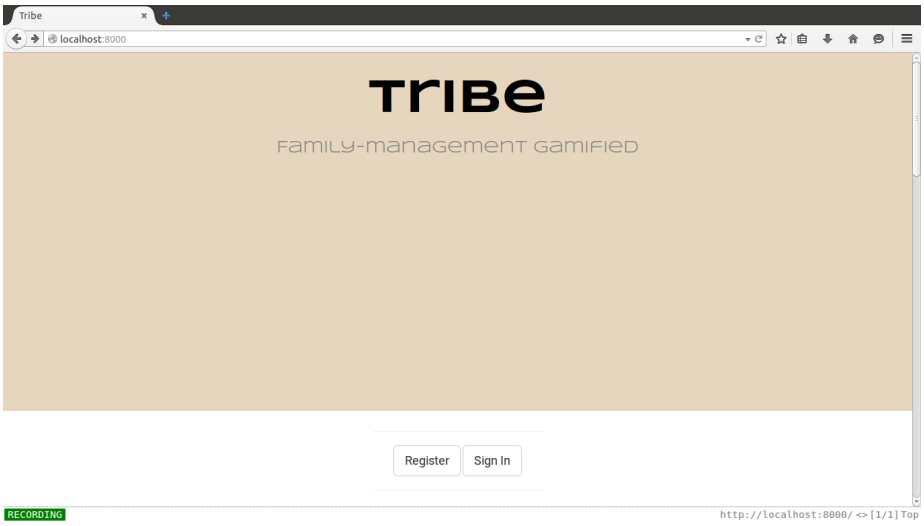

Figure 3: The home page

When first accessing the web interface, the user will be presented with the home page. Scrolling down here will give an overview of the application so the user can decide if they want to sign up.

To register as a new user, click the register button. This will present the registration page.

Fill in the required information on the registration page. You will need to enter an email address, name and password.

If any of the information is invalid the user interface will alert you and not allow you to progress.

You will then be prompted to enter the name of your "tribe" (family or group).

You will then be shown the Tribe page with a list of Tribe members.

Inviting new Tribe Members Click the "Invite new members" button.

Enter the email addresses of family members here. They will be displayed as pending under Invitees. When they register an account they are added to your Tribe and removed from Invitees.

User Profiles To view details about a Tribe Member click their name or image on the Tribe page.

To edit a profile, click on the Edit button.

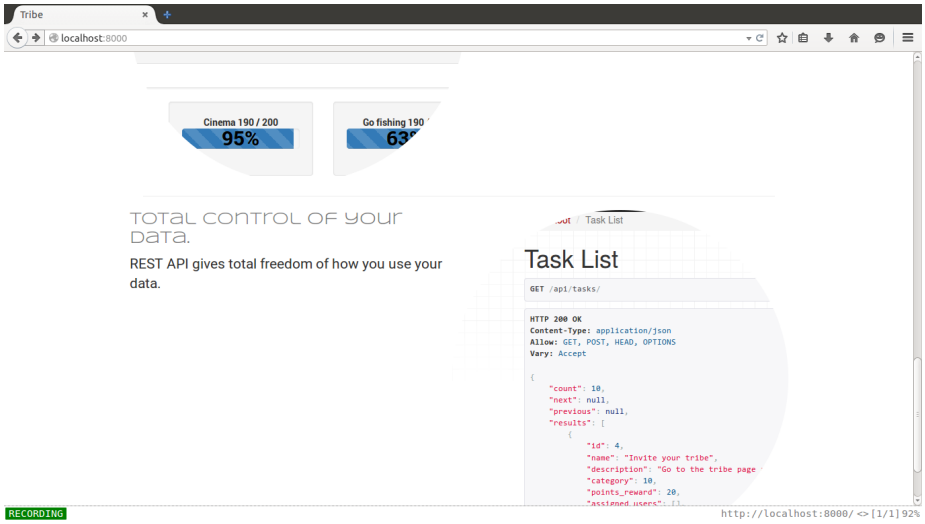

Figure 4: Promotional information on the home page

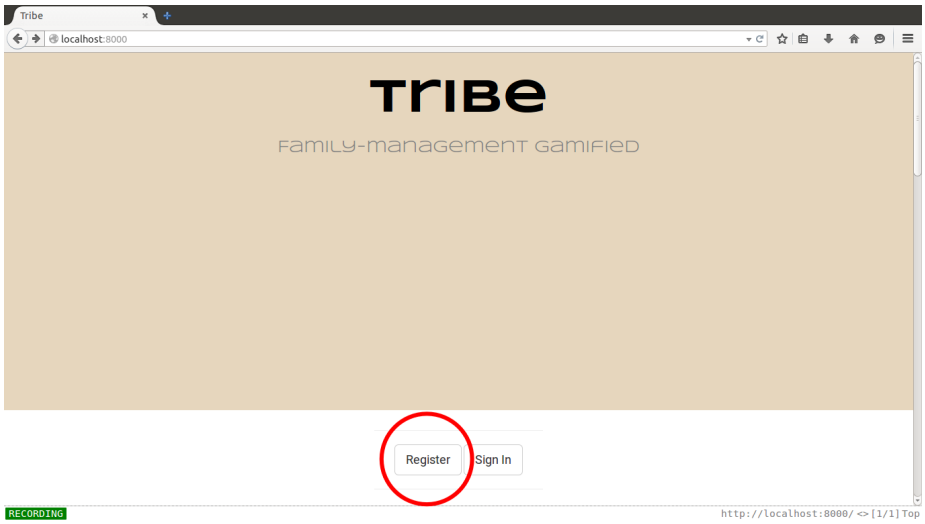

Figure 5: Register button

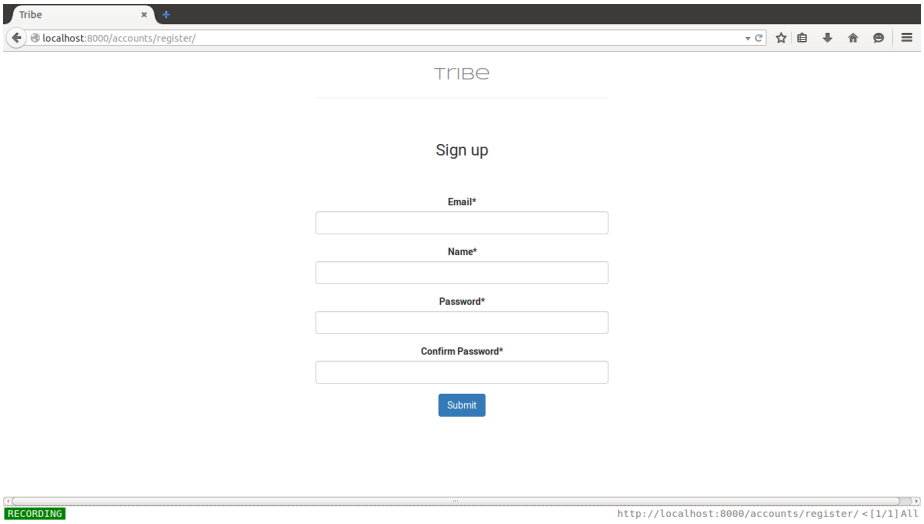

Figure 6: Registration page

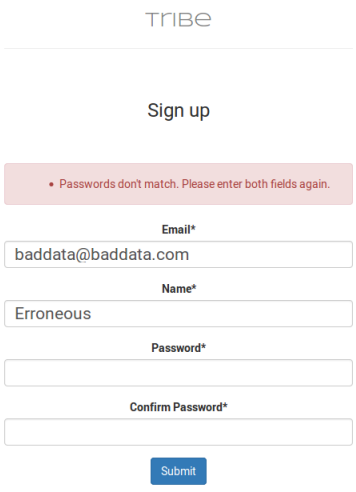

Figure 7: Invalid data entered

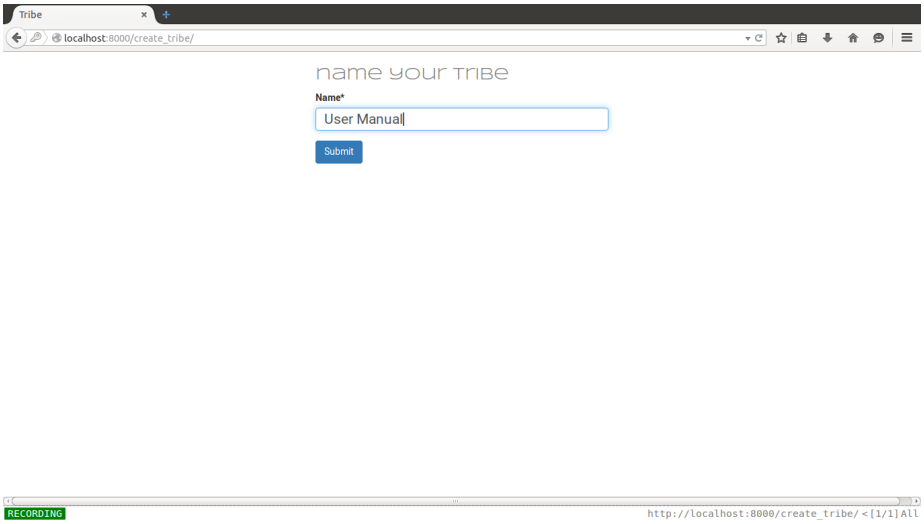

Figure 8: Naming your tribe

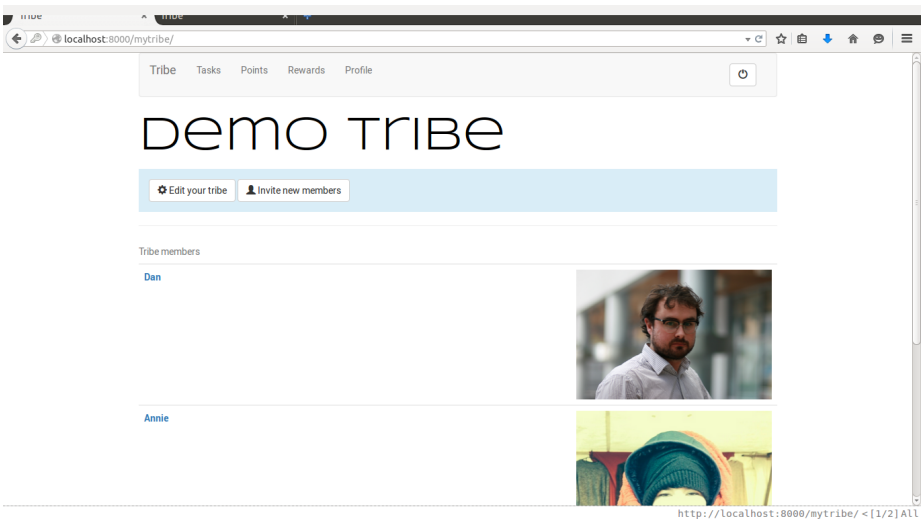

Figure 9: Tribe page

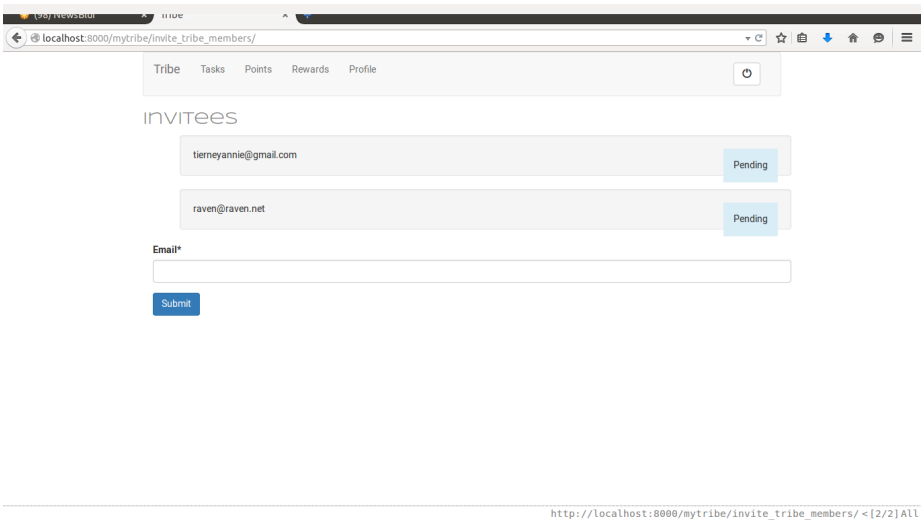

Figure 10: Inviting new Tribe Members

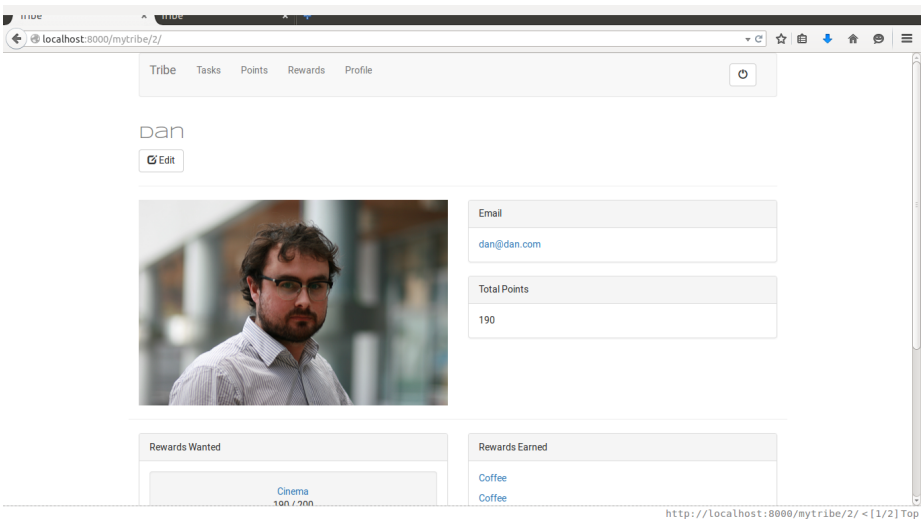

Figure 11: A Tribe Member Profile

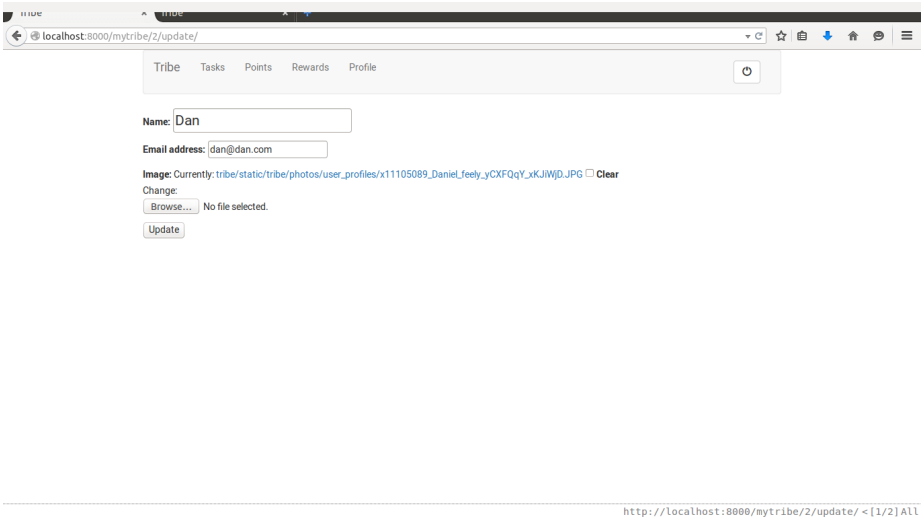

Figure 12: Editing a Profile

Mobile App

REST API

Technical Report

Python

Tribe was developed in the Python language. I found that the Python language had good documentation available online such as this comprehensive tutorial: http://learnpythonthehardway.org/. I found that syntax for Python was quite similiar to Java which I had studied on my course.

Django

Django was the MVC web framework which I used. I found Django was similiar to Ruby in Rails which I had also studied.

Django provides an ORM - Object Relational Mapper, which is used to store Python objects (models) in the database. A sample model from my project looks like as follows:

```
class Task(models.Model):

    def __str__(self):
        return self.name
```

```
 name = models.CharField(max_length=200)
 category = models.ForeignKey(Category)
tribe = models.ForeignKey('tribe.Tribe', related_name="tasks", null=True)
 description = models.CharField(max_length=200)
 points_reward = models.IntegerField()
 assigned_users = models.ManyToManyField(
                                     'tribe.TribeUser',
                                     related_name="tasks",
                                     blank=True,
                                     )

 #Available days
 monday = models.BooleanField(default=True)
 tuesday = models.BooleanField(default=True)
 wednesday = models.BooleanField(default=True)
 thursday = models.BooleanField(default=True)
 friday = models.BooleanField(default=True)
 saturday = models.BooleanField(default=True)
 sunday = models.BooleanField(default=True)

 #Available from/to
 time_available_from = models.TimeField(null=True, blank=True)
 time_available_to = models.TimeField(null=True, blank=True)

 #Available date
 date_available = models.DateField(null=True, blank=True)
 date_available_to = models.DateField(null=True, blank=True)

 @property
 def available_now(self):
  return self.checkIfAvailable(datetime.datetime.today().date())

 def available_to(self, user):
     # if no assigned users assume available to all
     if (not self.assigned_users.exists()):
         return True

     # if in assigned users it's available
     if (self.assigned_users.filter(id=user.id).exists()):
         return True

     return False

 def checkIfAvailable(self,date):
```

```
    # if already completed
    if CheckIn.objects.filter(task=self):
        return False

    if self.date_available:
        if self.date_available_to:
       if self.date_available <= date <= self.date_available_to:
                return True
            else:
                return False
        elif self.date_available != date:
            return False
        ## check time
        return True

    # below happens if no date_available set
    day_of_week_of_date=date.weekday()

    days = {
        0: self.monday,
        1: self.tuesday,
        2: self.wednesday,
        3: self.thursday,
        4: self.friday,
        5: self.saturday,
        6: self.sunday,
    }

    if days[day_of_week_of_date] == True:
        return True

    return False

@property
def has_been_checked_in_on(self):

    if self.checkins.all():
        return True

    return False

@property
def days_remaining(self):
    if self.date_available_to:
    td = self.date_available_to - datetime.datetime.now().date()
        return td
```

The Fields are mapped to rows in the database and metadata such as `max_length` can be added.

Source Control

Git was used for source/version control. This allowed me to keep track of changes and roll back if necessary.

Github (https://github.com/) was used to store my Git repository. Github offers up to five free private repositories to students through the Student Developer Pack. I was able to sign up to the programme using my NCI email address: https://education.github.com/pack

Most students will be familiar with Github however during the course if this project I learned about some extra features it has in terms of graphing and statistics. One that I found interesting was the Punch Card which shows on what days you push the most commits. As I'm a part-time student the majority of my commits were weekends and evenings.

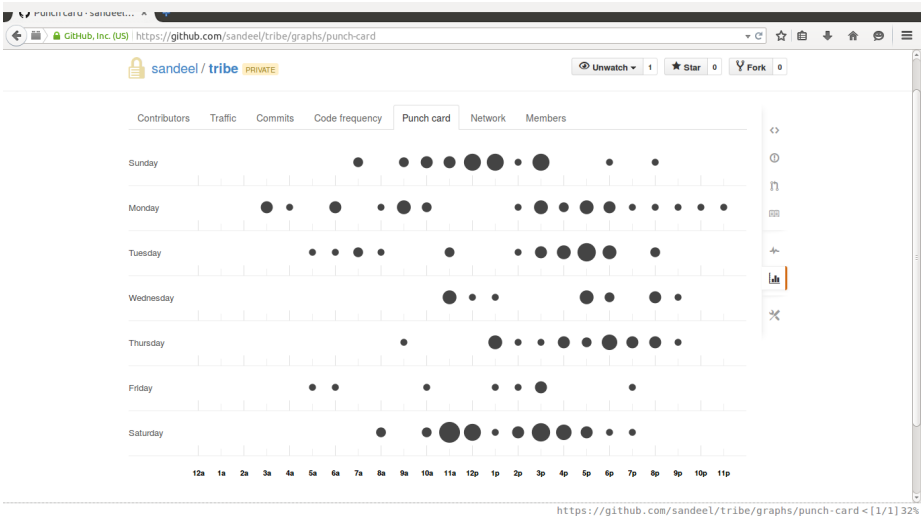

Figure 13: Github Punch Card for my project

AGILE/SCRUM

I used an Agile development process to manage my project. After my requirements document was completed I divided the User Stories into Sprints. I then mapped out each Sprint using online Scrumboard Taiga.io (https://taiga.io/).

In my opinion using an AGILE development process was very beneficial. It allowed me to keep track of what elements still needed to be coded and also that I kept on track to complete the project on time. I found Taiga.io was very user friendly and allowed me to visualise my progress and also to quickly add or amend tasks ang User Stories. I would highly recommend Taiga to other students who are using SCRUM.

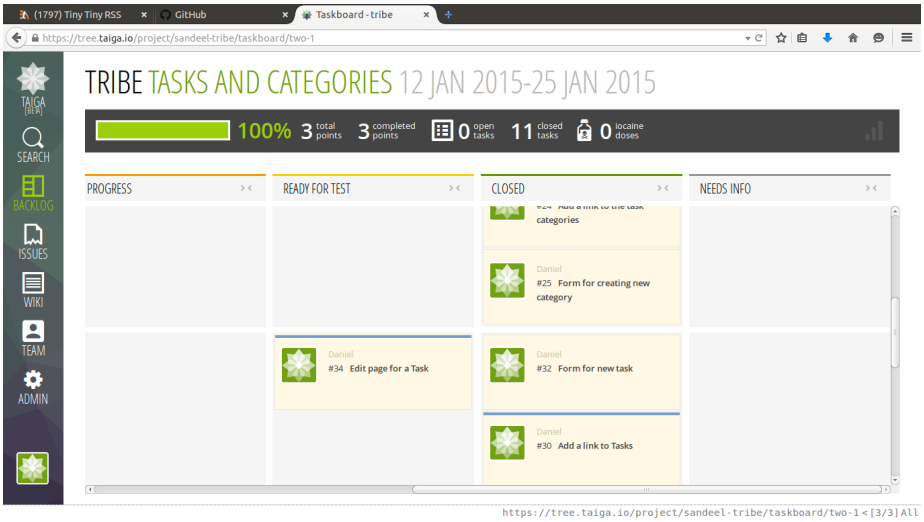

Figure 14: Taiga.io Scrumboard

Testing

Unit Tests

I wrote Unit Tests for each of my classes. After changing a signficant part of the codebase I ran the unit tests to see if anything had been broken.

Unit tests comprised of a significant amount of code and were time-consuming to write, however I think they were of benefit in the long run. As the application became more complex the tests ensured that individual elements still functioned correctly.

An example of unit tests testing individual methods of a class:

```
def test_check_if_available_by_date(self):
    """
    Ensure Tasks only available on date set
    """
    task = Task()

    todays_date = timezone.now().date()
```

```
    some_day_last_week = (todays_date - timezone.timedelta(days=7))

    task.date_available = some_day_last_week
    assert(not task.checkIfAvailable(todays_date))

    task.date_available=todays_date
    assert(task.checkIfAvailable(todays_date))

def test_check_if_available_by_day_of_week(self):
    """
    Ensure Tasks only available on the days of week set
    """
    task = Task()

    todays_date = timezone.now().date()

    task.monday=False
    task.tuesday=False
    task.wednesday=False
    task.thursday=False
    task.friday=False
    task.saturday=False
    task.sunday=False
    assert(not task.checkIfAvailable(todays_date))

    task.monday=True
    task.tuesday=True
    task.wednesday=True
    task.thursday=True
    task.friday=True
    task.saturday=True
    task.sunday=True
    assert(task.checkIfAvailable(todays_date))
```

I also wrote tests to ensure my REST API was returning the expected data. Example:

```
def test_create_task_via_api(self):
    """
    Ensure we can create a new task object via API
    """
    url = reverse('task-list')
    self.client.login(email=self.user.email, password='password')
    response = self.client.post(url, self.data, format="json")
   self.assertEqual(response.status_code, status.HTTP_201_CREATED)
```

```
        """
        Ensure redirects to login when not authenticated
        """
        self.client.logout()
        response = self.client.post(url, self.data, format="json")
    self.assertEqual(response.status_code, status.HTTP_403_FORBIDDEN)
```

Continuous Integration

As well as manually running my Unit Tests, I also researched Continuous Integration. Continuous Integration refers to a system of automatically running tests aftir code is committed to ensure the build has not been broken.

Travis CI (https://travis-ci.com/) was used for Continuous Integration on my project. Travis CI is a web service which links to a Github account to provide continuous integration. Travis CI is notified when your Github repository is updated and then clones the repository and runs the instructions in the .travis.yml file to download the required dependencies and run the tests for your application. If the build fails the service will notify you via email. It will also email you when the build moves frim being broken to being fixed.

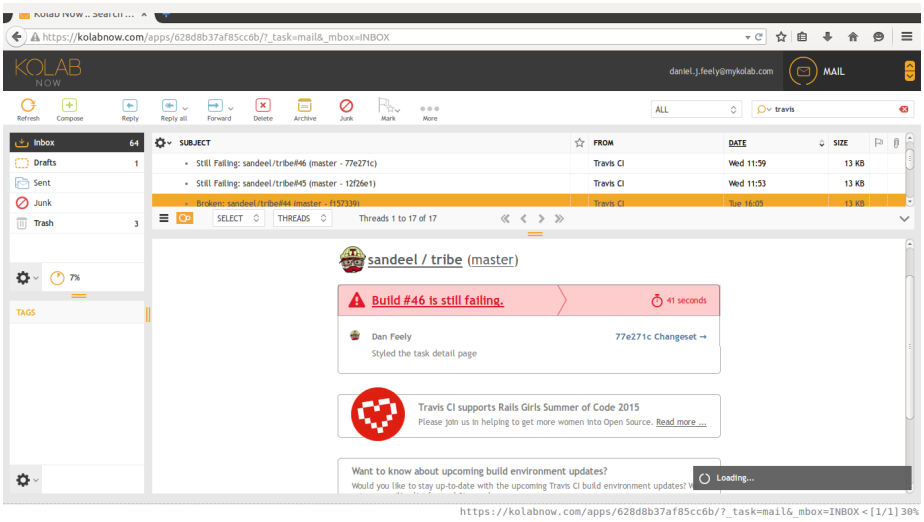

Figure 15: Travis CI build broken notification

Using Travis CI to activate continuous integration with my project's repository was not time-consuming and was user-friendly. I would highly recommend this service to other students.

I created a .travis.yml file in my repository's base directory. The travis.yml had the following content:

I set the language to Python so Travis knows which interpreter to use.

```
language: python
```

Tell Travis the Python version. This is very important as between Python versions especially version 2 to 3 code may become invalid.

```
python:
  - "3.4"
```

The following tells Travis to install the required libraries for my project. The requirements.txt file is in the base folder of the repository and is simply a list of my required libraries and their versions.

```
install:
  - pip install -r tribe/requirements.txt
```

The next section tells Travis what it has to do to run my tests. Each line of this represents a command to be run.

```
script:
  - cd tribe
  - python manage.py test
```

Signing up for a Travis account was quite straightforward. It is important to note that for my purposes travis-ci.com was used and not travis-ci.org, because travis-ci.org only allows for public accounts and for academic purposes I needed to keep my builds private.

I signed up for the free trial using my Github account. This was a one-click operation. I was then able to navigate to the billing settings and notify Travis that I had a Github student account. Travis are part of this plan so I was then allowed free access to a private account.

I was then able to switch on or off repositories from my Github account.

Travis-CI helpfully then sent me emails stating when my build had failed and when it was fixed.

The build can be watched in real-time if you log into your Travis account and then push to your Github repository.

There are also various tools available on Travis's web interface. One of the more useful views I found was the "Build History" view.

I was also able to put a "light" on my repository's Github page which turned green when build was passing and red when build failing.

Overall I think that while using continuous integration didn't gain me a huge amount because I was only a one-person team, it still had some benefits and didn't take too much time to set up. It was also of benefit to learn this skill as I can see how continuous integration would be extremely useful in large teams who are working together on a single codebase.

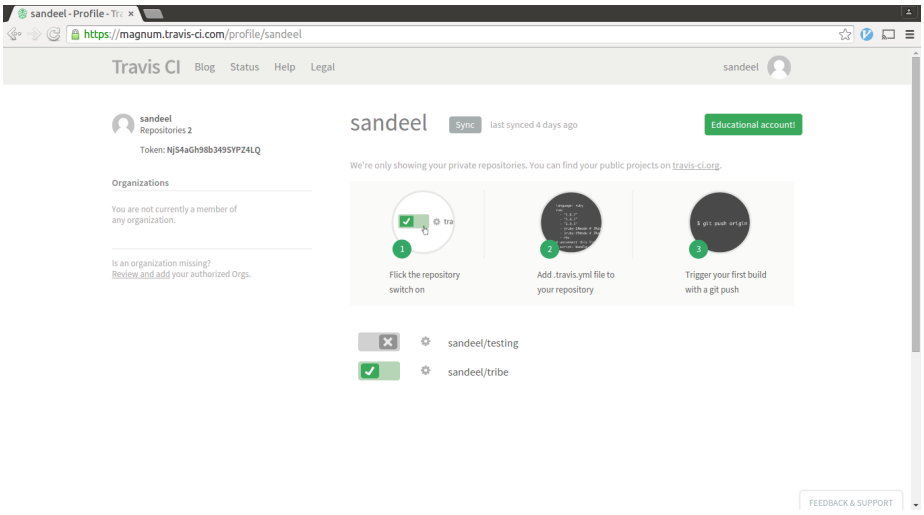

Figure 16: Activating repositories in Travis-CI

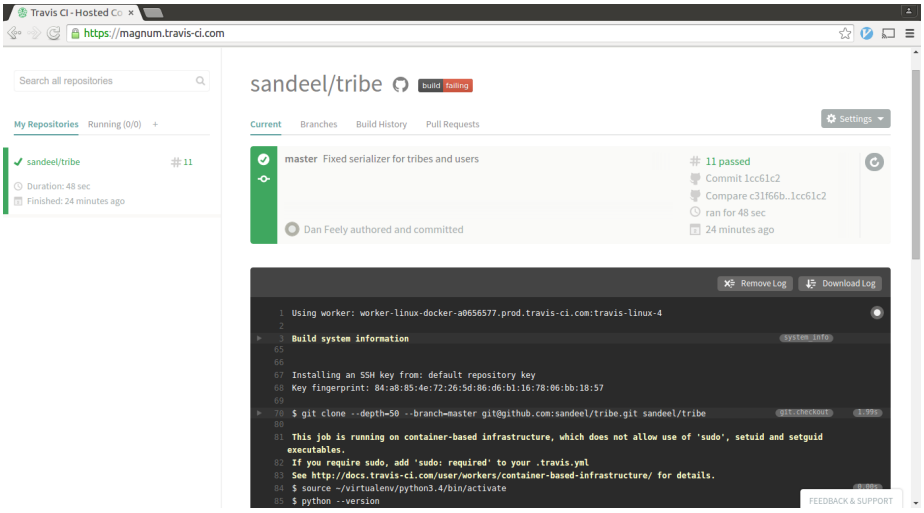

Figure 17: Real-time build on Travis server

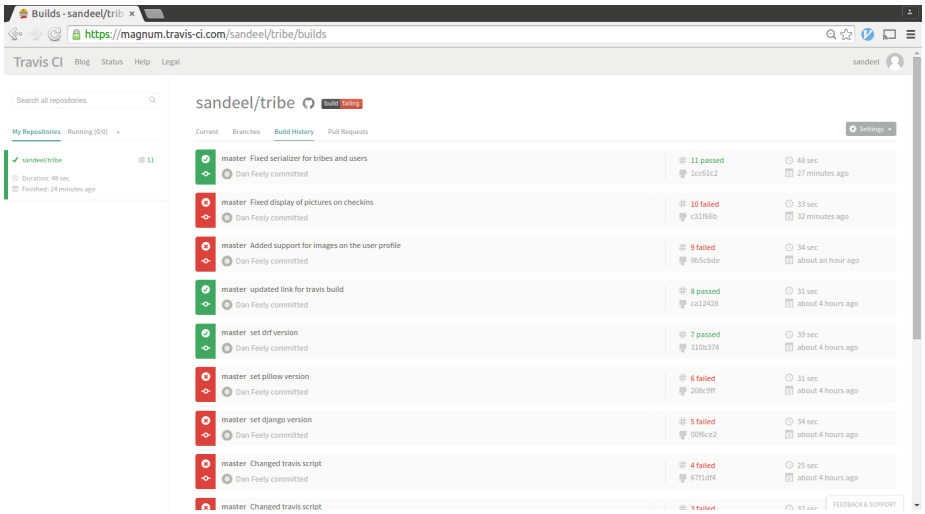

Figure 18: Travis-CI build history for my project

Deployment

To deploy my application to cloud hosting I considered both Amazon EC2 and Digital Ocean. I had used both of these services on previous projects.

Amazon EC2 provides a lot more flexibility than Digital Ocean. However it is also overwhelming as there are a lot of settings per server. EC2's billing is also slightly confusing as it is pay-per-use.

Digital Ocean offers a more simplified billing system and also has a simple set-up process. I decided to go with Digital Ocean for this reason and because I was happy with the service on previous projects.

I deployed to my Digital Ocean droplet using Git.

```
apt-get update
apt-get install git
git clone https://github.com/sandeel/tribe.git
cd tribe/tribe
```

To install the python libraries needed, I first had to install Python's package manager, PIP.

```
apt-get install [[python-pip]]
```

And some Python development libraries:

```
apt-get install python-dev
```

I stored all additional requirements for the project in a text file called requirements.txt. PIP can parse through this file and install all the dependencies:

```
pip install -r requirements.txt
```

Then to run the development server for testing:

```
python manage.py runserver 0.0.0.0:8000
```

My project is available at the url http://46.101.10.24/

Timeline

2014-12-08

- Came up with a name for the application
- Created the git repo locally and on Github
- Researched several different time-tracking and AGILE solutions
- Decided to go with Taiga.io
- Created a project for tribe on Taiga.io
- Created some initial user stories on Taiga.io

2014-12-12

- Created Powerpoint slides for the mid-point presentation
- Put together a very rough demo of the application using Django

2014-12-13

- Presented the idea and received feedback from Eamonn

2014-12-14

- Put together my Requirements Specification document
- Submitted the Requirement Specification to Moodle
- Emailed supervisor my Requirement Specification requesting feedback
- Started putting all user stories from requirements into Taiga.io
- Creating a list of all entities in the system

2014-12-16

- Started coding the basics of the system, eg. user accounts, log-in
- Created directory layout, code repository, etc.

2014-12-17

- More work on the system basics
- Began researching Django REST Framework and reading the documentation

2014-12-18

- Completed the tutorial for the Django Rest Framework: http://www.django-rest-framework.org/tutorial/
- Researched the HTTPie tool https://github.com/jakubroztocil/httpie as a means of testing my REST API

Sprint 1: Users and Tribes (28th Dec 2014 - 11th Jan 2015)

2014-12-28

- Mapped out 10 sprints in detail on Taiga.io leading up to project completion date

2014-12-29

- Continued tutorial on Django Rest Framework
- Began to prepare the project documentation

2014-12-30

- Completed the tutorial on Django Rest Framework

2014-12-31

- Researching and mapping out permissions in the Django Rest Framework
- Created permission for allowing unregistered users to create a new user (eg. for registering via mobile app)

2015-01-01

- Almost completed sign up new user by API

2015-01-02

- Completed sign up via API and also implemented creating a tribe via the API

2015-01-04

- Started to implement inviting new users.

2015-01-10

- Working on adding new users to a Tribe. Almost ready for test.
- Uploaded December diary to Moodle

2015-01-11

- Finished form for adding a new tribe member to your tribe
- Wrote unit test for above functionality

Sprint 2: Tasks and Categories (12th Jan 2015 - 25th Jan 2015)

2015-01-12

- Added models for Tasks and Categories.
- Added a view for Task

2015-01-14

- Working on the models/database tables for Task
- Form for new Task

2015-01-14

- Forms for editing Tasks and viewing details.

2015-01-20

- Added models, forms, views etc. for Categories
- Added some more user stories to Taiga.io project
- Got most of the user stories laid out in sprints in Taiga.io

2015-01-21

- Created an API serializer for Categories

- Created API endpoint for creating and viewing categories
- Wrote tests for creating Categories via API

2015-01-22

- Created a serializer for Task
- Created API endpoints for Task

2015-01-25

- Started refactoring the code so that the forms use the app's own API

Sprint 3: Task Completion (26th Jan 2015 - 08th Feb 2015)

2015-01-26

- Created CheckIn class.

2015-01-28

- Added a "Check In" button to tasks which creates an instance of a check in for a user on that task and awards points. This uses the API in the background.

2015-01-31

- Created an approvals system for tribe leaders to approve tasks
- Started working on some test/example data

2015-02-01

- Wrote several more tests

2015-02-02

- Wrote more tests
- Investigating and reading about Apache Cordova for mobile app
- Started working on the mobile app.
- Researching Ngrok to publicly host the site for the app to communicate with.

2015-02-03

- Working more on the mobile app and working on styling to ensure site looks ok on both mobile and desktop
- Added basic points calculation for users (done by counting up the points awarded for each checkin which has been approved)

2015-02-04

- Redesigned home page and user interface (prototype design for finished product). Started CSS style sheet for site.
- Changed permissions for some views eg. Task list.

2015-02-05

- Investigated continuous integration service Travis CI.
- Created account with travis and config file to tell it how to test my code
- Builds passing. Added small button to my project's github page to indicated if builds are passing or not.
- Added a collapsible navigation sidebar to the site. Ensured it also works on mobile

Sprint 4: Points System

10th February - 22nd February 2015

2015-02-17

- Created calculation methods for total points for a Tribe. (still need to write tests for these)
- Created a display for these points on a per-day basis on the points page

Sprint 5 "Rewards"

2015-02-24

- Created a model for a Reward
- Created serializer for a Reward
- Created views for adding Rewards

2015-02-26

- Created views for editing rewards
- Researching drawing graphs in HTML5
- Researching the native Progress element in HTML5
- Added a basic progress bar for weekly points using bootstrap for the styles

2015-03-10

- Revamped the UI

2015-03-11

- Working on rewards and points required for rewards
- rewards can now be assigned to users

2015-03-14

- Changes to the user model. New users now need a Name
- Added tests to ensure new users have names and get a default tribe created
- Researched testing JSON responses
- Started to create a test family which gets created by a script (for test data)
- Worked a lot on created test data. To do this I had to make tweaks to some models
- Redesigning the tasks page

2015-03-18

- Researching Django Crispy Forms
- Designed the login page

2015-03-19

- Working on the points page. Now shows the rewards and a users progress towards rewards.
- Points page now shows points today, this week, and total points.
- Still working on the script for making fake data. Much more useful now as makes use of a Python fake data generator which creates random users, tasks and rewards.

2015-03-20

- Working on the user's profile page.
- Changes to form for editing user.

2015-03-22

- Working on progress bars for rewards on the points page.

2015-03-23

- More work on the models for Rewards and Achieved rewards.
- Points page now shows complete when reward achieved.
- Achieved rewards now show on the user's homepage
- Changes to the test data generator

2015-04-16

- uploaded showcase information

2015-04-18

- Worked on getting tests working with Travis-CI
- Added user photo support
- Redesigned profile page to allow for photos

2015-04-25

- Styled the sign-up page
- Added to the script for creating test data
- Changes to Tribe management

2015-05-02

- Changed some styling
- Changes to the points page

2015-05-03

- Working on the look of the site
- Working on documentation

2015-05-05

- Redesigned the tasks page
- Split tasks out into categories
- Implemented bootstrap accordion

2015-05-06

- Redesigning a lot of the interface
- Modifying the stylesheets to add colour

- Working on the documentation

2015-05-07

- Added the promotional text to the homepage

2015-05-11

- Testing the mobile interface
- Finalising documentation
- Trying out designs for the poster