

---

# **An Analysis and Comparison of Flask, Django and Pyramid**

---

Kieran Burns - 40272382

Submitted in partial fulfilment of  
the requirements of Edinburgh Napier University  
for the Degree of  
BEng (Hons) Computing

School of Computing

April 2, 2020

### **Authorship Declaration**

I, Kieran James Burns, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

*Signed:*

Handwritten signature of K Burns in black ink.

*Date:* April 2nd 2020

*Matriculation no:* 40272382

### **General Data Protection Regulation Declaration**

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

KBurns

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

### **Abstract**

With the prominence of a wide range of methods of implementing a website, this report aims to analyse and compare three frameworks, Flask, Django and Pyramid, each based in Python, to determine which of them best suits which types of potential user base. Using existing research and development work in each of the three frameworks, as well as a few alternate contemporaries, three separate identical websites were implemented (one using each of the three frameworks), sharing various non-framework resources. The results of this implementation show that each of the three frameworks have their own best-use scenarios, but are not evenly matched on that front, with the potential use-cases of each framework generally mirroring their use percentages discussed within the existing research. The research and development within the project also briefly explores the idea of porting between the three frameworks, attempting to discover if there is a potential upgrade/downgrade path, or alternatively if any resources can be shared between them at all. Due to the unreliability and potential bias, further research is required to develop a more conclusive answer as to which framework is objectively best in each situation, but this project does work as a good means of aiding new or uninformed developers on many of the advantages and disadvantages of each of the frameworks, giving an especially good view at developing with each from the eyes of a user who's new to each of the frameworks in question.

## Contents

<b>1</b>	<b>Introduction</b>	<b>10</b>
1.1	Project Background . . . . .	10
1.2	Research Questions . . . . .	10
1.3	Deliverables . . . . .	10
1.4	Purpose and Demographic . . . . .	11
1.5	Structure of this Report . . . . .	12
<b>2</b>	<b>Literature Review</b>	<b>14</b>
2.1	Defining a Framework . . . . .	14
2.2	Popular Non-Python Technologies . . . . .	15
2.2.1	AJAX . . . . .	15
2.2.2	PHP . . . . .	16
2.2.3	Ruby on Rails (Rails) . . . . .	16
2.2.4	jQuery . . . . .	16
2.2.5	Zend . . . . .	17
2.2.6	ASP.NET . . . . .	17
2.2.7	Perl . . . . .	18
2.3	Popular Python Web Frameworks . . . . .	18
2.3.1	Django . . . . .	19
2.3.2	Flask . . . . .	19
2.3.3	Tornado . . . . .	19
2.3.4	Bottle . . . . .	20
2.3.5	Pyramid . . . . .	20
2.3.6	CherryPy . . . . .	20
2.4	Why Only Flask, Pyramid & Django? . . . . .	21
2.5	Exploring Flask in Further Detail . . . . .	22
2.6	Exploring Django in Further Detail . . . . .	25
2.7	Exploring Pyramid in Further Detail . . . . .	28
2.8	The Upgrade Path . . . . .	30
<b>3</b>	<b>Comparison Criteria and Methodologies</b>	<b>34</b>
3.1	Getting Started/The Basics . . . . .	34
3.2	Using Traditional Web Design Aspects . . . . .	34
3.3	Basic-to-Intermediate Framework Features . . . . .	35
3.4	Database Interactivity and Data Security . . . . .	36
<b>4</b>	<b>Implementation</b>	<b>38</b>
4.1	Websites Using Each of the Frameworks . . . . .	38
4.1.1	Specification . . . . .	38

4.1.2	Tools Used . . . . .	42
4.1.3	Shared Elements . . . . .	43
4.1.4	Flask Approach . . . . .	44
4.1.5	Django Approach . . . . .	46
4.1.6	Pyramid Approach . . . . .	48
4.2	Comparing the Implemented Sites . . . . .	52
<b>5</b>	<b>Results and Evaluation</b>	<b>54</b>
5.1	Objective Comparisons Made . . . . .	54
5.1.1	Project Startup and The Basics . . . . .	54
5.1.2	Utilising Traditional Web Development Aspects . . . . .	54
5.1.3	Routing and Requests . . . . .	55
5.1.4	Database Interactivity and Security . . . . .	55
5.1.5	Shared Resources and Porting Potential . . . . .	56
5.2	Personal Reflection on the Project . . . . .	57
5.2.1	Positive Aspects of My Implementation . . . . .	57
5.2.2	Negative Aspects of My Implementation . . . . .	57
5.2.3	What Would I Have Done Differently? . . . . .	58
5.3	Recommendations based on the Research and Results . . . . .	59
5.3.1	Flask . . . . .	59
5.3.2	Django . . . . .	59
5.3.3	Pyramid . . . . .	60
5.4	Research Question Answer Summary . . . . .	61
<b>6</b>	<b>Conclusion and Future Work</b>	<b>62</b>
	<b>Appendices</b>	<b>67</b>
<b>A</b>	<b>Initial Project Overview</b>	<b>67</b>
A.A	Project Title . . . . .	67
A.B	Overview of Project Content and Milestones . . . . .	67
A.C	Main Deliverables . . . . .	67
A.D	Target Audience for the Deliverables . . . . .	68
A.E	The Work to be Undertaken . . . . .	68
A.F	Additional Information/Knowledge Required . . . . .	68
A.G	Information Sources that Provide a Context for the Project . . . . .	68
A.H	Importance of the Project . . . . .	68
A.I	The Key Challenges to Overcome . . . . .	69
<b>B</b>	<b>Second Formal Review Output</b>	<b>70</b>

<b>C</b>	<b>Weekly Supervisor Meeting Notes</b>	<b>72</b>
C.A	Trimester 1, Week 1 (20/09/2019) . . . . .	72
C.B	Trimester 1, Week 2 (27/09/2019) . . . . .	72
C.C	Trimester 1, Week 3 (04/10/2019) . . . . .	73
C.D	Trimester 1, Week 4 (11/10/2019) . . . . .	73
C.E	Trimester 1, Week 5 (16/10/2019) . . . . .	73
C.F	Trimester 1, Week 6 (23/10/2019) . . . . .	73
C.G	Trimester 1, Week 7 (No Meeting) . . . . .	74
C.H	Trimester 1, Week 8 (06/11/2019) . . . . .	74
C.I	Trimester 1, Week 9 (13/11/2019) . . . . .	74
C.J	Trimester 1, Week 10 (20/11/2019) . . . . .	74
C.K	Trimester 1, Week 11 (29/11/2019) [Interim Review] . . . . .	75
C.L	Trimester 1, Week 12 (04/12/2019) . . . . .	75
C.M	Tr1 Exams and Winter Break . . . . .	75
C.N	Trimester 2, Week 1 (No Meeting) . . . . .	76
C.O	Trimester 2, Week 2 (24/01/2020) . . . . .	76
C.P	Trimester 2, Week 3 (No Meeting) . . . . .	76
C.Q	Trimester 2, Week 4 (06/02/2020) . . . . .	76
C.R	Trimester 2, Week 5 (13/02/2020) . . . . .	76
C.S	Trimester 2, Week 6 (21/02/2020) . . . . .	76
C.T	Trimester 2, Week 7 (27/02/2020) . . . . .	76
C.U	Trimester 2, Week 8 (05/03/2020) . . . . .	77
C.V	Trimester 2, Week 9 (No Meeting) . . . . .	77
C.W	Trimester 2, Week 10 (No Meeting) . . . . .	77
C.X	Trimester 2, Week 11 (26/03/2020) . . . . .	77
<b>D</b>	<b>Project Timeline Gantt Chart</b>	<b>78</b>
<b>E</b>	<b>Implemented Website Screenshots</b>	<b>79</b>

**List of Tables**

1	Comparisons and Metrics . . . . .	53
---	-----------------------------------	----



**List of Figures**

1	Python Web Framework usage statistics, JetBrains (2019) . . .	21
2	Flask project file structure, Pallets (2010) . . . . .	32
3	Pyramid project file structure, Pylons Project (2019) . . . . .	33
4	Django project file structure, Mischback (2018) . . . . .	33
5	Main Page Wireframe Diagram . . . . .	38
6	Search Page Wireframe Diagram . . . . .	39
7	Sign Up Page Wireframe Diagram . . . . .	39
8	Login Page Wireframe Diagram . . . . .	40
9	New Listing Wireframe Diagram . . . . .	40
10	Navigation Bar Wireframe Diagram . . . . .	41
11	Profile page Wireframe Diagram . . . . .	42
12	Home Page while not logged in . . . . .	79
13	Signing up as a new user . . . . .	80
14	Signup validation . . . . .	81
15	Logging in . . . . .	82
16	Login validation . . . . .	83
17	Home Page when logged in . . . . .	83
18	Searching for all games with “dark” in the title . . . . .	84
19	Creating a new listing for a game . . . . .	85
20	Validation on creating a new listing . . . . .	86
21	Deleting a listing owned by the user . . . . .	87
22	Editing a listing owned by the user (uses the same validation as in <b>figure 20</b> ) . . . . .	88
23	My Listings page, which only displays listings created by the logged in user . . . . .	88
24	Clicking the Purchase button on another user’s listing . . . . .	89
25	Manually entering an invalid URL (getting an error-404) . . . . .	89
26	Logged in user’s account page . . . . .	90
27	Confirm page after clicking “delete account” . . . . .	90
28	Validation on Change Password option . . . . .	91

**Acknowledgements**

I extend my thanks to all of my Family, Friends and the University Staff for all the encouragement and support they have given me during the duration of the project.

A special thanks to my project supervisor Peter Barclay for all of the help and advice throughout the project.

A special thanks also to my Uncle, Robert Burns for proof reading and critiquing the full written body of my report to help me make my writing the best I could.

## 1 Introduction

### 1.1 Project Background

According to various sources, over recent years Python has become one of the most popular and “in demand” programming languages on the market and is still rapidly growing (See: GitHub (2018); Cass, S. (2019); TIOBE (2019)). This is likely due to Python’s utility in various types of development, ranging from traditional applications development, to mobile development, to web development. This project focuses on the web development aspect of the language, exploring the various frameworks which can be employed in creating web applications of differing scopes and scales.

### 1.2 Research Questions

The overall aim of this project is to establish the practical utility of each of the three key frameworks in question by means of comparative analysis. In doing this, research questions will be answered to coincide with the purpose and demographic of the project as a means of elaborating on the most useful information to potential readers. In particular, the main research questions are:

- What key advantages and disadvantages arise from using a Microframework such as Flask over a Megaframework such as Django, and vice versa?
- Are there situations where a “Goldilocks” (flexible mid-sized) framework such as Pyramid could and should be used over a Microframework or Megaframework?
- Is there a clear path allowing users to port existing Web Applications between any of the three frameworks in question? (E.g. Port an existing Flask application to Django with minimum effort or challenge).

### 1.3 Deliverables

The first of the deliverables is a set of test criteria to which each of the three frameworks in question are compared using. This is to prove the methods behind the implementation, and to allow other people to replicate and modify the testing process if they wish to test frameworks in a similar manner themselves.

The second of the deliverables is three identical implemented websites (one using each framework), based upon the test criteria. These will be used to prove or disprove the theories and ideas behind the test criteria, as a means

of applying the concepts explored within the proceeding Literature Review.

The last of the deliverables is a means of comparing the three frameworks using a series of metrics and comparisons made from my experiences developing with each of the three frameworks. These can be used by a developer in helping to discern which of the three frameworks may best fit their own Web Development project. The key purposes of this are to function as both a means of collating the results of the implementation a whole, as well as function as an abbreviated and simplified version of the results stage of this project.

#### **1.4 Purpose and Demographic**

The key purpose of this project is to explore the potential uses of each of the three web frameworks and develop web applications using each of them. An attempt to answer each of the Research Questions will also be made during the process. This purpose serves in aid of satisfying the target demographic of this project, which can be split in to three main groups:

Firstly, there are those who are new to Web Development, or new to Python Web Frameworks, who are looking to find which of the three frameworks is the best to start with, based on a project they have in mind. This group will be catered to the most during the implementation stage of the project, as many of the criteria of comparison for the three frameworks will cover ground that experienced users of any of the frameworks may already hold as “common knowledge” (or something to a similar extent). This is because for newer users, clear documentation is especially important in aid of grasping key concepts and vital techniques.

The Second of the three groups are those with moderate experience with Python Web Development, who are interested in the unique, useful or generally interesting features of each of the frameworks. This group will be catered to in a lesser extent than those of the first group, but still to enough of a degree to make some aspects of the implementation worthwhile reading about. This is because some of the comparison criteria will explore features which are unique or much more heavily stacked in the favour of certain frameworks over the others as part of discovering which frameworks best suit certain specific user groups.

The Last of the three groups is those who may be interested in the idea of porting between the frameworks. This group might contain developers

who are especially experienced in at least one of the frameworks and are looking to branch out to another of them, or potentially developers who are simply curious to learn whether porting applications between the frameworks is possible.

### 1.5 Structure of this Report

This report is divided into six chapters, each with their own individual sub-sections, where each chapter presents and explores a different stage of the project as a whole.

**Chapter 1** introduces the project from a broad perspective and outlines the themes and ideas that will be explored throughout the duration of the rest of the project. The chapter also explores the target demographic of the report and the deliverables that are hoped to be achieved.

**Chapter 2** is a literature review, used to explore the themes and ideas discussed in **Chapter 1** through the means of existing sources. It aims to clearly define what a framework is in context of computing, highlight some popular Python web development frameworks, explain why the three chosen frameworks of Flask, Django and Pyramid are being explored, alongside the utility and capacity of each of the frameworks, as well as briefly exploring existing materials on the topic of the “upgrade path”.

**Chapter 3** aims to collate and discuss different methodologies and criteria to which the frameworks can be directly compared against each other, exploring their use in further understanding the utility and capacities of each of the frameworks in a fair and impartial fashion.

**Chapter 4** is dedicated to the project’s implementation, discussing the design and compromises made throughout. This Chapter will also present a set of criteria and metrics to which each of the sites are compared on, based on the information collated in **Chapter 3**, alongside any unconsidered findings from the implemented sites.

**Chapter 5** discusses the results of the implementation stage compared to the findings of the literature review, discussing each of the research questions. It also evaluates the successes and shortcomings of the implementation as a whole.

Lastly, **Chapter 6** serves as a conclusion to the project, summarising the

overall outcomes and discussing possible work towards the project that could be done in the future.

## 2 Literature Review

### 2.1 Defining a Framework

In the real world, a framework is often spoken of in the context of construction as an underlying structure used to support a building. In computing this concept is mirrored by software frameworks, which support applications in similar ways to how construction frameworks support the structural integrity of buildings.

Christensson, P. (2013) defines a computing framework as “a platform for developing software applications. It provides a foundation on which software developers can build programs for a specific platform.”. This definition reinforces the idea of using a software framework to lay the groundwork for aspects of a system or application, covering certain necessities, making the task of developing the system or application simpler for the developer or team who are using the framework. This can then pose the question, what exactly does a framework do? Christensson goes on to further explain “a framework may include predefined classes and functions that can be used to process input, manage hardware devices, and interact with system software.”. This breaks down a framework to be an all-encompassing (umbrella) term, used to define supporting code created by another developer to be used to assist in completing a task. However, this broad definition then begins to clash with a separate computing concept, known as a library.

Wozniwicz, B. (2019) highlights the differences between a framework and library. Firstly, defining a library as follows; “A library is like going to Ikea. You already have a home, but you need a bit of help with furniture. You don’t feel like making your own table from scratch. Ikea allows you to pick and choose different things to go in your home. You are in control.”. Following the definition of a library, Wozniwicz defines a framework as so; “A framework, on the other hand, is like building a model home. You have a set of blueprints and a few limited choices when it comes to architecture and design. Ultimately, the contractor and blueprint are in control. And they will let you know when and where you can provide your input.” These analogies provide key insights to the differences between the two. The differences can be boiled down to where a framework is an existing piece of software, which is added on to, whereas a library is used as snippets of software, used in aid of the construction of other pieces of software. To that extent, the two are almost the opposite of one another, clearing up the possible confusions between the two.

## 2.2 Popular Non-Python Technologies

Although not utilised as key element of this project, a vast selection of alternative methods for the implementation of web applications which do not utilise Python are available. Many of these methods predate modern Python web development by a notable window or have much wider reaching influence and/or history. Due to the influence many of these technologies have on certain design decisions within the frameworks to which this report focuses on, the technologies are deserving of some exploration as a means of providing context to some of the history of the frameworks.

Forgoing the obvious general-use, commonplace technologies such as “HTML”, “CSS” and “JavaScript”, which are each utilised, to some extent, in most higher-end web applications by necessity, Li, Das & Dowe (2014) briefly touch upon a few useful technologies for web application development, listing “AJAX” as a key client-side technology, and “PHP” and “Ruby on Rails” alongside Python as server-side technologies. Alongside the technologies noted by Li, Das & Dowe, Kohan, B. (n.d.) also notes a sizeable series of other technologies, most notably “jQuery” for the client-side, and “Zend”, “ASP.NET” and “PERL” for the server-side. It should be worth noting that the utilities provided by each of these technologies do not directly intersect with Python, nor with each other, instead overlapping in certain key areas. To prove this, each of these technologies should be briefly explored.

### 2.2.1 AJAX

Asynchronous JavaScript And XML (AJAX) is somewhat of an outlier compared to the other technologies explored within this section, as it’s more accurately described as a technique rather than a whole technology, being a method of a web page communicating with a server. As a result of this, there is no official website to “advertise” or “sell” AJAX, and instead external sources must be relied upon to properly understand the purpose and demographic of AJAX. Woychowsky, E. (2006) describes the philosophy of AJAX as “not using bandwidth just because it’s there”, describing the efficient approach AJAX takes when handling operations. Woychowsky goes on to further explain this is because AJAX is reluctant to unload and reload already loaded parts of a web page, which may appear to cause “strange behaviour” within a web page when attempting to update sections of the page without reloading the whole page. In essence, AJAX could be summed up as being an efficient method of client-server communications. This overlaps with the client-server communications seen within Python web frameworks where templates are used.



### 2.2.2 PHP

The official website for PHP (The PHP Group, 2020) describes it as “a popular general-purpose scripting language that is especially suited to web development.” as well as “Fast, flexible and pragmatic”. Further within the site the documentation section can be found, containing an apt description of the capabilities of PHP. Here it is explained that the “main target field” of PHP is Server-side scripting, used to deploy HTML pages with interactive elements using PHP tags within standard HTML code. This feature is similar to the templates used within Python web frameworks to, some extent. Another key feature advertised about PHP is the support for a “wide range of databases”. This range includes most common SQL databases such as MySQL and SQLite, as well as some NoSQL databases such as MongoDB and CouchDB. This replicates openness and capability of Python web frameworks such as Django and Pyramid to run a wide range of different database types to suit any given developer’s needs.

### 2.2.3 Ruby on Rails (Rails)

The official documentation for Rails (Hansson, D. H., n.d.) describes it as a framework written in the programming language Ruby (hence Ruby on Rails). The purpose of Rails is described as being “designed to make programming web applications easier by making assumptions about what every developer needs to get started.”. This is further explained as being the result of the opinionated nature of Rails, meaning there is a specifically “best” or “intended” way to tackle each task and problem. Hence, the framework forgoes modifiability and configurability for the sake of a more streamlined development experience. This goes in the face of some notable Python web frameworks, especially Pyramid, as explained within its documentation (Pylons Project, 2019), which tend to prefer a vast array of possible options to allow the development experience to be as user-oriented as possible, compared to the convention-oriented nature of Rails, which tends to be more friendly to newer users. Outside of this, Rails takes similar approaches to many Python web frameworks in various aspects of development such as in routing and templates.

### 2.2.4 jQuery

Much like AJAX, jQuery is somewhat of an outlier among the other technologies explored within this section. This is because jQuery is a JavaScript library. The reason jQuery is worth note is because it is a means of making

JavaScript much easier to utilise in a wide range of different types of website, simplifying some of the more complicated elements of JavaScript such as the aforementioned AJAX technique alongside other JavaScript staples such as DOM and Event Handling. These improvements are highlighted on the official website of jQuery (The jQuery Foundation, 2020) where the small size, high speed, and feature rich nature of the library are highlighted. Although somewhat detached from what Python web frameworks provide to web developers, there are some small areas of overlap between Python web frameworks and jQuery. The most notable of the areas of overlap is that of event handling. The key difference however is that while Python frameworks tend to rely on an interaction between the client and server to handle the dynamic elements, which would be found within the template of the current page, jQuery does all event handling locally on the client-side.

### **2.2.5 Zend**

Zend is a framework consisting of a collection of PHP packages, to be used as an extension of PHP to aid in the development of web applications. The official website for Zend (Zend, 2020) explains one of the key use cases of the framework being as a “component library”, containing a wide range of “loosely coupled” elements, which can each be utilised independently of each other. Although this functionality is practical and useful for developers using Zend, the key selling point is the Model-View-Controller (MVC) approach it can utilise in PHP, something which is not an innate capability of the language. This is the key overlap between Zend and many of the Python frameworks in question, such as Flask, since both take the MVC approach to development.

### **2.2.6 ASP.NET**

ASP.NET is Microsoft’s take on web applications development. It is a progression of an older technology also made by Microsoft, simply known as ASP or Classic ASP. The key difference is that ASP.NET was made for C#, Microsoft’s current flagship programming language, instead of VBScript which was used for the original ASP. The official site for ASP.NET (Microsoft, 2020) explains that it contains all the tools and libraries a developer could require for all their web applications needs. This includes the base framework to allow features such as request handling, a templating language known as Razor to allow dynamic web pages, libraries for common web patterns such as MVC, and authentication systems which include multi-factor and external authentication. As well as these features added to C# by ASP.NET, all the base

C# capabilities are also available, including database handling. This means that ASP.NET has a very large area of overlap with the Python web frameworks in question, as well as with Ruby on Rails. Although there are some small differences between ASP.NET and Python frameworks such as Django, the only key differences are in environment and base language, meaning the choice in whether to use ASP.NET or a Python-based web framework is primarily rooted in the developer's preference, experience, work environment and whether they would prefer to use open source or proprietary tools.

### **2.2.7 Perl**

Perl is a programming language with facilities to incorporate web applications development. The official site for the language (Perl.org, 2020) discusses the aims of Perl as a language, being the intention of the language being “practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal)”. Much like most of the Python web frameworks in question, Perl is also unopinionated, allowing developers a much wider range of options than they may have otherwise. Perl in fact has a lot of overlap with Python frameworks such as Django, performing very similar actions. Myhrvold, C. (2014) explores this, explaining why Perl has gradually been losing its user-base over time, due to the stagnation the language has faced when compared to its closest rivals such as Python's Django framework and Ruby on Rails, as these frameworks have evolved and grown while Perl has lacked quite an expansive evolution. Myhrvold goes on to further explain that Perl is not absolutely obsolete in the present day due to its inclusion in many legacy systems and usage within certain circles, however summarises that Perl in its current state doesn't offer much of a distinct advantage over other Python alternatives, often just leaving a feeling of frustration for the user.

## **2.3 Popular Python Web Frameworks**

Due to the popularity of Python in web development, and the wide range of various capabilities and scopes that different developers may look for when choosing a framework, a sizeable sum of useful frameworks have been made. Guardia, C. (2016) lists the six most popular Python web frameworks (“ones to watch”) as; Django, Flask, Tornado, Bottle, Pyramid and CherryPy. To fairly compare each of the frameworks, a moderate understanding of each is necessary. To achieve this, each must be analysed individually before being compared.

### 2.3.1 Django

Django is a full-stack web framework, often referred to as a “Megaframework”. Its three key traits, as advertised on the official website (Django Software Foundation, 2019), are its low development time requirements, security aspects and scalability. The official site advertises the low development time requirements as “Ridiculously Fast”; dubs the security elements as “Reassuringly Secure” and claims the scalability as “Exceedingly Scalable”. The official site also focuses on Django’s fully loaded and versatile nature, covering a vast amount of possible applications of the framework in many different scenarios, aiming to prove that it deserves its title as a “Megaframework”.

### 2.3.2 Flask

Flask is a simple lightweight web framework which focuses primarily on simplicity and ease of access, earning the title of “Microframework” due to its barebones compact nature. The official community page for Flask (Ronacher, A., 2015), within “The Pallets Projects” website (which handles the maintenance and development of Flask, alongside a selection of other frameworks and libraries), advertises Flask as being “designed to make getting started quick and easy”. The advertised properties serve to enforce the idea of Flask being ideal for both beginners and in use when undergoing small tasks, as well as perhaps having some utility in larger projects, though more work is probably required when compared with more “fully-developed” frameworks.

### 2.3.3 Tornado

Tornado is a specialised web framework, with a focus on holding long-lived connections to large amounts of concurrent users. Dory, Parrish & Berg (2012) describe Tornado as being something they had “all fallen in love with” due to three key advantages of the framework: speed, simplicity, and scalability. They also describe the framework as being “robust enough to handle serious web traffic, yet is lightweight to set up and write for, and can be used for a variety of applications and utilities.”. Despite this acclaim however, Tornado is still very much a specialised framework, mainly relying on its specialist features of an “asynchronous networking library” and “non-blocking network I/O”, as advertised on the official site (The Tornado Authors, n.d.), to stand out amongst the vast amount of other Python web frameworks.

### 2.3.4 Bottle

Bottle is another simple lightweight web framework, given the title of “Microframework”. Makai, M. (2019a) lists the three key uses of bottle as: “Prototyping Ideas”, “Learning how web frameworks are built” and “Building and running simple personal web applications”, each inferring the small scale and simplistic nature of the framework. The official website for Bottle (Hellkamp, M., 2019) reinforces this idea, primarily advertising the “simple and lightweight” nature of Bottle. The site also highlights that Bottle is “distributed as a single file module and has no dependencies other than the Python Standard Library.”, furthering the idea of the framework being very basic when compared to its contemporaries.

### 2.3.5 Pyramid

Pyramid is a medium-sized web framework with a focus on being as flexible and accessible as possible. The official Pyramid website (Pylons Project, 2019) dubs the frameworks as being the “Goldilocks Solution” to web frameworks, equating the framework to a popular fairy tale, in the sense that the framework is not too simple, nor too complex, being “just right” in terms of scope and scale. The website goes on to compare the framework as being the ideal median between a “Megaframework” and a “Microframework”, as well as explaining the tagline “Start Small, Finish Big, Stay Finished”. This focusses on the easy and fast options when first starting a project, while still retaining high-end development possibilities, as well as aiming to retain the relevance of older code written using the framework. Thus, this keeps older applications from needing major maintenance as the framework evolves and grows.

### 2.3.6 CherryPy

CherryPy is yet another simple lightweight “Microframework”. The official website for the framework (The CherryPy team, 2018) coins the framework as “A Minimalist Python Web Framework” to enforce this. The key talking point of the framework is its “smaller source code developed in less time.”. This is due to the strong fundamentals the framework has built up since its initial release in 2002, making it the oldest framework within the six in this section. The framework boasts its maturity as a framework, having had “many final, stable releases.” and being “proven reliable for real-world use” as stated on the official site.

## 2.4 Why Only Flask, Pyramid & Django?

With a solid background on the listed six most popular frameworks explored earlier in this report, the question of why this project only spans Flask, Django and Pyramid is likely to be asked.

The project spanning Flask and Django is much more obvious than that of the other four, due to the landslide popularity of Flask and Django over all other Python web frameworks. The Python Developers Survey (JetBrains, 2019), which covers 20,000 developers from over 150 different countries, containing a vast amount of different statistics around Python as a whole, has a poll which shows the percentage of developers who use various web frameworks. These statistics can be seen in **figure 1**, where each of the six frameworks covered within the previous section can be seen, alongside a few others.

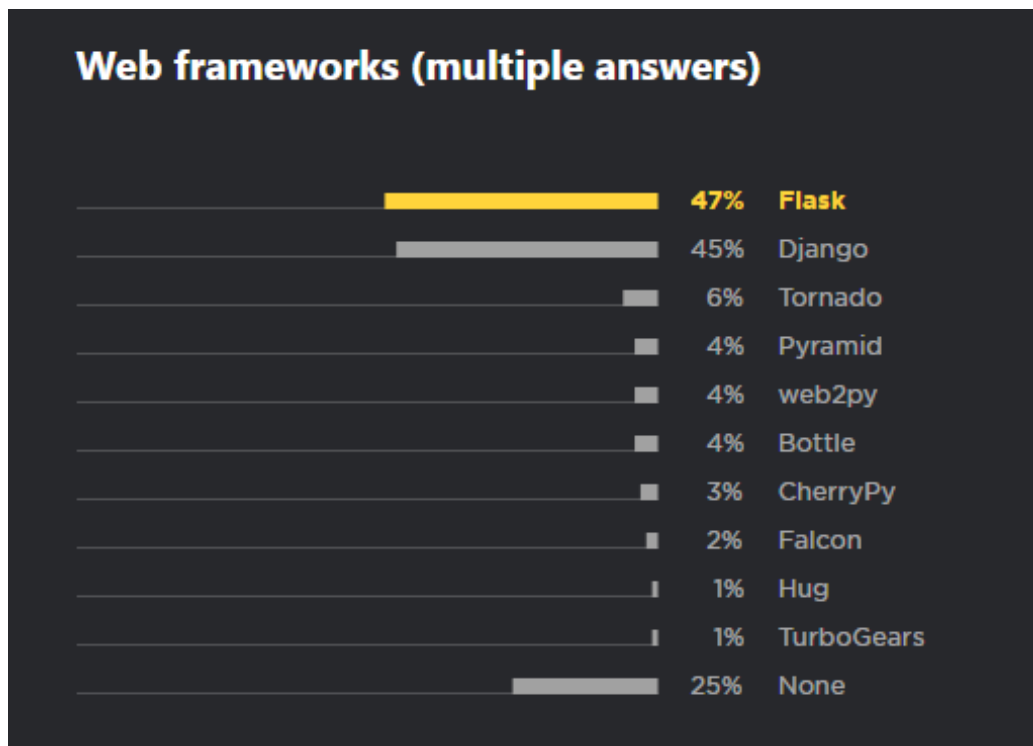


Figure 1: Python Web Framework usage statistics, JetBrains (2019)

These statistics help in proving the predominance of Flask and Django over the others, aiding in the justification in analysing and utilising both frameworks in the scope of this project.

Both Bottle and CherryPy fit roles similar to Flask, for the most part on a surface level, making them interchangeable to some extent. However, since Flask has a vastly larger userbase than the other two (as seen in **figure 1**), it can be inferred that Flask has much more actively available support, both in the form of examples which can be inspected, and from other developers which may be accessible in places such as help forums.

Since Pyramid aims to fit a role bridging the gap between Microframeworks and Megaframeworks (as discussed in the prior section) it is an interesting test subject, to decide if it fits its advertised role in a useful and meaningful way, making it worth exploring within the scope of this project.

The specialised nature of Tornado makes it a poor fit for the scope of this project as it aims to explore a much more general perspective in respect of the application of each of the frameworks.

For to each of these reasons, the project's scope is limited to Flask, Pyramid and Django. Each of these three frameworks are reviewed in the succeeding sections below.

## 2.5 Exploring Flask in Further Detail

It's difficult to dispute that Flask is a popular framework, and as with anything, popularity tends to be for a reason. In the case of Flask, there are a number of aspects to the framework, which when combined, make up its popularity.

The first aspect worth exploring is Flask's simplicity. As explored within the Popular Python Web Frameworks section of this report, one of Flask's key selling points is how "quick and easy" it makes the early stages of a new project. Makai, M. (2019b) attempts to explain why this is the case, comparing the framework to the likes of Django;

"Flask is considered more Pythonic than the Django web framework because in common situations the equivalent Flask web application is more explicit. Flask is also easy to get started with as a beginner because there is little boilerplate code for getting a simple app up and running."

Makai uses the idiom "Pythonic" to refer to how close web applications written using Flask are to the most traditional forms of Python programming. Comparing this to Django, which is a relatively less traditional framework,

more detached from Python, Makai argues that this is a strong means of presenting the idea of familiarity with regard to Flask. This idea of familiarity is especially useful in generating interest in Flask due to the nature of Python itself, having many close similarities to other object-oriented programming languages such as Java and C++, meaning users with any level of experience using such languages will likely find starting with Flask to be much more favourable over Flask's contemporaries, such as Django. Makai's second point, speaking about a lack of much "boilerplate code" is also a positive sign regarding the simplicity of Flask. In programming, boilerplate code is code that is required in specific places of a program for the sake of getting "something" to work as intended. In the case of web frameworks, boilerplate code is used when setting up essential methods to allow the framework to actually function. In Flask only around five lines of boilerplate code is required to allow the most basic "Hello world" application to run. Comparing this to Django, which on creation of a new web application generates a series of over 10 ".py" files (Python files), each with their own series of several lines of boilerplate code, really shows how little is required for a Flask app, comparatively.

The second aspect of Flask worth exploring is the framework's flexibility of usage. Due to its nature as a Microframework, Flask comes packaged with very little in terms of "extra features", focusing more on the bare minimum to allow a basic web application to function. This, however, is not a huge issue due to the ability to easily add extensions. Maia, I. (2015) discusses this in some detail:

"Flask does not come with bundled functionality in terms of database integration, a forms library, administration interface, or migration tools. You can have these through extensions..., but they are all external to Flask."

Here, Maia explains the idea of using extensions to add features to Flask but warns that all extensions are entirely external to the framework, which means they are not officially managed and updated by the development team who work on Flask. This means that some of the extensions may not be as reliable as developers may hope. Maia then goes on to analyse some potential positive uses for Flask's bareness, one of the most positive being:

"What if you need a specific set of libraries to work together in your project and you don't want the web framework getting in the way; that's another very good scenario for Flask as it gives you the bare minimum and lets you put together everything else you may need."



Maia suggests the barebones nature of Flask is very positive when talking about conflicts using a variety of different libraries simultaneously, as the framework is likely to be compact and flexible enough not to disrupt the libraries, unlike other, larger contemporaries of Flask.

The third aspect attributed to the popularity of Flask is the community around the framework. Due to the framework being popular, therefore having more support from members of its community, it draws more users, which in turn increases the framework's popularity. However, this attribute can only function efficiently if the framework is also worthwhile to utilise, allowing it to build its initial userbase in the first place. In the case of Flask, this is shown to be true as the framework has flourished over all of its contemporaries in the Microframeworks category, and almost every single other Python web framework on the market, barring Django, which it closely matches in popularity (as shown previously in **figure 1**).

Another part of the community aspect stems from the documentation of the framework. Good documentation is always important within computing, given the knowledge provided is instrumental to a user gaining a solid understanding of the documented code. This is no different in the world of frameworks, and team behind Flask understood this, ensuring the framework had a comprehensive user guide covering every major feature of it. Pallets (2010) is the official full comprehensive user guide to the framework.

Despite all the benefits however, Flask is not without its flaws. First, Flask lacks a database layer, as explained by Maia, I. (2015), so requires external extensions, which in some (although rare) cases, can cause major conflicts with other extensions added to cover other key features of a site. This is not ideal because such conflicts may require solutions which leave a negative impact on the project as a whole, such as having to remove or entirely rework planned features.

Second, Flask's previously mentioned lack of boilerplate code can become an issue further into a project. This is because using Flask extensively on a larger-scale project can sometimes have developers "reinventing the wheel" just to implement basic features in certain areas. Issues of this kind are rectified within contemporary frameworks of a larger scale, such as Django, which allow for much less repeated code, in exchange for having less flexibility in certain places.

As well as the above issues, Flask does simply miss out on having certain

conveniences that larger full-stack frameworks or Megaframeworks provide. Alongside the previously mentioned lack of a database layer, Herman, M. (2019) lists various other lacking capabilities in Flask. These include: User Authentication, Administrator Options (such as an admin panel) and Advanced security options (such as protection from Injection attacks), all of which can be rectified by Flask extensions. Nevertheless, these will require more work to function when compared to the “batteries included” philosophies of larger-scale frameworks such as Django, which come packaged with all of the features above. These Flask extensions are also not covered in any practical detail within the Flask documentation, meaning users must rely on the documentation provided by the creators of each extension they use.

To summarise, Flask is a simple, lightweight and flexible framework with various potential uses. The framework is user friendly due to its close-knit nature to the most fundamental forms of Python, something which it holds over its contemporaries, which tend to be more detached from the basic Python programming fundamentals. The framework has a decently sized and active community, making finding support and solutions to problems much easier than that of smaller contemporaries including Bottle and CherryPy, as well as being supported by a strong set of developer documentation. Despite the positives, the framework is however outclassed by larger-scaled frameworks where larger sized web applications are considered.

## **2.6 Exploring Django in Further Detail**

Django is a streamlined and efficient framework, coming integrated with nearly all of the full-stack development staples a developer could need to build a fully functional and secure website at a pace described on the Django official site (Django Software Foundation, 2019) as “Ridiculously fast”. The caveat with this, however, is that Django is a much more complicated framework to utilise when compared to its wide range of much simpler contemporaries. This additional complexity of the framework can be off-putting to new users, especially when compared to frameworks such as Flask, which are far more approachable.

One further point of complexity within Django is its project structure. A Flask project takes a simple form of a single file, which can be built out from, Django on the other hand starts with a selection of different “.py” files (Python files) which are created automatically when starting a new Django project using the means detailed within the official documentation (Django Software Foundation, 2019). Thus, each file within a new Django project

contains a different definition for a different feature handled by the framework. Forcier, Bissex & Chun (2008) detail a few of these files used, within the context of a very basic website; “`__init__.py`”, “`manage.py`”, “`settings.py`” and “`urls.py`”.

The “`init`” file, although not specific to just Django, is used to package the files together to address each other, akin to Namespaces within C# or packages within Java. The “`manage`” file is used for working with the likes of permissions, flags and paths, used to save time where some specific project settings are concerned. The “`settings`” file contains all key default settings for the project, including database information, debugging flags and other important variables. The “`urls`” file is a configuration file, used to map URL patterns to actions performed by web applications. With such a range of varying files users need to manage and understand, new users can be alienated by such design.

Another complexity within Django is the framework’s opinionated nature. Mozilla (2019) describes opinionated frameworks as “Those with opinions about the ‘right way’ to handle any particular task. They often support rapid development in a particular domain (solving problems of a particular type) because the right way to do anything is usually well-understood and well-documented. However, they can be less flexible at solving problems outside their main domain and tend to offer fewer choices for what components and approaches they can use.”. This description adequately fits Django as far as many areas of the framework are concerned, since being packaged with specific components by default could lead a user to believe there are only specific ways of performing different tasks. Mozilla goes on to claim that Django is “somewhat opinionated”, fitting the role of “best of both worlds”, further describing it as providing “A set of components to handle most web development tasks and one (or two) preferred ways to use them. However, Django’s decoupled architecture means that you can usually pick and choose from a number of different options or add support for completely new ones if desired.” This description much better fits the framework as a whole where Mozilla justifies that the framework has the option to be flexible. However, to a new user of the framework, who is too unsure of the fundamentals of working with Django, the framework may appear to be “too restricting”, making the developer work in despite the framework, instead of collaboratively with it.

Both of these complications predominantly affect newer or less-experienced users, acting as something of a “learning curve” in the path to mastering Django, and generally become positively impactful further into a developer’s

Django “career”, saving time and resources on much larger scale projects.

Since the framework comes packaged with all of the essentials to making a website, many users of Django will just use what they’ve got. In terms of databases, the developer has the choice of an SQL database, using PostgreSQL, MySQL, or SQLite, as well as the option to use Oracle databases. However, what if a user wants to use something else, such as a NoSQL database such as MongoDB? Otero, C. (2012) explores this possibility in detail. Otero describes the process of implementing support of MongoDB databases as “easy”, with the drawback of losing usage of the automated admin panel provided by Django. Otero goes on to provide an example using MongoDB, having implemented a working connection to a database using the Python library “pymongo” in only four lines of code, with another four dedicated to adding data, and two lines for querying the database and displaying the result. This shows the possibility and simplicity of integrating certain libraries while still using Django, further reinforcing the previously discussed “somewhat opinionated” nature of the framework.

Much like Flask, due to the overall popularity of Django, there is a vast community of users who utilise the framework. This is bolstered by the very detailed official documentation made by the Django Software Foundation (2019), a set of documentation which thoroughly explains each of the key features of the framework in great detail, with various provided examples, in a very well organised fashion. Many members of the Django community online have also made beginner tutorials to assist in streamlining the beginner experience, as a means of enticing more users into using the framework, which can be seen clearly with a single Google search which returns hundreds of up to date tutorials over various formats such as plain text and video.

To summarise, the biggest selling point of Django is its full-stack nature as a framework, which it prides itself on, streamlining development incredibly well once users get over the early learning hurdles. Due to the range integrated development packages, alongside the various conveniences of the framework such as the boilerplate code used to save time when repeating arduous sections of code, Django is an ideal framework for developers working on larger scale projects. The documentation and community of the framework makes it reliable enough to use professionally due to the availability of resources and people capable of working with the framework. Additionally, it also has the relevant security elements for most major systems in place, by default, removing a level of risk when developing various types of projects, and helping reduce development time in projects where further security is

required.

## 2.7 Exploring Pyramid in Further Detail

As discussed earlier in this report, Pyramid is said to be a “Goldilocks” framework, aiming to fit the middle ground between Microframeworks and Megaframeworks. With the vast differences between the Microframework Flask and the Megaframework Django, it is reasonable to question whether, or not, there is feasible space between them where Pyramid could be effectively utilised.

When comparing Pyramid to Flask, some observations can be drawn. An article by Brown, R. (2015) discusses some of the basic comparisons that can be made between Pyramid, Flask and Django.

One point emphasised by Brown is in relation to the new project creation and bootstrapping aspects of each of the frameworks. Brown highlights that Pyramid takes a similar approach to Django when creating a new project, where they are various different files and folders created, each with different intentions for different aspects of the project. There are notably less files created when starting a new Pyramid project by default, when compared to Django. Brown highlights that the key advantage Pyramid holds over the other two frameworks here is its flexibility. This is because Pyramid’s proprietary bootstrapping tool “pcreate” allows the user to select any number of project templates, as well as scaffolds, which function as premade outline applications, which are simply a case of “filling in the blanks”. On the other end of the spectrum, when creating smaller scale applications, Pyramid can also operate out from a singular file, similar to Flask, while still being open to easy and convenient expansion at a later point.

Another point highlighted by Brown is that each of the frameworks’ approaches to templating (dynamic web page elements). Where templating is concerned, both Django and Flask are mostly set in stone, Django using its proprietary “Django template language”, and Flask using the Django inspired “Jinja2”. Pyramid on the other hand is far more flexible, supporting various templating languages, which include the likes of Jinja2 and Mako, alongside various others. However, Pyramid does also come packaged with its own templating language named “Chameleon” by default. Using Chameleon, the Python part of the templating is very similar to that of templating in Django or Flask, however there are differences with the actual template. This is because Chameleon uses XML-based templating, which requires an

entirely different approach to templating when compared to the other two. At the end of the article, Brown deduces that Pyramid is a far more flexible framework than Flask or Django, since it can be used effectively on a much wider range of application types, going from small scale to high end. Brown then goes on to highlight that the extensive flexibility of Pyramid could be part of the reason as to why it isn't as popular as the other two, chalking it up to an overwhelmingly wide range of approaches and options, to the point where users have difficulty deciding what to use, compared to the much more streamlined experiences provided by Django and Flask.

Grehan, R. (2011) briefly discusses the database approach taken by Pyramid, being "policy free". This means that Pyramid has no preferences as to which database format is used, whether it be SQL or NoSQL based. Grehan also highlights this "policy free" nature in regard of the templating used within Pyramid, suggesting that the previously discussed Chameleon is in reality only included with Pyramid as a convenience, instead of being a suggested templating language or even any sort of desired standard. Lastly, Grehan discusses the multiple approaches Pyramid has for mapping URLs to code, having both a traditional method used by various frameworks, including those outside of Python, such as Rails, as well as having a Python specific technique at the developer's disposal. Each of the points put forward by Grehan further the idea of Pyramid being widely flexible, far more so than Django or Flask.

With the apparent flexibility and available choices provided by Pyramid, questions could be proposed as to why it is not nearly as popular as the Django or Flask. This leads on to a need to explore the weaknesses and shortcomings of Pyramid to properly understand where it falls short of the other two frameworks. The key is that, as already briefly discussed by Brown, R. (2015), Pyramid has a very wide range of possible usages, with various ways of implementing an array of different kinds of features. This can lead to a few issues such as a split in community consensus on how certain problems should be tackled, and the requirement for a developer to need to explore various different sub-communities and different sets of documentation, leaving the very flexibility of the framework acting against itself in such cases. The documentation from the official Pyramid website (Pylons Project, 2019) attempts to mitigate these types of issues by providing suggested means of tackling many development issues revolving around the tools packaged as part of Pyramid. However, this is by no means foolproof due to the vast range of available external solutions, alongside those included with the framework. This essentially means that, when compared to frameworks like Django and

Flask, Pyramid can give the impression of being much less well-organised and streamlined.

To summarise, Pyramid is a very flexible framework with large areas of overlap with both Flask and Pyramid. This ultimately leads to the question of whether or not Pyramid is truly a necessary or useful middle ground. This will be explored further within the Implementation stages of this project. Based upon the research discussed in this literature review, it can currently be deduced that Pyramid can be a worthwhile and useful middle ground between Django and Flask, but only if the developer has the necessary skills, understanding and patience to be able to utilise Pyramid effectively.

## **2.8 The Upgrade Path**

Based upon sources researched as part of the above literature sections, some areas of direct or potential overlap can be observed within the three frameworks. When compared in certain ways, each of the three frameworks can align in what may be seen as an “upgrade path”, starting with the more barebones Flask, then changing up to Pyramid or Django part-way through a project without excessive amounts of retreading where already completed features of an implementation are concerned. As well as this, these same principles can be applied to possibly “downgrading” from a framework which may be too excessive or “overcomplicated” for the task at hand. To fully understand this, key aspects of the frameworks must be analysed to discern if the potential of changing between frameworks could be reasonable.

The most obvious point of overlap between the three frameworks is that each of them use Python as their groundwork. This means that normal Python code utilised within any of the frameworks can be directly transferred between the three frameworks. This includes things such as: logic, arithmetic, classes/class structure, and the importing and utilisation of standard Python libraries.

Brown, R. (2015) highlighted the similarities between Jinja2, the templating language used by Flask, and Django’s proprietary templating language. Since Pyramid can also utilise Jinja2, this leads to an area of possible overlap between all three frameworks, where templates can, potentially, be transferred between each of the frameworks with little-to-no actual hard work required. Brondsema, D. (2009) demonstrates the points in which Django templates and Jinja2 templates can be translated to one another, mostly relying on minor syntax adjustments. Django’s official documentation (Django Software

Foundation, 2019) also demonstrates that Jinja2 templates can be used directly with Django with more modern versions of the framework. This makes the templating a strong point of overlap between all three of the frameworks, making it noteworthy for the potential upgrade/downgrade paths for each of the frameworks to each other.

Where databases are considered, all three frameworks have the ability to easily utilise SQL based databases, as well as NoSQL databases, although a little more work may be required for the latter where Django is concerned, as explored by Otero, C. (2012). This makes databases another area where already implemented materials could be directly transferred between the frameworks without a large amount of effort.

Nevertheless, project structure is a point where differences between each of the three frameworks could become problematic, since the file structures and required files of each of the three differ quite drastically. Corresponding examples of the project structure in each of the three frameworks can be seen in **figure 2**, which shows the file structure of an example Flask application, **figure 3**, which shows the file structure of an example Pyramid application and **figure 4**, which shows the files structure of an example Django application. This presents a slight problem for the upgrade path, since it makes entire “.py” files much more difficult to port over directly.



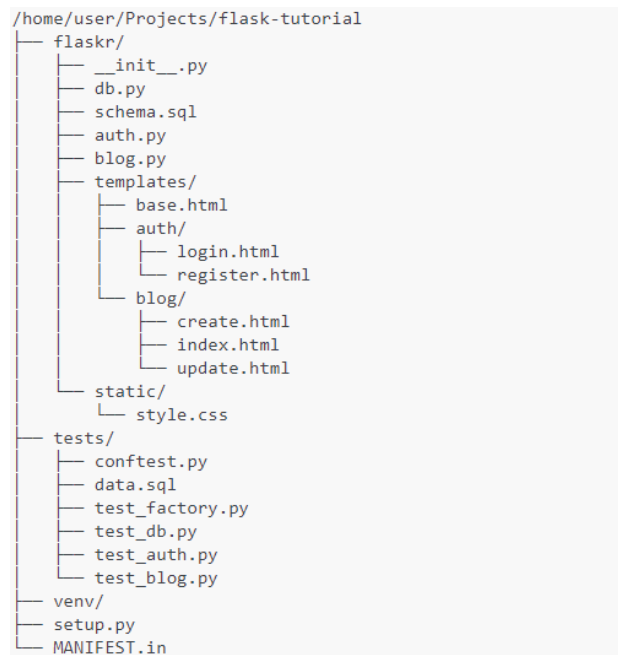


Figure 2: Flask project file structure, Pallets (2010)

Third-party extensions, plugins and addons is another point where differences between the three frameworks come into play. As touched upon by Maia, I. (2015), these third-party features are most often made specifically to serve a single purpose for a single framework. This can cause problems in cases where there is not an equivalent option within the framework that it is being changed to, or if the purpose fulfilled by the added feature is implemented in a different way, requiring potentially extensive reworking to enable a transferal between frameworks.

Collating the above points, it becomes apparent that although code snippets, templates and database elements are directly capable of being ported between the three frameworks, changing the framework used within an application entirely is a more difficult process. The elements researched to be potentially interchangeable between the frameworks are explored further within the Implementation stage of this project, with the idea of fully porting between frameworks being left as an area of research to be potentially explored further in the future.

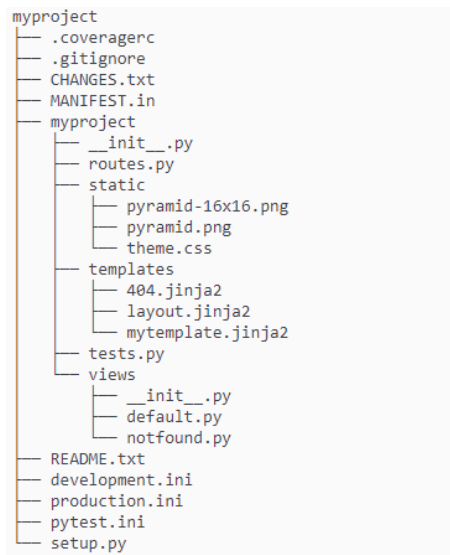


Figure 3: Pyramid project file structure, Pylons Project (2019)

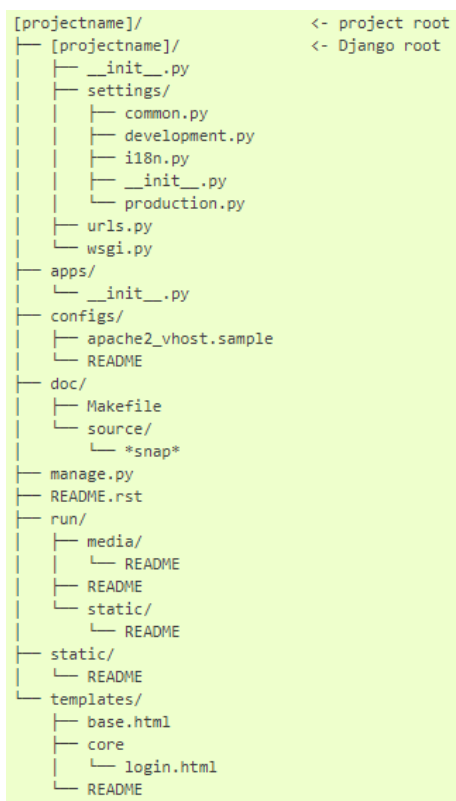


Figure 4: Django project file structure, Mischback (2018)

### 3 Comparison Criteria and Methodologies

A list of different criteria to use in comparing the three frameworks is required before implementing the web applications. A moderately-sized set of different criteria to show the common strengths and weaknesses of each of the frameworks can be collated, based partly on the various sources explored in the literature review, as well as through my own personal experiences developing with each of the frameworks. These criteria with each of their contained methodologies can be split into different sections based on which features they cover and which types of developer those features may apply to.

#### 3.1 Getting Started/The Basics

This category pertains to the most fundamental and simple functionality of each of the frameworks. The methods which can be applied to this category are as follows:

- Installing the framework.
- Starting a new project.
- Creating the most basic possible web application to be run (“hello world” type web application).
- Launching and hosting the web application.

Each of the above methods utilise the key elements for running any web application using any of the three frameworks and thus must be compared against each other to allow the start-up phase of a new project using each framework to be understood fairly in relation to one another. This is important from the perspective of newer users, as these simple elements can be seen as the “make or break” points for a new user on deciding if they are definitely going to use the framework for their project.

The four criteria in this category also allows the points made by Makai, M (2019b) about Flask being notably easy to start a new project with to be put into practice.

#### 3.2 Using Traditional Web Design Aspects

Using Python with a web framework will change steps of the development process of a web application, when compared to traditional means. Despite this, there is still a place for some of the traditional development technologies within the development cycle of a web application made using the frameworks. The methods to test within these criteria are as follows:

- Utilising hard-coded HTML pages using the framework.
- Using CSS and/or Bootstrap for styling pages using the framework.

Due to the frameworks all using templates, a variation on the classic HTML page format, hard-coded HTML will be somewhat uncommon in many projects using the frameworks, however these hard-coded pages do still hold a certain place in scenarios where no dynamic elements are required, saving a small amount of time and effort in the development cycle. The means by which these hard-coded pages are managed and utilised by the framework leads to the need for comparing this functionality using the three frameworks.

Basic CSS and Bootstrap, a commonly used, easy to utilise CSS library (Bootstrap team, n.d.), both hold important places in the development cycle regarding the User Experience (UX) side of web applications. It is important to compare how easily each of the frameworks allows this fundamental to be utilised with both hard-coded HTML pages, as well as templates.

JavaScript is not included within this section as the sorts of tasks it may perform in non-Python implementations such as searching and sorting operations are handled by the Python code. This does not mean JavaScript is redundant within all possible Python implementations. However given the scope of this project, and to maintain clarity of purpose, it is simply not used.

### **3.3 Basic-to-Intermediate Framework Features**

These criteria contain various different methods, each pertaining to a different commonplace feature that is expected to be handled by the framework. These methods are as follows:

- Hosting web pages (both static and dynamic) contained within files external to the “.py” file running the site.
- Dynamically adapting the content of a page when first loading the page.
- Dynamically adapting the content of a currently open page, without reloading it.
- Navigating between different web pages on the same site.
- Retrieving data input by a user (such as inputbox contents).
- Input validation on inputs received from a user.
- Account system where a new account can be created, or an existing account can be logged into and out of.

- Retention of login status between different pages on the same site (session).
- Dynamic URLs (for example, in the context of viewing existing user account pages).
- Redirecting a user to a different web page (such as sending the user to the login page if they are not logged in, when trying to access certain content).
- Handling errors such as 404 (page not found) errors with custom error handling pages.

Each of the above features have varied approaches using each of the three different frameworks in question. In some cases, there will be similarities and overlaps in these approaches, and in other cases they will be entirely different. Because of this, it is important to compare the features between the three frameworks to determine which features are best managed, by which frameworks, and to what extents.

Each of the criteria in the section allows the most noteworthy features of each of the three frameworks documentations' (See: Pallets (2010); Django Software Foundation (2019); Pylons Project (2019)) to be respectfully put to the test and compared.

### **3.4 Database Interactivity and Data Security**

Due to the specialised natures of database interactivity and data security, they were split off from the general framework features, and grouped together due to their close-knit nature. This was done because of both the significance of these features, as well as being due to the fact that they are not handled to the same extent in each of the three frameworks. Where Django comes with much of these features integrated by default, Flask relies on third-party extensions to handle these features. The methods used to compare these are as follows:

- Reading information from a database.
- Writing records to a database.
- Amending information already stored in a database.
- Deleting records from a database.
- Appropriate encoding for sensitive data (such as passwords).

Considering the sheer importance these features have when looking at intermediate-to-large-scale web applications, such as those used for online business, the comparison of the features between each of the frameworks is important. These features also have a range of different approaches and takes, such as which type of database to use, since both SQL and NoSQL databases can be used, making it further worth comparing the optimal options of each of the three frameworks.

These criteria especially allow Django the opportunity to exhibit the security prowess advertised on its official site (Django Software Foundation, 2019). The criteria here also gives Flask the opportunity to prove that it can manage security and database interactivity to some extent, with the help of third-party extensions, as a means of rebutting the points made by sources such as Herman, M. (2019), which criticise Flask for its lack of security and database facilities.

## 4 Implementation

### 4.1 Websites Using Each of the Frameworks

#### 4.1.1 Specification

The objective of creating one website for each of the frameworks, where each site hosts the same content, was to have the ideal grounds to implement and compare each of the criteria explored in **Chapter 3**. To reach this objective, I decided that a used (second-hand) video game sales website was an ideal form for each of the sites to take, as it allowed each of the criteria to be utilised, while also demonstrating the practical real-world utility of each of the three frameworks and integrating one of my hobbies, being video games. As well as the main sites, a set of three basic “Hello Word” type applications were made using each of the frameworks, as a means of gauging the amount of work required to fully implement each of the main sites. These basic sites will be touched upon briefly as part of **Chapter 4.2**.

To aid in visualising the appearance of the web pages before beginning to code them, I created wireframe prototypes for the important pages. First, there is the main page, as can be seen in **figure 5** which contains each of the game listings, ordered in newest posted listing first by default. This page would also be used in displaying the results of using the “search” feature, as well as when using the “my listings” feature, both running filters over the information displayed.

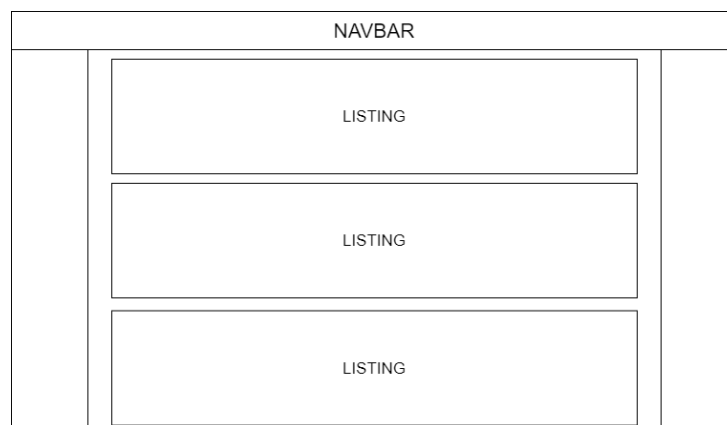


Figure 5: Main Page Wireframe Diagram

The wireframe of the search function which can be used to filter the results down to the user’s criteria can be seen in **figure 6**. This page is somewhat simple, and has room for more criteria to be added, however in its current

state it is sufficient for demonstrating the premise of searching through the data.

NAVBAR		
	<div>Title</div> <div></div>	
	<div>Platform</div> <div>PLATFORMS ▼</div>	
	<div>Sort By</div> <div>SEARCH CRITERIA ▼</div>	
	<div>Search</div>	

Figure 6: Search Page Wireframe Diagram

An important function of the site was the user account system, where a user can create an account, which can be seen in **figure 7**, as well as login, which can be seen in **figure 8**. When a user is not logged in on the site, certain features would be inaccessible, such as the ability to create a new listing, instead redirecting the user back to the login page.

NAVBAR		
	<div>Username</div> <div></div>	
	<div>Password</div> <div></div>	
	<div>Re-enter Password</div> <div></div>	
	<div>Sign Up</div>	

Figure 7: Sign Up Page Wireframe Diagram



NAVBAR		
	<div>Username <input type="text"/></div> <div>Password <input type="password"/></div> <div>Login</div>	

Figure 8: Login Page Wireframe Diagram

When a user is signed in to the site, they can create a new listing, meaning they can add a post of their own to the site's database. This can be seen in **figure 9**. Much like the search function, this is quite barebones in this implementation, but is sufficient in showing the possibilities of the feature if it were further developed.

NAVBAR		
	<div>Title <input type="text"/></div> <div>Platform <input type="text" value="PLATFORMS"/></div> <div>Price <input type="text" value="£"/></div> <div>List</div>	

Figure 9: New Listing Wireframe Diagram

Whenever the site is in use, it will always have a navigation bar (navbar) at the top of the page. This bar allows the user to quickly access different areas of the site in an effective fashion. The wireframe diagram for this can be seen in **figure 10**. It can be seen there are two versions of the bar. This is because one is displayed when the user is logged in and the other is displayed when the user is not.

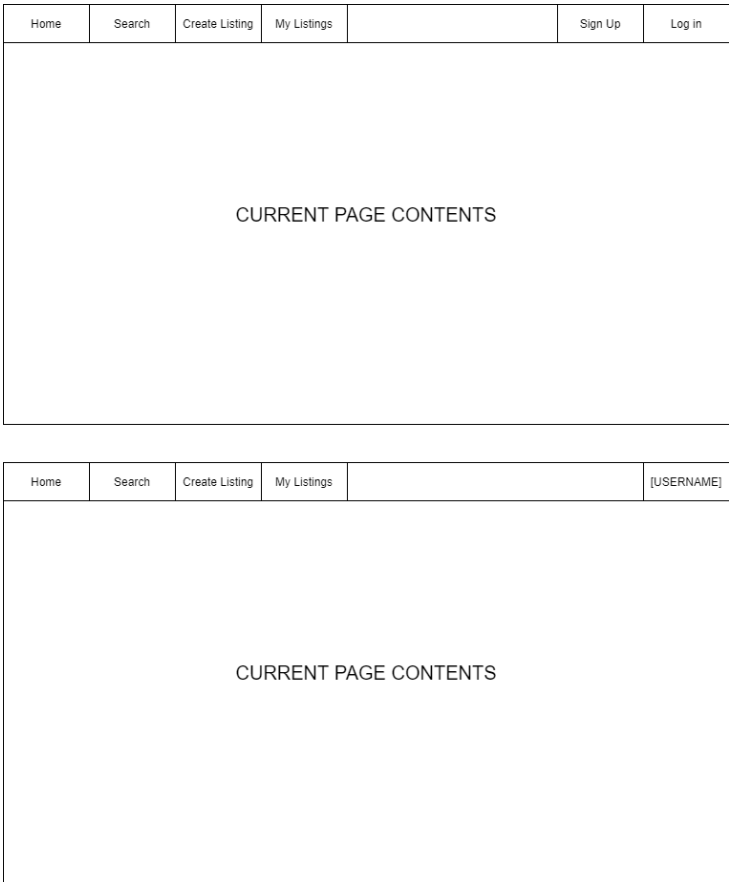


Figure 10: Navigation Bar Wireframe Diagram

Each of the buttons when the user is not signed in have been mentioned above. This leaves the possibility of the user clicking on their own username when signed in left to discuss.

In the case that the user clicks on their own username, they will be taken to a barebones page with account options, these options will include deleting the account and changing the password. A generalised wireframe of this page can be seen in **figure 11**.

A wireframe diagram of a profile page. At the top is a horizontal bar labeled "NAVBAR". Below it, the page is divided into three vertical sections by thin lines. The middle section contains three stacked text input fields labeled "Current Password", "New Password", and "Re-enter New Password". Below these fields is a rectangular button labeled "Change Password". At the bottom of the middle section, separated by a dashed horizontal line, is a red rectangular button with white text that reads "DELETE ACCOUNT (CANNOT BE UNDONE)".

Figure 11: Profile page Wireframe Diagram

#### 4.1.2 Tools Used

The backbone of the implementations of each of the three sites is, as explained in **Chapter 1**, the programming language Python. At the beginning of the project, Python 3.7.4 (the latest version at the time) was downloaded and installed from the official site (Python Software Foundation, 2020) to be used throughout development. Using Python’s “pip” command from the command prompt, different packages were installed to be used for development.

The framework Flask (ver 1.1.1) was installed using the “pip” command, as entailed within the Flask documentation (Pallets, 2010).

Flask-BCrypt is an extension for Flask used to handle certain forms of data encoding and encryption which Flask struggles with on its own. It (ver 0.7.1) was installed using the “pip” command, as shown in the extension’s documentation (Countryman, M., n.d.).

The framework Django (ver 2.2.5) was installed using the “pip” command, as entailed within the official site (Django Software Foundation, 2019).

For encryption when using Django, an official library was used with the “pip” command, as depicted within the Django documentation (Django Software Foundation, 2019).

The framework Pyramid (ver 1.10.4) was installed using the “pip” command, as entailed within the official site (Pylons Project, 2019).

Cookiecutter is a command-line tool used for creating template projects, speeding up the starting phase of a new project. It was used with Pyramid in this project as a means of replicating the easy startup packaged with Django by default. It was installed using the “pip” command, as shown in the Pyramid documentation (Pylons Project, 2019).

Passlib is an encryption library which hosts various methods of securing data. Within this project it was used as a part of the Pyramid implementation to mirror the BCrypt hashing used within both the Flask and Django implementations. It (ver 1.7.2) was installed using the “pip” command, as found within the Pyramid documentation (Pylons Project, 2019).

Jinja2 is the templating language used for the templates on each of the three sites. Each of the three frameworks are compatible with Jinja2 by default, so no downloads or installations were required for the language or any compatibility extensions for it.

Bootstrap is a CSS library used to make the process of styling web pages more slick and convenient. As outlined on the official website (Bootstrap team, n.d.), Bootstrap does not need a download to function, instead opting for the approach of using a CDN (Content Delivery Network), meaning it only needs a URL (web address) within the header of a HTML based document (such as the templates used within this project) to be utilised.

SQLite is a SQL based database service used for hosting databases locally. Since this fits the scope of the implementation, being smaller-scaled websites, it was an ideal tool. SQLite3 (ver 3.31.1) was downloaded and installed from the official website (SQLite, n.d.).

#### **4.1.3 Shared Elements**

In order to keep the comparisons between the three frameworks as solely focused on *just* the frameworks as possible, an attempt was made to reuse as many of the non-framework elements across all three of the sites, as could be done within reason, while still sticking to good development practices and without actively hindering any of the frameworks, by forcing them to use any notably worse methods of implementation.

One key element reused throughout all three sites is the templates, since

each of the frameworks are fully compatible with Jinja2, by default, meaning no workarounds or compromises were required, only single commands in the configurations of the Pyramid and Django implementations.

Another key element is the database structure, keeping the same two databases (listings and users) for each of the three implementations, where each database was implemented using SQLite3.

Lastly, a key element shared between all three implementations are moderately-sized sections of standard Python code, most notably the sections used for input validation and for sorting results sent back to a user when the “search” function was utilised. Here, the user can specify how they would like their results to be returned, allowing price (high-low and low-high) as well as newest first or oldest first.

The three sites are identical in appearance and function on the client side (other than the port they run on when run locally), with each of the three implementations being self-contained, meaning data persisted by one of the implementations is not persisted by the other two. Screenshots of the various pages within the site with a few existing test records can be seen in **figures 12 through 28** located in **Appendix E**.

#### 4.1.4 Flask Approach

When implementing the Flask version of the site, the single Python file approach was taken. This means that excluding the templates and databases, the entire web app can be found within a single file. To keep the file clean and have it retain readability and modifiability, I made separate routes and functions for each of the GET and POST methods for each page of the site. For example, this means the GET method of the login page, which requests the appropriate template from the server to allow a user to view the page can be found in the code as such:

```
@app.route("/login")
def login():
    ### Function Contents ###
```

The POST method of the login page, which takes the user’s input into the fields within the page and returns them to the server will subsequently be found as follows:

```
@app.route("/login", methods = ["POST"])
```

```
def login_post():  
    ### Function Contents ###
```

Where routes with variable URLs were used, angled brackets surround the variable part within the code. For example, the POST method function to return the user's input on the page where the user edits a listing they own to the server is as follows within the code:

```
@app.route("/edit-listing-<number>", methods=["POST"])  
def listingEdit_post(number):  
    ### Function Contents ###
```

This means that if a user was on the page “#website#/edit-listing-12” (where #website# is the address the site is being hosted from), “number” within the code would be 12.

Where the database connection is concerned, I followed the Flask documentation and used the sqlite3 module, which comes packaged as part of Python 2.5 onward. This made database interaction a straightforward process, where queries were made using SQL statements such as SELECT and DELETE.

When displaying pages in Flask, the jinja2 templates were served using the `render_template()` command. However, traditional html pages could also be served using either the same `render_template()` command if they were stored within the “templates” folder within the project folder, or alternatively with the `app.send_static_file()` command if stored elsewhere, the latter of which was being used to display the static page shown when selecting “Purchase” on one of the listings within the site, serving the page from the “static/html” folder.

Where pages redirected the user to a different page, such as in the case of selecting “Create Listing” whilst not logged in, which directs the user to the login page instead, the `redirect()` command was used, where the route of the page the user is being redirected to is within the brackets, for example `redirect("/login")`.

To retrieve the user's input from a post request, the command `request.form[]` was used, where different elements of the page where information is being taken from are specified within the square brackets, such as `request.form["username"]`, which takes the input from the html element named “username” within the POSTed form being requested from.

Since Flask does not come packaged with any tools to securely encrypt pass-

words, the extension Flask-BCrypt was used. This extension allows password hashing in a single line of code using the Bcrypt function, as well as allowing users to compare an unencrypted password to an encrypted one, returning true or false on whether the two match, instead of decrypting the password for comparison, making it a sufficiently secure method of securing user information in this implementation.

To retain user information such as login status between pages, Session is used. Session comes as a part of Flask and allows variables to be stored server-side to be accessed by an active user, based on secure Cookie information within the user's browser.

Utilising custom error pages in Flask is done in a similar way to routing. For example the custom page displayed if the user gets an error 404 (page not found) within the code is as follows:

```
@app.errorhandler(404)
def error404(e):
    return render_template('404.html', wrongURL = request.url),
404
```

#### 4.1.5 Django Approach

For the implementation of the Django version of the site, I followed the new project startup guide from the Django documentation (Django Software Foundation, 2019). This means that instead of starting from a single file, I began with an entire project folder equipped with various Python (.py) files to manage and maintain different areas of the project.

The file “settings.py” found within the “GameSalesSite/GameSalesSite” folder has been modified so that the path names for each of the two existing databases (Listings and Users) would be recognised by Django to be used within the application. This file is also where the type of Hashing used is defined, being “BCryptSHA256”.

The file “urls.py” found within the “GameSales” folder of the project handles all of the interaction between the URL in the user's browser and the function called to deal with the request. There is one “path” in the file for each unique URL the site handles. For example the path for the login page appears as `path('login', views.login, name='login')`, which calls the appropriate function in the “views.py” file pertaining to the URL.

Where variable URLs are concerned, angled brackets are used, with the key difference being that the variable type must also be declared. For example the URL to access and modify a listing the user owns appears as `'edit-listing-<int:number>'` within the path command.

The file `"views.py"` found within the `"GameSales"` folder contains the main bulk of the custom code made for this application. It contains functions for each of the routes from `"urls.py"` such as `home`, which can be seen as `def home(request):`, where `"request"` is added to each function to pass in the request type (GET or POST) as well as any information pertaining to the user, such as the contents of a POST request and session cookies to persist the login status of the user.

To retrieve POST information, the command `request.POST[]` is used. For example to request the username from a POSTed form, the command `request.POST["username"]` is used.

Where the Jinja2 templates are concerned, a small change was required within each template with a submittable form (such as the login page), where Cross Site Request Forgery (CSRF) Tokens were required by Django to allow the template to be rendered. This is a basic security feature implemented by Django to prevent CSRF attacks, a type of attack OWASP (2020) defines as "An attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request."

Where database interactivity is concerned, Django takes the approach of using models. The models are stored within the `"models.py"` file found within the `"GameSales"` folder. There are two models within this project, one for each of the two database tables used, Listings and Users, where each model contains variables to act as one record. The models are called within the `"views.py"` file, and are used to create, load and edit records from the databases without the need for SQL statements. For example the command `allRows = Listings.objects.using('listings_db').all()` connects to the Listings database (specified as `"listings_db"` within `"settings.py"`), and queries for all records. This query can then further refined later, such as in the case where `allRows.filter(seller=getUsername(request))` is used, which filters through all of the records found previously, and returns only those where the `"Seller"` field is equal to the username of the logged in user.



Hashing passwords is a simple procedure in Django using “User”, since passwords are automatically Hashed in this function when being written to the “password” element. For example, the password “toHash” on its own can be Hashed with:

```
hash = User()
hash.set_password(toHash)
```

Where `hash.password` returns the Hashed password. A non-Hashed password, in this case “notHashed”, can be compared against an already Hashed password such as one read from a database, in this case “alreadyHashed”, using:

```
hash = User(password = alreadyHashed)
if hash.check_password(notHashed):
    ### Passwords Match, Do Something ###
```

Using Django there is no way to render a static HTML file from the static folder. Instead static HTML can be rendered using Jinja2 (despite only containing HTML, with no Jinja contents) from the templates folder of the project.

Customer error handling pages can be made in Django with handlers for each individual error number. For example for the page responding to error 404 (page not found) is defined using `def handler404(request, *args, **argv):`. One difference in the 404 page displayed by Django when compared to the Flask and Pyramid implementations is that the incorrect URL to be displayed on the page is not passed in, instead being rendered within the template using `request.build_absolute_uri()`.

#### 4.1.6 Pyramid Approach

The Pyramid documentation (Pylons Project, 2019) explores various ways of starting a new project including single-file distribution and manually creating the workspace for the project. However, for this implementation I took one of the suggested approaches of using Cookiecutter, a project templating tool. This means that the main project files were automatically generated, to be modified as required.

Throughout the implementation two notable files were modified, “routes.py” and “site.py”. The “routes.py” file is in charge of managing the routing between the URL on the user’s side and the corresponding function within the “site.py” file. Each unique URL on the site is stored within this file, each

following the format of `config.add_route()`, with the URL and corresponding route name being within the brackets. For example, the route for the login page takes form as `config.add_route('login', '/login')`, where the first statement is the route name, and the second is the URL. Where variable URLs are used, the variable elements are stored within braces, for example within the page that allows users to edit a listing, the route can be seen as `config.add_route('listingEdit', '/edit-listing-number')`.

The file “site.py” found within the “views” folder contains the main bulk of the modified code within the Pyramid implementation. The functions for each route can be found within this file, such as the route for the login page which appears as follows:

```
@view_config(route_name='login',
              renderer='../templates/login.jinja2')
def login(request):
    ### Function Contents ###
```

A noteworthy part of Pyramid is that it has its template to be rendered within the decorator (`view_config`) as the practice recommended within the documentation, instead of always rendering within the return statement like many other Python web frameworks do. Despite this, deciding which template to use at the return statement is still possible, and is necessary if multiple templates have the potential of being rendered within the same function. For example, within the implementation the Search facilities display a page for the user to enter their search criteria on the GET method, then on the POST method the search criteria are sent to the server, where a page displaying the search results is then rendered, all without changing the URL. To allow for different templates to be used, the `render_to_response()` command is used, where within the brackets the template to be rendered is specified, followed by the template variables and lastly passing in “request”. Within the Search function, this appears as follows:

```
return render_to_response('../templates/mainPage.jinja2', {
    'pageType' : "search-results",
    'listings' : listings,
    'results' : results,
    'order' : order,
    'signedIn' : loginStatus(request),
    'username' : getUsername(request),
}, request=request)
```

Where templates are concerned, Pyramid does not innately understand files in the “.html” format, instead each of the templates require the “.jinja2”

extension. The lack of handling basic HTML files is something of an issue when attempting to render static HTML pages. As a workaround, the static HTML files can be given the `.jinja2` extension and still be rendered using the templating engine, regardless of not actually containing any Jinja within the file. The static files can still also be stored within the static folder, separate from the templates if desired.

To retrieve information from a POST request, the command `request.POST[]` is used. For example, in the case of retrieving the entered username from its field, `request.POST["username"]` can be found.

Redirecting in Pyramid requires a little bit of extra work compared to other Python web frameworks. This is because Pyramid does not have a built in Redirect command, instead an entire route URL is required to then be used with the `HTTPFound()` command. For example, within the implemented site the user will be redirected to the login page if they are not already logged in when attempting to access certain content such as creating new listings. This redirect appears as follows:

```
url = request.route_url('login')
return HTTPFound(location=url)
```

To persist login status between pages of the site, session is used. Session comes as part of Pyramid and is used to store information on the server which is accessed based on Cookie information in the user's browser.

Although Models are included as a part of Pyramid, able to be utilised as a means of representing database records, they were not used as part of this implementation because they did not feel necessary within the scope of the project. One reason for this is because they are not needed to use the querying without SQL statements provided by SQLAlchemy, a module that comes packaged as a part of Pyramid. To query using SQLAlchemy, the tables being queried must first be defined using an engine as part of SQLAlchemy, then the query must be executed. For example a query to acquire all database rows from the Listings database within the implementation appears as such:

```
query = db.select([listings_db])
allRows = connection.execute(query).fetchall()
```

In the above example "db" is the abbreviated name SQLAlchemy is imported under and "listings\_db" is the Listings table of the "listings.db" database, opened using the engine provided by SQLAlchemy.

The Passlib module was installed to handle password Hashing as shown in

the Pyramid documentation (Pylons Project, 2019). This module allows for simple and straightforwards Hashing, for example Hashing “unHashedPW” is done as `bcrypt.hash(unHashedPW)`. A Hashed password, in this case “HashedPW”, can be compared to a non-Hashed password, in this case “unHashedPW”, using the command `bcrypt.verify(unHashedPW, HashedPW)`.

Custom error pages in Pyramid are done in a similar way to standard routing, with the key difference being that instead of defining a route name, a context is defined, detailing the type of error being handled. For example, the custom error 404 (page not found) page within the implementation is called as follows:

```
@view_config(context=HTTPNotFound,
renderer='../templates/404.jinja2')
def error404(request):
    ### Function Contents ###
```

## 4.2 Comparing the Implemented Sites

	Flask	Django	Pyramid
Installation	Single Pip Command	Single Pip Command	Single Pip Command
Starting a New Project	Manually Create app.py	3 Lines in the Command Prompt	Manually Created Python File or Cookiecutter
Lines of code written for basic test application	8 – app.py	12 – views.py 7 – urls.py	15 – app.py
Launching test application	Single Line in the Command Prompt	Single Line in the Command Prompt	Single Line in the Command Prompt
Lines of code written for GameSalesSite	452 – app.py	445 – views.py 18 – urls.py 27 – models.py	581 – site.py 14 – routes.py
Display static HTML files	Yes	Yes, using template engine	Yes, but must have format of a template for template engine
Using CSS/Bootstrap with templates	Yes	Yes	Yes
Separate HTML into different file from Python	Yes	Yes	Yes
Dynamically adapt content of a page when loading	Yes – Jinja2	Yes – Jinja2	Yes – Jinja2
Dynamically adapt content of an already loaded page	Yes – POST request handling + Jinja2	Yes – POST request handling + Jinja2	Yes – POST request handling + Jinja2
Navigating between pages of the same site	Yes – Relative links or Hyperlinks	Yes – Relative links or Hyperlinks	Yes – Relative links or Hyperlinks

ctd...

Navigating to external sites	Yes – Hyperlinks	Yes – Hyperlinks	Yes – Hyperlinks
Redirecting to different pages	Yes - redirect	Yes - redirect	Yes - HTTPFound
Retrieving user input (such as inputbox content)	Yes – POST request handling + Jinja2	Yes – POST request handling + Jinja2	Yes – POST request handling + Jinja2
Input validation with error handling	Yes – Python code + Jinja2	Yes – Python code + Jinja2	Yes – Python code + Jinja2
Account system with create, login and logout	Yes – session + SQLite3 module	Yes – sessions + Django model	Yes – session + SQLAlchemy
Retention of Login status between different pages on the site	Yes – session	Yes – session	Yes – session
Dynamic URLs	Yes – within angled brackets in routing	Yes – within angled brackets with variable type in routing	Yes – within braces in routing
Custom Error Handling	Yes	Yes	Yes
Database interactivity (Read, Write, Edit and Delete)	Yes, using SQLite3 Python module	Yes	Yes, using SQLAlchemy
Password Hashing	No – third party module required	Yes – first party module may be needed for different encryption types	No – third party module required

Table 1: Comparisons and Metrics

## 5 Results and Evaluation

### 5.1 Objective Comparisons Made

#### 5.1.1 Project Startup and The Basics

When looking at the rows of **table 1** pertaining to the starting of a new project and the basic test application made, it can be observed that the notably simpler seeming startup of a new Flask application lines up with the point made by Ronacher, A. (2015), in which the author notes that the design philosophy of Flask makes starting a new project “quick and easy”. This is a key advantage of Microframeworks such as Flask over Megaframeworks such as Django, since the ease of use when starting a new project with new technology can be a make-or-break factor in deciding whether to use the technology for the rest of the project, worth noting as a part of the answer to the Research Question pertaining to comparing the key aspects of Flask compared to Django.

#### 5.1.2 Utilising Traditional Web Development Aspects

As can be observed within the results within **table 1**, all three frameworks have similar capabilities when it comes to utilising traditional web design elements such as basic HTML and CSS. The most notable areas in which frameworks stray from being able to use the traditional elements as expected are in both Django and Pyramid’s slight respective difficulties in handling static HTML files. Where Flask can simply serve any static HTML files from any location within the project directory (such as the “static” folder), Django requires all “.html” files to be rendered through a template engine, which means the static HTML files must be kept within the “templates” folder of the project by default. This is another aspect of Flask which can be seen as an advantage over Django with regard to the Research Question comparing such aspects.

Detaching itself one step further from the other two frameworks, Pyramid cannot utilise “.html” files at all by default, instead requiring a template engine with its own respective file type (such as “.jinja2”, as used in my implementation). Despite this, the unlike Django and Flask, the templates can be stored anywhere within the project directory by default, so although the file type needs to be changed to that of a templating engine’s format, the file can still be stored in a directory of the developer’s choosing (such as the “static” folder).

Each of the three frameworks can utilise HTML based templates, meaning each are fully compatible with CSS and Bootstrap, as can be seen within my implementation.

### 5.1.3 Routing and Requests

During the **Implementation** stage of the project, it was noted that each of the three frameworks had slightly different takes on routing. While Flask keeps its routes as part of the methods they are associated with, the other two frameworks keep their routes in a separate file where each route calls the view method they are associated with. The only notable differences between the Django and Pyramid in this respect are solely based upon the syntaxes of the frameworks. This could be considered an advantage of Django over Flask in some respects as part of the research question pertaining to such, due to the code cleanliness and readability provided by keeping repetitive aspects like routing separate from the main view functions.

Where Redirecting was concerned, I found that Flask and Django took the same approach of using a simple “redirect” command at the return statement of a function. Pyramid on the other hand did not have a direct means of redirecting a user, but did have an alternative in “HTTPFound”, requiring the whole URL of a route to first be recovered using “route\_url”, then being passed in to the “HTTPFound” at the return statement of the function. This makes the Pyramid approach slightly more complex than the approaches of the other two frameworks.

Each of the three frameworks have very similar means of handling requests, each using GET and POST requests to handle user interaction. Each also uses Sessions as part of the requests sent to and from a user, storing information in the user’s cookies to persist information such as login status.

### 5.1.4 Database Interactivity and Security

As explored during the **Literature Review**, each of the three frameworks has a range of options to select from when deciding on a database to adopt for development. Flask has a range of third-party extensions to select from, alongside those directly supported by Python by default, Django supports a small range of databases by default with a wide array of third-party backends as explained by its documentation (Django Software Foundation, 2019),



and Pyramid comes packaged with SQLAlchemy, allowing support for various SQL based databases, with third-party extensions existing for NoSQL databases as well.

SQLite3 is one database type found and used within the **Implementation** stage of the project which all three frameworks can support without the need of any third-party modules (with Flask using the “sqlite3” base Python module to make up for its lack of packaged database support).

Of the three frameworks, Django makes use of its Models system, streamlining database interaction when compared to the “Connect as Needed” approach taken with the other two frameworks in the **Implementation**, where a new connection must be made from scratch each time database connectivity is used. This is another advantage of Django over Flask with regard to the research question discussing such. Despite the approach I used, Pyramid also does have its own take on the Models system found within the documentation of the framework (Pylons Project, 2019), however it is more complex to utilise than that of Django, especially when using multiple different databases as I did.

As explored within the **Literature Review**, Django is the only framework of the three with a direct approach to security, coming packaged with various tools and utilities to aid in making the framework “Reassuring Secure” as advertised on the site (Django Software Foundation, 2019). This is another key advantage of Django over Flask concerning the research question discussing such, since security elements are integral to most websites in some form. To make up for the lack of packaged security utilities in the **Implementation**, third-party extensions to handle password Hashing were required for Flask and Pyramid, whilst Django only required a small first-party addon to allow consistency in using the same “SHA256” algorithm across all three sites. Due to the different extensions used, the test records made for the databases had to be remade for Pyramid because it formatted Hashed passwords in a different way from the other two frameworks.

### 5.1.5 Shared Resources and Porting Potential

As explored between the **Literature Review** and **Implementation**, there are various elements that could be/were shared between each of the sites. Within the **Implementation** I shared the Jinja2 templates which were compatible with all three frameworks, various sections of standard Python code such as a sorting algorithm and each of the databases used. Despite this

however, there were many elements that could not be translated between the three frameworks with any sort of ease or any simplicity. These include the routing, database interactivity and security elements of the site. Because of these discrepancies, my answer to the research question regarding the portability of the frameworks between each other is that the frameworks are not directly portable between one another, although various elements can be shared between them, if desired.

## **5.2 Personal Reflection on the Project**

### **5.2.1 Positive Aspects of My Implementation**

I would consider one of the major strengths my approach to the project is the results highlighting the general objective points to be compared between each of the frameworks. These include highlighting the different capabilities, capacities and dependencies of each of the frameworks, such as the need for third-party modules to handle security aspects not covered by the framework in the cases of Flask and Pyramid.

Another point of my approach I consider to be strong is the perspective I provide regarding the perspective of a new user with each of the frameworks. This is because prior to the project, I had no experience using Django or Pyramid, and had only used Flask on one small-scale project. This resulted in my impressions and opinions of the framework concerning how the experience using them may feel to a new user being organic and natural.

Additionally, a good personal lesson which I have learned from the project is in guarding my expectations when using new tools. When starting the project, I expected Django to be the point I had the most difficulty with and would require the most development time, however this was not the case and Django actually took the least time of the three frameworks for me to complete. Pyramid, on the other hand, I expected to require more time than Flask and less than Django, where in actuality the difficulties I experienced in developing the site while using it led to it needing the most development time of the three. This was a good lesson since it taught me to avoid time-based assumptions until I have the adequate experience to make such predictions.

### **5.2.2 Negative Aspects of My Implementation**

One negative aspect of the project was that each of the three implementations contained some noteworthy amounts of repeated code within them-

selves, which could have been subjugated to functions. Some examples of this would be the database “connect-execute-close” code within the Flask and Pyramid sites and the code used to organise the listings to be sent to the template within each of the three sites.

A noteworthy challenge I had to overcome as part of the project was in navigating the Pyramid documentation in attempt to find correct or good practices in executing different aspects of the site. This was because I found the documentation to be messy and hard to find the appropriate information within. Due to this, I suspect that my Pyramid implementation may not have ended up as well refined or optimised as it could be for the sake of fair and unbiased comparison. As well as this, I feel that this oversight ended up wasting a notable amount of my development time which could have been used to further refine each of the sites or develop extra features to improve the outcome of the project as a whole.

The most notable negative aspect of my implementations is that the three sites alone do not constitute enough data to gather adequate information to make truly objective observations. To properly counteract this, a sizeable pool of data from external sources with different preferences and experiences would have been required, which was planned as part of the project during the mid-stages of development but had to be relegated due to time concerns. The ideas behind this plan can be seen within the Future Work section of the **Conclusion and Future Work**.

### 5.2.3 What Would I Have Done Differently?

If I were to go back and restart the project from scratch, I would have started the **Implementation** stage of the project much earlier within the development timeline. This would have allowed for some time to allocate to at least starting the Git based metric tool that was planned in the mid-stages of the project, as discussed within the prior subsection and within the Future Work section of the **Conclusion and Future Work**.

As an alternative if I were to redo the project with the information I have now, I would likely alter the scope of the project to have only covered Flask and Django, forgoing Pyramid for the sake of having more time. This would have allowed for me to much further flesh out the Flask and Django implementations with the excess time I would have had from not spending so much development time working on Pyramid. Had the two implementations been

further fleshed out, the scope and scale of the two would have been larger, allowing for the data they compared to be more accurate, akin to medium scale real-world projects, allowing for more realistic comparisons.

### 5.3 Recommendations based on the Research and Results

#### 5.3.1 Flask

Based on the points discussed within the **Literature Review**, Flask is an easy to recommend framework for developers who are new to Python web development and are looking to learn the fundamentals, as well as developers who are interested in creating simple web applications in a short time span. Based on the same points, Flask is not worth recommending to developers who have larger ambitions. Those who are looking for a framework to use over a longer term, potentially with a sizeable team, to maintain a medium-scale to large-scale web application, implementing security and administrative features, may find Flask frustrating or detrimental to their projects.

The above points mostly align with my own personal experience from the **Implementation** section of the project. The one key difference is that I would argue that Flask can be worth recommending to certain developers with larger-scale ambitions with the knowledge that various modules may need to be installed to compensate for features Flask lacks when compared to feature-rich Megaframeworks such as Django.

#### 5.3.2 Django

The benefits of Django discussed within the **Literature Review** make it worth recommending to those with the time to learn how to utilise it effectively in the context of medium-large projects, especially those with required security elements or the ambition to expand and develop their web app over time.

In the same vein, Django may not be ideal for smaller scale developers, especially those without experience using the framework before starting a project, as it has a much steeper “learning curve” compared to frameworks such as Flask. The framework can be seen as being “overkill” or “bloated” when working on particularly basic web applications (such as those for personal use only), and thus is not worth recommending to developers who fall under this category.

My personal experience of the framework from the **Implementation** section of the project leads me to disagree with the argued “overkill” nature of using the framework for smaller scale web applications. This is because of the amount of work Django does on behalf of the user. Whilst it still may be more complex to initially develop with, compared to Microframeworks like Flask, I believe it to be much easier to use in maintaining and updating existing applications. With its various systems of only having to fully develop certain things once then simply utilise them with a single line of code when required, such as the Models system for database interaction and the Routing used for abbreviating URLs and attaching them to methods, Django proves its ease of use.

### 5.3.3 Pyramid

Based on the findings of the **Literature Review**, the capabilities of Pyramid make it ideal for developers who wish to take on a project of which they’re unsure of the scope or end goal when starting, as well as those who wish to start small on a project and frequently expand upon it.

Based on these same findings, Pyramid may not be ideal for those who have a clear project scope in mind at the start of a project when compared to other frameworks, including Django and Flask, which tend to be much more straightforward to utilise in the context of a project that fits within one of either of their most ideal user groups. This is because between the two frameworks, most projects ideas can be implemented effectively and efficiently.

During the **Implementation** stage of the project, I personally found Pyramid the most difficult to work with of the three frameworks. This was due to the documentation, which I found to be somewhat difficult to navigate, especially compared to that of the other two frameworks. In addition, there is an obvious difficulty in effectively using the somewhat lacking and often inconsistent publicly available resources, likely owing to the “over-flexibility” of the framework, as discussed by Brown, R. (2015).

As a result of my poor experience with Pyramid, and the unnecessary-feeling difficulties I faced when using it during the implementation stage of the project, I cannot personally recommend the framework. Despite this, I do recognise the points made by the findings of the literature review as being valid. Consequently, for the research question pertaining to if situations where Pyramid could and should be used over frameworks such as Flask and

Django exist; I can agree that Pyramid can, and does, hold a niche role for users with a certain level of development skill and a specific vision for their site in mind, who require a mid-sized framework.

#### 5.4 Research Question Answer Summary

What key advantages and disadvantages arise from using a Microframework such as Flask over a Megaframework such as Django, and vice versa?

Points in favour of Flask:

- Easier and more simplistic project startup process.
- Much more lenient in ways of serving traditional elements such as static HTML.

Points in favour of Django:

- Easier to manage routing due to much improved readability and code cleanliness.
- Models system, making database connectivity far more efficient within the code.
- Strong focus on security, meaning many vital security features come as part of the framework, instead of requiring third-party addons or extensions.

Are there situations where a “Goldilocks” (flexible mid-sized) framework such as Pyramid could and should be used over a Microframework or Megaframework?

Yes, but only in niche situations where users with a certain level of development skill and a specific vision for their site in mind. In most cases however, frameworks more akin to Flask or Django are more useful.

Is there a clear path allowing users to port existing Web Applications between any of the three frameworks in question? (E.g. Port an existing Flask application to Django with minimum effort or challenge).

There is not a clear path between any of the three frameworks, however different elements such as Templates, Database elements and standard Python code can be shared directly.

## 6 Conclusion and Future Work

The intent of the project was to analyse and compare the three frameworks in question, functioning as an aid to various potential user bases in attempt to help inform which framework may be the best for which types of developer. To this extent, it was discovered that each of the three frameworks have their own uses, although they are not equally distributed, as they do not each best fit the same amount of scenario types.

Despite the findings of the project being useful at proving certain points, they are quite heavily subjective in certain ways, especially where developer-oriented metrics such as lines of code, as well as unused metrics such as code complexity are concerned. Due to this, future work is recommended.

The main piece of future work that I would have liked to have undergone during the project is the development of a Git pull based system, where different selections of public online repositories would have been pulled from sites like GitHub. This idea was discussed to be added to the project around the mid-stage, but was inevitably dropped due to the time constraints. The main idea is that online code repositories would be filtered by tags indicative of each of the three frameworks. The found repositories would then be pulled, where different scans and searches would be used to attempt to work out a various metrics such as the lines of code, the amount of different unique commands used and the amount of third-party addons required, as well as other metrics. When combined with the results of my own implementation run through these same criteria, it would be able to indicate whether, or not, my results were appropriately indicative and representative of the capacities, capabilities and shortcomings of each of the frameworks as a whole. This would mean more objective comparisons would be able to be made, allowing for much for straightforward and specific recommendations.

It can be concluded that Django was found to be the best suited to the widest range of tasks on its own, although being slightly complex for very new users and taking the most work to start a new project with. Flask was stood out as being the most ideal for new users who are first learning how to operate with Python-based web frameworks or with development in general. Nevertheless, the framework's main shortcomings rest in its reliance on third-party modules for many of the more advanced features it may require, and that it begins to struggle, when compared to Django where large-scale projects are concerned. Pyramid was concluded to be the least commonly useful of the three frameworks, being only particularly effective, when com-

pared to the others, in situations where the user has an exact scope in mind and has adequate enough experience to be able to properly utilise the broad range of possible ways the framework can tackle many different tasks.



## References

- Aslam, F., Mohammed, H., Musab, J., & Munir, M. (2015) *Efficient Way Of Web Development Using Python And Flask*. International Journal of Advanced Research in Computer Science, 6(2), 54-57.
- Bootstrap team (n.d.) *Bootstrap*. Retrieved February 3, 2020, from <https://getbootstrap.com/>
- Brondsema, D. (2009). *Convert Django Templates to Jinja2*. Retrieved January 31, 2020, from <https://splike.com/>
- Brown, R. (2015). *Django vs Flask vs Pyramid: Choosing a Python Web Framework*. Retrieved September 22, 2019, from <https://www.airpair.com>
- Cass, S. (2019, September 6). *The Top Programming Languages of 2019*. Retrieved October 14, 2019, from <https://spectrum.ieee.org>
- Christensson, P. (2013, March 7). *Framework Definition*. Retrieved October 22, 2019, from <https://techterms.com>
- Countryman, M. (n.d.). *Flask-BCrypt*. Retrieved February 11, 2020, from <https://flask-bcrypt.readthedocs.io>
- Django Software Foundation (2019). *The Web framework for perfectionists with deadlines | Django*. Retrieved October 31, 2019, from <https://www.djangoproject.com>
- Dory, M., Parrish, A., & Berg, B. (2012). *Introduction to Tornado*. Sebastopol, CA: O'Reilly Media.
- Forcier, J., Bissex, P., & Chun, W. (2008). *Python Web Development with Django*. Boston, MA: Addison-Wesley Professional.
- GitHub (2018). *Projects | The State of the Octoverse*. Retrieved October 14, 2019, from <https://octoverse.github.com>
- Grehan, R. (2011, August 10). *Pillars of Python: Pyramid Web Framework*. InfoWorld.com; San Mateo [Electronic version]. San Francisco, CA: Infoworld Media Group.
- Guardia, C. (2016). *Python Web Frameworks*. Sebastopol, CA: O'Reilly Media.
- Hansson, D. H. (n.d.). *Getting Started with Rails*. Retrieved January 9, 2020, from <https://guides.rubyonrails.org>

Hellkamp, M. (2019). *Bottle: Python Web Framework*. Retrieved November 1, 2019, from <https://bottlepy.org>

Herman, M. (2019, September 29). *Django vs. Flask in 2019: Which Framework to Choose*. Retrieved November 13, 2019, from <https://testdriven.io/blog>

JetBrains (2019, February 5). *Python Developers Survey 2018 Results*. Retrieved November 2, 2019, from <https://www.jetbrains.com/research>

Kohan, B. (n.d.). *Guide to Web Application Development*. Retrieved January 9, 2020, from <https://www.comentum.com/guide-to-web-application-development.html>

Li, Y., Das, P., & Dowe, D. (2014, July). *Two decades of Web application testing – A survey of recent advances*. Information Systems, 43(1), 20-54.

Maia, I. (2015). *Building Web Applications with Flask*. Birmingham: Packt Publishing.

Makai, M. (2019a). *Bottle – Full Stack Python*. Retrieved November 1, 2019, from <https://www.fullstackpython.com/bottle.html>

Makai, M. (2019b). *Flask – Full Stack Python*. Retrieved November 12, 2019, from <https://www.fullstackpython.com/flask.html>

Microsoft (2020). *What is ASP.NET?*. Retrieved January 9, 2020, from <https://dotnet.microsoft.com>

Mischback (2018). *Project Structure*. Retrieved January 31, 2020, from <https://django-project-skeleton.readthedocs.io/en/latest/structure.html>

Mozilla MDN contributors (2019, August 18). *MDN web docs – Django Introduction*. Retrieved November 19, 2019, from <https://developer.mozilla.org>

Myhrvold, C. (2014, February 3). *The Fall Of Perl, The Web's Most Promising Language*. Retrieved January 21, 2020, from <https://www.fastcompany.com>

Otero, C. (2012, February 22). *Using MongoDB with Django*. Retrieved October 8, 2019, from <https://developer.ibm.com>

OWASP (2020). *Cross Site Request Forgery (CSRF)*. Retrieved March 12, 2020, from <https://owasp.org>

Pallets (2010). *Welcome to Flask – Flask Documentation*. Retrieved November 12, 2019, from <http://flask.palletsprojects.com>

Perl.org (2020). *The Perl Programming Language*. Retrieved January 9, 2020, from <https://www.perl.org>

Pylons Project (2019). *Welcome to Pyramid, a Python Web Framework*. Retrieved November 2, 2019, from <https://trypyramid.com>

Python Software Foundation (2020). *Welcome to Python.org*. Retrieved February 15, 2020, from <https://www.python.org>

Ronacher, A. (2015). *Flask | The Pallets Projects*. Retrieved October 31, 2019, from <https://palletsprojects.com>

SQLite (n.d.). *SQLite*. Retrieved February 3, 2020, from <https://www.sqlite.org>

The CherryPy team (2019). *CherryPy – A Minimalist Python Web Framework*. Retrieved November 2, 2019, from <https://cherrypy.org>

The jQuery Foundation (2020). *jQuery*. Retrieved January 9, 2020, from <https://jquery.com>

The PHP Group (2020). *PHP: Hypertext Preprocessor*. Retrieved January 9, 2020, from <https://www.php.net>

The Tornado Authors (n.d.). *Tornado Web Server*. Retrieved November 1, 2019, from <https://www.tornadoweb.org>

TIOBE (2019). *TIOBE Index*. Retrieved October 14, 2019, from <https://www.tiobe.com/tiobe-index>

Woychowsky, E. (2006). *Ajax: Creating Web Pages with Asynchronous JavaScript and XML*. Upper Saddle River, NJ: Prentice Hall.

Wozniewicz, B. (2019, February 1). *The Difference Between a Framework and a Library*. Retrieved October 22, 2019, from <https://www.freecodecamp.org>

Zend (2020). *Zend Framework*. Retrieved January 9, 2020, from <https://framework.zend.com>

# Appendices

## A Initial Project Overview

### A.A Project Title

Comprehensive Analysis and Comparison of Flask, Pyramid and Django  
(Working title)

### A.B Overview of Project Content and Milestones

- Briefly defining what a web framework is, and why one is used.
- Discussing the “as advertised” properties of the three Python-based frameworks in question; Flask, Pyramid and Django.
- Creating a basic set of expectations for each of the three frameworks based on the “as advertised” properties.
- Researching each of the three frameworks, studying various research papers of varying relevance to further understanding in regards to the expectations of each of the three frameworks.
- Exploring the idea of the framework “Upgrade path” (upgrading from Flask -> Pyramid -> Django), and which features can and cannot be directly ported between each of the frameworks.
- Creating a relevant set of test criteria to which each of the three frameworks can be put through.
- Implementing the test criteria in the form of three near-identical websites to confirm similar and different approaches in development between the three frameworks.
- Creation of a means of identifying which of the three frameworks would best suit any given user, created using the findings of the implementation.
- Evaluation of implemented features to prove/disprove usefulness of them and decide whether the project was a success or not.

### A.C Main Deliverables

- A set of test criteria for each of the three frameworks to be put through.
- Three near-identical implemented websites based on the test criteria.
- A means of identifying which of the three frameworks would be the most beneficial to any given user based on their specification and desired features in their own website.

**A.D Target Audience for the Deliverables**

Anyone who desires to build a website using a Python-based framework, whom are unsure which framework would be the most suitable for the features they intend to implement. Alternatively, the deliverables may prove useful for those who are looking to understand the Python framework “Upgrade path”, and if it will be of any use to them.

**A.E The Work to be Undertaken**

- Investigation of existing research regarding each of the three frameworks in question.
- Construction of criteria in which each of the three frameworks can be compared.
- Implementation of three near-identical websites, one for each of the three frameworks, to which the generated criteria can be applied.
- Designing of a means to allow a user to work out which of the three frameworks will work best for them, based on their own specifications and desired features in their own website.

**A.F Additional Information/Knowledge Required**

- Expansion of my current existing knowledge and skills in the use of the Flask framework to extend to the scope of the project.
- Learning how to use the Pyramid and Django frameworks to a professionally competent level with no significant prior experience with either.

**A.G Information Sources that Provide a Context for the Project**

- Brown, R. (2014). Django vs Flask vs Pyramid.
- Grehan, R. (2011). Pillars of Python.
- Aslam, F. et al. (2015). Efficient Way of Web Development Using Python And Flask

**A.H Importance of the Project**

A convenient means of deciding on which framework best suits any given context is an ideal tool for newer or less experienced users of Python-based web design as a means of making the early stages of a new project simpler, as many existing means are quite vague and unhelpful, especially to newer or less knowledgeable users.

**A.I The Key Challenges to Overcome**

- Developing a fair systematic approach to comparing each of the frameworks.
- Ensuring each of the three implemented websites are each developed with the same level of care and from a neutral standpoint, so that no incorrect bias or favouritism is applied to any one framework over the others.

**B Second Formal Review Output****SOC10101 Honours Project (40 Credits)****Week 9 Report****Student Name:** Kieran Burns**Supervisor:** Pete Barclay**Second Marker:** Simon Wells**Date of Meeting:** 29.11.2019

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **yes**

If not, please comment on any reasons presented

**Yes.**

Please comment on the progress made so far

**It appears that an appropriate amount of effort has been put in.**

Is the progress satisfactory? **yes no\***

Can the student articulate their aims and objectives? **yes**

If yes then please comment on them, otherwise write down your suggestions.

**These are generally appropriate to the students programme.**

Does the student have a plan of work? **yes**

If yes then please comment on that plan otherwise write down your suggestions.

Does the student know how they are going to evaluate their work? **yes**

If yes then please comment otherwise write down your suggestions.

**Student articulated some metrics but could consider applying them to a wider more objective set of projects using the targeted frameworks.**

Any other recommendations as to the future direction of the project

**Not at present.**

Signatures: Supervisor      Pete Barclay  
                  Second Marker      Simon Wells  
                  Student      Kieran Burns

Please give the student a photocopy of this form immediately after the review meeting; the original should be lodged in the School Office with Leanne Clyde

\* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided



**C Weekly Supervisor Meeting Notes****C.A Trimester 1, Week 1 (20/09/2019)**

- Confirming the idea that one of the main points of the project is exploring the points in which each of the three frameworks diverged from one another.
- Ensuring I'm aware of the resources available on Moodle, especially the important dates during the project.
- Ensuring I know where to look in regards to gathering resources and have a tool for managing them.
- Discussed the Initial Project Overview due at the end of Week 3 (04/10/19), to ensure I have a good general idea of what needs to be done by then.
- Briefly speaking on getting started working through tutorials on Pyramid and Django, so that my knowledge in using those frameworks parallels my existing knowledge in Flask.
- I was asked to work on an at least 1-page document to clarify the terms and parameters of the project in writing, to be done by the middle of week 2 (Mon – Wed – 23/09/19 – 25/09/19)
- Discussed making a personal project schedule

**C.B Trimester 1, Week 2 (27/09/2019)**

- Briefly discussed my first draft of the Initial Project Overview (IPO), which I worked on beyond the 1-page document asked for during the Week 1 meeting, since the 1-page covered a good majority of the points in the IPO.
- Discussing References in a little more detail than week 1, covering the resources I found between Week 1 and 2's meetings, with some pointers in how references will be used during the project (not all references need to be used for major points etc).
- Tasked with getting the frameworks set up and working on my own Personal Computer.
- Tasked with making a first draft of a project schedule.
- Tasked with taking notes on any possible implementation methods that come to mind/are found within references to be used during later stages in the project.

**C.C Trimester 1, Week 3 (04/10/2019)**

- Discussed my start on working with each of the three frameworks since the Week 2 meeting.
- Briefly discussed the documentation I wrote since the Week 2 meeting on the setup and basic usage of each the three frameworks.
- Discussed IPO to ensure it was ready for submission.
- Discussed next steps in the project.
- Discussed starting the Literature review, and more places to find relevant resources.
- Briefly discussed new resources I found since the Week 2 meeting.
- Briefly discussed taking clear notes of every stage of the project, to ensure everything that could be useful is documented properly.
- Further discussed my personal project schedule, deciding it is to be written in fortnightly blocks, rather than weekly.

**C.D Trimester 1, Week 4 (11/10/2019)**

- Discussed my project schedule.
- I was suggested to convert my schedule to a Gantt chart for presentation and readability.
- Discussed some of my references in detail.
- Further discussed the direction I'm looking to take with the project, and which topics I look to cover as part of my literature review.

**C.E Trimester 1, Week 5 (16/10/2019)**

- Discussed my Gantt chart I made.
- Small suggestions made for changes to be made to Gantt Chart.
- Discussed my report introduction.
- Discussed my report plan.
- Discussed Literature Review in detail in preparation for work to be done on it before next meeting.

**C.F Trimester 1, Week 6 (23/10/2019)**

- Discussed Finished First Draft Project Introduction
- Discussed Work Done on Literature Review thus far

**C.G Trimester 1, Week 7 (No Meeting)**

No Meeting this week due to supervisor's unavailability

**C.H Trimester 1, Week 8 (06/11/2019)**

- Discussed Literature Review progress made over the past 2 weeks.
- Discussed the work that needs to be done over the coming weeks.
- This week's aim is to complete the main body sections pertaining to Flask and Pyramid.
- Next week's aim is to complete the main body section pertaining to Django, as well as complete a first draft of comparison criteria to be used in the implementation stage of the project.

**C.I Trimester 1, Week 9 (13/11/2019)**

- Discussed work done over the past week.
- Was suggested to turn my implementation criteria into both a table for overview, and paragraphs for further detail, possibly including potential sources to add more academic purpose to the implementation as a whole.
- Discussed Interim report, being told that it should be a single document in the form; Introduction, Literature review (so far), next steps (implementation criteria & plans), appendices (Gantt chart, meeting notes & any other relevant side-documents).
- Spoken about the idea of "general consideration" when using topics in the literature review that are a given to users within a field (e.g. "The sky is blue" won't have academic sources to state the fact).
- Discussed next steps; Finish Flask section of Literature review, work on Django and Pyramid sections, and work on adding the appropriate contents to a document to begin work on the interim document.

**C.J Trimester 1, Week 10 (20/11/2019)**

- Discussed work done over the past week.
- Discussed progress on interim review report.
- Set date for Interim meeting.
- Discussed refining implementation criteria in further detail.
- Discussed evaluating sources as part of the literature review, in hopes to possibly improve the quality in certain areas.
- Given access to a previous student's (unrelated) Masters report to help in formulating the format contents of the implementation criteria and methodologies.

**C.K Trimester 1, Week 11 (29/11/2019) [Interim Review]**

- Advised to broaden background on languages and environments outside of Python as part of literature review.
- Advised to tidy and further separate elements of the timeline, as well as amend certain headings.
- Advised to tackle the subjectivity of personally generated metrics, such as lines of code or “code complexity”.
- Suggested to adopt the LaTeX format sooner rather than later.
- Reminded to include the full list of references at the end of my paper, which were forgotten as part of the submitted interim report.
- Highly suggested to expand the implementation stage of the project. Discussed possibilities:
  - Culminate a set of public projects using each framework off of GitHub, to be used in calculating “more objective” takes on the metrics found when personally implementing with each of the three frameworks.
  - Make a tool to port either whole web applications or parts of web applications, such as a template translator (Jinja2  $\Leftrightarrow$  Django Templating Language).

**C.L Trimester 1, Week 12 (04/12/2019)**

- Discussed retaining 3-site implementation for personal work/proof of competence utilising each of the frameworks
- Discussed exploring small, randomly pulled sets of applications from GitHub tagged with each of the three frameworks, to apply various metrics to:
  - Lines of Code
  - File Size
  - Number of Classes
  - Trends
- Discussed implementing simple, small-scale translation tool between Jinja2 and Django Templating Language to demonstrate the similarities between the two templating languages.

**C.M Tr1 Exams and Winter Break**

No Meetings Held During This Period

**C.N Trimester 2, Week 1 (No Meeting)**

No Meeting this week due to supervisor's unavailability

**C.O Trimester 2, Week 2 (24/01/2020)**

- Discussed progress made on literature review over Winter Break.
- Discussed next steps in beginning the actual implementation, with the aim of having the first of the websites finished in 2-3 weeks.
- Discussed ideas on what form the websites should take, such as a rental service or discussion board.

**C.P Trimester 2, Week 3 (No Meeting)**

No Meeting this week due to supervisor's unavailability

**C.Q Trimester 2, Week 4 (06/02/2020)**

- Discussed current progress with Flask implementation.
- Discussed next steps.
- Briefly discussed the analysis and comparisons to follow the implementation.
- Decision made to drop the idea of making Git pull system as a means of generating additional metrics.

**C.R Trimester 2, Week 5 (13/02/2020)**

- Looked over and discussed the finished implementation of the Flask website.
- Discussed next steps in moving on to Pyramid and Django.
- Further discussed the writing to follow the implementation.

**C.S Trimester 2, Week 6 (21/02/2020)**

- Briefly discussed finished Django implementation.
- Talked about what lengths to go in certain areas of report writing, such as the detail when discussing the visual design of the web pages.
- Sent copy of current draft of main report to be looked over.

**C.T Trimester 2, Week 7 (27/02/2020)**

- Discussed progress made on the Pyramid implementation.
- Discussed backup plan in the case that the Pyramid site is not finished by March 1st.
- Reminded Pete about reading through my current draft of the main report, which was forgotten about between last meeting and this one.

**C.U Trimester 2, Week 8 (05/03/2020)**

- Briefly discussed the completion of the Pyramid implementation, concluding all the web development work as part of the project.
- Discussed small changes to make to existing sections of Chapter 3 of the report based on the implementation.
- Discussed the Evaluation section of the report, to ensure appropriate links are made between the literature review and implementation to add to the flow, integrity and context of the section and report as a whole.
- Discussed creating a table to collate results in the implementation section, to then reference later as part of the evaluation section.

**C.V Trimester 2, Week 9 (No Meeting)**

No Meeting this week due to supervisor's unavailability

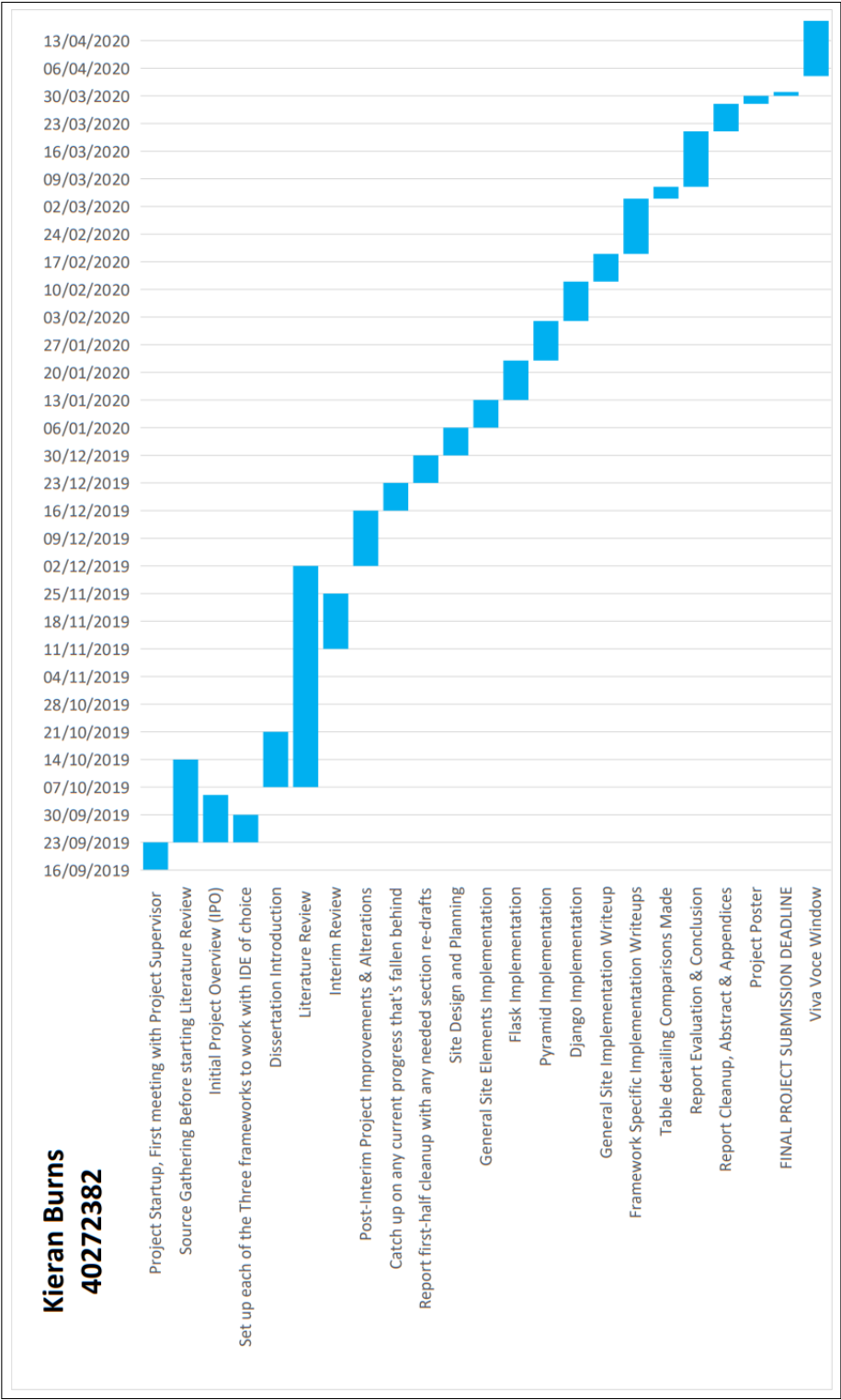
**C.W Trimester 2, Week 10 (No Meeting)**

No Meeting this week due to emergency closures following the Coronavirus outbreak.

**C.X Trimester 2, Week 11 (26/03/2020)**

- Briefly discussed project completion.
- Discussed finishing touches required for wrapping-up the project.

D Project Timeline Gantt Chart



## E Implemented Website Screenshots

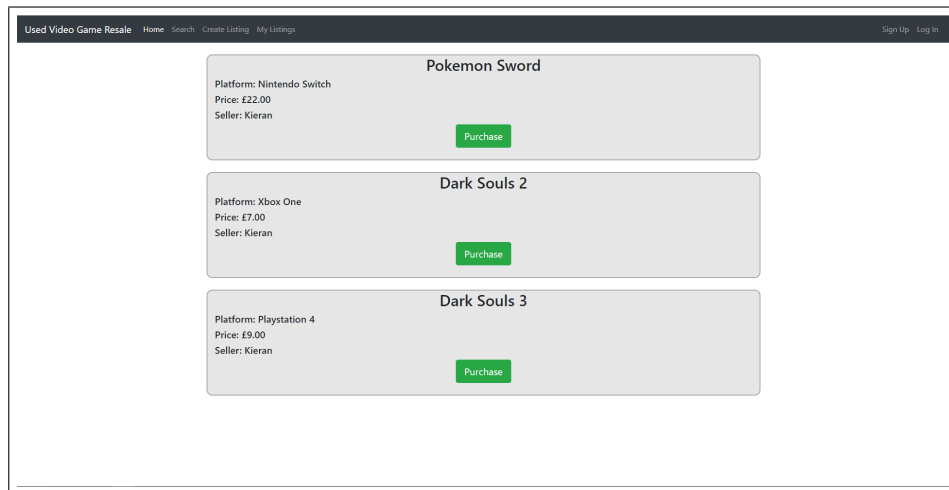


Figure 12: Home Page while not logged in



Used Video Game Resale

HomeSearchCreate ListingMy Listings

Sign UpLog In

Sign Up

Username

Username

Password

Password

Re-Enter Password

Re-Enter Password

Sign Up

Used Video Game Resale

HomeSearchCreate ListingMy Listings

Sign UpLog In

Sign Up

Username

NewUser123

Password

\*\*\*\*\*

Re-Enter Password

\*\*\*\*\*

Sign Up

Figure 13: Signing up as a new user

Used Video Game Resale

HomeSearchCreate ListingMy Listings

Sign UpLog In

Username already exists! Did you mean to log in?

Sign Up

Username

Username

Password

Password

Re-Enter Password

Re-Enter Password

Sign Up

Used Video Game Resale

HomeSearchCreate ListingMy Listings

Sign UpLog In

Passwords do not match!

Sign Up

Username

Username

Password

Password

Re-Enter Password

Re-Enter Password

Sign Up

Figure 14: Signup validation

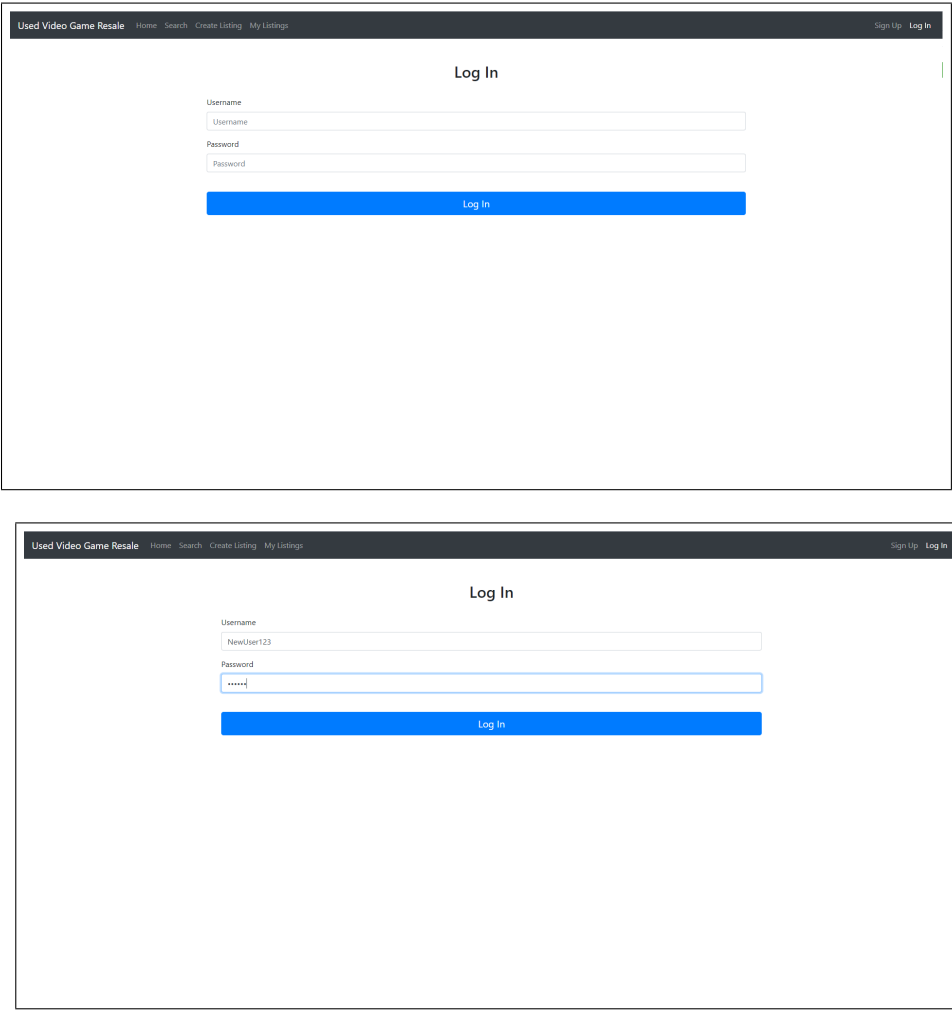


Figure 15: Logging in

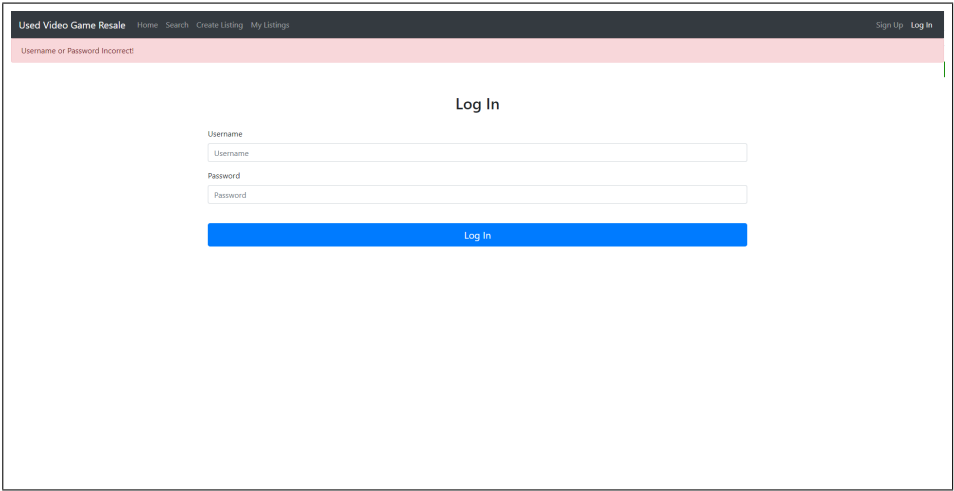


Figure 16: Login validation

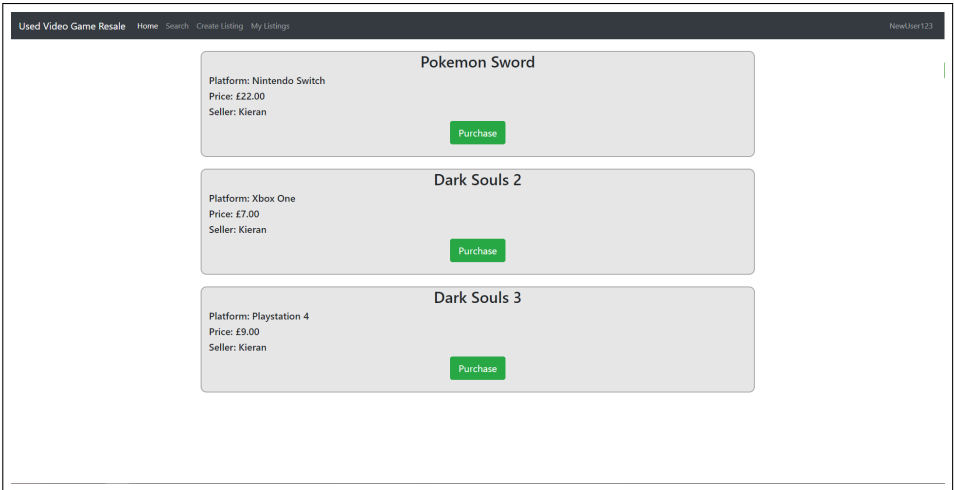


Figure 17: Home Page when logged in

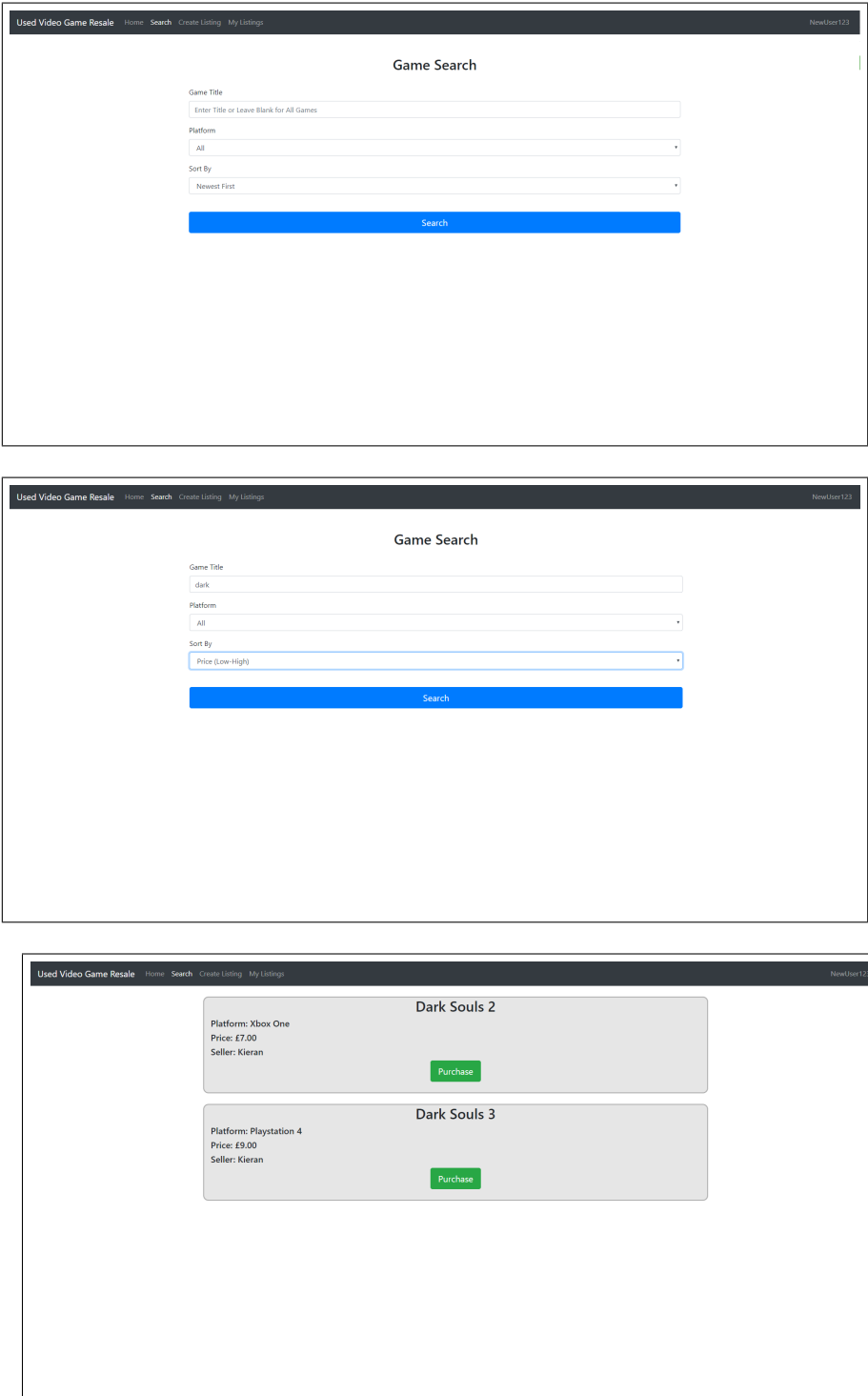


Figure 18: Searching for all games with “dark” in the title

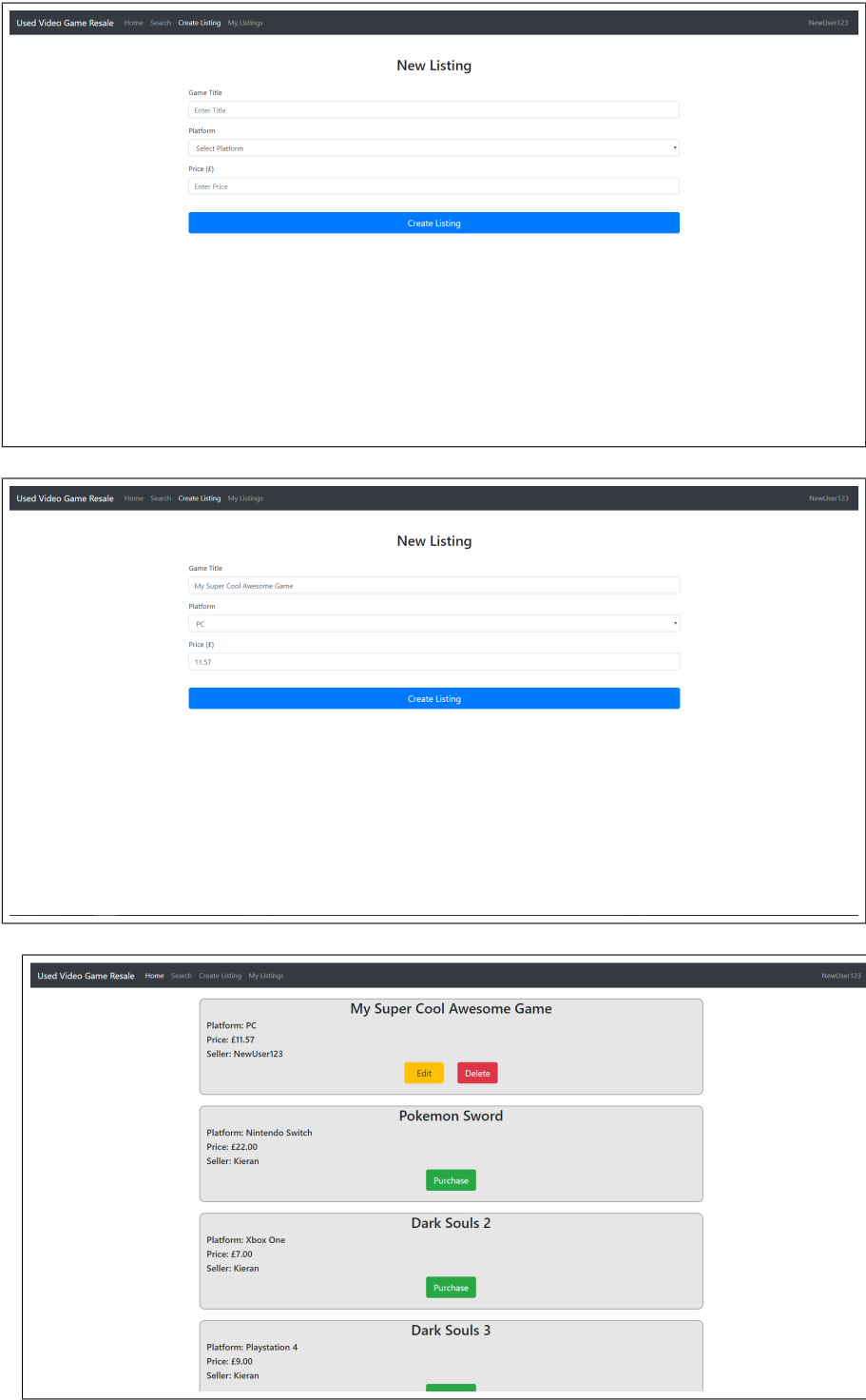


Figure 19: Creating a new listing for a game

Invalid Platform! Use the dropdown menu to select the platform.

New Listing

Game Title

Enter Title

Platform

Select Platform

Price (£)

Enter Price

Create Listing

Invalid Price! Ensure price is in the format £££.pp. (EG/ 7.00)

New Listing

Game Title

Enter Title

Platform

Select Platform

Price (£)

Enter Price

Create Listing

Figure 20: Validation on creating a new listing

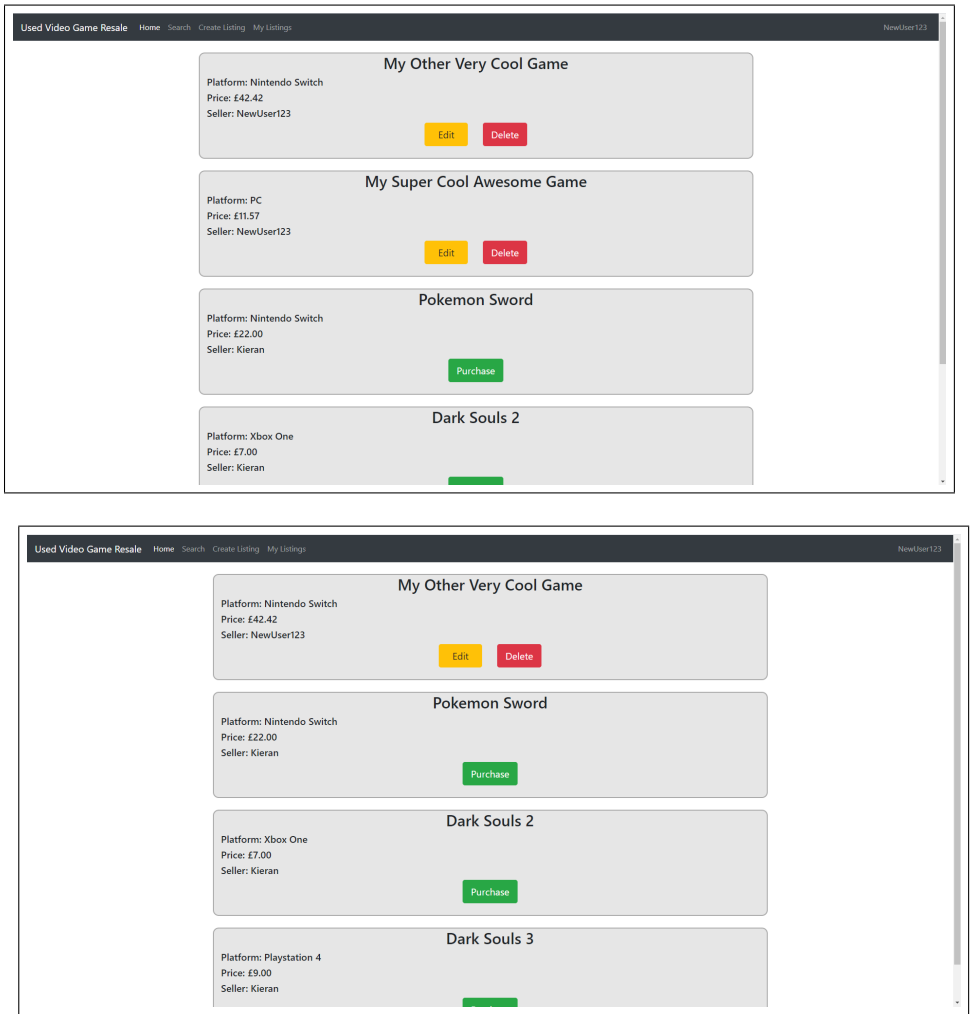


Figure 21: Deleting a listing owned by the user



The screenshot shows a web application interface for editing a listing. At the top, a dark navigation bar contains the text 'Used Video Game Resale' and links for 'Home', 'Search', 'Create Listing', and 'My Listings'. The user 'NewUser123' is logged in. The main content area is titled 'Edit Listing' and contains three input fields: 'Game Title' with the value 'My Other Very Cool Game', 'Platform' with a dropdown menu showing 'Nintendo Switch', and 'Price (£)' with the value '42.42'. Below these fields is a prominent blue button labeled 'Save Changes'.

Figure 22: Editing a listing owned by the user (uses the same validation as in **figure 20**)

The screenshot shows the 'My Listings' page. The navigation bar is identical to Figure 22. The main content area displays a single listing card for 'My Other Very Cool Game'. The card shows the platform as 'Nintendo Switch', the price as '£42.42', and the seller as 'NewUser123'. At the bottom of the card are two buttons: a yellow 'Edit' button and a red 'Delete' button.

Figure 23: My Listings page, which only displays listings created by the logged in user

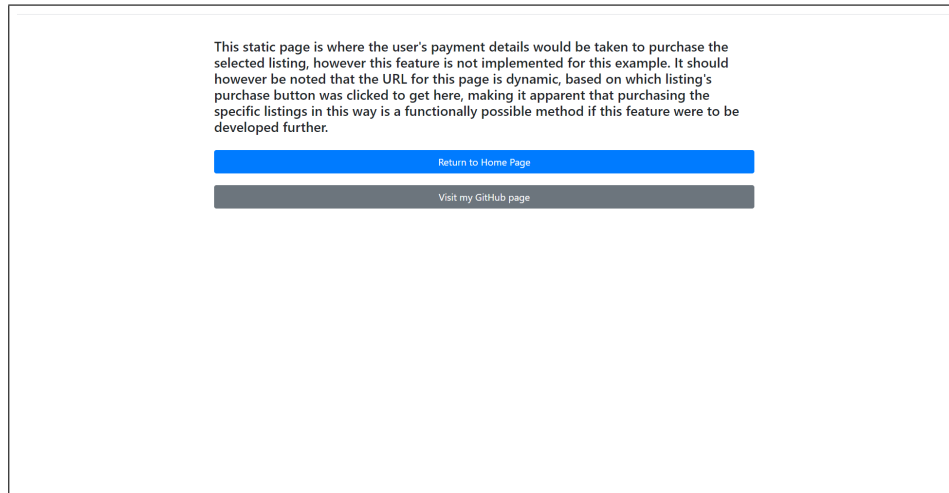


Figure 24: Clicking the Purchase button on another user's listing

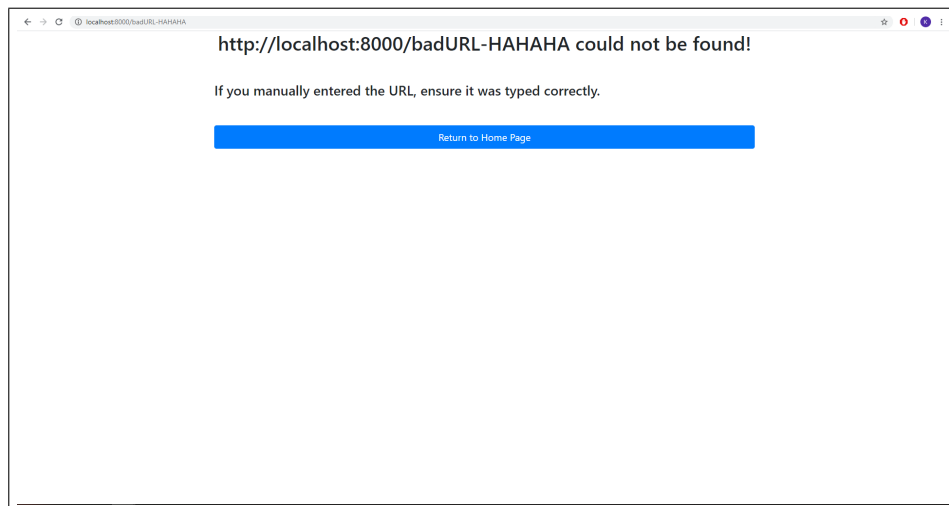
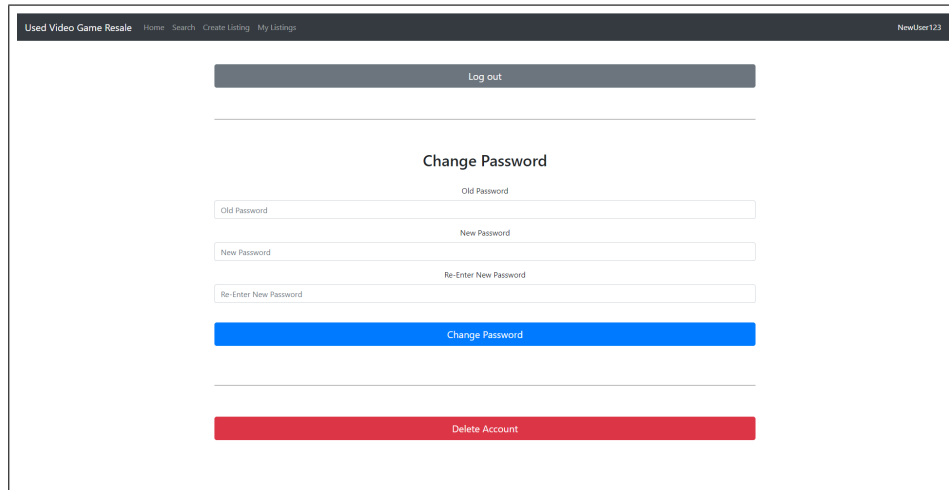
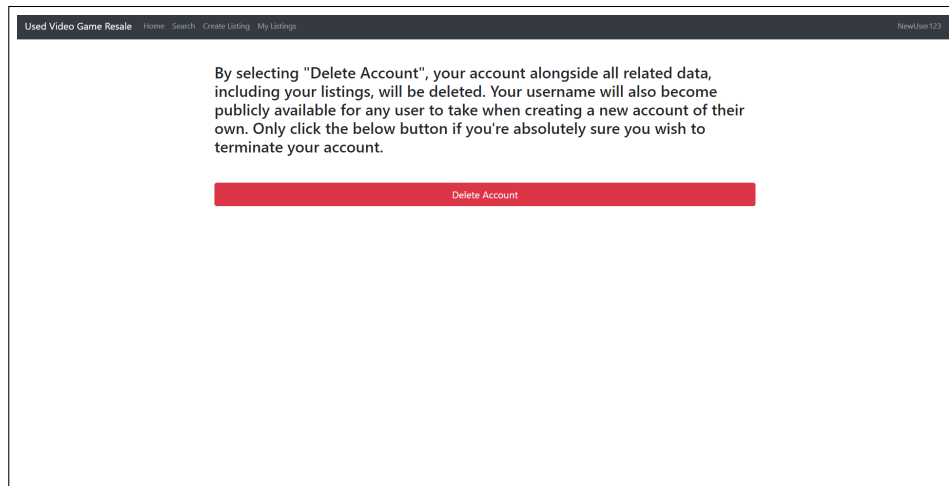


Figure 25: Manually entering an invalid URL (getting an error-404)



The screenshot shows a web application interface for a user's account. At the top, a dark navigation bar contains the text "Used Video Game Resale" on the left and "NewUser123" on the right. Below the navigation bar, a grey button labeled "Log out" is centered. The main content area is titled "Change Password" in bold. Below the title, there are four input fields arranged in two rows: "Old Password", "New Password", and "Re-Enter New Password". A blue button labeled "Change Password" is positioned below the input fields. At the bottom of the form, a red button labeled "Delete Account" is visible.

Figure 26: Logged in user's account page



The screenshot shows a confirmation page for deleting an account. At the top, a dark navigation bar contains the text "Used Video Game Resale" on the left and "NewUser123" on the right. The main content area contains a paragraph of text: "By selecting 'Delete Account', your account alongside all related data, including your listings, will be deleted. Your username will also become publicly available for any user to take when creating a new account of their own. Only click the below button if you're absolutely sure you wish to terminate your account." Below the text, a red button labeled "Delete Account" is centered.

Figure 27: Confirm page after clicking "delete account"

Used Video Game Resale

HomeSearchCreate ListingMy Listings

NewUser123

Log out

Old Password is incorrect!

Change Password

Old Password

New Password

Re-Enter New Password

Change Password

Delete Account

Used Video Game Resale

HomeSearchCreate ListingMy Listings

NewUser123

Log out

Passwords do not match!

Change Password

Old Password

New Password

Re-Enter New Password

Change Password

Delete Account

Figure 28: Validation on Change Password option