Expertise
and insight
for the future

Devndra Ghimire

Comparative study on Python web frameworks: Flask and Django

Metropolia University of Applied Sciences

Bachelor of Engineering

Media Engineering

Bachelor’s Thesis

5 May 2020

Author(s) Title Number of Pages Date	Devndra Ghimire Comparative study on Python web frameworks: Flask and Django. 37 pages + 0 appendices 5 May 2010
Degree	Bachelor of Engineering
Degree Programme	Media Engineering
Specialisation option	Software Engineering
Instructor(s)	Kari Salo, Senior Lecturer

The purpose of the thesis was to the study the various features, advantages, and the limitation of two web development frameworks for Python programming language. It aims to compare the usage of Django and Flask frameworks from a novice point of view. The theoretical part of the thesis presents the various types of programming languages and web technologies. In the practical part, however, the study is divided into two parts, each part observing the respective web application framework.

In order to perform the comparison, a social network and eCommerce like application was built for Flask and Django respectively. The comparison was started by developing the social network application first with Flask and finished with the e-commerce application using Django. Python programing language, SQLite database for the backend and HTML, JavaScript, and Ajax were used for the frontend technology. At the end of the project, both applications were deployed to the cloud platform called Heroku.

After the comparison, it was found that the most significant advantages of Flask were that it provides simplicity, flexibility, fine-grained control and quick and easy to learn. On the other hand, Django was easy to work with because of its extensive features and support for libraries. Another main advantage of Django is its scalability. It is best fit for a large-scale application. Each framework has its limitations and radiates a fair share of disadvantages. For example, Django is a bit cumbersome for smaller sized applications. However, Flask is too simple to not have the necessary features within the framework

As a result of this study, it seems that both frameworks are capable of serving production-grade web applications while having a fair share of advantages and disadvantages.

Keywords	Django, Flask, Python, Backend, Frontend, Web Framework

Contents

1 Introduction

From the inception of modern programming languages in the early 50s of the 20th century, numerous programming languages have been invented. A few have been discontinued along the road, and a few have survived with appreciation from computer scientists, programmers, and engineers. Of those survived, languages such as C, C++, Java, Python, JavaScript, and Ruby have been the go-to programming languages to build small to enterprises applications for desktop, mobile and other handheld devices.

While building a software application, one of the most important tasks of a programmer is to make the code readable, understandable and reusable. Often termed as DRY methodology, adopting this methodology not only reduces the boilerplate code in the codebase but also organizes the same logic in one place and makes it reusable in other parts of the codebase for the same purpose.

Another difficulty for the programmer is to understand the code which is written by other fellow programmers. If the same logic is repeated and found in many places, following the small piece of code could be a very demanding task as the source of truth for the same can be found in many places in the codebase resulting in multiple interpretations increasing the complexity. Using the software frameworks, boilerplate, duplicate and unstandardized code can be reduced.

A software framework is a set of standardized libraries and tools for a programming language. It helps to build a concise software application efficiently. Because of its code reusability and efficiency, it allows a programmer to bootstrap an application in no time and start implementing the business logic. Software frameworks come in different types for different programming languages.

In this thesis, however, we will try to compare and uncover the various aspects of the two most popular web application frameworks of Python programming language, i.e. Flask and Django. According to the documentation, Flask is as a microframework. Because of it being lightweight and having the features of flexibility and extensibility, it can be bootstrapped in no time. On the other hand, Django is known as a “battery included” framework, which claims to have most of the required extensions and libraries to bootstrap a generic application providing a developer more time on implementing the business logic.

2 Web Application Development

2.1 Background

Web application development is a process of developing software applications that can run on websites. Even though web application development follows the software development process, the technology and the architecture used for it is quite different. The software application that runs on a personal computer (PC) might not depend on the internet in contrast to the web application that depends upon the remote servers.

The web applications and the client-server technology have come quite far in comparison to the simple standalone phone book app made by Tim Berners-Lee in 1989 [1]. Nowadays, web application comes in different shapes and sizes such as static, dynamic, content management system, e-commerce, and gaming to live content sharing portals. Commonly shared technology of applications mentioned above types is the backend and frontend technology.

Backend development deals with the logical side of the web application. It mainly concerns the programming languages, core architecture and the logics. Those logics are mainly written in programming languages that can run on computer servers. It also influences how the data is stored, accessed and served from the servers. In Figure 1 below the green dotted line shows the backend part of the application, which consists of server-side scripts, frameworks, database and APIs.

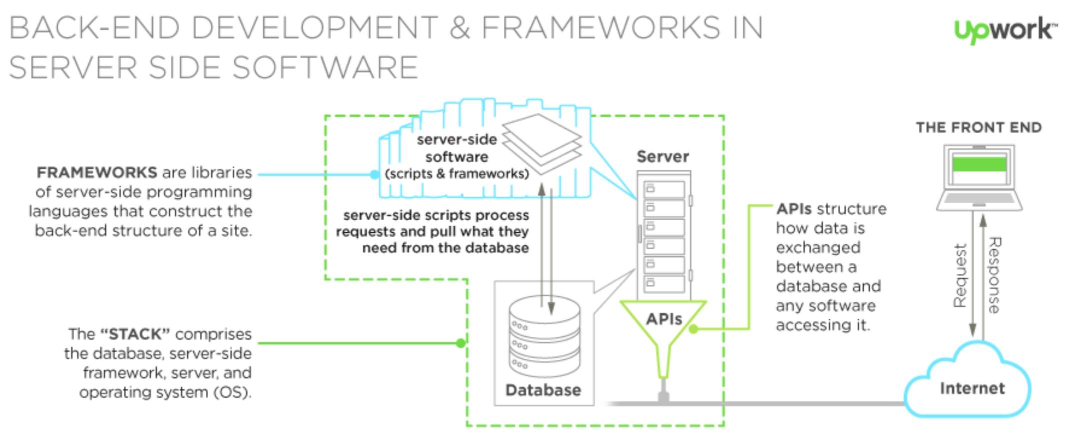

BACK-END DEVELOPMENT & FRAMEWORKS IN SERVER SIDE SOFTWARE
upwork
FRAMEWORKS are libraries of server-side programming languages that construct the back-end structure of a site.
server-side software (scripts & frameworks)
server-side scripts process requests and pull what they need from the database
Server
The "STACK" comprises the database, server-side framework, server, and operating system (OS).
Database
APIs
APIs structure how data is exchanged between a database and any software accessing it.
THE FRONT END
Request
Response
Internet

Figure 1 Back End Development [2]

On the other hand, frontend development is mostly concerned with the aesthetic and the content displaying part, which is also known as client-side development. One of its difficult challenges is to be able to show the material in the different types of devices and browser. Many devices have its own Software Development Kit (SDK) which should be followed to serve the same content that is served in the browser. Website design, usability and user-friendliness are the essential factors that are addressed during the development. The figure below is an example of how the frontend side of an application works.

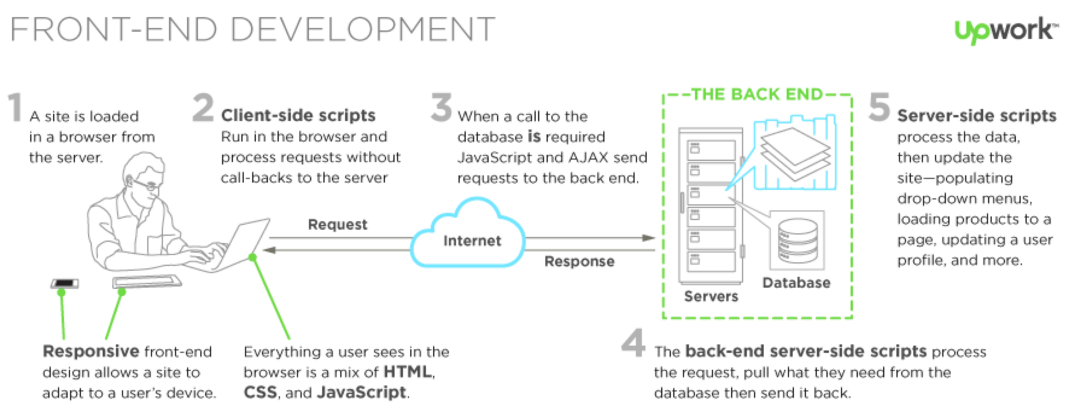

FRONT-END DEVELOPMENT
upwork
1 A site is loaded in a browser from the server.
2 Client-side scripts Run in the browser and process requests without call-backs to the server
3 When a call to the database is required JavaScript and AJAX send requests to the back end.
THE BACK END
5 Server-side scripts process the data, then update the site—populating drop-down menus, loading products to a page, updating a user profile, and more.
Request
Internet
Response
Servers
Database
Responsive front-end design allows a site to adapt to a user's device.
Everything a user sees in the browser is a mix of HTML, CSS, and JavaScript.
4 The back-end server-side scripts process the request, pull what they need from the database then send it back.

Figure 2 Front End Development [3]

2.2 Backend Technologies

Python

Founded in December 1989 by Guido Van, Python is one of the most popular high-level general programming language languages [4]. It is a dynamically typed language, often called duck typing in software engineering. It is an interpreted language; hence, it does not require any separate compiling time, but rather it is compiled to byte code and executed instantaneously.

One of the reasons for Python's popularity is its readability. Because the readable code is a dream for a software engineer, it is important because one should be able to understand the importance of the code regardless of the author. The code readability indeed depends upon the author. But Python steps ahead on this game with its very linguistic type programming language. It has also been touted as the easy way to learn the language. It will help programmers reduce the time spent on learning another programming language.

Because of its object-oriented architecture, it is not only the language for scripting anymore. The language has gone beyond the scripting and moved forward to the web and big data area. For web development purposes, frameworks such as Flask, Django, and Tornado exist. Also, the frameworks mentioned above are the go-to frameworks when it comes to the development of a proper web application in a short time. Apart from the web application development, it has also made its strong presence among data scientists. The abundance of the library and tools along with the broad community and clear documentation are its advantages. Since it is written in C [5], the performance-hungry application can take advantage of accessing the C's core data structure and functionality.

From its invention to date, it has gone through many changes while 2.7 is the most widely used Python version of all times. Python 3.7.0 [6] however, is the latest stable version at the time of writing.

Java

Java is a platform-independent high-level programming language. It was invented by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems in the year 1991 [7]. It is a compiled language. It, however, uses a two-step process of compilation. Firs it is compiled to bytecode by javac which is also known as the primary java compiler. Compiled bytecode will then be interpreted in by the Java Virtual Machine (JVM). Not all of the bytecode is compiled at once. The only required sequences of bytecode are compiled by the JVM's Just in Time Compiler (JIT) [8].

Java is renowned because of its portability in nature. The language itself is statically typed [8]. Compared to Python, it can execute multiple threads at once. Java is also widely used in enterprise applications. It is also one of the popular server-side programming languages. If Applets were the client-side applications, Serverlets would be the portable server-side applications that are server agnostic. Frameworks such as Spring MVC, JSF (JavaServerFaces), Struts, Hibernate, and Vaadin are a few of many. Java also has a strong presence in Big Data with technologies such as Hadoop, Apache Spark, and MapReduce. Because of its performance and interoperability, most of the time Java is selected to develop either a web, mobile or an enterprise application.

PHP

PHP, which stands for pre-processed hypertext, is a widely used programming language and it is particularly well suited for dynamic web applications development. PHP is a scripting language that only interprets program code in the execution phase of the program. PHP can be used on many different platforms and operating systems. In addition, the PHP source code is open source, which means that it may be researched, edited and distributed to the extent permitted by the license. Language syntax is easy, and it is mostly based on C, Java and Perl. [9]

C++

C++, often called as a first object-oriented language, was initially developed by Bjarne Stroustrup as an extension for C programming language. It is also one of the old programming languages. It is a compiled language. After its standardization in the year 1980, it has evolved into the modern programming language. It is a high-level language designed to perform faster and less error prone. It is widely used in performance-intensive applications such as games, web servers, operating systems, computer networks and embedded systems. Because of its efficiency, it has been the first choice in high energy physics, biology, animation, graphics, virtual reality and scientific computing [10].

GO

Go is an open-sourced programming language mostly developed by Google [11]. It is often termed as a C-like programming language or “C for the 21st century. The Go programming language was designed to overcome the common pitfalls of other programming languages such as slow compilation time, cumbersome code, memory handling and support for multiple cores. In 2009 Go became an open-source programming language [12]. Because of it being a system programming language, it is largely used in the distributed system and highly scalable network servers.

Go is also used in developing the web-based application. Frameworks like Gin, Beego, Iric, Echo etc. are a few examples.

Kotlin

Kotlin is a statically typed cross-platform programming language developed by JetBrains in 2010. It runs on the Java Virtual Machine (JVM). It became the primary language for

Android application development on May 7, 2019. Today, Kotlin has become the most popular programming language for Android development. It is being used by more than half of developers to build an android application. The biggest reason for Kotlin's popularity after Java is that the developer needs to write less code [13]. It provides more expressive syntax than Java with less boilerplate code. It has combined object-oriented and functional programming features. Interoperability with Java, less verbose code, null safety, no runtime overhead, and a wide range of collections are its main features [14]. JetBrains code editor IntelliJ Idea has native support for it. Apart from that, it is an open-source programming language.

Ruby

Ruby programming language is an object-oriented programming language developed by the Japanese computer scientist Yukihiro Matsumoto in 1990 [15]. One of the critical features of the Ruby programming language is, it is interpreted and dynamically typed. It allows procedural and functional styles of programming. It is easy to use with a steep learning curve. Since it is an interpreted language, the software developer will instantly know what part of the code is faulty. This makes the developer life more comfortable. Another benefit of this programming language is that it comes with a separate program called Interactive Ruby (irb) [16] which lets a programmer write a small script and experiment in no time. Ruby is also actively used for web development. One of the leading web frameworks for developing web applications for Ruby is called Ruby on Rails (ROR).

JavaScript

JavaScript is a dynamically typed language with asynchronous design. Brendan Eich developed it in May 1994.It was originally named Mocha and Live Script later. However, the original purpose of JavaScript was to add support for Java applets in the browser and beyond make their development so easy for non-experienced Java developers could do them, it was finally named JavaScript. [17]

Although JavaScript was easy to use, it was not warmly received by all developers, as it seemed to be unfinished in many ways. Despite its difficult beginnings, JavaScript is now practically the only programming language used in web browsers. It is capable of doing complex calculations, interactions, metaprogramming and closures [18]. The asynchronous styles of programming have resulted in its wider popularity in the backend community. One of the widely used JavaScript technologies for backend web development in Node.js.

2.3 Front End Technologies

HTML

HTML (Hypertext Mark-up Language) is the standard mark-up languages for the Web pages. It was created by Berners – Lee in the year 1989. It is a standard which describes the structure of web pages in which series of elements co-exists. Those elements then, instructs the browser how to display the content. These elements are represented by tags such as heading, paragraph, table, section etc. [19]. It is one of two core technologies required to build a dynamic or static web page. It is a skeleton or the layout of the web page.

HTML has wide ranges of support for the contents from text to spreadsheets, video clips, animations pictures [20]. The web page often called as a document in a technical term, the document type defined at the beginning of the page determines what type of document it is and how it should be taken by the web browser while rendering it. Being the essential part of the world wide web, it has gone through the various phases of development from 1991 till date, while HTML being the initial version and HTML5 being the latest one respectively.

CSS

W3C (Worldwide Web Consortium), CSS (Cascading Style Sheets) defines CSS as the language describing the presentation of Web pages, colours, fonts and the layouts.CSS allows one to stylize the HTML document according to their need. It contains a set of style rules with which the web pages could be rendered. Not only to that, but it also helps render the page responsively on the devices of different shapes and sizes.

There are two style formatting rules for the CSS. One is called inline styling, and another one is external styling. In the external styling, however, the styling rule sits in a different file. This could then be linked by a unique tag in the HTML document. The cascading part of the CSS refers to how the rules are applied to the HTML elements. The HTML document is styled hierarchically. Therefore, it is the responsibility of CSS to find the precedence of the style rules, accordingly, ultimately establishing a cascading effect [21].

JavaScript

As mentioned in 2.2.8, JavaScript was initially developed to replace the Java Applets which were used in the web browser. In 1996, one of the dominant browsers creating company, Netscape, submitted JavaScript to the Ecma International to influence its presence in all of the browser. Even though each browser has its implementation of the JavaScript, the underlying standard for it is the same. Chrome uses v8 JavaScript Engine; Internet Explorer uses Jscript in contrast to the Mozilla, which uses the standard JavaScript [22].

Fast forward almost twenty years; JavaScript has become one of the most popular languages for frontend technology. A simple script can be added inline within the HTML document or in a separate file with a link tag in the header file [23]. This way the HTML document knows which script to lead from which location. The main impacting feature of JavaScript is the ability to load or refresh the page content without reloading the whole page. This can be achieved by targeting only the concerned tag. Based on the requirement, it also helps on adding the CSS style dynamically. There are many JavaScript libraries available on the internet, which makes a developer’s life easier by leveraging the inner work of JavaScript and help focus on the aesthetic part of it. Some examples are React, Vue.js, jQuery Backbone.js, and Angular.

3 Python Web Frameworks

3.1 Flask

Flask is a micro-framework designed to create a web application in a short time. It only implements the core functionality giving developers the flexibility to add the feature as required during the implementation [24]. It is a lightweight, WSGI application framework. This framework can either be used for pure backend as well as frontend if need be. The former provides the functionality of the interactive debugger, full request object, routing system for endpoints, HTTP utilities for handling entity tags, cache controls, dates, cookies etc. [25]. It also provides a threaded WSGI server for local development including the test client for simulating the HTTP requests. Werkzeug and Jinja are the two core libraries

The Jinja, however, is another dependency of the Flask. It is a full-featured template engine. Sandboxed execution, powerful XSS prevention, template inheritance, easy to do debug, configurable syntax is it's few of many features. In addition, the code written in the HTML template is compiled as python code. Figure 3 below shows the Flask logo.

Figure 3 Flask logo [26]

Since Flask is often termed as a prototyping framework, it does not include the abstraction layer for the database or any sorts of validation and security whatsoever. Therefore, Flask has given full flexibility to the implementor to add the requirements.

There are extensions available for the Flask frameworks. Libraries but not limited to are, gunicorn for server, SQLAlchemy for database, Alembic for database migration management, celery & Redis for an asynchronous task runner, Flask-WTF form for form validation and Flask-limiter for rate-limiting the web requests. Flask is available for Python 3 and the newer version; it is also available in PyPy and easily installable with Python's official package manager pip [27].

3.2 Django

Django was introduced in 2003 when Adrian Holovaty and Simon Willison began using Python to build applications [28]. The motivation for developing Django was the need to get applications done on a schedule of days or even hours. Two years after starting to create a web application framework, in 2005, it was released as open source. Today, Django has several tens of thousands of users and development figures. [The Django Book 2009a.]

Figure 4 Django official Logo [29]

Because Django is a web application framework based on Python, you need Python to use it. Supported versions of Python are 2.6.5 and 2.7, but Django also offers experimental support from version 3.2.3 to version 3.3. Python comes with a lightweight testing server, but for proper use, the upcoming Django-based site needs an Apache server and a mod_wsg module. Django officially supports PostgreSQL, MySQL, Oracle and SQLite databases. [30]

Django also loosely applies MVC architecture. It becomes evident in Django, parts of the applications you created. In essence, models.py, views.py and urls.py, and the HTML template.Models.py contains a description of the information and provides functions for creating, retrieving, updating and deleting the data. View.py transmits the site content HTML template. The role of the urls.py file, in turn, is to configure which view file is called. With this structure and file functions, editing an application does not require editing many files, which in turn saves time.

3.3 Pyramid

The pyramid is a web application framework developed as a part of the Pylons Project. It is an open-sourced framework with BSD license. The pyramid is also built-in MVC architecture in mind. It was inspired by other libraries like Zope, Pylons 1.0 AND Django It takes a similar approach as Flask when it comes to the implementation details. Pyramid does not come with any ORM or other prebaked libraries.

Therefore, Pyramid gives more power and flexibility when it comes to adding new features. Pyramid is tested against Python versions 2.7, 3.4, 3.5, 3.6 and 3.7 [31]. A few of its features are as follows:

1. Templating System
2. Static assets
3. Testable
4. Generate dynamic URLs
5. Debugger mode
6. Extendibility with add-ons (libraries)
7. Built-in support for HTTP session
8. URL routing

Pyramid is available in PyPy and installable with the Python package manager pip. Below is a sample app with a very simplistic route.

```python
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response
from pyramid.view import view_config


@view_config(
    route_name='home'
)
def home(request):
    return Response('Welcome!')

if __name__ == '__main__':
    with Configurator() as config:
        config.add_route('home', '/')
        config.scan()
        app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 6543, app)
    server.serve_forever()
```

Figure 5 Pyramid App

3.4 Bottle

The Bottle is a minimal microframework, also termed as a true Python framework. It is a single file framework. Hence, it is lightweight and fast to implement. Bottle framework has no dependency outside the Python standard library and does not have many features, as other frameworks do [32]. The small applications and prototyping greatly benefit this framework. It comes with the following features out of the box [33].

1. Template engine and support for mako, jina2 and cheetah
2. Utilities like form data handling, file upload, cookies, headers and HTTP related metadata

3. Built-in development HTTP server
4. Support for other WSGI servers
5. URL routing

```
from bottle import route, run, template

@route('/hello/<name>')
def index(name):
    return template('<b>Hello {{name}}</b>!', name=name)

run(host='localhost', port=8080)
```

Figure 6 A simple bottle app with a view

3.5 CherryPy

In late June 2002, the first version of CherryPy was released by Remi Delon. It is the first framework of Python programming language [34]. It is a microframework with flexibility by design; hence, it is extensible. It has built-in tools such as sessions, authorization, caching, routing, JSON handling and support for databases. It comes with the HTTP/1.1-compliant WSGI thread pool server out of the box [35]. It can run multiple HTTP servers with multiple ports at once. Configuration management is one its dominant feature. For the extendibility, CherryPy's plugin system can be used. Tools can be created and used without any other dependencies. It is both available for Python 2 and 3 and also available for CPython Jython and PyPy.

```
import cherrypy

class Root(object):
    @cherrypy.expose
    def index(self):
        return "Hello World!"

if __name__ == '__main__':
   cherrypy.quickstart(Root(), '/')
```

Figure 7 A simple CherryPy app with a view []

3.6 Tornado

Tornado is one of the Python web frameworks which focuses more on the network and speed while including all the web frameworks features. It is primarily based on a Friend-Feed Framework developed by Bret Taylor and designed to serve a tremendous amount of traffic without any performance downtime [36]. Tornado can be used when the application needs web sockets, long polling or long-lived connections. Because of its asynchronous interactive connection feature, it benefits greatly from the network-intensive applications. Not only does it come with the performance-boosting features, but it also has features such as templating, routing, escaping, support for international localization, request handler and application configuration to build a simple to scalable web applications [36]. See Figure 8 below:

```
import tornado.ioloop
import tornado.web

class MainHandler(tornado.web.RequestHandler):
    def get(self):
        self.write("Hello, world")

def make_app():
    return tornado.web.Application([
        (r"/", MainHandler),
    ])

if __name__ == "__main__":
    app = make_app()
    app.listen(8888)
    tornado.ioloop.IOLoop.current().start()
```

Figure 8 Tornado framework simple app with a view

4 Comparison and Observation

In this section, we will try to dissect the experience and findings of the two similar but different apps that were built for the experiment. One application implements the ability to post and share content, i.e. a small social network application which is built on top of Flask. Another one, however, is a small e-shopping portal where one can browse and shop the products using the said implemented portal. This application is built using the Django web framework along with the required technologies.

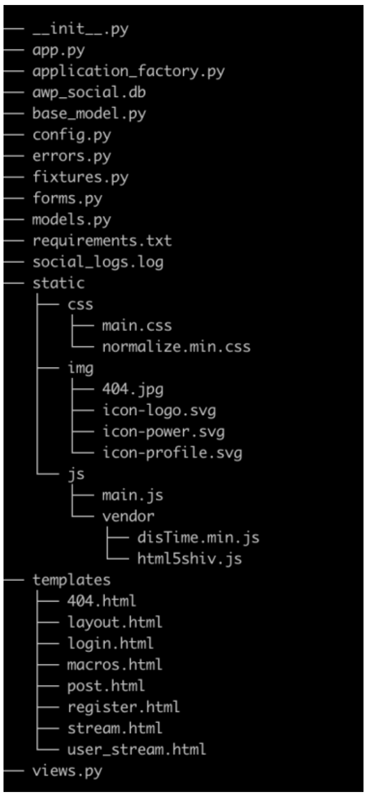
Both applications use SQL for the database and utilize the ORM from the frameworks of its own or utilize the third-party library or extension. The comparison also includes the observation on its core features, security, documentation and learning curves but not limited to those mentioned. Both the project has been done in a prototype in mind and does replicate the production level code. It includes both the backend and frontend using technologies like HTML, AJAX and JavaScript on the frontend. Even though both the frameworks are full-stack frameworks, the comparisons are more inclined towards the backend side of it.

4.1 Flask

Flask, as discussed earlier, is the microframework which facilitates on developing a prototype, small to large size web application. Since the project was mimicking a social network application like Facebook where one can add, follow and unfollow friend and post and comment on the posts etc., it required an easy to set up and quick result-oriented framework, hence the framework was the number one choice for the study.

4.1.1 Design Pattern

Even though the Flask does not strictly state that the application structure and design should be in one specific way, the social network app tries to mimic the suggested pattern. The directory structure is in a tree-like structure where it finds all the necessary views, templates models to execute the expected outcome. The app does use the application factory, logging and API/view exceptions for the error handling. Not only that it fully utilizes the view decorators. The application instance is created with the application factory, which is highly recommended by the framework itself. The Figure below shows the directory structure of the app.



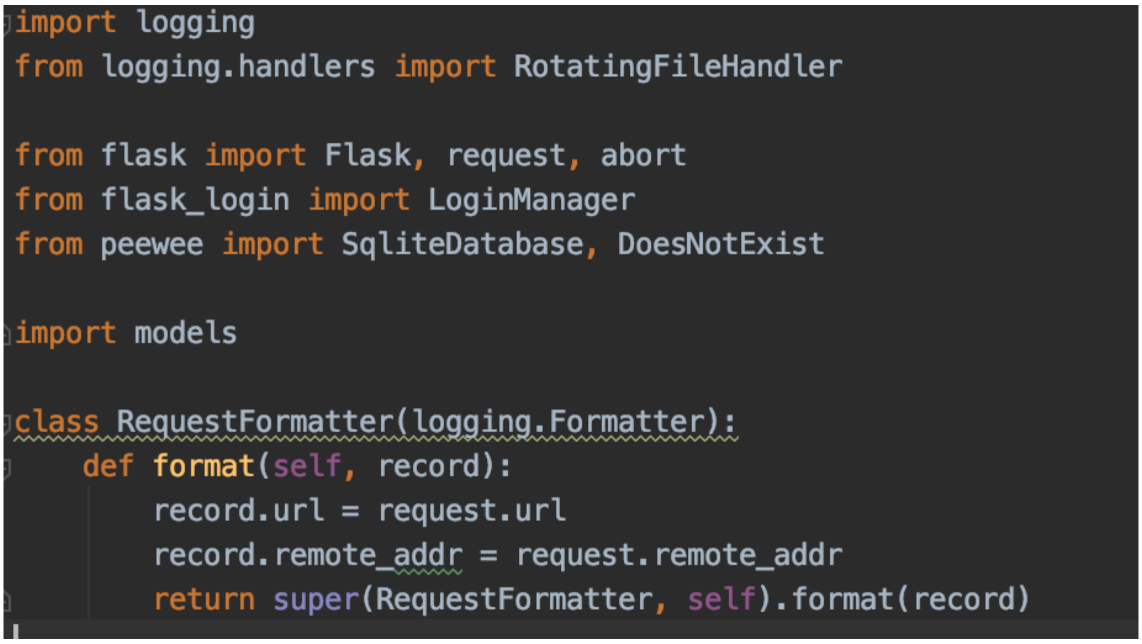
```
├── __init__.py
├── app.py
├── application_factory.py
├── awp_social.db
├── base_model.py
├── config.py
├── errors.py
├── fixtures.py
├── forms.py
├── models.py
├── requirements.txt
├── social_logs.log
├── static
│   ├── css
│   │   ├── main.css
│   │   └── normalize.min.css
│   ├── img
│   │   ├── 404.jpg
│   │   ├── icon-logo.svg
│   │   ├── icon-power.svg
│   │   └── icon-profile.svg
│   └── js
│       ├── main.js
│       └── vendor
│           ├── disTime.min.js
│           └── html5shiv.js
├── templates
│   ├── 404.html
│   ├── layout.html
│   ├── login.html
│   ├── macros.html
│   ├── post.html
│   ├── register.html
│   ├── stream.html
│   └── user_stream.html
└── views.py
```

Figure 9

4.1.2 Requests and Routing

Requests

Flask handles the request context automagically. Therefore, the need of passing request object to the function is not necessary. The request object is bootstrapped when the WSGI server observes a new request. Similar to the application context, Flask provides the request context from where the request object can be accessed. It is a global object which is available throughout the application lifetime.



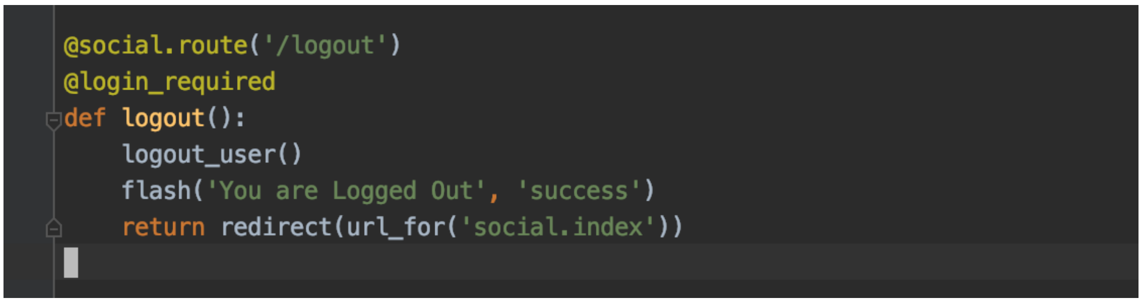
```python
import logging
from logging.handlers import RotatingFileHandler

from flask import Flask, request, abort
from flask_login import LoginManager
from peewee import SqliteDatabase, DoesNotExist

import models

class RequestFormatter(logging.Formatter):
    def format(self, record):
        record.url = request.url
        record.remote_addr = request.remote_addr
        return super(RequestFormatter, self).format(record)
```

Figure 10 Flask requests

Above snippet of the code is from the app where the logger is being configured. In this snippet, the request is being imported directly from the Flask and accessed directly inside the RequestFormatter class, hence no need to pass it explicitly to the function or the class.

Routing

As routes are the primary interface for the app or the end-user, it is usually constructed with the URL pattern. And these route's request parameter and the body can then consume by the views accordingly. In Flask, the view is decorated by the route decorator. The path can be constructed directly in the decorator by giving the name with the standard URL pattern. It is effortless to add a route in a Flask application. See Figure 11.



```
@social.route('/logout')
@login_required
def logout():
    logout_user()
    flash('You are Logged Out', 'success')
    return redirect(url_for('social.index'))
```

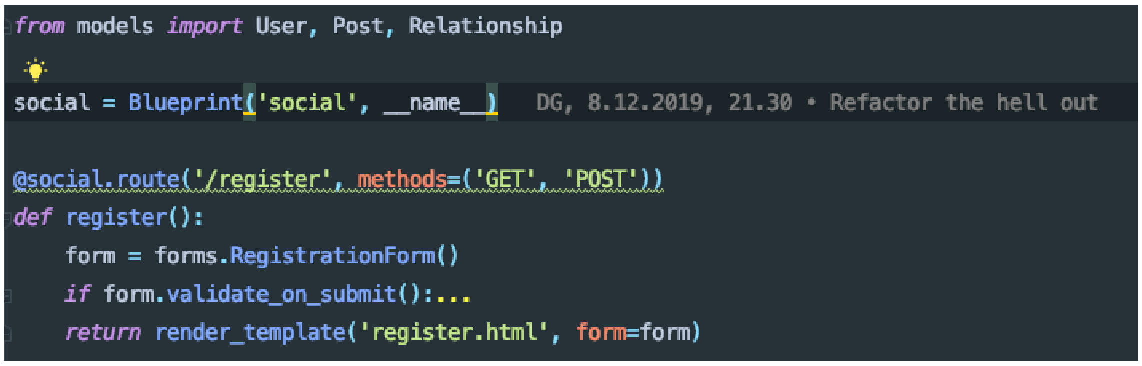
Figure 11 Flask routing

In the snippet above the **logout** is the view function and the route are `/logout`. When `http://localhost:5000/logout` is accessed via a web browser the `logout` view function is triggered, the user is logged out and redirected to the route for the `social.index.` which is the root route, i.e. `http://localhost:5000/.`

Blueprints

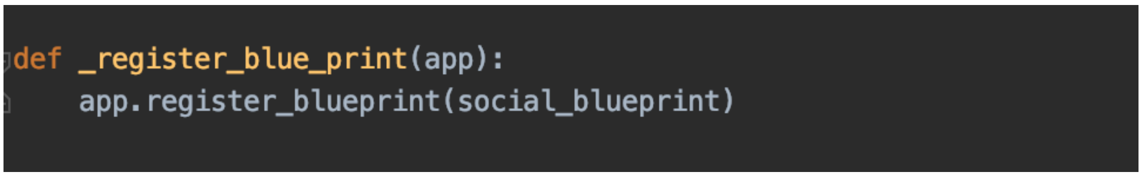
A blueprint is an object similar to the Flask application; however, it needs to be attached to the app itself. In Flask, one application can have multiple blueprints. This helps to organize the views, or so to say group them by either the domain-specific purpose or endpoint type. Once the blueprint is registered, and the endpoint is grouped, it is possible that the grouped views can act differently than another blueprint. Flask provides the `before_request` and `after_request` functions which can then be modified to inject or change the request-response cycle based on the needs.

Its standard that blueprint definition is done in the `__init__.py` file of the package, but for the brevity, the `social` blueprint is defined in the view file of the app as shown in the figure below.



```
from models import User, Post, Relationship

social = Blueprint('social', __name__)    DG, 8.12.2019, 21.30 • Refactor the hell out

@social.route('/register', methods=('GET', 'POST'))
def register():
    form = forms.RegistrationForm()
    if form.validate_on_submit():...
    return render_template('register.html', form=form)
```

Figure 12 View attached with the social blueprint



```python
def _register_blue_print(app):
    app.register_blueprint(social_blueprint)
```

Figure 13 Blueprint registration

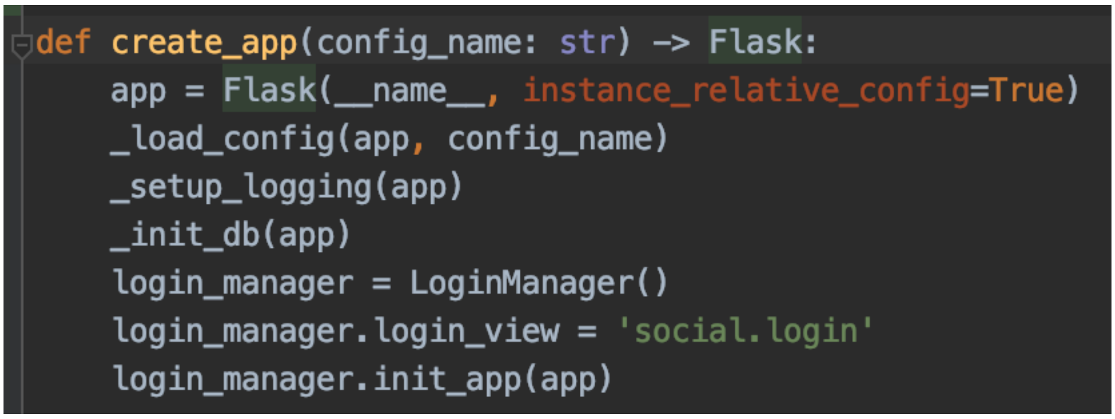
The figure above shows the registration of the blueprint named `social`.

4.1.3 Infrastructure and Configurations

Database

Flask provides every possibility of freedom when it comes to selecting a database. With that flexibility in mind, the social app was built using the relational database SQLite. Flask does not provide automatic database migration. Flask does not come with the support for the migration. The library called Alembic, which then can be used to generate the migration file. One needs to know what to look for. In the sample project, however, no migration has been used. All the database table has been generated during the application bootstrapped with the help of models which was provided by the object relation mapper (ORM) called Pewee.

Since the database is bootstrapped within the app. The database cannot be used outside the application context. The database was initialized and connected as shown in Figure 14 below



```python
def create_app(config_name: str) -> Flask:
    app = Flask(__name__, instance_relative_config=True)
    _load_config(app, config_name)
    _setup_logging(app)
    _init_db(app)
    login_manager = LoginManager()
    login_manager.login_view = 'social.login'
    login_manager.init_app(app)
```

Figure 14 Application bootstrapped

Security

Since the Flask is a microframework, it is the developers/implementer’s responsibility to think about the safety of the application. Having said that there are plenty of extensions to use which solves the issues. One, for example, ORM, already handles the problems of SQL injections. For CSRF (Cross-Site Request Forgery) however, there is an extension called Flask-WTF for forms which already utilizes the CSRF protection. If the form is not used inside the application, then wrapping the app by CSRFProtect from the Flask-WTF form library should do the job.

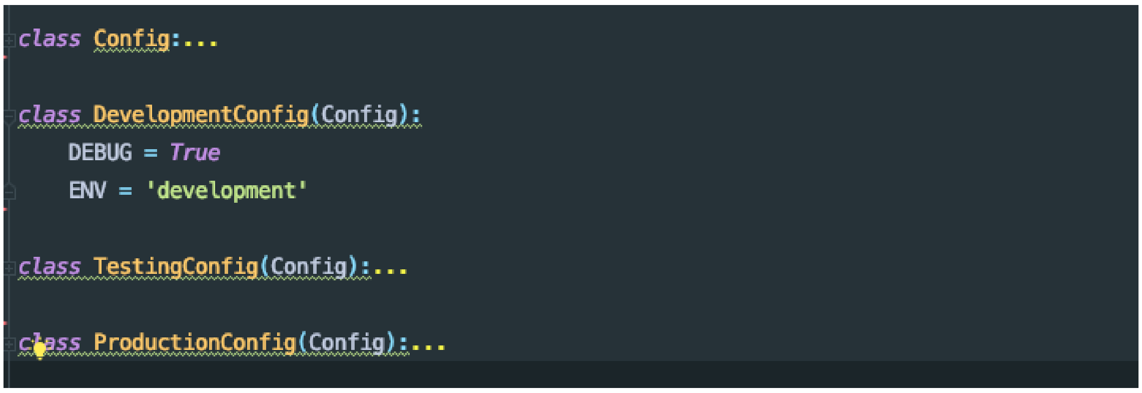
For user login and logout, the Flask-Login library was used. It was registered to the main view `social.login`. The login manager was then initialized during the application bootstrap. This can be seen happening in Figure 14 above.

Configurations

Every application needs a separate configuration for separate purposes. For example, settings should be different for the testing, development and production purposes. It is not best practice to have all the configuration in one object or one file. Because the configuration is very much an integral part of an application of the infrastructure. It is crucial to separate the setting purpose specific. This will help in securing the one-off passwords, database connection URL, private SSH keys and many more.

Generally, it is not a good idea to store those in the file itself; instead, those configurations should be picked up from the hosted machine environment variable and converted to the purpose-specific object. For example, the sample app uses the object approach while also adding the possibility of reading the required configuration from the environment variables. See Figure 15 below.

The sample app provides three different configurations, DevelopmentConfig for having the settings for app development purpose, TestingConfig for application testing purpose and ProductionConfig, which is used in the production environment when the app is live.



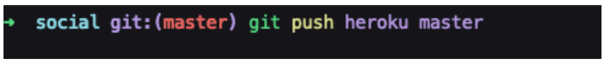
```
class Config:...

class DevelopmentConfig(Config):
    DEBUG = True
    ENV = 'development'

class TestingConfig(Config):...

class ProductionConfig(Config):...
```

Figure 15 Configuration handling

Deployment

To make the application accessible on the internet, Heroku a Platform as a Service provider is used. The reason behind choosing the Heroku is because of its seamless deployment process, easiness of the deployment, logging and its scalability.

Since the app is minimal in size, only one dynos/containers were used to deploy, and With Heroku CLI, the containers can be increased to the desired numbers. No additional resources were used except for the web. For the internet, the app uses the gunicorn WSGI HTTP server. This server is configured in a file called “Procfile” within the project from which the Heroku knows what type of server it should require to run. Because Heroku works with git, it can be easily deployed with one-liner command, as shown below.



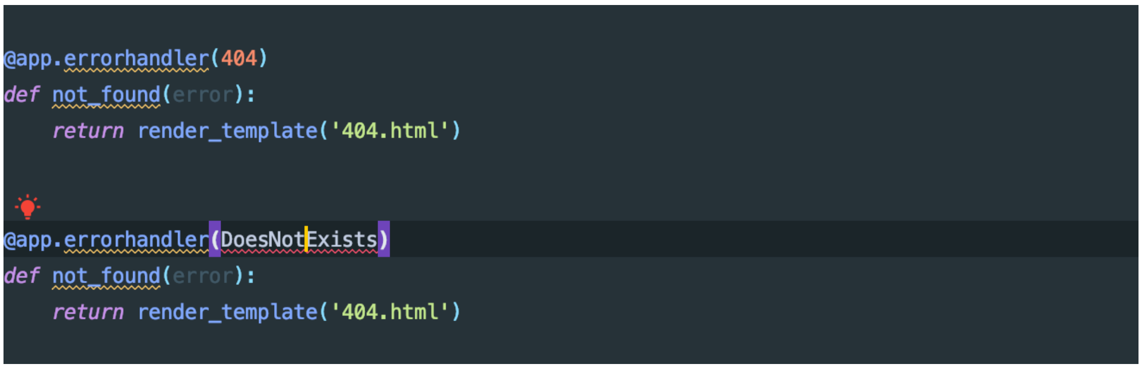
```
→ social git:(master) git push heroku master
```

Figure 16 Deploy in Heroku

4.1.4 Error handling

The application uses Python’s error handling techniques while throwing and catching exceptions when needed. This allowed the application to run seamlessly. With the help of logging and error tracking tools like Sentry. It made it much more comfortable to track and monitor the errors. However, Sentry was not used for brevity.

Flask approach to error handling is to bind the generic/custom python exceptions to the defined error handler. This error handler can be decorated to the view to render the error message, accordingly, as shown in the figure below.



```
@app.errorhandler(404)
def not_found(error):
    return render_template('404.html')

@app.errorhandler(DoesNotExists)
def not_found(error):
    return render_template('404.html')
```

Figure 17 Error Handling

4.1.5 Caching

Since the application created during this project was built with the minimal viable product mentality, hence any caching mechanism was not used whatsoever. The cache is useful when the application is getting a lot of traffic, meaning, a user might request the same content frequently. To reduce the server load from the same traffic, the same content can be saved to the preferred cache and then serve to the user from the cache instead of the primary server

According to the framework's official documentation, we can find few supported caching mechanisms. Currently supported types of caches are Filesystem Cache, Redis Cache, UWSGI Cache, Memcached Cache and SASLM Cache. Above mentioned types of caches have already built-in support for it. To use the Redis and Memcached Cache, however, the separate instance should be configured in the backend. For Memcached and SASL type cache, Pylibmc must be installed.

4.1.6 Flexibility

During the development of the framework, few extensions were required to make the app functional. Form, login manager, template engine and encryption etc. were needed. These libraries were installed need basis; hence the application is not bloated with overly installed libraries.

Flask framework is very much flexible in various aspects. As mentioned above the freedom of choosing the libraries when needed is the most important one. This helped to focus on the development of the app features instead of worrying about the performance and its maintainability. This helped to rocket the speed of the development and implementing the features like follow/unfollow functionality. From templating to security, all the features were ready and available to install. A few of the main factors are as follows:

1. It’s based on Model View Controller design pattern
2. Supports both function and class-based views
3. ORM agnostic pattern s
4. Abundant libraries to use for the full-fledged application from client to server-side
5. If any publicly available extensions do not cater to the need, then new extensions can be written easily
6. Pytest, Unittests, Nosetests are easily applicable for testing.

4.1.7 Developer Support

Documentation

Flask has an extensive documentation with full of examples and implementation details. The documentation can be found along with other application built under the umbrella of Pallets Project. Quite often, when developing an application, the main issue for the developer is to find the proper instruction of the implementation detail. The first point of the solution is usually the official documentation. Also, the focus is on the maturity and the comprehensiveness of the application matter.

During the development of the application, Flask’s documentation, however, has not been the issue. Implementation details have been organized and explained to the point where it’s needed the most. It also includes Jinja template engine and the Werkzeug WSGI toolkit documentations.

For the beginners, the documentation contains a section called QuickStart. A minimal and simpler application can be built by just following the previously mentioned section. Instruction is quite easy to read, straight forward and understandable. The documentation can be found on https://flask.palletsprojects.com/

Learning Curve

With the help of the Flask’s extensive documentation and the broader community in Stack Overflow, the time taken for learning and getting started was quite fast. Most of the general issues regarding know-how are covered in the documentation; therefore, the need for looking for help during any implementation was quite minimal.

Flask’s API is very more straightforward and easier to comprehend. They are quite straight forward, and it does exactly what it says unlike other APIs having hidden logic inside. Community plays a vital role in the popularity of an open-source software application. Flask, however, is very well known around the python developers on StackOverflow. Currently, on that platform, there are more than 35 thousand flasks tagged questions with answers. With that number of the problem posted, most of the general issues on the usage of the framework can already be found.

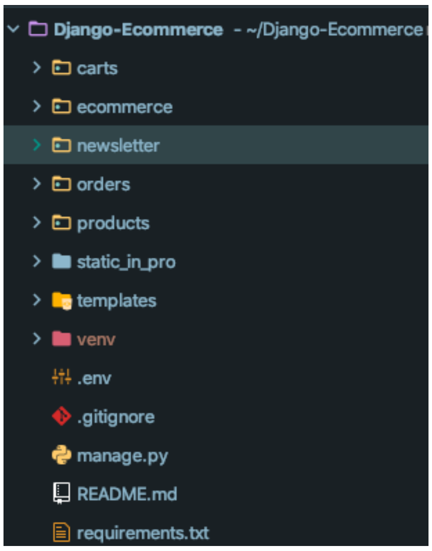
4.2 Django

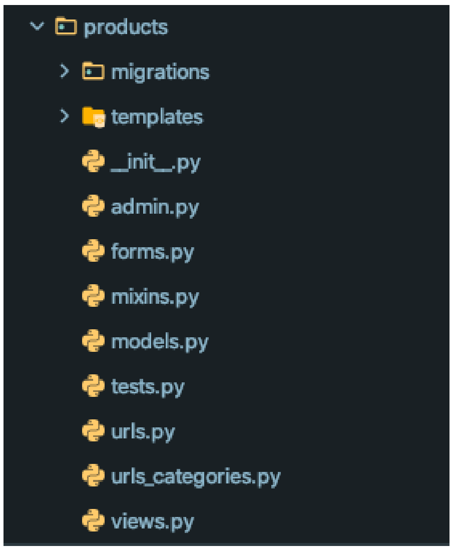
As discussed, earlier Django is a full-fledged battery included web application framework. An MVP version of an e-commerce application is build using the Django framework. Although the app developed for the comparison is different, the main motive of the study will be the same. The study should reasonably provide the distinctive properties of the framework from getting started steps to final deployment. For the study, Django 1.8 is used with python 2.7

4.2.1 Design Pattern

Often touted as the battery included web application framework, Django does come with a lot of functionality and support. Much smaller application can be bootstrapped within the main application are make them talk them talk to each other with simple APIs. It has its own rules and way of implementing the functionality. It also follows the MVC principle while saving valuable time of the developer. This is because most of the heavy lifting has already been done as long as the implementation follows the Django own rulesets.

As shown in the picture below, an organization of different sections of apps are packaged in a separate directory. Within those directories, we can see “admin.py”, “forms.py”, “models.py” and “views.py”, “urls.py” those are the standard filename from where Django tries to find the resources. Most of the database related implementation resides on the models.py and views.py consisting of the endpoint serving functions and urls.py containing the routings.



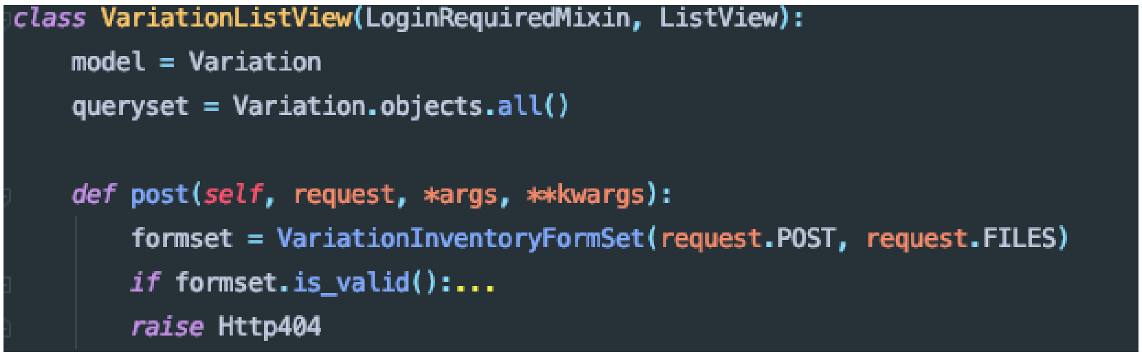


4.2.2 Requests and Routing

Requests

Django handles the requests differently in contrast to the Flask's request. In Django, requests object needs to be explicitly provided to the view function or the class. The request object then contains all the required information from the current application state to the current session. When a client requests a resource from an application, the HttpRequest object is created, and the corresponding view function is called. If the request object needs to be modified or accessed, it needs to be provided explicitly to the implemented function.

As shown in the figure below the request context is explicitly passed through the "post" function of a class-based view or access it through the class scope.

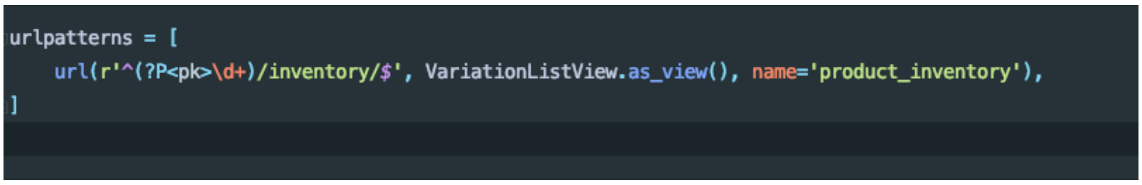


```
class VariationListView(LoginRequiredMixin, ListView):
    model = Variation
    queryset = Variation.objects.all()

    def post(self, request, *args, **kwargs):
        formset = VariationInventoryFormSet(request.POST, request.FILES)
        if formset.is_valid():...
        raise Http404
```

Figure 18 Django requests

Routing

In Django routing is handled through URL dispatcher. The view function/class is written somewhere and the URL with the required pattern. Default URL dispatcher is already configurable in the settings. Once the root URL is configured, other URL routes can be added. These routes can either be global or application specific. The routes can be configured in the “urls.py” as shown below. “VariationListView” is called when a user requests http://0.0.0.0:5000/products/1/inventory endpoint.

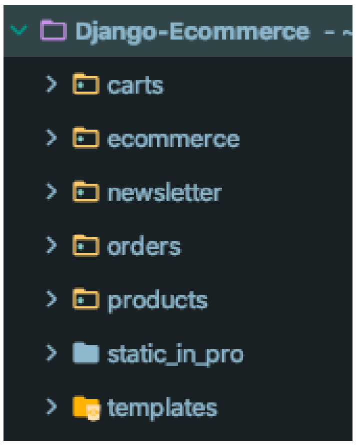


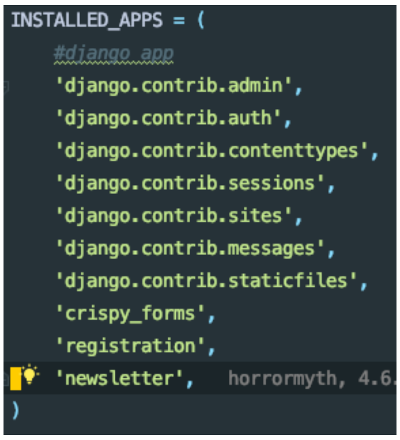
```
urlpatterns = [
    url(r'^(?P<pk>\d+)/inventory/$', VariationListView.as_view(), name='product_inventory'),
]
```

Figure 19 Django URL routing

Blueprints

Blueprint is the application like object that has the information regarding the views, templates and static assets. Django, however, takes a different approach of organizing and managing the application views, templates and assets. Django has a root project, and within that project, many applications can be created as shown below. These applications need to be added to the INSTALLED_APPS list found in the settings file





```
INSTALLED_APPS = (
    #django app
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'crispy_forms',
    'registration',
    'newsletter',    horrormyth, 4.6.
)
```

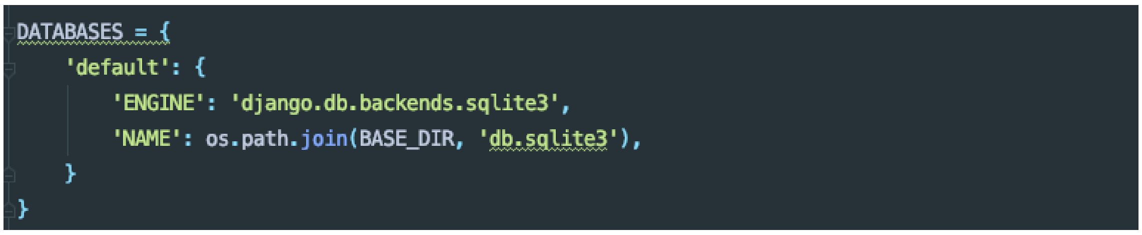
Figure 20 Django project structure

Each application can have its own sets of views, templates and assets. These views are routed with its URL pattern with the main route being configured in the main URL conf file. Therefore, needing of a blueprint in Django is very minimal.

4.2.3 Infrastructure and Configurations

Database

Django supports many variants of the database. Majority of the database backend is already included in the framework. According to the official documentation website [], curPostgresSQL, MariaDB, MYSQL, Oracle and SQLite are supported out of the box. In this project, however, the SQLite database was used with a very minimal relational database model is used. Database host, its backend and the port can be added in the settings file of use. For brevity the project does not use the separate database server, instead it uses the same host as a database server as shown below.



```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Figure 21 Database backend

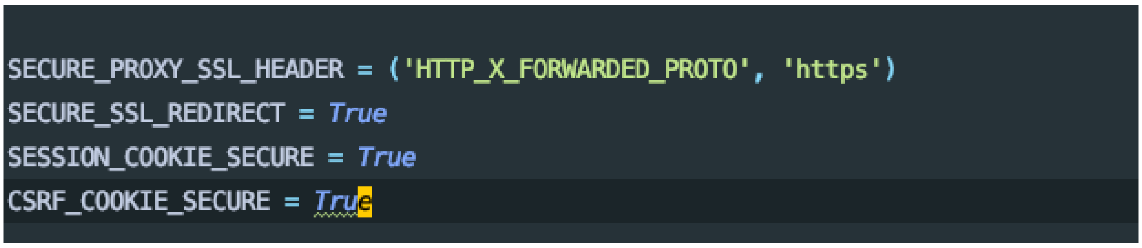
Security

Security is the most important aspects of any web application. It plays a vital role in managing and maintaining the end user's information. There are various types of web security issues and threats lurking around the web. Every day new threats and vulnerabilities can be found. Not all risks are mitigated immediately, nor future threats are known. However, adopting the preventive measures and using the best practices is what a web application framework should vouch for. This is because the users use web application, and the user's confidential information must be intact in the server.

Since Django is an open-sourced project, all the security aspects of the framework are thought out. It covers the following security issues.

- Cross-site request forgery protection (CSRF)
- Cross-site scripting (XSS)
- SQL injection
- Clickjacking
- Host header validation
- SSL/HTTPS

Django's templating system protects the app from the majority of XSS attack by. In the case of CSRF, it has a Csrf Middleware which checks whether the requests referring header is coming from the same origin. Django comes with inbuilt ORM which helps one to query the database without hardcoding the SQL code. These queries and query sets are constructed using query parameterization. By using these standard query sets most of the SQL injections issues are resolved already. In terms of Clickjacking problem, the framework comes with the middleware called X-Frame-Options. Django comes with the possibility to configure the SSL/HTTPS, so that end to end requests and response are encrypted. All of the above configurations can be done, as shown below.



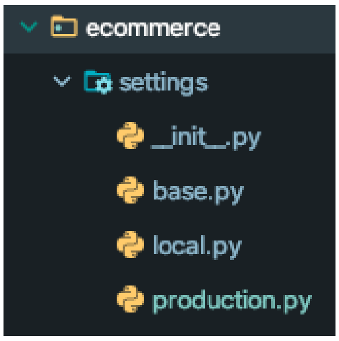
```
SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')
SECURE_SSL_REDIRECT = True
SESSION_COOKIE_SECURE = True
CSRF_COOKIE_SECURE = True
```

Figure 22 Django security configurations

Configurations

Configurations are the heart of the whole application. These contain important information to runt the applications. In Django, it is termed as settings, and these settings can include various details from setting the behaviour, installing application within the framework, middleware's, secrets, database backend, URL, host etc.

Since there are no bulletproof rules to use the configuration, Django provides the freedom on how to add the settings as long as the main configuration is configured in the WSGI file. The e-commerce project, however, has a three configurations file one being the base file where the standard settings sit. Separate configuration file was defined for the local development environment and production environment, as shown bel



```
ecommerce
  settings
    __init__.py
    base.py
    local.py
    production.py
```

Figure 23 Configurations

The rationale behind separating the configuration file is to separate it based on the purpose and the environment. It is better to use the separate configurations for each environment because each configuration file has different sets of configuration that is only required for each environment. This will isolate the configurations and help not leak the secrets and sensitive information such as production database host, URL, api keys e.t.c.

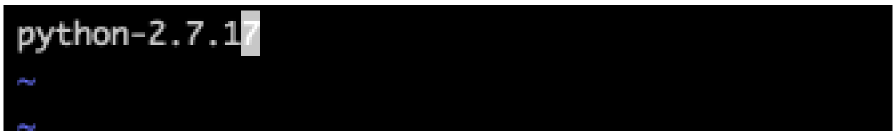
Deployment

Django requires the configuration file for all of its settings. This setting file should also contain the required deployment configuration. The e-commerce project built for this study utilizes the configuration file for the deployment. It was deployed by creating a separate application in the Heroku.

Following mandatory settings were configured in the settings file for the production

1. SECRET_KEY,
2. ALLOWED_HOSTS
3. DEBUG,
4. CACHES
5. ALLOWED_HOSTS
6. DATABASES
7. STATIC_ROOT and STATIC_URL
8. MEDIA_ROOT and MEDIA_URL

Apart from the above settings, few were set in the ".env" files which were added as environment variables in the Heroku as shown in the figure below. Since the application runs on Python 2.7, the version was specified on a file called "runtime.txt" as shown below



```
python-2.7.17
~
~
```

Figure 24 Configure python version Heroku

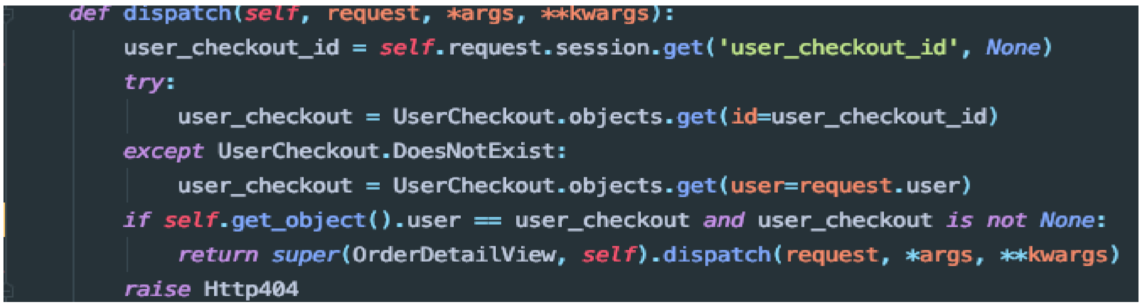
4.2.4 Error handling

Django's approach to application error handling is the same as Python's way of error handling. This includes using Pythons exception. Django comes with many Exception classes which are quite handy when handling the erroneous actions in the code. When needed, these exceptions can be thrown and handled accordingly.

Django's exceptions are divided as follows:

1. Django Core Exceptions - Mostly includes core exceptions such as
    a. AppRegistryNotReady,
    b. ObjectDoesNotExist
    c. EmptyResultSet
    d. FieldDoesNotExist
    e. MultipleObjectsReturned
    f. SuspiciousOperation
    g. PermissionDenied
    h. ViewDoesNotExist
    i. MiddlewareNotUsed
    j. ImproperlyConfigured
    k. FieldError
    l. ValidationError
    m. RequestAborted
    n. SynchronousOnlyOperation
2. URL Resolver Exceptions
    a. Resolver404
    b. NonReverseMatch
3. Database Exceptions
4. HttpExceptions
    a. UnreadablePostError
5. Transaction Exceptions
    a. TransactionManagementError
6. Testing Framework Exception

In this project, Django provided exceptions, as well as Python's built-in core exceptions, are used as shown in Figure 24 below.



```
def dispatch(self, request, *args, **kwargs):
    user_checkout_id = self.request.session.get('user_checkout_id', None)
    try:
        user_checkout = UserCheckout.objects.get(id=user_checkout_id)
    except UserCheckout.DoesNotExist:
        user_checkout = UserCheckout.objects.get(user=request.user)
    if self.get_object().user == user_checkout and user_checkout is not None:
        return super(OrderDetailView, self).dispatch(request, *args, **kwargs)
    raise Http404
```

Figure 25 Exception Handling

4.2.5 Caching

The user's web requests are sent to the server and server response back with the requested content. If the number of users is very high and the requests have to do the whole cycle for the same content time, and again then, this will hamper the performance of the application as well as the server. Eventually increasing the request-response lifecycle. Any web sites that receive a relatively high number of requests benefits by having a cache system in place.

Django already comes with the support for various kind of caching framework described below.

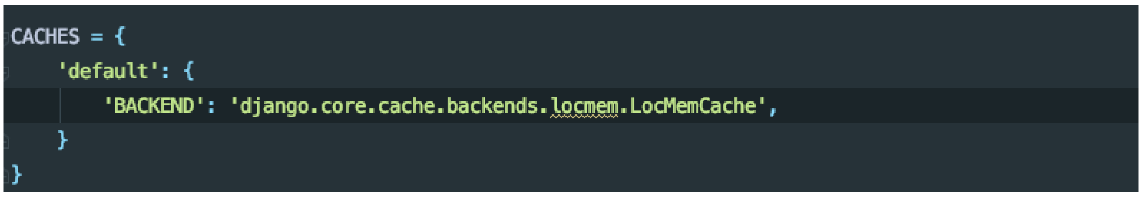
1. Memcached: General purpose distributed in-memory caching system. This is one of the fastest and the preferred caching system for Django

2. Database Caching: As the name suggests the all the cached content are stored in the database table and queried when needed.

3. Filesystem Caching: With this type of caching, the cached values are stored in a separate file.

4. Local-Memory Caching: This is the simple version of the Memcached while functionality being the same. This, however, stores the cached value in the same server where the application runs.

All of the above-caching frameworks can be configured in the setting file with the respective backend. Even though the project did not require any caching framework, it uses Local-Memory Caching. It is configured as shown in Fig



```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
    }
}
```

Figure 26 Django LocalMem Cache

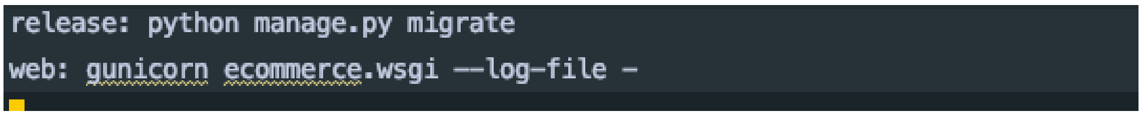
4.2.6 Flexibility

With plenty of features included, it is fair to assert that Django is a battery included framework. This gives developers more room to think about the product and its features within those criteria. These inbuilt supports are mature and thoroughly tested. Since it is an open-sourced project, al the source code is publicly available in the GitHub. Features that are benefited by the developer during the development of an application are as follows.

- Matured ORM
- Multi-site and multi-language support;
- MVC layout
- RSS and Atom feeds
- AJAX support
- URL routing
- Easy Database Migrations;
- Session handling;
- HTTP libraries and templating libraries;
- Code Layout (you can plug new capabilities by using applications);
- Default Admin

However, when it comes to the development flexibility, Django seems to lack behind. This is because developers need to follow Django's way of implementation. At times making a significant change which would have been benefitted by a small change is a bit too cumbersome. This is mainly because not always the technical implementation is support. To overcome this issue developer has to write their own custom Django features for it. Django might not be the right framework for the smaller size web application, because not all its inbuilt features are needed. Especially when deploying a feature, all components get deployed together because it is monolithic.

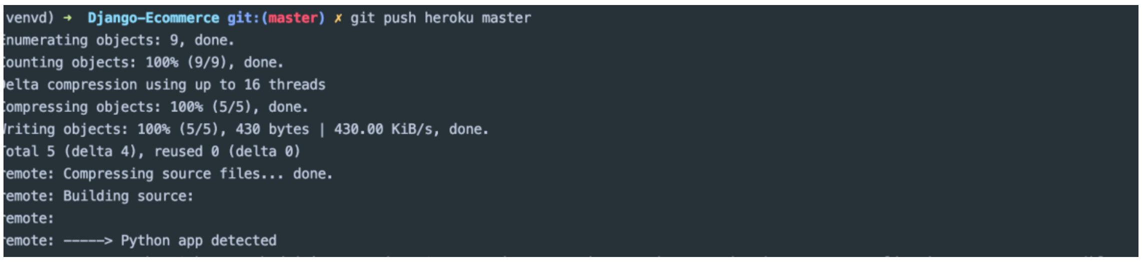
Application path a WSGI configuration for the Heroku was configured in the file called "Procfile". For performing the database migrations, "release:" tag was added in the previously mentioned filename. Above was configured, as shown in the picture below.



```
release: python manage.py migrate
web: gunicorn ecommerce.wsgi --log-file -
```

Figure 27 Django Procfile configuration

Since the Heroku was connected with the Github, the project can be deployed seamlessly deployed. The prerequisite for it to be able to get deployed from local computer is to have the Heroku configured locally and logged in. With that configured, the project can be deployed with a simple one-liner command, as shown in the picture below.



```
venvd) ➜  Django-Ecommerce git:(master) ✗ git push heroku master
numerating objects: 9, done.
ounting objects: 100% (9/9), done.
elta compression using up to 16 threads
ompressing objects: 100% (5/5), done.
riting objects: 100% (5/5), 430 bytes | 430.00 KiB/s, done.
otal 5 (delta 4), reused 0 (delta 0)
emote: Compressing source files... done.
emote: Building source:
emote:
emote: -----> Python app detected
```

Figure 28 Django app deployment to Heroku

4.2.7 Developer Support

Documentation

Django's documentation is one of the most extensive and detailed ones. Developers do not have to read the code to understand how the implementation should be done. It covers from getting started to the final deployment. Not only that, but it also includes the latest rele4ase notes, backwards-incompatible changes, online topics and discussions on development and its scalability.

It also comes with the basic tutorial for the very beginners. Following the instructions in the tutorial one can quickly bootstrap the project in no time. From January 1, 2020, Python software foundation has officially stopped supporting the python version 2.7. Django also has added a set of instructions to help current running projects migrate from Python 2.7 to Python 3.0. The documentation is publicly available at http://docs.djangoproject.com.

Learning Curve

During the implementation of the e-commerce project, the foremost important aspect of the Django was that most of the necessary details are covered in the documentation. Developers do not need to go out of the documentation for the standard implementation. However, it is not always the case when there is a need for a different approach when solving the same problem. For example, if one has to modify the view to work differently than it might require much work. This requires much investigation and reading the source code.

The framework itself comes with wide ranges of features and supports. Hence it is inevitable that documentation will also be vast. Nevertheless, developers who have some experience with Python programming language will likely to learn the framework faster than developers coming from other languages. On the other hand, it has a broader community, and one can get help from the fellow developers very quickly. At the time of writing, there are 227,839 Django tagged questions posted on stackoverflow.com, which is one of the popular platforms for tech-related questions and answers.

5 Conclusion

The goal of the project was to compare the two Python web development frameworks. The comparison was made while developing a simple web application from each of the frameworks. Every framework has its advantages and disadvantages. Both, the app tries to fulfil the same goal with a different approach because of the framework's features and limitations.

Every web application is made to solve problems with a particular business motive. These business motives might come from a small start-up to a large corporation. Usually, products are the facets of the business motive. At times the product needs to be built and shipped and evaluated quickly and vice versa. Depending upon the business decision, and the available resources, one can choose the suitable framework. Both frameworks are matured and production ready. Each has its unique and standard features.

One of the main achievements of this project was that the comparative study helped understand both frameworks. When tested through various approaches to solve the problems, it was found that both frameworks have their advantages and disadvantages. Both frameworks seem to be viable options if they take into use when solving business problems. There are also limitations to the study as not all aspects of the framework have been covered. Based on the study, it is evident that Django can be best fit for large-scale projects with the cost of the learning curve. Flask is best fit for the prototyping and small-scale projects but not limited to it. Flask can be learned and set up quickly, but when it comes to managing and maintaining, it requires more work than the former.

There are many limitations to this study primarily because not all of each framework's features were studied. This type of study requires highly detailed comparison of each and every feature.

6 References

1. Cern. (2019). *A short history of the Web | CERN*. [online] Available at: https://home.cern/science/computing/birth-web/short-history-web [Accessed 26 Oct. 2019].

2. Upwork.com. (2019). *THE FUNDAMENTALS: THE FRONT END VS. THE BACK END*. [online] Available at: https://content-static.upwork.com/blog/uploads/sites/3/2015/05/05110024/Back-end-dev-logo.png [Accessed 26 Oct. 2019].

3. Upwork.com. (2019). *A Beginner's Guide to Front-End Development*. [online] Available at: https://content-static.upwork.com/blog/uploads/sites/3/2015/05/05110037/Front-end-dev1.png [Accessed 27 Oct. 2019].

4. www.oreilly.com. (n.d.). *History of Python - Core Python Programming [Book]*. [online] Available at: https://learning.oreilly.com/library/view/core-python-programming/0130260363/0130260363_ch01lev1sec2.html [Accessed 1 Nov. 2019e].

5. Kuhlman, D. (2011). *A Python Book: Beginning Python, Advanced Python, and Python Exercises*. Platypus Global Media, p.14.

6. Python.org. (n.d.). *PEP 537 -- Python 3.7 Release Schedule*. [online] Available at: https://www.python.org/dev/peps/pep-0537/ [Accessed 1 Nov. 2019].

7. Introduction to the Java Environment (2011). *Java in a Nutshell, 7th Edition*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/java-in-a/9781492037248/ch01.html#idm45941151548920 [Accessed 24 Nov. 2019].

8. Chapter 1 The History and Evolution of Java (2019). *Java: The Complete Reference, Eleventh Edition, 11th Edition*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/java-the-complete/9781260440249/ch01.xhtml#ch01 [Accessed 24 Nov. 2019].

9. Php.net. (2019). *PHP: What is PHP? - Manual*. [online] Available at: https://www.php.net/manual/en/intro-whatis.php [Accessed 4 Nov. 2019].

10. History (2019). *C++ for Lazy Programmers: Quick, Easy, and Fun C++ for Beginners*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/c-for-lazy/9781484251874/html/477913_1_En_25_Chapter.xhtml [Accessed 1 Dec. 2019].

11. The Origins of Go (2019). *The Go Programming Language*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/the-go-programming/9780134190570/ebook_split_005.html [Accessed 2 Nov. 2019].

12. The history of Go (2009). *Mastering Go*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/mastering-go/9781788626545/c3fbaeb6-9974-4bbd-a282-57866c463ce5.xhtml [Accessed 3 Dec. 2019].

13. www.oreilly.com. (n.d.). *1. getting started: A Quick Dip - Head First Kotlin [Book]*. [online] Available at: https://learning.oreilly.com/library/view/head-first-kotlin/9781491996683/ch01.html#welcome_to_kotlinville [Accessed 2 Nov. 2020a].

14. Features of Kotlin (2019). *Kotlin for Enterprise Applications using Java EE*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/kotlin-for-enterprise/9781788997270/187c79f5-f084-4df8-8fe9-659639db6e6a.xhtml [Accessed 4 Nov. 2019].

15. www.oreilly.com. (n.d.). *Writing and Running Ruby Programs - The Ruby Workshop [Book]*. [online] Available at: https://learning.oreilly.com/library/view/the-ruby-workshop/9781838642365/C14197_01_Final_ePub_SZ.xhtml#_idParaDest-9 [Accessed 12 Nov. 2019].

16. More with Less: Code the Way You Want (2019). *Head First Ruby*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/head-first-ruby/9781449372644/ch01.html#ruby_philosophy [Accessed 3 Nov. 2019].

17. 8.2 History of JavaScript (2019). *Web Programming with HTML5, CSS, and JavaScript*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/web-programming-with/9781284091809/xhtml/23_Chapter08_02.xhtml#ch8lev1_2 [Accessed 5 Nov. 2019].

18. Front Matter (2019). *JavaScript Next: Your Complete Guide to the New Features Introduced in JavaScript, Starting from ES6 to ES9*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/javascript-next-your/9781484253946/html/481812_1_En_BookFrontmatter_OnlinePDF.xhtml [Accessed 30 Oct. 2019].

19. W3C (2008). *HTML & CSS - W3C*. [online] W3.org. Available at: https://www.w3.org/standards/webdesign/htmlcss [Accessed 4 Nov. 2019].

20. www.oreilly.com. (n.d.). *Chapter 1. Understanding How the Web Works - HTML, CSS and JavaScript All in One, Sams Teach Yourself: Covering HTML5, CSS3, and jQuery, Second Edition [Book]*. [online] Available at: https://learning.oreilly.com/library/view/html-css-and/9780133795165/ch01.html#ch01lev1sec1 [Accessed 19 Nov. 2019a].

21. W3.org. (2019). *Cascading Style Sheets*. [online] Available at: https://www.w3.org/Style/CSS/Overview.en.html [Accessed 25 Dec. 2019]

22. www.oreilly.com. (n.d.). *MVC and Classes - JavaScript Web Applications [Book]*. [online] Available at: https://learning.oreilly.com/library/view/javascript-web-applications/9781449308216/ch01.html#I_sect11_d1e506 [Accessed 11 May 2020c].

23. www.oreilly.com. (n.d.). *Lesson 4 Understanding JavaScript - Sams Teach Yourself HTML, CSS, and JavaScript All in One, Third Edition [Book]*. [online] Available at: https://learning.oreilly.com/library/view/sams-teach-yourself/9780135167069/ch04.xhtml#ch04lev1sec2 [Accessed 23 Nov. 2019b].

24. *Flask: Building Python Web Services*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/flask-building-python/9781787288225/ch01.html [Accessed 26 Nov. 2019].

25. www.oreilly.com. (n.d.). *1. Installation - Flask Web Development, 2nd Edition [Book]*. [online] Available at: https://learning.oreilly.com/library/view/flask-web-development/9781491991725/ch01.html#ch_install [Accessed 2 Dec. 2019a].

26. RedHat (2018). *Flask*. Available at: https://developers.redhat.com/blog/wp-content/uploads/2018/06/python-flask-logo.png [Accessed 3 Dec. 2019].

27. flask.palletsprojects.com. (n.d.). *Extensions — Flask Documentation (1.1.x)*. [online] Available at: https://flask.palletsprojects.com/en/1.1.x/extensions/ [Accessed 5 Dec. 2019]

28. www.oreilly.com. (n.d.). *Introduction to Django - The Definitive Guide to Django: Web Development Done Right, Second Edition [Book]*. [online] Available at: https://learning.oreilly.com/library/view/the-definitive-guide/9781430219361/ch01.html#django_apostrophy_s_history [Accessed 5 Dec. 2019c].

29. Django.org (2020). [online] Djangoproject.com. Available at: https://static.djangoproject.com/img/logos/django-logo-positive.png [Accessed 12 Dec. 2019].

30. docs.djangoproject.com. (n.d.). *Databases | Django documentation | Django*. [online] Available at: https://docs.djangoproject.com/en/3.0/ref/databases/ [Accessed 7 Dec. 2019].

31. *Python Web Frameworks*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/python-web-frameworks/9781492037873/ch02.html#idm139950201323888 [Accessed 7 Nov. 2019].

32. www.oreilly.com. (n.d.). *18. The Web, Untangled - Introducing Python, 2nd Edition [Book]*. [online] Available at: https://learning.oreilly.com/library/view/introducing-python-2nd/9781492051374/ch18.html#bottle [Accessed 10 May 2020b].

33. zoriana (n.d.). *Bottle, Python framework for building websites*. [online] Quintagroup. Available at: https://quintagroup.com/cms/python/bottle.py [Accessed 20 Dec. 2019].

34. www.oreilly.com. (n.d.). *Some Frameworks to Keep an Eye On - Python Web Frameworks [Book]*. [online] Available at: https://learning.oreilly.com/library/view/python-web-frameworks/9781492037873/ch02.html#idm139950199964416 [Accessed 24 Dec. 2019f].

35. www.oreilly.com. (n.d.). *CherryPy in Depth - CherryPy Essentials [Book]*. [online] Available at: https://learning.oreilly.com/library/view/cherrypy-essentials/9781904811848/ch04.html#ch04lvl2sec25 [Accessed 26 Dec. 2019c].

36. Introduction (2009). *Introduction to Tornado*. [online] O'Reilly | Safari. Available at: https://learning.oreilly.com/library/view/introduction-to-tornado/9781449312787/ch01.html#what-is-tornado [Accessed 6 Dec. 2019].

37. www.oreilly.com. (n.d.). *Asynchronous Web Services - Introduction to Tornado [Book]*. [online] Available at: https://learning.oreilly.com/library/view/Introduction+to+Tornado/9781449312787/ch05.html#asynchronous-web-services [Accessed 30 Dec. 2019b].