Plaskon - A Learning Server for Python Flask

Huibo He

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science

Supervisor: Glenn Strong

April 2022
Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

April 19, 2022
Plaskon - A Learning Server for Python Flask

Huibo He
Master of Science in Computer Science
University of Dublin, Trinity College, 2022

Supervisor: Glenn Strong

The starting age to learn programming decreases year by year. Most students start from block-based programming languages, like Scratch. However, their final goal should be programming in text-based languages, like Python. Due to the syntax complexity, environments that allow them to transit from block-based to text-based gradually are in demand. Aside from the language syntax barrier, the configuration of the local development environment is also complicated for them. Web-IDEs can be one solution. There are existing web-IDEs with various functionalities, including assisting learners learning. Pytch is a web-IDE designed to help students transition from Scratch to Python. It allows the users to develop Scratch-like games or animations using Scratch-like event-driven methods in Python. There is a clear need for a system that can allow learners to move on to more traditional programming tasks. This proposed system, Plaskon, is designed to continue Pytch’s work. Instead of cartoons, it teaches the students to implement web applications using a lightweight web framework Python Flask. With this system, the users can further expand their Python knowledge learned from Pytch. After practising with Plaskon, the users should have the capability to understand Python programs and be ready to learn more complex programming in Python.
Acknowledgments

First of all, I would like to thank my supervisor, Professor Glenn Strong, for all the guidance and help he gave me in each step of my research and work. Second, I am grateful to my family for all their support throughout these five years. With them standing behind me, I do not need to worry about anything, so I can walk forward firmly. I also wish to thank my boyfriend, John Kelly, for his company, patience, and love. Thank Bohyun for the help throughout my last year in college. Thank all the friends and classmates I’ve been working with. And at last, I want to thank myself, for my courage, my persistence, and my confidence.

Huibo He

University of Dublin, Trinity College
April 2022
# Contents

<table>
<thead>
<tr>
<th>Abstract</th>
<th>ii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgments</td>
<td>iii</td>
</tr>
<tr>
<td><strong>Chapter 1  Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Aim and Objectives</td>
<td>1</td>
</tr>
<tr>
<td>1.3 Document Structure</td>
<td>2</td>
</tr>
<tr>
<td><strong>Chapter 2  State of the Art</strong></td>
<td>3</td>
</tr>
<tr>
<td>2.1 Overview</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Block-based vs Text-based Programming</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Pytch</td>
<td>4</td>
</tr>
<tr>
<td>2.4 Web-IDE</td>
<td>5</td>
</tr>
<tr>
<td>2.4.1 Learner-Content Interaction using Web-IDE</td>
<td>6</td>
</tr>
<tr>
<td>2.4.2 Learner-Learner Interaction using Web-IDE</td>
<td>7</td>
</tr>
<tr>
<td>2.4.3 Learner-Instructor Interaction using Web-IDE</td>
<td>7</td>
</tr>
<tr>
<td>2.5 Summary</td>
<td>8</td>
</tr>
<tr>
<td><strong>Chapter 3  Technology</strong></td>
<td>9</td>
</tr>
<tr>
<td>3.1 Overview</td>
<td>9</td>
</tr>
<tr>
<td>3.2 Container Technology</td>
<td>9</td>
</tr>
<tr>
<td>3.3 Container vs Virtual Machine</td>
<td>10</td>
</tr>
<tr>
<td>3.4 Docker</td>
<td>11</td>
</tr>
<tr>
<td>3.4.1 Docker Container Network</td>
<td>12</td>
</tr>
<tr>
<td>3.5 Summary</td>
<td>13</td>
</tr>
<tr>
<td><strong>Chapter 4  Design</strong></td>
<td>14</td>
</tr>
<tr>
<td>4.1 Overview</td>
<td>14</td>
</tr>
<tr>
<td>4.2 Architecture</td>
<td>15</td>
</tr>
</tbody>
</table>
Chapter 5 Implementation

5.1 Overview ............................................. 23
5.2 Docker ................................................. 23
  5.2.1 Dockerfile ........................................ 24
  5.2.2 Docker Images Commands ......................... 25
  5.2.3 Docker Container Commands ....................... 26
5.3 Server .................................................. 27
  5.3.1 Communicate with Docker Containers ................. 27
  5.3.2 Flask Session ..................................... 29
  5.3.3 File Downloading ................................ 31
5.4 Front-end .............................................. 31
  5.4.1 Browser Simulation ................................. 31
  5.4.2 Navigation Bar ................................... 33
  5.4.3 Counting Down Timer .................. 34
5.5 Tutorials .............................................. 35
5.6 The easyHtml API .................................... 36
5.7 Summary ............................................... 37

Chapter 6 Evaluation

6.1 Overview ............................................. 38
6.2 Aim Review ........................................... 38
6.3 Objectives Review .................................... 39
6.4 System Reflective Evaluation ......................... 40
  6.4.1 Strengths ......................................... 40
  6.4.2 Weaknesses ....................................... 40
  6.4.3 Comparison with Other Web-IDEs .......... 41
6.5 Future Work ........................................... 42
6.6 Summary ................................................. 42

Chapter 7 Conclusions ................................. 43

Bibliography ............................................. 44
List of Tables

4.1 Summary of Description and Objectives of the tutorials .............. 19
List of Figures

2.1 An example tutorial screenshot from Pytch ........................................ 5
3.1 Architecture comparison between container and VM .......................... 11
3.2 Docker Architecture ........................................................................ 12
4.1 The road map of transition from Scratch to Python ............................ 14
4.2 An Image of the system architecture design ....................................... 16
4.3 An example tutorial screenshot of Plaskon ........................................ 17
4.4 The final resulting web page for tutorial 9 .......................................... 20
Chapter 1

Introduction

1.1 Motivation

Nowadays, many people begin learning programming at an early age. The vast majority start with a simple block-based programming language such as Scratch. In the actual world, however, text-based programming languages such as Python, Java, and C++ are used to create practically all applications and software. As a result, the students must transition from block-based programming to text-based programming.

This transition is challenging because text-based programming languages have more sophisticated syntax than block-based ones. The English spelling may also be a barrier for specific users at an early age. Students can create simple games and short animation videos in block-based programming environments. They are inspired to study further, practice more, and develop their logical thinking skills due to the graphical outputs. Another benefit of starting with a block-based programming environment is that students can build and run their programs directly in the browser. Local configurations are not required, so they can concentrate on learning to program.

Those who are in the process of transitioning from block-based to text-based programming may find applications with similar features useful.

1.2 Aim and Objectives

In the transition process, there can be multiple steps. Scratch and Python are chosen to be the start and destination points of the transition. Pytch is the first step in the transition, a web integrated development environment (web-IDE) that allows the users to write Scratch-like code and event-driven methods using Python. The users can also develop the same simple games and short videos as in Scratch.
This proposed system is the next step. It aims at the learners who have finished practising with Pytch, already have the basic knowledge and experience of Python language and are willing to move forward to develop a different type of visualised outcome, like web applications. After using this system, the users are expected to be ready for working with more complex Python language eventually. The system teaches the users about web development using Python Flask. The users can strengthen their Python knowledge learned from Pytch when using this system, and they can also learn about web development and different Python data structures. The resulting web pages are also a visualised output that can continue increasing the user’s interest in learning programming.

The objectives of this project are listed below. They are divided into two stages: the research and technical development stages.

Research stage:
- Identify a suitable application domain for web-IDE for a sensible next step for Pytch users
- Research about existing web-IDEs and how they operate

Having completed the first two objectives, it is possible to identify a third objective, forming the technical development portion of the project, which is the development stage. It also contains a technology-related research objective.

- Develop a web-IDE that allows users to write, compile and run Python Flask in the browser
  - Research about technologies that provide servers that the users can use to run their Flask applications
  - Implement tutorials that can teach the users about Python Flask
  - Implement an API that users can use to write HTML script in Python

1.3 Document Structure

The rest of this dissertation is structured as follows: Chapter 2 discusses the state of art research relevant to this proposed system. Chapter 3 presents the technology selected to use and how it can support the desired features of the proposed system. Chapter 4 then states the architecture designed and the decisions made for this work based on the research results. The implementation process is described detailedly in Chapter 5, including the challenges encountered and corresponding solutions applied. Chapter 6 evaluates this project, reviews the work based on the goal and objectives set, and suggests future work that can be done with more time. In the end, Chapter 7 concludes this dissertation.
Chapter 2
State of the Art

2.1 Overview

Python is the most popular programming language globally. It is a high-level language focusing on the readability of the code. Python has become the best choice to introduce programming to advanced learners. Its online community is supportive. Learning programming does not limit to computer science students. Python can be learned and used for any programmable problem. In secondary schools, the interest in learning and using Python has also increased in recent years. However, there are challenges for the students who first begin to learn to program. According to Rodrigo et al., motivation, problem-solving skills, engagement and language syntax are the main challenges for the students, especially when transiting from Scratch-like block-based programming to Python like text-based programming. Section 2.2 will discuss the two types of programming. Pytch is one of the environments that can help the learners in this transition process. And in Section 2.3, Pytch is introduced in more detail.

Web development can be a practical way to help the students overcome some of the challenges caused by the transition. In addition, the visualisation of the implemented websites can help build up students’ motivation and sense of achievement. Zhou indicates that it is beneficial for students to learn web programming early when they freshly gain programming knowledge and logical thinking. This can also increase the learner’s confidence and interest in learning programming.

Flask is a microframework of Python which uses Jinja2 Template Engine and Werkzeug WSGI toolkit for HTTP requests and responses. It is popular because of its simplicity and flexibility. It is lightweight but extensible, which contains a collection of code, providing methods to build a web framework with basic functionalities. With Python Flask, the students can simultaneously practice Python language and web development.
skills. A Web-IDE can provide a way for the students to write their code and see the resulting websites on the same browser page. Section 2.4 will talk about existing web-IDEs.

2.2 Block-based vs Text-based Programming

Block-based programming environments, like Scratch and Alice, allow users to interact with mouse control to build executable blocks of codes. It is friendly to learners who cannot understand programming languages or are unfamiliar with English words. Moreover, it is straightforward to use. However, the computational algorithms are not simple to master when developing complex programs. [42] Combining logic teaching and hands-on programming could be better.

On the other hand, text-based programming is more general in developing real applications and software. Python, Java, and C/C++ are all text-based programming languages. Text-based programming has such a dominant position because all kinds of editors can edit texts, and it is also easy to work with external tools, like source controls. Although text-based programming can cause compile errors, it is more flexible to manipulate, unlike block-based programming. [11]

Hidekuni et al. ‘s research [37] shows that teaching young students directly in a textual programming language using Processing is possible. However, the student’s motivation to learn depends on the teaching methods. On the contrary, by teaching students using visual programming language, which is also known as a block-based programming language, the motivation of students increases. [38]

Block-based programming is simple to learn and motivates learners better, whereas text-based programming is more flexible and reliable. The transition from block-based to text-based programming is essential but challenging because it takes time to learn the syntax and memorise the concepts and usages of the language.

2.3 Pytch

Pytch [23] is an online development environment that supports the students in transit from Scratch to Python. Rather than focusing on the usage of a text editor, it focuses on the programming paradigm. It provides a package with which the students can write Python code in their familiar Scratch style to develop the same games and short cartoon videos as in Scratch. The picture below is a screenshot of one of the Pytch tutorials. [21]
Pytch uses the simple fundamental Python syntax and easy-to-understand event-driven method names to present what a text-based programming language like Python should look like. The environment is divided into three sections: the in-browser editor, the interactive output window, the used files, text outputs, and the error window. After practising with Pytch, the users should be able to understand and utilise the basic knowledge of Python, including the usage of decorators, variables, methods and other simple syntaxes.

Pytch is the first step of the gradual transition from Scratch to Python. The users will be ready to move further after mastering programming in Pytch.

### 2.4 Web-IDE

Pytch is a web-IDE. IDE is short for Integrated Development Environment, which developers use to implement, compile, debug and execute code. A web-IDE is a type of IDE that is browser-based, with which the developers do not require to download, install or configure dependencies. A web-IDE can be accessed from any device anywhere with an internet connection. Though this could be a disadvantage of web-IDEs for security concerns, they have become widely used, especially for distance learning. Non-computer
science students can also benefit from web-IDE because they only need to focus on studying and understanding the programming itself rather than complicated configurations.  

Various languages can be implemented using web-IDEs. Replit is one of the most popular web-IDEs, which can support multiple languages and packages. Users do not require to download or configure anything while using it. Compared with other powerful web-IDEs, like CodeChef and Ideone, replit requires the least transferred data, and it is also available for interaction and real-time collaboration.

Web-IDE is not limited to popular programming languages. For example, RESOLVE is a language used to facilitate mathematical reasoning, and there is also a web-IDE for RESOLVE. It is used in [3], which gives positive results from both students and instructors.

There are three types of interaction in learning: learner-content, learner-learner and learner-instructor. This section will introduce web-IDEs from these three perspectives of functions.

### 2.4.1 Learner-Content Interaction using Web-IDE

Learner-content interaction is the way of individual learning that the learners use available content to study by themselves. The existing web-IDEs for assisting individual learning mainly have two types. One is to help the students transfer from block-based programming to text-based programming. The other one is to increase the interactivity between the learners and the web-IDEs so that the web-IDE can be used more widely.

Since block-based programming environments are mostly running in browsers, it is easier for the learners to get familiar with textual programming languages in the same way without worrying about local IDE configurations. Harmonik is a block-code tool designed to help novice programmers learn textual programming languages, including C/C++, C#, Java and Python. And PyBlockly, a similar tool, focuses more on the Python language. Users can assemble different blocks of Python code to form logic. The code blocks can be translated into executable Python code 1:1 correspondingly so that the students can get used to Python syntax. Blockly is used in Gao et al.’s new pedagogical method for introductory programming courses, PBL-VP. It applies the Problem-Based Learning method and utilises Visual Programming with Blockly, which helps the students engage in programming, understand the implementation and develop computational thinking skills.

Apart from the transition process, web-IDE’s basic interactivity features are also essential for learning programming. Hegde et al. proposed a web-IDE for C language
specifically, which provides a terminal where the users can interact with the program in real execution time. Similarly, the CS50 sandbox, developed by Malan [16], is an environment for executing untrusted code securely. It provides an HTTP-based API that can also support command lines and other standard inputs, including files. Its editor, CS50 Run, allows the users to write their code in any language. CodeSkulptor [33] is a Python focused web-IDE. It uses an interactive GUI library that can help trace the frame/objects diagram and then display them in the console, which is helpful for users to debug their program.

Frame-based editing is a concept proposed by Kolling et al. [11]. It aims to combine the beneficial features of both block-based and text-based systems. A Java-like language is designed called Stride. All the statements are provided by frames, and the users can write in the slots to complete the program. With the frame-based editing idea and on top of Stride, Strype [13] is designed for Python language. Similarly, the frames present the program’s structure, and the slots are used for user input. Simple Python code can be edited and run in the browser, and the outputs are shown on the same page. Besides using frame-based editing, Strype also provides a “convert to Python file” option to allow the users to download the Python script. The main difference between PyBlockly [32] and Strype is that in Strype, the code structure is in frames, not blocks.

2.4.2 Learner-Learner Interaction using Web-IDE

In the process of programming education, teamwork is an unavoidable lesson each student needs to learn. However, students programming on their local machines may not be efficient for the learner-learner interaction, in other words, collaboration. Web 2.0 describes a more social and user-centric modern web. When students learn programming, teamwork skills, communication skills, and experience of collaborating with different people are also essential for them. IDE 2.0 [8] and IDEOL [36] are two example web-IDEs that apply web 2.0 techniques to provide real-time collaborations so that students can work concurrently in-browser. IDEOL also has a real-time discussion board with the tagging mechanism. The students show high satisfaction with the experience of collaboration using the web-IDE. [20]

2.4.3 Learner-Instructor Interaction using Web-IDE

Web-IDE can also assist with learner-instructor interaction. The CS50 sandbox has a CS50 Check component [16], which provides an auto-grading framework for the students to assess their assignments. The instructors also have access to the results and give feedback to help the students with their learning based on the results. The discussion
board provided by IDEOL [36] also allows the instructors to interact with the students. The instructors can follow the student’s work by tracing their work logs maintained by IDEOL.

CodeHelper [15] presents another solution to perform learner-instructor interaction for online learning. Not all the students feel comfortable discussing their questions publicly. Some would prefer to have 1 to 1 sessions with the instructors. CodeHelper allows the instructors to open real-time sessions so that students can paste their code to the shared editor and ask for suggestions.

2.5 Summary

This chapter firstly compares the differences between block-based and text-based programming and indicates the importance of the transition. Then highlighted an online environment Pytch, which is the fundamental step for this proposed system. Finally, it describes the existing web-IDEs and their various functionalities. The majority of them focus on providing environments where students can learn and practice programming in the browser. However, only a few tend to develop a solution to help students transition from block-based programming to text-based. Apart from that, only replit’s community may contain tutorials designed by some users. As a result, the existing web-IDEs do not demonstrate much of their educational function. Therefore, it shows that it is reasonable and valuable to design and implement a web-IDE, which provides tutorials that can help the students further learn programming after practising with Pytch. The next chapter will present the research result of the technology that can support this work.
Chapter 3

Technology

3.1 Overview

The previous Chapter describes and analysis multiple existing web-IDEs and their various functionalities, which proves that the goal of this project to develop a web-IDE to help the students transit from Pytch to Python is practical. Since Pytch uses event-driven concurrent programming, a hybrid of Python and Scratch style program design, the proposed system does away with the concurrency and provides a mainstream Python model by teaching the users to use Python Flask to develop web applications. Furthermore, web applications require servers to handle HTTP requests and responses, which is also event-driven programming. Therefore, the next question becomes how this system can support multiple servers running for different users to host their in-browser running applications.

Container technology that claims to offer a way to handle this problem is taken into consideration. First, this Chapter discusses container technology and compares it with the virtual machines in sections 3.2 and 3.3. Then docker, a platform that applies container technology, is introduced in section 3.4.

3.2 Container Technology

A container, in English terms, is a large object that holds something and transports it. And in technical terms, a container provides a virtualised environment for one or more pieces of software to run in isolated user spaces. When developers finish implementing an application in the development environment, they need to test it in a testing environment and then deploy it to a production environment. This process requires three environments and one application, and the three environments may not be exactly the same. A different environment in either hardware or software may cause issues to the application. 

[28]
The container technology provides a faster way to deploy the applications in the same environments by applying the same operating system and dependencies to eliminate the effect caused by the differences.

Containers also have a positive impact on the development process. The review of Pahl et al. [21] shows that containers can support continuous development in the cloud. Many large corporations are already using container technology to host their applications. It makes the application life cycle management agile and accessible. [28] As for security concerns, using containers can avoid unwanted interactions among programs. Each program is isolated in the container. However, there are multiple types of networking in container technology that support different communications types among them.

3.3 Container vs Virtual Machine

Virtualisation is to create a virtual version of something. In computer science, virtualisation is more technical. It is to create a virtual version of an operating system or hardware. Container-based virtualisation virtualises operating systems and relevant dependencies on a single Linux kernel. It is also called operating system virtualisation. With containers, the applications share one operating system, resulting in a small size of deployments, which means that one host machine can store a large number of containers. [2] But the applications are not absolutely isolated because of the sharing host machine, which may cause security issues. If the operating system gets attacked, it will influence all the applications using the container on the system, resulting in a single point of failure. [29]

Virtual machine (VM) uses hypervisor-based virtualisation. The hypervisor is the software that can help virtualise the hardware shared by the applications deployed on the virtual machine. [28] The applications are completely isolated from each other. They can install any operating system that they require, which is flexible. But this also results in its large size. It takes a longer time for the applications to start compared with the ones deployed on containers where the operating system of the containers is already running. The figure below 3.1 [41] compares the application deployment using a virtual machine and a container.

Though the container-based services provide a smaller-sized, faster and easier to manage application deployment, they may not always overperform VM-based services. All three scenarios of Salah et al.’s experiment [27] show that the VM-based services perform better than container-based. They also point out that the experiments use Amazon EC2, where the containers are running on top of EC2 VMs rather than bare metal, which can be why VM-based services’ are overperforming. Containers are popular because of their
lightweight, but their security issue is concerning. Manco et al. find that VM can also be lightweight as long as they are small with a fast enough toolstack. The LightVM they designed proves that achieving isolation and good performance simultaneously is possible.

The container-based virtualisation is a better choice when efficiency is the priority, and hypervisor-based virtualisation, the virtual machine, is better when isolation and security are the priority.

### 3.4 Docker

Docker is an open platform using container-based virtualisation to provide services that allow software deployment in containers. The figure below shows the details of docker architecture.

Docker uses a client-server architecture. Docker daemon is located in the Docker host, responsible for building, running, and delivering the containers. The client communicates with the docker daemon using REST API. Users send Docker commands to the Docker client to interact with the Docker host. Docker images are files that contain containers configuration. They are stored in repositories called Docker Registry. Docker Hub is a public Docker Registry. There are various base Docker images stored on Docker Hub.

When a user needs to create a container, the Docker daemon will pull the required Docker image and download it to the client machine. After the downloading finishes, the container will be running based on the Image. The Docker images are read-only templates. Every Docker image starts with a base image which can generally be found on Docker Registry. Depending on the requirements, they can also be customised from scratch using dockerfile.
Dockerfile is a file containing the instructions of image specification. When the user requests specification to build the Image, each command in the dockerfile will create a new layer on top of the current Image. The resulting image has a stack of layers. The containers are created based on the Docker image, so it is essential to keep them lightweight. Writing similar commands in one instruction can help reduce the layers of the image stack and ensure a small size. An interesting observation from a survey shows that most of the activities of developing a dockerfile are time-consuming, especially for novice developers. Most of them use the trial and error method to solve the issues they met during the dockerfile development.

3.4.1 Docker Container Network

Containers are used to host applications, but sometimes, these applications require to communicate with each other. There are four networking modes in docker for the containers on the same host to communicate. They are None mode, Bridge mode, Container mode and Host mode.

The None network provides the highest security and isolation. The containers with None mode cannot connect with other containers on the same host or the external network. The Bridge networking allows the containers to have their own network namespace and IP address. Though it is not as secure as None mode, it is the optimal choice if a user needs to balance container performance and isolation. The Container mode networking allows a group of containers to share the same namespace and IP address, distinguished
by port numbers. This mode’s security level is medium, and it is helpful if a user owns multiple containers. And at last, in the Host mode, all the containers on the same host share the host machines’ network namespace and IP address. It has the lowest security of the four modes.

3.5 Summary

This chapter lays the foundation for selecting the technology to be used in this proposed system. The difference between container technology and the virtual machine is the object they visualised. Container technology virtualises the operating system, and the applications using the same operating system in containers can share one server machine. On the other hand, virtual machines virtualise the hardware so that deployed applications can install any required operating system. Overall, container technology is more suitable for this proposed system because all the users are expected to use the same system to run their designed web server. Therefore, Docker is selected to be used in this work. The next chapter will dig into more details about the design of this project.
Chapter 4

Design

4.1 Overview

The previous chapters present the results of the research objectives listed in section 1.2. They also establish the direction of this project which is to design and implement a web-IDE that can support the user at the second step of the transition process from block-based programming language Scratch to text-based programming language Python. The figure 4.1 presents the planned transition process. As described in section 2.3, Pytch is the first step that allows the users to write Scratch-like code and implement event-driven methods in Python to develop the same simple games and animations as in Scratch. Therefore, the learners who finish practising with Pytch should be familiar with basic Python syntax.

This proposed system, Plaskon, is the second step (the third box in the figure) of the transition. It aims to lead the learners to learn more about the Python language. The system is also designed as a web-IDE, and it is used to teach the students to develop a
website because visualised outputs can motivate the learners better. Scratch and Pytch both use visualised outcomes to keep the students' interests. Another reason web development is selected at this step is that it also uses event-driven methods that the users already understand. The users for Plaskon are expected to be already familiar with the fundamental knowledge of Python taught in Pytch.

Replit [26] is one of the most popular web IDEs. Users can write, compile and execute their code in replit without spending time installing packages. When users start to run their code on replit, the server will assign them a container with all necessary packages installed. The container will then execute the user code and return the result to the user’s browser. Learnt from replit, this work also chooses to use containers.

Docker is selected to be the one that supports the container component of Plaskon. Like replit, this system also provides each user with a docker container to host their implemented web server. Each user uses the same operating system and dependencies. Therefore, only the system needs to be virtualised, and container technology is chosen over virtual machines. In addition, using containers can also ensure the system’s lightweight.

This chapter presents the architecture designed for the system in section 4.2. The rest sections discuss the detailed design decisions for the server, docker and the API provided by the system.

4.2 Architecture

The system’s architecture is illustrated in the figure 4.2. The Flask learning server and the docker container are the system’s two components. This system is designed to help students learn the Python programming language by utilising the Python Flask web framework. As a result, the Python Flask is decided to be used to implement the system’s primary structure.

The main server’s structure is shown on the left side of the architecture diagram. The server, like other applications, has a back-end, a front-end, and storage. As previously stated, the back-end is written in Python Flask. The usage of Python Flask as a framework makes communication between the server and client sides more efficient. Because this project is currently a prototype and the server is built to run locally, the data for this system is stored locally. The storage will be moved to the cloud databases if deployed to the cloud. At present, using local storage is sufficient for the proposed web-IDE.

The other vital component, the docker container, is presented on the right-hand side of the diagram. It contains an Alpine Linux system as the operating system with the necessary packages installed. In addition, the easyHtml API, which allows the user to write HTML scripts in Python, is also provided in the container component. When the
users submit their code, their code will be executed in the containers. The containers are then hosting the users’ web servers. The server uses the IP addresses received from the containers to form URLs and display the users resulting web pages.

The following sections will discuss each component of the system in more detail.

### 4.3 Server

The server is used to interact with both the users and the containers. Since the users cannot directly communicate with the assigned containers, the server also plays a role as a bridge between them. It provides various functions to support the users to learn about web development using Python Flask.

The features of this server are listed below:

- In-browser editor
- Browser simulation
- Assign containers to users within time-limited sessions

Apart from the main features to form the web-IDE, there are other functionalities to support the users’ learning process. Inherited from Pytch, this system also provides tutorials to show the users how to use this system and how to program in Python Flask.
Most of the existing web-IDEs do not have these features, and they grant the users flexibility to implement. However, the tutorials guide them to get familiar with the system and the programming language. Their programming outcomes following the tutorial will give them a sense of achievement and motivate them. Furthermore, the tutorials can be downloaded to the users’ local machines to practice locally. The tutorials and the downloading function are discussed in sections 4.3.2 and 4.3.3.

### 4.3.1 Overview

The home page, tutorials, user’s project, and user login are four parts of this server. Users can switch between each page using the navigation bar.

The home page contains an introduction to this project, and the get-started tutorial page provides a guideline to explain how to use this system. This project aims to teach the learners about developing web applications using Python Flask. Containers are used as the server for the in-browser programming. Users need to log in to be assigned a container and use it as their temporary server. For avoiding unnecessary latency caused by repeatedly starting a new container, each user is assigned a container for a limited time. The container will keep running and executing the user’s code until the user chooses to log out or reach the time limit. Local storage is used instead of databases. Thus, users’ activities and programs will not be permanently stored associated with the user. The system will eliminate the information after the sessions finish.

![Figure 4.3: An example tutorial screenshot of Plaskon](image)
The tutorial and user’s project pages have the same layout related to the in-browser editor and visualisation. The figure 4.3 displays one of the tutorial pages. The box on the left is the editor. Users can learn from the provided tutorial code and make changes. When the submit button is clicked, the browser will send the code back to the server. Then the server will check if the user has already been assigned a container. If not, a new container will be created for the user. Then the server copies the code file to the container to be executed. The box on the right is where the resulting web page is visualised. It is retrieved from the container server’s host URL, differentiated by the container IP address. This ensures that the user’s programs will not influence each other. The input box above the web visualisation displays the current URL. Users can edit the URL to manually switch between the routes they implement in the script. They can also copy the URL and paste it into another browser tab to display.

Besides the web-IDE essential components, a download button is above the editor. All the tutorial-related files will be zipped and downloaded to the user’s local machine, including the API and CSS files provided when it is clicked.

There are no significant differences in layout on the user’s project page. The editor on this page only contains the default essential Python Flask server structure. Users can develop their simple web applications in the browser.

### 4.3.2 Tutorials

The users are assumed to have basic knowledge of the Python syntax learned from Pytch. The tutorials are designed to demonstrate a more complex usage of Python programming. Nowadays, social media are popular among people. Therefore, to motivate the learners more, the series of tutorials is designed as a step-by-step guide to developing a blogging website, which has the typical social media features, including login, posting different types of blogs, liking, commenting and deleting a blog. There are nine tutorials in total with ready-to-execute code provided. When the users click the submit button, the code in the editor will be sent back to the server, and the resulting web page will be displayed on the right-hand side of the screen. The descriptions and objectives of the nine tutorials are listed in the table 4.1.

With the tutorials, users can review and strengthen the knowledge of basic computational logic that they gained from Pytch. They can also learn about the requests and responses, which are essential for web applications. Since web design also needs knowledge of HTML and CSS, which can help make the pages attractive. This project also implements an easyHtml API to help with front end development, which will be discussed in section 4.5. A static default CSS file is also provided. Because this system focuses on
<table>
<thead>
<tr>
<th>Tutorial</th>
<th>Description</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Print out “Hello World”</td>
<td>Learn the primary usage of flask</td>
</tr>
<tr>
<td>2</td>
<td>Print out “Hello World” using HTML</td>
<td>Learn the relationship between flask server and templates</td>
</tr>
<tr>
<td>3</td>
<td>User login/out by manipulating the URLs</td>
<td>Introduce flask session and redirect URL</td>
</tr>
<tr>
<td>4</td>
<td>User login/out using a navbar</td>
<td>Introduce flask-nav</td>
</tr>
<tr>
<td>5</td>
<td>User input - text blog</td>
<td>Learn about “POST” requests and responses</td>
</tr>
<tr>
<td>6</td>
<td>User input - picture blog</td>
<td>Learn about uploading and storing files</td>
</tr>
<tr>
<td>7</td>
<td>Blog interaction - Like and Delete</td>
<td>Learn about “GET” requests</td>
</tr>
<tr>
<td>8</td>
<td>Blog interaction - Comment</td>
<td>Strengthen the knowledge of python dictionary and list objects</td>
</tr>
<tr>
<td>9</td>
<td>Combine all features</td>
<td>Complete the simple blog website</td>
</tr>
</tbody>
</table>

Table 4.1: Summary of Description and Objectives of the tutorials

Python language, the users are not yet allowed to modify the design of their resulting website in the web-IDE. However, the files can be downloaded to users' local machines to set up their own Flask instances to publish their Flask server with any design they prefer.

The final complete tutorial blog web page is shown in the picture 4.4.

4.3.3 Session and Download Option

As mentioned in the previous section, sessions are used for the users to log in so that they can be assigned a container. Python Flask has a session package that can store users' information. For this work, only usernames are needed. More information could be collected if a database were used for a productive project. Each session has a 3-hour time limit, and a counting down clock is presented on the page after the user logs in. Within a user’s session, the container is entirely operated by the user. If the user accidentally closes the browser, they should be still in the session when they re-access the system. The Flask session will pop the username and stop the relative container when the time-limited is reached. This can help decrease the number of running containers that are not in use.

The project aims to lead the users who are willing to learn and master Python programming language one step forward towards it. The destination of the learning process is that the learners can develop applications individually on their local IDE. The web-IDE is configured and convenient, but it is not sufficient for real-world programming. Therefore, this system provides users with a download option to run the program locally. The downloaded file includes the Flask application structure, the API provided to create
HTML scripts, the tutorial’s CSS script, and their server code. There is no requirement to modify the code to run locally if the correct packages are installed. The download function allows the users to test the learning outcomes in their selected IDE, giving them a sense of achievement.

### 4.4 Docker

Docker containers play an essential role in this system. Since this work aims at teaching the learners more about Python by leading them to develop simple web applications, each user needs to have a server to host their websites. The containers are used to support
this purpose. As discussed in section 3.4, Docker is selected to provide basic, reliable functionality for this proposed system.

### 4.4.1 Docker Image

A docker image contains container configurations, including an operating system and required dependencies. Multiple Docker containers can be built based on a specific docker image. The docker image designed for this proposed system uses the alpine Linux system as the base image and applies default bridge networking. Alpine has the smallest size among all the Linux systems available on docker, which can decrease the waiting time when users initially start their servers. As discussed in section 3.4.1, bridge networking in docker ensures that the containers built from the image can not influence each other. It allows each container to connect to the external network with its IP address. Python and pip are manually installed on the Docker Image through the dockerfile. With pip installed, the Flask related packages are also installed.

The files structure that Python Flask used is critical to the implementation. Therefore, the file hierarchy is also built in the designed docker image. Flask requires two directories: templates and static. The “templates” folder stores all the HTML, and the “static” folder contains CSS scripts and other static files. Both directories are under the “usr” directory in the docker image. An easyHtml API is implemented and provided to the users to edit the HTML files in the server Python script. The API is also kept in the “usr” directory of the docker image.

### 4.4.2 Docker Container

The docker containers that work as users’ servers are created based on the docker image described above. When a user submits their code, the script will be copied to the user’s container by the server and executed. The server can retrieve the container IDs and IP addresses through the terminal commands and preserve them in the Flask session. With the IP addresses, the web page of the customised running server on the container will be displayed. Users can interact with their designed web applications. When the user finishes with the container and logs out or reaches the time limit of their session, the container will be stopped.

### 4.5 The easyHtml API

Jinja2 template is used in Python Flask HTML files. Simple Python syntax is allowed in the template with a particular format. The template enables the users to create a base file
with the standard structure to eliminate duplicate code among multiple HTML scripts. In the base HTML, a block-based type of implementation is used so that in an individual HTML script, the user only needs to edit a specific block.

This system and its tutorials all use Python Flask and the Jinja template. An easy-Html API is implemented with a base.html file provided by Flask bootstrap, containing all necessary blocks, including title, style, and content. When the user initialises an object with the API, an HTML file extended from the bootstrap base file is created.

At the beginning of the design, each API method opens the newly created HTML file and writes to it. Through the implementation process, a string property of the object is considered to use instead. When a new component of the HTML is added, the string is updated. The string will be written to the file when the user calls the write() function at the end of editing. Modifying the string property is more efficient than writing to the file directly in every statement. The users can only edit their HTML files in the content block. To help the users focus on the main program, the style block and navbar block are available as a whole, which is unchangeable. The style block contains a simple CSS script to make the resulting web page look attractive, and the navbar block is connected to the navigation bar design in the server code. If the user is willing to download the files, the API and CSS scripts will also be in the downloaded file.

This API supports most of the general tags in HTML, including but not limited to \texttt{\textless p\textgreater}, \texttt{\textless div\textgreater}, \texttt{\textless h1\textgreater}, \texttt{\textless form\textgreater}, \texttt{\textless input\textgreater}, \texttt{\textless a\textgreater}, \texttt{\textless span\textgreater}, \texttt{\textless img\textgreater}. Because Jinja2 allows the users to write Python syntax in HTML, the API also provides methods that allow the users to implement the conditional and loop statements in the HTML scripts.

### 4.6 Summary

This system is designed as a web-IDE that allows users to learn and practice Python programming by developing a simple web application to move forward further from Pytch to Python. Docker containers built upon the Alpine Linux system based docker image with necessary dependencies, directories hierarchy and the API provided are used as the users’ servers. The web applications developed by the users can be executed in their assigned docker container, and the resulting web pages are displayed on the same page in the browser. In the next chapter, the implementation of these designs will be discussed.
Chapter 5

Implementation

5.1 Overview

The design chapter describes the high-level architecture and decisions made regarding the proposed system. This chapter will explain the detailed implementation of the development, including the challenges met and solutions applied during the process.

When a person starts to learn a new programming language, the first program they write will generally be printing out a ”Hello World” String. Since this project is to design a web-IDE, the implementation begins with allowing users to write a simple ”Hello World” program in the browser editor and display the result on the same page. Docker is the system’s vital component that ensures this web-IDE works as expected. Therefore, the following section presents the implementation of docker. Then, the development of the server will be explained, including how the server communicates with the docker containers. After that, the front-end implementation will be discussed, followed by sections describing the tutorials and the easyHtml API provided for the users.

5.2 Docker

As discussed in section 3.4, docker containers can provide a virtual environment containing all the dependencies an application requires and support the application running on various operating systems without extra configurations. This system decides to assign the users docker containers so that they can host their web servers. The docker containers are created upon the read-only docker images. In this proposed system, the users will program in the browser, and the server will send the code to their container. The docker image contains the essential packages, Flask file structure, and required API and ships them to each container.
Official Docker images are provided in the docker hub, including ubuntu, alpine, Python, etc. Because this project aims to teach the users Python Flask, a Python image is selected to use at the beginning. Then, it is found that when the container is running a Python image, the command line only accepts Python code. The container provides a Python environment for the users to code, which does not meet the requirements. The desired image should be based on an operating system that contains a file hierarchy and can accept terminal commands. As a result, Ubuntu and Alpine are considered. Both of them are Linux systems. The container used in this project only needs to be able to host the Flask application that the users implement. It is designed to be lightweight so a smaller image size would be preferred. According to docker hub, the latest version of a ubuntu image is 27.24MB, and an alpine image is 2.68MB. Therefore, an alpine image is selected to be used.

The alpine image is not the final image used for this system because it only contains the base Linux system. In addition, the file hierarchy of Flask and packages are still needed to be installed. Thus, a dockerfile is required to assemble the desired image.

5.2.1 Dockerfile

A dockerfile is a text-based document that contains all the instructions to build a customised docker image. Each command constructs a layer, and the layers stack together to form the final docker image. The dockerfile usually starts with a command that pulls an existing image from the docker hub as the base system. Alpine is the base Linux system that is used for this project. It is known that applications have different versions, whereas images have tags to represent their versions. The format defines a specific version of an image should be `image_name : version_tag`. The tag `latest` is used to mark the newest version of an image. Therefore, the first line in the dockerfile for this project is like this.

```bash
FROM alpine:latest
```

After the base operating system layer is established, packages are needed to be installed. Commands that can run on the alpine system are required in the dockerfile. The instruction `RUN` in the dockerfile can create a new layer on the top of the current image, and the corresponding command will be executed in the system when building the image. It can run all the instructions that can carry out on the base system.

Python3 is one of the packages that need to be installed on alpine. On Ubuntu, one of the most popular Linux distributions, `apt - get update` is used to retrieve the latest list of available packages, whereas, on alpine, it is `apk update`. And instead of `install, add` is applied to install the packages. Apart from Python3, pip3 is the package manager for
Python, which is also essential for this image. With pip, Flask and related dependencies can be installed. The instructions in dockerfile are:

```
RUN apk --update add python3 py3-pip
RUN pip3 install flask flask-bootstrap flask-nav
```

Besides the packages installed, since this web-IDE is used for Python Flask programming learning and practising, the correct file hierarchy in the server is also essential. The user code is intended to be stored in the ‘usr’ directory of the alpine system. In addition, the ‘templates’ and ‘static’ folders should be present within the directory. The ‘templates’ folder stores all the HTML files created by the user, and the static files, including CSS scripts and pictures, will be saved in ‘static’. “mkdir” is the command to create a directory in Linux.

```
RUN mkdir -p /usr/templates /usr/static/css
```

The code snippet contains two directories separated by a space, which is because each RUN command will create a layer on the current image, and the size of the image is expected to be as small as possible. Therefore, merging multiple instructions in one command can minimise the number of layers on top of the base system layer. In addition, the flag -p in the code allows the system to create parent directories if it does not exist because alpine does not have a ‘static’ folder in ‘usr’.

After the file hierarchy has been established in the image, specific files required to execute user code must be copied into the image. The command COPY in the dockerfile can duplicate the local files to the docker image. There are three scripts that all users need to use. The API for the users to program HTML in the Python file, the simple CSS script and a Python file that contains all the necessary libraries to import to the user’s code. Their Python scripts are copied into the ‘usr’ directory of the image, and the CSS is copied to the ‘static/css’ folder. The syntax for the COPY command in the dockerfile is:

```
COPY local/directory usr/image/directory
```

The dockerfile consists of the structure of the desired docker image, but it needs to be built and pushed to the Docker repository to be used. Docker commands are used for this.

### 5.2.2 Docker Images Commands

The docker commands used in this project can be divided into two purposes, one is for building the docker image, and the other is for running the docker container. The
dockerfile is saved in the root directory of the project. A simple command like the below
can build the desired docker image based on the stacked layer in the dockerfile.

```
docker build .
```

The ‘.’ represents the current directory so that the dockerfile will be automatically
located. Then the base alpine image will be pulled from the docker hub. Then the
commands will be executed one by one to add to the base layer and form the final image.
But this only means that the docker image is built. It still needs to be pushed to a docker
repository to be able to be accessed. The docker repository is one of the docker objects.
It is similar to the Github repository, but it stores docker images. Developers can register
accounts on Docker Hub and then push the built docker images. The tag of the image
can specify its version. If a tag is not presented, it will be marked as “latest”. The syntax
is shown below:

```
docker push user_name/docker_repo_name:tag
```

After being pushed to the repository, the docker image is ready to be used. The
command below can list all the top-level images, their repository, tags and sizes.

```
docker images
```

### 5.2.3 Docker Container Commands

In this project, the docker container commands help test if the containers work as ex-
pected. A new container needs to be created based on the built docker image using the
code below. Three flags \(-itd\) are applied. \(-i\) represents interactive, with which the
container will keep the standard input open no matter if it is attached. The flag \(-d\) is
responsible for detaching the container to let it run in the background. The user code
must be copied to the running container, and the copy command must be executed in
the local terminal. Keeping the container running in the background makes this process
easier. The \(-t\) flag tells the docker to assign a virtual terminal session for the container
so that it is possible to interact with it.

```
docker run -itd user_name/docker_repo_name:tag
```

When the container is running in the background, using docker ps can help check
the information of the currently running containers, which contains the container ID, the
image used, current command, created time, current status, the port used and the name.
The container’s name is randomly assigned by docker when starting it. It is required when
copying the user code to the container. The snippet below is used to copy and transfer files from the local machine to the 'usr' directory in a particular running container.

```bash
docker cp user_code/directory container_name:usr
```

After the testing code is received, the container needs to be brought to the foreground to be interacted with. The container ID is required for the `docker attach` command to achieve this. When the container is attached, it can be used as a virtual system and controlled by using commands in its terminal.

```bash
docker attach container_ID
```

There are two ways to exit the running container. One is to type the command `exit` in the terminal, which will exit and also stop the running container. The other is to use the keyboard, press `Ctrl + P` then `Ctrl + Q` to detach the container instead of stopping it. The second method is helpful for the testing process. It is good to stop them from running by using the command `docker stop` followed by the specific container ID after testing with the containers.

As mentioned from the beginning, the docker container commands are used mainly for testing during the process of this project. The following section will discuss how the server communicates with the container.

### 5.3 Server

The server is implemented in Python Flask, which provides a convenient way to pass parameters between the back-end and front-end. There are four major routes/pages, the home page, tutorial page, user’s project page and login page. Flask has a method `render_template`, which can locate the HTML file in the 'templates' folder and render it. The home page renders the HTML script that describes the system’s purpose and functionality. However, implementing the tutorial page and the user’s project page is challenging. These two pages have the same layout and features, except on the user’s project page, there is only the bare bone of the program structure provided. This project’s most essential yet challenging feature is the communication between the server and the users’ docker containers.

#### 5.3.1 Communicate with Docker Containers

At the first stage of development, the docker container commands are considered to be used in the server directly. The os module for Python provides an interface that the server
can interact with the operating system. The original idea was to use this module to create and communicate with the docker containers — however, the communication stalls when one container is running. The os module cannot support entering commands into the running container’s terminal, which means that the user’s code cannot be executed after being copied to the container. Then, docker-py, a docker engine API, is discovered, which can simplify the communication procedure.

When a user first submits their code, they are assigned a UUID. Their submitted code will be stored as a file using their UUID as the filename. Then the system will call the docker client to create a container for this user. The client is instantiated by the from_env() function of the docker library. It is configured from the local environment’s variables, the same as those used by the Docker command-line clients. The client has an object called containers used to manage all the containers on the server. The container object has the same methods as using the terminal command line. For example, the run() method for the client containers works the same way as entering docker run in the system terminal. The flags used in the command become parameters in the docker API method.

The procedure of starting the container and copying and executing the file in the container on the server is the same as testing the containers using the local machine terminal. The run() method in client.containers returns a container object which keeps all the essential information as its attributes, including its ID and name. The os module is used to copy the user code file to the container using the name attribute of the container object. This is because no method in the API supports copying the files from the local machine to the container. The code snippet below shows the start of a new container.

```python
client = docker.from_env()
container = client.containers.run("user_name/repo_name:tag",
                              detach=True, tty=True, stdin_open=True)
os.system("docker cp user_file/directory "+container.name+":usr")
```

The exec_run() method of the container object allows the server to pass the command to run the Python Flask script in the container. However, this method does not support the standard instruction of cd to change the working directory. Therefore, the complete directory of the Python file needs to be passed in the command. But this is not the only problem. As soon as the container starts to run and the Flask file is executed, the whole system goes into a stalled status because it focuses on running the Flask server in the container. Only when the Flask server gets interrupted the system server starts to work again. This problem would result in only one user can use the system at one time, which is not ideal.

Then threading is considered to solve this problem because if the user’s container Flask
server is running in one thread and the system’s central server can run in another, they won’t interfere with each other. Python has a threading module that can work for this purpose. The thread for a container can be created by calling the instructor `Thread` of the threading module with a `run_container` function as its target. The container ID and the to be executed filename are passed into the `run_container` function. The function uses the ID to retrieve the container object and then executes the command to run the Python script in the container. After creating the thread, the method `start()` in the threading module can let the thread begin to run. The host machine CPU will assign time slots for each running thread so that they can work at the same time. The code snippet used by the Docker command line is shown below.

```python
# in create_container function
x = threading.Thread(target=run_container,
                     args=(session["containerID"], filename))
x.start()

def run_container(containerID, filename):
    container = client.containers.get(containerID)
    container.exec_run("python3 usr/" + filename, tty=True)
```

It can be noticed that the container ID parameter passed to the target function is saved in a session dictionary when creating the thread. This session used is the Flask session which will be discussed in the next section.

### 5.3.2 Flask Session

Flask session is an extension of Flask. It can be used to store the user’s information. This system only uses local storage rather than a database. However, some user information must be stored during the session to avoid multiple users sharing the same container. The session indicates the time between a user’s login and logout of the system. A container is assigned to the user within the session, and they can keep using it until they log out or run out of session time. Cookies are how the user’s information is stored in the browser, and the sessions store the data on top of the cookies. The server also encrypted the data using a secret key configured in the Flask application. Using the session means that during a particular period, if a user closed their browser by mistake, when they open it up again, the system will still remember their session, and they can continue practising.

This system limits the time of each session to three hours due to avoid the running of unused containers. The time can be set during the application configuration process.
Apart from the session lifetime, the session type also needs to be configured. It represents how the session interacts with the server, i.e. where the users’ information is stored. This system only uses local storage. Thus the session type is set to be a filesystem. The configuration process is shown below, and the session object is created by loading the configured app into it.

```python
app = Flask(__name__)  # create the flask application instance
app.secret_key = "this is a secret key!"
app.permanent_session_lifetime = datetime.timedelta(hours=3)
app.config["SESSION_TYPE"] = "filesystem"
Session(app)
```

The utilisation of the Flask session is simple. It is used as a dictionary object. When a user inputs their username, it would be stored in `session["username"]`. A random UUID will be generated and stored in the session as well. After the user’s initial code submission, the system needs to start a container for the user. The container ID and IP address will be saved in the session. The keys to the stored data can be customised. Section 5.4.1 will talk about retrieving the container’s IP address.

The system will remove users’ information when they log out or reach the time limit. The `pop()` method is used for this purpose. The server will pop the username and user ID and check if they have any assigned containers. If so, the system will stop the container and pop the container information from the session. The code snippet below gives a simple example of a Flask session.

```python
@app.route(’/login’, methods=["GET" ,"POST"])
def login():
    if request.method == "POST":  # get user input
        session["username"] = request.form.get("username")
        return "Hello " + session["username"]
    return render_template("login_form.html")

@app.route(’/logout’)
def logout():
    session.pop("username",None)
    return f"You are logged out"
```
5.3.3 File Downloading

The file downloading feature uses a library named shutil, which provides methods to manipulate files on the machine. When the user clicks the download button, the system will copy the user’s script to a folder with the basic Flask file hierarchy using shutil.copyfile. Because the easyHtml API and other helper scripts are also included in the directory, the folder needs to be zipped to be ready to send to the user’s local machine. The make_archive method in shutil can help with this. Then the final step is to use Flask’s function send_file as the response to this download request, and the archived file will be downloaded to the user’s selected position. The parameter as_attachment is set to be True so that the file will be downloaded rather than displayed to the user.

There was a challenge that the file downloaded to the machine could not be updated during the test. In other words, the system cached the first downloaded file. The cache_timeout parameter in Flask send_file can be used to set how long the file will be cached in the system. It is designated as 0 in this work to allow users to download a new zipped file when they want.

5.4 Front-end

The front-end design for this system looks relatively simple. An in-browser editor and a browser simulator form a web-IDE. But several challenges are encountered while implementing. This section is going to discuss the challenges and the solutions applied.

5.4.1 Browser Simulation

Since this system aims to teach the users to program in Python Flask, a web framework, it is essential that the user can interact with the resulting web page in the browser. Then simulating the browser seems to be a good idea. An inline frame is used to embed an HTML document inside another HTML. The embedded HTML can be a static file from the local machine or a hyperlink. Then the question becomes how to fetch the particular URL for a specific container.

When a Python Flask server is running, it always provides an initial URL that is accessible. But when the server is running on the container, it is nearly impossible to retrieve the terminal messages. Another problem is that the container Flask server cannot be set to run on localhost because it is the same host hosting the system, and the port will be already in use. Python Flask provides a way to make the server public under the same network, using 0.0.0.0 as the host. All the tutorials script running on the containers are using this host so that it can be accessed externally by using the container IP.
A docker container command `docker inspect` is used to obtain a container’s IP address. This command can return low-level information about the container object given the container ID. The way to acquire the container IP address is shown below. The IP address is saved in the network setting of the container.

```bash
docker inspect -f '{{range .NetworkSettings.Networks}}\n  {{.IPAddress}}{{end}}\n' containerID
```

The default port that the Python Flask server uses is port 5000. Then the URL for the user’s web page can be formed with the container’s IP address. The URL should be like `http://container_ip:5000`. This URL is used as a parameter of the iframe in HTML, and then the resulting page will be displayed in the browser.

However, simply passing the URL to the iframe is not enough. During the test, it was found that it takes at least two submissions before the resulting web page is displayed in the iframe window. If the system sleeps for 1 second using `time.sleep(1)`, it seems to work as expected. This is because when the first time a user submits a script, the Flask server in the container needs time to start. Unfortunately, there is no specific information about how long exactly a Flask server requires to start and run. Therefore, the only solution is to keep trying to connect the URL until it gives a 200 response which means the server is ready.

The first attempt at this solution is to repeatedly send requests to the URL in a loop and check the responses. But it quickly reaches the maximum limit of retries, which means manually retrying to connect is not the best practice. A Retry module in urllib3, a user-friendly Python HTTP client, is discovered to use. A retrying strategy can be used with the Retry object by tuning the `connect` and `backoff_factor` attributes. This system sets `connect` as three, which means it tries to connect the given URL three times. It sets the `backoff_factor` as 0.5, which means it waits for 0.5 seconds before next time trying to connect. Though the backoff delay seems to have the same function as letting the system sleep, using the retry object is more appropriate because it is uncertain how long it needs to wait. After configuring the retry strategy, it is applied by the HTTP adapter and mounted to the request session to be used. The code snippet is shown below.

```python
requests_session = requests.Session()
retry = Retry(connect=3, backoff_factor=0.5)
adapter = HTTPAdapter(max_retries=retry)
requests_session.mount("http://", adapter)
requests_session.get(url)
```

When a user already has the container server running and submits another piece of
code. The system copied the code to the container in the background with the Flask server still running. Since the code script has the same name as before, it won’t influence the Flask server. Instead, it will only update the code. However, a similar problem occurs. The user needs to submit the code twice to see the update in the display window. The same reason that Flask takes time to run should cause this. The URL used is already connected, so it does not need to retry to connect to the server. Using the `time.sleep(1)` again can solve this problem, but there are no alternative solutions yet.

From the picture in section 4.3.1 it can be noticed that there is an input box above the display window. This input box imitates the URL input from a regular browser. When the user’s server starts to run, the home page’s URL will be shown in the input box. Users can modify the URL to check different pages they created. This input component is implemented using forms from HTML. When the user clicks the ‘OK’ button and submits the input form, it will send a POST request to the system server. Then the server will read from the request form, get the user input URL, and render the new page in the iframe window.

### 5.4.2 Navigation Bar

A navigation bar is a bar that is generally on the top of a web page with tabs that can be clicked and lead the users to different pages of the website. The navigation bar designed for this system uses another extension of Python Flask — Flask-nav. It eases the implementation process of the navbar.

The element `View` is one of the crucial links between the functions and the tabs in the navbar. The first argument that `View` takes is a String that will be the tab’s name. And then, it takes the connected function name and parameters (if any) to connect with the endpoint. It works the same as the `url_for()` method in Flask. The arguments in “View” will pass on to the `url_for()` method to get the link.

`Subgroup` is another element used for the navbar in this work. It consists of multiple views to form a menu. This system provides nine tutorials for the user to learn and practice. The tutorials build up a tutorial menu in the navigation bar.

After setting up all the views and subgroups, the `Navbar` element is used to assemble the top level of the navigation bar. The code snippet below shows the example of using Flask-nav to implement the navigation bar.

```python
nav = Nav(app)  # initialize
@nav.navigation('my navbar')
def create_navbar():
    home_view = View('Home', 'home')
```
As can be seen, in the decorator, the string 'my navbar' is a name assigned to the navbar used in the HTML file. This system does not implement much CSS script. Instead, the Flask-bootstrap extension is used to make the web application look attractive. Each HTML file is extended from the base file in bootstrap. There is a block named navbar, in which the customised navbar will be added. It needs to locate the object first by its name, 'my navbar' in this case, and then render it in the block. The code snippet is shown below.

```
{# block navbar #}
{{ nav.my_navbar.render() }}
{# endblock #}
```

### 5.4.3 Counting Down Timer

Since each user will be assigned a container to use this system and their time limit is 3 hours, it is convenient for the learners to know the time they left in their sessions. A counting down timer shown on the page is designed for this purpose.

When a user logs in, the time will be stored in the session as the start time, and it will also add on three hours and be stored as the finish time. Then the server will check if the current time reaches the finish time before every request is sent using the decorator `@app.before_request` from Flask. The user will be logged out automatically if the time expires.

As for displaying the time, JavaScript code is added to the HTML. The server will pass the user’s finish time to the HTML templates. And on the front end, JavaScript will calculate the time this user left and convert the time difference into “hours: minutes: seconds” format and display it on the screen. The variable type stored in Flask Session is Python’s DateTime, and it is required to be converted into Date type in JavaScript and use the `getTime()` method to retrieve the time with the correct format.

This system provides the learners with tutorials to help them learn about Python and Python Flask. The tutorials apply the basic syntax that they should have learned from Pytch and also expand the knowledge that the students would meet in real Python programs. The following section will explain the implementation of the tutorials.
5.5 Tutorials

The tutorials provided for the users are designed to create a simple blogging web application. Each tutorial teaches the user a piece of Python Flask code and related web server design knowledge. The objectives of the tutorials are shown in the table in section 4.3.2. Flask session and Flask-nav extension are both used in the tutorials. However, the design of tutorials aims to lead the learners one step further to be able to develop applications using Python. Basic knowledge like the usage of global variables and decorators are already introduced in Pytch. Therefore, these tutorials will teach the users more complex Python syntax, including data structures and web server requests and responses.

The first four tutorials teach the basic syntax of Flask. The rest of them contain more complex Python syntaxes. The fifth tutorial shows the learners how to retrieve information from a form request by their variable names and store them in a dictionary structure. And the sixth tutorial demonstrates reading a file input instead of a general text blog. When there is a file input request, the server needs to read from request.files rather than request.form. Flask has a method called secure_filename(), which takes a filename as a parameter and converts it into a secure version. Then the file can be safely stored in the filesystem. The path.join method in the os module is used to form the complete directory of the location where the file will be held. The implementation of uploading files with Flask is shown in the code snippet below. Since the tutorial code will be copied to the container, the directory of the upload folder must be set in ‘usr’ accordingly.

```python
if 'file' in request.files:
    file = request.files['file']
    filename = secure_filename(file.filename)
    saved_filename = os.path.join(UPLOAD_FOLDER, filename)
    file.save(saved_filename)
```

To make the resulting web page look more like a blogging application, the like, comment and delete buttons are designed with the posts. This allows the learners to practise and understand more about the dictionary and list data structures and how to maintain them. There are eight keys in the post dictionary: index, content, image, username, DateTime, like, comment_clicked, and comment. The “comment_clicked” is used to check if a specific blog comment button is clicked so that an extra input box will appear, and the user can write their comment in it. Each post dictionary is stored in the blog list in memory. For future updates, tutorials can be added to teach the users how to set up databases to save the blogs in the database.
Except for the POST requests mentioned above, the GET request is also an essential web server component. When a user clicks the blog’s like, comment or delete button, the client will send a GET request to the server with the blog’s index as the parameter. In Flask, using \texttt{request.args} can check the request and using \texttt{request.args.get(parameter\_name)} can get the value of the parameter. For example, when a user clicks the delete button of blog 1, the server will check if the GET request contains an argument named “delete”. If so, it will redirect to a function that can retrieve the particular blog and delete it. The Flask method \texttt{redirect()} and \texttt{url\_for()} are used for this purpose. The example code is shown below.

```python
#main function
@app.route(’/’, methods=[’GET’, ’POST’])
def home():
    .......#some code
    if request.args:
        #check if there is a GET request
        if ’delete’ in request.args:
            index = request.args.get(’delete’)
            return redirect(url\_for(’delete\_post’, index=index))
    .......#some code

@app.route(’/delete<int:index>’)  
def delete\_post(index):
    post\_position = index\_list\_index(index)  #locate the post
    post\_list.pop(post\_position)  #delete the post
    index\_list.pop(post\_position)  #delete related index
    return redirect(url\_for(’home’))
```

To design a web application, there have to be HTML scripts implemented. Therefore, an easyHtml API is developed to help the users focus on learning about Python Flask. The next section will discuss the implementation of the easyHtml API.

### 5.6 The easyHtml API

The easyHtml API is developed for the users to design their web page in Python script. When the user creates an Easyhtml constructor, an HTML file will be created in the templates folder. The HTML file is extended from the base file of the Flask-bootstrap extension. It includes the basic HTML structure and multiple blocks. The title block, styles block, navbar block and content block are the four blocks that this API supports.
to modify. The user can add the navbar block and styles block as a whole, which means they cannot edit the script. The users can only design in the content block.

The usage of the API is relatively simple due to the easy-to-understand method names. However, the users need to remember to add the ending tag when editing certain tags. In HTML, the tags are in pairs, and in the Jinja2 template, when finishing implementing one block or one statement, it needs to end with an end statement.

The constructor of the API takes a string as a parameter to use as the filename, and it also maintains a String type property. The user-designed HTML script will be added to the string property in each method. And at the end of editing, a write() function has to be called so that the string will be written into the file and form a proper HTML script. A sample code from the API is shown below.

```python
class Easyhtml:
    def __init__(self, name):
        # constructor
        self.name = name
        os.system("touch usr/templates/"+name+".html")
        self.string = "{% extends 'bootstrap/base.html' %}\n"

    # function to add a string as title
    def addTitle(self, title):
        self.string += "{% block title %}\n" +
        title + "\n{% endblock title %}"

    # function to write the HTML file
    def write(self):
        file = open("usr/templates/"+self.name+".html", "w+")
        file.write(self.string)
        file.close()
```

5.7 Summary

This section comprehensively presents the proposed system’s implementation. Essential docker knowledge has been acquired from the development. Though the connection between the server and the docker containers had a few challenges and problems, the overall system operates appropriately. Furthermore, the designed tutorials can help the users to understand and improve their knowledge of Python and Flask. The next Chapter will evaluate this system and the completion of the objectives of this project.
Chapter 6

Evaluation

6.1 Overview

The previous chapter describes the implementation process, which marks the completion of the development of this proposed system. However, it is critical to evaluate and review the project after the accomplishment. Therefore, this chapter first will review the goal and objectives set in the first chapter, then discusses the system’s strengths and weaknesses and compares it with the existing web-IDEs. At last, a plan for future work will be presented.

6.2 Aim Review

This system aims to assist the transition process for the users who want to move forward from Scratch and Pytch to Python. Both Scratch and Pytch encourage the users to write event-driven methods, which means the program will do something when a particular event happens. This proposed system, Plaskon, inherits this scheme by teaching the learners about web development because the request and response structure between the server and client in web development is also event-driven. Furthermore, Scratch and Pytch’s programs provide visualised outputs, including animations and interactive games, which motivates the users to be more interested in learning programming. And with Plaskon, the users can learn about developing web applications, which are still visualised outcomes but more practical and realistic. Unfortunately, this system did not test with real users due to the time limit. Therefore, theoretically, this designed and implemented web-IDE achieves the overall goal of helping the students move on from Pytch to Python.
6.3 Objectives Review

The objectives set at the beginning of this work are listed below as a reminder.

Research stage:

• Identify a suitable application domain for web-IDE for a sensible next step for Pytch users

• Research about existing web-IDEs and how they operate

Development stage:

• Develop a web-IDE that allows users to write, compile and run Python Flask in the browser
  
  – Research about technologies that provide servers that the users can use to run their Flask applications
  
  – Implement tutorials that can teach the users about Python Flask
  
  – Implement an API that users can use to write HTML script in Python

The research shows that motivation and language syntax are two of the main challenges for young students’ programming education. Though it is possible for the students to learn to program using text-based programming languages directly, the teaching method is preferred to include visualised explanations to keep them more interested. Visualisation is a powerful tool to help the students concentrate and be motivated. Thus, web development is a suitable domain for this web-IDE for learners seeking to move further from Pytch.

The existing web-IDEs are mainly built to improve three different types of interactions, learner-content, learner-learner and learner-instructor. This system contributes to improving learner-content interaction by providing the learners with an environment and related tutorials to study and practice Python language further. The usage of web-IDE can successfully avoid users’ distractions by eliminating the process of downloading, installing and configuring users’ local machines.

A web application generally requires at least one machine to host the server so that it can communicate with the end-users. This means that this implemented web-IDE needs to select a technology that supports users hosting their web server. Container technology and Virtual Machine are the two options, but the previous one is chosen because only the operating system and relative dependencies need to be virtualised for this work. Docker is used to working with the containers. Each user’s container is built upon a designed Docker image with Alpine as the base operating system.
The tutorials developed for this work guide the users to build a simple blogging website step by step, introducing them to more Python syntax and explaining the web development concepts, including requests and responses. The corresponding designed easyHtml API ensures the learners focus on the Python language practising. It also demonstrates to the users how to apply an API in the Python code.

Overall, this work achieves all the objectives set at the beginning. There are a few challenges related to connecting the server and container more efficiently, but the system works as expected.

### 6.4 System Reflective Evaluation

This proposed system is designed and developed under limited time constraints. Like every application, it has its strengths and weaknesses.

**6.4.1 Strengths**

The first strength is that this system is developed as a web-IDE. It saves the users’ time and energy to install packages and configure local IDEs. It is friendly to those learners who are at a young age or not computer science majors.

Another advantage is that this system helps the users to learn Python by teaching them about web development. This is a feature that most of web-IDEs do not have. However, the powerful platform Replit supports this feature. Replit also provides IDEs for different programming languages. Though this system only supports Python, it gives the users an environment where they can concentrate on their learning. It is helpful for those students who need an easy-to-use and less distractive environment. This system also displays the users’ resulting web page in a small window with an extra user’s input area to manipulate the URLs like in the browser itself.

The system provides an environment for the learners to develop their web applications. Moreover, it also produces a series of tutorials and an easyHtml API to help the users learn the web development. Compared with other web-IDEs, this feature is inherited from Pytch, which assists students in learning programming goals in a true sense.

**6.4.2 Weaknesses**

Section 4.2 introduces the system architecture, which mentions that this system uses local storage which could be one of the weaknesses of this work. From a user perspective, it is convenient if an educational system can record the learning process so that they can pick
up where they left off every time they log in. The proposed system also has a feature that
the learners can design their own web application freely in the “My Project” section, and
it will be helpful if their project can be saved in the system. A proper database will be
needed instead of local storage to achieve these purposes.

Another weakness this system has is that there is no testing with real users for feedback
due to time constraints. It is only tested multiple times locally using Ubuntu 18.04 system.
The testing would be the first thing to do if a longer time is granted.

6.4.3 Comparison with Other Web-IDEs

After implementing this web-IDE, the question becomes why the users should choose
Plaskon over the others. Before comparing with the other web-IDEs that support compiling
Python language, it is essential to mention it again that this work is the second step
to assist the students who are willing to transition from Scratch to Python. The first step,
Pytch, is introduced in section 2.3. The users of this project should already have the basic
knowledge of Python syntax, including classes, decorators, variables, conditional state-
ments, and others taught in Pytch. All the web-IDEs from the research are standalone
environments for various purposes rather than developed upon another application.

Replit is the most competitive web-IDE, which has been mentioned multiple times
already. However, the advantage of this work over replit is that it is not as powerful
as replit so that the users can focus on the learning process. An easyHtml API is also
provided to assist the users in writing the HTML scripts to improve the user’s concentra-
tion on Python. If a user is willing to learn web development more than practice Python
language itself, they have the choice to use replit instead. This work can be improved to
support users implementing in HTML and other front-end languages, but it will be off
track from the original purpose.

Strype [33] is a new platform that aims at assisting users in transit from block-based
programming to text-based programming, more specifically, Python. It uses a frame-
based method that provides the Python syntax structure and allows the users to fill in
slots. It is easy to use and helpful for the learners during the transition process. However,
Pytch can already teach more Python syntax than Strype, for instance, the usage of the
Class. This system, Plaskon, as the second step following Pytch, can provide the users
with even more knowledge of Python. Moreover, the outputs of programs in Strype are
the same as in local IDE, which can help the students to get familiar with text editors.
But to keep the learners motivated, Pytch and Plaskon’s visualised outputs have more
advantages.

Additionally, this system allows the user to download their code and the Flask file
hierarchy and the API provided. Strype has features to convert the users’ programs into Python code, but it is only limited to one Python script.

Though the users can choose any web-IDEs or applications that match their purpose, this proposed system has significant advantages over the others for learners who want to transition from Pytch to Python.

6.5 Future Work

As discussed above, this system has some weaknesses, but they can be improved over time. The first thing that should do if there is more time is the testing. The feedback from testing with real users will benefit the improvement process. The storage solution can consider using a database in the future. A cloud database could be applied for better practice if the system were deployed to a cloud service. But it needs to be considered that the server must be deployed on more than one machine. Otherwise, a single point of failure may occur, influencing all the users.

From the assisting programming education perspective, there are also a few things that can be improved. Kakeshita et al. [10] developed a tool named pgtracer, which provides feedback functions to the users for fill-in-the-blank questions in a source program. This is similar to Strype, but it is more flexible because the blanks can also be in the Python syntax structure. Applying this feature to the proposed system can help students evaluate their learning.

This system provided an easyHtml API so that the learners could write simple HTML scripts in Python. However, this API still has space to be refined, and if there is a similar API for CSS, it will be more attractive to the students keen on the designs.

6.6 Summary

This chapter evaluates the implemented system, Plaskon. The system performs well overall, though there are a few spaces to improve. Moreover, compared with other web-IDEs with similar features, this system presents its benefits by providing attractive outputs, tutorials and API. If a longer time is granted, testing can be done with real users, and databases will be considered to use.
Chapter 7

Conclusions

With the increasing interest in learning programming from a young age to develop logical thinking, web-IDEs play a vital role. Therefore, the requirements to assist the students in transitioning to programming in text-based languages like Python are high. The proposed system Plaskon is a web-IDE designed to guide the students to learn Python further by teaching them to develop web applications, following a previous project Pytch. The system has been built, and the technical infrastructure works well. The next step will be to evaluate with users.

Python is a popular programming language with a large variety of packages. Mastering Plaskon is still the start of learning Python. It will be interesting to expand the features of the web-IDE to explore more suitable packages for learners.
Bibliography


[4] Docker overview. [https://docs.docker.com/get-started/overview/](https://docs.docker.com/get-started/overview/)


