information

Article

A Study of Learning Environment for Initiating Flutter App Development Using Docker

Soe Thandar Aung *, Nobuo Funabiki *, Lynn Htet Aung, Safira Adine Kinari, Mustika Mentari and Khaing Hsu Wai

Department of Information and Communication Systems, Okayama University, Okayama 700-8530, Japan; lynnhtetaung@s.okayama-u.ac.jp (L.H.A.)

* Correspondence: soethandar@s.okayama-u.ac.jp (S.T.A.); funabiki@okayama-u.ac.jp (N.F.)

Abstract: The *Flutter* framework with *Dart* programming allows developers to effortlessly build applications for both web and mobile from a single codebase. It enables efficient conversions to native codes for mobile apps and optimized *JavaScript* for web browsers. Since utilizing a wide range of widgets in *Flutter* ensures consistent experiences on various devices for users, it becomes crucial in programming education by providing a unified environment for learning app development while reducing the need for platform-specific knowledge. However, the setup of the *Flutter* environment is challenging for novice students due to its multiple steps, such as installing dependencies and configuring environments. To support independent learning for these students, it is essential to simplify the setup by providing user-friendly instructions and automated tools. In this paper, we present a *Docker*-based environment for *Flutter* app developments across *Windows*, *Linux*, and *Mac* through *Visual Studio Code*, ensuring a unified learning experience. This paper aims to simplify complex configurations and address the obstacles encountered by students when initiating *Flutter* projects. For the evaluation, we prepared three simple *Flutter* projects along with the setup environment in a *Docker* container. Then, we asked 24 Master's students at Okayama University, Japan, to install the environment and modify the source codes in the projects independently by following the given instructions. The results show that all the students successfully completed the assignments, which confirms the efficiency and validity of our proposal.

Keywords: *Flutter*; *Dart*; app; *Docker*; *Visual Studio Code*; environment; code modification

Citation: Aung, S.T.; Funabiki, N.; Aung, L.H.; Kinari, S.A.; Mentari, M.; Wai, K.H. A Study of Learning Environment for Initiating Flutter App Development Using Docker. *Information* **2024**, *15*, 191. https://doi.org/10.3390/info15040191

Academic Editors: Ricardo Queirós and Mário Pinto

Received: 29 February 2024
Revised: 25 March 2024
Accepted: 28 March 2024
Published: 30 March 2024

1. Introduction

In recent years, the integration of mobile application development into educational curricula has become increasingly vital. With the pervasive use of mobile devices and the growing demand for digital skills, proficiency in app development is a valuable asset for students entering the workforce. Educational institutions, including the *Association for Computing Machinery (ACM)*, recognize the importance of incorporating mobile computing topics throughout the computer science curriculum [1].

The growing interest in leveraging mobile technologies for educational purposes, as noted by Rushby [2], has led to a surge in mobile learning research productivity. This productivity, explored by Kukulska-Hulme and Traxler [3], delves into learning potentials and associated critical issues. However, teaching mobile development can present challenges, particularly regarding the diversity of platforms and the complexities involved in setting up development environments. Hsu, Ching, and Snelson [4] highlighted the importance of elevated research efforts in mobile learning, emphasizing that enhanced conceptual and theoretical guidance is essential for supporting its design and research aspects. The benefits afforded by mobile technologies have prompted instructional technologists and researchers to adopt an educational perspective in developing multimedia applications for mobile devices aimed at enhancing teaching and learning. Within the expanding body of literature,

substantial initiatives are in progress to formulate models and frameworks customized for shaping mobile learning experiences and environments [5].

In this context, the choice of framework and programming environment is crucial. *Flutter*, a framework developed by *Google*, paired with the *Dart* programming language, offers a compelling solution. *Flutter* provides a unified platform for building applications across multiple devices, including mobile phones, tablets, and desktop computers, using a single codebase. Its "hot reload" feature enables rapid iteration and experimentation, making it well suited for educational purposes. Additionally, *Dart* offers a modern and efficient language for app development, with robust support for asynchronous programming and reactive patterns. By integrating these technological advancements, educational institutions can effectively harness mobile technologies to enhance learning experiences and environments, aligning with the ongoing research efforts in the field.

While there are alternative frameworks and languages available for mobile development, *Flutter* and *Dart* offer unique advantages. Traditional approaches often involve developing separate codebases for different platforms, such as using *Java* or *Kotlin* for *Android* and *Swift* or *Objective-C* for *iOS*. Cross-platform frameworks such as *React Native* and *Xamarin* also exist but may introduce additional complexities or limitations. By contrast, *Flutter* streamlines the development process, enabling students to focus on core concepts without the overhead of platform-specific knowledge or cumbersome setup procedures.

However, setting up the environment can be challenging for newcomers stepping into *Flutter* application developments due to the multiple steps of installing various dependencies, configuring the development environment, and understanding how to use the *Flutter* SDK effectively. Additionally, configuring IDEs such as *Visual Studio Code* or *Android Studio*, along with the device or emulator setup for testing, demands specific plugin installations, debugging configurations, and device connections. These complexities raise the hurdles to newcomers as they navigate through the intricacies of *Flutter* developments. Moreover, the setup process often involves installation steps that are susceptible to version conflicts or configuration errors, which may further complicate the initial learning experience.

To support novice students in learning *Flutter* independently, it is essential to facilitate the setup of its development environment to be as user-friendly and streamlined as possible. By providing simplified instructions and tools that automate the installation of the dependencies and configurations of the environment, students can focus more on learning *Flutter* itself without becoming stuck in the setup process. This approach ensures that students can quickly start *Flutter* application developments, empowering them to explore and experiment with confidence. Additionally, understanding the *Flutter* framework and *Dart* language usage is pivotal for students in application development, given significant impacts on the development process.

To further simplify the setup process and ensure a consistent development environment across different operating systems, we leverage *Docker*. *Docker* is a containerization platform that allows applications to be packaged with their dependencies, ensuring reproducibility and portability. By encapsulating the *Flutter* development environment within a *Docker* container, we provide a hassle-free solution for students, eliminating the need for manual configuration and dependency management.

Our proposal aims to address the challenges faced by students when learning *Flutter* and mobile app development. By providing a streamlined development environment and guided projects, we empower students to focus on learning core concepts and building practical skills. This approach not only facilitates the teaching of mobile development but also fosters creativity and innovation among students as they explore the possibilities of *Flutter* and *Dart*.

In this paper, we present a development environment based on *Docker* that simplifies the setup of *Flutter* applications. Students can create *Flutter* apps directly within a containerized environment, accessible through *Visual Studio Code (VSCode)* on various operating systems, including *Windows*, *Linux*, and *Mac*. Three sample *Flutter* projects are integrated into the *Docker* container, providing students with hands-on learning opportunities. By

encapsulating the *Flutter* development environment within a *Docker* container, we provide a hassle-free solution for students, eliminating the need for manual configuration and dependency management. This contribution is significant as it addresses the challenges faced by novice developers when setting up their development environment, enabling them to focus more on learning core concepts and building practical skills in *Flutter* and *Dart*.

For evaluations of the proposal, we requested 24 first-year Master's students in Okayama University, Japan, who have no experience with *Flutter*, to install the *development environment* and modify the three projects in the *Docker* container in the assignments by following the prepared instructions. Their completed assignments were collected via *pCloud*, and their feedback was gathered through a *Google Form*. The results show that all of the students successfully completed the assignments and confirm the efficiency and validity of the proposal.

The rest of this paper is organized as follows: Section 2 introduces related works in the literature. Section 3 introduces adopted open-source software. Section 4 presents the *development environment* for *Flutter*. Section 5 shows the three *Flutter* projects in the assignments. Section 6 evaluates the proposal through applications to novice students. Section 7 concludes this paper with future works.

2. Literature Review

In this section, we provide a comprehensive review of the literature relevant to the topics discussed in our paper. We organize the section with three main themes: interactive learning environments, the *Flutter* framework, and *Docker* technology. Each subsection begins with an introduction to the respective topic, followed by a review of relevant studies.

2.1. Interactive Learning Environments

Interactive learning environments are integral in modern education, offering personalized and efficient platforms for teaching and learning. Here, we review various studies contributing to the understanding and enhancement of interactive learning environments. To identify the relevant literature, we conducted a systematic search across academic databases, focusing on articles published within the last decade. We selected papers based on their relevance to the themes of interactive learning environments, considering factors such as their contribution to theory, methodology, and practical implications.

In [6], Jackson et al. contributed consistent, personalized development environments for educational settings, addressing challenges faced by academia and industry due to inconsistent setups. They explored *Docker* advancements in creating isolated, easily distributable app environments, and the Visual Studio Code's role in customizing these for students' specific learning needs. Overall, the emphasis is on the importance of these practices in improving learning experiences in academic settings.

In [7], Drigas et al. delved into the transformative potential of mobile learning (m-learning) in educations, breaking classroom limitations. They highlighted how diverse content, from podcasts to virtual lessons, empowers learners regardless of their devices. This research examines specific mobile applications within both formal and informal educations, evaluating their usability using the *ISO 9241 part11* [8] standard. Their paper also considers current trends aligning with curriculum guidelines and differentiated instructional theories, emphasizing the evolving landscapes of educational paradigms enabled by mobile technologies.

In [9], Costa et al. explored enhancing student programming educations through robotics, which starts by identifying learning obstacles and potential solutions. A framework, termed "dragon-robot", is proposed and analyzed for its influence on student learning. Based on this analysis, they suggested a solution, showcasing the potential of utilizing robotics to improve comprehension and applications of computer programming concepts among students.

Similarly, in [10], Costa et al. concentrated on introductory-level computer programming and addressed challenges faced by students in this phase. They introduced a solution,

an editor that provides output responses, to facilitate student interactions. They evaluated the framework's impact and identified varying degrees of effectiveness among support tools. Ultimately, it highlighted the efficacy of certain support tools in aiding beginning learners in programming within web-based graphic environments.

In [11], Tung et al. introduced *PLWeb* as a comprehensive exercise management system designed to enhance the teaching and learning of computer programming. It addresses the limitations of existing program submission and assessment systems by focusing on aiding instructors in creating incremental programming exercises. *PLWeb* offers an *integrated development environment (IDE)* serving as both an authoring tool for instructors to design exercises and a user-friendly editor for students to study and submit solutions. Additionally, it features visualized learning status to assist students facing challenges and includes a plagiarism detection tool. Overall, *PLWeb* aims to improve the effectiveness of programming education by providing a versatile platform for instructors to design exercises and for students to engage with programming content.

In [12], Jambalsuren et al. investigated novice programmers' learning performance in a *VLB* programming environment designed to reduce cognitive load and effort. The system utilizes a knowledge tree constructed from student-written source code to track changes and enhance learning performance by minimizing error messages and improving code quality.

In [13], Hamada et al. presented an interactive learning environment designed for teaching information and communication theory and related courses. The environment integrates various modules catering to different learning styles, including a movie-like module and an animated hypertext introductory module explaining fundamental concepts. Additionally, it features a self-assessment module with interactive tests and examinations. The environment can be used as a standalone application or as an applet within any web browser. Evaluation experiments and comparative analyses were conducted to measure its performance in the classroom.

In [14], Kao et al. developed an AR-based learning system for programming education involving 98 fifth graders. Their study divided students into high interactive AR, low interactive AR, and traditional learning groups. The results showed that high interactive AR improved programming achievements, motivation, cognitive load, and technology acceptance. While AR alone did not boost motivation, it led to better programming achievements compared to traditional learning. This study highlighted the importance of highly interactive designs, such as puzzle cards, in enhancing learning outcomes.

In [15], Paiva et al. introduced the *FGPE* project, addressing the need for e-learning tools in programming education, especially during the *COVID-19* pandemic. It outlined *FGPE AuthorKit* and *FGPE PLE*, enabling exercise creation, gamification, and student progress tracking. Positive feedback from educators underscored its value, with ongoing efforts to integrate it into learning management systems and enhance mobile accessibility.

This literature review provides insights into the challenges and advancements in interactive learning environments and offers valuable knowledge for educators and researchers in the field of education technology.

2.2. Flutter

Flutter, as a cross-platform app development framework, has garnered significant attention in recent years. In this subsection, we review studies assessing the performances and capabilities of *Flutter* as compared to other frameworks. We focused on articles and technical reports related to *Flutter* development and comparison studies, based on their relevance to our research questions and methodology. They offer valuable information for developers considering its adoption for cross-platform app development.

In [16], Mahendra et al. aimed to assess the performance of *Android*-based cross-platform app development frameworks using various metrics such as CPU and memory usage, response time, frame rate, and app size. They conducted experiments to compare the frameworks against native *Android* developments. Their findings indicate that native

Android apps perform the best, followed by *Flutter*, a widget-based cross-platform framework, which suggests that while cross-platform frameworks offer efficiency, native apps still lead in overall performances.

In [17], Zahra et al. aimed to compare *Flutter* and *React Native*, specifically, on their automated testing capabilities. They focused on various aspects such as reusability, integration, and compatibility by developing a *To-Do List* app in both frameworks using *Testproject.io* for automated testing. The findings suggest that *React Native* performs better in terms of reusability and compatibility, while both frameworks showed similar performances in integration capabilities. This research provides insights for developers considering cross-platform frameworks, highlighting the strengths and differences in automated testing aspects between *Flutter* and *React Native*.

2.3. Docker

Docker technology revolutionizes application developments and deployments by providing a lightweight, portable containerization solution. In this subsection, we review studies exploring *Docker's* advantages, real-world applications, and usage patterns. We selected studies based on their relevance to our research questions and methodology.

In [18], Rad et al. served as an introductory exploration of *Docker*, highlighting its advantages for developers and administrators. *Docker* is depicted as an open platform facilitating the creation, distribution, and execution of applications via *Docker Engine*. The inclusion of *Docker Hub* as a cloud service for application sharing is emphasized. Furthermore, they emphasized the cost-efficiency of *Docker* in replacing traditional virtual machines with *Docker* containers, reducing expenses associated with rebuilding cloud development platforms. Overall, they outlined the features and benefits of *Docker*, particularly, its potential for cost reductions and streamlined application deployments in the cloud environment.

In [19], Ibrahim et al. investigated the usage patterns of *Docker Compose* in over 4000 open-source *Github* projects, focusing on understanding its real-world applications. They revealed that a significant portion of projects (26.8%) employ *Docker Compose* even for single-component applications, indicating its widespread usage. However, they noted that many projects rarely update their *Docker Compose* files (30% remain unchanged), and most applications utilize basic options rather than advanced ones (only 4.3% utilize security-related options). Despite *Docker Compose* evolving to *version 3*, few projects adopt newer versions, with some even downgrading due to compatibility issues. Their study suggested that while *Docker Compose* is prevalent, there is a tendency among users to stick to basic functionalities and earlier versions, prompting the need for further investigations on improving adoptions of advanced *Docker Compose* features, if necessary.

In [20], Cito et al. addressed the growing need for reproducibility in software engineering research by highlighting the challenges related to replicating scientific results due to undocumented assumptions, dependencies, and configurations in published codes and data. They introduced *Docker* containers as a solution to the issues, emphasizing their roles in aiding the reproducibility of research artifacts. The technical briefing discussed how *Docker* containers can effectively mitigate the challenges, and outlined their applications in software engineering, offering a promising approach to enhance reproducibility in this field.

Our review provides insights into *Docker's* features, benefits, and usage patterns, offering valuable knowledge for developers and researchers in the field of software development and deployment.

3. Adopted Software Tools

In this section, we introduce the *software tools* that are adopted in this paper for completeness and readability.

3.1. 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.

3.2. Dart

Dart [22] is seamlessly integrated with *Flutter*, 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.

3.3. Docker

Docker [23] is a containerization platform, and it has become a pivotal tool in modern software developments, providing a lightweight and efficient solution to package, distribute, and deploy applications across diverse computing environments. By encapsulating applications and their dependencies into containers, *Docker* ensures consistency and reproducibility in software deployments, addressing compatibility issues that often arise across various operating systems and infrastructure setups. This approach enhances application portability, streamlining the development-to-deployment life cycle.

3.3.1. Docker Container

A *Docker container* is a compact, executable software package that contains an application and its dependencies. Operating within *Docker* environments, it ensures consistent deployments across varied computing settings. *Docker containers*, derived from *Docker images*, maintain isolation for uniform application executions, standing as pivotal components in *Docker's* ecosystem. They offer smooth deployments and address compatibility challenges across diverse environments.

3.3.2. Docker Image

A *Docker image* is a template for an application, containing everything needed to run the application, from the codes to the settings. There are two main ways to create a *Docker image*. One is using a **Dockerfile** that describes the set of instructions that specify how to build an image. This file helps create consistent images with the *"docker build"* command. Another involves modifying a **running container** and saving it as a new image using *"docker commit"*.

3.3.3. Docker Compose

A *Docker compose* simplifies the management of multi-container *Docker* applications. Using **YAML** files, it eases the configurations and deployments of interconnected services within a unified application context. By defining complex service architectures, *Docker compose* reduces complexity in managing dependencies and ensures smooth executions of interconnected components in the *Docker* environment. Its functionality simplifies the coordination of containerized services, enabling efficient handling of complex setups by developers.

3.4. GitHub

GitHub [24] is a web-based platform for version control. While it hosts both open-source and private repositories, its core functionality is based on *Git*, an open-source version control system. *GitHub* provides tools for collaborations and enables developers to manage and track changes in their code bases. Users can host and review the codes, manage the projects, and collaborate with others through provided features such as pull requests, issues, and wikis.

3.5. Visual Studio Code

Visual Studio Code (VS Code) [25] is a free source code editor developed by *Microsoft*. *VS Code* supports various programming languages and provides several features such as syntax highlighting, debugging, intelligent code completion, and version control integration. It also offers an extensive library of extensions to enhance the functionality and customize the editor for different development environments.

3.6. pCloud

pCloud [26] is a secure cloud storage service that allows users to store, manage, and share files. It offers a unique URL generation feature, allowing users to gather submissions directly into their *pCloud* accounts without the need for recipients to have their own accounts. This feature simplifies the process of collecting assignments or documents from multiple users. Once uploaded, the files can be easily accessed, managed, and downloaded from the *pCloud* dashboard or the specified folder.

4. Docker-Based Flutter Development Environment

In this section, we present the *Docker-based Flutter development environment* for novice students.

4.1. Overview

The *Docker-based Flutter development environment* is designed and implemented for novice students to start mobile application developments without struggling in complex environment setups. In this environment, they can start the mobile application learning process by modifying the exercise *Flutter* projects included in the environment. Figure 1 shows the overview of the environment.

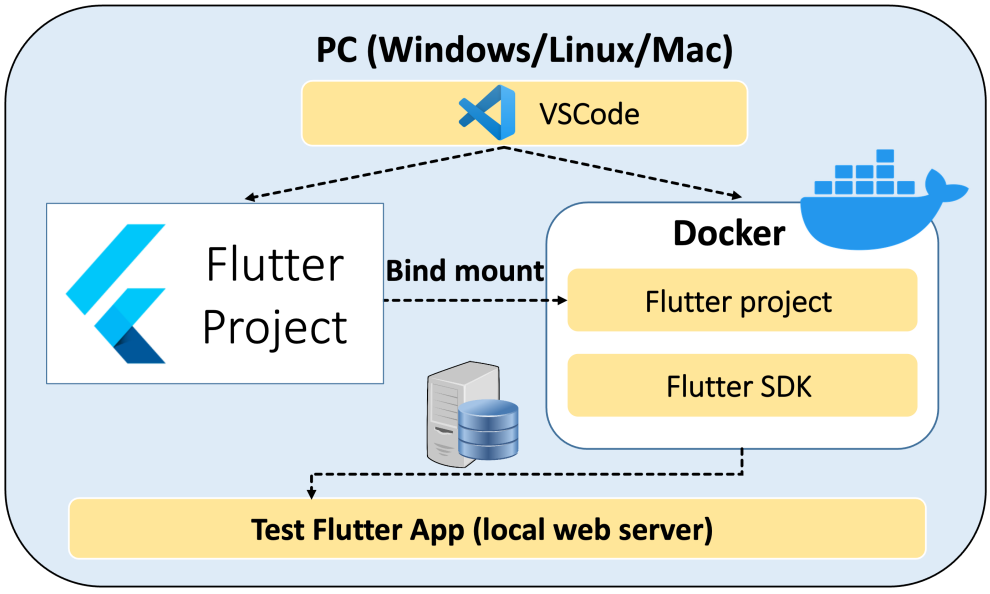

Figure 1. Overview of Docker-based Flutter development environment.

4.2. Procedure for Building Docker Image

In this subsection, we present the procedure of building the *Docker* container image for this environment. Firstly, *Docker* program installation is initiated according to the operating system of the computer. Following this, the appropriate *Ubuntu base Docker image* is selected, compatible with the system architecture (amd64/arm64), using the command *'docker pull'*. Subsequently, container customization commences by initializing the container from the Ubuntu image using the command *'docker run'*. Within the container, essential tools such as *Git* for version control, *Xserver-Xorg* for graphical user interface support, *GTK-3-dev* for GUI development, and *Curl* for file transfer are installed. The *Flutter SDK* is then cloned from the official *Git* repository, ensuring the correct version and setting up necessary environment variables. Finally, the configured container is saved as the *Docker image*, named ***flutter_docker_image:v1.1***, using the command '*docker commit*'.

4.3. Creating System Startup File

In this subsection, we discuss the process of creating a system startup file for the *Flutter environment Docker container image*, which is pushed to our laboratory's *Docker Hub*. To utilize this image, two essential files are required, according to different operating systems—*Windows*, *Linux (Ubuntu)*, and *Mac*—which we have prepared and pushed to the *GitHub* repository.

1. **docker-compose.yml:** This manages the configurations of services and containers essential for an application, especially beneficial in complex development setups involving multiple services. Listing A1, provided in the Appendix A, comprises sets of code utilized to define and manage the *Docker* application in this proposal. It is structured as follows: Initially, the *Docker Compose* version (`version:"3"`) is specified for compatibility with the *Docker* engine. Next, a service named (`flutter:`) is defined to organize and manage services within the *Docker Compose* environment. Then, the *Docker* image (`image:flutter_docker_image: v1.1`), which is generated in Section 4.2, containing the *Flutter* runtime environment, is described for use by the *flutter* service. Following that, environment variables are set, and a volume (`volumes://c/flutter_workspace:/root/workspace`) from the host machine is mapped to the directory within the container. Finally, an extra hostname and its corresponding IP address (`extra_hosts:flutter:127.0.1.1`) are defined for the container to ensure proper hostname resolution within the *Docker* environment.
2. **devcontainer.json:** This manages *Visual Studio Code's Remote Development* extension, aiming to streamline *Flutter* app development within a consistent and isolated environment directly from within the *VSCode* editor. Listing A2, consisting of sets of code provided in the Appendix A, is structured as follows: Firstly, the development container name (`"name":"Flut ter Docker"`) is specified to provide a descriptive identifier. Then, the path to the *Docker Compose* file (`"dockerComposeFile":["../docker-compose.yml"]`) is defined for configuring the development container. Next, the service (`"service":"flutter"`) defined in the *Docker Compose* file is specified to ensure the correct service is utilized. After that, the user to be used (`"remoteUser":"root"`) is set to ensure proper permissions and access control within the container when connecting remotely to the development container. Later, various settings for *Dart* and *Flutter* development (`"settings":`) are specified to ensure proper configuration, and the necessary extension to be installed within *VSCode* is also specified to enhance the development experience within the IDE. Ultimately, the workspace directory on the host machine and development container (`"workspaceMount":"source = ${localWorkspace}/workspace, target=/root/workspace"`) are mounted to provide a consistent location for project files and directories.

5. Three Flutter Projects for Programming Learning Startup

In this section, we present three *Flutter* projects for students to start learning programming using the installed development environment. They are embedded in the *Docker container* as code modification exercises.

5.1. Three Flutter Projects for Exercises

The three *Flutter* projects are prepared for exercises by considering the suitability of novice students who are beginners of *Dart* programming to create *Flutter* mobile applications. Through solving the exercises, they can learn use of *Container*, *ListView*, and *AlertDialog* in *Flutter*.

5.1.1. **Exercise-1**: Container

Exercise-1 asks to modify the size and the color of the container widget in the interface, as shown in Figure 2. The objective, the task description, and the expected learning outcome for this exercise are described as follows:

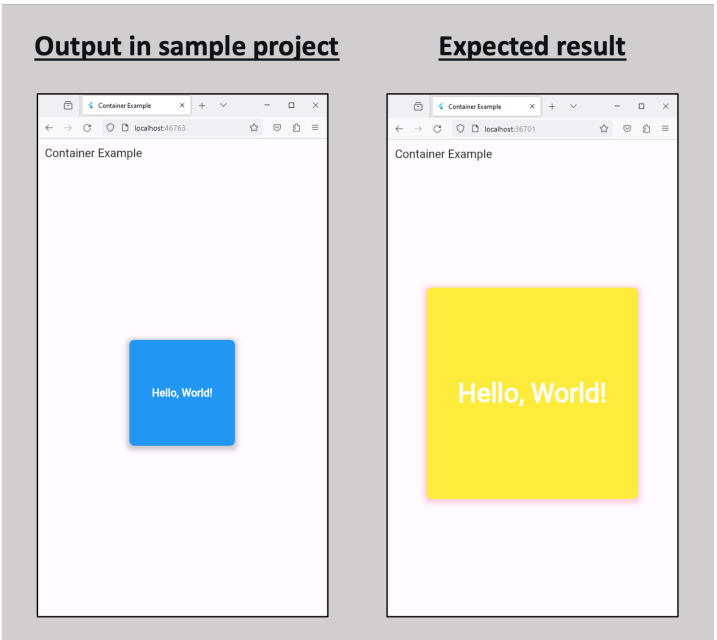

Figure 2. *Container* project in **Exercise-1**.

- **Objective:** This exercise focuses on understanding and modifying the container widget in *Flutter* as a fundamental UI element used to encapsulate other widgets.
- **Task description:** Tasks involve adjusting the size, color, shade, and text within the container widget, highlighting the customization option for visual properties.
- **Expected learning outcome:** Students gain proficiency in customizing the container widget, understanding how to manipulate the size, color, shadow, and text properties, and laying the groundwork for the UI customization in *Flutter*.

5.1.2. **Exercise-2**: ListView

Exercise-2 asks to reverse the order of the widgets in the interface, as shown in Figure 3.

- **Objective:** This exercise focuses on *ListView* as the *Flutter* widget used for displaying scrollable lists of widgets and challenges students to manipulate its functionality.
- **Task description:** Tasks focus on reordering the elements and altering the indicators in *ListView*.
- **Expected learning outcome:** Students develop proficiency in managing the *ListView* functionality for data order and visual indicators, enhancing their skills in handling scrollable lists in *Flutter* apps.

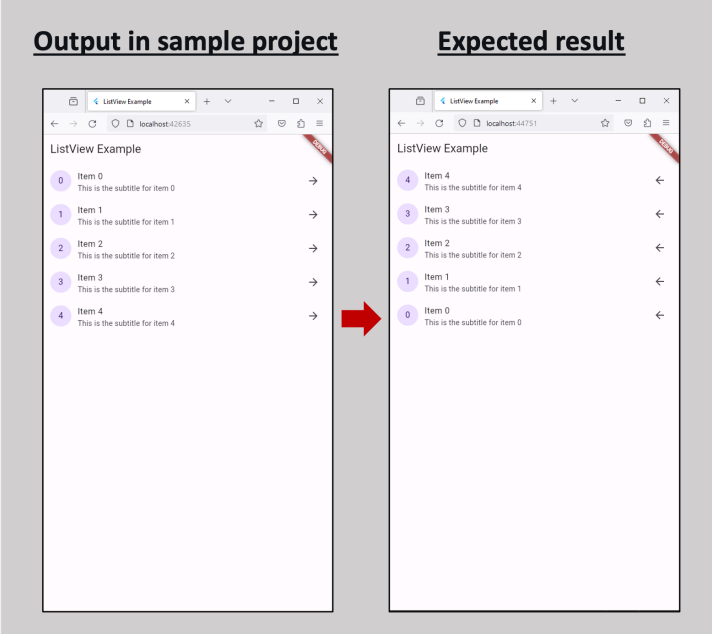

Figure 3. *ListView* project in **Exercise-2**.

5.1.3. **Exercise-3**: AlertDialog

Exercise-3 asks to change the alert message in the interface, as shown in Figure 4.

- **Objective:** This exercise focuses on the *AlertDialog* widget that is used to display the critical information or interact with users.
- **Task description:** Tasks revolve around changing the icon color, button style, and dynamic text message in *AlertDialog*, which emphasizes customizations in dialog box interactions.
- **Expected learning outcome:** Students acquire proficiency in utilizing *AlertDialog*, and understanding of how to customize the icon color, button style, and dynamic content presentation within a dialog box in *Flutter*.

5.2. Modification Guidance for Exercises

To help students complete the projects in the exercises, we provide the modification guidance to each exercise. Table 1 lists the essential items as the guidance for successful completions of the three exercises.

Table 1. Modification guidance.

Exercises		Modification Guidance
Exercise-1	Container	- box size (400 × 400) - box color (yellow) - box shade (pink) - box text (50)
Exercise-2	ListView	- show list in descending order - modify arrow direction
Exercise-3	AlertDialog	- icon color (red) - button style (outlined) - button text

5.3. Development Environment Use for Exercises

To start the *Flutter* project development environment in the *Docker* container, students need to perform the following procedure, where the instruction is provided on *GitHub*:

1. Install *Docker* and *VSCode* according to the operating system of the target PC.

2. Import the three extensions for *Flutter*, *Remote Development*, and *Docker* in *VSCode*.
3. Obtain the *Docker* container image for the *Flutter* development environment, using the command "*docker pull*".
4. Download the *GitHub* project that contains the essential files, *(docker-compose.yml/ devcontainer.json)*, or, alternatively, clone the project using the command *"git clone"* if students already installed *Git* in the PCs.
5. Open the downloaded project in *VSCode*, initiate the containerized development to access the remote development, and activate the *Flutter* development environment in the *Docker* container.

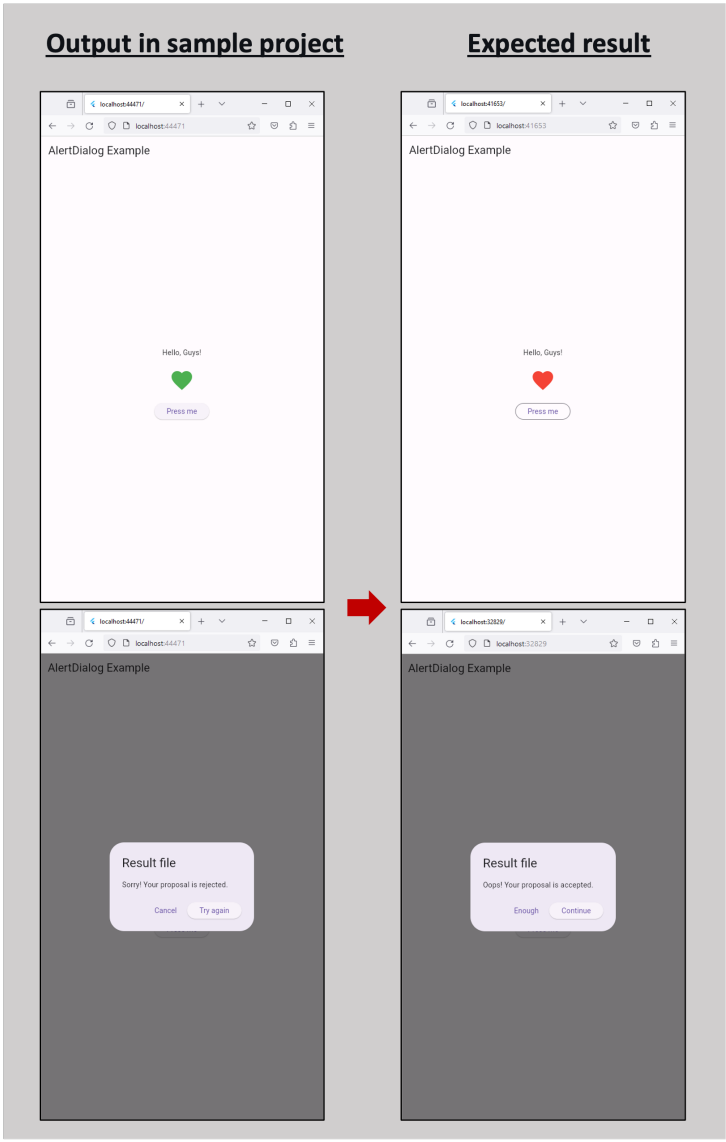

Figure 4. *AlertDialog* project in **Exercise-3**.

5.4. Access to Exercises

After connecting to the *Flutter* development environment in the *Docker* container, students can start solving the three exercises included in the container using the following steps:

1. Use the command *"ls -al /root"* in the *Docker* container to view the three exercises.
2. Transfer each exercise to the designated workspace in the container using the command *"scp -r /root/exercise1 /root/workspace/"*.

3. Access the exercise directory using the command *"cd /root/workspace/exercise1"*. This allows students to navigate to the specific exercise folder, where they can modify the source code according to the provided modification guidance.
4. Initiate the *Flutter* web server by executing the command *"flutter run -d web-server"*.
5. Navigate to the local web server address with *"http://localhost:port"* in the web browser to preview the output generated by the source code.

Upon completing the modifications for each exercise, students are required to submit the modified code files via the provided URL for *pCloud*.

6. Evaluation

In this section, we evaluate the efficiency and validity of the proposed *Docker-based Flutter development environment* through applications to novice students.

6.1. Evaluation Setup

For the installation, we prepared the *GitHub* document that explains how to install and use it. It includes the instructions for installing *Docker* and *VSCode*, downloading the *development environment Docker image* from *Docker Hub*, connecting the container, modifying the projects in the exercises, and submitting the answer files to *pCloud*. Each part in the document was explained with texts and images to improve the readability. A separate PDF file was made for each operating system of *Windows*, *Linux*, and *Mac*.

Then, we asked 24 first-year Master's students in the information and communication department at Okayama University, Japan, who had no experience with *Flutter*, to set up the development environment and solve the exercises independently. It was worth noting that these students came from a diverse range of undergraduate courses in Okayama University, Tokushima University, and universities in China.

All of the 24 students had no prior experience with *Flutter*, but they did possess foundational knowledge in programming languages, such as *C*, *C++*, *Java*, and *Python*, acquired through coursework and practical experiences related to their theses. Therefore, the focus of our evaluation was to observe how students with existing programming skills in these languages adapted to learning mobile application development using *Flutter*. Throughout the evaluation process, one teaching assistant provided guidance and support to ensure a conducive learning environment.

Next, we collected the data from students after the course. A teaching assistant introduced the course, and the students could set up the development environment either in the classroom or at home. They were tasked with solving the exercises independently and submitting the correct modification code answer files for three exercises via a *pCloud* URL. Finally, the students filled in the questionnaire for the system feedback provided via *Google Form*.

By including students from diverse academic backgrounds, we aimed to capture a comprehensive understanding of how individuals with programming experience in traditional languages transitioned to *Flutter* development. This approach allowed us to assess the effectiveness of our instructional materials and the feasibility of integrating *Flutter* into the curriculum for students with varying levels of programming expertise.

6.2. Experiences of Students

First, we investigate the students' experiences that are related to the proposed *Flutter* development environment using the *Docker* container and *VSCode*. Table 2 shows the four questions to the students and their answers from the students that were collected through *Google Forms*.

The results show that many students are familiar with *GitHub* and *VSCode*. This suggests that the utilization of the proposed environment may not pose significant challenges for them. Conversely, *fewer* students are acquainted with the *mobile app* and *Docker*. This development-environment-setup exercise becomes a valuable opportunity for students to know the important tools for software development.

Table 2. Questions and answers on experiences related to proposal.

No.	User Experience-Related Questions	# of Students	
		Yes	No
Q1	Do you have any experience with *mobile app development*?	10	14
Q2	Are you familiar with *GitHub*?	16	8
Q3	Are you familiar with *Docker*?	10	14
Q4	Are you familiar with *VSCode*?	21	3

6.3. Results of Three Exercises

Next, we present the analysis of the results obtained from students' performance on the exercises detailed in (Section 5.1). As depicted in Table 3, all of the students successfully completed the three exercises. Consequently, it can be inferred that our proposal is valuable for novice students, as it mitigates the complexity of environment setups, allowing them to concentrate on *Flutter* application developments.

Table 3. Results of three exercises by students.

Total # of Students	Correctness		
	Exercise-1	Exercise-2	Exercise-3
24	100%	100%	100%

6.4. Usability Evaluation

Finally, we investigate the usability of the proposal by the students through the *System Usability Scale (SUS)* [27]. The choice of the *SUS* for evaluating our interactive learning environment was motivated by its widespread use, comprehensive assessment capabilities, and robustness in capturing user feedback. Additionally, we focus on measuring the level of usability, and the *SUS* offers a broader scope, capturing users' overall perceptions of usability through a standardized questionnaire, while other simpler methods, such as the *Usability Metric for User Experience (UMUX)*, focus specifically on user experience metrics related to usability. Table 4 shows the questions used to evaluate the usability as well as the answers from the students.

Table 4. Questions and answers on system usability.

No.	System Usability Scale Questions	# of Students				
		Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Q1	I think that I would like to use this system frequently.	0	3	10	7	4
Q2	I found the system unnecessarily complex.	3	5	6	9	1
Q3	I thought the system was easy to use.	0	3	8	10	3
Q4	I think that I would need the support of a technical person to be able to use this system.	3	3	7	7	4
Q5	I found the various steps in this system were well integrated.	0	3	7	7	7
Q6	I thought there was too much inconsistency in this system.	7	7	6	4	0
Q7	I would imagine that most people would learn to use this system very quickly.	0	3	7	10	4
Q8	I found the system very cumbersome to use.	3	10	7	4	0
Q9	I felt very confident using the system.	0	2	10	10	2
Q10	I needed to learn a lot of things before I could get going with this system.	4	4	6	9	1

Table 5 shows the grade for the comprehensive assessment of the system usability with the *System Usability Scale (SUS)* score between 0 and 100. Among 24 students, two students' scores (8.33%) are *Best Imaginable (Grade A)*, three students' scores (12.50%) are *Excellent (Grade B)*, one student's score (4.17%) is *Good (Grade C)*, twelve students' scores (50%) are *OK (Grade D)*, and six students' scores (25%) are *Poor (Grade E)*. No students (0%) gave the *Worst Imaginable (Grade F)*. The results indicate that 75% of the 24 students are satisfied with the proposed system in the usability.

Table 5. Grade associated with SUS score.

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

6.5. Analysis of Students' Prior Experience with SUS Scores

We conducted a correlation analysis to investigate the relationship between students' prior experience with mobile app development, *GitHub*, *Docker*, *VSCode*, and their *SUS* scores. The analysis results, as depicted in Figure 5 for experienced students and Figure 6 for non-experienced students, revealed significant correlations. Correlation Analysis 1 and 2 represent analysis results upon experienced students described in Figure 5, while the remaining Correlation Analysis 3, 4, and 5 represent analysis results upon non-experienced students described in Figure 6. These correlations shed light on the effectiveness of our proposal and underscore the importance of tailored support for students with varying levels of experience in development tools.

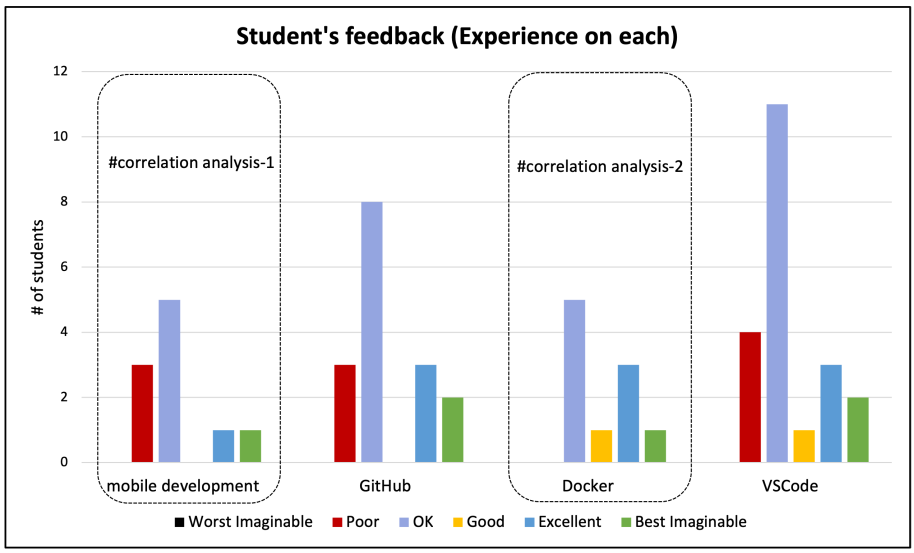

Figure 5. Analysis of Experienced students' feedback.

- **Correlation Analysis-1:** Among experienced students in mobile development, 3 students rated it as *'Poor'*, while 7 students rated it as *'OK'*, *'Excellent'*, or *'Best Imaginable'*. This indicates a generally positive perception of our proposal, but it is also recognized that further improvements could enhance its effectiveness.
- **Correlation Analysis-2:** Interestingly, there was no negative perception of our proposal among experienced students with *Docker*. This suggests that our proposal could be well received and effective for them due to the simplification of environment setup facilitated by *Docker*.

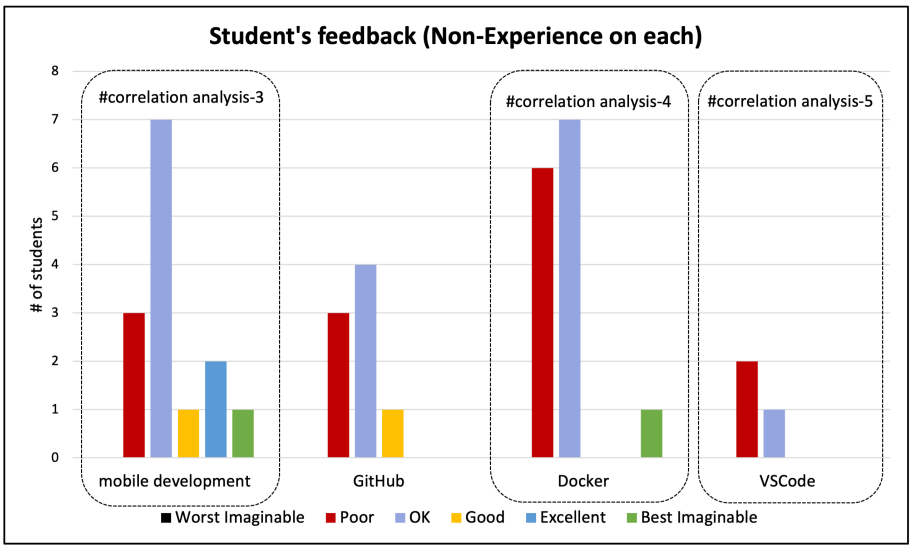

Figure 6. Analysis of Non-Experienced students' feedback.

- **Correlation Analysis-3:** Non-experienced students in mobile development provided mixed ratings for usability. While the majority rated it as *'OK'*, indicating moderate usability, three students rated it as *'Poor'*, suggesting a need for further simplification. Conversely, four students rated it as *'Good'*, *'Excellent'*, or *'Best Imaginable'*. Overall, our proposal could be beneficial for mobile application development.
- **Correlation Analysis-4:** Non-experienced students generally rated *Docker's* usability lower compared to other tools. The majority rated it as *'Poor'*, suggesting that they may have faced difficulties in setting up and using *Docker* for development purposes. This highlights the importance of providing additional support and resources for students who are new to containerization technology.
- **Correlation Analysis-5:** The majority of non-experienced students rated *VSCode's* usability as *'OK'*, indicating a moderate level of comfort with the development environment. However, there were also some *'Poor'* ratings, suggesting that some students may have found it challenging to use *VSCode* effectively for development tasks.

6.6. Findings

First, we summarize the findings from our experiments and implications to reflect on the proposed *Docker-based Flutter development environment*.

6.6.1. Efficiency of Proposed Environment

The proposed environment effectively mitigated the complexities associated with the environment setup for novice students. By encapsulating all the necessary tools and dependencies within a *Docker* container, students were able to start mobile application developments without struggling with intricate configurations. The 100% success rate of completing the three exercises indicates the efficiency of our approach in facilitating learning and development.

6.6.2. Usability and User Experience

The usability evaluation using the *SUS* score provided insights into overall user experiences. While the majority of students expressed positive perceptions regarding the system use, integrations, and learnability, there were some concerns raised about the complexity and inconsistency within the system. The areas for improvement were highlighted based on feedback received from three students. Additionally, specific challenges were reported by two students during experiments.

Firstly, one student encountered an error upon mounting a file from the host into the *Docker* container using *Windows OS*. *Docker* attempted to mount the ".gitconfig" file from

the host directory into the container directory but encountered the issue as it expected the source path to be a directory rather than a file. To solve this, we verified the existence of the ".gitconfig" file in the specified host directory and ensured that *Docker* had the proper permission to access it. Additionally, we double-checked the correctness of the specified path and the file permission. By addressing these, we solved the mounting issue during the *Docker* container initialization.

Next, another student encountered an error stating "Fatal Error: com.docker.backend cannot start" while using *Docker Desktop* on *Mac OS (M1 chip)*. We solved the issue by reinstalling it to ensure the fresh installation. We also checked resource conflicts or permissions issues on the system. Additionally, we ensured that *Docker Desktop* was updated and compatible with the hardware. With these steps, we successfully solved the error and enabled the student to use *Docker Desktop* without further issues.

These experiences and feedback from the students highlighted that the proposal should be improved by providing clearer explanations for novice students for ensuring smoother setup processes.

6.6.3. Learning Outcomes

The successful completion of the exercises by all the students suggests that the proposed environment effectively facilitated learning outcomes. By providing structured exercises and modification guidance, the students gained proficiency in modifying *Flutter projects* while understanding fundamental concepts, such as *Container*, *ListView*, and *AlertDialog*. This indicates the effectiveness of the hands-on learning approach facilitated by the *Docker*-based environment.

6.7. Limitations

Second, we address the limitations of this study, covering the three key areas: familiarity with tools, usability challenges, and generalizability.

6.7.1. Familiarity with Tools

While the majority of students were familiar with the adopted software tools such as *GitHub* and *VSCode*, fewer students had prior experiences with *Docker* and *mobile app development*. This limitation could affect the generalizability of our findings, since the effectiveness of the proposed environment might vary based on students' prior knowledge and familiarity with the tools.

6.7.2. Usability Challenges

The *SUS* evaluation highlighted the areas of concerns, regarding the system complexity and inconsistency. The usability challenges could impact overall user experiences and hinder students' productivity. Addressing these issues through refinements of user interfaces and guidances could improve the usability of the proposed environment.

6.7.3. Generalizability

This study involved a specific group of first-year Master's students at Okayama University, Japan, which might limit the generalization of the findings for broader student populations. Future studies could involve diverse student groups to assess the applicability of the *Docker*-based environment across different educational contexts and backgrounds.

In summary, the *Docker-based Flutter development environment* demonstrated efficiency in facilitating mobile application learning for novice students. While the findings indicate positive outcomes, addressing limitations such as usability challenges and variability in the tool familiarity could enhance the overall effectiveness and usability of the environment. Further researches and refinements will be needed to optimize the environment for broader educational contexts.

7. Discussion and Conclusions

This paper presented the *Docker-based Flutter development environment* with *Visual Studio Code* and three simple exercise projects for modifying source codes for starting to learn *Flutter* programming by novice students.

For evaluations, the proposal was assigned to 24 Master's students of Technical Engineering course at Okayama University, Japan, including the installation. Then, all the students successfully completed the exercises, which confirmed the efficiency and validity of our proposal. Furthermore, for the questionnaire on the usability, the majority of the students provided positive answers indicating moderate to high usability.

Our proposal not only addresses the educational needs of teaching mobile app development but also aligns with the principles of next-generation programming educations. By leveraging *Docker* to streamline setup processes and promote modern software development practices, we aim to equip students with both technical skills and an understanding of contemporary development tools. Moreover, our approach highlights *Flutter's* cutting-edge capabilities for cross-platform app developments, including native-like experiences and rapid iteration with hot reload. By introducing students to *Flutter*, we equip them with skills crucial to the dynamic software industry.

Our implementation carefully considers course requirements, security considerations, and the diverse operating systems prevalent in next-generation programming education. While our focus has been on practical aspects, we recognize the importance of continuously refining and adapting the environment to evolving educational needs and technological advancements. Furthermore, our approach highlights the versatility of *Flutter* and *Dart* in mobile and web app developments, as well as the collaborative learning experiences facilitated by the *Docker* container environment. By integrating sample projects and guided exercises, we foster collaborative learning environments where students can share insights and collectively enhance their skills.

While our study has made significant strides in addressing the educational challenges of teaching *Flutter* and *Dart*, we acknowledge certain limitations. Future research should explore potential challenges faced during the implementation and evaluation, as well as opportunities for further refinement of the learning environment.

In future works, we plan to expand the repertoire of exercises for *Flutter* mobile applications and introduce an automatic answer-checking feature. We will also distribute the proposal to students in various universities to improve our *Docker-based Flutter development environment*, making it more effective and user-friendly for programming learners worldwide. Alongside this, we will analyze how our environment compares to traditional teaching methods and explore new ways to incorporate gamification elements to boost student engagement and motivation. These efforts aim to continuously enhance our *Docker-based Flutter* development environment, ensuring a comprehensive and enjoyable learning experience for programming students globally.

Author Contributions: Conceptualization, S.T.A. and N.F.; methodology, S.T.A.; software, S.T.A., L.H.A., S.A.K., M.M. and K.H.W.; investigation, S.T.A. and L.H.A.; Visualization, S.T.A., L.H.A., S.A.K., M.M. and K.H.W.; writing—original draft preparation, S.T.A.; writing—review and editing, S.T.A. and N.F; supervision, N.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable

Informed Consent Statement: Not applicable

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

1. This file in **Listing** A1 is used to define and manage the *Docker* application. It manages the configurations of the services and containers needed for an application and is commonly used when interacting with complex development setups involving multiple services.

Listing A1. docker-compose.yml

```
version: ''3''
services:
  flutter:
    image: soethandara/plas_flutter_docker:v1.1
    environment:
      - HOME=/root
      - no_proxy=127.0.0.1,localhost
    volumes:
      - ~/.gitconfig:/home/root/.gitconfig
      - .:/root/workspace
      - //c/flutter_workspace:/root/workspace
    network_mode: ''host''
    extra_hosts:
      - flutter:127.0.1.1
```

2. This file in **Listing** A2 is used with *Visual Studio Code* with the *Remote–Containers* extension. It defines the development container configurations, including the settings for the container environment, extensions, and development tools.

Listing A2. devcontainer.json

```
{
    ''name'': ''Flutter Docker'',
    ''dockerComposeFile'': [''../docker-compose.yml''],
    ''service'': ''flutter'',
    ''remoteUser'': ''root'',
    ''remoteEnv'': {
      ''DISPLAY'': ''${localEnv:DISPLAY}'',
      ''PATH'': ''/usr/local/sbin:/usr/local/bin:
      /usr/sbin:/usr/bin:/sbin:/bin:/opt/flutter/bin''
    },
    ''settings'': {
      ''dart.flutterCreateAndroidX'': true,
      ''dart.flutterCreateIOSLanguage'': ''swift'',
      ''dart.flutterTestLogFile'': ''test_results.xml'',
      ''dart.flutterTestLogBufferSize'': 8192000,
      ''dart.flutterTestLogFileVerbose'': true,
      ''dart.sdkPath'': ''/opt/flutter/bin/cache/dart-sdk'',
      ''dart.flutterPath'': ''/opt/flutter/bin/flutter'',
      ''terminal.integrated.shell.windows'': ''C:\\Windows\\System32\\
        cmd.exe'',
      ''dart.devToolsLogFile'': ''devtools.log''
    },
    ''runArgs'': [''--privileged'', ''-P''],
    ''extensions'': [''dart-code.flutter''],
    ''workspaceMount'': ''source=${localWorkspaceFolder}/workspace,
  target=/root/workspace,type=bind,consistency=delegated,
    ''workspaceFolder'': ''/root/workspace''
}
```

References

1. Force, C.T. *Computing Curricula 2020: Paradigms for Global Computing Education 2020*; Association for Computing Machinery: New York, NY, USA, 2021. [CrossRef]
2. Rushby, N. Editorial: An agenda for mobile learning. *Brit. J. Edu. Technol.* **2012**, *43*, 355–356. [CrossRef]

3. Kukulska-Hulme, A.; Traxler, J. Learning Design with Mobile and Wireless Technologies. In *Rethinking Pedagogy for the Digital Age: Designing and Delivering e-Learning*; Beetham, H., Sharpe, R., Eds.; Routledge: London, UK, 2017; pp. 180–192.
4. Hsu, Y.-C.; Ching, Y.-H.; Snelson, C. Research priorities in mobile learning: An international Delphi study. *Can. J. Learn. Technol.* **2014**, *40*, 1–22. [CrossRef]
5. Koole, M.L. A model for framing mobile learning. In *Mobile Learning: Transforming the Delivery of Education and Training*; Ally, M., Ed.; AU Press: Edmonton, AB, Canada, 2009; pp. 25–47.
6. Jackson, S.; Wurst, K.R. Teaching with VS code DevContainers: Conference workshop. *J. Comput. Sci. Coll.* **2022**, *37*, 81–82.
7. Drigas, A.; Angelidakis, P. Mobile applications within education: An overview of application paradigms in specific categories. *Int. J. Interact. Mob. Technol.* **2017**, *11*, 17–29. [CrossRef]
8. *ISO 9241 Part11*; ISO 9241-11: Ergonomics of Human-System Interaction—Part 11: Usability: Definitions and Concepts. International Organization for Standardization: Geneva, Switzerland, 2018.
9. Costa, C.J.; Aparicio, M.; Cordeiro, C. Web-based graphic environment to support programming in the beginning learning process. In Proceedings of the Entertainment Computing (ICEC), Lecture Notes in Computer Science, Heidelberg, Germany, 26–29 September 2012; pp. 413–416. [CrossRef]
10. Costa, C.J.; Aparicio, M.; Cordeiro, C. A solution to support student learning of programming. In Proceedings of the Workshop on Open Source and Design of Communication (OSDOC '12), Lisboa, Portugal, 11 June 2012; pp. 25–29. [CrossRef]
11. Tung, S.H.; Lin, T.T.; Lin, Y.H. An exercise management system for teaching programming. *J. Softw.* **2013**, *8*, 1718–1725. [CrossRef]
12. Jambalsuren, M.; Cheng, Z. An Interactive Programming Environment for Enhancing Learning Performance. In *Databases in Networked Information Systems, DNIS 2002*; Lecture Notes in Computer Science; Bhalla, S., Ed.; Springer: Berlin/Heidelberg, Germany, 2002; Volume 2544, pp. 201–212. [CrossRef]
13. Hamada, M.; Hassan, M. An interactive learning environment for information and communication theory. *EURASIA J. Math. Sci. Tech. Ed.* **2017**, *13*, 35–59. [CrossRef]
14. Kao, G.Y.-M.; Ruan, C.-A. Designing and evaluating a high interactive augmented reality system for programming learning. *Comp. Hum. Beh.* **2022**, *132*, 107245. [CrossRef]
15. Paiva, J.C.; Queirós, R.; Leal, J.P.; Swacha, J.; Miernik, F. Managing Gamified Programming Courses with the FGPE Platform. *Information* **2022**, *13*, 45. [CrossRef]
16. Mahendra, M.; Anggorojati, B. Evaluating the performance of Android-based cross-platform App development frameworks. In Proceedings of the 6th International Conference on Communication and Information Processing (ICCIP '20), Tokyo, Japan, 27–29 November 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 32–37. [CrossRef]
17. Zahra, H.A.; Zein, S. A systematic comparison between Flutter and React Native from automation testing perspective. In Proceedings of the International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), Ankara, Turkey, 20–22 October 2022; pp. 6–12.[CrossRef]
18. Rad, B.B.; Bhatti, H.J.; Ahmadi, M. An introduction to Docker and analysis of its performance. *IJCSNS Int. J. Comp. Sci. Net. Secu.* **2017**, *17*, 228–235.
19. Ibrahim, M.H.; Sayagh, M.; Hassan, A.E. A study of how Docker Compose is used to compose multi-component systems. *Empir. Soft. Eng.* **2021**, *26*, 128. [CrossRef]
20. Cito, J.; Gall, H.C. Using Docker containers to improve reproducibility in software engineering research. In Proceedings of the IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), Austin, TX, USA, 14–22 May 2016; pp. 906–907. [CrossRef]
21. Flutter. Available online: https://docs.flutter.dev/ (accessed on 26 February 2024).
22. Dart. Available online: https://dart.dev/overview/ (accessed on 26 February 2024).
23. Docker. Available online: https://docs.docker.com/get-started/overview/ (accessed on 26 February 2024).
24. GitHub. Available online: https://docs.github.com/en (accessed on 26 February 2024).
25. Visual Studio Code. Available online: https://code.visualstudio.com/docs (accessed on 26 February 2024).
26. pCloud. Available online: https://docs.pcloud.com/ (accessed on 26 February 2024).
27. System Usability Scale (SUS). Avail-able online: https://credoagency.co.uk/usability-in-cro-the-system-usability-scale-sus/ (accessed on 26 February 2024).