Implementations of Web Application Systems for Docker Image Generation and Flutter Programming Exercise Answer

March, 2025

Lynn Htet Aung

Graduate School of
Natural Science and Technology

(Doctor's Course)
Okayama University

Dissertation submitted to
Graduate School of Natural Science and Technology
of
Okayama University
for
partial fulfillment of the requirements
for the degree of
Doctor of Philosophy in Engineering

Written under the supervision of

Professor Nobuo Funabiki

and co-supervised by
Professor Satoshi Denno
and
Professor Yasuyuki Nogami

Okayama University, March 2025.

To Whom It May Concern

We hereby certify that this is a typical copy of the original doctor thesis of
Mr. Lynn Htet Aung

Signature of

the Supervisor

Prof. Nobuo Funabiki

Seal of

Graduate School of

Natural Science and Technology

Abstract

Nowadays, *web applications* built with free and accessible open-source technologies play crucial roles in university education by improving their efficiency, scalability, and accessibility. They can reduce costs and integrate tools effectively to create dynamic and accessible educational environments through enabling collaborative learning.

In this thesis, I present implementations of two web applications using various open source software including *Docker*. The first web application is the *Docker image generation assisting tool (DIG-assist) for user-PC computing (UPC) system*. This tool adopts *Angular JavaScript* for creating user interfaces, *PHP Laravel* for handling logic and data models using *Rest API*, *MySQL* database, and Shell scripting for running the whole program.

Another web application is the *answer platform in Flutter Programming Learning Assistant System (FPLAS)*. This platform incorporates *Node.js* and *Express* frameworks for the user interface, the *Flutter/Dart* framework for the language environment, the *Nginx* web application server for automatic UI image capturing, and an image-based *User Interface (UI)* testing tool for image comparison.

As the first contribution of this thesis, I present the *Docker image generation assisting tool (DIG-assist) for user-PC computing system (UPC)*.

As a high-performance, low-cost distributed computing platform for the use in university education, the *User-PC computing (UPC) system* has been studied in this group. It is based on the *master-worker* model. The *master* accepts the computing jobs from users and assigns them to *workers* for processing. To run various applications for jobs as isolated containers on personal computers (PCs) that may have different environments as the *workers*, the UPC system adopts the *Docker* technology to bundle every necessary software into a single *Docker* image when the *master* sends the job to a *worker*.

Previously, *Docker* images in the UPC system were generated manually through a multi-step process. First, a *Docker file* had to be created as a text file by manually typing the required *Docker* instructions for the computing job. However, the previous system did not provide a dedicated user interface for this process. Then, the *Docker image* was generated based on the *Docker file* instructions and was saved as a *tar archive* file using the *Docker* save command. These steps were time-consuming and required constant user involvement. Therefore, a tool to assist with *Docker* image generation is essential to make the UPC system more accessible to users, regardless of users' knowledge and experience on *Docker* technology.

In the implementation, to create a user-friendly dynamic *Graphical interface (GUI)* on a web browser, the *Angular JavaScript* framework was used for the client side. For handling logic and data models on the server, the *PHP Laravel* framework was adopted. Data transfer between the browser and the server were accomplished using *RestAPI*. The data is stored and managed through a *MySQL* database. The *Docker* image generated by the tool is speedily uploaded to *Docker Hub* using *Shell Scripting* for the entire process. For evaluations, I collected 30 *Docker* files by reverse-

engineering Docker images from *GitHub* and generated the *Docker* images using the proposal. The results validated the effectiveness of the proposal.

As the second contribution of this thesis, I present the *answer platform* in *Flutter Programming Learning Assistant System (FPLAS)*.

The *Flutter* framework with *Dart* programming has become popular due to its ability to create applications for mobile, web, and desktop environments using a single codebase. To assist novice students, my group has developed the *Flutter Programming Learning Assistant System (FPLAS)* for use in university education. *FPLAS* is implemented in *Visual Studio Code* and runs within a *Docker* container. Students will modify *Dart* source codes in *Visual Studio Code* based on the exercise instructions. They can click the run button to view the output as a UI image. After capturing the image on their computers, they submit both the source code and the image to the teacher through *Moodle*.

To help teachers check submissions from students, my group has developed an image-based *User Interface (UI) Testing* tool using *Flask Python* framework. It compares students' answer images with the correct images using *ORB* and *SIFT* algorithms in *OpenCV*. The results are stored in a *CSV* file. However, if students submit images that are blurry, incomplete, or have incorrect backgrounds, this tool is hard to compare them accurately. In such cases, teachers must crop and adjust the images manually, which is a time-consuming job. Besides, the previous system had separate implementations for students and teachers. This setup caused inefficiencies and delays. To integrate the implementations, I designed a unified web-based answer platform that can provide automatic feedback with the correct answers during assignments.

In this implementation, the four *Docker* images are managed by *Docker Compose* configuration. This implementation reduces teacher workloads and provides more efficient solutions for students. It incorporates the *Node.js* and *Express* frameworks for the user interface, the *Flutter/-Dart* framework for the *Flutter* environment, the *Nginx* web application server for automatic UI image capturing, and an image-based *User Interface (UI)* testing tool for image comparison. For evaluation, I asked 10 graduate students at Okayama University, Japan, to install the implemented answer platform on their PCs and solve five exercise problems. All students successfully completed the tasks, confirming the validity and effectiveness of the proposed system.

In future works, I will develop other functions for handling various *Docker*-based jobs, integrating *GPU* support for the *User-PC Computing (UPC)* system, and improving advanced exercise assignments by incorporating *YOLO* object detection, machine learning, and *AI* into the *Flutter Programming Learning Assistant System (FPLAS)*. Additionally, efforts will be directed toward expanding the scalability and flexibility of the system to accommodate more users and diverse learning environments, ensuring adaptability to a wide range of educational needs.

Acknowledgements

I would like to express my heartfelt gratitude to everyone who supported and guided me throughout the completion of this thesis at Okayama University, Japan. To all who have been part of this journey, I simply want to say that you are the greatest blessing in my life.

First and foremost, I owe my deepest gratitude to my honorable supervisor, Professor Nobuo Funabiki for his excellent supervision, meaningful suggestions, persistent encouragements, and invaluable assistance at every stage of my Ph.D. study. His thoughtful comments and guidance have been instrumental in helping me complete my research papers and present them effectively. Moreover, he was always patient and supportive whenever I needed his advice, not only in my academic pursuits but also in my daily life in Japan. His guidance has truly been a gift, and needless to say, this thesis would not be possible to complete this thesis without his active support and mentorship.

I extend my heartfelt thanks to my Ph.D. co-supervisors, Professor Satoshi Denno and Professor Yasuyuki Nogami, for their continuous support, guidance, insightful suggestions, and proofreading of this thesis. I am also grateful to my former Assistant Professor Minoru Kuribayashi from Okayama University and Professor Wen-Chung Kao from National Taiwan Normal University and all my course instructors for their enlightening knowledge and discussions.

I deeply appreciate all members of the FUNABIKI Lab for their support during my studies and Ms. Keiko Kawabata for her administrative assistance throughout my Ph.D.

I greatly acknowledge the Monbukagakusho Honors Scholarship (JASSO Scholarship) and my sister for providing financial during my Ph.D study.

My special thanks go to Ms. Soe Thandar Aung (my sister) for her valuable advice and unwavering support, which enabled me to start my Ph.D study. I also extend my sincere gratitude to Dr. Hein Htet, Mr. Anass Barrahmoune, Mr. Patrick Ong, Dr. Nyein Myint Myint Aung, Dr. Ei Ei Htet, Dr. Khaing Hsu Wai, Dr. San Haymar Shwe and Dr. Shune Lae Aung and international friends. Your support during challenging times, as well as our shared thoughts and experiences, mean a lot to me.

Last but certainly not least, I am eternally grateful to my beloved father, mother, two elder sisters, younger brother and girlfriend for their unconditional love, unwavering support, patience, and confidence in me, which have been my greatest source of motivation and my ultimate reward. I am truly proud and blessed to have you all in my life.

Lynn Htet Aung
Okayama University, Japan
March, 2025

List of Publications

Journal Paper

1. **Lynn Htet Aung**, Nobuo Funabiki, Soe Thandar Aung, Xudong Zhou, Xu Xiang, and Wen-Chung Kao, "A Web-Based Docker Image Assistant Generation Tool for User-PC Computing System," *MDPI Information*, Vol.14 (6), No. 300, pp. 1-16 (June 2023). DOI: doi.org/10.3390/info14060300.

2. **Lynn Htet Aung**, Soe Thandar Aung, Nobuo Funabiki, Htoo Htoo Sandi Kyaw, and Wen-Chung Kao, "An Implementation of Web-Based Answer Platform in the Flutter Programming Learning Assistant System Using Docker Compose," *MDPI Electronics*, Vol. 13 (24), No. 4878, pp. 1-24 (Dec 2024). DOI: doi.org/10.3390/electronics13244878.

International Conference Paper

3. **Lynn Htet Aung**, Nobuo Funabiki, Soe Thandar Aung, and Wen-Chung Kao, "A Design of UI Image Generation Method for Flutter Programming Learning Assistant System," IEEE 13th Global Conference on Consumer Electronics (GCCE), pp. 42-43 (Kita-Kyushu, Japan, 2024).

Other Papers

4. **Lynn Htet Aung**, Nobuo Funabiki, Hein Htet, Xudong Zhou, Xu Xiang, Minoru Kuribayashi, "An Implementation of Docker Image Generation Tool for User-PC Computing System," IEICE Technical Report of the Institute of Electronics, Information and Communication Engineers (Osaka City Lifelong Learning Center), Vol.122, No.330, SS2022-32, pp. 13-18 (Jan 2023).

5. **Lynn Htet Aung**, Nobuo Funabiki, Minoru Kuribayashi, Haruki Gunji, Yuto Kokubun, and Kiyoshi Ueda, "An Extension of User-PC Computing System for NS-3 Network Simulations," IEICE Technical Research Report of the Institute of Electronics, Information and Communication Engineers (Okinawa Convention Center), Vol.122, No. 406, NS2022-233, pp. 378-382 (March 2023).

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Background of User-PC Computing (UPC) System

As a distributed computing platform, the *User-PC Computing (UPC)* system [1] adopts a master-worker architecture to receive jobs at the *master* and run them on the *workers*. The *UPC master* receives jobs from users manually or from application systems online, as shown in Figure 1.1. Then, it makes the *Docker images* for the jobs so that they can run on *worker PCs* with various environments as *Docker containers*. When a worker completes a job, it sends back the result to the master.

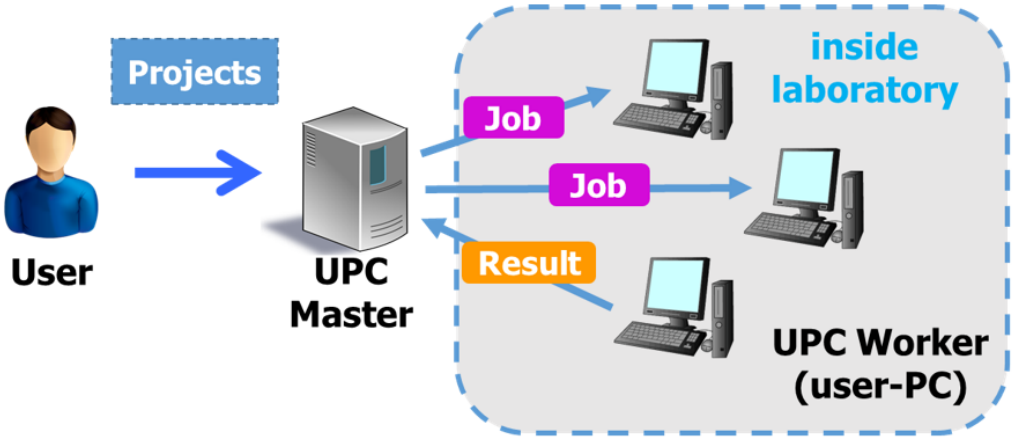

Figure 1.1: Overview of UPC System.

A user of the UPC system needs to prepare the *Docker file* of a plain text file by writing the necessary *Docker instructions* containing the keys and the values. Thus, the user needs to learn how to make a *Docker file* and how to write *Docker instructions* for each application as the online *GitHub* projects. This can become a bottleneck for novice users at using the UPC system.

To generate a *Docker image*, the process involves three steps that are performed manually. The first step is to prepare the *Docker commands* by converting them from the *Docker* file. The user needs to create the commands that will be used to generate the *Docker* image. Next, the user runs the command on the terminal to generate the *Docker image*. This involves executing the prepared commands. Finally, the user saves the generated *Docker image* as a tar file in the file system. This is important for future reuse and to ensure that the *Docker image* is available whenever it is needed.

Overall, the generation of *Docker* images requires careful and precise manual steps to ensure the correct outcome.

Manual generations of *Docker images* has several drawbacks that need to be addressed to improve the usability of the UPC system. Firstly, multiple processes have to be handled manually, making the process tedious and error-prone. Secondly, creating a *Docker file* requires a certain level of knowledge, making it difficult for inexperienced users. Thirdly, the process is time-consuming, which can lead to delays in the deployment of *Docker images*. Fourthly, the storage can become an issue when working with large *Docker* images, potentially leading to storage constraints. Finally, the manual generation of *Docker images* cannot be processed in real-time, limiting its overall efficiency. Addressing these drawbacks is crucial for improving the efficiency and usability of the UPC system.

1.2 Background of Flutter Programming Learning Assistant System (FPLAS)

FPLAS [2] is a *Docker*-based *Flutter* development environment that is designed for novice students to initiate mobile application developments while avoiding the complexities of environment setups. Figure 1.2 shows an overview of FPLAS. It includes preparing a *Docker container* image with the system startup files on *Docker Hub*. Additionally, two system startup files for different operating systems were added to *GitHub*. Sample *Flutter* projects are provided as code modification exercises to guide students in using this environment. Comprehensive instructions on *GitHub* were provided to assist students in effectively utilizing the system, including steps for installing *Docker* and *VS-Code*, downloading the *Docker image*, connecting to the container, accessing the exercise projects, and submitting answer files to the teacher through *Moodle*.

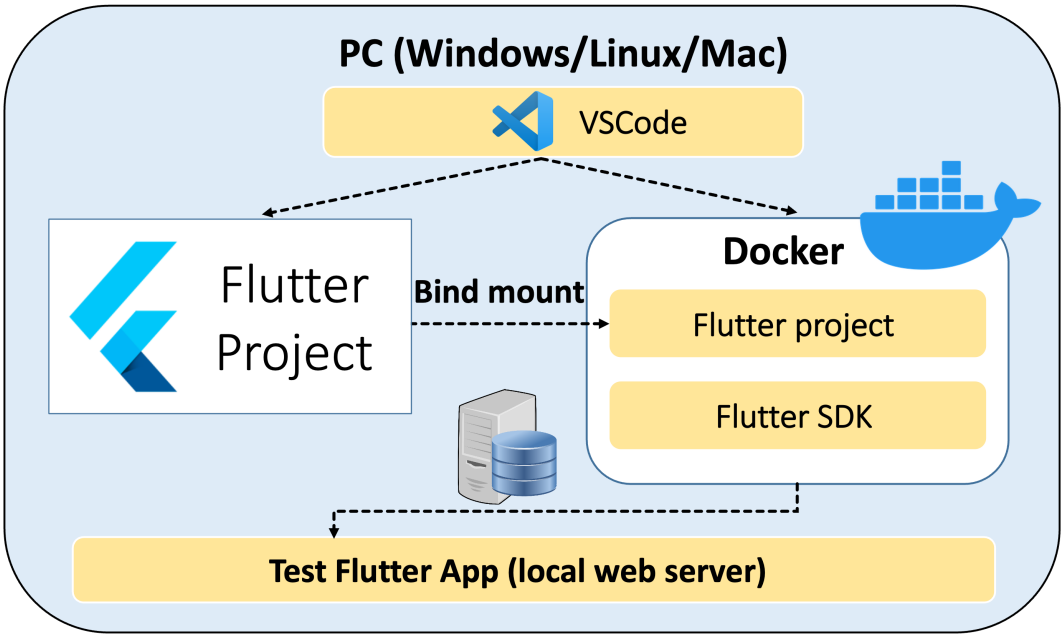

Figure 1.2: Overview of Docker-based Flutter development environment.

1.3 Contributions

In this thesis, I have carried out the following research contributions.

1.3.1 Docker Image Generation Assisting Tool in the UPC System

The first contribution of the thesis is the design and implementation of a web-based *Docker image generation assisting tool (DIG-assist)* [3] for the UPC system. To create a user-friendly graphical interface (GUI) on a web browser, I utilized the *Angular JavaScript* framework for the client side. For handling logic and data models on the server, I adopted the *Laravel* framework on *PHP*. Data transfer between the browser and the server is accomplished using *RestAPI*. The data are stored and managed through a *MySQL* database. The *Docker image* generated by the tool is speedily uploaded to *DockerHub*. I used *Shell scripting* to speed up the entire process. To achieve higher efficiency and flexibility in the implementation, I adopted the *Model-View-Controller (MVC)* design pattern, to handle the client, server, and database separately. I selected 30 popular *Docker* projects from *DockerHub* generated their *Docker images* using the tool to validate our proposal.

1.3.2 Answer Platform in FPLAS using Docker Compose

The second contribution of the thesis is the implementation of a web-based *answer platform* for the *Flutter Programming Learning Assistant System (FPLAS)* [4] by integrating four *Docker* images using *Docker Compose*. The primary contributions of this study are as follows:

Extension of Answer Platform for Flutter Programming: The existing web interface of the answer platform [5] is extended and modified to meet the requirements of *Flutter* programming with five exercise assignments.

Integration of Existing Flutter Environment: The existing *Flutter* environment is integrated to compile *Dart* source code that will be submitted from students through the answer platform interface, with the removal of the *VS-Code* setup.

Adoption of Nginx Web Application Server: A *Nginx* web application server is introduced to automate the capture of UI images generated by student code.

Integration of Existing UI Testing Tool: An *image-based UI testing tool* is incorporated to validate student-submitted UI images by comparing them with reference UI images. This integration automates the validation process, ensuring objective and efficient feedback for students.

Docker Compose Integration for Four Images: The components used in the system are packaged into separate *Docker images* and are combined into one image using a single *Docker Compose* configuration. This setup enables data sharing among the components and simplifies deployment and management within a *Docker container*.

Installation Setup and Learning Speed: The proposed solution reduces the complexity of the installation procedure, ensures a quicker and more consistent setup across diverse operating systems, provides immediate feedback to students, and enhances their learning speeds by allowing them to complete assignments more efficiently compared to the existing *VS-Code*-based *Flutter* environment setup.

1.4 Contents of Dissertation

The remaining part of this thesis is organized as follows: Chapter 2 presents adopted open source software tools. Chapter 3 reviews *user-PC computing (UPC)* system. Chapter 4 presents the design and implementation of the *Docker image generation assisting tool (DIG-assist)* and the evaluations. Chapter 5 reviews *Flutter Programming Learning Assistant System (FPLAS)*. Chapter 6 presents the implementation of FPLAS answer platform and its evaluations. Chapter 7 introduces relevant works in literature to this thesis. Finally, Chapter 8 concludes this thesis with future works.

Chapter 2

Adopted Open Source Software

This chapter introduces the adopted open-source software in this thesis.

2.1 HTML

HTML [7] is the standard language used to create and structure content on the web. It defines the structure of a webpage by using tags or elements, such as headings, paragraphs, images, links, and lists. *HTML* provides the basic skeleton of a webpage, organizing the content and linking it with other resources, such as stylesheets and scripts, to enhance functionality and appearance.

2.2 CSS

CSS [8] is a style sheet language used to control the presentation and layout of an HTML content. It allows developers to specify how *HTML* elements should appear on the screen, such as colors, fonts, sizes, spacing, positioning, and overall layout. *CSS* helps separate the content (*HTML*) from the design (styling), making a webpage more visually appealing and easier to maintain.

2.3 Bootstrap

Bootstrap [9] is one of the most popular CSS frameworks for building responsive and mobile-first web applications. It provides developers with a well-structured grid system, pre-designed components, and JavaScript plugins, simplifying the process of creating visually appealing and responsive web layouts.

2.4 PrimeNG

PrimeNG [10] is a UI component library designed specifically for *Angular* applications. It provides developers with a comprehensive set of highly customizable and feature-rich components that streamline the process of building modern web applications. With its seamless integration with *Angular*'s features like directives, data binding, and dependency injection, *PrimeNG* makes developing interactive and dynamic user interfaces more efficient.

2.5 JavaScript

JavaScript [11] is a high-level programming language used to create interactive and dynamic elements on a webpage. It enables developers to add functionality such as animations, form validations, user interactions such as button clicks, and the ability to manipulate *HTML* and *CSS* in real-time. *JavaScript* runs on the client-side in the user's browser, allowing a webpage to be more responsive and interactive without needing to reload the page. It is an essential tool for modern web developments.

2.6 Angular

Angular [12] is a front-end framework for building single page application (SPA) using *TypeScript*. *TypeScript* is a superset of *JavaScript* and has been developed by *Microsoft*. It comprises a set of seamlessly integrated libraries encompassing a diverse range of functionalities, such as routing, form management, and client-server communications. It also allows an intuitive user interface in a web application.

2.7 EJS

The user interface is developed using *Embedded JavaScript (EJS)* [13], *Bootstrap*, and *CSS*, with managements handled by *Node.js* and *Express.js*. *EJS* serves as a template engine, enabling the seamless rendering of *JavaScript* within *HTML*. It is commonly used with the *Express.js* framework to embed dynamic *JavaScript* codes into HTML templates, facilitating client-side rendering and dynamic content generation.

2.8 Node.js

Node.js [14], an open-source server environment, is compatible with multiple PC platforms such as *Windows*, *Linux*, and *Mac OS*. It serves as an interpreter and runtime environment for executing *JavaScript* source codes on the server. It is versatile, supporting the implementation of both desktop and server applications. Consequently, developers can use the same programming language, *JavaScript*, to create both the front-end and back-end of application systems.

2.9 Laravel

Laravel [15] is an open-source server-side *PHP* framework designed to streamline and accelerate web application developments. It offers a comprehensive set of built-in features, making it user-friendly and efficient. With a modular packaging system and robust dependency management, *Laravel* enables developers to seamlessly integrate additional functionalities without starting from scratch. The framework is built on the *Model-View-Controller (MVC)* architecture, ensuring clear separations of concerns. Additionally, it serves as a reliable back-end for *JavaScript* single-page applications, supporting modern web development practices.

2.10 Python

Python [16] is a high-level, general-purpose programming language known for its simplicity, versatility, and efficiency. It supports multiple programming paradigms, including object-oriented, procedural, and functional programming, making it adaptable for a wide range of applications. Widely used in web application development, data analysis, machine learning, and automation, *Python*'s extensive library ecosystem and readability make it a powerful tool for research and development projects.

2.10.1 Watchdog

Watchdog [17] is a *Python* library that monitors specified directories for file system events, such as creation, modification, and deletion, and logs changes to the console, offering real-time feedback on directory activity.

2.10.2 Playwright

Playwright [18] is an open-source framework for end-to-end web application testing, supporting multiple browsers and platforms. It runs tests in both headless and headed modes, making it ideal for automating browser interactions and UI testing. To enhance this experience, *Playwright* recommends using the official *Playwright pytest* plugin, which provides context isolation and supports running tests across multiple browser configurations out of the box.

2.11 Flask

Flask [19] is a lightweight *Python* web framework designed for simplicity and flexibility. Its modular design allows easy integrations of third-party libraries, making it ideal for both small and large projects. It supports key features for web applications like routing, request handling, and templating, and its active community and documentation provide valuable resources for developers of all experience levels.

2.12 OpenCV

OpenCV (Open Source Computer Vision Library) [20] is an open-source software library that specializes in computer vision and machine learning. Designed to provide a common infrastructure for computer vision applications, it supports a wide range of functionalities, including image processing, object detection, facial recognition, and motion tracking. *OpenCV* is written in *C++* and has interfaces for *Python*, *Java*, and *MATLAB*, making it accessible to a broad audience of developers and researchers. Its comprehensive tools and algorithms enable the fast developments of real-time vision applications, from fundamental image transformations to complex video analytics. Widely used in academia and industry, *OpenCV* benefits from a strong community that contributes to its extensive documentation and continuous improvement, ensuring it remains at the forefront of computer vision technology.

2.13 Flutter

Flutter [21] is an open-source *software development kit (SDK)* and is widely acknowledged for its capability in designing user interfaces, boasting cross-platform compatibility extending to *iOS*, *Android*, web, desktop, and embedded systems. It enables developers to craft applications that smoothly adjust to each platform while optimizing code reuse. Using a widget-based architecture, *Flutter* enables swift development of high-performance apps with features such as *hot reload* for real-time code-to-interface updates. Its extensive collection of customizable widgets and robust community supports foster rapid developments, delivering visually captivating and responsive apps tailored to diverse platform needs.

2.14 Dart

Dart [22] is seamlessly integrated with *Flutter*, a *Google*'s toolkit for creating native apps across mobile, web, and desktop. As the primary language for *Flutter*, *Dart* offers simplicity, efficiency, and performance, enabling developers to build visually appealing user interfaces with ease. With the *hot reload* feature, developers can quickly update running apps, speeding up iterations and experimentation. *Dart* with *Flutter* empowers developers to design high-quality apps with smooth animations, rich interfaces, and top-notch performances across platforms, while reducing development time and maintaining code consistencies.

2.15 MySQL

MySQL [23] is an open-source relational database management system (RDBMS) that is based on *Structured Query Language (SQL)*. It is widely used for storing, organizing, and managing data in a structured format. *MySQL* enables users to create and manage databases, execute queries, and perform data manipulation operations. It supports multiple platforms and provides robust tools for database security, scalability, and performance optimization. Common use cases of *MySQL* include web applications, data analytics, and enterprise software.

2.16 Shell Scripting

Shell scripting [24] is the process of writing and executing scripts in a shell environment to automate tasks on a *Unix/Linux*-based operating system. A shell script is a text file containing a sequence of commands written in a shell programming language such as *Bash*, *Zsh*, or *Sh*. These scripts can perform various tasks, including file manipulation, program execution, and system administration. Shell scripting enhances productivity by automating repetitive tasks, simplifying complex operations, and allowing for quick execution of system-level functions.

2.17 Docker

Docker [25] is a containerization platform that packages applications and dependencies into standardized containers, ensuring consistency across environments. *Docker file* [26] is a script that

defines the steps to build a *Docker image* [27], specifying the base image and required dependencies. A *Docker image* is a portable snapshot of a whole application, including its code, libraries, and dependencies, used to create containers. *Docker container* [28] is a running instance of a *Docker image*, providing an isolated environment for applications.

2.18 Docker Compose

Docker Compose [29] is a tool that simplifies running applications on multiple *Docker containers* with a single configuration file called *docker-compose.yml*. This file defines every container, or "service", that the application needs, including images, connections, shared storage, and necessary ports. With *Docker Compose*, all the containers can be started, scaled, or stopped together, making the management simpler and more efficient. *Docker Compose* is particularly useful for multi-component applications, such as a web server and a database, allowing them to work as a single, coordinated system.

2.19 DockerHub

DockerHub [30] is a cloud-based platform that allows developers to create, manage, and distribute containerized applications. It serves as a centralized repository where users can store and share *Docker images*, enabling seamless collaboration and efficient deployment. With features like automated builds, private repositories, and integration with CI/CD pipelines, *DockerHub* simplifies the process of developing and deploying applications in various environments. It also provides access to a vast library of pre-built images, including official images from trusted vendors, making it an essential tool for modern *DevOps* workflows.

2.20 Nginx

Nginx [31] is a high-performance, open-source web server that also serves as a reverse proxy, load balancer, and *HTTP* cache. Renowned for its speed and efficiency, *Nginx* efficiently handles a large number of concurrent connections while consuming minimal resources. It excels in serving static content, managing dynamic content via backend servers, and balancing traffic across multiple servers. Its modular architecture and scalability make it a popular choice for modern web applications, enhancing performance, security, and load distribution in large-scale systems.

2.21 GitHub

GitHub [32], a web-based platform for version control, plays a pivotal role for developers globally. Utilizing *Git*, an open-source version control system, *GitHub* facilitates hosting both open-source and private repositories. As comprehensive suite of tools, it enables collaborations, allowing developers to efficiently manage and track changes in their code bases. Through features like pull requests, issues, and wikis, *GitHub* encourages a collaborative environment where users can host, review, and manage projects effortlessly.

Chapter 3

Overview of User-PC Computing (UPC) System

This chapter provides an overview of the *User-PC Computing (UPC)* system using *Docker*. It is composed mainly of two components, the *UPC master* and the *UPC workers*.

3.1 UPC Overview

The *User-PC Computing (UPC)* system has been studied to offer a low-cost, scalable, and high-performance computing platform. It is based on the *master-worker model*. The *UPC master* accepts computing jobs from users and assigns them to *UPC workers* to be processed. To run various applications for jobs as isolated containers on different personal computers (PCs), which may have different environments, the UPC system adopts *Docker* technology. This technology bundles every necessary software into a single *Docker* image when the master sends the job to a worker. After that, the worker executes the *Docker* image/job and returns the CPU time and result back to the master.

3.2 UPC Master

The *UPC master* adopts *Python* and *Docker* to provide the functions such as job creation, manual *Docker image* generation, and job submission. By utilizing *Python*'s multi-threading library, it efficiently manages concurrent processes between the UPC master and the UPC worker.

3.2.1 Job Creation and Manual Docker Image Generation

The users upload their applications to the master's job queue. To request running a job in the system, a user needs to prepare a *Dockerfile* and write *Docker* instructions that specify the necessary keys and values for each application. These instructions often reference examples from *GitHub* projects. Once the *Dockerfile* is prepared, it is converted into a *Docker* image using the *Docker build* command. The generated *Docker image* is then saved as a tar file [33] in the file system using the *Docker save* [34] command. This process requires careful and precise manual steps to ensure the *Docker image* is generated correctly.

3.2.2 Job Submission

After generating the *Docker image*, the job is submitted and inserted into the job queue to an available worker in the system. The worker picks up the top job in the queue and processes the job, utilizing its CPU resources. Once completed, the results generated by the workers are sent back to the master for storage and further processing. This process is continued until the queue becomes empty.

3.3 UPC Worker

The *UPC worker* adopts *Python* and *Docker* to provide the functions such as the job acceptance and the job execution. A UPC worker which is connected to the UPC master is assigned its own thread, which runs on a specific CPU core to ensure the efficient job execution.

3.3.1 Job Acceptance

After accepting the *Docker images* for jobs from the UPC master, the UPC worker checks the job type and triggers to run the job as a standalone *Docker* container. The worker verifies that the *Docker image* is correctly formatted and all the dependencies are included. Once validated, the worker initializes the designated *Docker* container and begins the execution process.

3.3.2 Job Execution and Results

The job execution program in the worker loads and runs the received *Docker* image as a container until the process is complete. When the process is completed successfully, the total CPU time and results are saved in a CSV file, including job submission, acceptance, and execution details. Finally, the total CPU time and job results are returned to the UPC master.

3.4 Summary

In this chapter, I presented an overview of the *User-PC Computing (UPC)* system using *Docker*. I discussed the implemented functions of the UPC master and UPC worker. In the next chapter, I will present the design and implementation of the web-based *Docker* image generation assisting tool (DIG-assist).

Chapter 4

An Implementation of Docker Image Generation Assisting Tool

This chapter presents the implementation of the *Docker Image Generation Assisting Tool (DIG-assist)*. The implementation details, based on the *Model-View-Controller (MVC)* architecture, are discussed.

4.1 Overview

Previous works supposed that the *Docker* image is generated manually in the UPC system and the *Docker* file is made as a text file by manually typing the necessary *Docker* instructions for the computing jobs. The dedicated user interface, however, is not offered in the UPC system. Then, the *Docker* image is generated using the *Docker* instructions in the *Docker* file and is stored as the *tar* archive file by using the *Docker* save command on the file system. Altogether, these steps are time-consuming and require constant involvements of users. Therefore, assisting the process of *Docker* image generation is an essential step to make the UPC system handy to a wider range of users regardless of their knowledge and experiences with *Docker* processing.

4.2 Software Architecture

The proposed *Docker Image Generation Assisting Tool* consists of four components: client interface [35], server, shell scripting, and database (Figure 4.1).

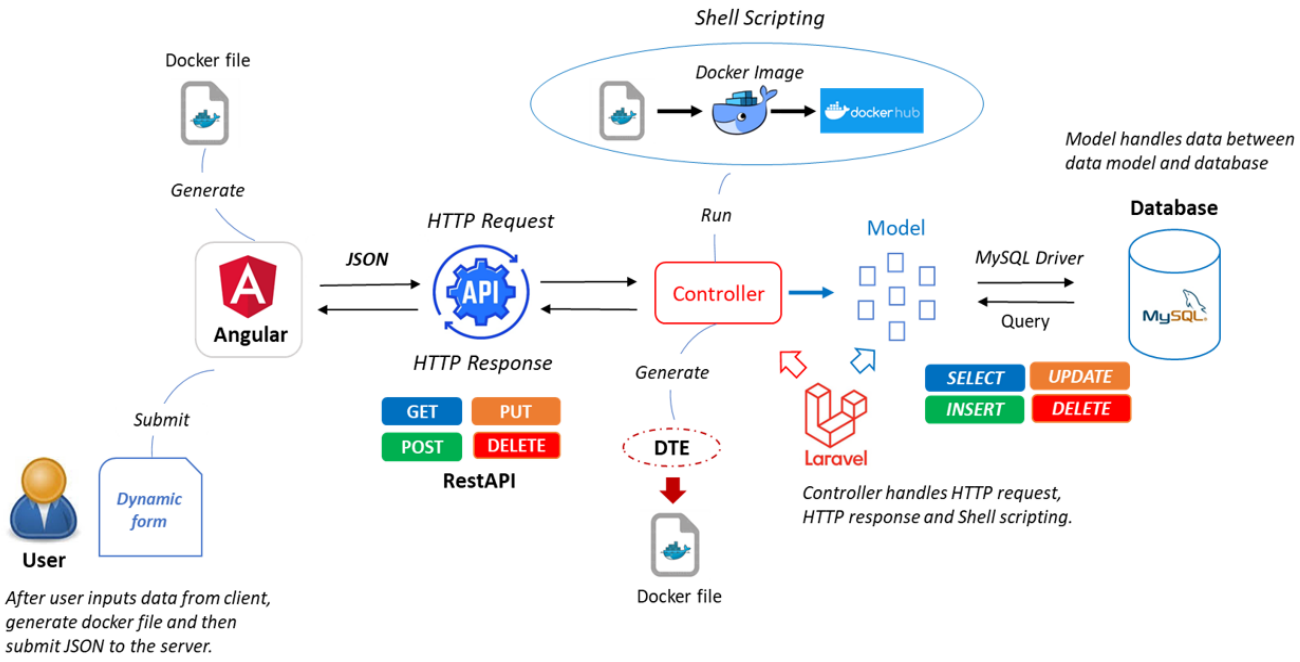

Figure 4.1: Software architecture of Docker image generation assisting tool.

The *Angular JavaScript* framework is used to implement the client interface for *View (V)*. It creates dynamic graphical user interfaces [36] with the *primeNG* and *CSS* frameworks. *Angular JavaScript* is the module component-based framework written by *TypeScript* [37]. It provides a module component-based architecture for creating efficient single-page web applications, with routing, services, directives, configuration environments, a built-in *HttpClient* module, an *AngularCLI* command-line interface, rich built-in libraries, efficient project structures, and suitability for single-page applications (SPA).

The *PHP Laravel* framework is used for *Controller (C)* on the server side to handle logic and data models through *RestAPI* [38]. *Laravel* comes with a host of rich built-in libraries, a separate controller and data model architecture, and a configuration system that supports various services, including database access. Additionally, *Laravel* enables the creation of *RESTful APIs*, which are simple *Application Programming Interfaces (APIs)* that enable interactions with *RESTful* web services. REST (REpresentational State Transfer) transfers data using the HTTP [39] protocol's GET, POST, PUT, and DELETE methods.

Shell Scripting is used to run instruction commands step-by-step to generate the *Docker* image from a *Docker* file and upload it to the designated repository in *Docker Hub* speedily.

The *MySQL* database is selected as the *Model (M)* to store data for the individual *Docker* file instructions for the respective applications. *MySQL* has the schema-based architecture, which facilitates easy data import. It is worth noting that when a *NoSQL* database is used, only the data format needs to be adjusted. *MySQL* provides a straightforward approach to database management, including database creation, table creation, data storage, data export, data import, and data transfer using simple query methods such as inserting, updating, deleting, and selecting.

4.3 Client-side Implementation

In this section, I discuss the installation setup, and the operational flow of both experience and inexperienced users on *Docker* technology.

4.3.1 Installation Setup

To set up the client side, *AngularCLI* [40] must be installed to enable command-line usage of *Angular*. *Node.js* must also be installed using *Node Version Manager (NVM)* [41], which allows for switching between different versions of *Node.js*. *Angular* utilizes a built-in package.json [42] file, with npm [43] collecting all installed libraries into one place.

4.3.2 Operational Flow

The *Docker* manage module comprises five web page components: two creating pages for experienced and inexperienced users, one listing page, and two editing pages for experienced and inexperienced users. These components are implemented with *HTML* for the presentation layer, *TypeScript* for logic and service layers, and *CSS/PrimeNG* for the design layer.

For prior knowledge or experienced users as shown in Figure 4.2, the 'creating' page provides a dynamic user interface that suggests *Docker* instructions. The respective *Docker* instructions for different programming languages are stored in a *JSON* configuration file. When a user chooses a programming language, the instructions will appear automatically as inputs. On this page, the user inputs the job's name, description, operating system, and programming language. The *Docker* instructions are then filled in the respective input boxes and can be modified by the user. Once completed, the user can generate a *Docker* file by clicking the submission button. The *Docker file* is saved on the PC and sent to the server side. Users can edit *Docker* file instructions and generate a new *Docker* file on the corresponding editing page.

Figure 4.2: Creating page of DIG-assist tool for prior knowledge or experienced user.

Without prior knowledge or inexperienced as shown in Figure 4.3, the creating page provides step-by-step instructions on how to create the *Docker* file. On this page, the user inputs the job's programming name, dependency file, project folder, project version number, port number, and application running command. After filling in the requirements, the user can click the submit button to send the data to the server through *HTTP API* methods. Then, the server will filter the requested data and then send them to the *Docker Template Engine (DTE)* to generate the *Docker* file. Users can edit *Docker* file instructions and generate a new *Docker* file on the corresponding editing page.

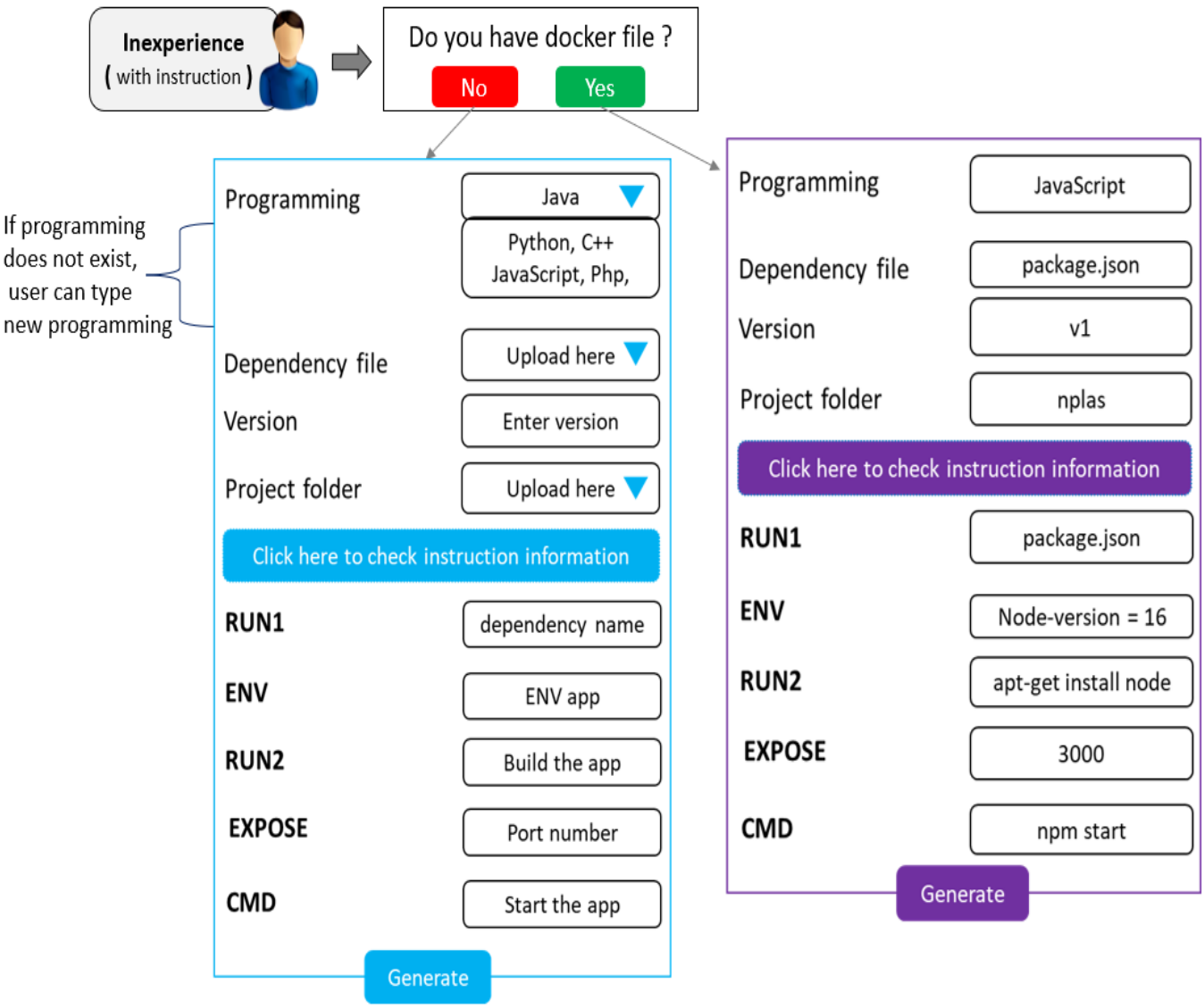

Figure 4.3: Creating and Updating pages of DIG-assist tool for without prior knowledge or inexperienced user.

The listing page as shown in Figure 4.4 displays a table with columns for the name, description, status, and action for each *Docker* file by fetching the data from the the database.

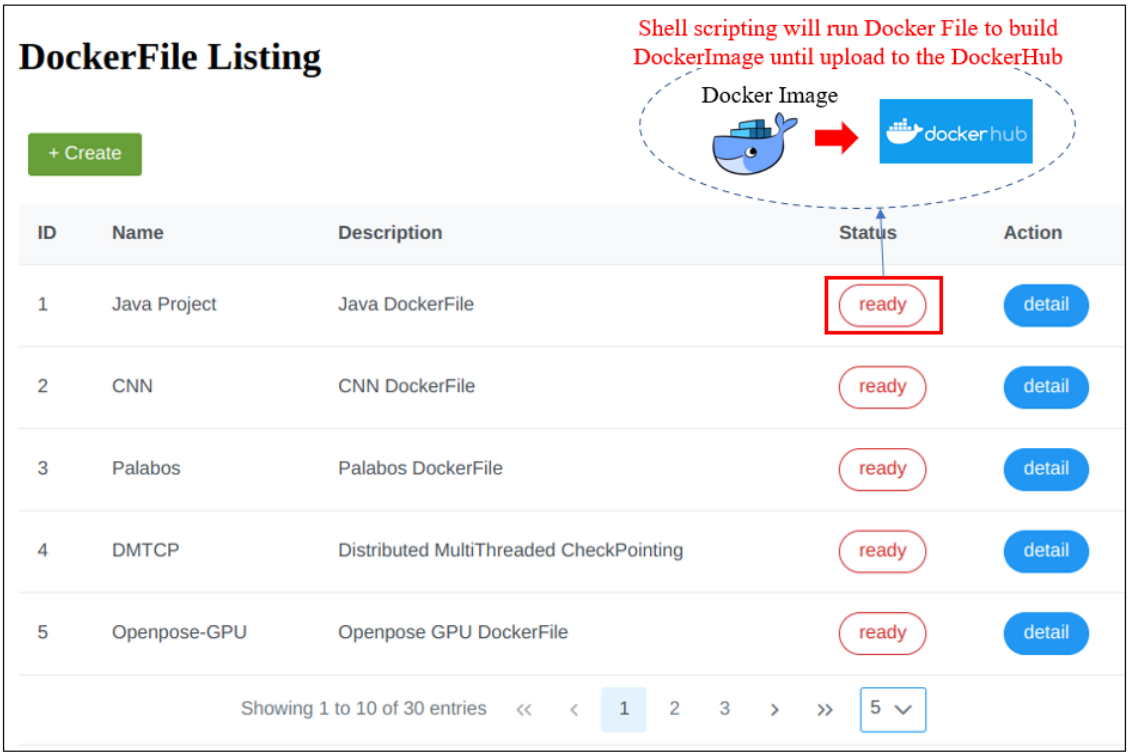

Figure 4.4: Listing page of DIG-assist tool.

The user can search or filter with two option such as name and status in the listing of generated Docker files. The data are displayed page-by-page automatically using pagination. The status column includes a "ready" button that runs *Shell Scripting* to generate a corresponding *Docker* image and push it to *Docker Hub*. The status column includes a "pushed" button that runs *Shell Scripting* to update a corresponding existing *Docker* image and push it again to *Docker Hub*. If the status column changed a "updated" button, it cannot update the Docker file and generate again a corresponding *Docker* image by running *Shell Scripting* to push it to *Docker Hub*. The action column has a "detail" button that redirects the user to the editing page to view the *Docker* file instructions.

4.3.3 Specific Features and Functionality

As for the specific features and functionality of the DIG-assist tool, I offer a range of capabilities for experienced and inexperienced users.

For experienced users, they have access to predefined *Docker* instructions through a *JSON* [44] file, the ability to add and remove new instructions, support for key selection, and a custom function (Listing 4.1) to convert *JSON* format to *Docker* format.

For inexperienced users, the tool offers options for single or multi-programming selection, uploading project folders and dependency files, *Docker* instruction hints, and user guides. Additionally, the *DIG-assist* tool provides one-click submission for generating *Docker* files and image generation with shell scriptings for *Linux*, *Windows*, and *MacOS* operating systems. The tool also includes support for *Docker* image uploading to *Docker Hub* using a *Shell Scripting*. Overall, the *DIG-assist* tool has a lot of useful features that can make *Docker* file creation and management much easier.

Listing 4.1: Custom function.

```
1
2   /**
3       * Format Docker File
4       * @param instructions
5       */
6   formatDockerFile(instructions: any) {
7     let obj: any = [];
8
9     instructions.forEach((element: { key: any; value: any }) => {
10      let keys = element.key;
11      let values = element.value;
12      for (var i = 0; i < instructions.length; i++) {
13        obj[keys] = values; // modify key pair style
14      }
15    });
16
17    let str = JSON.stringify({ ...obj }); // object to JSON
18    let format = str.replace(/[{}]|@@/g,'').replace(/":"/g,'')
19            .replace(/","/g,'\n').replace(/"/g,'');
20          return format;
21    }
```

I developed five web pages to support the creation, listing, editing, and deletion of Docker files. To address the *CORS* issue between the client and server, I added API support and a *Docker Template Engine (DTE)* through the *HTTP* protocol, along with data validation between *HTTP* requests and responses. Additionally, I implemented data model handling between the server and database, with support for multiple database systems.

4.4 Server-side Implementation

In this section, it is mainly composed of three portions: installation setup, operational flow, system operation.

4.4.1 Installation Setup

On the server side, the *Laravel* framework must be installed using *Laradock* [45] as the complete *PHP* [46] development environment for *Docker*, which provides support for various pre-configured services commonly used in *PHP* development. As a software package collection, *Laravel* supports the built-in composer.json [47] file for managing dependencies.

4.4.2 Operational Flow

Generally, *Laravel* is divided into two parts, the Controller portion to manage the control logic between the client and server, and the Data model portion to transfer data between the server and the database. To handle the Controller portion, I have created five *Application Programming Interface (API)* routes utilizing *Rest API* HTTP methods. These include the 'create' API route to generate a *Docker* file using the POST method, 'getAll' API route to retrieve the *Docker* file from the database using the GET method, 'getDetailById' API route to retrieve the *Docker* file

by id using the GET method, 'update' API route to update the *Docker* file by id using the PUT method, and 'delete' API route to delete the *Docker* file by id using the DELETE method. To fix the *CORS* issue between the client and server sides, I have implemented *CORS* middleware in all *API* routes. This issue arises due to permission requirements when the client connects to the server with different domains for URL through *HTTP* requests.

I set up the database along with three tables for applications, programming, and application instruction mapping to store the *Docker* file for the project. To establish a connection between the *Laravel* data model and the *MySQL* database, I utilized the built-in *MySQL* database driver configuration provided by *Laravel*. Once the connection is established successfully, data can be transferred from the data model to the database for storage.

4.4.3 Docker Template Engine (DTE)

For users without prior knowledge or experience, it provides a *Docker Template Engine (DTE)* that simplifies the process of creating a *Dockerfile*. After receiving the input data from the create or update forms, the *Docker Template Engine (DTE)* (Figure 4.5) prepares plain *Dockerfiles* with customizable parameters for various programming languages and networking, and other environments such as *C++*, *PHP*, *Python*, *Java*, *JavaScript*, *OpenPose*, *OpenFoam*, *Palabos* and *NS-3 simulator*.

The engine dynamically inserts the required parameters into the *Docker* file based on the input data. Once the template customization is complete, the *Docker Template Engine (DTE)* generates the final *Docker* file and stores it in both the file system under specific project names and the database.

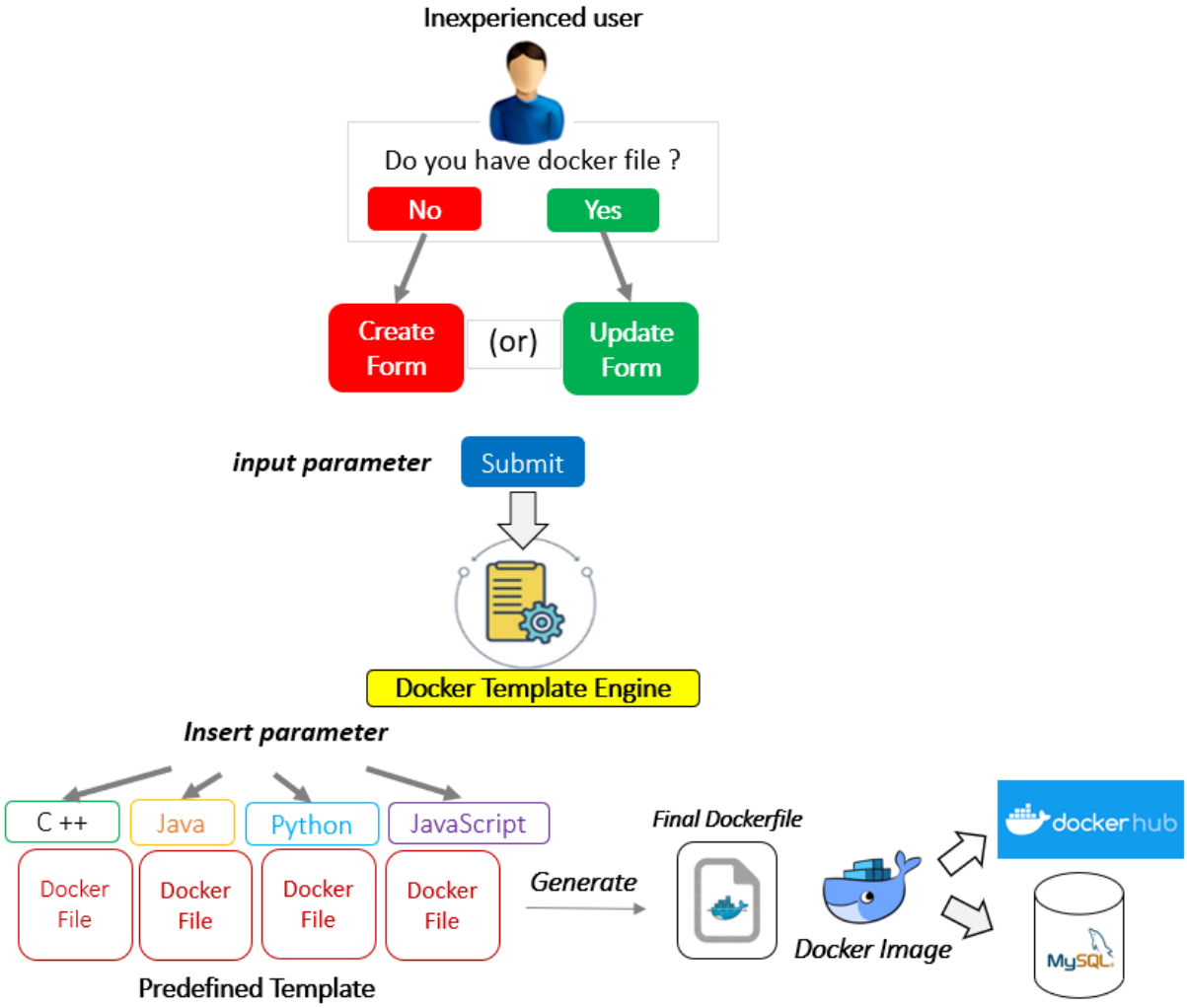

Figure 4.5: Docker Template Engine (DTE) for without prior knowledge or inexperienced users.

4.5 System Operation

The operation of the system involves three main components: client and server interaction, the *Docker Template Engine (DTE)*, and server and database interaction. Each part is detailed below.

4.5.1 Client and Server Interaction

Creating and editing a *Dockerfile* requires time and effort from users. Initially, users fill in or update input data on the client side. The client-side application first converts the input from *JavaScript* format to *JSON* format, as the server accepts data only in *JSON* format. This conversion ensures proper communication between the client and server, as *JSON* is the standard data exchange format.

When users submit the data by clicking the submit button, the client sends the *JSON* data payload to the server using the POST method through the create route of the REST API. For viewing purposes, the client requests the listing API route via the GET method to retrieve and display multiple *Dockerfiles* in a table format. The table includes rows, columns, pagination, and actions like Detail and Delete.

For editing a specific *Dockerfile*, the client redirects to the edit page by calling the getDetailById route through the GET method and passing the *Dockerfile* ID. Once changes are made, users submit the updated data via the update route using the PUT method. If users decide to delete a *Dockerfile*, the client calls the delete route via the DELETE method to remove the file from the database.

To generate a *Docker* image, users click the Ready button on the *Dockerfile* listing page. This action triggers the server to execute shell scripting commands that create the *Docker* image and push it to *Docker Hub*.

4.5.2 Server and Database Interaction

The server validates data received from the client and interacts with the database to manage storage and retrieval of *Docker* file records. Upon receiving a request from the client, the server validates the input data and stores it in the database. For listing purposes, the server retrieves all *Docker* files using the getAll route through the GET method and sends the data back to the client for display in a table format.

When users edit a *Docker* file, the server processes the request through the getDetailById route to fetch the corresponding record from the database. Updated data is sent via the update route and saved in the database using the PUT method. Similarly, the delete route processes requests to remove a specific *Dockerfile* from the database based on its ID. The server also executes *Shell Scripting* to handle *Docker* file conversion to *Docker* images. These images are stored in the file system and pushed to *Docker Hub* for use in various environments.

4.6 Database Implementation

In this section, it is mainly composed of two portions: *MySQL Database* [48] connection and three tables application.

4.6.1 MySQL Database Connection

To manage the database in the tool, I adopted to use *MySQL*. I started by creating a *MySQL* user account and establishing a connection between the server and the database using the built-in *MySQL* database driver configuration in *Laravel*.

4.6.2 Three Tables Creation

Next, I set up three tables in one database: the *application* table as shown in Figure 4.6 for storing application information, the *programming* table as shown in Figure 4.7 for storing programming language information, and the *application instruction mapping* table as shown in Figure 4.8 for storing Docker instructions related to the application table ID.

```
1 •  SELECT * FROM funabiki_docker_manage.application;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

#	id	name	description	os_type	programming	image_status	status	created_by	created_date	modified_by	modified_date
1	1	NPLAS	nplas description	1	1	1	1		2022-09-09 ...		2022-09-09 0...
2	2	EPLAS	eplas description	2	1	1	0		2022-09-09 ...		2022-09-09 0...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 4.6: Application table structure.

```
1 •  SELECT * FROM funabiki_docker_manage.programmings;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

#	id	name	type	status	created_by	created_date	modified_by	modified_date
1	1	javascript	1	0		2022-09-0...		2022-09-09...
2	2	java	1	0		2022-09-0...		2022-09-09...
3	3	php	1	0		2022-09-0...		2022-09-09...
4	4	python	1	0		2022-09-0...		2022-09-09...
5	5	rubyonrails	1	0		2022-09-0...		2022-09-09...
6	6	nodejs	2	0		2022-09-0...		2022-09-09...
7	7	laravel	2	0		2022-09-0...		2022-09-09...
8	8	django	2	0		2022-09-0...		2022-09-09...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 4.7: Programmings table structure.

Limit to 1000 rows

```
1 •  SELECT * FROM funabiki_docker_manage.application_instruction_mapping;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

#	id	application_id	instruction_key_id	instruction_value	status	created_by	created_date	modified_by	modified_date
1	1	1	1	node:12.18.1	0		2022-09-0...		2022-09-09...
2	2	1	12	NODE_ENV=prod...	0		2022-09-0...		2022-09-09...
3	3	1	13	/app	0		2022-09-0...		2022-09-09...
4	4	1	14	['package.json', 'pa...	0		2022-09-0...		2022-09-09...
5	5	1	15	npm install --produ...	0		2022-09-0...		2022-09-09...
6	6	1	14	. .	0		2022-09-0...		2022-09-09...
7	7	1	16	['node', 'server.js']	0		2022-09-0...		2022-09-09...
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 4.8: Application Instruction Mapping table structure.

4.7 Evaluation

For evaluation, I measured the CPU time required to run *Docker* images/tasks in various environments on a UPC worker with an *Intel® Core™ i9-10900K CPU @ 3.70 Hz with 20 cores and 64GB RAM*. To assess the validity and effectiveness of the *Docker Image Generation Assisting (DIG-assist)* tool, I conducted a series of evaluations.

4.7.1 Evaluation Setup

First, I selected 30 popular projects and downloaded the corresponding *Docker* images from online *GitHub* or *Docker Hub*. Next, I created *Docker* files for these images by reversing them with *alpine/dfimage Docker* image that can regenerate *Docker* file from existing *Docker* image . The parameters for the *Docker* instructions in the files were collected from the *JSON* files stored in *Docker Hub* for each selected project. These parameters were used as inputs on the *Docker file* generation page of the tool. The resulting *Docker files* were saved in both the file system and the server's database and could be viewed on the *Docker file* listing page.

Finally, I generated *Docker images* by clicking the corresponding button on the page, which ran the *Docker* build shell scripting commands. These images were then saved as tar files using the *Docker* save command and could be used as jobs in the UPC system.

To compare the efficiency between manual and generation assisting tools of *Docker* files and images, I conducted a study on a load of generating a *Docker* file in the conventional approach versus the load of generating the same file speedily using the proposed tool. In the manual generation approach, the user is required to locate and input the corresponding *Docker* instructions into a text file (Listing 4.2). In contrast, the DIG-assist tool approach only requires the user to input or select the necessary parameters on the input page of the DIG-assist tool, followed by a simple button-clicking process to generate the *Docker* file and image. The results of this study showed that the DIG-assist tool process significantly reduced the time and effort required for *Docker* file and image generation, indicating that the DIG-assist tool can greatly enhance the efficiency of the *Docker* image generation process.

Listing 4.2: Docker file example.

```
1
2  FROM openjdk:16−slim−buster
3
4  MAINTAINER lynnhtetaung@s.okayama−u.ac.jp
5
6  RUN apt−get update; apt−get install −y curl \
7      && curl −sL https://deb.nodesource.com/setup_16.x | bash − \
8      && apt−get install −y nodejs \
9      && curl −L https://www.npmjs.com/install.sh | sh
10
11 WORKDIR /usr/src/app
12
13 COPY . /usr/src/app
14
15 RUN npm install
16
17 EXPOSE 4000
18
19 CMD [‘npm’,‘start’]
```

4.7.2 Result and CPU Time for 30 Docker images

By conducting this setup procedure, the proposed DIG-assist tool successfully generated 30 *Docker images* for the projects corresponding to the *Docker* files on *Docker Hub*. Table 4.1, which shows that the measured CPU time and size required for all projects was less than 20 seconds.

Table 4.1: CPU time and size results of various Docker projects.

No.	Project Name	CPU Time (hh:mm:ss)	Size of Docker Image
1	CFD-OpenFOAM	00:00:07	1.2 GB
2	CNN	00:00:04	450 MB
3	Palabos	00:00:04	77.8 MB
4	DMTCP	00:00:05	131 MB
5	Openpose-GPU	00:00:14	3.38 GB
6	NS-3 Simulator	00:00:16	3.66 GB
7	NPLAS	00:00:06	578 MB
8	Flask	00:00:03	76 MB
9	Django	00:00:05	436 MB
10	JavaJDK	00:00:03	464 MB
11	Node.js	00:00:11	1.25 GB
12	OpenPose	00:00:18	4.09 GB
13	MongoDB	00:00:06	695 MB
14	GCC	00:00:11	1.92 GB
15	RubyOnRails	00:00:03	174 MB
16	Golang	00:00:07	302 MB
17	PostgreSQL	00:00:04	377 MB
18	ReactNative	00:00:15	2.6 GB
19	Flutter	00:00:15	2.2 GB
20	Nginx	00:00:04	142 MB
21	Laravel	00:00:07	726 MB
22	Vue.js	00:00:08	535 MB
23	Ruby	00:00:12	174 MB
24	Apache	00:00:05	143 MB
25	AngularJS	00:00:04	133 MB
26	ASP.NET	00:00:03	122 MB
27	CakePHP	00:00:05	145 MB
28	Svelte	00:00:07	709 MB
29	SpringBoot	00:00:04	146 MB
30	Tornado	00:00:03	112 MB

4.7.3 Security Issues and Challenges

The proposed system incorporates a middleware layer using the PHP *Laravel* framework to validate all incoming data from the web interface, which plays a crucial role in preventing security vulnerabilities. The middleware acts as a gatekeeper, intercepting requests before they reach the core application logic. This ensures that only well-formed and validated data is processed by the system.

To validate the data, we use *Laravel*'s built-in validator service, which provides a robust mechanism to ensure that all required fields are correctly filled, and that no missing or malformed data enters the system. The validator also checks for consistency in data types and format, which helps prevent malicious users from submitting harmful payloads. This validation layer is particularly critical in blocking fake inputs and malicious scripts that could compromise the integrity of the system.

Additionally, Laravel's validator service includes support for custom validation rules, which can be tailored to specific security requirements. This helps mitigate risks such as SQL injection, cross-site scripting (XSS), and other common web application vulnerabilities.

Together, these two layers of security-input validation and custom rule enforcement-form a strong defense against unauthorized data. By ensuring that only clean, valid, and secure data is accepted into the system, these layers help prevent potential attacks and ensure that the data used to generate *Docker* images is safe and accurate. This significantly contributes to maintaining the overall integrity and trustworthiness of the system.

4.8 Summary

In this chapter, I presented an implementation of the *Docker image generation assisting (DIG-assist) tool*. I discussed the overview, software architecture, implementation details and evaluation of DIG-assist tool. In the next chapter, I will review the overview of the *Flutter Programming Learning Assistant System (FPLAS)* .

Chapter 5

Overview of Flutter Programming Learning Assistant System (FPLAS)

This chapter reviews *Flutter Programming Learning Assistant System (FPLAS)* that consists of the answer platform, *Flutter* development environment, and the image-based *UI testing* tool. *Docker* was utilized to efficiently distribute each application to a large number of students.

5.1 Overview

In this section, I introduce the browser-based answer platform that supports *Java* and *Python* programming language exercise assignments for the students by using *Docker* and *Node.js*, the *Flutter* development environment based on *Visual Studio Code* with a remote development container and docker-compose.yml settings, and the image-based *UI testing* tool that allows a teacher to validate the accuracy of a student's answer image compared to correct answer images using the *Python Flask* framework, as well as the *SIFT* and *ORB* algorithms of the *OpenCV* library.

5.2 Answer Platform

The *answer platform* is a web application that was developed using *Node.js* and the *Express.js* [49] framework, designed to help novice students complete programming exercises while tracking their learning activities through a personal platform. It follows the *Model-View-Controller (MVC)* architecture. Currently, it supports *Java* and *Python* programming. For the model (M), the *Java/JUnit* framework is used to execute tests of answer source code in code-writing problem (CWP) for Java, while the *unittest* framework is used for the same purpose in *Python*. For the view (V), *EJS* is adopted. For the controller (C), *JavaScript* is used.

5.3 Flutter Development Environment

The *Flutter* environment is a *Docker*-based *Flutter* development environment that was designed for novice students to create mobile applications without requiring a complex setup. It offers a pre-built *Docker* container and system startup files on *GitHub*, along with sample *Flutter* projects and comprehensive setup instructions. To begin, students need to install *Docker* and *VSCode* [50], import necessary extensions, and obtain the *Docker* container. They can download or clone the

GitHub project, open it in *VSCode*, and activate the *Flutter* environment. Once connected to the web application server of this *Flutter* environment, students can solve the provided exercises by transferring them to the workspace, modifying the source code by following the guidelines, and previewing their works. Upon completions, students submit their modified code files and UI image outputs via *Moodle* or a web service to a teacher, who manually assesses all the submissions by executing the source code and checking the output UI images on their PCs.

5.4 Imaged-Based UI Testing Tool

The *image-based UI testing tool* is presented to automate the evaluation of UIs generated by students' answer images using the *Python Flask* framework. This method includes executing the answer code, cropping the unnecessary black borders, capturing the output UI image from the code manually, and comparing it with the corresponding correct UI image generated by running the model code for each exercise assignment. To perform this comparison, the *ORB (Oriented FAST and Rotated BRIEF)* [51] and *SIFT (Scale-Invariant Feature Transform)* [52] algorithms in *OpenCV* library are used. These algorithms detect and describe unique elements in the images, enabling a robust comparison that identifies similarities and differences with high precision. By automating the UI testing process, this method significantly reduces the grading workload of a teacher, improves assessment accuracy, and provides timely feedback to students. This setup makes it easier for a teacher to evaluate a large number of student results automatically while offering individualized feedback.

5.5 Summary

In this chapter, I presented an overview of the *Flutter Programming Assistant System (FPLAS)*. I discussed the details of the separate implemented web application systems: the answer platform, *Flutter* development environment, and an image-based *UI testing* tool. In the next chapter, I will present a web-based answer platform for the *Flutter Programming Learning Assistant System* using *Docker* Compose.

Chapter 6

Implementation of FPLAS Answer Platform

This chapter presents the implementation of *web-based answer platform* for the *Flutter Programming Learning Assistant System (FPLAS)* using *Docker Compose*.

6.1 Overview

In this section, I present the implementation of the web-based answer platform for the *Flutter Programming Learning Assistant System (FPLAS)* by integrating the four *Docker images* for the *answer platform*, *Flutter* environment, *Nginx* web application server, and image-based *UI testing* tool using *Docker* compose.

6.2 Software Architecture

Figure 6.1 presents the software architecture of proposal. It integrates the *answer platform*, the *Flutter* environment, the *Nginx* web server, and the image-based *UI testing tool*.

On the *client side*, *Flutter/Dart* programming exercise assignments are created by adding new directories and files to the existing *answer platform*.

On the *server side*, the *Flutter* environment, where *Visual Studio Code* is excluded, and the image-based *UI testing* tool are integrated and enhanced with new functionalities for monitoring file changes, capturing output UI images automatically, compiling a *Flutter* source code into *JavaScript*, and transferring the generated *JavaScript* code to the *Nginx* web server's configuration directory. This setup displays three images such as the student's answer image, the correct answer image, and the image highlighting the differences between them with red boxes.

The *Nginx* web server is newly utilized to render the compiled *JavaScript code* from the *Flutter* environment, allowing the output UI image to be displayed. This step is necessary because built-in *Flutter* commands like $--chrome$ and $--web\text{-}server$ cannot be executed directly in the browser using a *Docker* setup. Overall, the four *Docker* images are created and run as multiple containers using a single *Docker* compose configuration.

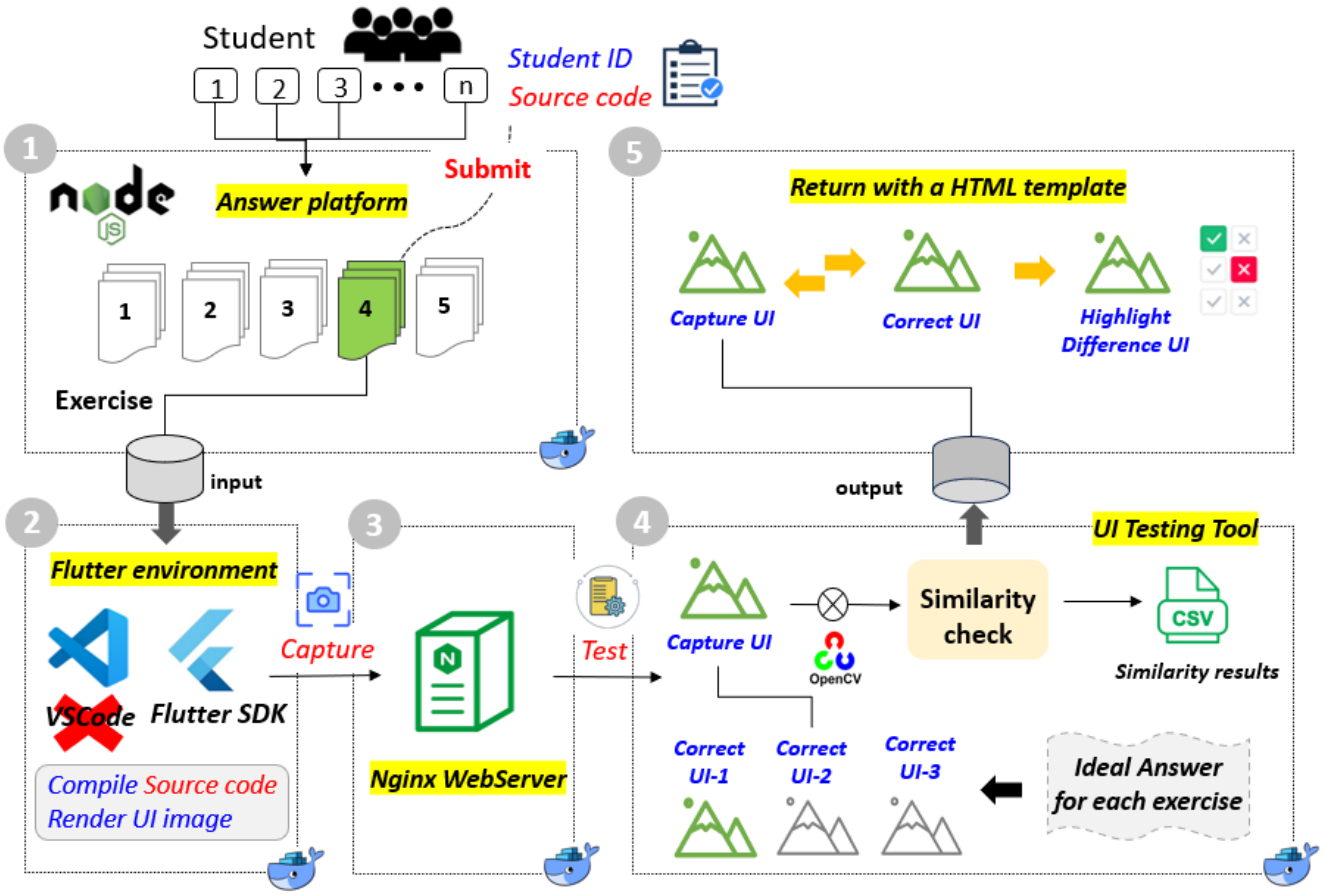

Figure 6.1: Software architecture of a web-based answer platform for the FPLAS.

6.3 Client-side Implementation

In this section, I present operational flow and answer platform of *Flutter Programming Learning Assistant System (FPLAS)*.

6.3.1 Operational Flow

On the client side of the *FPLAS*, the *answer platform* was enhanced with four extensions, as shown in Figure 6.2. First, the theme color and user interface (UI) designs were updated to a blue theme using *CSS* styling and the *Bootstrap* framework, improving user engagement compared to other programming exercises for *Java* and *Python*. Second, an input text box was added to the homepage for a student to enter their student ID, which is sent as one of the input parameters to the server and stored in the browser's local storage. Third, a new *fplas.js* file was created under the route directory specifically for *Flutter* to prevent structural duplication with other programming languages like *Java* and *Python*. This file manages data transfer to the view by following the *Model-View-Controller (MVC)* architecture.

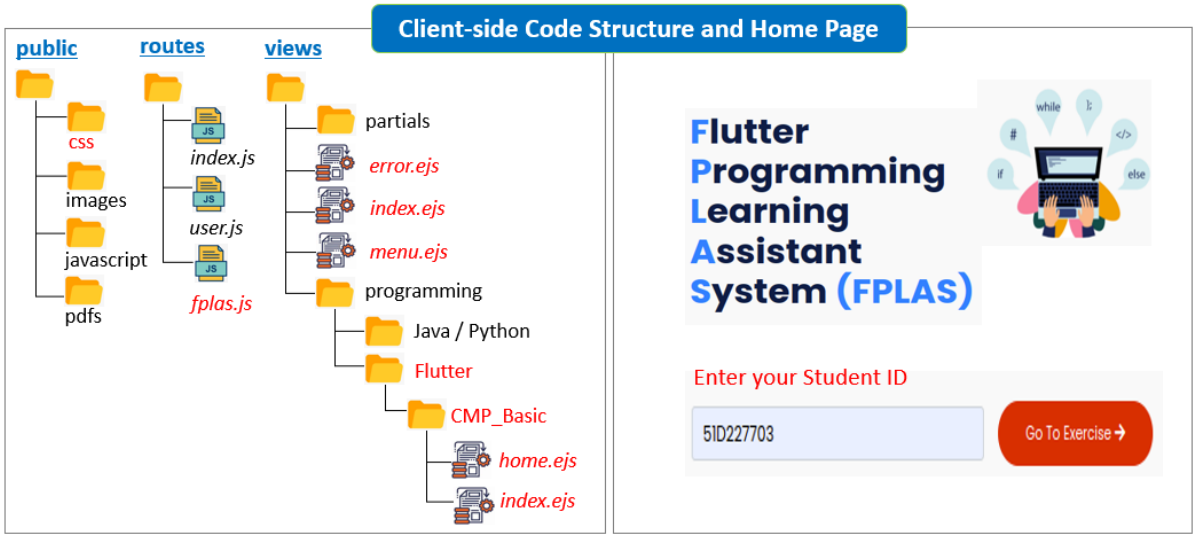

Figure 6.2: Client-side code structure (**left**) and home page (**right**) for the *FPLAS*.

6.3.2 Answer Platform

In this section, new web pages for one set of exercise problems, named *Flutter CMP (Code Modification Problem) Basic*, were created in the 'views' directory to handle the data manipulation through a *JavaScript* controller. The *Flutter CMP UI* consists of two layouts, as shown in Figure 6.3. The left side of the UI displays the instructions for modifying the sample source code. The right side hosts a source code editor that is adopted by the *CodeMirror* library [53]. Once the code is modified following the user guide, a student clicks the *'Run'* button to submit the student ID and the modified code to the server via the *HTTP POST* method using the *REST API*. The server validates and processes the input step by step, generating an *HTML* template response. If there are no errors, a successful message in a *HTML* template is returned. Otherwise, a *HTML* template with an error message is sent back.

Then, the result is displayed in a dialog box, showing the three images: the student's answer image, the correct answer image, and the difference-highlighting image. This allows a student to compare the answer code's output with the correct solution, as shown in Figure 6.4. To achieve a full score of 100%, a student must continue modifying the source code until the answer image fully matches the correct answer image.

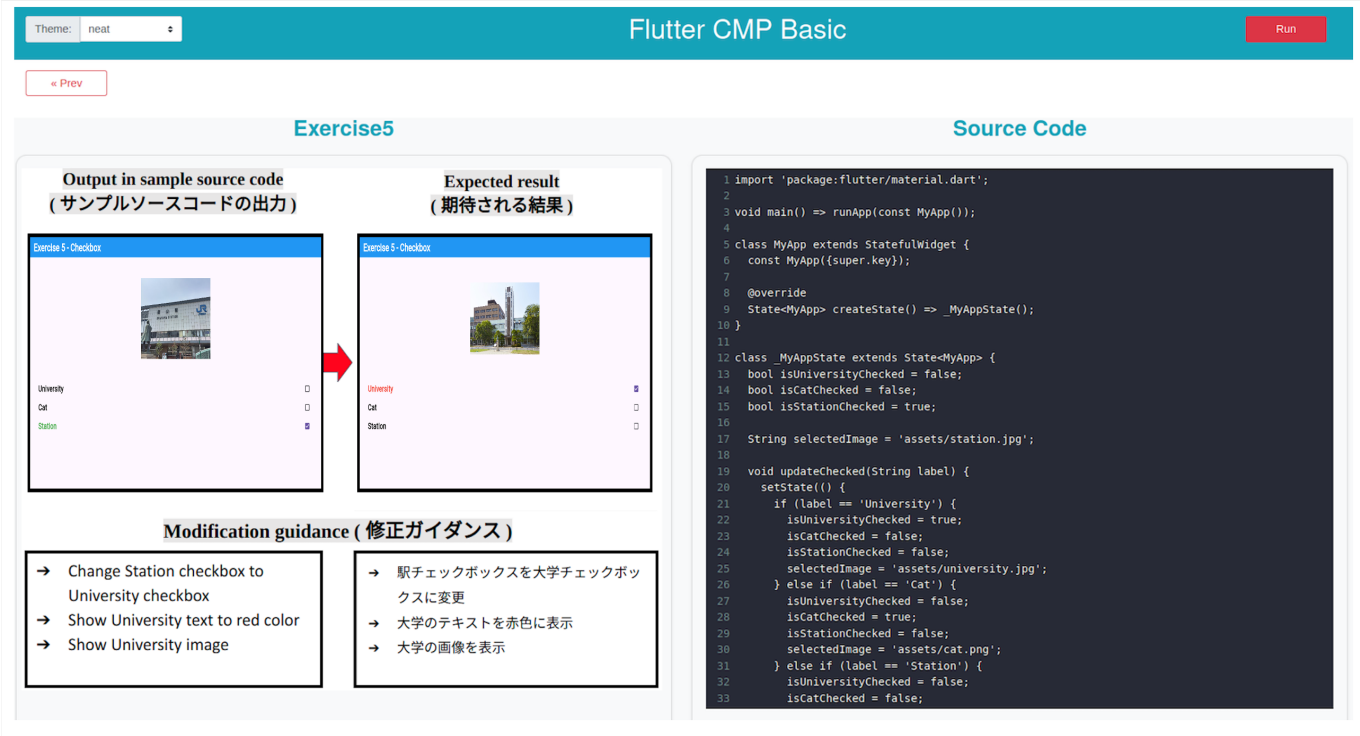

Figure 6.3: Exercise 5 assignment for the proposed answer platform.

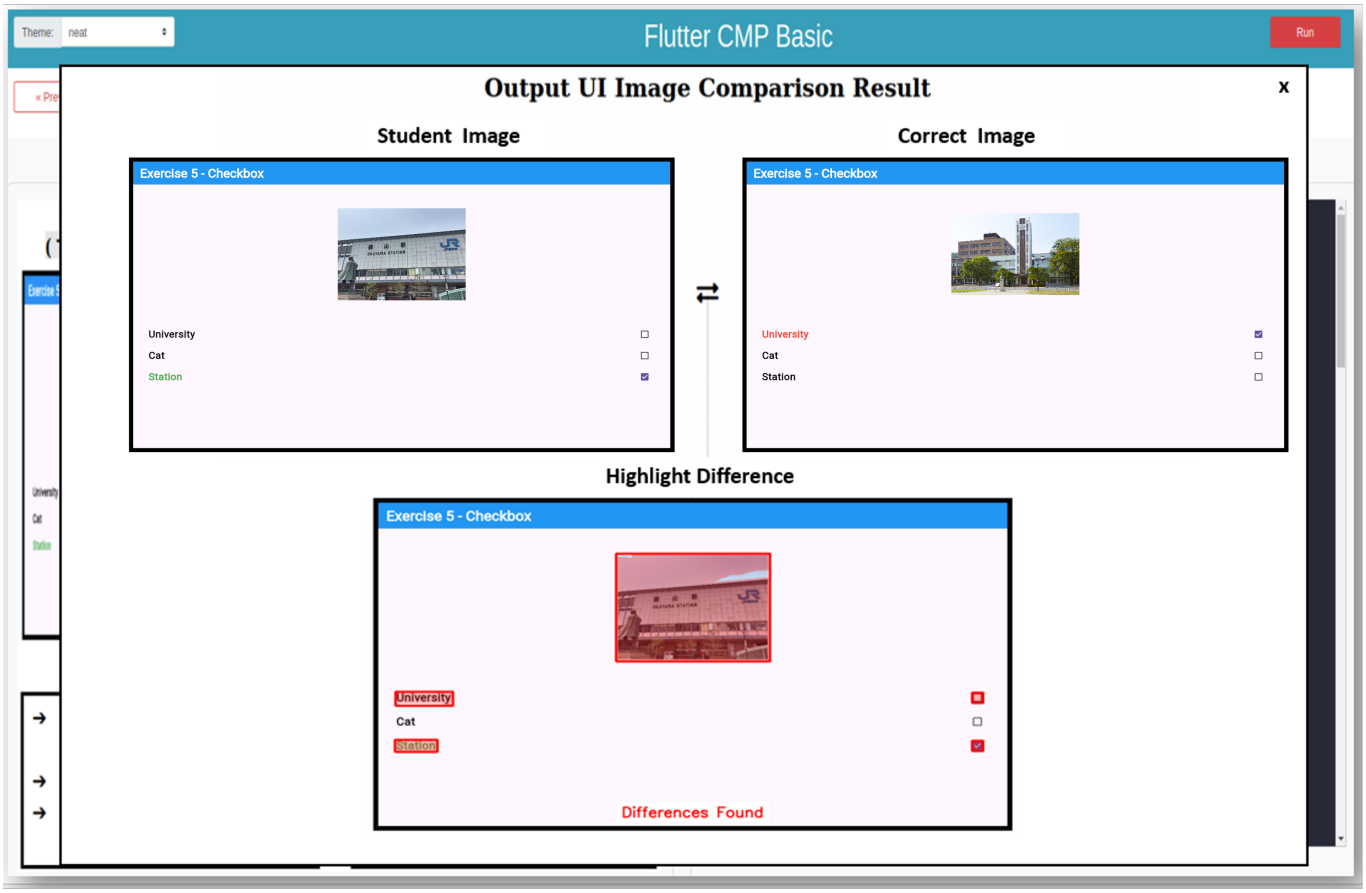

Figure 6.4: Highlighting differences with red boxes for incorrect answers in Exercise 5.

6.3.3 Five Flutter Exercises

In this subsection, I introduce five simple *Flutter* projects as exercises for novice students to start learning *Flutter/Dart* programming. These exercises cover basic *Flutter* widgets and components to help students learn how to build interactive user interfaces. Each exercise includes the sample source code, the corresponding output UI, and the expected result. The task descriptions for the exercises are as follows:

- **Exercise 1 – Container:** Modify the card's text, color, size, and text color as shown in Figure 6.5.

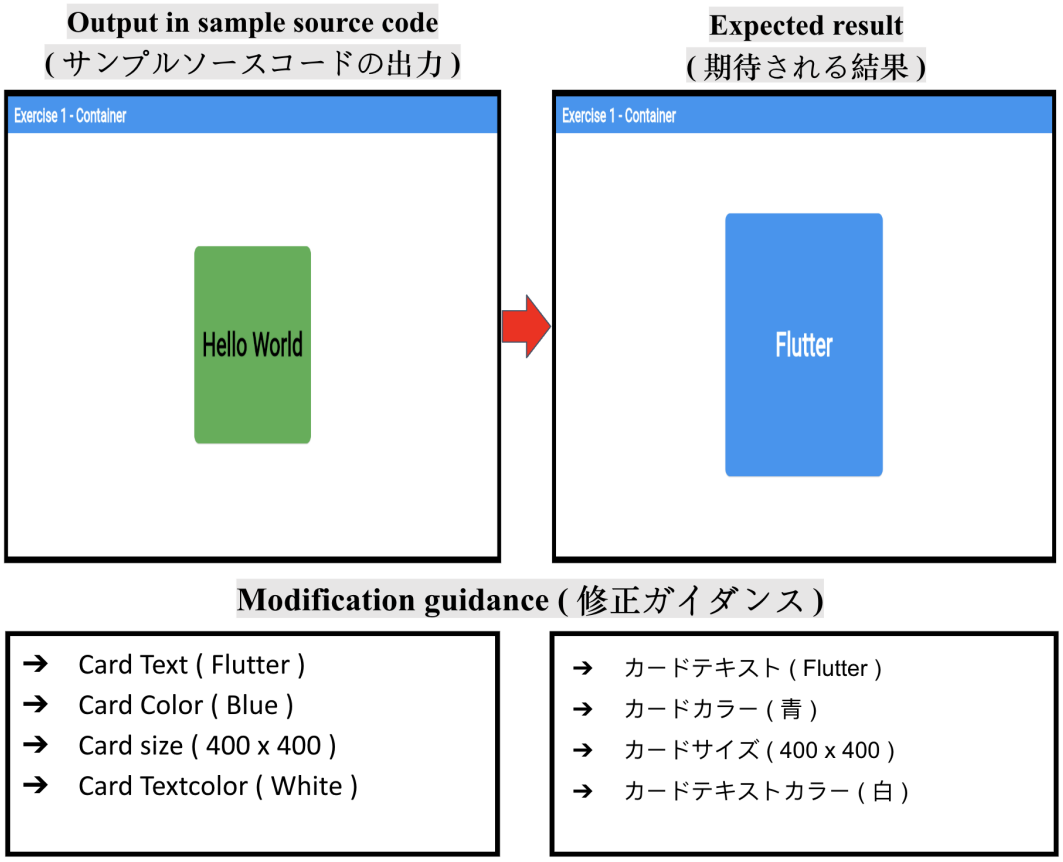

Figure 6.5: Exercise 1 - Container Assignment.

- **Exercise 2 – ListView:** Rearrange the given items from ascending order to descending order as shown in Figure 6.6.

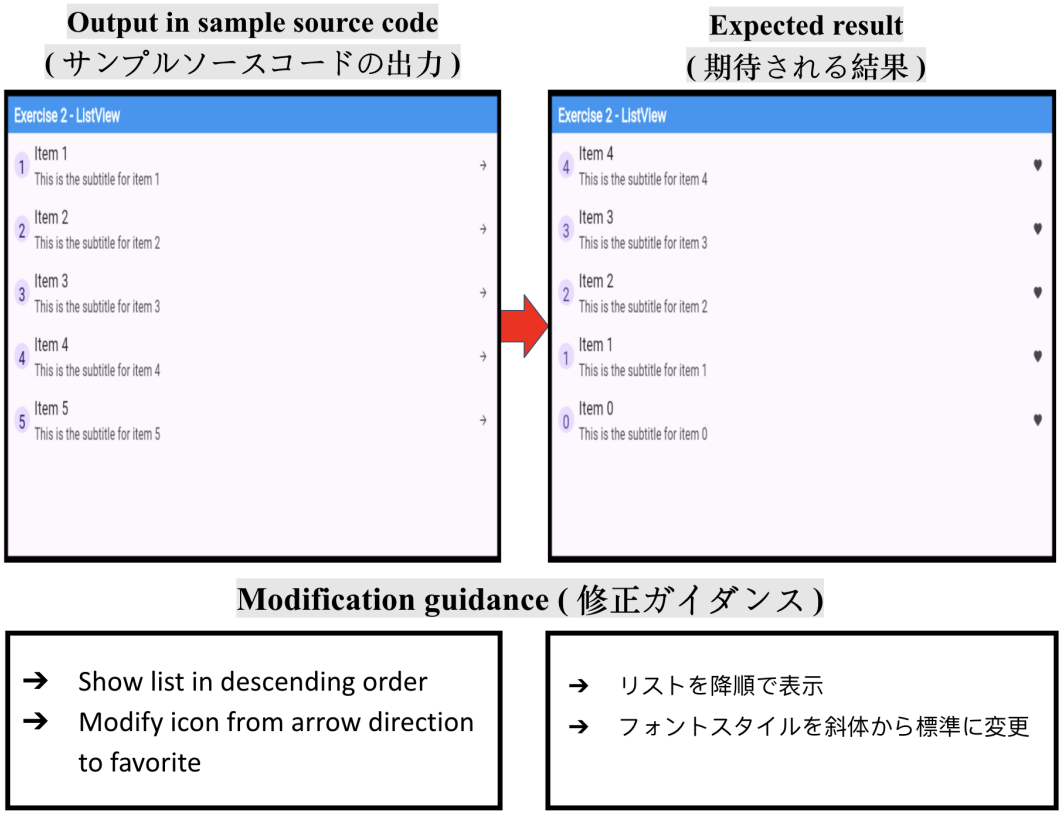

Figure 6.6: Exercise 2 - ListView Assignment.

- **Exercise 3 – Horizontal ListView:** Change the list from horizontal to vertical and ensure that each colored container has dimensions of $150 \times 150 \times 150$ as shown in Figure 6.7.

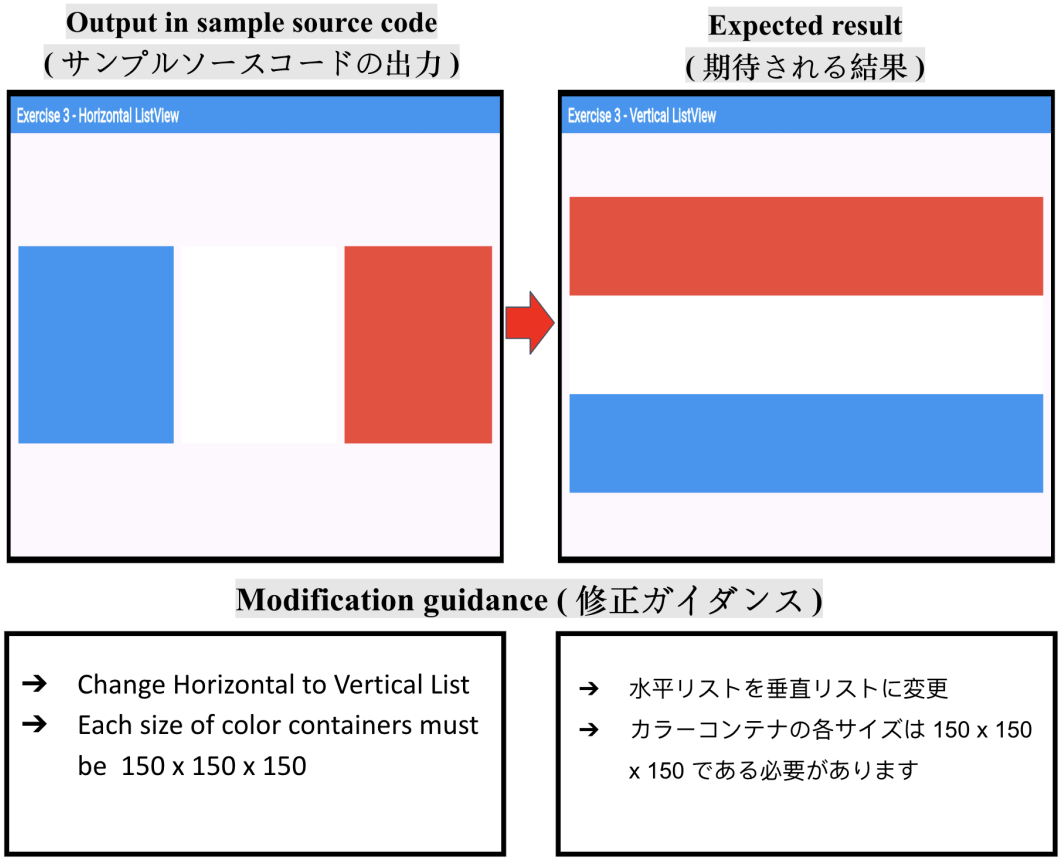

Figure 6.7: Exercise 3 - Horizontal ListView Assignment.

- **Exercise 4 – Bottom Navigation Bar:** Navigate from the business page to the home page, which should contain an icon, text with a font size of 40, and a button with a font size of 30 as shown in Figure 6.8.

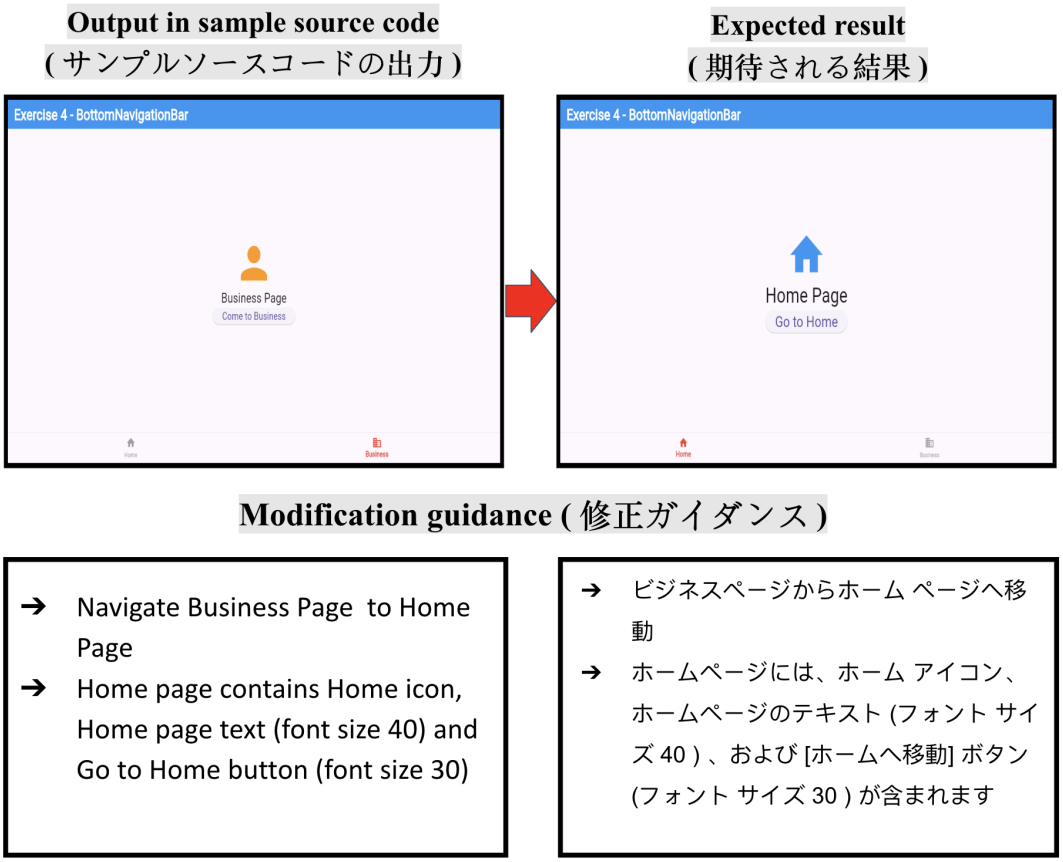

Figure 6.8: Exercise 4 - Bottom Navigation Bar Assignment.

- **Exercise 5 – Checkbox:** Select the checkbox from the station to the university, change the text color from red to green, and replace the station image with a university image as shown in Figure 6.9.

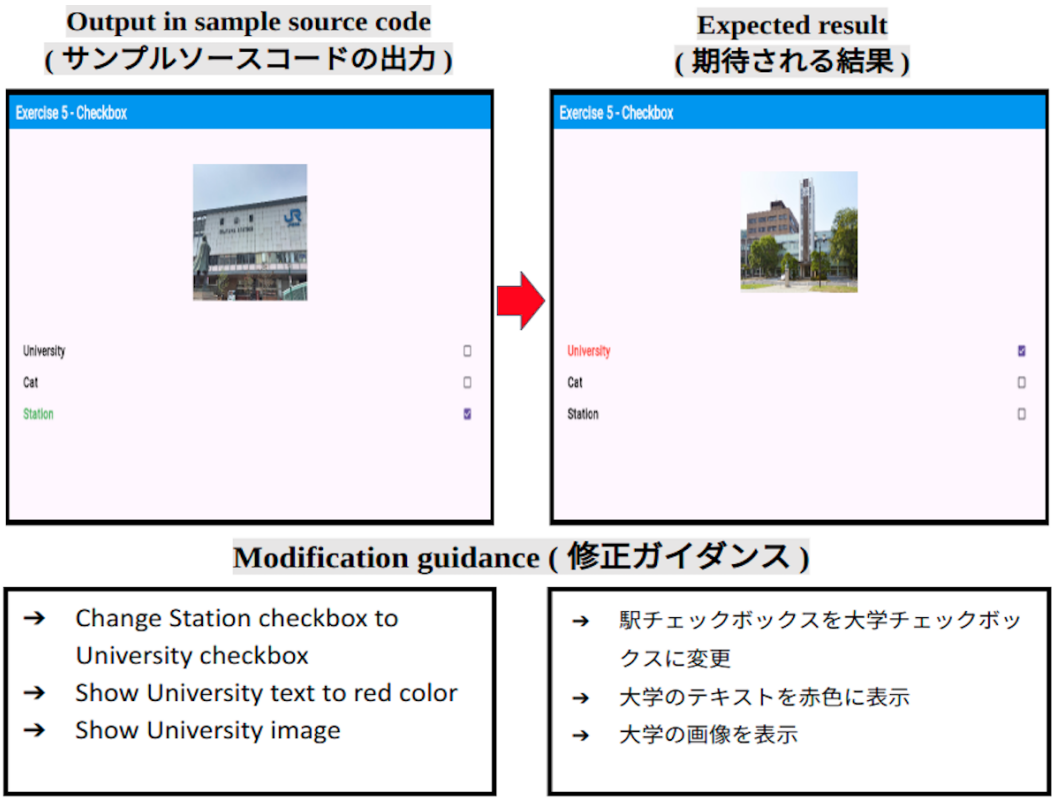

Figure 6.9: Exercise 5 - Checkbox Assignment.

6.4 Server-side Implementation

In this section, I present three extensions for the *Flutter* environment, the *Nginx web server*, and the image-based *UI testing* tool.

6.4.1 Flutter Environment

For the *Flutter environment*, the *Visual Studio Code* integration and the *iOS/Android* settings were removed to create a lightweight setup designed exclusively for web development. The necessary files and directories of the code structure shown in Figure 6.10, along with their roles, are as follows:

- **.dart-tool** and **.idea:** These handle extensions, *XML* files, and the cache required for the *Flutter engine* to run.
- **assets:** This stores the images and fonts used in a *Flutter* project.
- **test:** This contains the test files for unit testing.
- **web:** This stores icons, favicons, *index.html*, and *manifest.json* required for *Flutter* in web development.
- **lib:** This stores the *Flutter/Dart* source code sent from the *answer platform* as a *main.dart* file.

- **build:** This stores the obtained *JavaScript* files after compiling the *main.dart* file, which contains the *Flutter/Dart* source code from the 'lib' directory.
- **pubspec.lock:** This manages the lock files for the library packages defined in *pubspec.yaml*.
- **pubspec.yaml:** This handles the package manager for the libraries and the configuration settings of the *Flutter* project.

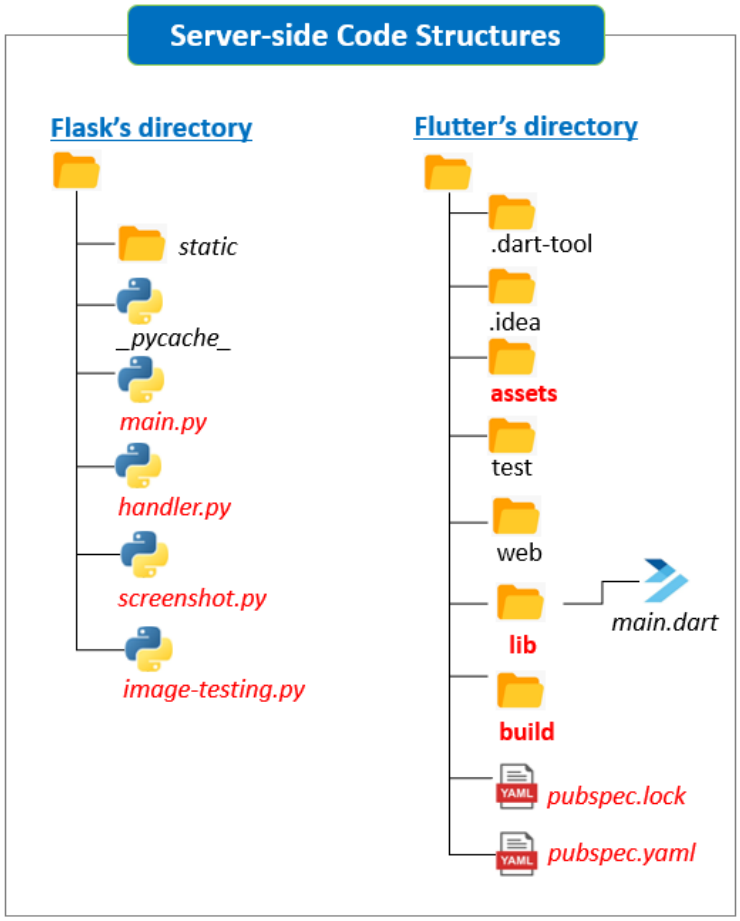

Figure 6.10: Server-side code structures for the FPLAS.

6.4.2 Nginx Web Server

For the *Nginx* web server, the official Docker image of *Nginx* is adopted to render the compiled *JavaScript* code from the *Flutter/Dart* code and display the output UI images in the browser. This step is necessary because the default *Flutter* running commands, such as −−chrome and −−web-server modes, cannot directly execute the *Flutter/Dart* code in the browser when using a *Docker* setup.

6.4.3 Image-based UI Testing Tool Modification

For the image-based *UI testing* tool, the new functionalities were implemented for file-watching service monitoring, automatic output UI image capturing, compiling *Flutter/Dart* source code to

JavaScript, transferring the generated *JavaScript* code to the *Nginx* web server's configuration directory, and displaying the three images: the student's answer image, the correct answer image, and the image highlighting the differences between them with red boxes. The necessary files and directories of the code structure shown in Figure 6.10, along with their roles, are as follows:

- **static:** This stores the necessary images, such as the correct answer image, the error image, and the incoming screenshot image associated with the student ID.
- **−−pycache−−:** This contains the cache files as backups for the *Python* files.
- **main.py:** This acts as the *entry point* of the *Python* application and handles the success/error status of the *handers.py*, *screenshot.py*, and *image-testing.py* programs by linking them together.
- **handlers.py:** This provides a file-watching service to monitor the incoming *Flutter/Dart* source code stored in the 'lib' directory of the *Flutter* project.
- **screenshot.py:** This manages the execution of the *Flutter/Dart* code from the 'lib' directory, compiles the *Flutter/Dart* code to generate the corresponding *JavaScript* code, copies this code to the *Nginx* web server's configuration directory, renders the *JavaScript* code within *Nginx* to display the UI output in the browser, and captures a screenshot image of the UI output.
- **image-testing.py:** This updates the *UI testing* tool to display the image highlighting the differences between the answer image and the correct answer with red boxes, using the *SIFT/ORB* algorithms in the *OpenCV* library.

6.5 Creation System Setup

After completing the implementation, the answer platform's *Docker* image was generated by preparing the necessary commands in the *Docker file* (Listing 1 in Appendix A) and uploaded to the *DockerHub* repository. Additionally, corresponding *Docker* images were created for the *Flutter* environment, the *Nginx* web server, and the image-based UI testing tool by preparing their *Docker files* (Listings 2–4 in Appendix A), and these were also uploaded to the *DockerHub* repository.

6.5.1 System Setup by Teacher

The preparation steps for the system setup by a teacher are as follows:

- **Step 1:** Install *Python* along with a virtual environment to add the required libraries, the *Flutter SDK* to test the *Flutter* compilation, *Git*, *Docker*, and *Docker Compose*.
- **Step 2:** Prepare the *Docker files* for the *answer platform*, the *Flutter environment*, the *Nginx web server*, and the *image-based UI testing tool* with the required instructions after completing the implementation.
- **Step 3:** Build the four *Docker images* using the *Docker build* command, and tag each image with the name and version number.

- **Step 4:** Upload the *Docker* images to the *Docker Hub* repository using the *Docker push* command.
- **Step 5:** Prepare the *docker-compose.yml* file (see Listing 5 in Appendix 8) to manage the four *Docker images* for running multiple containers with a single configuration.
- **Step 6:** Configure the directory path in the *docker-compose.yml* file to bind the mount between the *Docker container* and the *host computer* using *Docker bind mount* [54].
- **Step 7:** Create a project in the *GitHub* repository and upload all the necessary files along with the instructions to be distributed to the students.

6.5.2 System Installation by Student

The installation steps for a student are as follows:

- **Step 1:** Install *Docker* based on the operating system on the student's PC.
- **Step 2:** Pull the four *Docker images* using the *Docker pull* command from the *Docker Hub* repository.
- **Step 3:** Clone or download the project containing the *docker-compose.yml* file and *README.md (user guide)* from the specified *GitHub* repository.
- **Step 4:** Change the directory path in the *docker-compose.yml* file to the student's PC directory path to locate the *output* folder containing their source code answer files after completing the assignments.
- **Step 5:** Run the *docker-compose up* command to execute the *Docker images* as multiple containers using *Docker Compose*.
- **Step 6:** Navigate to *localhost* on *port 4000* in the browser to start the system.

6.6 Evaluation

In this section, I evaluate the efficiency and validity of the proposed system for the *FPLAS* through an application aimed at novice students.

6.6.1 Evaluation Setup

For the installation, I prepared the user guide in *English* and *Japanese* versions, which explains how to install and use the platform. The guide includes instructions for installing *Docker*, downloading four *Docker images* from the *Docker Hub* repository, connecting the containers with *Docker Compose*, modifying the projects in the exercises, and submitting the answer files via *Google Forms*. Each section of the document is explained using text and images to improve readability. Separate PDF files were created for each operating system such as *Windows*, *Linux*, and *MacOS*.

Then, I asked 10 graduate students in the Department of Information and Communication Systems, Okayama University, Japan, to set up the development environment and solve the exercises independently. Each of them had different skills and expertise in programming. Among them, five

students had no prior experience in *Flutter*, although they possessed foundational knowledge in programming using C, C++, Java, and Python. Consequently, the primary objective of our evaluation was to assess how our proposed answer platform could assist these students in effectively adapting to and learning *Flutter* for mobile application developments.

Following the completion of the assignments, the students submitted their solutions for five exercises through *Google Forms*. Additionally, they completed a questionnaire designed to gather feedback and further comments on the system.

6.6.2 Result for Student Learning Experiences

First, I examined the students' experiences related to the proposed *FPLAS* answer platform using *Docker* and *Git*. Table 6.1 presents the four questions posed to the students and their responses, which were collected via *Google Forms*.

6.6.3 Results and Difficult rates of Five Exercises

Next, I present an analysis of the results and the difficulties encountered by the students during the completion of the exercises, shown in Table 6.2. As shown in the table, all students successfully completed the five exercises with a correctness score of 100% across all exercises, which highlights the effectiveness of the answer platform in guiding students through the learning process. Despite this success, varying levels of difficulty were reported by students, particularly in **Exercises 2, 3, 4,** and **5**, with the most challenging being **Exercise 4**, which received a difficulty rating of **40%**. **Exercises 2, 3,** and **5** received a moderate difficulty rating of **20%**, while **Exercise 1** was considered the easiest, with no reported difficulties **(0%)**.

This variation in difficulty levels provides valuable insights into how different types of *Flutter* exercises impact novice learners. Despite the challenges, the fact that all students achieved full correctness scores indicates that the exercises were appropriately designed to gradually increase in complexity, allowing students to build on their understanding of *Flutter* concepts as they progressed. Overall, the exercises facilitated a balanced learning experience that encouraged steady improvement and mastery of introductory *Flutter* development.

Table 6.1: Questions and answers on experiences related to the proposed system.

No.	**User Experience-Related Questions**	**# of Students**	
		Yes	**No**
Q1	Do you have any experience with *mobile app development*?	5	5
Q2	Are you familiar with *Flutter* programming?	4	6
Q3	Are you familiar with *Git*?	9	1
Q4	Are you familiar with *Docker*?	9	1

Table 6.2: Results and difficulty ratings of the five exercises by the students.

Results by Students	**Correctness**				
	Exercise 1	**Exercise 2**	**Exercise 3**	**Exercise 4**	**Exercise 5**
Score	100%	100%	100%	100%	100%
Difficulty Rating	0%	20%	20%	40%	20%

6.6.4 Result for System Usability Scale (SUS) Questionaries

Finally, I evaluated the usability of the proposed system using the *system usability scale (SUS)* [55]. The *SUS* is a widely adopted and reliable tool for assessing the usability of various systems, providing a quick yet comprehensive measure of users' perceptions. It consists of 10 standardized questions rated on a *5-point Likert* scale, capturing both positive and negative aspects of usability.

I chose the *SUS* due to its ability to offer a holistic view of usability, which includes ease of use, complexity, and users' overall satisfaction. This contrasts with other methods, such as the *Usability Metric* for User Experience (UMUX), which focuses more narrowly on specific aspects of user experience related to usability. SUS's broader scope allows for a more comprehensive understanding of how users interact with and perceive the system. Table 6.3 presents the questions used to evaluate usability and the corresponding responses from the students.

Table 6.3: Questions and responses used to assess the system usability scale.

No.	System Usability Scale Questions	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Q1	I found the system easy to use, install, and set up.	1	0	1	4	4
Q2	I found the system unnecessarily complex.	1	3	0	2	4
Q3	I thought the instructions for completing assignments were clear and easy to follow.	1	0	1	6	2
Q4	I needed technical support to set up the system on my PC.	6	0	0	2	2
Q5	I felt confident using the system without needing to learn much before getting started.	0	2	1	4	3
Q6	I found the system slow in terms of response time.	1	2	4	2	1
Q7	The system's feedback was helpful in guiding me to correct solutions when verifying code.	0	0	2	6	2
Q8	I had to modify the provided source code several times to get the correct answer.	1	3	2	4	0
Q9	I feel that my Flutter skills have improved after solving exercises using this system.	0	1	0	6	3
Q10	I am not satisfied with the overall performance of the system.	8	2	0	0	0

Table 6.4 shows the grades for the comprehensive assessment of the system's usability, with *system usability scale (SUS)* scores ranging from 0 to 100. Among the 10 students, one student gave a score of 15.15%, corresponding to Grade A (*Best Imaginable)*; no students' scores corresponded to Grade B (*Excellent)*; three students gave a score of 33.32%, corresponding to Grade C (*Good)*; five students gave a score of 44.69%, corresponding to Grade D (*OK)*; and one student gave a score of 6.82%, corresponding to Grade E (*Poor)*. No students (0%) gave the *Worst Imaginable (Grade F)*. The results indicate that 66% of the 10 students were satisfied with the proposed system in terms of usability.

Table 6.4: Grades Associated with SUS Scores

SUS Score	Grade	Rating
$85 \leq x \leq 100$	A	Best Imaginable
$80 \leq x < 85$	B	Excellent
$70 \leq x < 80$	C	Good
$50 \leq x < 70$	D	OK
$25 \leq x < 50$	E	Poor
$x < 25$	F	Worst Imaginable

6.6.5 Result for Analysis Between Students' Feedback and SUS Score

I conducted a correlation analysis to explore the relationship between user experience-related questions and *SUS* scores for both experienced and non-experienced students, focusing on *mobile application development*, *Flutter*, and *Docker*. The analysis revealed significant correlations for both groups. **Correlation Analysis 1** and **Correlation Analysis 2** examined the relationships within the *mobile application development* and *Flutter* categories, as shown in Figure 6.11, while **Correlation Analysis 3** focused on *Docker*, as shown in Figure 6.12. These correlations highlight the effectiveness of our proposed system and underscore the importance of providing tailored support to students with varying levels of expertise in development tools.

The results of the correlation analysis are as follows:

- **Correlation Analysis 1:** Among the six students experienced in *mobile application development*, one rated it '*Grade A (Best)*', two rated it '*Grade C (Good)*', and three rated the proposed system as '*Grade D (OK)*'. Students who are not experienced in *mobile application development* gave mixed feedback. One rated it '*Grade E (Poor)*', one rated it '*Grade C (Good)*', and two rated it '*Grade D (OK)*'. Students experienced in *mobile application development* were generally positive, indicating the need for user experience improvements, while the varied responses from students who are not experienced in *mobile application development* suggest that the proposed system requires refinement to offer a consistent experience across skill levels.

- **Correlation Analysis 2:** Among the five students experienced in *Flutter*, one rated it '*Grade A (Best)*', two rated it '*Grade C (Good)*', one rated it '*Grade D (OK)*', and one rated it '*Grade E (Poor)*'. Among the five students who are not experienced in *Flutter*, one rated it '*Grade C (Good)*', and four rated it '*Grade D (OK)*'. The feedback suggests that, although the proposed system is generally acceptable, improvements in usability and user interface are needed, especially for less experienced users, to address concerns and enhance the overall perception.

- **Correlation Analysis 3:** Among the nine students experienced in *Docker*, one rated it '*Grade A (Best)*', one rated it '*Grade E (Poor)*', four rated it '*Grade D (OK)*', and three rated it '*Grade C (Good)*'. The only student who is not experienced in *Docker* rated it '*Grade D (OK)*'. Most students found the proposed system acceptable, regardless of experience. However, the one "*Poor*" and several "*OK*" ratings from users experienced in *Docker*

suggest that improvements are needed to enhance the platform's usability, even for those familiar with these tools.

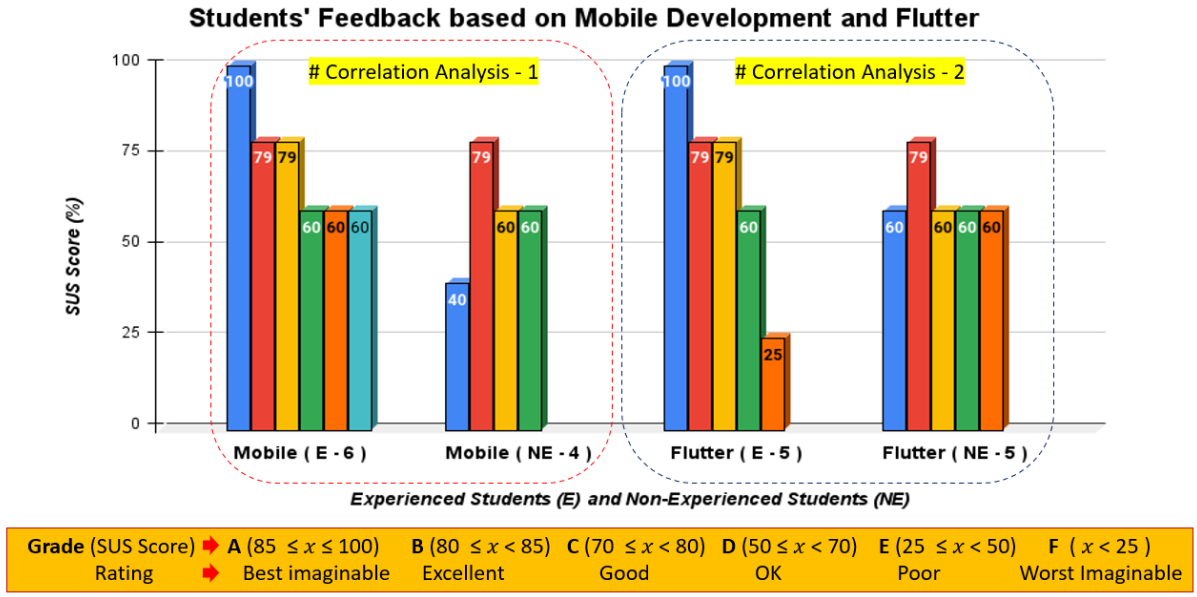

Figure 6.11: Analysis of feedback on *mobile application development* and *Flutter*.

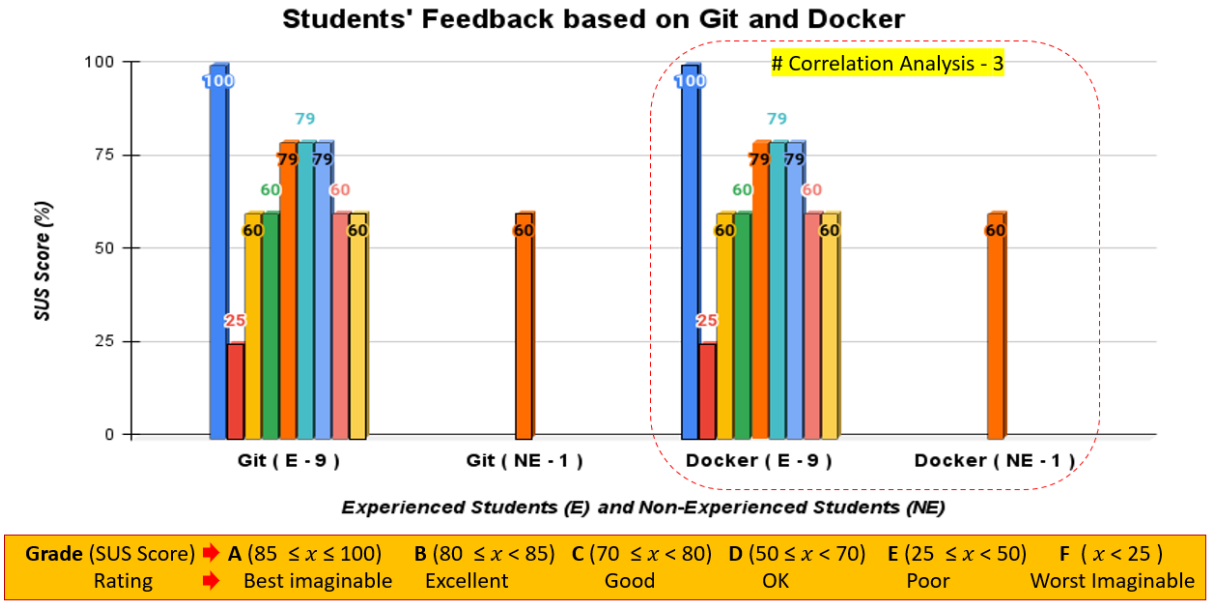

Figure 6.12: Analysis of students' feedback based on *Docker*.

It is noteworthy that one experienced student rated the proposed system as '*Grade E (Poor)*' in all correlation analyses, even though this student was familiar with the tools used in the system. This unexpected feedback suggests that there might be hidden issues or challenges with the platform that even experienced users struggle with. It is important to look into this further to understand what might have caused this negative response and make improvements in the future.

6.6.6 Security Issues and Challenges

This proposal leverages *Docker Compose* to manage four *Docker* images, allowing seamless data sharing and communication between containers through a single, unified network. By utilizing *Docker Compose*, the system ensures that the various containers are properly connected and their interactions are automatically handled. This setup simplifies the management of the Dockerized environment and promotes efficient communication across the different services involved.

However, as with any system that involves the use of containers, security concerns arise, particularly when dealing with sensitive data such as the correct answer images used during evaluations. Students could potentially manipulate or modify the answer images stored within the running containers, compromising the integrity of the evaluation process. To address these security concerns, the system implements several key measures to ensure the security and authenticity of the data being used.

- **Read-Only and Encrypted Answer Images:** To prevent unauthorized modifications, the correct answer images are set to read-only and are encrypted. By setting the images as read-only, the system ensures that no user or process can alter the images during container runtime. The encryption adds an additional layer of security by making it significantly more difficult for unauthorized users to tamper with or decode the answer images.

- **Source Code Submission Requirement:** During evaluations, students are required to submit both their source code and the corresponding answer images. This ensures that teachers can verify whether the submitted source code produces the correct result as depicted in the answer image. By cross-referencing the students' code with the pre-defined correct image, teachers can easily detect any attempts to alter the results or submit incorrect work. This process also helps to ensure that the submitted source code aligns with the expected output, thereby preserving the fairness and accuracy of the evaluation process.

These security measures help mitigate potential risks and ensure that the evaluation process remains secure and reliable, protecting the integrity of both the source code and the evaluation results.

6.7 Summary

In this chapter, I presented an implementation of the web-based answer platform in *Flutter Programming Learning Assistant System (FPLAS)*. I discussed the overview, software architecture, client and server side of the proposed answer platform.

Chapter 7

Related Works in Literature

In this section, I introduce related works in literature relevant to this thesis.

7.1 Generation of Docker File and Image

In [56], Kitajima et al. proposed a method to recommend the latest version of *Docker* image for the automatic base image update of *Docker* file.

In [57], Yin et al. presented a specialized tagging approach for *Docker* repositories to address the problem of automatically multi-labeling a large number of repositories.

In [58], Hassan et al. presented a novel approach to recommend updating *Docker* files by analyzing software environments during software evolutions. This method tracked environment accesses from the codes to extract environment-related scopes of both old and new software versions.

In [59], Huang et al. developed a fast-building method for accelerating *Docker* image to be adopted in the efficient development and deployment of the container. They adopted a file caching mechanism to minimize the expensive file downloading, to repeat the operations of the execution of *Docker* instructions, and reuse the cached *Docker* image layers from the disk.

In [60], Schermann et al. proposed a structured information method on the state and evolution of *Docker* files on *GitHub*. They collected over 100,000 unique *Docker* files in 15,000 *GitHub* projects and analyzed them to recommend the researchers to write best practices of *Docker* file.

In [61], Nüst et al. suggested following the simple rules to writing understandable *Docker* files for typical data science during docker image building.

In [62], Zhong et al. proposed an automatic recipe generation system named Burner. It enables users with no professional computer background to generate recipes.

In [63], Lu et al. presented an empirical case study by smelling the real-world *Docker* files on *Docker Hub* to avoid accidental deletions of important temporary files that are needed in *Docker* image layer processing.

In [64], Zou et al. analyzed the industrial 2923 Dockerized projects and a small number of open-source software on the branches of GitHub versions control system.

In [65], Wu et al. proposed an empirical study on *Docker* file changes for 4110 open-source projects hosted on *GitHub*. They measured the frequency, magnitude, and instructions of *Docker* file changes and reported how it was co-changed with other files.

In [66], Xu et al. presented to detect the temporary file smell with dynamic and static analysis. In the image-building process, temporary files are frequently used to import applications and data.

Careless use of *Docker* file may cause temporary files to be left in the image, which can increase the image size.

In [67], Zhang et al. approached an empirical study on a large dataset of 2840 projects on the impacts of *Docker* file evolutionary trajectories on quality and latency in the *Docker*-based containerization.

In [68], Zhou et al. presented a semi-supervised learning-based tag recommendation approach, SemiTagRec, for *Docker* repositories, which contains four components.

In [69], Wu et al. presented how to enhance the project maintenance and practice *Docker* file by smelling in open-source *Docker*-based software developments. They showed an empirical study on a large dataset of 6334 projects to help developers gain some insights into the occurrence of *Docker* file smells, including their coverage, distribution, co-occurrence, and correlation with project characteristics.

In the range of papers [56]-[69], there are novel ideas proposed for searching online data sources of *Docker* files, analyzing and updating *Docker* instructions to generate *Docker* files and images, and storing *Docker* instructions using *Git* repositories. These ideas provide valuable insights and comparison opportunities for the proposed tool's functionality.

Additionally, the reviewed papers provide strategies for efficiently retrieving *Docker* files from diverse online sources, techniques for analyzing and optimizing *Docker* instructions (such as static analysis, vulnerability detection, and code pattern recognition), and best practices for managing *Docker* instructions using *Git* repositories. These insights enable the proposed tool to build a comprehensive dataset, improve the quality, security, and efficiency of *Docker* files through automated updates and recommendations, and utilize *Git* for effective version control, organization, and collaboration.

7.2 Software Tool

In [70], Kuflewski et al. presented a comparative study of the *Laravel* and *Symfony PHP* frameworks, using a model that evaluates seven dimensions: features, multilingualism, system requirements, architecture, code organization, continuous integration, and documentation. The results demonstrate that this model helps IT developers select the most suitable *PHP* framework.

In [71], Horton et al. presented *DockerizeMe*, a technique for inferring the dependencies needed to execute *Python* code snippets without import errors.

In [72], Forde et al. presented a tool *repo2docker* that checks the minimum requirements to reproduce a text file by building a Docker image based on a repository path or URL. The goal is to minimize the efforts needed to convert a static repository into a working software environment.

In [73], Sunardi et al. presented a comparative study between the *Laravel* and *Slim* frameworks in implementing the *MVC (Model View Controller)* architecture. MVC is a widely used design pattern for interactive software systems, separating data manipulation (Model), interface display (View), and process management (Controller) for clearer, more structured development.

In [74], Wodyk et al. compared the performance of *MySQL* and *PostgreSQL* relational databases in PHP applications using the *Laravel* framework. They evaluated query performance, including simple queries and those with column and table concatenation.

The papers [70]-[74] offer valuable insights for creating application tools by selecting the appropriate programming frameworks, databases, and design patterns based on specific requirements. These ideas can guide the design and generation of *Docker* files and images optimized for performance, maintainability, and scalability.

By examining these studies, the proposed tool can explore various methodologies for selecting frameworks, considering functional and non-functional requirements, performance benchmarks, and *Docker* compatibility. This will enable the tool to make informed decisions when recommending or generating Docker files and images with the most suitable programming frameworks.

Furthermore, the papers emphasize the importance of selecting the right databases based on data size, performance, scalability, and consistency. The proposed tool can incorporate these criteria when generating *Docker* files and images, ensuring efficient data storage and retrieval.

The comparative analysis of *DockerizeMe* [71] and *repo2docker* [72] tools provides additional insights into tool design. By studying these tools, the proposed tool can learn from their implementation strategies, improve its own structure, and apply similar design principles.

Overall, these studies propose methods for searching online data sources for *Docker* files, storing large datasets, analyzing *Docker* files, recommending changes, and generating *Docker* files and images for selected frameworks, databases, and design patterns. They also highlight the role of *Git* repositories for version control and change management. The business logic, data analysis, programming language choices, system design, and testing methodologies discussed in these papers are valuable for the proposed tool, which incorporates many of these techniques for its implementation.

7.3 Mobile Learning

In [75], Costa et al. focused on teaching beginner programming through interactive editors that provide immediate output, showing that some support tools are more effective than others for learning.

In [76], Costa et al. examined using robotics to improve programming education by proposing a framework (*dragon-robot*) to address identified learning challenges.

In [77], Ferreira et al. introduced *Cognitive-Driven Development (CDD)*, a design technique that helps developers manage cognitive complexity, and applied it to *Flutter*. Their study found that *CDD* improves code quality and readability for novice developers.

In [78], Khan et al. discussed the global shift towards *Mobile Learning (M-Learning)*, especially in educationally advanced countries. They outlined five key lessons in adopting *M-Learning*, particularly for countries lagging in implementations, like those in the Middle East.

In [79], Crompton et al. provided a systematic review of mobile learning research from 2010-2016, noting the growing use of mobile devices in higher education and their positive impacts on student achievements, particularly in language instructions.

In [80], Hsu et al. used the *Delphi* method to identify future mobile learning research priorities, emphasizing teaching strategies, theory development, and infrastructure.

In [81], Koole et al. introduced the *FRAME* model, which examines the convergence of mobile technology, human learning, and social interaction. This model addresses modern educational challenges like information overload and collaboration.

These studies highlight the effectiveness of mobile learning platforms, cognitive-driven approaches, and collaborative frameworks in improving programming educations across both mobile and web-based environments.

7.4 Collaborative Coding

In [82], Goldman et al. presented *Collabode*, a web-based *Java* IDE designed to support synchronous collaborations between programmers. It uses an algorithm that isolates individual errors, allowing users to compile error-free portions of code while ignoring incomplete changes from collaborators.

In [83], Pathak et al. presented a real-time collaborative code editor using *React*, *Node.js*, and *Socket.IO*. It integrates live chat to enable seamless remote collaborations, allowing developers to code and communicate in real-time.

In [84], Tung et al. introduced *PLWeb*, a system to assist instructors in designing programming exercises and helping students practice. It features a user-friendly editor, plagiarism detection, and visualized learning tools to support incremental learning.

In [85], Feiz et al. presented focuses on identifying app screen similarities and changes using machine learning models. Their system can detect screen transitions and interactions like dialogues or keyboards appearing or disappearing, useful for app crawling and automation.

In [86], Wu et al. explored UI screen parsing from screenshots, allowing automated understanding of UI elements and their relationships. The system significantly improves screen parsing accuracy, benefiting accessibility, task automation, and code generation.

These studies establish foundations for my web-based answer platform. Insights into collaborative coding tools will guide the exercise management, while advancements in screenshot analysis will enhance functionality, creating a comprehensive environment for effective learning and programming exercise management.

7.5 Docker

In [87], Cito et al. discussed the importance of reproducibility in software engineering and how *Docker* containers can mitigate issues related to undocumented assumptions, dependencies, and configurations, aiding in the reproducibility of research artifacts.

In [88], Kithulwatta et al. evaluated the performance of *Apache* and *Nginx* web servers deployed on *Docker*, finding that both servers demonstrate high availability and comparable performance metrics, although *Apache* outperformed *Nginx* in requests per second.

In [89], Ibrahim et al. explored the benefits of *Docker* and *Docker Compose* for deploying multi-component applications. Their study examines the usage patterns of *Docker Compose* in open-source projects, revealing that many applications utilize basic features instead of advanced options.

In [90], Badisa et al. focused on optimizing *Docker* image sizes to improve resource utilization and deployment efficiency. It discusses various techniques for size reduction, including eliminating unnecessary dependencies and utilizing *Nginx* as a reverse proxy to enhance application performance.

Together, these studies guide my decisions in selecting appropriate web servers, configuring *Docker* and *Docker Compose*, and optimizing container deployment for my implementation.

Chapter 8

Conclusion

In this thesis, I presented the implementations of two web applications for university education using various open-source software, including *Docker* technology.

The first contribution presented the development and evaluation of a web-based *Docker image generation assisting (DIG-assist)* tool designed to assist the *Docker* image generation process in the *user-PC computing (UPC)* system. The tool utilizes the *Angular JavaScript* framework on the client side to implement the view (V) of the MVC model for interactive interfaces. On the server side, the *PHP Laravel* framework is employed to implement the controller (C) for handling *HTTP* requests and responses between the client and server, while *MySQL* database is used to handle data manipulations and storage for model (M). *Shell Scripting* is used to generate *Docker* files for new jobs and their corresponding *Docker* images, which are then uploaded directly to the designated location on *Docker Hub*. Furthermore, the tool incorporates a worker-side code modification function to enable users to modify the source code of running jobs and update the *Docker* image at a worker to speed up source code changes during development. To evaluate the tool, I conducted experiments of generating 30 *Docker* images using the DIG-assist tool from *Docker* files collected through reverse processing from online *GitHub* or *Docker Hub*. The experiment results showed that the proposed tool successfully generated and ran *Docker* images, confirming the validity of the proposal.

The second contribution presented a personal web-based answer platform for the *Flutter Programming Learning Assistant System (FPLAS)*, aimed at simplifying the learning process for novice programmers through the innovative use of *Docker* compose. The platform integrates four *Docker* images as multi-container instances and extends three prior implementations to create a unified learning environment. The primary goal was to address key challenges in programming education, such as reducing setup complexity, providing immediate feedback, and supporting diverse operating systems, which were effectively achieved through the proposed system.

Several enhancements were introduced, such as theme color changes, additional routes, a student ID input box, and an answer dialog box to display student answer images alongside correct images for visual comparison. For the *Flutter* environment, the system incorporated a file-watching service to monitor source code changes and automatically capture output UI images. The compiled *Flutter/Dart* code was transferred to an *Nginx* web application server, enabling real-time displays of three images: the student's answer, the correct answer, and the highlighted differences. These features streamlined the process of verifying solutions and enhanced the feedback mechanism, addressing the study's goal of improving student learning outcomes.

To validate the proposed platform, I developed five exercise projects with modification guidance and sample source code. These projects allowed students to practice modifying, executing,

and verifying their code, promoting active and practical learning. The system was tested by 10 graduate students at *Okayama* university, where all of them successfully installed the platform and completed the exercises. The application results confirmed the effectiveness. Additionally, usability questionnaires revealed positive feedback, with students rating the system's usability as moderate to high.

In future work, I will develop other functions for handling various *Docker*-based jobs, integrating GPU support for the *User-PC Computing (UPC)* system, and improving advanced exercise assignments by incorporating *YOLO* object detection, machine learning, and AI into the *Flutter Programming Learning Assistant System (FPLAS)*. Additionally, efforts will be directed toward expanding the scalability and flexibility of the system to accommodate more users and diverse learning environments, ensuring adaptability to a wide range of educational needs.

Appendix A

1. The file in Listing 1 defines and manages the installation of *Node.js* and the necessary npm packages and dependencies required for running the answer platform, along with essential setup instructions.

Listing 1: Answer platform Docker file.

```
FROM node:16-buster as base

LABEL email=lynnhtetaung@s.okayama-u.ac.jp
RUN apt-get update; apt-get install -y curl \
   && curl -sL https://deb.nodesource.com/setup_16.x | bash - \
   && apt-get install -y nodejs \
   && curl -L https://www.npmjs.com/install.sh | sh

WORKDIR /app

COPY . /app

RUN npm install

EXPOSE 4000

CMD [``npm'', ``start'']
```

2. The file in Listing 2 is used for the *Flutter* extension. It specifies the necessary *Flutter SDK* and system configurations for the container environment, extensions, and development tools.

Listing 2: Flutter environment Docker file.

```
# Use a Debian-based image to install Flutter
FROM debian:latest AS flutter-build

# Install required packages
RUN apt-get update \
   && apt-get install -y --no-install-recommends \
      curl git wget unzip \
   && apt-get clean \
   && rm -rf /var/lib/apt/lists/*

# Clone Flutter SDK
RUN git clone https://github.com/flutter/flutter.git /usr/local
   /flutter

# Set Flutter in PATH
ENV PATH=''/usr/local/flutter/bin:/usr/local/flutter
/bin/cache/dart-sdk/bin:${PATH}''

# Check Flutter installation and enable web support
RUN flutter doctor
RUN flutter channel master \
   && flutter upgrade \
   && flutter config --enable-web

# Set up the working directory and copy application files
WORKDIR /app
COPY . /app

# Install dependencies and build the Flutter web app
RUN flutter clean \
   && flutter pub get \
   && flutter build web

# Serve the built web application
EXPOSE 8080
CMD [''flutter'', ''build'']
```

3. The file in Listing 3 is used to manage the *Nginx* web server. It configures the web hosting environment for the compiled *JavaScript* code and is commonly used to render this code to generate a UI output image.

Listing 3: Nginx Docker file.

```
FROM nginx:1.25.2-alpine

COPY build/web/* /usr/share/nginx/html

EXPOSE 80

CMD [''nginx'', ''-g'', ''daemon off;'']
```

4. The file in Listing 4 is used to configure the image-based *UI testing* tool. It installs essential tools and libraries within the container environment to enable automated *UI testing*. This setup includes *Watchdog* for monitoring file changes, *Playwright* for automatically capturing UI output images, and *OpenCV* for analyzing and comparing images to assess similarity.

Listing 4: UI testing tool Docker file.

```
# Use a Debian-based image to install Python and Playwright
FROM debian:latest AS python-playwright

# Install required packages
RUN apt-get update \
   && apt-get install -y --no-install-recommends \
      python3 python3-pip python3-venv \
      fonts-liberation libatk-bridge2.0-0 libatk1.0-0
      libatspi2.0-0 libcairo2 libcups2 \
      libgbm1 libgtk-3-0 libnspr4 libnss3
      libpango-1.0-0 libu2f-udev libxcomposite1 \
      libxdamage1 libxrandr2 \
   && apt-get clean \
   && rm -rf /var/lib/apt/lists/*

# Set up a Python virtual environment and install necessary
   packages
RUN python3 -m venv /venv \
   && /venv/bin/pip install --upgrade pip setuptools wheel \
   && /venv/bin/pip install flask watchdog opencv-python
   Pillow pytest-playwright \
   && /venv/bin/playwright install \
   && /venv/bin/playwright install-deps

# Set the Python virtual environment in the PATH
ENV PATH=''/venv/bin:$PATH''

# Set up the working directory and copy application files
WORKDIR /app
COPY . /app

# Expose the port for Flask
EXPOSE 5000
CMD [''python3'', ''main.py'']
```

5. The file in Listing 5 defines and manages various *Docker* images and configures the services required to run as multi-container setups for the *answer platform*, *Flutter* environment, *Nginx* web server, and image-based *UI testing* tool in a single file. This file is commonly used in complex development environments that require multiple services, as it simplifies the management of volumes, networks, ports, and bind mounts.

Listing 5: docker-compose.yml.

```
services:

 answer-platform:
   image: lynnhtetaung/fplas-answer-platform:v1
   volumes:
     - /var/run/docker.sock:/var/run/docker.sock
     - /usr/bin/docker:/usr/bin/docker
     - /home/lynnhtetaung/Desktop/results:/app/addon/output

   networks:
     - app-network
   ports:
     - 4000:4000

 flutter-environment:
   image: lynnhtetaung/fplas-flutter-environment:v1
   volumes:
     - shared-data:/app/build/web
     - /var/run/docker.sock:/var/run/docker.sock
     - /usr/bin/docker:/usr/bin/docker
   networks:
     - app-network
   ports:
     - 8000:8000

 nginx-server:
   image: lynnhtetaung/fplas-nginx-webserver:v1
   volumes:
     - shared-data:/usr/share/nginx/html
   networks:
     - app-network
   ports:
     - 8080:8080

 image-testing-tool:
   image: lynnhtetaung/fplas-image-testing-tool:v1
   volumes:
     - shared-data:/app/build/web
     - /var/run/docker.sock:/var/run/docker.sock
     - /usr/bin/docker:/usr/bin/docker
   networks:
     - app-network
   ports:
     - 5000:5000

volumes:
 shared-data:

networks:
 app-network:
   driver: bridge
```

Bibliography

[1] H. Htet, N. Funabiki, A. Kamoyedji, M. Kuribayashi, F. Akhter, and W.-C. Kao, "An implementation of user-PC computing system using Docker container," Int. J. Future Comput. Commun. (IJFCC), vol. 9, no. 4, pp. 66-73, Dec. 2020.

[2] S. T. Aung, N. Funabiki, L. H. Aung, S. A. Kinari, M. Mentari, K. H. Wai, "A Study of Learning Environment for Initiating Flutter App Development Using Docker," *MDPI Information*, vol. 15, no. 191, Apr. 2024.

[3] L. H. Aung, N. Funabiki, S. T. Aung, X. Zhou, X. Xiang, and W. -C. Kao, "A Web-Based Docker Image Assistant Generation Tool for User-PC Computing System," *MDPI Information* vol. 14(6), no. 300, June 2023. https://doi.org/10.3390/info14060300.

[4] L. H. Aung, S. T. Aung, N. Funabiki, H. H. S. Kyaw, and W. -C. Kao, "An Implementation of Web-Based Answer Platform in the Flutter Programming Learning Assistant System Using Docker Compose," *MDPI Electronics*, vol. 13, no. 4878, Dec. 2024. https://doi.org/10.3390/electronics13244878.

[5] S. T. Aung, L. H. Aung, N. Funabiki, S. Yamaguchi, Y. W. Syaifudin, and W. -C. Kao, "An implementation of web-based personal platform for programming learning assistant system with instance file update function," *Eng. Lett.*, vol. 32, pp. 226-243, Feb. 2024.

[6] S.T. Aung, N. Funabiki, L. H. Aung, S. A. Kinari, K. H. Wai, M. Mentari, "An Image-Based User Interface Testing Method for Flutter Programming Learning Assistant System," *MDPI Information*, vol. 15, no. 464, Aug. 2024.

[7] "HTML: Hypertext Markup Language," Internet: https://developer.mozilla.org/en-US/docs/Web/HTML, (Accessed on 28 Dec., 2022).

[8] "CSS," Internet: https://developer.mozilla.org/en-US/docs/Web/CSS, (Accessed on 28 Dec., 2022).

[9] "Bootstrap," Internet: https://getbootstrap.com/, (Accessed on 9 Dec., 2024).

[10] "PrimeNG," Internet: https://www.primefaces.org/primeng/, (Accessed on 28 Dec., 2022).

[11] "JavaScript," Internet: https://developer.mozilla.org/en-US/docs/Web/JavaScript,(Accessed on 28 Dec., 2022).

[12] "Angular," Internet: https://angular.io/, (Accessed on 28 Dec., 2022).

[13] "EJS," Internet: https://ejs.co/, (Accessed on 9 Dec., 2024).

[14] “Node.js,” Internet: https://nodejs.org/en/, (Accessed on 28 Dec., 2022).

[15] “Laravel,” Internet: https://laravel.com/, (Accessed on 28 Dec., 2022).

[16] “Python,” Internet: https://www.python.org/, (Accessed on 9 Dec., 2024).

[17] “Watchdog,” Internet: https://pypi.org/project/watchdog/, (Accessed on 9 Dec., 2024).

[18] “Playwright,” Internet: https://playwright.dev/python/docs/intro, (Accessed on 9 Dec., 2024).

[19] “Flask,” Internet: https://flask.palletsprojects.com/en/3.0.x/ , (Accessed on 9 Dec., 2024).

[20] “OpenCV,” Internet: https://docs.opencv.org/4.x/, (Accessed on 9 Dec., 2024).

[21] “Flutter,” Internet: https://docs.flutter.dev/, (Accessed on 9 Dec., 2024).

[22] “Dart,” Internet: https://dart.dev/overview/, (Accessed on 9 Dec., 2024).

[23] “MySQL,” Internet: https://dev.mysql.com/, (Accessed on 28 Dec., 2022).

[24] “Shell Scripting Tutorial,” Internet: https://www.shellscript.sh/, (Accessed on 28 Dec., 2022).

[25] “Docker,” Internet: https://docs.docker.com/get-docker/, (Accessed 28 Dec., 2022).

[26] “Dockerfile,” Internet: https://docs.docker.com/engine/reference/builder/, (Accessed on 28 Dec., 2022).

[27] “Docker Image,” Internet: https://hub.docker.com/-/docker, (Accessed 28 Dec., 2022).

[28] “Docker Container,” Internet: https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-a-container/, (Accessed on 9 Dec., 2022).

[29] “Docker Compose,” Internet: https://docs.docker.com/compose/, (Accessed on 9 Dec., 2022).

[30] “DockerHub,” Internet: https://hub.docker.com/signup/, (Accessed on 28 Dec., 2022).

[31] “Nginx,” Internet: https://nginx.org/en/, (Accessed on 9 Dec., 2024).

[32] “GitHub,” Internet: https://docs.github.com/en, (Accessed on 9 Dec., 2024).

[33] “Tar File—WinZip,” Internet: https://www.winzip.com/en/learn/file-formats/tar, (Accessed on 28 Dec., 2022).

[34] “Docker Save,” Internet: https://docs.docker.com/engine/reference/commandline/save/, (Accessed on 28 Dec., 2022).

[35] "Web Browser," Internet: https://en.wikipedia.org/wiki/Web-browser, (Accessed on 28 Dec., 2022).

[36] "Graphical User Interface," Internet: https://en.wikipedia.org/wiki/Graphical-user-interface, (Accessed on 28 Dec., 2022).

[37] "TypeScript," Internet: https://www.typescriptlang.org/,(Accessed on 28 Dec., 2022).

[38] "RestAPI," Internet: https://www.ibm.com/cloud/learn/rest-apis, (Accessed on 28 Dec., 2022).

[39] "HTTP: Hypertext Transfer Protocol," Internet: https://en.wikipedia.org/wiki/Hypertext-Transfer-Protocol, (Accessed on 28 Dec., 2022).

[40] "AngularCLI," Internet: https://angular.io/cli, (Accessed on 28 Dec., 2022).

[41] "NVM-SH/NVM: Node Version Manager," Internet: https://github.com/nvm-sh/nvm, (Accessed on 28 Dec., 2022).

[42] "Package.json," Internet: https://docs.npmjs.com/cli/v9/configuring-npm/package-json, (Accessed on 28 Dec., 2022).

[43] "NPM Docs," Internet: https://docs.npmjs.com/, (Accessed on 28 Dec., 2022).

[44] "JSON," Internet: https://www.json.org/json-en.html, (Accessed on 28 Dec., 2022).

[45] "Laradock," Internet: https://laradock.io/, (Accessed on 28 Dec., 2022).

[46] "PHP," Internet: https://www.php.net/, (Accessed on 28 Dec., 2022).

[47] "Composer," Internet: https://getcomposer.org/download/, (Accessed on 28 Dec., 2022).

[48] "MySQL Database Service Guide," Internet: Availableat:Https://docs.oracle.com/en-us/iaas/mysql-database/doc/getting-started-mysql-database-service.html, (Accessed on 28 Dec., 2022).

[49] "Express," Internet: https://expressjs.com/, (Accessed on 9 Dec., 2024).

[50] "Visual Studio Code," Internet: https://code.visualstudio.com/docs, (Accessed on 9 Dec., 2024).

[51] "ORB," Internet: https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html, (Accessed on 9 Dec., 2024).

[52] "SIFT," Internet: https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html, (Accessed on 9 Dec., 2024).

[53] "CodeMirror," Internet: https://codemirror.net/, (Accessed on 9 Dec., 2024).

[54] "Docker Bind Mount (Bind Mount)," Internet: https://docs.docker.com/engine/storage/bind-mounts/, (Accessed on 9 Dec., 2024).

[55] "System Usability Scale (SUS)," Internet: https://credoagency.co.uk/usability-in-cro-the-system-usability-scale-sus/, (Accessed on 9 Dec., 2024).

[56] S. Kitajima, A. Sekiguchi, "Latest image recommendation method for automatic base image update in dockerfile," in Proc. 18th Int. Conf. on Serv. -Orient. Comput. (ICSOC), Dubai, United Arab Emirates, pp. 547–562, Dec. 2020. https://doi.org/10.1007/978-3-030-65310-1.40.

[57] K. Yin, W. Chen, J. Zhou, G. Wu, J. Wei, "Star: A specialized tagging approach for Docker repositories," in Proc. 25th Asia-Pacific Soft. Eng. Conf. (APSEC), Nara, Japan, pp. 426-435, Dec. 2018. https://doi.org/10.1109/apsec.2018.00057.

[58] F. Hassan, R. Rodriguez, X. Wang, "Rudsea: Recommending updates of Dockerfiles via software environment analysis," in Proc. 33rd ACM/IEEE Int. Conf. on Automated Soft. Eng. (ASE), Montpellier, France, pp. 796–801, Sept. 2018. https://doi.org/10.1145/3238147.3240470.

[59] Z. Huang, S. Wu, S. Jiang, H. Jin, "FastBuild: Accelerating docker image building for efficient development and deployment of container," in Proc. 35th Symposium on Mass Storage Systems and Technologies (MSST), Santa Clara, CA, USA, pp. 28-37, May 2019. https://doi.org/10.1109/msst.2019.00-18.

[60] G. Schermann, S. Zumberi, J. Cito, "Structured information on state and evolution of dockerfiles on github," in Proc. 15th International Conference on Mining Software Repositories, Gothenburg, Sweden, pp. 26–29, May 2018. https://doi.org/10.1145/3196398.3196456.

[61] D. Nüst, V. Sochat, B. Marwick, SJ. Eglen, T. Head, T. Hirst, B. D. Evans, "Ten simple rules for writing Dockerfiles for reproducible data science," *PLoS Comput. Biol.*, vol. 16, no. e1008316, Apr. 2020. https://doi.org/10.1371/journal.pcbi.1008316.

[62] S. Zhong, D. Wang, W. Li; F. Lu, H. Jin, "Burner: Recipe automatic generation for HPC container based on domain knowledge graph," *Wirel. Commun. Mob. Comput.*, vol. 16, no. 4592428, Jan. 2020. https://doi.org/10.1155/2022/4592428.

[63] Z. Lu, J. Xu, Y. Wu, T. Wang, T. Huang, "An empirical case study on the temporary file smell in Dockerfiles," *IEEE Access*, vol. 7, pp. 63650-63659, . Mar2019. https://doi.org/10.1109/ACCESS.2019.2905424.

[64] W. Zou, W. Zhang, X. Xia, R. Holmes, Z. Chen, "Branch Use in Practice: A large-scale empirical study of 2,923 projects on GitHub," in Proc. IEEE 19th Int. Conf. on Soft. Quality, Relia. and Secu. (QRS), Sofia, Bulgaria, pp. 306–317, July 2019. https://doi.org/10.1109/qrs.2019.00047.

[65] Y. Wu, Y. Zhang, T. Wang, H. Wang, "Characterizing the occurrence of dockerfile smells in open-source software: An empirical study," *IEEE Acces*, vol. 8, pp. 34127–34139, May 2020. https://doi.org/10.1109/access.2020.2973750.

[66] J. Xu, Y. Wu, Z. Lu, T. Wang, "Dockerfile TF smell detection based on dynamic and static analysis methods ,"in Proc. IEEE 43rd Annual Comput. Soft. and Apps. Conf. (COMPSAC),

Milwaukee, WI, USA, pp. 185–190, July 2019. https://doi.org/10.1109/compsac.2019.00033.

[67] Y. Zhang, G. Yin, T. Wang, Y. Yu, H. Wang, "An insight into the impact of dockerfile evolutionary trajectories on quality and latency," in Proc. IEEE 42nd Annual Comput. Soft. and Apps. Conf. (COMPSAC), Tokyo, Japan, pp. 138–143, July 2018. https://doi.org/10.1109/compsac.2018.00026.

[68] J. Zhou, W. Chen, G. Wu, J. Wei, "Semitagrec. A semi-supervised learning based tag recommendation approach for docker repositories," in Proc. of the 18th Int. Conf. on Soft. and Systems Reuse (ICSR), Cincinnati, OH, USA, pp 132–148, June 2019. https://doi.org/10.1007/978-3-030-22888-0_10.

[69] Y. Wu, Y. Zhang, T. Wang, H. Wang, "Dockerfile changes in practice: A large-scale empirical study of 4,110 projects on github," in Proc. of the 27th Asia-Pacific Soft. Eng. Conf. (APSEC), Singapore, pp. 247-256, Dec. 2020.https://doi.org/10.1109/apsec51365.2020.00033.

[70] K. Kuflewski ; M. Dzieńkowski, "Symfony and Laravel—A comparative analysis of PHP programming frameworks," *J. Comput. Sci. Inst.*, vol. 21, pp. 367–372, Oct. 2021. https://doi.org/10.35784/jcsi.2749.

[71] E. Horton, C. Parnin, "Dockerizeme: Automatic inference of environment dependencies for Python code snippets," in Proc. 41st Int. Conf. on Soft. Eng. (ICSE), Montreal, QC, Canada, pp. 328–338, May 2019. https://doi.org/10.1109/icse.2019.00047.

[72] J. Forde, T. Head, C. Holdgraf, Y. Panda, G. Nalvarete, B. Ragan-Kelley, E. Sundell, "Reproducible research environments with Repo2Docker," in Proc. of the RML Works. -Repro. in Machine Learning (ICML), Stockholm, Sweden, pp. 1-5, July 2018.

[73] A. Sunardi, Suharjito, "MVC Architecture: A comparative study between Laravel framework and Slim framework in freelancer project monitoring system web based," *Procedia Comput. Sci.*, vol. 157, pp. 134–141, Sept. 2019. https://doi.org/10.1016/j.procs.2019.08.150.

[74] R. Wodyk, M. Skublewska-Paszkowska, "Performance comparison of relational databases SQL Server, MySQL and PostgreSQL using a web application and the Laravel framework," *J. Comput. Sci. Inst.*, vol. 7, pp. 358–364, Oct. 2020. https://doi.org/10.35784/jcsi.2279.

[75] C. J. Costa, M. Aparicio, C. Cordeiro. "Web-based graphic environment to support programming in the beginning learning process ," in Proc. of the Ent. Comput. (ICEC) Lect. Notes in Comput. Science, Heidelberg, Germany, pp. 413–416, Sept. 2012.

[76] C. J. Costa, M. Aparicio, C. Cordeiro, "A solution to support student learning of programming," in Proc. Work. on Open Source and Design of Commu. (OSDOC '12), Lisboa, Portugal, pp. 25–29, June 2012.

[77] R. Ferreira, V. H. S. C. Pinto, D. B. R. C. Souza, G. Pinto, "Assisting Novice Developers Learning in Flutter Through Cognitive-Driven Development," in Proc. Brazi. Symp. on Soft. Eng. (SBES '24), Curitiba, Brazil, pp. 1–10, Sept. 2024.

[78] A. I. Khan, H. Al-Shihi, Z. A. Al-Khanjari, M. Sarrab, "Mobile Learning (M-Learning) adoption in the Middle East: Lessons learned from the educationally advanced countries," *Telemat. Inform.*, vol. 32, pp. 909-920, Oct. 2015. http://dx.doi.org/10.1016/j.tele.2015.04.005

[79] H. Crompton, D. Burke, "The use of mobile learning in higher education: A systematic review," *Comput. Educ.*, vol. 123, pp. 53-64, Sept. 2018. http://dx.doi.org/10.1016/j.compedu.2018.04.007

[80] Y.-C. Hsu,Y.-H. Ching, C. Snelson, "Research priorities in mobile learning: An international Delphi study," *Can. J. Learn. Technol.*, vol. 40, pp. 1-22, Nov. 2014. http://dx.doi.org/10.21432/T2QP4X

[81] M. L. Koole, "A model for framing mobile learning," in Proc. Mobile Learning: Trans. the Deli. of Edu. and Train. Ally. M., Ed. AU Press, Edmonton, AB, Canada, pp. 25–47, Oct. 2009.

[82] M. Goldman, G. Little, C. M. Robert, "Real-time collaborative coding in a web IDE," in Proc. of the 24th Annual ACM Symp. on User Intf. Soft. and Tech. (UIST '11). Asso. for Comput. Mach. , New York, NY, USA, pp. 155-164, Oct. 2011 http://dx.doi.org/10.1145/2047196.2047215

[83] H. Pathak, H. Ritik, R. Pawar, Y. Pingle, "CodeFlow: Real-Time Collaborative Code Edito," in Proc. 2024 Int. Conf. on Know. Eng. and Comm. Systems (ICKECS), Chikkaballapur, India, pp. 1-5, Apr. 2024. http://dx.doi.org/10.1109/ICKECS61492.2024.10617308

[84] S. H. Tung, T. T. Lin, Y. H. Lin, "An exercise management system for teaching programming," *J. Softw.*, vol. 8, pp. 1718-1725, July 2013. http://dx.doi.org/10.4304/jsw.8.7.1718-1725

[85] S. Feiz, J. Wu, X. Zhang, A. Swearngin, T. Barik, J. Nichols, "Understanding Screen Relationships from Screenshots of Smartphone Applications," in Proc. 27th Int. Conf. on Intel. User Intf. (IUI '22). Asso. for Comput. Mach., New York, NY, USA, pp. 447-458, Mar. 2022. http://dx.doi.org/10.1145/3490099.3511109

[86] J. Wu, X. Zhang, J. Nichols, J. P. Bigham, "Screen Parsing: Towards Reverse Engineering of UI Models from Screenshots," in Proc. 34th Annual ACM Symp. on User Intf. Soft. and Tech. (UIST '21). Asso. for Comput. Mach., New York, NY, USA, pp. 470-483, Oct. 2021. http://dx.doi.org/10.1145/3472749.3474763

[87] J. Cito, H. C. Gall, "Using Docker containers to improve reproducibility in software engineering research," in Proc. IEEE/ACM 38th Int. Conf. on Soft. Eng. Comp. (ICSE-C), Austin, TX, USA, pp. 906-907, May 2016.

[88] W. M. C. J. T. Kithulwatta, K. P. N. Jayasena, B. T. G. S. Kumara, R. M. K. T. Rathnayaka, "Performance Evaluation of Docker-based Apache and Nginx Web Server," in Proc. 3rd Int. Conf. for Emerg. Tech. (INCET), Belgaum, India, pp. 1-6, May 2022. http://dx.doi.org/10.1109/INCET54531.2022.9824303

[89] M.H. Ibrahim, M. Sayagh, A. E. Hassan, "A study of how Docker Compose is used to compose multi-component systems," *Empir. Soft. Eng.*, vol. 26, no. 128, Sept. 2021. http://dx.doi.org/10.1007/s10664-021-10025-1

[90] N. Badisa, J. K. Grandhi, L. Kallam, "*Efficient Docker Image Optimization using Multi-Stage Builds and Nginx for Enhanced Application Deployment*," pp. 1-7, Aug. 2023. http://dx.doi.org/10.21203/rs.3.rs-3276965/v1