Interim Report:
Exploring Web Development with Python: A Comprehensive Study with Applications (Blog) Built in Flask and Django

Julia Gongala
B00402569

School of Computing, Engineering
and Physical Sciences

BSc (Honours) Web and Mobile Development
University of the West of Scotland

Supervisor: Tony Gurney
Moderator: Pablo Salva Garcia

Contents

Table of Figures

Table of Appendices

Abstract

In the fast-paced evolution of web development, Python has emerged as a versatile and powerful language, garnering attention for its simplicity, readability, and extensive ecosystem. This work embarks on a comprehensive exploration of Python's role in web development, with a focus on its frameworks, libraries, and practical applications. As businesses increasingly demand scalable and efficient web solutions, understanding the nuances of Python becomes imperative. Through this study, we aim to unravel the intricacies of Python's contributions to web development, examining its frameworks like Django and Flask, and elucidating real-world implementations. By delving into the language's integration with front-end technologies, we seek to provide a holistic understanding of Python's position in the dynamic landscape of contemporary web development.

1. Overview

The history of programming languages is a fascinating journey that reflects the evolution of technology and the changing needs of the computing industry. In the early days of computing during the 1940s and 1950s, programmers interacted directly with machine code and assembly languages, writing instructions that aligned with the hardware's architecture. However, as the complexity of tasks increased, the need for more abstraction and efficiency became evident. During these years, numerous programming languages have emerged, each with its own characteristics and purposes.

Some languages, such as JavaScript, Python, Java, C, and C++, have stood the test of time, becoming essential tools for computer scientists and programmers. These languages still serve as the foundation for applications across various platforms, from small utilities to large-scale enterprise systems on desktops, mobile devices, and handheld platforms.

Frameworks play a crucial role in modern programming, providing developers with pre-built structures and tools to streamline the development process. These frameworks, each with its unique syntax and architecture, offer a higher level of abstraction, allowing programmers to focus on application-specific logic rather than dealing with low-level details. Navigating through different frameworks can pose a learning curve, but the adaptability of programmers becomes paramount in leveraging the advantages that each framework brings to the table.

Collaboration among programmers, especially in diverse teams, is a multifaceted challenge that demands clear communication and unified practices. The choice of frameworks becomes pivotal in this context, as aligning them with team expertise and project requirements is essential for efficient collaboration. Integrating disparate frameworks within a project adds complexity, requiring seamless communication channels and a shared understanding of chosen frameworks to ensure a cohesive and productive development environment.

This intricate interplay of challenges and solutions in the programming landscape sets the stage for my thesis, 'Exploring Web Development with Python: A Comprehensive Study with Applications (Blog) Built in Flask and Django.' This research delve into the realm of web development with a comprehensive exploration of Python, a versatile programming language that as could be seen in Figure 1, Python has gained substantial popularity in recent years.

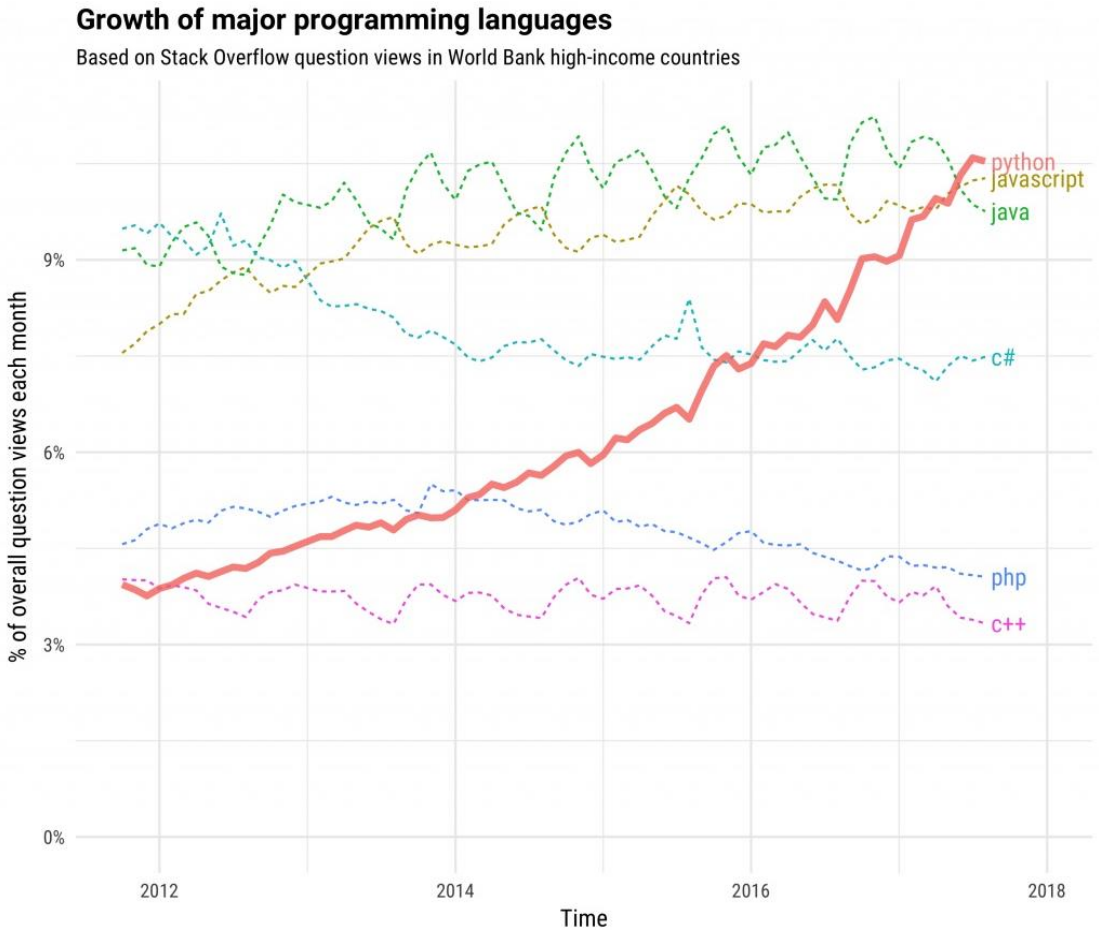


Figure 1: Growth of major programming languages (StackOverflow blog, 6/09/2017)

Focusing on two prominent web frameworks, Flask and Django, I conduct an in-depth study to elucidate their distinct features, strengths, and applications. Through practical examples and the development of a blog, I demonstrate how these frameworks facilitate the creation of dynamic and robust web applications. The choice of Flask, known for its simplicity and flexibility, and Django, celebrated for its batteries-included approach, allows for a comparative analysis, offering insights into the trade-offs involved in selecting a framework based on project requirements. Additionally, I address the challenges and benefits of integrating these frameworks into collaborative web development projects, emphasizing the importance of clear communication and shared understanding among team members. This study aims to contribute valuable insights for programmers and developers seeking to navigate the evolving landscape of web development using Python and its associated frameworks.

2. Literature Review

2.1 Background of web application development

Web application development involves creating and maintaining software applications that are accessed through web browsers. It encompasses both the front end and back end components, each playing a crucial role in delivering a seamless and interactive user experience. Let's delve into the details

Front – End

The front end, also known as the client-side, is the user interface and user experience layer of a web application. Figure 2 presents how Front-end communicates with user and Back-end. It includes everything that users interact with directly, such as buttons, forms, and visual elements. Front-end development primarily involves HTML, CSS, and JavaScript to create responsive and visually appealing interfaces.

Front-end technologies include HTML (Hypertext Markup Language), CSS (Cascading Style Sheets), and JavaScript. HTML structures the content, CSS styles the layout, and JavaScript adds interactivity.

The rise of front-end frameworks, such as React, Angular, and Vue.js, has streamlined development processes and enhanced the performance of web applications. JavaScript libraries and frameworks enable developers to build interactive and dynamic user interfaces more efficiently. (Lolugu, 2023)

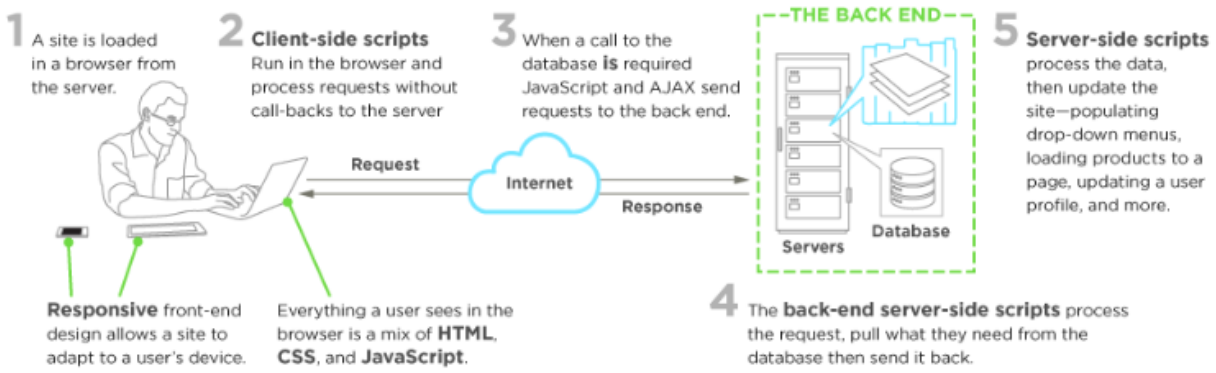


Figure 2: How Front-end development works (frontendmasters.com, n/d)

Back - End

Back-end development, or server-side development, involves creating the server, database, and application logic that enable the front end to function. It focuses on managing and storing data, as well as handling business logic and server-side operations, and this could be seen on Figure 3.

Back-end technologies encompass programming languages such as Python, Ruby, Java, and PHP, as well as server technologies like Node.js. Databases, such as MySQL, PostgreSQL, and MongoDB, are integral components for storing and retrieving data.

Martin Fowler, a renowned software developer and author, "Any fool can write code that a computer can understand. Good programmers write code that humans can understand." (Martin Fowler, 1999). This highlights the importance of writing clear and maintainable code in the back end to ensure the scalability and long-term viability of a web application.

Frameworks like Django (Python), Ruby on Rails (Ruby), and Express.js (Node.js) facilitate back-end development by providing pre-built modules and structures. These frameworks streamline the development process and adhere to best practices, enhancing the efficiency and reliability of the back end. (Rose, S. 2020)

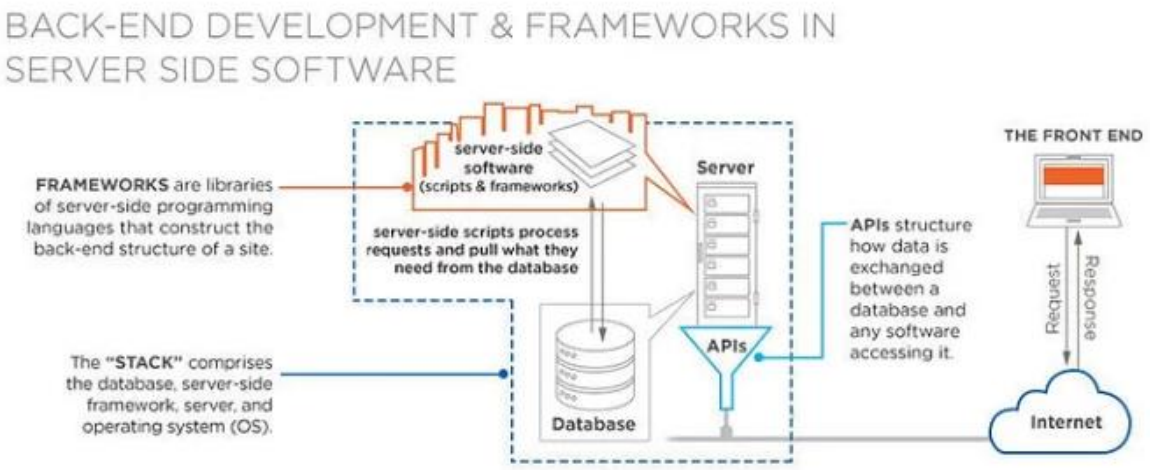


Figure 3: How Back-end development works (codeburst.io, 29/07/2020)

Integration and Collaboration

Successful web application development requires seamless collaboration between front-end and back-end developers. As Tim Berners-Lee, the inventor of the World Wide Web, aptly puts it, "The Web is more a social creation than a technical one. I designed it for a social effect—to help people work together". (Berners-Lee, 1999)

Application Programming Interfaces (APIs) act as the bridge between the front end and back end, enabling them to communicate and exchange data. This collaboration ensures that user interactions on the front end trigger the necessary processes and data manipulations on the back end.

In conclusion, web application development is a multifaceted process that involves both front-end and back-end development. The collaboration between these two components is essential for creating a cohesive, functional, and user-friendly web application. As technology continues to evolve, the field of web development remains dynamic, requiring developers to stay informed and adapt to emerging trends and best practices. (Sharma, R. 2023)

2.2 Front -end Technologies

HTML

One of the fundamental technologies in front-end development is HTML (HyperText Markup Language) created by Berners – Lee in 1989. As eloquently described by Jon Duckett in "HTML and CSS: Design and Build Websites," HTML is the backbone of web content, HTML provides the structure for web pages, giving them a logical and organized format that can be interpreted by web browsers. Duckett's insights underscore the foundational role of HTML in organizing information for effective presentation.

HTML uses a system of elements, tags, and attributes to annotate text within a document. These annotations define the structure of the content, such as headings, paragraphs, lists, images, links, and more. The markup is interpreted by web browsers to render the content in a visually organized manner.

HTML is governed by the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG), which establish and maintain the specifications for HTML. Adhering to these standards ensures consistency and compatibility across different browsers.

HTML works seamlessly with other web technologies, such as CSS (Cascading Style Sheets) for styling and layout, and JavaScript for dynamic behaviour. This trio HTML, CSS, and JavaScript forms the core technologies for building modern, interactive web applications.

In summary, HTML serves as the backbone of web content, providing the essential structure and organization for effective presentation. Understanding HTML is foundational for anyone involved in web development, whether they specialize in front-end design or work with more extensive full-stack applications. (Duckett, J. 2011).

CSS

CSS (Cascading Style Sheets) is another key front-end technology that contributes to the aesthetics and styling of web pages. CSS was proposed in 1994. Eric Meyer, an influential figure in the CSS community, defines that one of the fundamental principles of web development is the separation of concerns, and CSS excels in this regard. By separating content (HTML) from presentation (CSS), developers can make changes to the visual style of a website without altering the underlying structure or content. This modularity enhances code maintainability and facilitates collaboration among developers.

CSS is designed to be compatible with various web browsers, and modern browsers follow the specifications set by the W3C. However, achieving consistent cross-browser compatibility can sometimes be challenging, and developers may need to use vendor prefixes or employ other techniques to address browser-specific issues

In summary, CSS empowers developers to control the visual presentation of web pages, allowing for the creation of aesthetically pleasing and responsive user interfaces. The separation of content and presentation, as advocated by CSS, is a key principle in maintaining clean, modular, and maintainable code in web development projects. (Meyer, E. 2017)

JavaScript

JavaScript, commonly recognized as the language of the web, made its debut in 1995 with the aim of incorporating interactivity and dynamic behaviour into web pages. In "Eloquent JavaScript," Marijn Haverbeke notes JavaScript is the only language that can be run by all major web browsers, making it an essential tool for front-end developers. Its versatility and ubiquity make it a cornerstone technology for creating responsive and interactive user interfaces.

Over the years, JavaScript has evolved significantly, with the introduction of ECMAScript standards, bringing new features and capabilities to the language. The development of popular JavaScript libraries and frameworks, such as jQuery, React, Angular, and Vue.js, has further expanded the possibilities for building sophisticated web applications.

One of the notable characteristics of JavaScript is its ability to run both on the client-side and server-side. With the advent of server-side JavaScript frameworks like Node.js, developers can use a single programming language for building entire web applications, streamlining the development process and fostering code reuse.

JavaScript's asynchronous nature, facilitated by features like Promises and Async/Await, enables the creation of responsive and efficient web applications. This is particularly crucial for handling tasks such as fetching data from servers, ensuring a seamless user experience.

In addition to its dominance on the client-side, JavaScript is increasingly becoming relevant in the context of serverless computing and the development of progressive web applications (PWAs). Its integration with technologies like WebAssembly allows developers to leverage high-performance computing in the browser, opening up new possibilities for web applications.

In summary, JavaScript's enduring popularity and continuous evolution have solidified its position as a fundamental language for front-end development. Its ability to deliver interactivity, along with its versatility across both client and server environments, makes it an indispensable tool for creating modern and feature-rich web applications. (Haverbeke, M. 2018).

2.3 Back-end Technologies

PHP

PHP, which stands for "Hypertext Preprocessor," is a server-side scripting language that is widely used for web development. It was originally created by Rasmus Lerdorf in 1994 and has since evolved into a powerful and versatile language for building dynamic web applications. The development of PHP has been a collaborative effort, with contributions from a large community of developers worldwide.

PHP is widely used for building the server-side of web applications. It excels in handling HTTP requests, interacting with databases, and generating dynamic content. Its simplicity, combined with a vast ecosystem of extensions and frameworks like Laravel, Symfony, and WordPress, makes it a popular choice for web development.

It's important to note that PHP's development is an open-source effort with contributions from a diverse community of developers. The official PHP website (php.net) serves as a central hub for documentation and community discussions. The language continues to evolve, adapting to changing web development needs.(O'Reilly, M. 2020)

Ruby

Ruby is a dynamic, object-oriented programming language that has gained popularity for its simplicity and flexibility. It was created by Yukihiro "Matz" Matsumoto in the mid-1990s.

Ruby has become a favoured choice for backend development due to its clean syntax, object-oriented nature, and support for metaprogramming. Matz, the language's creator, designed Ruby with the philosophy of optimizing for programmer satisfaction, aiming to make coding not only efficient but also enjoyable. This emphasis on human-centric design has contributed to Ruby's widespread adoption. (Ruby)

The support for metaprogramming in Ruby is a powerful feature that allows developers to write code that can dynamically modify itself at runtime. This capability enables the creation of flexible and expressive solutions, facilitating the development of frameworks and libraries that leverage dynamic behaviours. Ruby on Rails, arguably the most prominent example of this, has revolutionized web development by providing a convention-over-configuration approach. This framework automates many aspects of web application development, allowing developers to focus more on the application's logic rather than spending time on repetitive configuration tasks. (Jalli, A. 2022)

Java

Java is a versatile programming language that has been widely used for backend development since its inception. Java was created by James Gosling and Mike Sheridan at Sun Microsystems and was officially released in 1995. It was designed with the principle of "Write Once, Run Anywhere" (WORA), meaning that Java programs can run on any device that has a Java Virtual Machine (JVM).

Java gained popularity for backend development due to its platform independence, strong community support, and robust features. It is often used to build large-scale, enterprise-level applications, and web services. (Obregon, A. 2023)

Java is particularly well-suited for backend development because of its features like multithreading, scalability, and the extensive set of libraries (such as Java EE, now Jakarta EE) that facilitate the development of robust and scalable server-side applications.

Java's long history, platform independence, and extensive ecosystem have contributed to its continued relevance in the world of backend development. It remains a popular choice for building robust and scalable server-side applications. (Niemeyer, P. and Knudsen, J. 2002)

C++

C++ is a general-purpose programming language created by Bjarne Stroustrup at Bell Labs in the early 1980s. It is an extension of the C programming language with object-oriented programming features. C++ was officially released in 1985.

C++ has gained widespread adoption in system-level programming and backend development due to its unique features and capabilities. Its close interaction with hardware and support for low-level memory manipulation make it well-suited for tasks that require optimization, control over resources, and efficiency. (Stroustrup, B. n.d.)

C++ continues to be a versatile language with a significant impact on system-level and backend development, providing a balance between performance and high-level abstractions. It is valued for its capability to build efficient and reliable backend systems across various domains.

2.4 Python

Python, renowned for its readability and adaptability, has emerged as a popular preference for back-end development. This is due to its diverse array of libraries and tools that significantly contribute to the development of robust server-side applications. Guido van Rossum, the visionary behind Python, founded the language in 1991 and recognized its widespread appeal (Van Rossum, G. 2009). Van Rossum aimed to design Python as a programming language that seamlessly combined user-friendly features with the robust capabilities inherent in more complex alternatives. His goal was to make the language accessible to beginners by embracing open-source principles and utilizing straightforward language constructs. Above all, van Rossum aspired to bring an element of enjoyment to computing. The name "Python" was inspired by the legendary British comedy group Monty Python (Enki Team, 2022).

In the realm of back-end development, Python's success can be attributed to its adaptability and integration capabilities. The language is adept at seamlessly interfacing with databases, facilitating efficient data management. As Alex Martelli, a Fellow at the Python Software Foundation, describes that Python's syntax and idioms are minimalist, but they can be extended to meet virtually all demands on a programmer. This flexibility enables developers to tailor solutions to the specific requirements of back-end systems. (Martelli, A. 2003)

Python's support for asynchronous programming has also played a pivotal role in its effectiveness for back-end tasks. With the advent of features like async/await, Python enables the development of scalable and responsive server-side applications. David Beazley, author of "Python Essential Reference", emphasized the significance of asynchronous programming in Python, suggesting, that asynchronous programming is becoming increasingly important as more and more services move to the web.

Furthermore, Python's extensive standard library contributes to its standing as a robust back-end technology. The inclusion of modules for tasks such as networking, file handling, and data serialization streamlines the development process. As Mark Lutz, author of "Learning Python: Powerful Object-Oriented Programming" pointed out that Python's support modules that make it easy for Python programs to do things that would be very complex in other languages. (Lutz, M. 2013)

In conclusion, Python's ascendancy in back-end technology is underpinned by its readability, versatility, and extensive standard library. The language's adaptability to diverse tasks, from database interaction to asynchronous programming, positions it as a formidable choice for developers crafting sophisticated server-side solutions.

From here, we will dive into a discussion about two one of the most popular frameworks Flask and Django that has been shown on Figure 4.

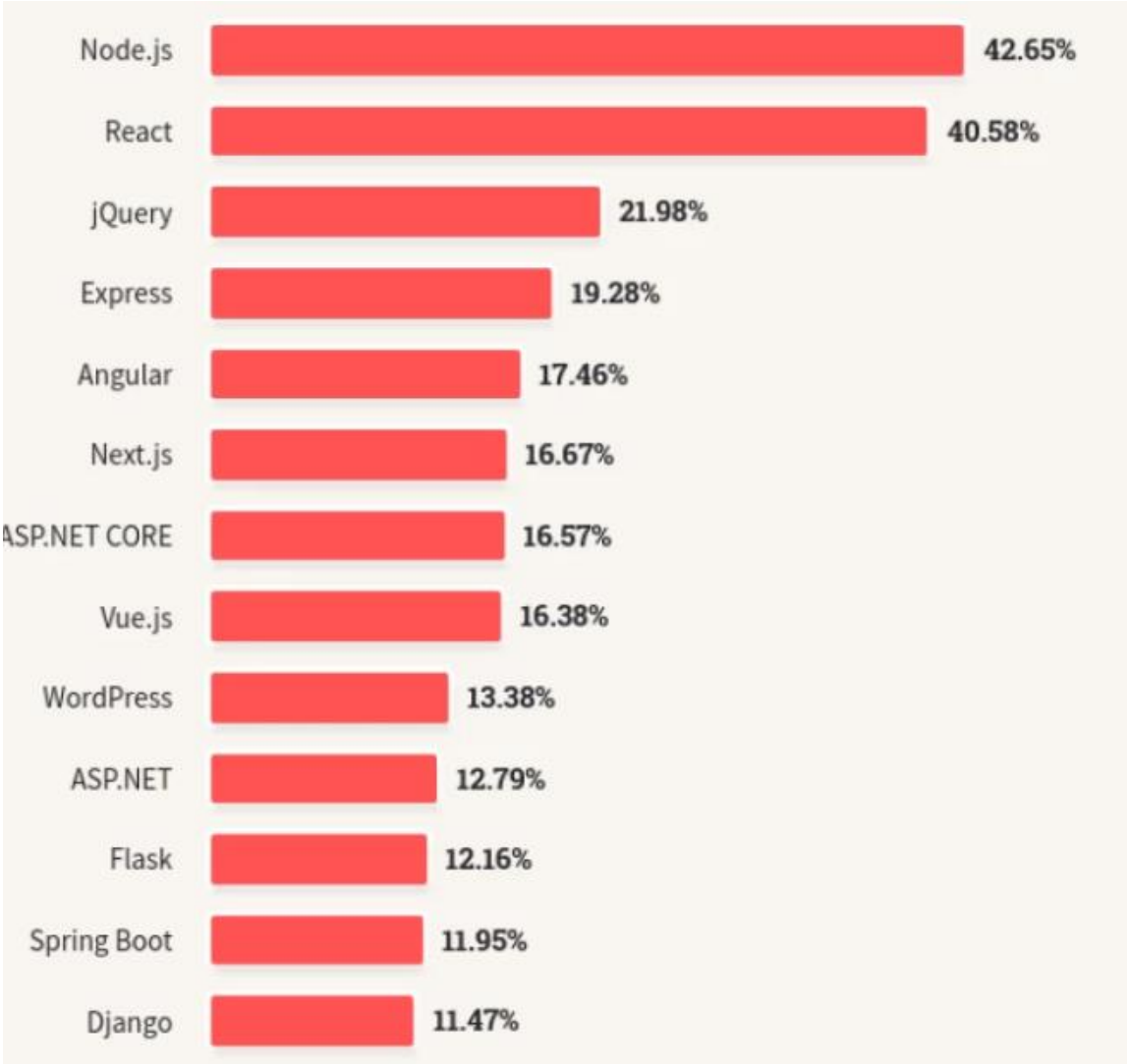


Figure 4: Django and Flask in Top 15 frameworks (survey.stackoverflow.co, 2023)

Django

Figure 5: Django official logo (djangoproject.com, n/d)

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It follows the Model-View-Controller (MVC) architectural pattern, helping developers create maintainable and scalable web applications. One of Django's key philosophies is the "Don't Repeat Yourself" (DRY) principle, promoting code reusability and reducing redundancy. (Vincent, W. S. 2023).

The framework emphasizes the use of reusable components, known as "apps," which are self-contained modules that can be easily integrated into different projects. This modular approach enhances code organization and fosters collaboration among developers. As Adrian Holovaty and the creator of Django, state in "The Definitive Guide to Django" (Holovaty, A., & Kaplan-Moss, J. 2009), Django encourages clean, pragmatic design; rapid development; and a clean, pragmatic approach to software architecture.

Django provides an Object-Relational Mapping (ORM) system that simplifies database interactions by allowing developers to work with database models using Python code. This abstraction enables the use of various database backends without altering the application's code significantly. In the book of Daniel Greenfeld and Audrey Roy Greenfeld in "Two Scoops of Django" it is explained that the Django ORM helps you interact with your database by providing a Pythonic interface to your data. (Greenfeld, D. R., & Roy Greenfeld, A. 2021),

The framework also includes a powerful templating engine that facilitates the separation of logic and presentation. This separation enhances code readability and maintainability. As Django's documentation states, “Django template system is not simply Python embedded into HTML. This is by design: the template system is meant to express presentation, not program logic”. (Django. n/d). It is a full-fledged programming language designed to be used by humans to express presentation logic in a way that separates it from the HTML. This approach is aligned with Django's goal of creating clean and readable code.

Django's built-in administrative interface is another notable feature, providing an out-of-the-box solution for managing application data. This feature is particularly useful during development and testing phases. According to "Django for Beginners" by William S. Vincent (Vincent, W. S. 2023), Django includes a 'batteries-included' admin interface that is easy to use and can be customized for different models.

In conclusion, Django's design principles, modular architecture, ORM system, templating engine, and administrative interface contribute to its effectiveness as a web framework. As it continues to evolve, Django remains a popular choice for developers seeking a robust and scalable solution for building web applications in Python.

Flask

Figure 6: Flask official logo (wikipedia.org, n/d)

Flask is a lightweight and versatile web framework for Python, known for its simplicity and flexibility. Developed by Armin Ronacher, Flask is designed to be easy to use while providing the necessary tools for building web applications. One of its core principles is to be unopinionated, allowing developers the freedom to choose the components they need for their projects.

In blog post "Design Decisions for Flask", Ronacher emphasizes Flask's minimalist philosophy, stating, Flask tries to remain a microframework, leaving many decisions up to the developer. This design choice empowers developers to select and integrate specific libraries and tools based on their project requirements, making Flask a pragmatic choice for a wide range of applications. (Ronacher, A. 2010).

Flask follows the WSGI (Web Server Gateway Interface) standard, making it compatible with various web servers and deployment options. This flexibility is highlighted by Miguel Grinberg in his book "Flask Web Development, 2nd Edition", where he describes that, Flask is a microframework, but it is not a toy. It is intended for professionals who want to get stuff done. (Grinberg, M. 2018).

One of Flask's notable features is its integrated development server, which simplifies the development process by providing a quick and easy way to test applications locally. As Ronacher mentions, Flask is easy to set up and get going within minutes, but it’s also powerful enough to scale up to complex applications. (Ronacher, A. 2010).

Flask's simplicity extends to its templating engine, Jinja2, which allows developers to create dynamic and modular HTML templates. In "Flask By Example", Dwyer highlights Flask's templating capabilities, stating, that the Jinja2 templating engine is used to dynamically build HTML, which allows for the creation of complex and reusable templates. (Dwyer, G. 2016).

While Flask provides the essential components for web development, it leaves room for developers to choose additional libraries and tools based on their preferences. This flexibility has contributed to Flask's popularity among developers who value a lightweight and adaptable framework for building web applications in Python. As Flask continues to evolve, its commitment to simplicity and extensibility remains central to its appeal in the Python web development ecosystem.

3. Preliminary Work

The project's inception occurred during a summer break, offering an opportunity to delve into Python as the primary programming language. A private GitHub repository was established for work storage, collaboration, and as a precautionary backup in the event of a computer crash. The initial focus was on Flask, which proved easy to set up. The simplicity of Flask's structure was quickly grasped, leading to active participation in real projects where features were added, and user interactions were improved.

A noteworthy aspect of this journey involved exploring Flask's connection with databases. Despite initial challenges, the integration of SQL Alchemy into existing projects provided valuable insights into efficient data handling and retrieval. These acquired skills played a direct role in optimizing the backend of the web applications under development.

To further illustrate the practical application of these skills, Figure 7 presents the main body of the Book Database Management System with Flask using SQL Alchemy (Yu, 2023). This figure serves as a tangible representation of the implemented concepts and showcases the effective utilization of Flask and SQL Alchemy in creating a functional database management system. The specific features and interactions depicted in the figure demonstrate how the knowledge gained during the learning process has been translated into a real-world project.

This hands-on experience with Flask and SQL Alchemy in the context of a Book Database Management System not only validates the learning journey but also provides a concrete example of the project's progress. The figure acts as a visual aid to justify the efforts invested in understanding and implementing these technologies, highlighting the practical outcomes achieved in the development of a robust and efficient web application backend.

```
from flask import Flask, render_template, request, redirect, url_for
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

##CREATE DATABASE
app.config['SQLALCHEMY_DATABASE_URI'] = "sqlite:///books.db"
# Create the extension
db = SQLAlchemy()
# initialise the app with the extension
db.init_app(app)


##CREATE TABLE
class Book(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(250), unique=True, nullable=False)
    author = db.Column(db.String(250), nullable=False)
    rating = db.Column(db.Float, nullable=False)

# Create table schema in the database. Requires application context.
with app.app_context():
    db.create_all()


@app.route('/')
def home():
    ##READ ALL RECORDS
    # Construct a query to select from the database. Returns the rows in
the database
    result = db.session.execute(db.select(Book).order_by(Book.title))
    # Use .scalars() to get the elements rather than entire rows from the
database
    all_books = result.scalars()
    return render_template("index.html", books=all_books)


@app.route("/add", methods=["GET", "POST"])
def add():
    if request.method == "POST":
        # CREATE RECORD
        new_book = Book(
            title=request.form["title"],
            author=request.form["author"],
            rating=request.form["rating"]
        )
        db.session.add(new_book)
        db.session.commit()
        return redirect(url_for('home'))
    return render_template("add.html")
```

```
@app.route("/edit", methods=["GET", "POST"])
def edit():
    if request.method == "POST":
        #UPDATE RECORD
        book_id = request.form["id"]
        book_to_update = db.get_or_404(Book, book_id)
        book_to_update.rating = request.form["rating"]
        db.session.commit()
        return redirect(url_for('home'))
    book_id = request.args.get('id')
    book_selected = db.get_or_404(Book, book_id)
    return render_template("edit_rating.html", book=book_selected)


@app.route("/delete")
def delete():
    book_id = request.args.get('id')

    # DELETE A RECORD BY ID
    book_to_delete = db.get_or_404(Book, book_id)
    # Alternative way to select the book to delete.
    # book_to_delete = db.session.execute(db.select(Book).where(Book.id
== book_id)).scalar()
    db.session.delete(book_to_delete)
    db.session.commit()
    return redirect(url_for('home'))


if __name__ == "__main__":
    app.run(debug=True)
```

Figure 7: Book Database Management System with Flask (Yu, 2023)

New decision was made to explore Django, a more comprehensive web framework. The organized project structure and integrated components within Django significantly facilitated the development process. Notably, working with Django models and configuring the admin interface proved transformative, particularly in efficiently managing content for the websites under construction.

The transition to Django's views and URL routing system marked a substantial departure from Flask, emphasizing a more structured approach to handling HTTP requests. Despite differences from Flask's Jinja, the Django template language demonstrated its prowess in rendering dynamic content.

The integration of Django into real-world projects at work became more seamless. The utility of built-in features such as form handling and user authentication systems proved invaluable in creating secure and user-friendly web applications. The initially overwhelming Django documentation gradually transformed into a reliable guide, aiding in navigating diverse challenges.

Towards the conclusion of the summer, experimentation with frontend frameworks commenced, introducing a layer of sophistication to the applications. The synergy between Django's backend capabilities and modern frontend technologies like Bootstrap and JavaScript libraries became evident in the refined and responsive interfaces developed. Noteworthy progress was made through various resources and online tutorials available on platforms such as YouTube and Udemy (Yu, 2023 and Pawar, 2023).

Figure 8 provides a visual representation of the main view and logic of the Expense Tracker app developed using Django. This figure serves as a justification for the acquired learning, showcasing the tangible application of Django in a real-world context. (Pawar, 2023)

```
from django.shortcuts import render,redirect
from .forms import ExpenseForm
from .models import Expense
import datetime
from django.db.models import Sum
# Create your views here.
def index(request):
    if request.method =="POST":
        expense = ExpenseForm(request.POST)
        if expense.is_valid():
            expense.save()

    expenses = Expense.objects.all()
    total_expenses = expenses.aggregate(Sum('amount'))

    #Logic to calculate 365 days expenses
    last_year = datetime.date.today() - datetime.timedelta(days=365)
    data = Expense.objects.filter(date__gt=last_year)
    yearly_sum = data.aggregate(Sum('amount'))

    #Logic to calculate 30 days expenses
    last_month = datetime.date.today() - datetime.timedelta(days=30)
    data = Expense.objects.filter(date__gt=last_month)
    monthly_sum = data.aggregate(Sum('amount'))

     #Logic to calculate 7 days expenses
    last_week = datetime.date.today() - datetime.timedelta(days=7)
    data = Expense.objects.filter(date__gt=last_week)
    weekly_sum = data.aggregate(Sum('amount'))

    daily_sums =
Expense.objects.filter().values('date').order_by('date').annotate(sum=Sum('
amount'))
```

```
    categorical_sums =
Expense.objects.filter().values('category').order_by('category').annotate(s
um=Sum('amount'))
    print(categorical_sums)

    expense_form = ExpenseForm()

    return
render(request,'myapp/index.html',{'expense_form':expense_form,'expenses':e
xpenses,'total_expenses':total_expenses,'yearly_sum':yearly_sum,'weekly_sum
':weekly_sum,'monthly_sum':monthly_sum,'daily_sums':daily_sums,'categorical
_sums':categorical_sums})

def edit(request,id):
    expense = Expense.objects.get(id=id)
    expense_form = ExpenseForm(instance=expense)
    if request.method =="POST":
        expense = Expense.objects.get(id=id)
        form = ExpenseForm(request.POST,instance=expense)
        if form.is_valid():
            form.save()
            return redirect('index')

    return render(request,'myapp/edit.html',{'expense_form':expense_form})

def delete(request,id):
    if request.method =='POST' and 'delete' in request.POST:
        expense = Expense.objects.get(id=id)
        expense.delete()
    return redirect('index')
```

Figure 8: Main view and logic for Expense Tracker App(Pawar, 2023)

4. Current progress and Future work

A Raspberry Pi 400 was required with the intention of transforming it into a web server for developmental purposes. The configuration of the Raspberry Pi was tailored specifically for web server functionalities. It is noteworthy that a change in perspective occurred regarding the hosting platform for deployment. While initially considering options such as Heroku and AWS the decision was made to leverage the capabilities of the Raspberry Pi as a self-contained web server. This choice was influenced by the desire for a localized and independent hosting solution, aligning with the project's evolving requirements. However, as the project progressed, an alternative perspective was introduced, suggesting the exploration of utilizing a Virtual Machine (VM) instead of the Raspberry Pi for hosting the web server. This shift in consideration was prompted by the potential benefits of a VM, such as scalability, ease of management, and the ability to run multiple isolated instances. Further research will be conducted to thoroughly evaluate the advantages and disadvantages of both the Raspberry Pi and Virtual Machine options, ensuring that the final decision aligns seamlessly with the project's evolving requirements.

Computing Honours

Tasks	Start Date	Days Needed	Completion	Adjusted Length
Project Specification Form	12/09/23	14	17/10/23	35
Planning Phase	17/10/23	30	17/11/23	31
Literature review	01/10/23	47	17/11/23	47
Interim Report	17/11/23	14	25/11/23	8
Development Phase	01/12/23	61		
Presentation Preparation	31/01/24	14		
Testing and Refinement	31/01/24	15		
Conclusions	15/02/24	15		
Final Report	01/03/24	19		

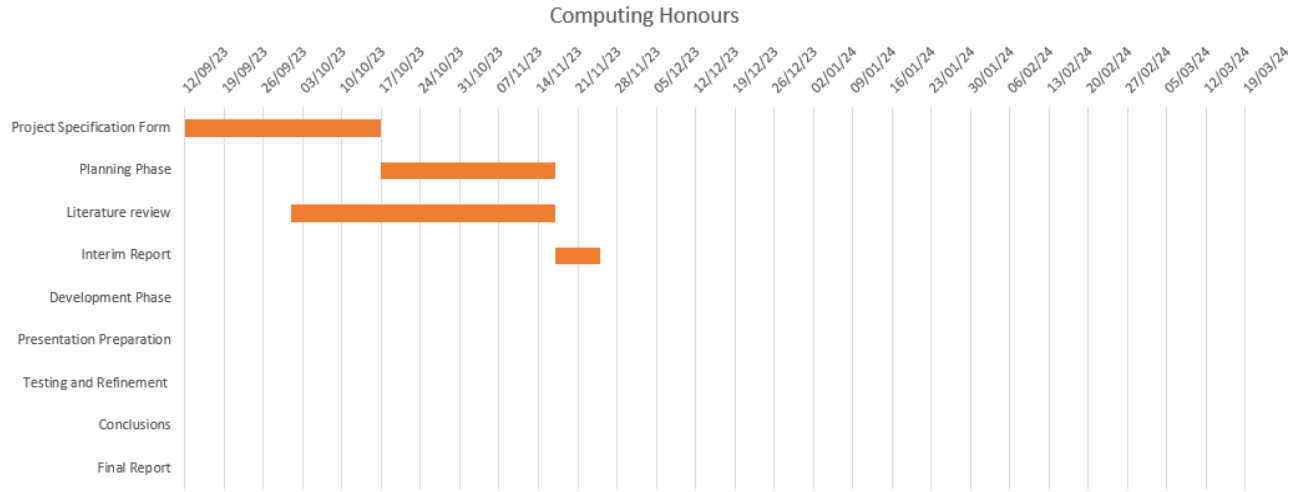
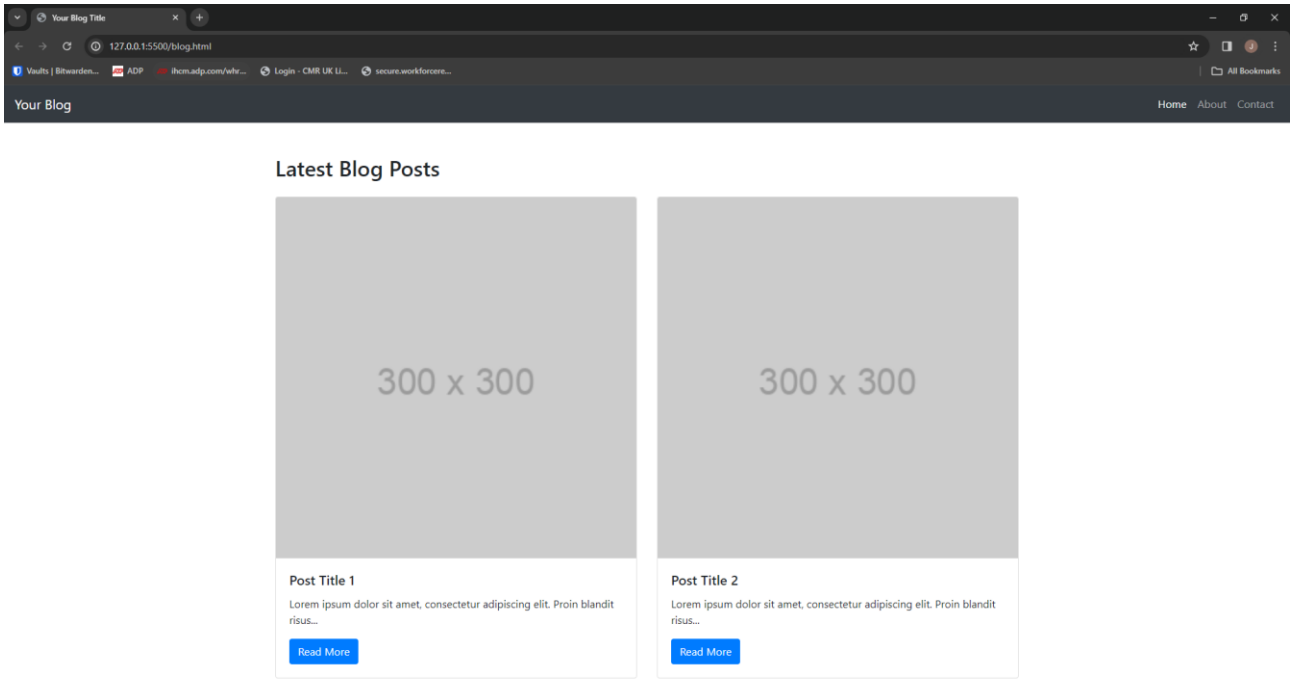


Figure 9: Gannt Chart

During project, various challenges surfaced, as depicted in the accompanying Figure 9, revealing delays in deadlines. The regular schedule has been significantly impacted. Consequently, the work has experienced delays as a result of circumstances. Active efforts are underway to mitigate the impact of these challenges, and understanding is appreciated during this period. is this be in objective style of third person writing.

The recent development phase of blog projects has been marked by significant achievements. Figure 10 presents the creation of template serves as a foundational cornerstone, providing a structured and consistent framework for blog content and body. This meticulous approach ensures that your posts maintain a professional and cohesive quality, contributing to a positive user experience for audience.




```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Your Blog Title</title>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.
css">
    <!-- Custom CSS -->
    <style>
        body {
            padding-top: 56px; /* Adjust according to your navbar height */
        }
    </style>
</head>
<body>
```

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">
    <a class="navbar-brand" href="#">Your Blog</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false"
aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav ml-auto">
            <li class="nav-item active">
                <a class="nav-link" href="#">Home <span class="sr-
only">(current)</span></a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">About</a>
            </li>
            <li class="nav-item">
                <a class="nav-link" href="#">Contact</a>
            </li>
        </ul>
    </div>
</nav>
<!-- Main Content -->
<div class="container mt-5">
    <h2 class="mb-4">Latest Blog Posts</h2>
    <!-- Blog Post List -->
    <div class="card-deck">
        <!-- Example Blog Post 1 -->
        <div class="card">
            <img src="https://via.placeholder.com/300" class="card-img-
top" alt="...">
            <div class="card-body">
                <h5 class="card-title">Post Title 1</h5>
                <p class="card-text">Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Proin blandit risus...</p>
                <a href="#" class="btn btn-primary">Read More</a>
            </div>
        </div>

        <!-- Example Blog Post 2 -->
        <div class="card">
            <img src="https://via.placeholder.com/300" class="card-img-
top" alt="...">
            <div class="card-body">
                <h5 class="card-title">Post Title 2</h5>
                <p class="card-text">Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Proin blandit risus...</p>
                <a href="#" class="btn btn-primary">Read More</a>
            </div>
        </div>
    </div>
</div>
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
<script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.3/dist/umd/popper.min
.js"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.j
s"></script>
</body>
</html>
```

Figure 10: Main blogs page – template and code

In addition to template creation, the successful setup of both projects has been achieved. During this process, differences have been observed. In the context of building a blog, these differences manifest in terms of simplicity, flexibility, and built-in features.

Flask is recognized for its simplicity and minimalism. It provides a lightweight and flexible framework that affords developers the freedom to choose their components and libraries. Following a micro-framework approach, Flask offers only the essentials necessary for web development. When building a blog with Flask, there is increased control over the components used, facilitating the customization of the application to specific needs. An illustrative example in Figure 11 presents a Flask route for displaying blog posts. is this be in objective style of third person writing.

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    # Logic to fetch and display blog posts
    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

Figure 11: Flask route

On the other hand, Django follows a full-stack web framework approach, providing a more opinionated structure and a set of built-in features. Django includes an ORM (Object-Relational Mapping) system, an admin panel, and a built-in authentication system, among other features. This can make it quicker to set up a blog with Django due to its "batteries-included" philosophy. Figure 12 presents example of a Django view for a blog:

```
from django.shortcuts import render
from .models import Post

def index(request):
    # Logic to fetch and display blog posts
    posts = Post.objects.all()
    return render(request, 'index.html', {'posts': posts})
```

Figure 12: Django route

In Figure 12, Django's ORM (assuming a Post model) handles database interactions without requiring explicit SQL queries.

In summary, it could be seen that Flask is often favoured for its simplicity and flexibility, allowing developers to choose their components. Django, with its built-in features, can expedite development but may be perceived as more heavyweight for simpler projects. The choice between Flask and

Django ultimately depends on the specific requirements and preferences of the developer or development team.

Another difference seen is in databases. Flask and Django take different approaches to database management, reflecting their overall design philosophies. Flask is a micro-framework, offering minimal built-in functionality, while Django is a full-stack web framework that comes with a robust set of features, including an ORM (Object-Relational Mapping) system.

In Flask, developers have the flexibility to choose their preferred database and ORM, or even use no ORM at all if they prefer raw SQL. Here's an example using Flask with SQLAlchemy, a popular SQL toolkit and Object-Relational Mapping (ORM) library:

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///blog.db'
db = SQLAlchemy(app)

class Post(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    content = db.Column(db.Text, nullable=False)

# Create tables
db.create_all()

# Sample usage
@app.route('/')
def index():
    posts = Post.query.all()
    # Logic to display blog posts
```

Figure 13: Flask App with SQLAlchemy Model for Blog Posts

In Figure 13, SQLAlchemy is used as the ORM, and a simple Post model is defined, representing a blog post. Developers have the freedom to choose different databases (e.g., PostgreSQL, MySQL) and customize the schema according to their requirements.

Django, on the other hand, comes with its own ORM, making it easier to manage databases without the need for additional configuration. Here is an example of defining a model in Django for a blog post see Figure 14:

```python
from django.db import models

class Post(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()

# Sample usage
def index(request):
    posts = Post.objects.all()
    # Logic to display blog posts
```

Figure 14: Django Model for Blog Posts and Sample Usage

The Django ORM abstracts away much of the database interaction, providing a higher-level API for creating, querying, and managing database records. It also includes features like automatic database migrations, simplifying the evolution of the database schema over time.

In summary, Flask offers more flexibility in choosing databases and ORMs, allowing developers to select tools that best fit their needs. Django, with its built-in ORM, streamlines the database management process, making it quicker to set up and use in projects like a blog. The choice between Flask and Django for database management depends on the developer's preference for flexibility versus convenience.

Looking ahead, the plan involves the simultaneous development of blogs using both Flask and Django with the basics of CRUD operations. Rigorous testing of the functionality of both frameworks will provide a nuanced understanding of their respective strengths and weaknesses in the context of blog development.

As the foundation of the blog development and literature review took shape, attention naturally shifted towards conducting a comparative analysis of Flask and Django. The focus is on understanding how each framework approaches the task at hand, considering factors such as ease of use, scalability, and community support. This meticulous examination serves as the basis for making an informed decision on which framework to choose as the backbone for the advanced blog project.

After carefully weighing the pros and cons of both frameworks, a pivotal decision will be made to move forward with one selected framework. Based on the gained knowledge so far, Django might be the better option.

Reference

Robinson, D. (2017). 'The Incredible Growth of Python', Stack Overflow Blog, [Online] Available at: https://stackoverflow.blog/2017/09/06/incredible-growth-python/ (Accessed: 01 November 2023).

Vardhan Lolugu, K. (2023). 'Let’s Code the User Experience', UX Design Bootcamp, [Online] at: https://bootcamp.uxdesign.cc/lets-code-the-user-experience-c47c72f50700 (Accessed: 01 November 2023).

Lindley, C. (No Date). 'What Does a Front-End Developer Do? Complete Guide to the Front-End Developer Profession', Frontend Masters. [Online] Available at: https://frontendmasters.com/guides/front-end-handbook/2018/what-is-a-FD.html (Accessed: 01 November 2023).

Fowler, M., Beck, K., Brant, J., Opdyke, W., & Roberts, D. (1999). Refactoring: Improving the Design of Existing Code. Addison-Wesley.

Rose, S. (2020). 'A Complete Guide to the Back-End Mobile App Development', codeburst, [Online]. Available at: https://codeburst.io/a-complete-guide-to-the-back-end-mobile-app-development-9609f5979231 (Accessed: 05 November 2023).

Berners-Lee, T. (1999). Weaving The Web: The Original Design and Ultimate Destiny of the World Wide Web. Harper Paperbacks.

Sharma, R. (2023). 'Best Practices for Front-End and Back-End Integration in Full Stack Development', [Online] Available at: https://medium.com/@rishani.ynr/best-practices-for-front-end-and-back-end-integration-in-full-stack-development-bbd21b36399c (Accessed: 05 November 2023).

Duckett, J. (2011). HTML & CSS: Design and Build Websites. Wiley.

Meyer, E., & Weyl, E. (2017). CSS: The Definitive Guide - Visual Presentation for the Web. O'Reilly Media.

Haverbeke, M. (2018). Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming. No Starch Press.

O'Reilly, M. (2020). Programming PHP, 4th Edition. O'Reilly Media, Inc.

Ruby 'About Ruby', [Online] Ruby. Available at: https://www.ruby-lang.org/en/about/ (Accessed: 10 November 2023).

Jalli, A. (2022) 'What Is the Ruby Programming Language?', [Online] Builtin. Available at: https://builtin.com/software-engineering-perspectives/ruby-programming-language (Accessed: 10 November 2023).

Obregon, A. (2023). 'A Journey Through Time: The History of Java Programming Language', [Online] Available at: https://medium.com/@AlexanderObregon/a-journey-through-time-the-history-of-java-programming-language-9b285d139333 (Accessed: 15 November 2023).

Niemeyer, P. and Knudsen, J. (2002). Learning Java: Volume 2. O'Reilly.

Stroustrup, B. (n.d.) 'Bjarne Stroustrup's Homepage'. [Online] Available at: http://www.stroustrup.com/ (Accessed: 21 November 2023).

Van Rossum, G. (2009) 'The History of Python: A Brief Timeline of Python', The History of Python. [Online] Available at https://python-history.blogspot.com/2009/01/brief-timeline-of-python.html (Accessed: 15 November 2023).

Enki Team. (2022). 'Guido van Rossum: The Father of Python'. [Online] Available at https://enki.tech/guido-van-rossum-the-father-of-python/ (Accessed: 21 November 2023).

Martelli, A. (2003). Python in a Nutshell. O'Reilly.

Lutz, M. (2013). Learning Python: Powerful Object-Oriented Programming. O'Reilly Media.

Vincent, W. S. (2023). Django for Beginners: Build Websites with Python and Django. WelcomeToCode.

Holovaty, A., & Kaplan-Moss, J. (2009). The Definitive Guide to Django: Web Development Done Right. Apress.

Greenfeld, D. R., & Roy Greenfeld, A. (2021). Two Scoops of Django 3.x: Best Practices for the Django Web Framework. Two Scoops Press.

Django. (n/d) The Django template language. [Online] Available at https://docs.djangoproject.com/en/4.2/ref/templates/language/ (Accessed: 10 November 2023).

Ronacher, A. (2010). Flask Design. Flask Documentation. [Online] Available at https://flask.palletsprojects.com/en/2.3.x/design/ (Accessed: 11 November 2023).

Grinberg, M. (2018). Flask Web Development 2e: Developing Web Applications with Python. O'Reilly.

Dwyer, G. (2016). Flask By Example: Unleash the full potential of the Flask web framework by creating simple yet powerful web applications. Packt Publishing.

Yu, A. 2023, '100 Days of Code - The Complete Python Pro Bootcamp for 2021', Udemy, [Online] Available at https://www.udemy.com/course/100-days-of-code/learn/lecture/22558842#overview (Accessed: 27 November 2023).

Pawar, A. 2023, 'Learn Django from scratch, build an E-commerce store, web-based PDF generators, web crawlers, APIs using Python & Django', Udemy, [Online] Available at: https://www.udemy.com/course/django-course/learn/lecture/35014088#overview (Accessed: 27 November 2023).

Appendices

Appendix A: Honours Project Specification Form

COMPUTING HONOURS PROJECT SPECIFICATION FORM
(Electronic copy available on the Aula Computing Hons Project Site)

Project Title: "Exploring Web Development with Python: A Comprehensive Study with Applications (Blog) Built in Flask and Django."

Student: Julia Gongala **Banner ID:** B00402569

Programme of Study: BSc (Hons) in Web and Mobile Development

Supervisor: Tony Gurney

Moderator: Pablo Salva Garcia

Outline of Project: *(a few brief paragraphs)*

Web development is a cornerstone of the digital age, driving the way we interact with information and services on the internet. In this dynamic landscape, Python, renowned for its versatility and user-friendly syntax, has emerged as a powerhouse for web application development. This dissertation embarks on an exciting journey, offering a comprehensive study enriched with hands-on applications. Our focal point is the creation of a fully functional blog website, a journey we undertake using two prominent Python web frameworks: Flask and Django.

Python's selection as the centrepiece of this exploration is not arbitrary but rooted in its growing popularity among developers. Python's elegant and readable syntax, coupled with a vast ecosystem of libraries and frameworks, has propelled it into the forefront of web development. Its seamless integration with web frameworks makes it an ideal choice for building robust and feature-rich web applications. Flask and Django, two noteworthy representatives of Python's web development prowess, each bring their unique strengths to the table, making them ideal subjects for this comprehensive study.

Our voyage commences with a brief introduction to the expansive realm of web development and Python's pivotal role within it. We set the stage for a deeper dive into Flask and Django, shedding light on their significance in the web development landscape.

As we navigate this study, we combine theoretical insights with hands-on practice. Our primary objective is to construct a functional blog website using both Flask and Django, unravelling their capabilities and dissecting their inner workings. This project-based approach ensures that we not only grasp the theoretical concepts but also gain practical experience in creating web applications. We explore the steps required to set up development environments, configure databases, implement user authentication, and craft user-friendly interfaces.

Furthermore, we undertake a comparative analysis of Flask and Django, offering a nuanced understanding of their strengths and limitations. This analysis empowers developers to make informed decisions when selecting the most suitable framework for their specific projects. It serves as a testament to Python's adaptability, showcasing how it caters to diverse web development needs.

Deployment, testing, and debugging, crucial facets of web development, receive their due attention in this exploration. We explore various deployment options for Flask and Django applications, ensuring that our web projects transcend the realm of local development environments and reach a global online audience.

In addition, we address the critical issue of web application security, examining common vulnerabilities and emphasizing best practices for mitigation. This discussion underscores the significance of secure coding practices within the context of web development with Python.

In conclusion, Exploring Web Development with Python: A Comprehensive Study with Applications (Blog) Built in Flask and Django embarks on a captivating journey through the expansive world of web development with Python. It underscores Python's pivotal role in this domain and showcases the capabilities of Flask and Django through the creation of a fully functional blog website. By the culmination of this exploration, readers will possess not only the knowledge but also the practical skills to embark on their web development ventures confidently or make informed choices when selecting the right framework for their projects.

A Passable Project will:

- Introduction to Web Development and Python
- Flask and Django Introduction
- Building basic Blog with Flask and Django where user will be able to add, delete, edit and see posts to compare both frameworks and select better for advanced blog.
- Comparative Analysis for both frameworks showing their features.
- Deployment and Testing of the application

A First Class Project will:

- Framework Comparison and Analysis. A detailed and insightful comparison between Flask and Django, highlighting not only their strengths and weaknesses but also their suitability for specific use cases – Blog using literature review and blog development
- Advanced Blog Development. A feature-rich and highly polished blog application built using selected framework, that has been chosen through comprehensive study to justify all arguments and decision, demonstrating not only the core functionality (CRUD) but also advanced features like user profiles, or uploading images. Advanced blog will incorporate modern practices and techniques.
- Demonstration Knowledge. A clear demonstration of knowledge in modern web development, Python, Flask, Django, and related popular technologies.
- Using References and Citations. Accurate and extensive citation and referencing of all sources, demonstrating a deep understanding of the academic context.

Reading List:

- *"Django for Beginners"* by William S. Vincen
- *"Django for APIs"* by William S. Vincent
- *"Two Scoops of Django 3.x"* by Daniel Roy Greenfeld and Audrey Roy Greenfeld
- *"Flask Web Development"* by Miguel Grinberg
- *"Flask By Example"* by Gareth Dwyer
- *"Building RESTful Python Web Services"* by Gaston C. Hillar
- *"Python Web Development with Django"* by Jeff Forcier, Paul Bissex, and Wesley Chun
- *"Test-Driven Development with Python"* by Harry J.W. Percival
- Django Official Documentation
- Flask Official Documentation

Resources Required: *(hardware/software/other)*

- Personal workstation with Visual Studio Code and Python installed
- Git and GitHub for version control and collaboration.

- Web browser for research and testing.
- Hosting platform (e.g., Heroku, AWS, or a VPS) for deployment.
- Testing tools

Marking Scheme:	**Marks**
Introduction	5%
Literature Review	15%
Development	30%
Comparative Analysis	30%
Testing	10%
Conclusion	10%

AGREED:

Student	**Supervisor**	**Moderator**
Name: Julia Gongala	**Name:** Tony Gurney	**Name:** Pablo Salva Garcia

IMPORTANT:

(i) By agreeing to this form all parties are confirming that the proposed Hons Project will include the student undertaking practical work of some sort using computing technology / IT, most frequently achieved by the creation of an artefact as the focus for covering all or part of an implementation life-cycle.

(ii) By agreeing to this form all parties are confirming that any potential ethical issues have been considered and if human participants are involved in the proposed Hons Project then ethical approval will be sought through approved mechanisms of the School of CEPS Ethics Committee.

Appendix B: Progress and Feedback Meeting Agenda

FORM A
COMPUTING HONOURS PROJECT (COMP10034)

PROGRESS AND FEEDBACK MEETING AGENDA

*(To be completed **before** the scheduled meeting)*

Student: Julia Gongala **Supervisor:** Tony Gurney

Meeting Number: 1 **Date/Time:** 18/09/2023 15:00

PROGRESS

Over the last month, the following tasks have been completed:

- I have worked on project title, thesis and research regarding best application to be created
- I have worked on research relating front end and back end technologies
- Raspberry Pi set up

The following tasks identified last month have not been completed or problems/issues have emerged that require attention:

- None

AGENDA FOR FORMAL MEETING (Example)

1. Discussion of progress so far – presenting idea.
2. Discussion regarding project specification form.
3. Discussion about major milestones – Gantt Chart creation
4. Discussion on tasks to be carried out.
5. Discussion of work to be undertaken towards formal submissions (e.g. Interim Report)

FORM B

COMPUTING HONOURS PROJECT (COMP10034)

PROGRESS AND FEEDBACK MEETING MINUTES AND PLAN

*(To be completed **after** the scheduled meeting)*

Student: Julia Gongala **Supervisor:** Tony Gurney

Meeting Number: 1 **Date/Time:** 18/09/2023 15:00

MINUTES

The following tasks and issues were discussed and specific actions agreed:

1. Discussed current work and tasks
2. Discussion regarding project specification form and final project plan
3. Discussion about creating 2 basic blogs and 1 advanced with better framework
4. Discussion about next milestones and gantt chart.

PLAN

The following tasks and timelines have been agreed both for the next month and beyond:

For the next month:

- Finish and submit project specification form
- Begin research and literature review regarding front end and back end technologies
- Interim Report focus

FORM A

COMPUTING HONOURS PROJECT (COMP10034)

PROGRESS AND FEEDBACK MEETING AGENDA

*(To be completed **before** the scheduled meeting)*

Student: Julia Gongala **Supervisor:** Tony Gurney

Meeting Number: 2 **Date/Time:** 27/11/2023 15:00

PROGRESS

Over the last month, the following tasks have been completed:

- Project specification form has been submitted and confirmed
- I have worked on advanced research relating front end and back end technologies
- Project set up for flask and Django blog
- Template for main page has been created
- Interim report ready for submission? – first feedback

The following tasks identified last month have not been completed or problems/issues have emerged that require attention:

- None

AGENDA FOR FORMAL MEETING (Example)

1. Summary of what I have done so far.
2. Discussion about Raspberry Pi as a host. Will it work?
3. Discussion about Interim Report. Any corrections to do?
4. Discussion about next Development Phase.

FORM B

COMPUTING HONOURS PROJECT (COMP10034)

PROGRESS AND FEEDBACK MEETING MINUTES AND PLAN

*(To be completed **after** the scheduled meeting)*

Student: Julia Gongala **Supervisor:** Tony Gurney

Meeting Number: 2 **Date/Time:** 27/11/2023 15:00

MINUTES

The following tasks and issues were discussed and specific actions agreed:

1. Discussed current work and progress
   Discussion regarding interim report, layout, and specifications, focusing on my narrative style and move towards a more objective style of third person writing.
   Discussion about development phase
2. Discussion about Virtual Machine, might be better than Raspberry Pi

PLAN

The following tasks and timelines have been agreed both for the next month and beyond:

For the next month:

- Finish, correct and submit Interim Report
- Research about Oracle VM VirtualBox
- Developing 2 basics blogs
- Preparations for presentation