

Platinum Analytics

Final Year Project Report

DT228

BSc in Computer Science

**Peter Collins**

**<Supervisor>**

School of Computing

Dublin Institute of Technology

**<Date>**



Abstract

### Platinum Analytics is a sports data analytics system which aims to gain insight into player performance by means of collecting and analysing player metrics with the aid of wearable sensor technologies.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<Student Name>

<Date>

Acknowledgements

Body text

Table of Contents

*(These are suggested contents only. You can add or modify if required)*

1. Introduction
   1. Overview of the project and the background behind it.
   2. Project Objectives
   3. Project Challenges
   4. Structure of the document
2. Research
   1. Eg. Research related to identifying the problem that this project solves, research into solution definition

Algorithm

Current Solutions

My program is an opensource solution which provides a sports analytics framework based on non-proprietary hardware and that is what sets it apart from competitors.

Tech Research

* 1. An overview of the technologies evaluated and selected or rejected and the rationale behind the key decisions.

1. Design
   1. Identification of a design methodology including why it was chosen
   2. Design of each of the project components eg: the UI, Network, Project Demonstration, source code layout
   3. Clearly identifying the list of features and use cases supported within the project.
2. Architecture & Development
   1. Overview of the system architecture and a diagram to represent all of the key elements within the architecture.
   2. Details of each component within the project, problems encountered and resolved, challenges overcome or worked around.
   3. Identify key development components;
   4. Identification/explanation of external APIs used versus own code ; List of classes of your code etc .
3. System Validation
   1. Testing
      1. What testing was performed, why it was selected and what are the key use cases within the project.
   2. Demonstration
      1. Identify what features can be demonstrated and show screen shots or reference a video online to show the project demonstration (for audience not at demo)
4. Project Plan
   1. Project Plan analysis and review of how it changed from the initial proposal including explanation of what changed and why, and suggestions on how to address this if the project was repeated.
5. Conclusion
   1. Analysis of the projects key elements identify the key learning obtained from the project and recommendations and suggestions for how the work can be improved on continued into the future.
6. Bibliography
7. Appendix

## 1. Introduction

## 2. Research

1. Research
   1. Eg. Research related to identifying the problem that this project solves, research into solution definition

Algorithm

Current Solutions

My program is an opensource solution which provides a sports analytics framework based on non-proprietary hardware and that is what sets it apart from competitors.

Tech Research

* 1. An overview of the technologies evaluated and selected or rejected and the rationale behind the key decisions.

### Introduction

#### Trilateration Algorithm Research

#### Current Alternative Solutions to Sports Analytics

##### Fitbit

Statsports

#### Technologies Evaluated, Selected and Rejected

##### Git

Git is a version control tool for managing code. More on git

###### Branches

**master** - This branch only contains finished releases. The code should always be in a production ready state, meaning that it is ready to be deployed with no more development or unit testing required.

**develop** - This branch is the branch with the latest delivered changes which are ready for release. This can also be known as a continuous integration branch in which multiple new features can be tested for compatibility with each other.

**feature** - There are feature branches for each different deliverable within the scope of the project.

**release** - This branch is created off the develop branch and merged back into master branch. Release branches represent all code from developed features which are tested and complete and ready for deployment. Release branches should be named *release* followed by the version number.

This branching scheme has been inspired by git workflow

###### Initialize a directory

Initialize a git repo in the current directory and fetch the platinum-analytics codebase.

git init

git remote add origin [git@github.com](mailto:git@github.com):peteehb/platinum-analytics.git

git fetch

git pull origin develop

###### Committing

A commit command is issued when a developer wants to store any changes made. The developer adds the changed files to the commit and finishes with a message to describe those changes. A push command is used to sync the local branch with the remote branch on github. Before issuing a push request, a pull request is issued which retrieves any changes which may have been made by other users to the same branch.

git pull

git add .

git commit -am "Commit Message"

git push

###### Branching

Branching in GIT allows the developer to test new features without changing the develop branch. The developer has no commitment to add the changes should they fail to meet the requirement of the feature. Any successfully developed and tested feature can be merged back into the develop branch.

git branch -b new-feature develop

###### Merging

When development is complete on a branch the changes are merged back into the develop branch by either opening a pull request or merging the branch with the merge command.

git checkout develop

git pull --no-ff

git merge new-feature

git push

When working in a team it is usually better to create pull requests instead of merging feature branches. The main advantage of using pull requests is that they give a way for other team members and managers to review code before it is accepted into the main branch. Github shows every change made to files, displaying updates and also changes which will cause conflicts.

###### Conflicts

Conflicts are collisions which arise "when two branches change the same part of the same file and then those two branches are merged." [help.github.com Resolving conflicts from the commandline] To resolve conflicts..resolve conflicts

###### Release a Version

When all of the development in the current scope of the project is completed a version can be released. A version branch is created off the master branch with a given version number. This branch is then tagged, which creates the release.

git checkout -b release-1.2 develop

git checkout master

git merge --no-ff release-1.2

git tag -a v1.2

git checkout develop

git develop

git merge --no-ff release 1.2

git push

##### SSH

Ssh was used to run commands on the raspberry pis. It allows remote login to the rpi, returning a console in which to run commands.

By placing my laptop's public into */home/pi/.ssh/authorized\_keys* I am able to login to the pi without entering a password. This process is repeated with the pi's keys placed on the db server. This facilitates the usage of scp to sync local files onto the db server, which also requires no input from the user.

SSH is also used to access the application and database servers to run commands to initialize these components.

##### VIM

Vim is a text editor found on almost all Linux systems. It is used to edit text files such as configurations on the Raspberry Pis, APP and DB servers.

##### CRON JOBS

A cron job is used to launch a python script which starts the bluetooth monitoring on the Raspberry Pis. This script is run on startup so the user does not need to interact with the pi at all once it has been powered on.

sudo crontab -e

@reboot /opt/platinum/mon/startup.sh

@reboot /opt/platinum/mon/syncfiles.sh

###### startrup.sh

python StartProgram.py 1 False

Another cron job is scheduled to run a python script to check if an internet connection has come available. If it has detected a wifi signal of a known network and established a connection to that network, the python script copies all local SensorReadings csv files to the application server.

###### syncfiles.sh

python SyncLocalFiles.py

### Conclusion

## 3. Project Design and Methodology

### Introduction

In this section I will give an overview of the software development methodology which was used and why it was chosen. I will also discuss in detail the design of each of the system's components, listing each components features and which use cases they provide for.

### Software Methodology

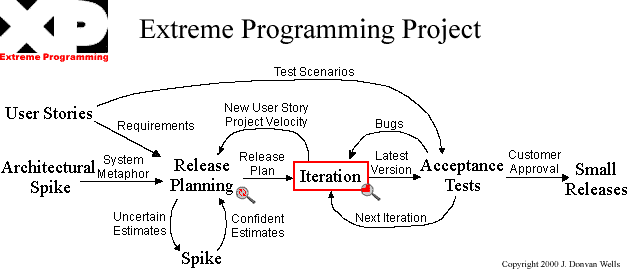
#### Introduction

Discuss software methodologies. Find sources

#### Identification of Methodology

Discuss xtreme programming

For this project I used an agile methodology. The Agile Process used was Extreme Programming. According to this definition from extremeprogramming.org, Extreme Programming "is a lot like a jigsaw puzzle. Individually the pieces make no sense, but when combined together a complete picture can be seen."



Xtreme facilitates the rapid prototyping of software which

### Use Cases

### Project Components

In this section I will give a detailed overview and presentation of each component in the project and detail how the components communicated with each other within the system.

#### Bluetooth Wearable Sensor Sensortag

*Sensortag* is the name of the chosen sensor which players will wear while being monitored by the Platinum Analytics program.

#### Bluetooth Monitor PNM-MON

*pnm-mon* is a Python application deployed on the three Raspberry Pi computers used for collecting the information from

#### Web Frontend PNM-WEB

*pnm-web* is the name given to the frontend web application which provides the web interface for users to interact with Platinum Analytics.

#### Database PNM-DB

*pnm-db* is a postgres backed SQL database server which is used for the persistent storage of all of the program's data. A REST API sits in front of the database which serves the program's data to users through *pnm-web* over the HTTP protocol.

### Conclusion

In this section I gave an overview of the software methodology which was employed in the project and why it was chosen, a list of the use cases which outline what functionality was required at a high level and an overview of the projects components.

1. Architecture & Development
   1. Overview of the system architecture and a diagram to represent all of the key elements within the architecture.
   2. Details of each component within the project, problems encountered and resolved, challenges overcome or worked around.
   3. Identify key development components;
   4. Identification/explanation of external APIs used versus own code ; List of classes of your code etc .

## 4. Architecture and Development

### Introduction

In this section I will give an overview of the system architecture representing all of the key elements which make up the project's architecture. I will also detail each of the components in the project

### System Architecture

Platinum Analytics is a multi-tiered application consisting of several hardware components. In this section I will detail the initialization, development and deployment of each component as well as giving an overview of how the system components interact with each other.

#### PNM-Mon - Architecture Setup & Deployment

*PNM-Mon* is the name given to the Python based Bluetooth monitoring program responsible for collecting data from the sensors worn by the players which runs on Raspberry Pi Linux computers. In all there are three Raspberry Pi computers deployed in this project, the minimum number required to perform the trilateration algorithm detailed previously in Chapter 2.

The setup of the Bluetooth monitoring component required the acquiring of a minimum of three Raspberry Pi computers. Each Raspberry Pi in turn required a Bluetooth radio, a Wi-Fi radio, a battery pack and a memory card to run the operating system off.

Two types of Raspberry Pi were used in this project, one Model 2-B and two Model 3-Bs. The Model 2B does not have onboard Bluetooth or Wi-Fi radios and so USB alternatives were used instead. The Model 3B does have onboard Bluetooth and Wi-Fi radios and so additional hardware was not required for these computers.

A battery pack was required for each Raspberry Pi so that they could be deployed around a sports pitch, outdoors and away from any source of mains power.

The memory card required for each Raspberry Pi allowed for the installation of the Raspbian Gnu-Linux operating system. Raspbian is a stripped down Debian distro which is specifically aimed to be run on the Raspberry Pi computer. Raspbian also comes with Python 2.7 installed by default, reducing the amount of additional software required to be installed on the system.

To configure the system on first boot each Pi was connected via a wired internet connection. Before we are able to run any commands the Pi we must find out the Pi's IP address and connect to it over SSH.

# Search for the Raspberry Pi's IP address

# Attempt to connect to each IP with the following command until the right # address is found

ssh pi@192.168.0.14

# Once the correct IP is found we can enter the password and connect to the # pi

Please enter password for user pi@192.168.0.14: \*\*\*\*\*\*\*

Once we have found the right IP address and entered the correct password for the *pi* user we gain access to the Pi. From here we are presented with a terminal interface on which to run commands on the Pi.

With an SSH connection to the Pi established, the following commands were run on the commandline which enable the Wi-Fi and Bluetooth radios and also install the software libraries required to run *PNM-Mon*.

# Enable the wireless and bluetooth radios

sudo iwconfig wlan0 up

sudo hciconfig hci0 up

# Add wireless network connection details to wireless client conf file

sudo vim /etc/wpa\_supplicant.conf

# Add the following lines to the file and save

network = {

ssid="wireless-network-name"

psk="wireless-access-code"

}

# With the network address added attempt to connect

sudo dhclient

# Install bluetooth monitoring software

sudo apt-get install bluez -y

The next step is to download the *PNM-Mon* software from github onto the Pi.

# Connect to git and download platinum-analytics repo

git init

git remote add origin [git@github.com](mailto:git@github.com):peteehb/platinum-analytics.git

git fetch

git checkout develop

With all of the software in place, the final step is to set up the scheduled tasks, also known as Cron jobs, which start *PNM-Mon* on boot and check for an internet connection every 5 minutes. Further details on the implementation of the scripts which these scheduled tasks run can be found later in this chapter.

# Open the super users task scheduler

sudo crontab -e

# Add the following lines to the file.

@reboot python /opt/platinum/mon/StartProgram.py 1 False

With the scheduled tasks added the Pi is now ready for deployment and requires no further input from the user, simply powering on the unit is all that is required to start *PNM-Mon*.

#### PNM-Mon - Functionality

This section will provide detail into the functionality and feature set of the *PNM-Mon* software, showing the modules used and discussing what each of the classes is used for.

The program is run with the following command by a scheduled job on startup of the Pi.

*python StartProgram.py 1 False*

The first argument given is the ID of the monitor. This ID is an integer between 1 - 3 and tells the system which monitor this is. It is important to know which ID the monitor is as the program expects the monitors to be placed in

Overview of the program flow

Overview of each class and what it is used for

Discuss the different modes, intermittent vs stable connection mode

#### StartProgram.py

**class** **StartProgram(**object**):**

**def** \_\_init\_\_**(**self**,** ble\_mon\_id**,** run\_locally**):**

self**.**ble\_mon\_id **=** ble\_mon\_id

self**.**run\_locally **=** run\_locally

self**.**monitor **=** BluetoothMonitor**(**mon\_id**=**ble\_mon\_id**)**

internet **=** self**.**check\_connection\_to\_internet**()**

**if** internet **and** **not** run\_locally**:**

self**.**writer **=** DatabaseDataWriter**(**remote\_url**=**'http://130.255.72.102:8000/sensor-reading/'**)**

**else:**

self**.**writer **=** CsvDataWriter**(**rel\_path**=**'/logs/'**,** filename**=**'SensorReadings'**,**  file\_header**=[**'rssi'**,** 'mac\_address'**,** 'timestamp'**,** 'distance'**,** 'receiver'**])**

**def** run**(**self**):**

**if** self**.**writer**.**verify\_writer\_accessible**():**

**while** **True:**

reading **=** self**.**monitor**.**get\_next\_reading**()**

reading\_saved **=** self**.**writer**.**write**(**data**=**reading**)**

**if** **not** reading\_saved**:**

**import** ipdb**;**ipdb**.**set\_trace**()**

**break**

**print** reading

**def** check\_connection\_to\_internet**(**self**):**

con **=** Process**(**'sudo ifconfig wlan0 up'**)**

dhclient **=** Process**(**'sudo dhclient'**)**

conn\_established **=** **False**

internet\_established **=** **False**

p **=** Process**(**'sudo ip route ls'**)**

**if** p**.**get\_error**():**

conn\_established **=** **False**

**if** p**.**get\_output**():**

conn\_established **=** **True**

**if** conn\_established**:**

**try:**

response **=** urllib2**.**urlopen**(**'http://216.58.198.68'**)**

**if** response**:**

internet\_established **=** **True**

**except** urllib2**.**URLError**:**

**pass**

**return** internet\_established

**def** exit**(**rc**=**1**):**

os**.**\_exit**(**rc**)**

**def** cli**():**

**if** len**(**sys**.**argv**)** **<** 3 **or** len**(**sys**.**argv**)** **>** 3**:**

exit**()**

ble\_mon\_id **=** sys**.**argv**[**1**]**

run\_locally **=** sys**.**argv**[**2**]**

bluetooth\_monitor **=** StartProgram**(**ble\_mon\_id**=**ble\_mon\_id**,** run\_locally**=**run\_locally**)**

bluetooth\_monitor**.**run**()**

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

cli**()**

#### BluetoothMonitor.py

**class** **BluetoothMonitor(**object**):**

**def** \_\_init\_\_**(**self**,** mon\_id**):**

self**.**mon\_id **=** mon\_id

self**.**bluetooth\_scanner **=** Process**(**'sudo hcitool lescan --duplicates'**)**

self**.**bluetooth\_monitor **=** Process**(**'sudo btmon'**)**

self**.**ble\_reading **=** **{}**

**def** get\_next\_reading**(**self**):**

next\_reading **=** **False**

**while** **not** next\_reading**:**

line **=** self**.**bluetooth\_monitor**.**get\_output**()**

**if** line**.**startswith**(**'> HCI Event'**):**

self**.**ble\_reading **=** **{}**

**if** line**.**startswith**(**'Address:'**):**

self**.**ble\_reading**[**'mac\_address'**]** **=** self**.**extract\_mac\_address**(**line**)**

**if** line**.**startswith**(**'RSSI:'**):**

self**.**ble\_reading**[**'rssi'**]** **=** self**.**extract\_rssi\_reading**(**line**)**

self**.**compile\_reading**()**

next\_reading **=** **True**

**return** self**.**ble\_reading

**def** get\_distance\_from\_rssi**(**self**,** \_rssi**):**

rssi\_at\_one\_meter **=** **-**40

pathloss\_exponent **=** 4.2

distance **=** 10 **\*** **((**rssi\_at\_one\_meter **-** \_rssi**)** **/** **(**10 **\*** pathloss\_exponent**))**

**return** round**(**distance**,** 2**)**

**def** extract\_mac\_address**(**self**,** line**):**

mac\_address **=** re**.**search**(**r'([0-9A-Fa-f]{2}[:-]){5}([0-9A-Fa-f]{2})'**,** line**).**group**(**0**)**

**return** mac\_address

**def** extract\_rssi\_reading**(**self**,** line**):**

linestripped **=** line**.**split**(**'RSSI: '**)[**1**]**

rssi\_string **=** re**.**search**(**r'^.\*?dBm'**,** linestripped**).**group**(**0**)**

rssi **=** int**(**rssi\_string**.**rstrip**(**' dBm'**))**

**return** rssi

**def** compile\_reading**(**self**):**

rssi **=** self**.**ble\_reading**[**'rssi'**]**

self**.**ble\_reading**[**'timestamp'**]** **=** int**(**time**.**time**()** **\*** 100**)**

self**.**ble\_reading**[**'receiver'**]** **=** self**.**mon\_id

self**.**ble\_reading**[**'distance'**]** **=** self**.**get\_distance\_from\_rssi**(**rssi**)**

#### Process Manager

**class** **Process(**object**):**

**def** \_\_init\_\_**(**self**,** command**):**

self**.**command **=** command

self**.**process **=** self**.**run**()**

**def** run**(**self**):**

process **=** subprocess**.**Popen**(**self**.**command**,** shell**=True,** stdout**=**subprocess**.**PIPE**,** stdin**=**subprocess**.**PIPE**,** stderr**=**subprocess**.**PIPE**)**

**return** process

**def** get\_output**(**self**):**

line **=** self**.**process**.**stdout**.**readline**().**lstrip**()**

**return** line

**def** get\_error**(**self**):**

error **=** self**.**process**.**stderr**.**readline**().**lstrip**()**

**return** error

**def** write\_line**(**self**,** line**):**

self**.**process**.**stdin**.**writeline**(**line**)**

#### DataWriter Base Class

**class** **DataWriter:**

@abstractmethod

**def** \_\_init\_\_**(**self**,** **\*\***kwargs**):**

**pass**

@abstractmethod

**def** open**(**self**,** **\*\***kwargs**):**

**pass**

@abstractmethod

**def** verify\_writer\_accessible**(**self**,** **\*\***kwargs**):**

**pass**

@abstractmethod

**def** write**(**self**,** **\*\***kwargs**):**

**pass**

@abstractmethod

**def** close**(**self**,** **\*\***kwargs**):**

**pass**

#### CsvDataWriter

**class** **CsvDataWriter(**DataWriter**):**

**def** \_\_init\_\_**(**self**,** **\*\***kwargs**):**

DataWriter**.**\_\_init\_\_**(**self**,** **\*\***kwargs**)**

self**.**rel\_path **=** kwargs**.**get**(**'rel\_path'**)**

self**.**filename **=** kwargs**.**get**(**'filename'**)**

self**.**file\_header **=** kwargs**.**get**(**'file\_header'**)**

self**.**csv\_file **=** **None**

self**.**file\_writer **=** self**.**open**()**

self**.**accessible **=** self**.**verify\_writer\_accessible**()**

self**.**write\_count **=** 0

self**.**new\_file **=** **None**

**def** open**(**self**):**

self**.**csv\_file **=** **None**

file\_writer **=** **None**

timestamp **=** utils**.**get\_timestamp**()**

self**.**new\_file **=** self**.**filename **+** timestamp **+** '.csv'

**try:**

self**.**csv\_file **=** open**(**join\_cwd**(**self**.**rel\_path **+** self**.**new\_file**),** 'wb+'**)**

**except** Exception**,** e**:**

**import** ipdb**;**ipdb**.**set\_trace**()**

**if** self**.**csv\_file**:**

**try:**

file\_writer **=** csv**.**DictWriter**(**self**.**csv\_file**,** self**.**file\_header**)**

file\_writer**.**writeheader**()**

**except** Exception**,** e**:**

**raise** Exception

**return** file\_writer

**def** verify\_writer\_accessible**(**self**):**

writer\_accessible **=** **False**

**try:**

writer **=** self**.**open**()**

**except** Exception**,** e**:**

**raise** Exception

**if** writer**:**

writer\_accessible **=** **True**

**return** writer\_accessible

**def** write**(**self**,** **\*\***kwargs**):**

write\_success **=** **False**

**if** self**.**write\_count **>** 99**:**

self**.**close**()**

self**.**file\_writer **=** self**.**open**()**

self**.**write\_count **=** 0

data **=** kwargs**.**get**(**'data'**)**

**try:**

self**.**file\_writer**.**writerow**(**data**)**

write\_success **=** **True**

self**.**write\_count **+=** 1

**except** Exception**,** e**:**

**raise** ValueError**(**e**)**

**return** write\_success

**def** close**(**self**):**

self**.**csv\_file**.**close**()**

command **=** 'sudo mv ' **+** join\_cwd**(**'/logs/' **+** self**.**new\_file**)** **+** ' ' **+** join\_cwd**(**'/readings/' **+** self**.**new\_file**)**

Process**(**command**)**

#### DatabaseDataWriter

**class** **DatabaseDataWriter(**DataWriter**):**

**def** \_\_init\_\_**(**self**,** **\*\***kwargs**):**

DataWriter**.**\_\_init\_\_**(**self**,** **\*\***kwargs**)**

self**.**remote\_url **=** kwargs**.**get**(**'remote\_url'**)**

self**.**accessible **=** self**.**verify\_writer\_accessible**()**

**def** open**(**self**,** **\*\***kwargs**):**

**pass**

**def** verify\_writer\_accessible**(**self**,** **\*\***kwargs**):**

db\_accessible **=** **False**

**if** self**.**remote\_url**:**

**try:**

response **=** urllib2**.**urlopen**(**self**.**remote\_url**)**

**if** response**:**

db\_accessible **=** **True**

**except** urllib2**.**URLError**:**

**pass**

**return** db\_accessible

**def** write**(**self**,** **\*\***kwargs**):**

write\_success **=** **False**

data **=** kwargs**.**get**(**'data'**)**

**if** self**.**accessible**:**

**try:**

r **=** requests**.**post**(**self**.**remote\_url**,** data**)**

**if** r**.**status\_code **==** 201**:**

write\_success **=** **True**

**except** Exception**,** e**:**

**raise** Exception

**return** write\_success

**def** close**(**self**):**

**pass**

#### PNM-DB - Architecture Setup and Deployment

This section will outline the steps required to setup the Linux server with the software required to deploy *PNM-DB*. *PNM-DB* runs on a Debian Linux Server, subsequently all of the following commands are runnable on any Debian based derivate distros, such as Ubuntu or Mint.

##### Postgres

The first piece of software required for *PNM-DB* is Postgres. Postgres is an open source object-relational database system. (1) It is the database used to persistently retain and store all data collected by the *PNM-Mon* monitors. The following commands install Postgres and also create the database *'pnmdb'* on the system. We also add the user *'platinum'* for access to the database from external applications.

# Update software packages to latest versions and install postgres

sudo apt-get update

sudo apt-get install postgresql -y

# Once postgres has been installed change to the 'postgres' system user and # create the pnmdb database and platinum user

sudo su postgres

psql

create database pnmdb;

\q

##### Django-Rest-Framework

With postgres successfully installed we can now deploy the *PNM-DB Web API*. Django-Rest-Framework was the tool chosen with which to build the API. Django-Rest-Framework is ...

One of the main reasons for using DRF is that Django provides a way of creating and managing postgres database tables through Python with the use of an object-relational-mapper. This means that we can define our postgres tables and relationships as Python objects or 'Models'.

We can then expose these Python objects over HTTP by Serializing the Models to JSON objects and making them accessible through URLs.

With these features that django-rest-framework provides, we can now server our database data over HTTP. Create-Read-Update-Delete requests made to defined URLS can now alter data in the database.

To setup the *PNM-DB Web API* the following steps were run on the database server.

# Connect to git and download platinum-analytics repo

git init

git clone git@github.com:peteehb

# Create a virtualenv, Install requirements and run pnm-db

mkvirtualenv pnm-db

pip install -r requirements.txt

cd pnmdb/

python manage.py syncdb

python manage.py makemigrations

python manage.py migrate

python manage.py runserver 0.0.0.0:8000

##### Deployment

The above command will only run *pnm-db* for as long as the user is on the server. This is not suitable for a production environment and so a more permanent method of deployment had to be found.

To overcome this issue an alternative method of deployment was required. The method decided upon was to run the django-rest-framework application persistently using UWSGI. This is achieved by installing a plugin for the preinstalled webserver Apache called *mod\_wsgi*. This plugin runs the django application in a process which it itself manages. This requires no input from the user once *mod\_wsgi* is installed and *PNM-DB* django application is properly configured.

# Install apache dev tools

sudo apt-get install apache-threaded-dev

# Get and install the mod\_wsgi plugin

wget https://github.com/GrahamDumpleton/mod\_wsgi/archive/4.4.21.tar.gz

tar xvfz mod\_wsgi\_4.4.21.tar.gz

./configure

make

sudo make install

# Load mod\_wsgi into Apache and enable

cd /etc/apache2/modes-available

vim mod\_wsgi.so

# Append the following line

LoadModule wsgi\_module /usr/lib/apache2/modules/mod\_wsgi.so

# Enable plugin

cd /etc/apache2/mods-enabled

sudo ln -s ../mods-available/mod\_wsgi.load

# Restart apache

sudo /etc/init.d/apache2 restart

One final step in deploying *pnm-db* is to add an entry to Apache's *sites-available* file. This is a file which tells the apache which websites and webapplications are running on the server. In this file we define the location of the wsgi application for *pnm-db*, along with its root directory and which port number it is accessed on. Once the file has been edited a final restart to the apache process will make *pnm-db* available and finish the deployment process.

# Tell Apache to listen to all connections made on port 8000.

Listen 8000

<VirtualHost \*:8000>

# Tell Apache the location of the wsgi application

WSGIScriptAlias / /opt/platinum/db/pnmdb/wsgi.py

# Create a process and run *pnm-db* in the defined virtualenv

WSGIDaemonProcess pnmdb python-path=/opt/platinum/db/pnmdb:/home/elpok/.virtualenvs/pnm-db/lib/python2.7/site-packages

WSGIProcessGroup pnmdb

Alias /static /opt/platinum/db/static

# Set permissions to allow access to *pnm-db* directory

<Directory /opt/platinum/db>

Options -Indexes

Order deny,allow

Allow from all

</Directory>

# Define log files for *pnm-db*

ErrorLog ${APACHE\_LOG\_DIR}/pnmdb\_error.log

CustomLog ${APACHE\_LOG\_DIR}/pnmdb\_access.log combined

</VirtualHost>

#### PNM-DB -Functionality

In this section I outline the Python Models which represent the database tables and the functionality which allows access to these objects through HTTP.

#### PNM-WEB Architecture and Deployment

### Conclusion

## 5. System Validation

### Introduction

In this section I detail the testing and validation done to the system to verify that the features developed provide for all of the use cases stated above in Chapter 3.

### Conclusion