

Bachelor's thesis

Degree programme in Information Technology

240S08

2016

Instructor – Patric Granholm

Laxmi Thebe

COMMUNITY PARENTING PLATFORM

– Development and Deployment Using the Django
Framework



Author – Laxmi Thebe

COMMUNITY PARENTING PLATFORM

- Development and Deployment Using the Django Framework

With the emergence of Information Technology, different tools are being developed and deployed even with the focus on promoting the wellbeing of people in this particular field. In parallel with this phenomenon, globalization has taken a new height in this era of information technology – scattering millions of Nepali people around the globe in search of opportunity.

In the light of the aforementioned context, this thesis project aims to bring thus scattered people around the globe to their own virtual local community with the noble cause of supporting a child in their upbringing in their local community by developing and deploying a web application built on Django Framework including features such as content generation, need declaration, and sponsorship offers.

This thesis covers the processes of development and deployment of the project, describes tools and technologies used, and orients towards future development facilitated with inclusion of scalability issues and security issues. Furthermore, it demonstrates the use of open source technologies for improving the wellbeing of people.

KEYWORDS:

Django Framework, Web Development, Python

FOREWORD

For bringing this little contribution from the vast ocean of knowledge, I immensely owe:

To my parents, for letting me see that ocean;

To all people before me, for creating that ocean;

To my teachers, for being the light on the path;

To my girls, for the encouragement and patience;

And to the Master, for orchestrating this drama.

.... I am thankful to my Instructor Patric Granholm for guiding me through this process, to my Mathematics Teacher Hazem Al-Bermanei and to my English Teacher Poppy Skarli that I have some soul satisfying memories of my school days.

CONTENTS

LIST OF ABBREVIATIONS (OR) SYMBOLS	6
1 INTRODUCTION	6
1.1 Community Parenting and Nepali context in brief	6
1.2 Project Aim	7
1.3 Author's Contribution	8
1.4 Thesis Overview	8
2 LITERATURE REVIEW ON TECHNOLOGIES USED	10
2.1 Django Framework	10
2.1.1 Django Philosophy	10
2.1.2 Why Django?	11
2.1.3 Django Request/Response Process	12
2.2 Development Tools	13
2.3 Python Packages	14
2.4 Deployment Tools	15
3 SETTING UP THE DEVELOPMENT ENVIRONMENT	17
3.1 Setting Up the Virtual Environment	17
3.2 Starting the Project	17
4 APPLICATION DEVELOPMENT PROCESS	20
4.1 Django Workflow	20
4.2 Apps and the related models in the project	20
4.3 Views in the project	25
4.4 Forms in the Project	28
4.5 URLConf in Django	30
5 APPLICATION DEPLOYMENT PROCESS	33
5.1 mod_wsgi and deployment of Django in Apache server	33
5.2 Deploying Django project with MySQL	35
5.3 Security Measures in Django Application	36
5.4 Database Security Issues	37

5.5 Scalability Issues in Application	38
6 CONCLUSION AND RECOMMENDATION	40
6.1 Recommendation	40
6.2 Personal Development in the context of thesis project	41
REFERENCES	43

FIGURES

Figure 1. Django Request/Response Cycle (Holovaty and Kaplan-Moss 2009).	13
Figure 2. Deployment of Django in Large Setup (Holovaty and Kaplan-Moss 2009).	39

PICTURES

Picture 1. The list of Python Packages.	15
Picture 2. Django Version 1.9.8.	18
Picture 3. 'manage.py' demonstrating administrative ability.	18
Picture 4. 'Post' model from 'posts' app.	21
Picture 5. 'Member' model from 'people' app.	22
Picture 6. Models from 'community' app.	23
Picture 7. Models from 'support' app.	24
Picture 8. Function-based views for 'contact' page.	25
Picture 9. Example of View from 'community' app.	26
Picture 10. Django Mixin.	27
Picture 11. Template Example.	27
Picture 12. StudentCreateView rendering the view in the browser.	28
Picture 13. Example of Form in the 'community' app.	29
Picture 14. Student model in 'community' app.	29
Picture 15. URLconf in 'people' app.	30
Picture 16. ROOT_URLCONF file.	31
Picture 17. Contact page on browser.	32
Picture 18. mod_wsgi installation confirmation.	33
Picture 19. Apache VirtualHost configuration file.	34
Picture 20. Database Settings in settings.py file.	35
Picture 21. Database access configuration file.	35
Picture 22. MySQL bind to localhost.	37
Picture 23. Firewall in Action.	38

LIST OF ABBREVIATIONS (OR) SYMBOLS

Abbreviation	Explanation of abbreviation (Source)
API	Application Programming Interface
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	HTTP Secure also known as HTTP over SSL/TLS
IDE	Integrated Development Environment
MTV	Model Template View
MVC	Model View Controller
ORM	Object Relational Mapper
REST	Representational State Transfer
SQL	Structured Query Language
SSH	Secure Shell Protocol
URL	Uniform Resource Locator
WSGI	Web Server Gateway Interface

1 INTRODUCTION

With the emergence of Information and Technology, the way of our living and interaction has changed in a way which has brought multitudes of longitudes and latitudes to intersect in a single point transforming the whole world into a global village. With this phenomenon, the current wave of globalization is largely fueled and more than 2 million Nepali people are working abroad in different parts of the world.

This thesis project aims to develop a web application which brings Nepali people from around the globalized world in that intersection called 'Community Parenting' - where the similar interest of different people from multitudes of intersection of longitudes and latitudes sponsors children back home in Nepal in different aspects of their necessities.

This thesis discusses the development and deployment process of the thesis project to its prototype version. Background information related to the project and future development is included as well to create a smooth transition of personal development during the thesis development process.

1.1 Community Parenting and Nepali context in brief

The community parenting term is used by different stakeholders to describe different approaches of involvement in the life path of children. For example, supporting a child through promotion of partnership with parents is one such approach whereas it could be also done by direct involvement with the activities of a child, too. Broadly, it can be defined as an action conducted to improve the well-being of children for their proper upbringing by carrying different community based projects and activities. In Nepal, children grow up among a number of family members and are never alone in their life. Our upbringing, mode of economic production, cultural values etc. strongly promote dependency and some degree of collectivism even in community level. We are dependent not only with someone within our relative circle, but also help each other in case of need within community. Our support system is largely based upon the 'give and take' principle – a reflection as well as foundation of dependent relationship. Consequently, it is easier to witness other community members to get involved in the support of child upbringing where the old adage "It takes a village to raise a child" is evidential.

According to documented statistics (Kaphle 2014), Nepal is highly dependent on remittance and more than 2 million of Nepali people are working abroad in different sectors which cover a wide spectrum of opportunity from around the globe. Most of them are deeply motivated to impart positive impact back home as demonstrated in various social media and news channels. Community parenting can be hence defined as an approach by people to positively contribute to their community by providing supports to the children in their community.

1.2 Project Aim

This thesis primarily aims to develop a prototype version of platform as a website using Django Framework which can be used as a tool to share knowledge to support school children by creating posts related to the school or village a user of the site belongs to or to demonstrate sponsorship to support the individual needs of children in Nepal. Although the concept is developed in the socio-economic context of the Nepal, it is equally viable as a tool in other developing countries. To achieve the project aim, the following objectives are set to be achieved:

- Implementation of a login system for users who are about to sponsor a children.
- Financial transaction in case of financial needs of the children. The financial needs can be sponsored even by anonymous users where as other needs requiring long term commitment would be only possible for verified users.
- Implementation of children in need profiles – they can be made visible for public or visible only for the community. The profile of the child includes basic explanation of the needs, and child description of the situation.
- A child can have multiple needs whose targets can be achieved by multiple users.
- A post can be created publicly or within community regarding the wellbeing of the community or of the child in need. For example, a user can ask specific regarding a particular details so that other members can comment on it.

To conclude the basic concepts of this project based on the breakdown of project aim, the project aims to involve local people to contribute to the wellbeing of children in their community back in Nepal by generating relevant contents or offering sponsorship to the relevant needs of children.

1.3 Author's Contribution

This thesis emphasizes developing a web platform using Django Framework to support a child in different needs – moreover, the platform can be also used as a tool to organize activities around a local group. The project, albeit in its preliminary stage, uniquely combines the different principles of developmental work and strengthens the position of Information Technology as a tool for humanity. Although the concept is developed in the socio-economic context of the Nepal, it is equally viable as a tool in other developing countries as well.

1.4 Thesis Overview

This section of the thesis primarily aims to elaborate the Table of Contents so that readers can be familiar with the main content of the thesis briefly. The structure and content of the thesis is described briefly as:

Chapter 1 (Introduction) introduces the concept, includes brief background analysis for the project, motivation, and most importantly provides overview of project aim.

Chapter 2 (Literature Review on Technologies Used) describes the tools and framework opted to develop the project. In its attempt to illuminate on technologies, it primarily describes the main framework used for the development of this project, i.e., Django Framework. As a bonus, it shed lights on other packages/libraries used in combination with this framework. Additionally, it discusses development and deployment tools.

Chapter 3 (Setting Up the Development Environment) discusses on procedures for setting up development environment which takes place before starting the coding of the project.

Chapter 4 (Application Development Process) describes basic concept of web application development in Django, and explains the structure of the project from development perspective. It also sheds light on some challenges faced by the author during the development process.

Chapter 5 (Application Deployment Process): discusses the deployment of the project. Although it is in the prototype version of development, the deployment of the project is carried for the purpose of demonstration. As the author has personally desired to develop a web application that scales up to serve requests in large volume, including deployment in this thesis makes sense. Security issues and scalability issues are discussed for that reason – in addition to talking about general deployment.

Chapter 6 (Conclusion and Recommendation): Discusses the development of personal perspective, concludes the thesis paper, and recommends further development.

As explained above, the thesis is structured to impart a gradual flow from start to finish – starting with reflecting the personal and academic premises in the beginning of the thesis project, and finish including the accomplishment of realization of prototype version and encouragement for future development.

2 LITERATURE REVIEW ON TECHNOLOGIES USED

From the development to the deployment of the project, various tools and technologies were used. The specific tools used for development and deployment are discussed in this chapter briefly whereas discussion on the web framework used, i.e., Django Framework, is included in more details. Generic tools - for example, SSH client for accessing remote server - are not elaborated as they go beyond the specific focus of the thesis.

2.1 Django Framework

Django is a Python web framework “for perfectionists with deadlines” which is based on the MVC (Model, View and Controller) design pattern. Django handles common web development task, is exceedingly fast and scalable, secure, and incredibly versatile (Django Software Foundation 2015c) - making it a tool for rapid development of web application with clean, concise, and maintainable codes.

In MVC design pattern, the models in Django represent the underlying data models in the backend whereas the templates represent the View. Interestingly and not to be confused, the view function in Django works as a controller (which can be contested if Django is said to opt MTV design pattern). Django adopts the philosophy of loose coupling – where different parts of a system are loosely dependent on each other hence a single part of the system has a single role which can be easily replaced with other similar functional components making components orthogonal. The MVC design pattern also enables loosely coupled models, views, and controllers components. For example, templates are used to dynamically generate HTML in Django framework – and other template engines can be used instead of Django’s own template system.

2.1.1 Django Philosophy

As described already, loose coupling is one of the fundamental philosophy of Django Framework which enables different stacks of the framework to work in cohesion but indepenednt of one another whenever and wherever possible. For example, the URLs

in Django are independent of the Python code that generates view and can be changed without changing a single line in view codes.

Less code is another Django design philosophy promoting the use of as less code as possible. Concise, clean and maintainable codes result from the less code philosophy. Quick development is yet another philosophy which can be summarized with “for perfectionists with deadlines” tag line. Code repetition is not promoted in Django as summarized in DRY (Don’t repeat yourself) philosophy. As an example, view inheritance is the concept in Django where a template which can be repeated in different pages of the site can be included as a template rather than hard coding in every pages. Django emphasizes normalization in contrast to redundancy in development practice. “Explicit is better than implicit” is a core Python principle also adopted by Django Framework which goes along with other principles nicely. (Django Software Foundation 2015b.)

2.1.2 Why Django?

A web framework handles common web development tasks as stated earlier. Using Django lets the developers to emphasize the specific aspects of the application they are developing rather than implementing the common aspects of web development frequently. As being a Python Framework, Django also follows Python’s “batteries included” philosophy hence including features which might not be implemented in most other frameworks.

Additionally, Django has a great community and good documentation – which makes it easier for beginners to make their hands dirty by enticing in real project development tasks. Moreover, it is a widely used open source framework with several third party packages mostly kept updated. The information about different Django communities can be found in official Django project site.

As Django adopts DRY philosophy, codes following Django philosophies are concise and readable. Being a Python framework, it can be deployed in any platform which supports Python. Such a portability is further enhanced by the use of ORM, i.e., object relational mapper, so that Django can be deployed with different database management systems. The popularity of Django framework, availability of Django

developers, and being a widely used established framework pushes cloud providers to offer support and services for easier deployment of the Django application.

Django also has a built-in admin panel which lets manage the users of the site and other database objects. It is possible to be familiar with Data Models from Django shell, and to some extent from the admin panel as it provides an application wise model-centric interface. In that way, it allows developers and non-technical staff to work together to develop a data-centric applications (Neuman 2015).

Most importantly, Django is tested and scalable. It is used by some heavy traffic sites like Eventbrite, Disqus, Instagram, Prezi, Pinterest, Washington Post and other notable sites. As Django is loosely coupled, different stacks can be unplugged and customized to fit the specific needs. The development and deployment of this project using Django framework within the limitation of time constraint is self-evident of Django's philosophy 'for perfectionists with deadlines' – where fairly challenging projects could be implemented cleanly, concisely, and efficiently.

2.1.3 Django Request/Response Process

An HTTP request from the browser is used to construct the HttpRequest object by the handler which is passed to the later components. Additionally, the server-specific handler also handles the response processing. Django has a middleware framework which intercepts the request/response processing and thus having the ability to alter Django's input or output. As an example, AuthenticationMiddleware intercepts the requests and links them with specific users using sessions (Django Software Foundation 2015d). As shown in the following diagram, the processing of View is entirely bypassed if any of the middleware returns HttpResponse. The view function is the last one to return the HttpResponse in this processing order. Exception middleware takes control in case of exception in view – which might either return HttpResponse or the exception is raised again. Ultimately, if the exception is not handled anywhere in the processing order, Django provides default views like HTTP 404 and HTTP 500 response.

The view function being one of the important concept in Django web development concept will be discussed further later in Chapter 4 where returning the HttpResponse would be evidential in practice.

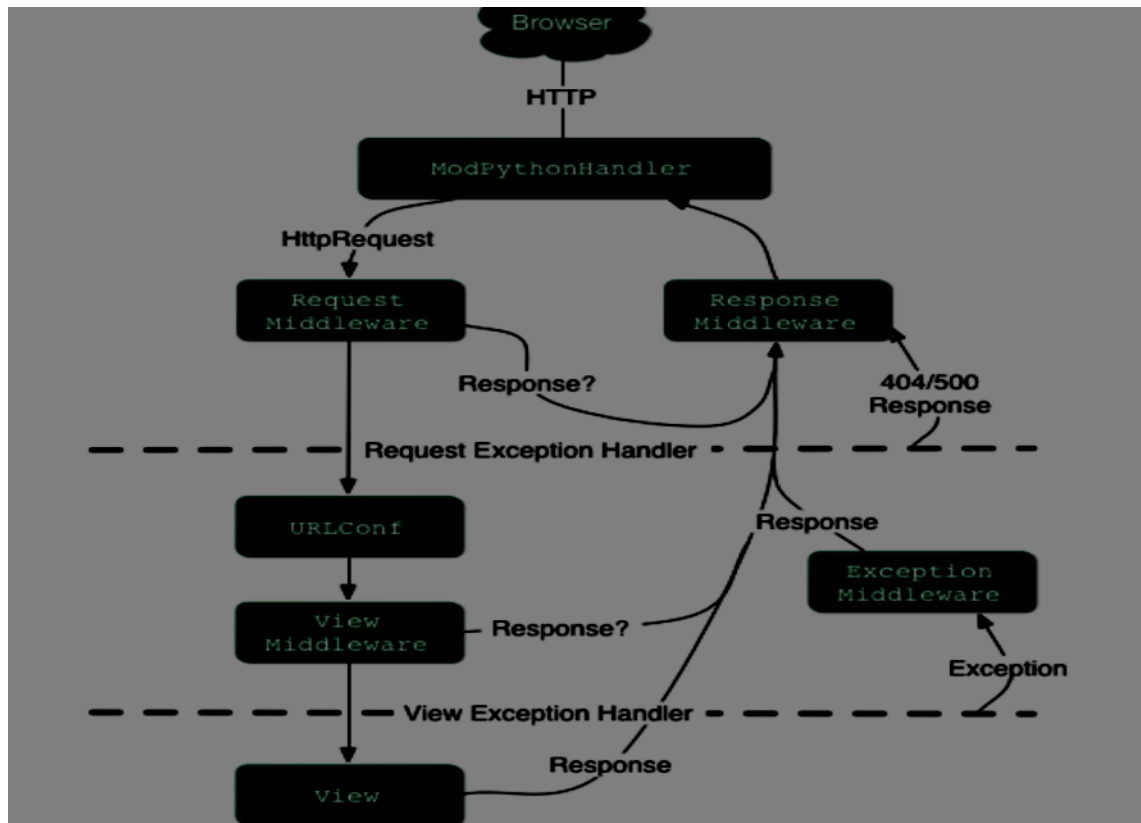


Figure 1. Django Request/Response Cycle (Holovaty and Kaplan-Moss 2009).

The last stage of request/response processing takes when the Response Middleware processes the HttpResponse to return to the browser. Additionally, resources related to the specific requests are handled by Response Middleware. (Holovaty and Kaplan-Moss 2009).

2.2 Development Tools

During the development process, two tools were used in particular – one of the tool being used for setting up development environment while the other tool for writing the code. The tools are described as follows:

Virtualenv: Virtualenv is one of the tool used for development – which creates isolated Python environments hence addressing the issue of dependencies, versions, and permissions (Bicking 2014). The project was developed in an isolated environment created using virtualenv which enabled the author to follow different tutorials developed using different versions of the same Python packages.

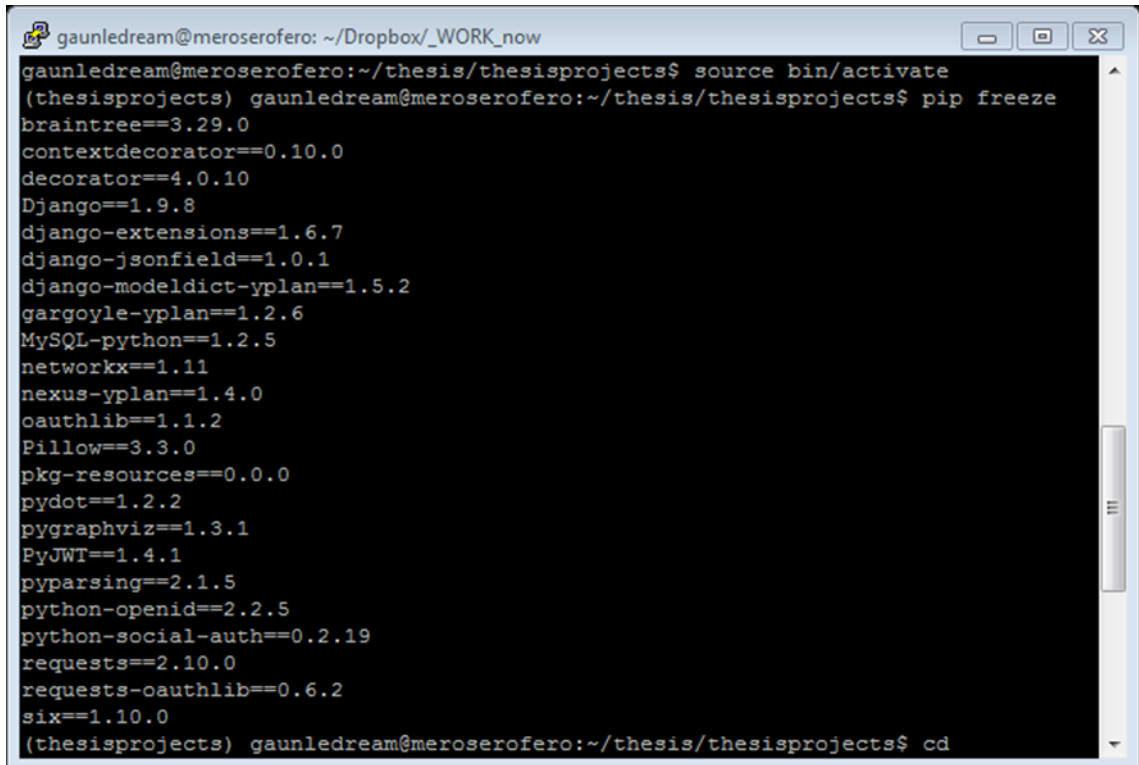
Vim: For the purpose of writing code, the Vim text editor was used in this project. Vim is a highly configurable advanced text editor which enables efficient editing of the text – a tool which can be used using SSH communication to the server shell to code the project. Often known as "programmer's editor", Vim is highly useful for programmer as it is lightweight and fast providing an easy way to edit text frequently (VIM 2016). Additionally, Vim is easily available in most of the Linux systems.

PyCharm IDE was one such tools, but the author chose to use Vim as the server was accessible to anywhere using SSH communication and hence it was easier to use Vim to write code from anywhere even during very short free time. As said previously, being lightweight, there was really no need to wait like a minute as in the cases of most IDEs as a single command `vim` in the Linux shell is enough to fire the Vim editor immediately. Moreover, there are so many different plugins for the Vim text editor which facilitates coding.

The Ubiquity of Vim, different plugins for Vim to assist coding, its efficiency, and its significance as text editor over SSH terminal session necessitated its utilization during the development stage.

2.3 Python Packages

Different Python packages were used during the development and deployment of the project and they serve different purposes. Some packages add functionality to the site whereas some packages are used as a utility tool for different purposes in different phases of the project development. Some packages add special features which can be used by other apps or apps written for the project. The following picture displays the list of packages installed in the virtualenv and utilized for the project during development, deployment and documentation:



```

gaunledream@meroserofero: ~/Dropbox/_WORK_now
gaunledream@meroserofero:~/thesis/thesisprojects$ source bin/activate
(thesisprojects) gaunledream@meroserofero:~/thesis/thesisprojects$ pip freeze
braintree==3.29.0
contextdecorator==0.10.0
decorator==4.0.10
Django==1.9.8
django-extensions==1.6.7
django-jsonfield==1.0.1
django-modeldict-yplan==1.5.2
gargoyles-yplan==1.2.6
MySQL-python==1.2.5
networkx==1.11
nexus-yplan==1.4.0
oauthlib==1.1.2
Pillow==3.3.0
pkg-resources==0.0.0
pydot==1.2.2
pygraphviz==1.3.1
PyJWT==1.4.1
pyparsing==2.1.5
python-openid==2.2.5
python-social-auth==0.2.19
requests==2.10.0
requests-oauthlib==0.6.2
six==1.10.0
(thesisprojects) gaunledream@meroserofero:~/thesis/thesisprojects$ cd

```

Picture 1. The list of Python Packages.

Some of the packages are worth to mention, whereas some utility packages can be removed as well. For example, a braintree is used to implement online payment – which is used for sponsoring a student for his or her financial need in the context of this project. MySQL-Python is used to implement the Django with MySQL database system – which is a database connector for Python. Pillow is installed so that an ImageField can be added in Django Models. python-social-auth is used to implement Facebook login to the system. All the required Python packages were installed using the pip python package management tool.

2.4 Deployment Tools

This thesis project was developed using Django Web Framework and other supporting python packages. This web application developed using Django Framework was deployed in an Apache Server using mod_wsgi and the database back end was implemented using MySQL database .

Apache Server: The Apache HTTP server is the most used server in Internet which supports different features – sometimes extending the core functionalities through compiled modules (one such module being `mod_wsgi` explained below). It is an open source software by Apache Software Foundation. In addition to being widely adopted, reliable, and secure, the virtual hosting feature in Apache was an additional reason for being chosen for this thesis project as a single web server is being used to serve multiple sites which are developed using different technologies.

mod_wsgi: It is an Apache module which can be used to host Python web application. The python web application must support the Python WSGI specification so that the application can be implemented with an Apache server using the `mod_wsgi` package (Dumpleton 2016a).

MySQL: Database systems play a central role in computing – particularly when large amounts of data are to be handled. Django supports different database back ends but not all features on all possible back ends. UTF-8 encoding is assumed to be used by the database back ends in Django. MySQL is a relational database management system – which means that data are stored in separate tables related to each other and defined by relationship rules enforced by database management system. It is known for its speed, reliability and scalability. (MySQL Documentation Team 2016) MySQL 5.5 or higher version of MySQL is supported. A Django's feature called `inspectdb` utilizes `information_schema` database in MySQL – which is meta-data about database schema and hence can generate the Django models using the database. Enforcement of transaction and referential integrity is delegated to the database system by Django. So, the MyISAM MySQL engine is not able to apply these two features hence the default engine InnoDB is not changed in this project.

To explain the complete thesis process briefly in term of technologies use, Django Framework was used as a web development framework with Vim and Virtualenv as a developing environment so as to deploy later in the Apache server using `mod_wsgi` and the MySQL database backend.

3 SETTING UP THE DEVELOPMENT ENVIRONMENT

This chapter discusses the setting up of the platform for the purpose of development just before actually starting the coding part. As famously known Django for its niche as “for perfectionists with deadlines” for the development part, it is relevantly easier to set up the development environment as well. The virtual environment was set up for the development in Ubuntu 16.04 known as Xenial Xerus as both development and deployment took place in the same platform. Additionally, the Django project was also created before starting to write code as part of the setting up of the platform. The following sections describe the process to clarify the process further.

3.1 Setting Up the Virtual Environment

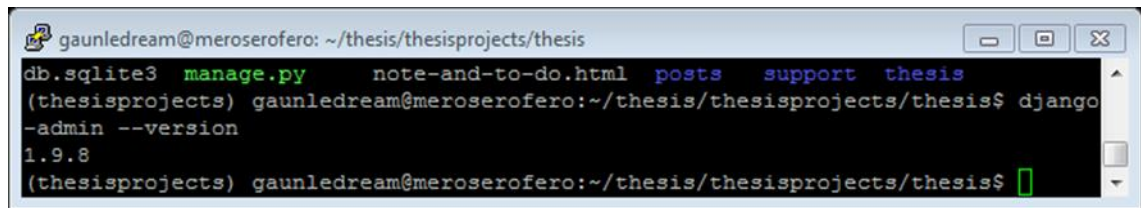
The web platform is developed using Python 2.7.12 and Django 1.9, hence, pip was installed for version 2 of Python. Once pip was installed, virtualenv was installed using the pip command tool. After installing the virtualenv tool, the new virtual environment called ‘thesisprojects’ was created in ‘thesis’ directory by issuing the virtualenv command.

As stated previously, virtualenv creates isolated and independent Python environment for development which should be activated individually – in the case of this project, by issuing the command to run the virtualenv activation script in the Linux terminal. The activation results in the change of the shell prompt to reflect the activation of the corresponding virtual environment. The virtual environment can be deactivated anytime using the ‘deactivate’ command, i.e., by issuing the command to run the virtualenv deactivation script. After the activation of the virtual environment, all the packages needed for the development can be installed using the pip tool included by default with the virtualenv (unless virtualenv is installed with --no-pip option) tool.

3.2 Starting the Project

After activating the virtual environment, Django Framework was installed using the pip tool. As the version was not specified, the latest django version was used for the

development of the project. Picture 2 verifies the Django version used for this thesis project.



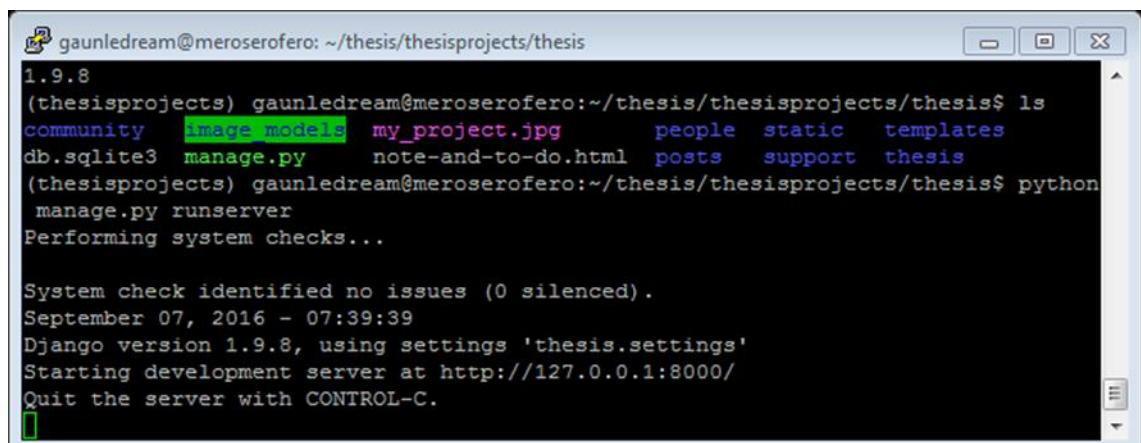
```

gaunledream@meroserofero: ~/thesis/thesisprojects/thesis
db.sqlite3  manage.py  note-and-to-do.html  posts  support  thesis
(thesisprojects) gaunledream@meroserofero:~/thesis/thesisprojects/thesis$ django
-admin --version
1.9.8
(thesisprojects) gaunledream@meroserofero:~/thesis/thesisprojects/thesis$

```

Picture 2. Django Version 1.9.8.

After installing Django, the Django project called 'thesis' was created within the activated virtual environment using the `django-admin` command which is a utility tool for administrative tasks. This process automatically creates a very important utility tool 'manage.py' to handle administrative tasks in the project directory. It is important to note that the 'settings.py' file is also created during this process – which contains configuration for the Django project. In contrast to `django-admin`, 'manage.py' is more convenient for use as it sets the 'settings.py' as the default configuration source file for the project by setting the "DJANGO_SETTINGS_MODULE" environment variable to 'settings.py' with the `os.environ.setdefault("DJANGO_SETTINGS_MODULE", "thesis.settings")` command. Moreover, it sets the package related to the project on `sys.path`. As this project involves working only with a single Django project and thus a single 'settings.py' file, 'manage.py' is better choice to 'django-admin' for administrative tasks. The following picture demonstrates the administrative capability of the 'manage.py' script by issuing command to run development server:



```

gaunledream@meroserofero: ~/thesis/thesisprojects/thesis
1.9.8
(thesisprojects) gaunledream@meroserofero:~/thesis/thesisprojects/thesis$ ls
community  image models  my_project.jpg  people  static  templates
db.sqlite3  manage.py  note-and-to-do.html  posts  support  thesis
(thesisprojects) gaunledream@meroserofero:~/thesis/thesisprojects/thesis$ python
manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
September 07, 2016 - 07:39:39
Django version 1.9.8, using settings 'thesis.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

```

Picture 3. 'manage.py' demonstrating administrative ability.

After creating the Django project, different Django applications can be created within a project which are integrated to work coherently as a component of the project. It is possible that a whole Django site can be developed with a single application within the project, but developing a project with different applications representing different components is good design practice. Moreover, as described in Chapter 2, Django promotes such practices. In practice, the 'manage.py' utility is used to create the application within the project.

In addition to developing 'posts' application within the Django project, a third-party Django app could be interchangeably used to achieve similar functional requirements as offered by 'posts' application. For this thesis project, community, posts, support and people apps were created and developed. The detail of the development process is described in the next Chapter.

4 APPLICATION DEVELOPMENT PROCESS

In Django, the application development process is robust and simple as well – particularly when it uses the MVC design paradigm. Being familiar with that paradigm – for example, by using Play Framework (which is also MVC framework) – helps to easily understand the application development process in Django. This chapter discusses Django Workflow in general, and describes the application development process in general by providing examples from the project itself.

4.1 Django Workflow

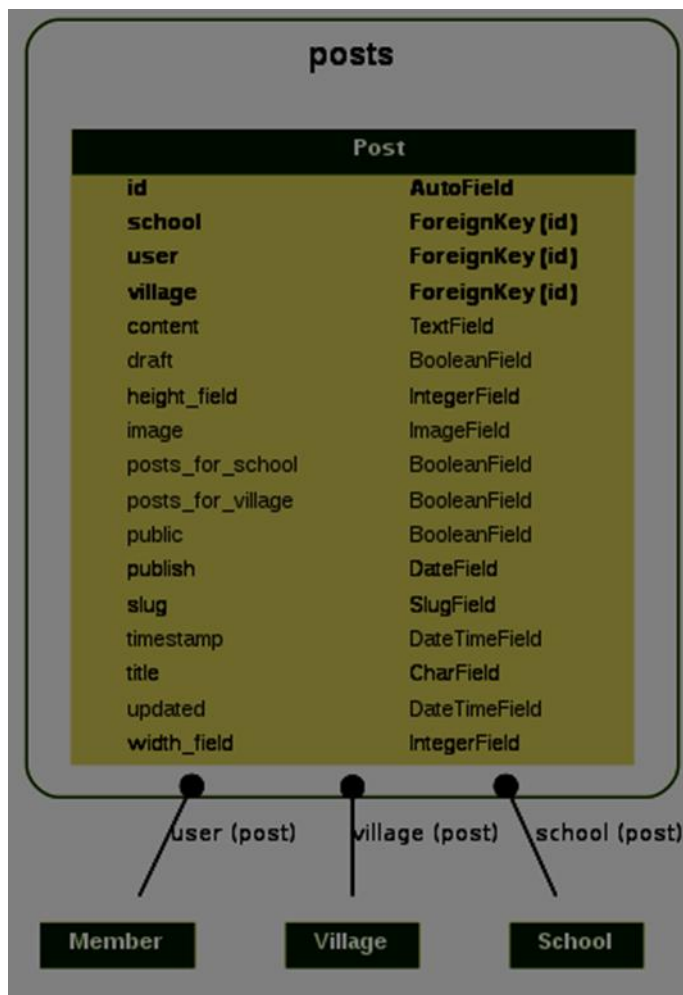
In Django application, URL is mapped to the view using URLconf which is defined in `urls.py` file. During the creation of the project, a `urls.py` (set to be `ROOT_URLCONF` in `settings.py`) is added where URLconf for the apps can be imported using `include()` function from `django.conf.urls`. When the requests are received by Django as URL, the appropriate view is called based on the URLconf to generate response to be rendered in template. In its simplest form, a Django work-flow consists of defining model, writing view, updating the URLconf for matching URL to view, and writing the templates. Database configuration can be setup already or left with the default configuration which is for SQLite as it is easier to deploy in later phase after the development as well.

For the project, the `urls.py` file was already created when the project was created with its own URL space. The apps can have their own URLconf file which can be imported to the `ROOT_URLCONF` i.e. URLconf of the project by using `include()` function. As apps play an important role in Django development, apps are defined in the next section; and models being the main component of the data centric apps, the models in each app will be illustrated with pictures. Views are important but it is inconvenient to describe all of them, so this paper discussed only briefly whereas URLconf and `urlpatterns` will be explained with the help of an example from the project.

4.2 Apps and the related models in the project

The thesis project contains four different apps called ‘posts’, ‘support’, ‘community’ and ‘people’. The app ‘people’ contains custom user information related to this project even

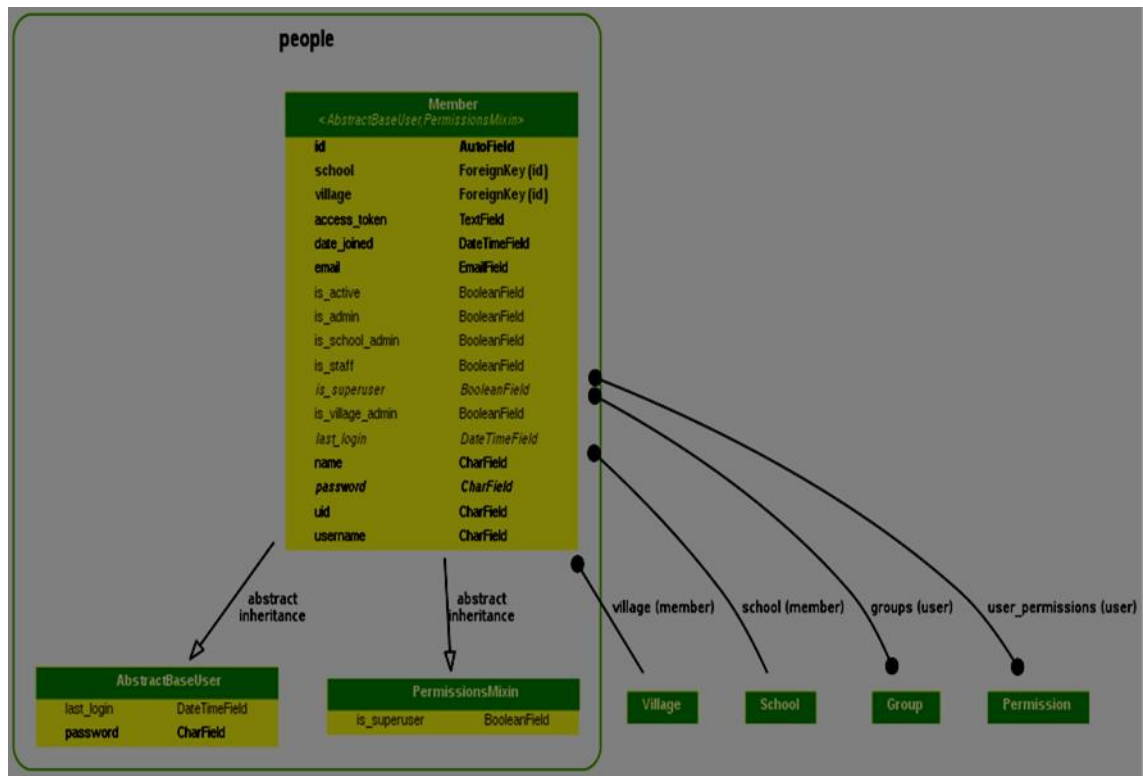
though users are registered using the python-social-auth package when users log in the site using Facebook credentials. By extending the AbstractBaseUser class from Django, custom fields are added for the users which are labeled as 'Member' in Django model in 'people' app. Thus created models also relate to the UserSocialAuth model from the python-social-auth package. The posts app is all about creating posts by 'Member' in 'people' app for 'community' or for public in general. The following picture demonstrates the graphical form of the model in posts app. Django-extensions was used together with pygraphviz to generate all the model graphs from four different apps. With Member, Village, and School models, the picture shows one to many relationship of the Post model.



Picture 4. 'Post' model from 'posts' app.

Similarly, the Member model demonstrated in the following picture reflects the inheritance of AbstractBaseUser and PermissionMixin class and its relationship with

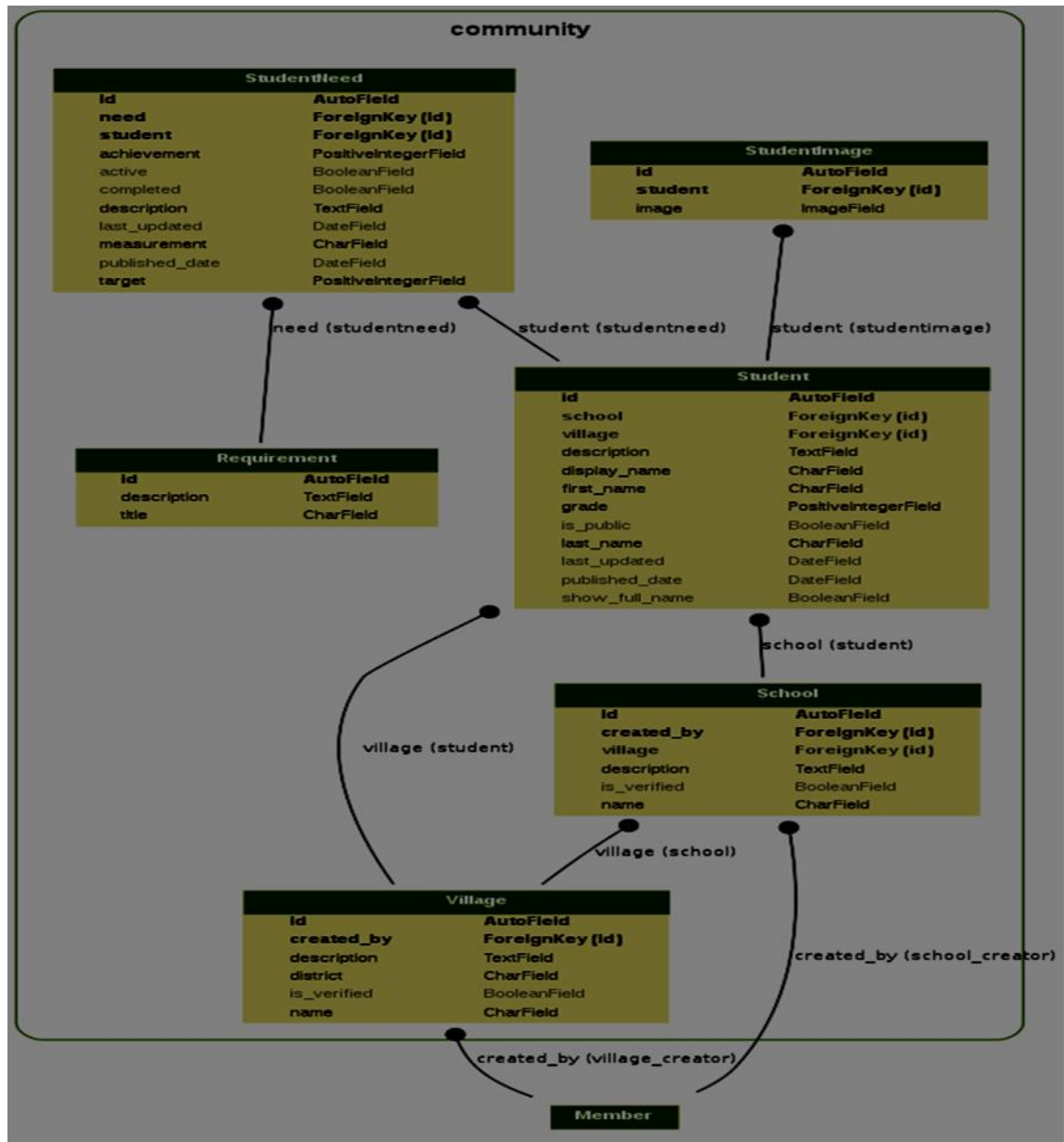
Village and School models. The other models which are related to Member, i.e., Group and Permission are resulted as an extension of Django authentication system.



Picture 5. 'Member' model from 'people' app.

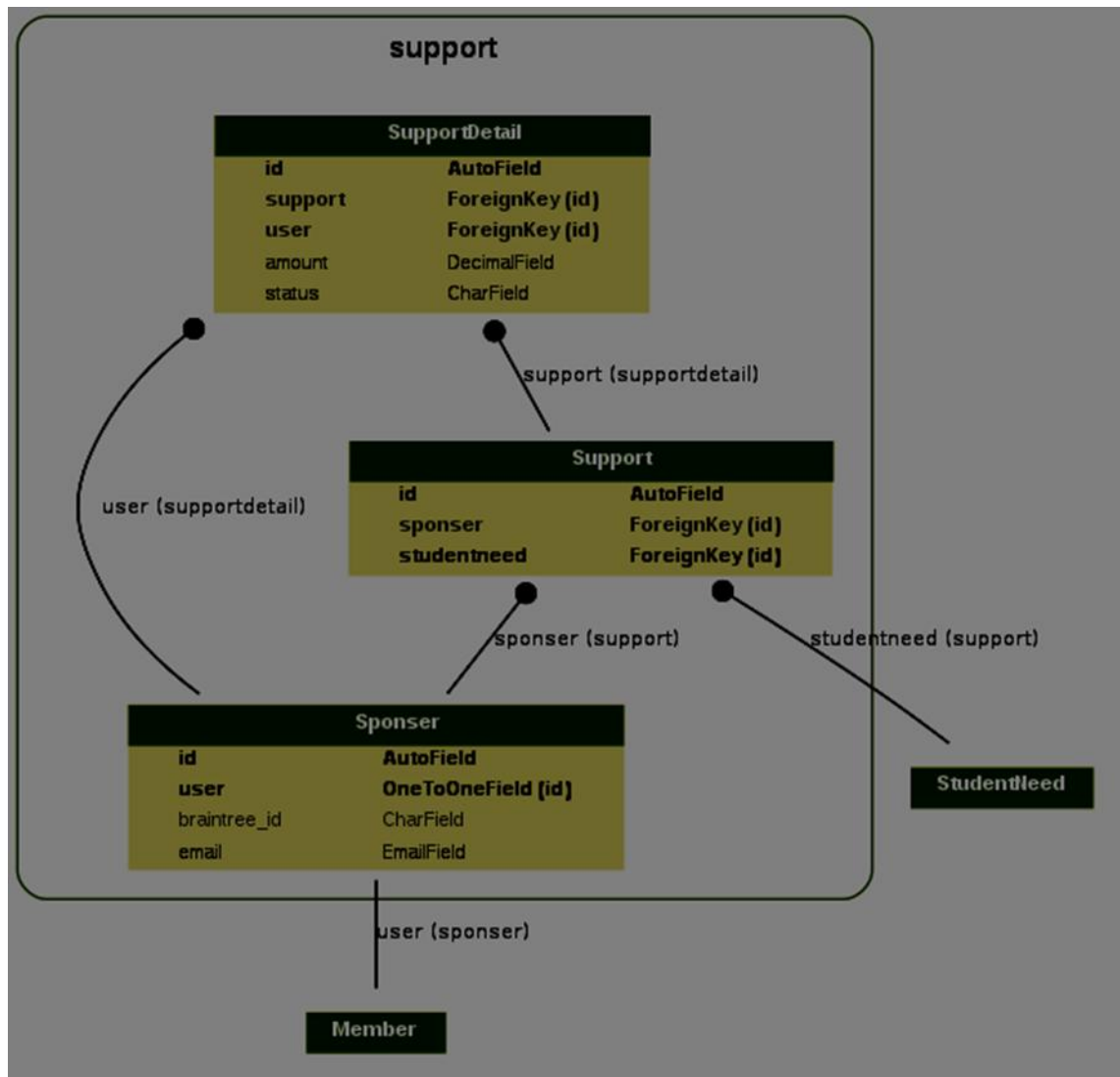
The 'community' app contains models which define entities related to community. The user could have been defined within this app, but our approach to the development was incremental based on functionalities rather than thorough database modeling in the beginning. As we started first to make the use of social logins to the site, the app 'people' was a result – which we let it be as a user from a community can also contribute outside of their community and to public in widest possible range. The models in 'community' app are as shown in the picture below. 'Student', 'Requirement', 'Village', 'School', and 'StudentNeed' are the core models in the community app. 'Requirement' is left generic which means a Requirement is not associated only with a student or a village or a school so that the admin can create the Requirement which applies for general student needs. StudentNeed is a special model – which defines one particular need of a particular student but can have multiple sponsors. StudentImage model is used to save image of the Student which is trivial from the app development perspective whereas important from the usability perspective. It is noteworthy to notice the relationship between School and Village with Member – which is labeled as

created_by in the picture – that a School or Village is created by the user who belongs to the local community. Logically, the users are restricted to create communities haphazardly as well.



Picture 6. Models from 'community' app.

The other important app is 'support' as it includes the details about the sponsors and related information in the context of meeting the need of the student or child. The models in 'support' app are as shown in the picture below:



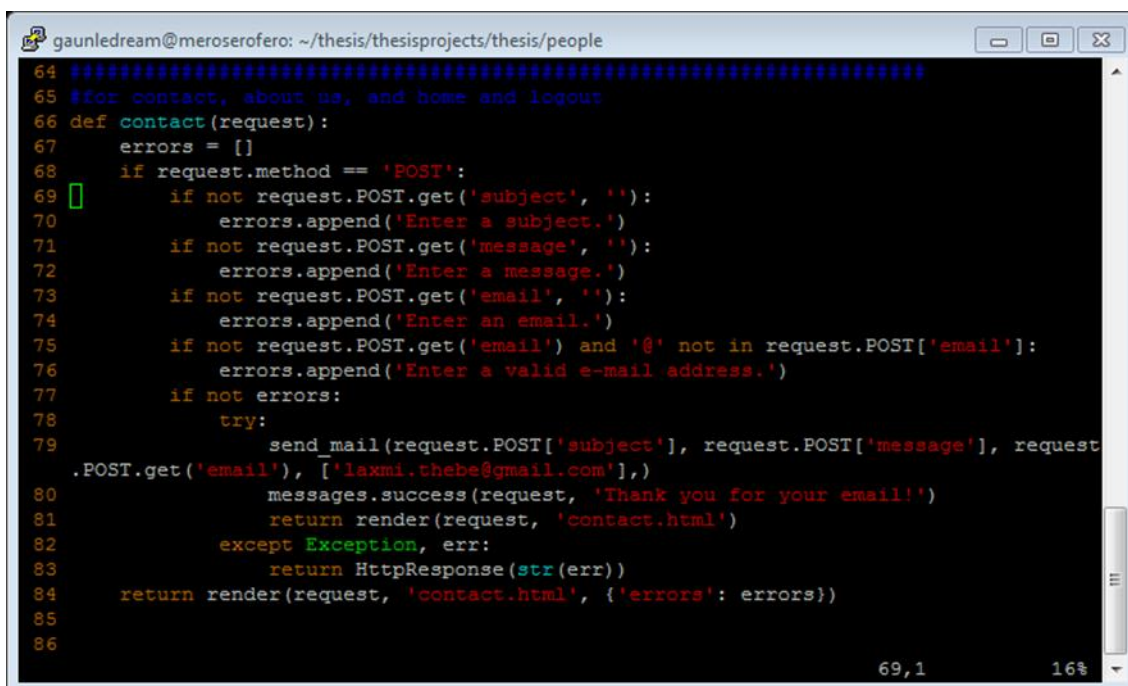
Picture 7. Models from 'support' app.

The Sponser model contains braintree_id which is used for financial transaction using Braintree payments. Support and SupportDetail are quite confusing, which are like promise and actions. In Support, a sponsor is recorded who promises to meet the particular StudentNeed (one need of one student) whereas in SupportDetail event of the supports are recorded so that a user can sponsor a particular need parallel with time. It is noteworthy to explain the many-to-many relationship between Sponser and StudentNeed through the Support model.

4.3 Views in the project

Views are like controllers which define what is to be passed to the template for rendering in the browser. Additionally, it uses form when user input is to be stored in the database. When an app is created within a project, a skeleton file for view called 'views.py' is created inside the app folder – which is edited according to the need of the app and what the particular app is supposed to do. There are two types of views in Django – function-based views and class-based views. For the purpose of apps, class based views are used whereas function based views are used in cases like contact page. Class-based views are good choices when the project becomes larger that inheritance is needed. Moreover, Django comes shipped with some generic class based views which can be inherited to our own class-based views.

The following picture illustrates the function based view which is used to generate contact page. This view does not use any models, although written in view file belonging to the people app.



```

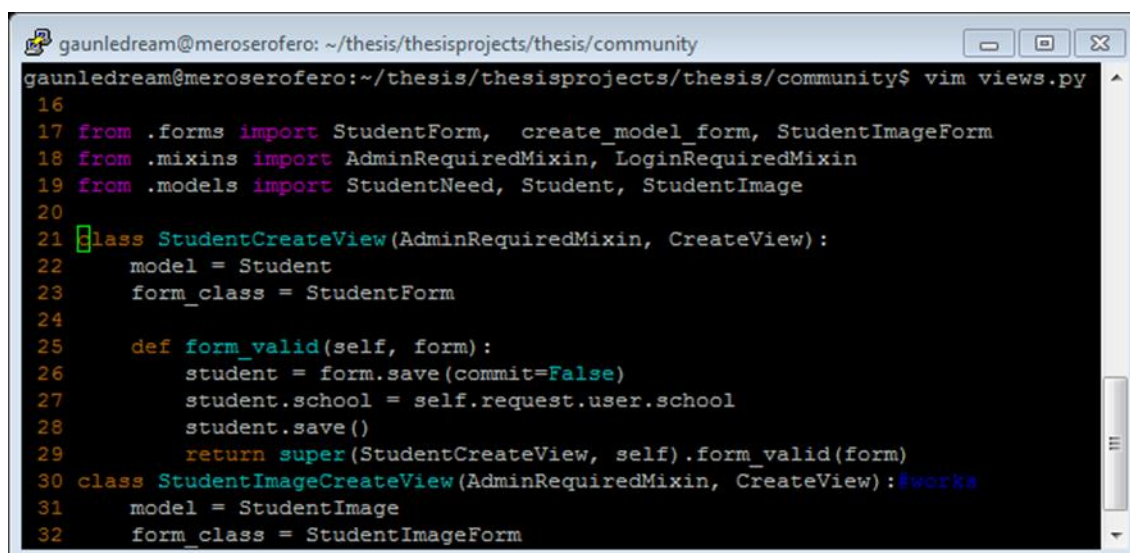
64 #####
65 #for contact, about us, and home and logout
66 def contact(request):
67     errors = []
68     if request.method == 'POST':
69         if not request.POST.get('subject', ''):
70             errors.append('Enter a subject.')
71         if not request.POST.get('message', ''):
72             errors.append('Enter a message.')
73         if not request.POST.get('email', ''):
74             errors.append('Enter an email.')
75         if not request.POST.get('email') and '@' not in request.POST['email']:
76             errors.append('Enter a valid e-mail address.')
77         if not errors:
78             try:
79                 send_mail(request.POST['subject'], request.POST['message'], request
80 .POST.get('email'), ['laxmi.thebe@gmail.com'],)
81                 messages.success(request, 'Thank you for your email!')
82                 return render(request, 'contact.html')
83             except Exception, err:
84                 return HttpResponse(str(err))
85         return render(request, 'contact.html', {'errors': errors})
86

```

Picture 8. Function-based views for 'contact' page.

As limited in the number is the function-based view in this project, there are many class-based views used in this project. The example shown below in the picture demonstrates StudentCreateView – which inherits the Django offered generic

CreateView and AdminRequiredMixin. The model is defined as Student (refer above in the model description) and form as StudentForm. When the form is valid during form submission, the student is saved but not committed as there is yet to set the School the student belongs to. As admin can add student only from the school to which the admin belongs, the school of the student is set the same as the admin's school before committing the save.



```

gaunledream@meroserofero: ~/thesis/thesisprojects/thesis/community
gaunledream@meroserofero:~/thesis/thesisprojects/thesis/community$ vim views.py
16
17 from .forms import StudentForm, create_model_form, StudentImageForm
18 from .mixins import AdminRequiredMixin, LoginRequiredMixin
19 from .models import StudentNeed, Student, StudentImage
20
21 class StudentCreateView(AdminRequiredMixin, CreateView):
22     model = Student
23     form_class = StudentForm
24
25     def form_valid(self, form):
26         student = form.save(commit=False)
27         student.school = self.request.user.school
28         student.save()
29         return super(StudentCreateView, self).form_valid(form)
30 class StudentImageCreateView(AdminRequiredMixin, CreateView):#works
31     model = StudentImage
32     form_class = StudentImageForm
  
```

Picture 9. Example of View from 'community' app.

The view is rendered using the default template name located default location – in this case, stored inside the app folder in templates/community/ as 'student_form.html' where community is app name, student is model name, and _form is template_name_suffix as shown in Picture 11. The AdminRequiredMixin is mixin which contains logic to allow only School Admin to do tasks. AdminRequiredMixin displayed below in the picture can be used in other class based views which needs feature that only School Admin is allowed to access the page – otherwise it returns a Response saying 'you are not school admin – check community/mixins and raise Http404' which is the message to generate a good-looking page when access is restricted for a user who is not School Admin.

```

gaunledream@meroserofero: ~/thesis/thesisprojects/thesis/community
1 from django.contrib.auth.decorators import login_required
2 from django.utils.decorators import method_decorator
3 from django.http import Http404, HttpResponseRedirect
4 from django.shortcuts import redirect
5
6
7 class AdminRequiredMixin(object):
8
9     @classmethod
10    def as_view(self, *args, **kwargs):
11        view = super(AdminRequiredMixin, self).as_view(*args, **kwargs)
12        return login_required(view) #to hide the admin site (can be done by changing the a
admin url)
13
14    @method_decorator(login_required)
15    def dispatch(self, request, *args, **kwargs):
16        if request.user.is_school_admin:
17            return super(AdminRequiredMixin, self).dispatch(request, *args, **kwargs)
18        else:
19            return HttpResponseRedirect("you are not school admin - check community/mixins and rai
se Http404")
20            #return Http404
21
22 class LoginRequiredMixin(object):

```

Picture 10. Django Mixin.

Additionally the project uses other mixins as well since they can be useful to class based views. For example, LoggingRequiredMixin is used extensively in this project to restrict access to some pages if the user is not logged in to the application.

```

gaunledream@meroserofero: ~/thesis/thesisprojects/thesis/community/templates/community
gaunledream@meroserofero:~/thesis/thesisprojects/thesis/community/templates/community$ cat student_form.html
{% extends "base.html" %}
{% block content %}
<div class="row">
    <div class="col-sm-8 col-sm-offset-2 text-center" style="background-color: rgb(204, 204, 204)">
        <form method="POST" enctype="multipart/form-data" action=""> {% csrf_token %}
        {{ form.as_p }}
        <input type="submit" value="Create Student" class="btn btn-default">
        </form>
    </div>
</div>
{% endblock content %}
gaunledream@meroserofero:~/thesis/thesisprojects/thesis/community/templates/community$

```

Picture 11. Template Example.

As shown in the template file above, `{{form}}` tag - which is the default form name in context - is all needed to render the form in the response where `{{form.as_p}}` is form rendering option where each element of form is displayed within paragraph (`<p></p>` tags). The picture 'StudentCreateView rendered' below demonstrates how the template from above is rendered in browser to the School Admin.

The screenshot shows a web application interface for creating a student. At the top, there is a navigation bar with links: Home, Search, My Serofero, Students, Posts, About, and Contact. A 'Logout' button is in the top right corner. The main content area contains a form with the following fields: 'First name' with the value 'Jange', 'Last name' with the value 'Created', 'Display name' with the value 'JangeCreated', 'Village' with a dropdown menu showing 'Sinam in taplejung', 'Grade' with the value '10', 'Is public' with a checked checkbox, and 'Show full name' with an unchecked checkbox. Below these fields is a message box that says 'This was created By Jange'. At the bottom of the form is a 'Create Student' button.

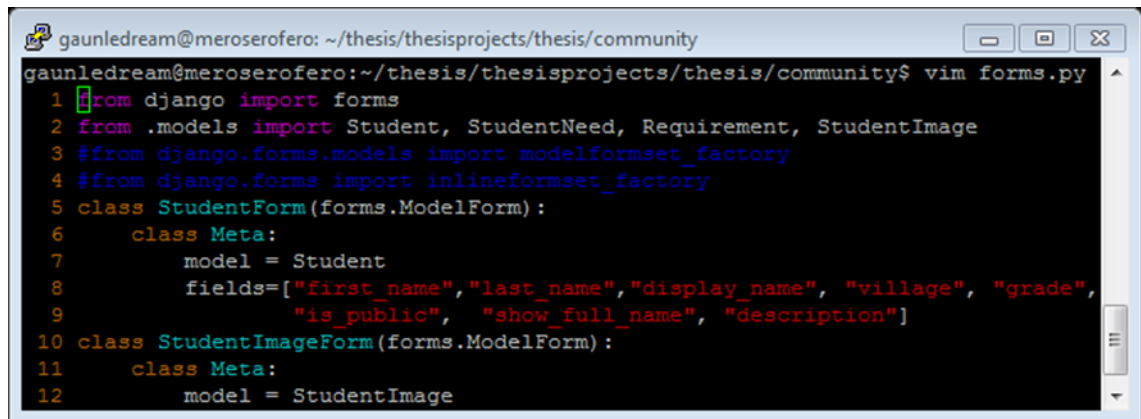
Picture 12. StudentCreateView rendering the view in the browser.

In conclusion, the function-based views are simple – they use method decorators for what mixins do in the class based views and always return `HttpResponse` object. Class-based views are convenient approach to writing views in Django, yet it takes time to become familiar with base classes shipped with Django to start working coding. The concept of context is also important to note in the case of view whether it is function-based or class-based views. Django compiles the template once and contains variable names within double curly braces. In the template example above (Picture 11), for example `{{form}}` is such a variable – which is passed to the template as dictionary variable containing 'key': 'value' pair. The class-based view – as it uses the base class shipped with Django – does not define the context of its own and uses the inherited one (Picture 9) whereas the function based view declares `{'errors':errors}` as context (Picture 8 – line 84). Finally, as described in Section 4.5, these views are invoked by Django once it matches the URL request sent by browser to the `urlpatterns` in `URLconf`.

4.4 Forms in the Project

Forms are an important component of the web development process – particularly if the web application is data-centric and accepts user inputs. It is possible to write the form elements in plain HTML but Django facilitates dealing with the form in many ways. Django Form is the class which works to generate what a normal form is supposed to do, whereas Django `ModelForm` associates the fields in the model with the form elements. Picture 13 demonstrates the `StudentForm` where fields in line 8 define what field from the `Student` model (Picture 14) are to be displayed as form elements (which

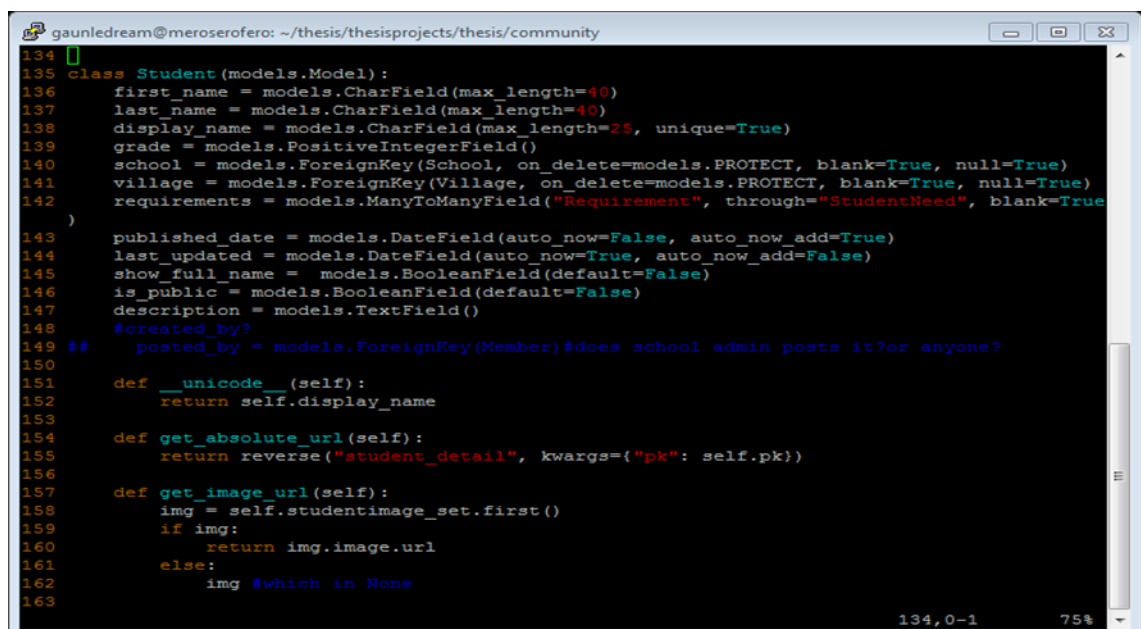
is shown in Picture 12 above). Additionally, Meta class is used to define model associated with this particular form.



```
gaunledream@meroserofero: ~/thesis/thesisprojects/thesis/community
gaunledream@meroserofero:~/thesis/thesisprojects/thesis/community$ vim forms.py
1 from django import forms
2 from .models import Student, StudentNeed, Requirement, StudentImage
3 #from django.forms.models import modelformset_factory
4 #from django.forms import inlineformset_factory
5 class StudentForm(forms.ModelForm):
6     class Meta:
7         model = Student
8         fields=["first_name","last_name","display_name", "village", "grade",
9             "is_public", "show_full_name", "description"]
10 class StudentImageForm(forms.ModelForm):
11     class Meta:
12         model = StudentImage
```

Picture 13. Example of Form in the 'community' app.

Comparing the form fields defined in the Meta class in StudentForm above in Picture 13 with the model fields in Student below in Picture 14, all fields are not displayed in the form when rendering in the browser. Some fields are set during the view processing, whereas some fields are set automatically; and some fields can be optional as well.



```
gaunledream@meroserofero: ~/thesis/thesisprojects/thesis/community
134
135 class Student(models.Model):
136     first_name = models.CharField(max_length=40)
137     last_name = models.CharField(max_length=40)
138     display_name = models.CharField(max_length=25, unique=True)
139     grade = models.PositiveIntegerField()
140     school = models.ForeignKey(School, on_delete=models.PROTECT, blank=True, null=True)
141     village = models.ForeignKey(Village, on_delete=models.PROTECT, blank=True, null=True)
142     requirements = models.ManyToManyField("Requirement", through="StudentNeed", blank=True)
143
144     published_date = models.DateField(auto_now=False, auto_now_add=True)
145     last_updated = models.DateField(auto_now=True, auto_now_add=False)
146     show_full_name = models.BooleanField(default=False)
147     is_public = models.BooleanField(default=False)
148     description = models.TextField()
149     #created_by?
150     posted_by = models.ForeignKey(Member, on_delete=models.PROTECT, blank=True, null=True)
151
152     def __unicode__(self):
153         return self.display_name
154
155     def get_absolute_url(self):
156         return reverse("student_detail", kwargs={"pk": self.pk})
157
158     def get_image_url(self):
159         img = self.studentimage_set.first()
160         if img:
161             return img.image.url
162         else:
163             return None
```

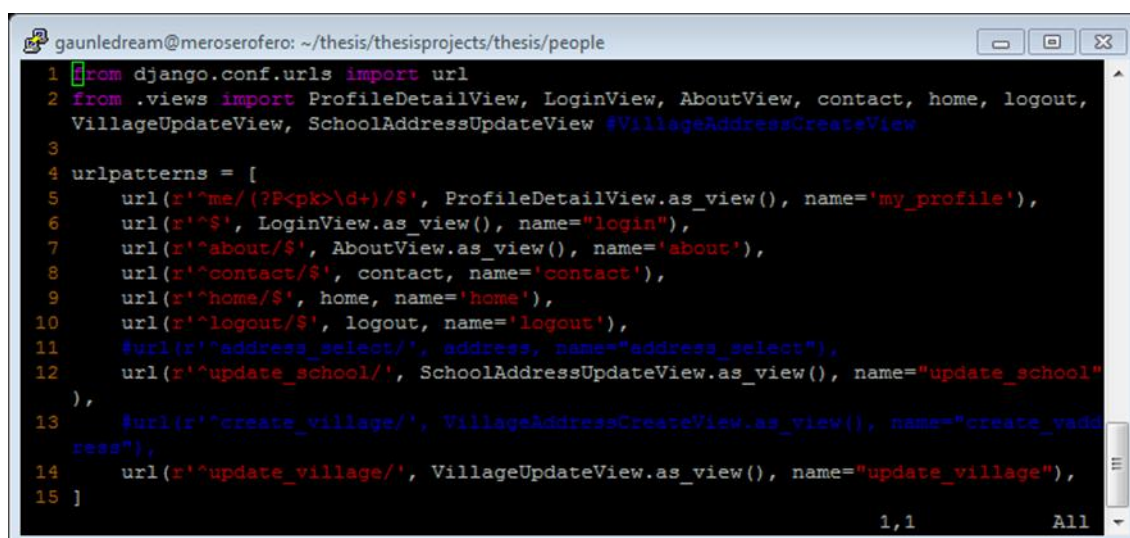
Picture 14. Student model in 'community' app.

In conclusion, a number of forms are used in this project. Some model forms are generated on the fly – for example, when editing or adding the student needs, a logged-in Student Admin sees only the students from his school which was only possible by generating the form on the fly. Additional complexities can appear in form handling when there is a needs for some customization – for example, when form fields validation is dependent on each other (for example, the user can not have the same password as their email). The form described in this section appears to be easy although it can be slightly daunting for the beginners when customization is required. For example, generating a model form on the fly was new in this project and took some time to deal with that issue.

4.5 URLConf in Django

Web pages are associated with the URL. When a user enters the URL in browser, this association between page and URL is the way Django knows what page to serve for a particular request. URLconf in Django is a Python module which Django uses to match the requested URL with the correct view to be served. Django uses regular expressions to declare URL patterns and proceeds in the pattern from top to bottom when a request comes until it finds the first match where it stops further processing of the pattern.

The following picture demonstrates the URLconf file for the 'people' app.



```

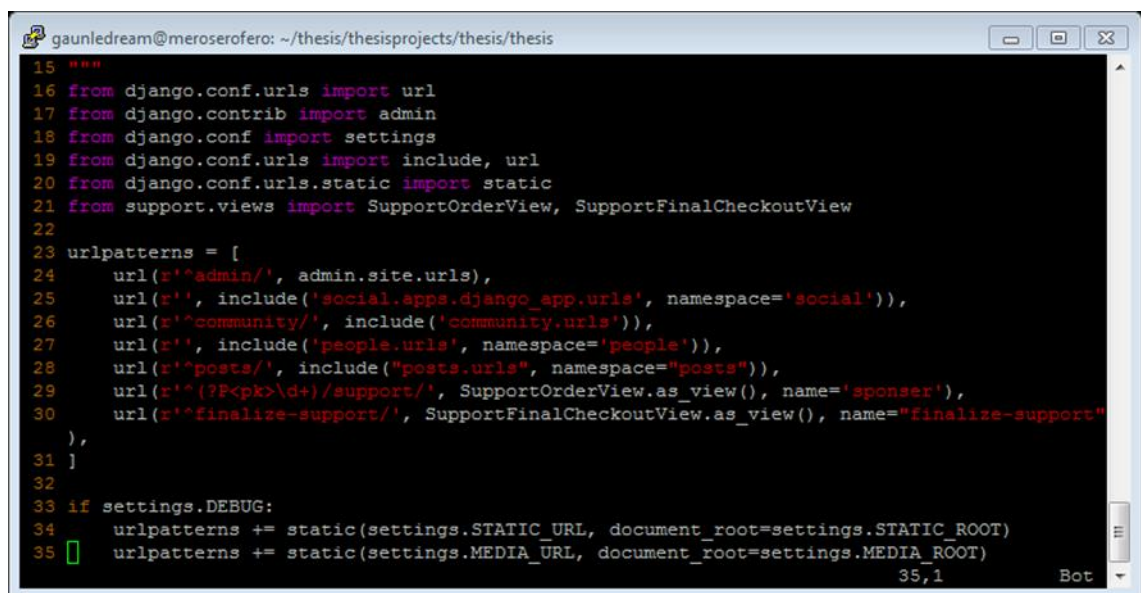
gaunledream@meroserofero: ~/thesis/thesisprojects/thesis/people
1 from django.conf.urls import url
2 from .views import ProfileDetailView, LoginView, AboutView, contact, home, logout,
  VillageUpdateView, SchoolAddressUpdateView #VillageAddressCreateView
3
4 urlpatterns = [
5     url(r'^me/(?P<pk>\d+)/$', ProfileDetailView.as_view(), name='my_profile'),
6     url(r'^$', LoginView.as_view(), name="login"),
7     url(r'^about/$', AboutView.as_view(), name='about'),
8     url(r'^contact/$', contact, name='contact'),
9     url(r'^home/$', home, name='home'),
10    url(r'^logout/$', logout, name='logout'),
11    #url(r'^address_select/', address, name="address_select"),
12    url(r'^update_school/', SchoolAddressUpdateView.as_view(), name="update_school"),
13    #url(r'^create_village/', VillageAddressCreateView.as_view(), name="create_vadd
14    url(r'^update_village/', VillageUpdateView.as_view(), name="update_village"),
15 ]
  
```

Picture 15. URLconf in 'people' app.

Django uses the following algorithm to figure out what page to serve:

- i. The root URLconf module – set as `ROOT_URLCONF` in `settings.py` file normally – is determined for use by Django unless overwritten by middleware in `HttpRequest` setting's `urlconf` attribute.
- ii. Django loads that URLconf to find urlpatterns i.e. `django.conf.urls.url()` instances.
- iii. The list of URL patterns are processed in order until the first match is found.
- iv. When the URL matches the regexes of the urlpatterns, the view specified in that pattern is imported and executed passing `HttpRequest`, arguments and keyword arguments if there are any.
- v. The best matching error-handling view is invoked in case no regex matches, or if an exception is raised during the request/response process – which is described in request/response process previously. (Django Software Foundation 2015e.)

The `urls.py` file in the above picture from the `people` app is imported in the `ROOT_URLCONF` file (created when the project was created in the beginning). The following picture reflects the import in line 27 so that Django can access to the urlpatterns defined in apps.

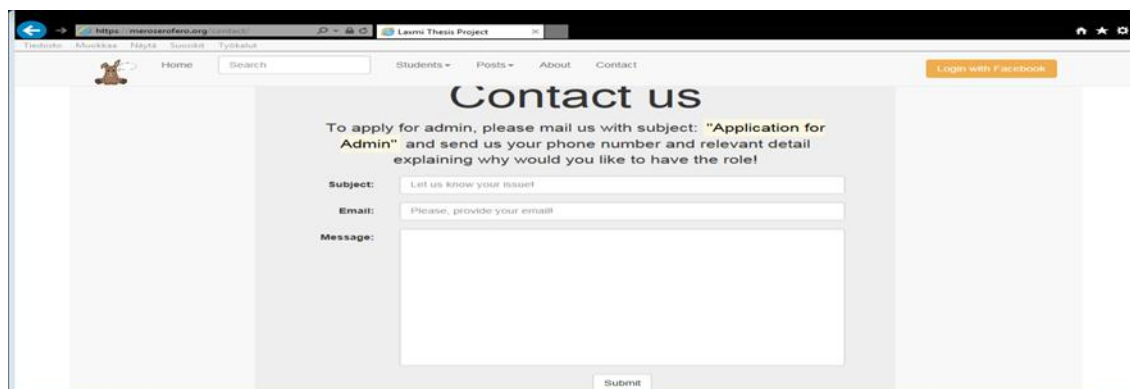


```

15 """
16 from django.conf.urls import url
17 from django.contrib import admin
18 from django.conf import settings
19 from django.conf.urls import include, url
20 from django.conf.urls.static import static
21 from support.views import SupportOrderView, SupportFinalCheckoutView
22
23 urlpatterns = [
24     url(r'^admin/', admin.site.urls),
25     url(r'', include('social.apps.django_app.urls', namespace='social')),
26     url(r'^community/', include('community.urls')),
27     url(r'', include('people.urls', namespace='people')),
28     url(r'^posts/', include('posts.urls', namespace='posts')),
29     url(r'^(?P<pk>\d+)/support/', SupportOrderView.as_view(), name='sponser'),
30     url(r'^finalize-support/', SupportFinalCheckoutView.as_view(), name='finalize-support'
31 ],
32
33 if settings.DEBUG:
34     urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
35     urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
36
37 Bot
  
```

Picture 16. `ROOT_URLCONF` file.

In this way, the urlpatterns defined in different apps is visible to the Django – hence urlpatterns can loosely hold different components together in this specific aspects of the development process.



Picture 17. Contact page on browser.

Since the URL <https://meroserofero.org/contact> matches with the `url(r'^contact/$', contact, name='contact')` urlpatterns as shown in line 8 in Picture 15 (URLconf in people app), Django imports the view called `contact` from `people.views` (`views.py` file in people app) as shown in the line 2 in Picture 15. The URL namespace can be used so that there would be no name conflict while reversing the named URL patterns even when there are same names for the URL patterns in different apps. In the example above, `name='contact'` is the name of the URL patterns. With namespace defined, the same name could be used in other app – which is effective for large projects developed by many developers. The name for urlpatterns is used to reverse to generate urls which is convenient in case where URLs are to be generated dynamically. So, the view `contact` is executed by Django – returning the response as shown in the Picture 17 if there is no error during the request/response process.

There are URLconfs in other apps as well and the basic way of writing URLconf is the same. The above URLconfs file also demonstrates the use of arguments which are passed to the views as described in the above described algorithm.

5 APPLICATION DEPLOYMENT PROCESS

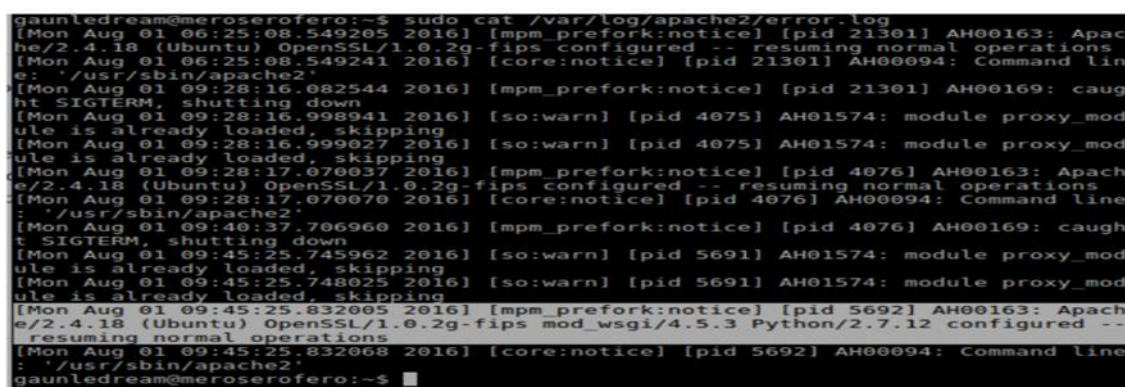
The thesis project was deployed in an Apache Web Server using mod_wsgi. The back end data was stored using the MySQL database management system. The following sections describe the setting up and configuration of the deployment process in detail.

5.1 mod_wsgi and deployment of Django in Apache server

mod_wsgi can be installed also with pip - a package management system used for installing and managing python packages (Python for beginners 2012) - command which generates configuration automatically; however for this project, the traditional approach of installation is opted where Apache is configured to load mod_wsgi manually. The source code in tar ball form was downloaded using the wget command and unpacked as:

```
gaunledream@meroserofero:~$ tar zxvf 4.5.3.tar.gz
```

As noted in the Google project page of the module, the apache2-dev package was also installed as in this case the apache was installed from package repository issuing the 'sudo apt-get install apache2' command in the Ubuntu server (Dumpleton 2016b).



```
gaunledream@meroserofero:~$ sudo cat /var/log/apache2/error.log
[Mon Aug 01 06:25:08.549205 2016] [mpm_prefork:notice] [pid 21301] AH00163: Apache/2.4.18 (Ubuntu) OpenSSL/1.0.2g-fips configured -- resuming normal operations
[Mon Aug 01 06:25:08.549241 2016] [core:notice] [pid 21301] AH00094: Command line: /usr/sbin/apache2
[Mon Aug 01 09:28:16.082544 2016] [mpm_prefork:notice] [pid 21301] AH00169: caught SIGTERM, shutting down
[Mon Aug 01 09:28:16.998941 2016] [so:warn] [pid 4075] AH01574: module proxy_module is already loaded, skipping
[Mon Aug 01 09:28:16.999027 2016] [so:warn] [pid 4075] AH01574: module proxy_module is already loaded, skipping
[Mon Aug 01 09:28:17.070037 2016] [mpm_prefork:notice] [pid 4076] AH00163: Apache/2.4.18 (Ubuntu) OpenSSL/1.0.2g-fips configured -- resuming normal operations
[Mon Aug 01 09:28:17.070070 2016] [core:notice] [pid 4076] AH00094: Command line: /usr/sbin/apache2
[Mon Aug 01 09:40:37.706960 2016] [mpm_prefork:notice] [pid 4076] AH00169: caught SIGTERM, shutting down
[Mon Aug 01 09:45:25.745962 2016] [so:warn] [pid 5691] AH01574: module proxy_module is already loaded, skipping
[Mon Aug 01 09:45:25.748025 2016] [so:warn] [pid 5691] AH01574: module proxy_module is already loaded, skipping
[Mon Aug 01 09:45:25.832005 2016] [mpm_prefork:notice] [pid 5692] AH00163: Apache/2.4.18 (Ubuntu) OpenSSL/1.0.2g-fips mod_wsgi/4.5.3 Python/2.7.12 configured -- resuming normal operations
[Mon Aug 01 09:45:25.832068 2016] [core:notice] [pid 5692] AH00094: Command line: /usr/sbin/apache2
gaunledream@meroserofero:~$
```

Picture 18. mod_wsgi installation confirmation.

The unpacked source code was configured with the ./configure command; the package was built with the make command and then installed in standard location by issuing the make install command – the location being dictated by Apache for its modules. The error log reveals the presence of mod_wsgi as highlighted in the picture above. After

confirming the presence of mod_wsgi, the Apache server was restarted with the 'sudo /etc/init.d/apache2 restart' command and the 'make clean' command was issued to clean up after installation. The downloaded tar ball and unpacked source files were deleted.

The virtual host configuration was updated to include the line 1 as shown in the figure below to load the mod_wsgi module. Additionally, the figure includes the complete virtual host configuration for the site which includes the implementation of HTTPS protocol for the secure communication between clients and the server. Note that there is only a single WSGIDaemonProcess defined for both HTTP and HTTPS as they run within the same daemon process.

```
gaunledream@meroserofero: ~/thesis/thesisprojects/thesis
1 LoadModule wsgi module /usr/lib/apache2/modules/mod_wsgi.so
2 <VirtualHost *:80>
3     ServerName meroserofero.org
4     ServerAlias www.meroserofero.org
5     ServerAdmin admin@meroserofero.org
6     #
7     DocumentRoot /home/gaunledream/thesis/thesisprojects/thesis/static
8     WSGIScriptAlias / /home/gaunledream/thesis/thesisprojects/thesis/thesis/wsgi.py
9     #WSGIPythonPath /home/gaunledream/thesis/thesisprojects/thesis:/home/gaunledream/the
10     sis/thesisprojects/lib/python2.7/site-packages
11     WSGIDaemonProcess meroserofero.org python-path=/home/gaunledream/thesis/thesisprojec
12     ts/thesis:/home/gaunledream/thesis/thesisprojects/lib/python2.7/site-packages
13     WSGIProcessGroup meroserofero.org
14     <Directory /home/gaunledream/thesis/thesisprojects/thesis/thesis>
15         <Files wsgi.py>
16             Require all granted
17         </Files>
18     </Directory>
19     #
20     Alias /favicon.ico /path to favicon
21     Alias /media/ /home/gaunledream/thesis/thesisprojects/media_cdn/
22     Alias /static/ /home/gaunledream/thesis/thesisprojects/static_cdn/
23     <Directory /home/gaunledream/thesis/thesisprojects/media_cdn>
24         Require all granted
25     </Directory>
26     <Directory /home/gaunledream/thesis/thesisprojects/static_cdn>
27         Require all granted
28     </Directory>
29     RewriteEngine on
30     RewriteCond %{SERVER_NAME} =www.meroserofero.org [OR]
31     RewriteCond %{SERVER_NAME} =meroserofero.org
32     RewriteRule ^ https://%{SERVER_NAME}%{REQUEST_URI} [END,QSA,R=permanent]
33
34     ErrorLog ${APACHE_LOG_DIR}/error.log
35     CustomLog ${APACHE_LOG_DIR}/access.log combined
36 </VirtualHost>
37 <VirtualHost *:443>
38     ServerName meroserofero.org
39     ServerAlias www.meroserofero.org
40     ServerAdmin admin@meroserofero.org
41     #DocumentRoot /home/gaunledream/thesis/thesisprojects/thesis/static
42     WSGIScriptAlias / /home/gaunledream/thesis/thesisprojects/thesis/thesis/wsgi.py
43     #WSGIPythonPath /home/gaunledream/thesis/thesisprojects/thesis:/home/gaunledrea
44     /thesis/thesisprojects/lib/python2.7/site-packages
45     WSGIDaemonProcess meroserofero.org
46     <Directory /home/gaunledream/thesis/thesisprojects/thesis/thesis>
47         <Files wsgi.py>
48             Require all granted
49         </Files>
50     </Directory>
51     #
52     Alias /favicon.ico /path to favicon
53     Alias /media/ /home/gaunledream/thesis/thesisprojects/media_cdn/
54     Alias /static/ /home/gaunledream/thesis/thesisprojects/static_cdn/
55     <Directory /home/gaunledream/thesis/thesisprojects/media_cdn>
56         Require all granted
57     </Directory>
58     <Directory /home/gaunledream/thesis/thesisprojects/static_cdn>
59         Require all granted
60     </Directory>
61     SSLCertificateFile /etc/letsencrypt/live/www.meroserofero.org/fullchain.pem
62     SSLCertificateKeyFile /etc/letsencrypt/live/www.meroserofero.org/privkey.pem
63     Include /etc/letsencrypt/options-ssl-apache.conf
64 </VirtualHost>
```

Picture 19. Apache VirtualHost configuration file.

After completing the VirtualHost file, the site was enabled and the Apache server was restarted again to bring the change to effect.

5.2 Deploying Django project with MySQL

For the deployment of Django with MySQL, the MySQL server was installed in the machine and the database called 'meroserofero' was created for the project. Moreover, practices for hardening database security were implemented as described later in this chapter. In the Django project's settings.py file, the database engine was set to be MySQL and the database access configuration was imported from another file as shown in the figure below:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'OPTIONS': {
            'read_default_file': '/home/gaunledream/thesis/thesisprojects/thesis/thesis/mydb.cnf',
        },
    },
}
```

Picture 20. Database Settings in settings.py file.

The database access configuration pointed above is as shown in the figure below. The password is masked to hide it from the readers. The default character set was set to be utf8 – which has briefly been discussed previously.

```
(thesisprojects) gaunledream@meroserofero:~/thesis/thesisprojects/thesis/thesis$ cat mydb.cnf
[client]
database = meroserofero
user = gaunledream
password = *****
default-character-set = utf8
(thesisprojects) gaunledream@meroserofero:~/thesis/thesisprojects/thesis/thesis$
```

Picture 21. Database access configuration file.

The package mysql-python was installed using the pip tool which is a Python interface for MySQL database i.e. database connection library. The development package files for database called libmysqlclient-dev were required to install in the system in order to

successfully install the mysql-python package in the virtual environment. After the process of deployment, the following security measures were considered.

5.3 Security Measures in Django Application

Security practices were adopted partly during the development and mostly during the deployment of the platform. Although the project is in its prototype version, the project in its nature is to be secured – particularly when it involves information about vulnerable people and financial transactions. For this reason, security requirements of the application were highly considered and the following measures were implemented:

In the project settings.py file, `DEBUG = True` was changed to `DEBUG = False`. Settings Debug on with the `DEBUG = True` option is only recommended during the development phase. In addition, the `DEBUG = True` option is also resource-consuming. For example, database queries will be saved in memory.

Host header validation was set in Django using `ALLOWED_HOSTS` setting. Actually, when `DEBUG` is set to `False`, it is necessary to set this option. In our case, it was set to be `meroserofero.org` as this domain is being served. This is a security measure against HTTP Host header attacks. (Django Software Foundation 2015a.)

The `SECRET_KEY` was not kept in settings.py file, rather it was saved in file and read from the file.

`ADMIN` was set in settings file – so that any error is reported by the Django site to the admin.

Regarding the security issues within the application, user privileges were used to limit the permissions for different users. To protect from cross site request forgery, Django provides the template tag which is used in all forms submitted from the browser using POST method. Only admins are allowed to create Student Profile – and normal users are only allowed to create posts and be sponsor. Financial donations can be offered even by anonymous users without logging in with Facebook logins but by providing an email address. One can only propose to be sponsor – and demonstrates that he or she is interested by paying certain amount of money (at the time, it is set up to be 1 dollar). Such requests are recommended for evaluation by the admin of the site (which can be also handed over to admin of the group i.e., school or village) so that the student needs

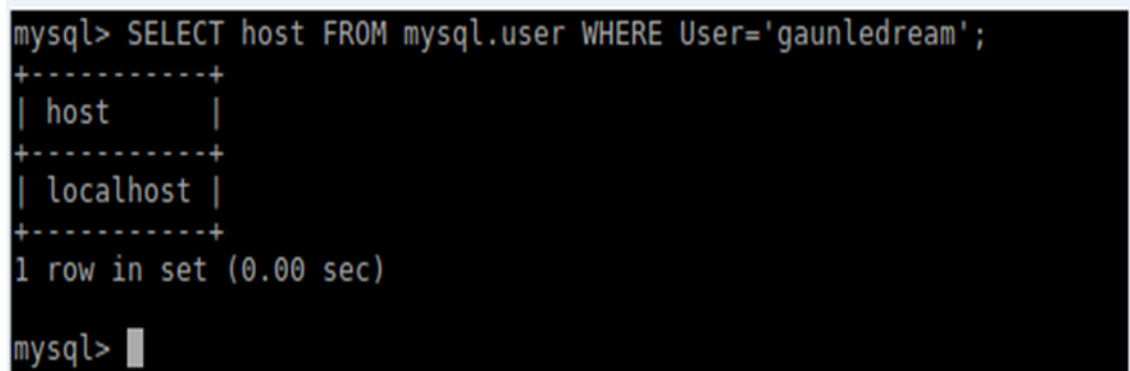
are met in practice as well. Additionally, HTTPS is implemented to protect the communication between client and server – so all requests in HTTP are redirected to HTTPS by Apache server by default.

For more information, security issues related to Django can be viewed by following this link (<https://docs.djangoproject.com/en/1.9/releases/security/>) which is the archive of security issues.

5.4 Database Security Issues

Databases have been the integral component of the websites and are of utmost important for data-centric sites. Because of such importance, the security of the database of this project was highly considered during the deployment of the project. Regarding this project, MySQL security was hardened using the following practices under consideration during the deployment:

- o the bind_address in MySQL configuration file is set explicitly to localhost as determined by the output demonstrated in the figure below.

A terminal window with a black background and green text. The prompt is 'mysql>'. The command entered is 'SELECT host FROM mysql.user WHERE User='gaunledream';'. The output is a table with one column 'host' and one row 'localhost'. Below the table, it says '1 row in set (0.00 sec)'. The prompt 'mysql>' is shown again at the bottom with a cursor.

```
mysql> SELECT host FROM mysql.user WHERE User='gaunledream';
+-----+
| host |
+-----+
| localhost |
+-----+
1 row in set (0.00 sec)

mysql> █
```

Picture 22. MySQL bind to localhost.

- o Uncomplicated Firewall (ufw) was installed and enabled to deny all incoming communication but SSH, HTTP, and HTTPS protocol as shown in the following output in figure.


```

gaunledream@meroserofero:~$ sudo ufw status verbose
[sudo] password for gaunledream:
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip

To Action From
--
80,443/tcp (Apache Full) ALLOW IN Anywhere
22 ALLOW IN Anywhere
80,443/tcp (Apache Full (v6)) ALLOW IN Anywhere (v6)
22 (v6) ALLOW IN Anywhere (v6)
gaunledream@meroserofero:~$ █

```

Picture 23. Firewall in Action.

- o User privilege was set to minimum so as to be incremented gradually as per need and separate users were created for different databases.
- o mysql_secure_installation utility was used to further harden the security in MySQL system.
- o MySQL root account was obfuscated by using different name. (Davidson, 2010)
- o SQL injection attack is protected in Django unless queries are executed with custom sql. In the case of this project, only Django's querysets are used which escapes the sql query by use of underlying database driver. (Nigel 2015.)

After securing the deployed system, scalability issues were considered as illustrated in the following section.

5.5 Scalability Issues in Application

For a project that aims big, scalability is equally important as security. Sometimes the performance overhead comes from latency in communication with other services. Cache could be implemented if application scales to such level. Varnish can be implemented – which is a piece of software with the ability to cache entire HTTP response hence able to serve response for particular request quickly and efficiently even without forwarding the requests to the Django server (Robenolt 2013).

Static files are served using the Apache Server itself.

mod_wsgi is implemented in daemon mode which makes predictable resource consumption due to the constant threads and processes involved.

The codes could be improved – for which Django Debug Toolbar can be used to fine tune the performance. It definitely involves significant mastery which the author is yet to develop.

Database sharding could be implemented if the data significantly grows. For example, data related to a group (in our case, either community or school) could be stored in a single database. Another option regarding scaling database would be MySQL replication – where Django can be configured to query inserts and updates to the masters and selects to the slaves (Shuping 2014).

The configuration of the deployment droplet (which is a virtual private server) could be increased when the traffic to the application grows.

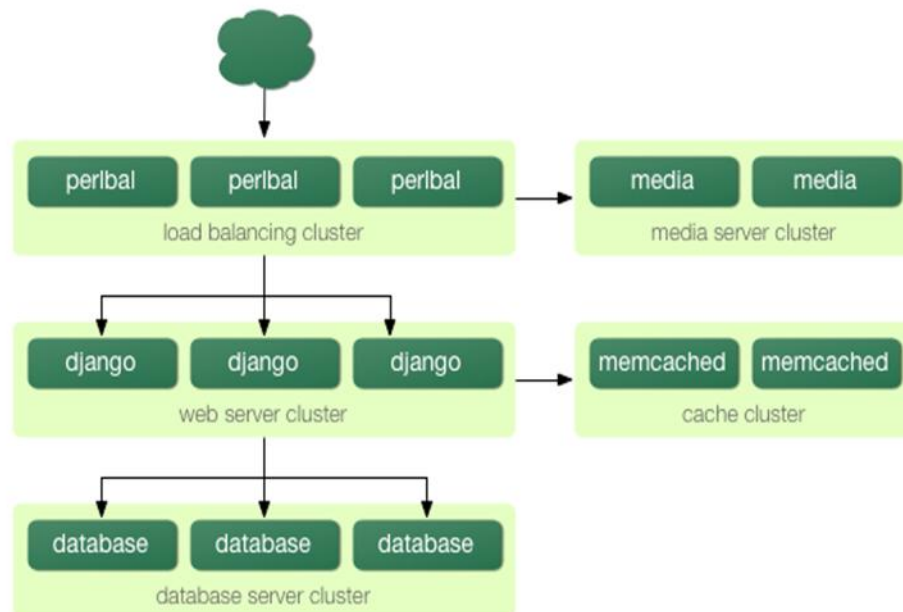


Figure 2. Deployment of Django in Large Setup (Holovaty and Kaplan-Moss 2009).

As discussed above, some of the scalability practices are already implemented whereas some of the practices are recommended for future development.

6 CONCLUSION AND RECOMMENDATION

As discussed in previous section, the endeavor of thesis project was to develop and deploy a Django web application to create community platform for supporting children upbringing in Nepal. The deployed version is available at <https://meroserofero.org> at the moment with the intention that it will be available in the same domain after the future development. The demo version of the project demonstrates the basic functionality out-lined in description of requirements. After completion of the development process, there were some issues encountered which need to be neddressed during the future development which are elaborated further in following sections.

6.1 Recommendation

Even though the project looks promising, there is much works to be done as well. For example, the issue of scalability is yet to be addressed – at least so that around two million people can be a user and approximately 10 percent of them would be using simultaneously as around two millions Nepali people are abroad. Pages which are static – for example, the ‘About Us’ page in the context of this project could have been served statically from the Apache server rather than passing through Django request/response process. Moreover, there are many usability issues which are to be addressed. One of the important ones is the login page – rather than using the homepage as login page, we could create some pop up window or a single page. Ajax functionality could be added in some of the graphical components as well. Regarding the web application itself, there is a need to redesign the overall architecture as well from the experience that is accumulated over the time. The approach to the design was to add functionalities incrementally from the beginning rather than starting up from planning – which is quite understandable as being the first time involvement in project of such scale with product as a basic requirement for the successful completion of the thesis. In the context of the development, we could also add features, such as project where activities that involve the well-being of the target group could be planned. Obviously, there are several promises as it is just the beginning of the project. Mobile application could be also implemented where the same backend could be used to serve data using REST API packages developed for Django Framework. Django

philosophies could be also used carefully – for example, DRY philosophy was not followed strictly when writing templates where template inheritance could have been used. Coding incrementally from the beginning was also the reason why there are some code repetitions particularly in templates. Testing is also an important aspect in web development which is also a missing part in this project. For learning purposes, it would not be recommended to use Facebook social plugin for comments. Yet, it was a preferred choice as users are logged in only using Facebook social login mechanism. The login mechanism could be also extended to include other services, such as Google, Twitter and others. In spite of these developments yet to be achieved, personal development was satisfactory as described in the next section.

6.2 Personal Development in the context of thesis project

Being the first endeavor in web development, the author has significantly gained skills related to web development on one hand; on the other hand, there are aspects to be developed further. The most important development is confidence to work in software projects even with no prior knowledge. Moreover, the author realized the importance of staying a bit longer sometime on the task at hand. Time management might be an important aspect in real life projects, but giving whatever time left for the working on the thesis proved to be significant on the thesis process. In the context of the technologies used, the author became familiar with Bootstrap front-end technology, with JavaScript to some extent, and with Django framework and Python programming in general. There were times when trying to figure out a single problem took hours, yet it was of immense satisfaction to get things done. Last but not least, the deployment of the project in the practical world helped to improve the skills with Linux server administration and other web domain related skills.

In conclusion, Nepal is a predominantly agriculture-based country – a significant part of the country being covered by villages in remote areas, the country is yet to be modernized in terms of information and technology. In one aspect, we have been using imported technology - a recent example being the number of mobile users more than the users of the toilet users which also implies the importance of developing mobile app of this platform. In that sense, mobile technology has revolutionized the way we are communicating inside Nepal. In addition to being dependent only upon borrowed technology, the author strongly argues that we (Nepali) need to develop tools that suit

our own needs or at least a perspective to use tools relevant to our own context. This thesis projects in itself might be of less significance at the moment, but it sheds light on the strong possibility of open source technologies to transform the way of our living in Nepal.

REFERENCES

- Bicking, I. 2014. Virtualenv. Consulted 01.08.2016 <https://virtualenv.pypa.io/en/latest/>.
- Davidson, M. 2010. MySQL Server Hardening. Consulted 29.08.2016 <http://security.stackexchange.com/questions/1138/mysql-server-hardening>.
- Django Software Foundation 2015 (a). Allowed_hosts. Consulted 12.08.2016 <https://docs.djangoproject.com/en/1.9/ref/settings/#allowed-hosts>.
- Django Software Foundation 2015 (b). Design Philosophies. Consulted 15.08.2016 <https://docs.djangoproject.com/en/1.10/misc/design-philosophies/>.
- Django Software Foundation 2015 (c). Django Overview. Consulted 12.08.2016 <https://www.djangoproject.com/start/overview/>.
- Django Software Foundation 2015 (d). Middleware. Consulted 19.08.2016 <https://docs.djangoproject.com/ja/1.9/topics/http/middleware/>.
- Django Software Foundation 2015 (e). URL dispatchers. Consulted 04.09.2016 <https://docs.djangoproject.com/en/1.10/topics/http/urls/>.
- Dumpleton, G. 2016 (a). Mod_wsgi. Consulted 24.08.2016 <https://modwsgi.readthedocs.io/en/develop/>.
- Dumpleton, G. 2016 (b). Summary Quick Installation Guide for Mod_wsgi. Consulted 22.08.2016 https://code.google.com/archive/p/modwsgi/wikis/QuickInstallationGuide.wiki#Apache_Requirements.
- Holovaty, A. & Kaplan-Moss, J. 2009. The Definitive Guide to Django : Web Development done Right. Berkeley: Apress.
- Kaphle, A. 2014. Nepal, once known for farming, now exports people; migrants earn big but face risks. Consulted 21.07.2016 <http://wpo.st/zqp02>.
- MySQL Documnetation Team 2016. What is MySQL?. Consluted 18.08.2016 <https://dev.mysql.com/doc/refman/5.7/en/what-is-mysql.html>.
- Neuman, E. 2015. "Ten Reasons Django is Perfect for Startups. Consulted 20.08.2016 <http://learn.onemonth.com/ten-reasons-django-is-perfect-for-startups>.
- Nigel, G 2015. Security in Django. Consulted 11.08.2016 <http://masteringdjango.com/security-in-django/>.
- Python for Beginners 2012. How to use Pip in Python?. Consulted 11.08.2016 <http://www.pythonforbeginners.com/basics/python-pip-usage>.
- Robenolt, M. 2013. Scaling Django to 8 Billion Page Views. Consulted 09.08.2016 <https://blog.disqus.com/scaling-django-to-8-billion-page-views>.
- Shuping, J. 2014. How to Scale Django. Consulted 10.08.2016 <http://www.javaworld.com/article/2362947/architecture-scalability/expert-interview-how-to-scale-django.html>.
- VIM 2016. Vim about. Consulted 23.07.2016 <http://www.vim.org/about.php>.