information

Article

An Independent Learning System for Flutter Cross-Platform Mobile Programming with Code Modification Problems

Safira Adine Kinari [1], Nobuo Funabiki [1,*], Soe Thandar Aung [1], Khaing Hsu Wai [1], Mustika Mentari [1] and Pradini Puspitaningayu [2]

[1] Graduate School of Environmental, Life, Natural Science and Technology, Okayama University, Okayama 700-8530, Japan; safiraak@s.okayama-u.ac.jp (S.A.K.); soethandar@s.okayama-u.ac.jp (S.T.A.); pjsu9uam@s.okayama-u.ac.jp (K.H.W.); pqt85hm5@s.okayama-u.ac.jp (M.M.)
[2] Department of Electrical Engineering, Universitas Negeri Surabaya, Surabaya 60231, Indonesia; pradinip@unesa.ac.id
* Correspondence: funabiki@okayama-u.ac.jp

Abstract: Nowadays, with the common use of smartphones in daily lives, mobile applications have become popular around the world, which will lead to a rise in *Flutter* framework. Developed by *Google*, *Flutter* with *Dart* programming provides a *cross-platform development environment* to create visually appealing and responsive user interfaces across mobile, web, and desktop platforms using a single codebase. However, due to time and staff limitations, the *Flutter/Dart* programming course is not included in curricula, even in IT departments in universities. Therefore, independent learning environments for students are essential to meet this growing popularity. Previously, we have developed *programming learning assistant system (PLAS)* as a web-browser-based self-learning platform for novice students. *PLAS* offers various types of exercise problems designed to cultivate programming skills step-by-step through a lot of *code reading* and *code writing* practices. Among them, one particular type is the *code modification problem (CMP)*, which asks to modify the given source code to satisfy the new specifications. *CMP* is expected to be solved by novices with little effort if they have knowledge of other programming languages. Thus, *PLAS with CMP* will be an excellent platform for independent learning. In this paper, we present *PLAS with CMP* for the independent learning of *Flutter/Dart programming*. To improve the readability of the source code by students, we provided rich comments on grammar or behaviors. Besides, the code can be downloaded so that students can check and run it on an IDE. For evaluations, we generated 38 CMP instances for basic and multimedia/storage topics in *Flutter/Dart* programming and assigned them to 21 master students at Okayama University, Japan, who have never studied it. The results confirm the validity of the proposal.

Keywords: Flutter; Dart; cross-platform; programming; code modification problem; PLAS; independent learning

Citation: Kinari, S.A.; Funabiki, N.; Aung, S.T.; Wai, K.H.; Mentari, M.; Puspitaningayu, P. An Independent Learning System for Flutter Cross-Platform Mobile Programming with Code Modification Problems. *Information* **2024**, *15*, 614. https://doi.org/10.3390/info15100614

Academic Editor: Aneta Poniszewska-Maranda

Received: 26 July 2024
Revised: 20 September 2024
Accepted: 5 October 2024
Published: 7 October 2024

1. Introduction

The use of mobile devices has become an integral part of daily life for a lot of people. In 2023, over 4.2 billion people worldwide used smartphones. Projections from *Statista* suggest that this number will continue rising so that over 6.3 billion people will use smartphones in the next five years [1].

As smartphone adoption grows, so does the demand for innovative and high-quality mobile applications. Based on *data.ai*, users worldwide downloaded over 255 billion mobile applications in 2023, demonstrating a strong interest in installing mobile applications that offer added value to smartphones [2]. This surge in application downloads has reflected the significant role of mobile applications that are relied upon for various purposes, including entertainment, communication, productivity, and shopping.

Traditionally, developers build applications using *Java* and *Kotlin* for *Android*, and *Swift* and *Objective-C* for *iOS*. These languages stand to be popular due to the fact that a lot of

developers have been acquainted with these platforms [3]. Native development allows developers to fully utilize the exclusive capabilities and features of each operating system. However, one major drawback is developers need to build and manage different source codes depending on platforms even for the same applications.

To address this challenge, *cross-platform application development* [4] has emerged as a way to create applications for developers to write code once and deploy it on multiple platforms. This approach saves time, effort, and costs while reaching wider audiences. *Cross-platform application development* has gained great popularity in recent years.

Among the most prominent frameworks for *cross-platform application development* is *Flutter*, which has been developed by *Google*. *Flutter* is an open-source UI development toolkit that enables developers to build natively compiled applications for mobile, including *Android* and *iOS*, web, and desktop platforms from a single codebase. It provides a rich set of *prebuilt widgets* and tools that help developers create attractive, responsive, and flexible user interfaces [5].

The following advantages can be observed in learning and using the *cross-platform application development* with *Flutter*:

1. A single source code using one language, *Dart*, can run on various platforms.
2. An attractive, responsive, and flexible user interface can be easily made using *prebuilt widgets* and tools.
3. Real-time code changes become possible by the *hot-reload* feature.
4. Comprehensive documentation and a supportive community make *Flutter* particularly suitable for self-learners, enabling individuals to independently tackle and resolve code modification problems.

As a result, *Flutter* has become favored by many well-known companies due to its numerous advantages, and *Flutter* has been adopted at well-known companies such as *Cryptograph*, *Postmuse*, *Hamilton*, *Apptree*, and *Google* [6].

However, this rapid growth in the popularity of *Flutter* has not been accompanied by a proportional increase in classroom instruction and lectures in universities. Many universities still lack sufficient resources in terms of teachers and/or time to deliver comprehensive lectures on *Flutter* development. As a result, there is a growing need for independent learning environments for new developers and students to learn *Flutter* with *Dart* programming on their own.

In response to this gap, we previously developed a web-based *Programming Learning Assistant System* (PLAS) for self-study of programming languages through practicing *code reading* and *code writing*. Currently, *PLAS* covers familiar programming languages such as *C*, *C++*, *Java*, *JavaScript*, *Python*, *Scratch*, and *Verilog*. *PLAS* provides various types of programming exercises to enhance the programming learning experience.

Building on this foundation, we have also studied the use of *Grammar Understanding Problem (GUP)* for learning *cross-platform application development* with *Flutter* in *PLAS*. Each question in *GUP* relates to a fundamental grammar concept of the *Flutter* [7]. Additionally, the *Learning Environment for Initiating Flutter App Development Using Docker* has been implemented for *Code Writing Problem (CWP)*. This environment makes it easier for novice students to set up the *Flutter* environment on their computers and independently modify the source code in the projects by following instructions [8].

In this paper, we present a system for independent learning using *Code Modification Problem (CMP)* for *Flutter/Dart programming* in *cross-platform application development*. A CMP instance asks to modify the given source code to generate the user interface as shown in the given screenshot. The correctness of any student answer is verified through *string matching* with the correct answer. A screenshot will help a student to understand the source code and answer the CMP instance. Novice students can learn the basics of *Flutter/Dart programming* through analyzing the screenshot of the user interface.

In this study, we explore the following *Research Questions (RQs)*:

- Can a student who has studied a major programming language such as *C* or *Java* but not *Flutter/Dart* programming modify its source codes without taking formal classes?
- Can such a student develop a basic knowledge of *Flutter/Dart* programming by solving CMP instances?

To explore the answers to these *RQs*, we generated CMP instances covering various topics and widgets in *Flutter/Dart* programming, and assigned them to students with no *Flutter/Dart* programming experience. In the CMP instances, it is requested to alter only the parameters, such as color, alignment, and other widget properties. Solving CMP instances helps students become familiar with *Dart* coding patterns and build a fundamental understanding of the *Flutter* framework.

For evaluations of the proposal, we generated 20 CMP instances for basic topics and 18 CMP instances for advanced topics of *Flutter/Dart programming*. We assigned them to 21 master's students at Okayama University, Japan, who have never studied it. Their solution results indicate that the students effectively fulfilled the assignments and delivered sufficient outcomes.

This paper's structure is outlined as follows: Section 2 contains a review of relevant research on the approach. Section 3 provides an overview of *Flutter* and *Dart Language*. Section 4 addresses the *code modification problem (CMP)* for Flutter. Section 5 presents experimental results. Section 6 concludes this paper with future works.

2. Literature Review

In this section, we explore a comprehensive review of the literature related to the topic of our paper. We examine several studies that have focused on the development of learning platforms for mobile applications.

In [9], Al-Hakim et al. presented an *Android*-based application for teaching algorithms and data structures. This application is implemented using *Java* and *Android Studio* for *Android 6.0* and above, containing 13 learning modules covering topics, such as introductions to algorithms and programming, data structures, and linked lists. The application is freely available for learning purposes. It uses *UML diagrams* that are similar to *activity diagrams* and *use case diagrams* for system processing and user interactions. It adopts *black-box testing* to check the answer. The authors conclude that this application can replace textbooks, enabling both independent and lecturer-assisted learning using mobile technology.

In [10], Septiana et al. presented a learning application on *Android Studio* for building *Android* applications. The authors used various components within *Android Studio*, including manifest files, *Java* source code, and resource files to build the core functionality of the mobile application. *Blender* was used for *3D development* and integrations. It is an *open-source 3D computer graphics* software tool that allows creating 3D models, animations, and visualizations. The integrated 3D contents from *Blender* enhanced biology learning experiences on *Android* devices with interactive features and multimedia presentations.

In [11], Tung presented a flashcard mobile application called *Memoa* using the *Flutter* framework, *Dart*, and *Google Firebase* services. This project aims to create gamified learning experiences for users struggling to retain knowledge from online courses. It covers flashcards, spaced repetitions, and gamification principles. The project follows an *Agile Kanban methodology* for developments. Key features include user authentication, cloud storage, the *Leitner algorithm* for spaced repetitions, and a virtual pet for gamification.

In [12], Yassine et al. presented a mobile application that teaches programming fundamentals by incorporating the *multi-agent system*, *domain ontology*, *checkpoint system*, and *Xamarin.Forms* for cross-platform developments. The goal is to appeal to wide audiences, reduce development and maintenance efforts, and offer versatile and engaging learning experiences.

In [13], Hu et al. discussed improving the teaching of *Mobile Application Development (MAD)* for students with diverse programming backgrounds using online and blended learning approaches. The study addresses the challenges posed by traditional lecture-based methods and text-based lab instructions, which did not adequately cater to students with differing programming experiences.

This literature review provides insights into advancements in the development of learning platforms for mobile application programming. It explores various methodologies, tools, and technologies that have been implemented to enhance the learning of mobile application programming.

To better understand and compare the features of these platforms, Table 1 summarizes their key attributes, advantages, and limitations.

Table 1. Comparison Features of Platforms.

Study	Platform/Tool	Key Features	Advantages	Limitations
Al-Hakim et al. (2021) [9]	Android-based app	13 learning modules, UML diagrams, black-box testing	Freely available, replaces textbooks, supports independent learning	Focused on Android only, lacks cross-platform compatibility
Septiana et al. (2019) [10]	Android Studio + Blender	3D development, interactive biology lessons	Enhanced user engagement through 3D	Limited to specific content (biology), lacks gamification features
Tung (2021) [11]	Flutter-based flashcard app	Gamification, spaced repetition, cloud storage	Cross-platform, gamified learning, uses modern frameworks	Primarily a flashcard system, limited scope for broader learning
Yassine et al. (2018) [12]	Xamarin.Forms-based app	Multi-agent system, domain ontology, checkpoint system	Cross-platform, versatile learning experiences	Complex to implement, challenging for beginners
Hu et al. (2021) [13]	Online/blended learning	Combines online and lecture-based learning	Flexible, addresses diverse learning needs, suitable for novices	Relies on text-based instructions, may not engage advanced students
CMP Flutter-PLAS	Flutter-based app	Code modification tasks, problem-solving focus, Flutter/Dart widget usage examples	Engages students in practical coding challenges, familiarizes them with Flutter/Dart syntax, enhances problem-solving skills	Requires some basic understanding of coding, but gradually builds expertise through practice

3. Overview of *Flutter* and *Dart*

In this section, we overview the *Flutter* framework and *Dart* language. In this project, we are using Flutter version 3.22.2 and Dart version 3.4.3.

3.1. Flutter Framework

Flutter has significantly contributed to the popularity of *Dart* by enabling the creation of high-performance mobile application for both *iOS* and *Android* using a single *codebase*. *Dart* is the primary programming language for *Flutter*, and it offers strong support for building applications. The primary benefit of *Flutter* lies in its capability to create applications that exhibit *close-to-native performance* and allow the utilization of a powerful rendering engine and an adaptable architecture based on *widgets* [5].

Flutter uses *widgets* as fundamental building blocks for user interfaces. Every visual and functional element in *Flutter* is a *widget* which is immutable. *Widgets* can be divided into *StatelessWidget* for static elements and *StatefulWidget* for dynamic elements, according to user interactions. The *widget-based architecture* of *Flutter* simplifies the development and management of complex and flexible user interfaces [5].

Figure 1 shows a Flutter application that uses three main widget components: *MaterialApp*, *Scaffold*, and *ElevatedButton*. The *MaterialApp* provides the framework of the app with a material design, the *Scaffold* provides the basic structure of the page including the app bar and body, while the *ElevatedButton* is a button (with a background) that depresses when clicked, and it contains the text "Hello, Flutter!" as its child widget. This combination of widgets creates a simple user interface.

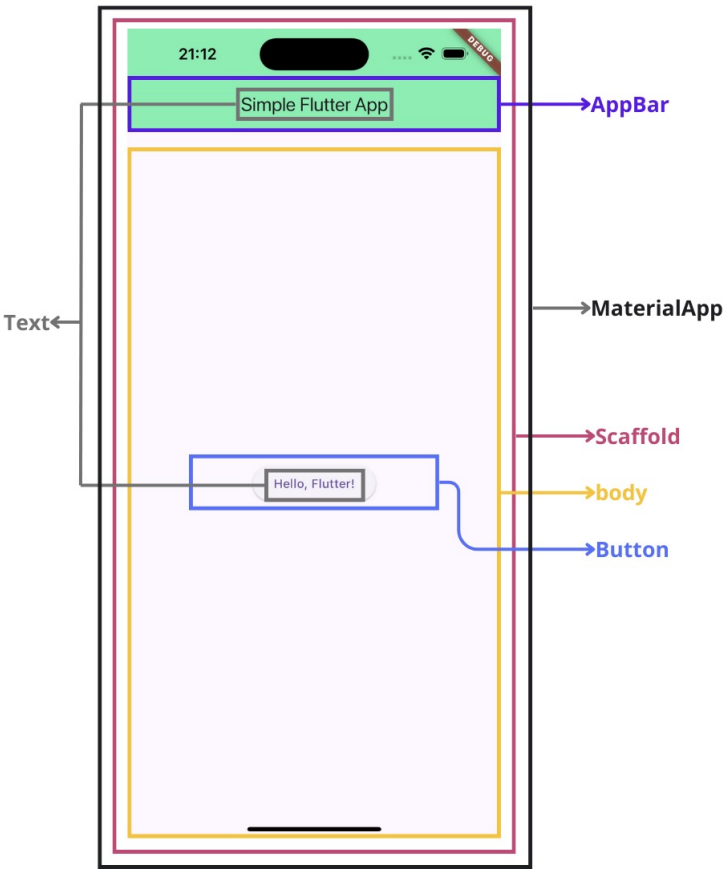

Figure 1. Flutter example application.

3.2. Dart Language

Dart is a programming language that is utilized for natively compiling mobile applications through *Flutter*. Both of them were developed by *Google* [14]. *Dart* is an *object-oriented*, *class-based* language in which the syntax is remarkably similar to *JavaScript*. This makes the transition to *Dart* using *Flutter* easy if one is familiar with *JavaScript*.

Google has developed *Dart* to overcome the limitations of *JavaScript* and achieve superior performance in web and mobile application development. One of the key advantages of *Dart* is its ability to be compiled into *JavaScript*, enabling developers to write code in *Dart* and execute them on web platforms that do not natively support *Dart* [14]. Moreover, *Dart* can be compiled to run on a virtual machine for *server-side applications*.

In addition, the *public package* system of *Dart* enables easy dependency management and a rich library ecosystem, making it simple for developers to integrate various functions and services into their applications [14].

The *Dart* code in Figure 2 generates a color randomly for the window, using the *Random* class in the *dart:math* library to generate a random number. The *getRandomColor()* method randomly selects a color from the *colors* list using the random number. The *colors* list consists of *Color* objects from the *Flutter* library, where each object defines a different color. The *getRandomColor()* method generates a random index by calling the *random.nextInt(colors.length)* method, where *nextInt(int max)* returns a random integer ranging from 0 to $max-1$. This

index is used to return the corresponding color in the list that is applied as the background color of the *Container* widget through the color property, where *Container(color: getRandomColor())* sets the *Container's* background color based on the selected color.

```dart
import 'package:flutter/material.dart';
import 'dart:math';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  final List<Color> colors = [
    Colors.red, Colors.blue, Colors.green,
    Colors.yellow, Colors.orange, Colors.purple,
  ];
  final Random random = Random();

  Color getRandomColor() {
    return colors[random.nextInt(colors.length)];
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Random Background Color'),
        ),  // AppBar
        body: Container(
          color: getRandomColor(),
          child: Center(
            child: Text(
              'Random Color',
              style: TextStyle(fontSize: 24.0, color: Colors.white),
            ),  // Text
          ),  // Center
        ),  // Container
      ),  // Scaffold
    );  // MaterialApp
  }
}
```

Figure 2. Dart Example Application.

4. Code Modification Problem for *Flutter/Dart* Programming

In this section, we present a *code modification problem (CMP)* for self-study of *Flutter/Dart* programming in PLAS.

4.1. Overview of PLAS

Previously, we developed a web-based *Programming Learning Assistant System* (PLAS) for the self-study of programming languages through practicing *code reading* and *code writing*. Currently, *PLAS* covers familiar programming languages such as *C*, *C++*, *Java*, *JavaScript*, *Python*, *Scratch*, and *Verilog*. *PLAS* provides various types of programming exercises to enhance the effectiveness of learning programming.

Figure 3 shows the learning path of a programming language in *PLAS*.

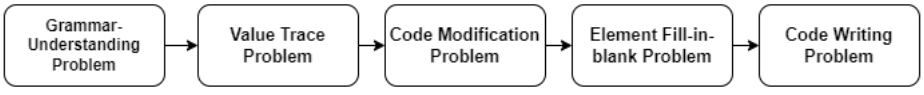

Figure 3. Flow of learning path in PLAS.

1. *Grammar—Concept Understanding Problem (GUP)* asks to answer the keywords or common libraries in the given source code that represent the concepts in the questions, which emphasizes the *grammar* study.
2. *Value Trace Problem (VTP)* asks to answer the values of important variables and output messages in the source code, which emphasizes the *code reading* study.
3. *Code Modification Problem (CMP)* asks to modify the source code to satisfy the new specifications, which emphasizes *code debugging* study.
4. *Element Fill-in-blank Problem (EFP)* asks to fill in the blank elements in the source code with the original words, which emphasizes the partial *code writing* study.
5. *Code Writing Problem (CWP)* asks to write the source code that can pass the *test code* from scratch, which emphasizes the *code writing* study.

Along this learning path in *PLAS*, first, students will grasp the various keywords of the programming language used in the source code *GUP*. Second, they will understand the behaviors of the source code with *VTP*. Third, they will study how to debug and modify the source code with *CMP*. Fourth, they will study how to complete the source code with *EFP*. Finally, they will study how to write the source code from scratch with *CWP*.

4.2. Overview CMP

A CMP instance consists of a *Dart* source code and a set of *screenshots* for mobile application pages. Among them, the initial screenshot displays the application page created by the *Dart* source code. The second screenshot depicts the page created by the source code modified by a student. These pages may have differences in parameters, widgets, functions, methods, properties, or variables. Challenges for the student are to carefully analyze source code and determine necessary modifications by comparing the original screenshots and the modified screenshots.

The elements targeted for modifications in the CMP include parameters such as color, size, positioning, and alignment, because they directly impact the visual layout of the mobile application. Ensuring these elements have only one correct modifications, which guarantees that the answer remains unique. Moreover, modifying them is highly beneficial in learning *Flutter/Dart* programming as it deepens students' understanding of *widget* properties and layout management. This hands-on approach reinforces key *Flutter/Dart* programming concepts and helps students grasp the principles of UI designs and widget behaviors.

The correctness of any answer from a student is checked by *matching string* with the registered correct answer. The system employs two types of matching:

- **Exact matching with spaces/tabs**: This compares the hashed value of the student's answer, including spaces and tabs, with the correct answer's hash.
- **Matching without spaces/tabs**: This compares the hashed value of the answer after removing any extra space or tab, ensuring that minor formatting issues do not affect the correctness.

In the CMP, we follow the requirement to create any problem code line that has only one correct modification. When a student modifies the problem source code, the correctness of his/her answer is checked through *string matching* by the JavaScript program at the web interface, while avoiding missing any correct answer nor allowing any incorrect one.

4.3. Modification Code Selections for CMP Instance

A source code in *cross-platform mobile programming* is composed of widgets and methods that define the structure and behaviors of the application. Thus, the following elements in a source code can be selected for modifications in a CMP instance:

- Variables and values;
- Conditional statements;
- Function and classes (with parameters, properties, methods).

It is noted that *Flutter* forms an interface by taking the concept of a *widget tree*. This means that each widget represents a class that has the key properties aiding the development of the interface. Therefore, the modifications in a CMP instance may include changes in shape, color, alignment, text content, shadow effects, and borders of widgets. To learn the effective use of widgets, students will frequently make modifications to data types, variables, conditional statements, functions, and classes. This hands-on approach helps students understand the interactive aspects of Flutter and Dart programming, thus enabling them to create *cross-platform mobile applications*.

4.4. CMP Generation Procedure

A CMP instance can be generated through the following procedure:

1. Prepare a source code containing the *Flutter/Dart* topics to be studied in this instance, such as *widgets*, classes, and methods.
2. Execute this source code and take the screenshots of the generated application pages.
3. Select the parts to be modified in this (original) source code as the questions to be answered by students.
4. Make the the modified source code by changing the selected parts. It is noted that currently, these steps are carried out manually, which will be automated in future studies.
5. Execute the modified source code and take the screenshots of the application pages.
6. Save the original source code and the modified source code in one text file. This text file is used as the input file to the *interface generation program* [15].
7. Execute the *interface generation program* with the input text file to generate the CMP instance files including *HTML*, *CSS*, and *JavaScript* for the answer interface on a web browser.
8. Add the screenshots in the generated files, since they are not included automatically, which will also be automated in future studies.

4.5. Example of Generating CMP Instance

Here, we discuss an example of a new CMP instance generation.

4.5.1. Original Source Code

Basically, in this paper, we selected a source code that covers basic topics or multimedia and storage topics to generate a new CMP instance. For this example, the source code in Listing 1 is selected as the original source code that covers the topic of inserting an image into an application. It is used in the CMP instance with ID = 3 for basic topics.

Listing 1 CMP original source code.

```
1  import "package:flutter/material.dart";
2
3  void main() {
4    runApp(const InsertingImage());
5  }
6
7  class InsertingImage extends StatelessWidget {
8    const InsertingImage({Key? key}) : super(key: key);
9
10   @override
11   Widget build(BuildContext context) {
12     return MaterialApp(
13       home: Scaffold(
14           appBar: AppBar(
15             //"AppBar" typically contains the title of the application.
16             backgroundColor: Colors.blueAccent,
17             title: const Text(
18               "Problem 3 : Insert Image",
19               style: TextStyle(color: Colors.white, fontWeight: FontWeight
                     .normal),
```

```
20              ), //It gives the title message "Problem 3 : Insert Image" and
                    its text style.
21              centerTitle: false, //It aligns the title to the left side of
                    the screen.
22            ),
23            body: Column(
24              children: [
25                Image.asset(
26                  //It displays an image.
27                  "assets/images/tamagotchi.png",
28                  alignment: Alignment.center,
29                  scale: 1, //The image size is unchanged from the original
                        size.
30                ),
31                const Padding(
32                  padding: EdgeInsets.symmetric(vertical: 10, horizontal:
                        20),
33                  child: Text(
34                    "This is Tamagotchi. Tamagotchi is a digital pet that
                          was popular in the 1990s and is still enjoyed by
                          many today.",
35                    textAlign: TextAlign.center, //It aligns the text to the
                          center.
36                  ),
37                )
38              ],
39            )
40          ),
41        );
42    }
43  }
```

4.5.2. Original Result

The results of the original source code for CMP basic topic instance ID = 3 in Figure 4 show the application that was created using source code in Listing 1. This application is offered as a reference for understanding the source code.

Figure 4. CMP original result.

4.5.3. Modified Source Code

The original source code can be converted to the modified source code by changing a few parameters and functions. It is important for students to understand the connections among *widget* classes, parameters, and properties when creating the application. Listing 2

displays the modified source code as the correct answer. In this instance, it is necessary to change the value of the scale parameter for the image and the value of the alignment parameter for the text content and to adjust the appbar content by changing the title and centering it.

Listing 2 CMP modified source code.

```
import "package:flutter/material.dart";

void main() {
  runApp(const InsertingImage());
}

class InsertingImage extends StatelessWidget {
  const InsertingImage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
          appBar: AppBar(
            //"AppBar" typically contains the title of the application.
            backgroundColor: Colors.blueAccent,
            title: const Text(
              "Bigger Image",
              style: TextStyle(color: Colors.white, fontWeight: FontWeight
                  .bold),
            ), //It gives the title message "More Big Image" and its text
                style.
            centerTitle: true, //It aligns the title to the center.
          ),
          body: Column(
            children: [
              Image.asset(
                //It displays an image.
                "assets/images/tamagotchi.png",
                alignment: Alignment.center,
                scale: 0.7, //It scales the image to 70% of the original
                    size.
              ),
              const Padding(
                padding: EdgeInsets.symmetric(vertical: 10, horizontal:
                    20),
                child: Text(
                  "This is Tamagotchi. Tamagotchi is a digital pet that
                      was popular in the 1990s and is still enjoyed by
                      many today.",
                textAlign: TextAlign.justify, //It justifies the text at
                     the left end.
                ),
              )
            ],
          )
      ),
    );
  }
}
```

4.5.4. Modified Result

Figure 5 shows the results of the application page from the source code that students modified.

Figure 5. CMP modified result.

4.6. Answer Interface for CMP

The answer interface for resolving this CMP instance in Figure 6 highlights the input form in red if the answer is incorrect. If it is correct, it keeps the white background. Students can continue submitting answers until they become correct. The answer interface automatically records the number of submissions and the answers for each attempt.

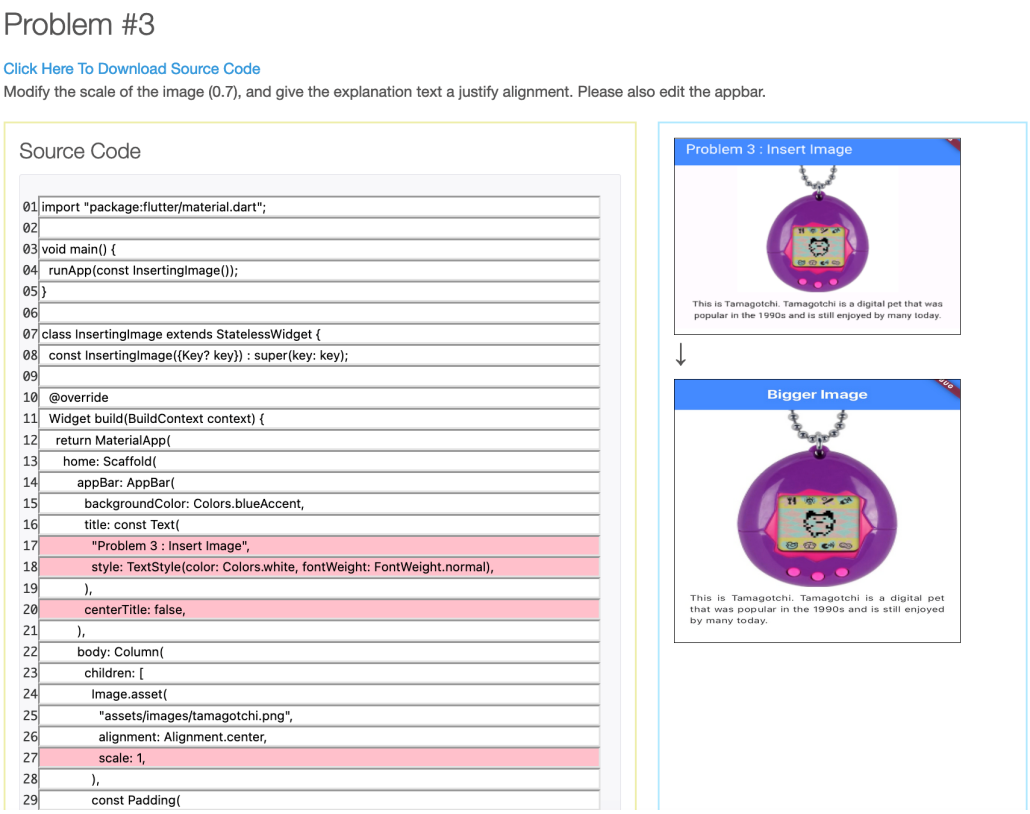

Figure 6. CMP answer interface.

4.7. Student Workflow for Solution Submission

Figure 7 outlines the workflow of modifying and submitting a solution to a CMP instance. First, a student reads the instruction and sees the initial web page by the given source code. Second, he/she directly modifies the source code in the interface. Third, he/she can download the current source code into a text file by clicking the "File Save" button, and run it on a web browser to see the output web page if necessary. Fourth, he/she submits the source code by clicking the "Answer" button.

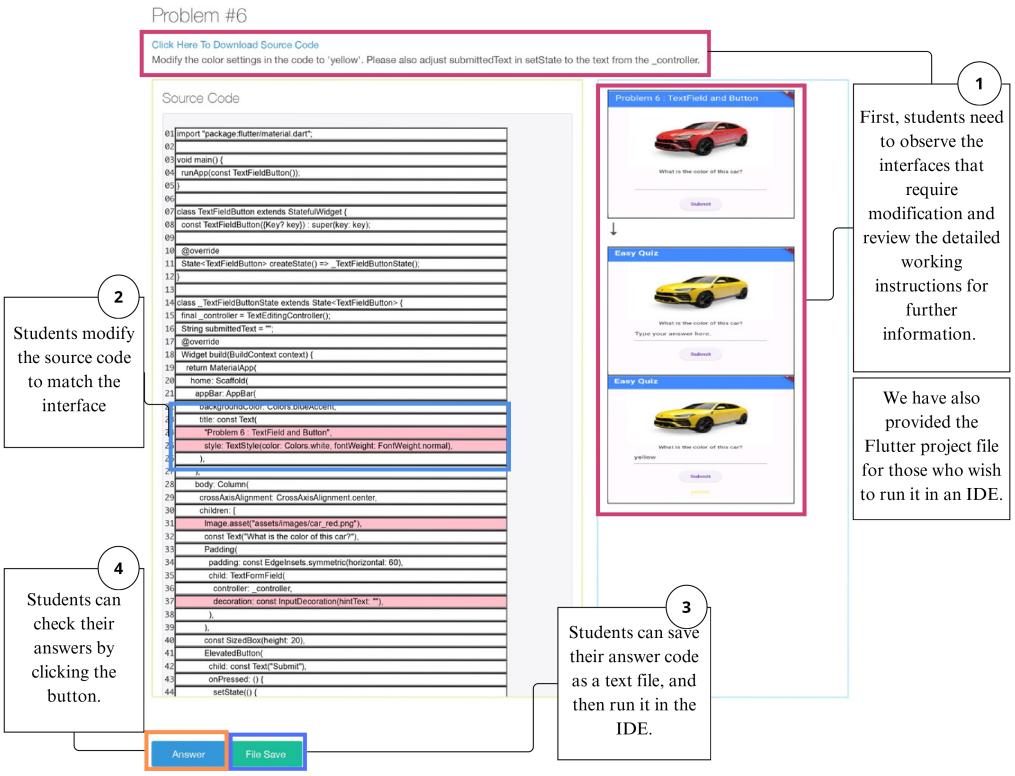

Figure 7. Student workflow for solution submission.

Since direct testing of the modified source code within the system is not available, we provide a feature that allows a student to download the modified code into a text file and run it on a web browser. It enables him/her to review the current answer and ensure its alignment with the required page before its submission. Additionally, we offer the function for downloading all the necessary packages and dependencies in a zip file, allowing the student to run any modified code locally in an IDE.

5. Evaluation

In this section, we evaluate the code modification problem for *Flutter/Dart programming* in *PLAS*.

5.1. CMP Instances

For this evaluation, we generated 20 CMP instances for basic topics in Table 2 and 18 CMP instances for multimedia and storage topics in Table 3 in *Flutter/Dart* cross-platform mobile application programming. These tables show the topic or widget, the number of lines in the source code, and the number of elements to be modified. The two sets of CMP instances were designed with the consideration of ensuring that novice students can solve them on a self-study basis.

Table 2. CMP instances for basic topics.

ID (CMP Instances)	Topic/Widget	# of LINES	# of Modified Elements
1	Text and Align	60	10
2	Container and AppBar	58	11
3	Insert Image	41	5
4	Column and Row	70	13
5	Device Height and Width	42	5
6	Textfield and Button	61	6
7	State Objects Change	53	5
8	TextField Type and Icon	62	15
9	Card Tile List	49	10
10	Using List View	45	7
11	Using Grid View	52	7
12	Scrolling View	60	8
13	Dropdown Button	72	5
14	Radio Button with Enum	51	6
15	Using Check Box Uses	65	7
16	Pop Up Dialog	69	11
17	Pop Up Snack Bar	66	6
18	Navigation to Other Page	72	12
19	Bottom Navigation Bar	69	8
20	Progress Bar	62	11

Table 3. CMP instances for multimedia and storage topics.

ID (CMP Instances)	Topic/Widget	# of Lines	# of Modified Elements
1	Passing Data to Next Page	98	12
2	Package Install	56	11
3	Add Video	79	8
4	Video Player Controls	99	15
5	Add Audio	66	9
6	Audio Player Controls	88	11
7	Future, Async and Await	97	12
8	File Image from Gallery	80	13
9	File picker	91	14
10	Take Image with Controls	112	16
11	Take Image and Save	115	18
12	Animated Container	70	10
13	Tween Animation	82	14
14	Hero Animation	88	14
15	Shared Preferences	75	13
16	Local Storage Create and Read	98	11
17	Local Storage Update	113	11
18	Local Storage Remove	104	9

5.2. Assignment to Students

Then, we assigned these instances to 21 first-year master students at Okayama University in Japan who have not formally studied *Flutter/Dart programming*. Before the assignments, we did not conduct any lectures on them. Instead, we provided references and websites that can be helpful in solving the CMP instances. The students spent two

class hours solving the 38 CMP instances, where one class hour has 100 min. They could continue solving them for one more week at home if necessary.

5.3. Solution Results for Basic Topics

First, we discuss the solution results for *basic topics*.

5.3.1. Results of Individual Students

In the beginning, we see the solution results for the individual students. Table 4 shows the number of answer submission times and the correct answer rate (%) for each of the 21 students among the 20 CMP instances for *basic topics*.

Table 4. Solution results of individual students for *basic topics*.

ID (Students)	# of Sub. Times	Ave. Correct Rate
1	270	99.83%
2	302	95.67%
3	563	97.99%
4	93	98.17%
5	204	97.98%
6	230	98.67%
7	104	25.00%
8	168	99.92%
9	288	97.12%
10	99	60.32%
11	50	98.68%
12	213	86.18%
13	29	24.86%
14	116	100.00%
15	208	100.00%
16	168	99.92%
17	165	99.74%
18	153	34.20%
19	228	100.00%
20	133	100.00%
21	140	100.00%

Distribution of Correct Answer Rate

Table 5 shows the distribution of the correct answer rates. Five students achieved the perfect score (100%), and 11 students did the score above 90%. On the other hand, only five students did the score bellow 90%. The results indicate that the generated CMP instances can be solved by novice students on a self-study basis. Thus, the proposed *PLAS* with *CMP* for *basic topics* is proper for independent learning of the *Flutter/Dart cross-platform mobile* programming by novice students.

Table 5. Correct answer rate distribution for *basic topics*.

Correct Rate Answer	# of Students
$\leq$65%	4
66–90%	1
91–99%	11
100%	5

Distribution of Answer Submission Times

Table 6 shows the number of answer submission times by the students to finalize their answer. The average number of submission times to solve the 20 CMP instances for *basics topics* is 186.85. Generally, the students could solve one instance by submitting answers

9.34 times on average, which suggests that the students carefully reviewed their answers before submissions.

Table 6. Submission times distribution for *basic topics*.

Sub. Times Range	# of Students
25–50	2
51–100	2
101–150	4
151–250	9
≥251	4

5.3.2. Results of Individual Instances

Then, we discuss the solution results for the individual instances. Table 7 shows the average correct answer rate and the average number of answer submission times among the 21 students for each of the 20 CMP instances. This table suggest that the difficulty varied across the instances, where the lowest rate is 96.90% for ID = 11 and the highest rate is 99.67% for ID = 1. Similarly, the number of answer submission times required varied, where the lowest one is 3.72 for ID = 10 and the highest one is 18.74 for ID = 7.

Table 7. Solution results of individual instances for *basic topics*.

ID (CMP Instances)	# of Students Who Answered	Ave. Rate (%)	Ave. # Sub.
1	21	99.67%	13
2	21	99.50%	8.81
3	21	99.07%	6.81
4	21	98.57%	12.71
5	21	99.32%	11.05
6	19	99.14%	14.21
7	19	97.42%	18.74
8	18	98.21%	15.39
9	18	98.53%	5.44
10	18	99.63%	3.72
11	18	96.90%	6
12	18	97.87%	8.94
13	18	98.15%	9.56
14	17	97.92%	7.76
15	17	99.64%	5.29
16	17	99.57%	12.41
17	16	98.48%	14.63
18	17	98.77%	6.18
19	16	97.74%	14.13
20	17	97.34%	18.65

5.4. Solution Results for Multimedia and Storage Topics

Next, we discuss the solution results of 17 students for 18 CMP instances for *multimedia and storage topics*. Unfortunately, four students did not solve them.

5.4.1. Results of Individual Students

In the beginning, we see solution results for the individual students. Table 8 shows the number of answer submission times and correct answer rate (%) for each of the 17 students among the 18 CMP instances for *multimedia and storage topics*.

Table 8. Solution results of individual students for *multimedia and storage topics*.

ID (Students)	# Sub. Times	Ave. # Rate (%)
1	263	99.52%
2	189	51.19%
3	44	98.07%
4	247	98.91%
5	175	98.21%
6	410	32.87%
7	429	99.66%
8	95	55.92%
9	253	94.86%
10	31	27.38%
11	47	99.95%
12	477	99.31%
13	136	49.31%
14	298	98.20%
15	152	32.85%
16	264	100.00%
17	257	99.56%

5.4.2. Distribution of Correct Answer Rate

Table 9 shows the distribution of the correct answer rates. One student achieved the perfect score (100%), and 10 students received a score above 90%. On the other hand, six students did not reach a score of 90%. The results indicate that the generated CMP instances can be solved by most novice students on self-study basis. Thus, the *PLAS* with *CMP* for *multimedia and storage topics* is also proper.

Table 9. Correct answer rate distribution for *multimedia and storage topics*.

Correct Rate Answer	# of Students
$\leq$65%	6
66–90%	0
91–99%	10
100%	1

5.4.3. Distribution of Answer Submissions Times

Table 10 shows the number of answer submission times by the students to finalize their answers. The average number of submission times to solve the 18 CMP instances for *multimedia and storage topics* is 221.59. The students solved one instance by submitting answers 12.31 times on average.

Table 10. Submission times distribution for *multimedia and storage topics*.

Sub. Times Range	# of Students
25–50	3
51–100	1
101–150	1
151–250	4
$\geq$251	8

5.4.4. Results of Individual Instances

Then, we discuss the solution results for individual instances. Table 11 shows the average correct answer rate and the average number of answer submission times among the 17 students for each of the 18 CMP instances. It is noted that some students did not have enough time to complete the CMP instances for *multimedia and storage topics*. This

table suggests that the difficulty is varied across the instances, where the lowest rate is 96.73% for ID = 6 and the highest is 99.39% for ID = 18. Similarly, the number of answer submission times is varied, where the lowest one is 5.18 for ID = 18 and the highest one is 38.94 for ID = 1.

Table 11. Solution results of individual instances for *multimedia and storage topics*.

ID (CMP Instances)	# of Students Who Answered	Ave. Rate (%)	Ave. # Sub.
1	17	97.78%	38.94
2	17	98.11%	17.06
3	17	98.66%	18.88
4	17	98.16%	14.76
5	17	98.66%	14.18
6	16	96.73%	25.13
7	14	97.57%	10.21
8	14	97.14%	16.21
9	14	97.25%	9.86
10	13	97.73%	11.15
11	12	97.30%	13.58
12	11	99.22%	9.91
13	11	98.43%	17.18
14	11	98.86%	10
15	11	98.30%	12.55
16	11	99.26%	8.18
17	11	99.03%	8.27
18	11	99.39%	5.18

5.5. Discussions

Understanding the challenges faced by novice students is crucial for improving educational approaches in mobile programming. Several studies have highlighted the significant challenges novice students face when working on mobile programming tasks. These difficulties often arise from the need to integrate *platform-specific* knowledge with foundational programming skills, along with the high cognitive demands of *mobile application development* [16,17]. This is consistent with our observations in the proposed learning system.

Building on these insights, we analyzed the frequent mistakes of students from their solution results.

5.5.1. Mistakes in Basic Topics

In the 20 CMP instances for *basic topics*, ID = 3, 5, 6, and 11 contain classes that require properties to be applied. Properties are variables defined in the class. Students struggled with properties such as *aligning*, *font weight*, and *font style*.

ID = 7 requires the addition of the *.toString()* method, which is used to convert any data type into *String*. However, students often struggled to understand how to use this method. To help students understand how to use it, a debugging exercise can be useful by showing the output of the method, which will be in future works.

ID = 11 contains an element that requests the use of *List* in the program. Although *List* functions similarly to *Array* in other programming languages, students are unsure how to use it in *Flutter/Dart*. To improve their understanding, it can be useful to provide the problems for understanding *collection data types* such as *lists*, *maps*, *sets*, and *enums*, combined with *data structures* on *Dart* with their use in *Flutter*.

5.5.2. Mistakes in Multimedia and Storage Topics

Next, we analyzed the frequent mistakes of students from their solution results in *multimedia and storage topics*.

In ID = 1, 7, 9, 11, 13, and 15, certain elements can be modified based on the given instructions or by referring to the provided screenshots. This often involves changing the color of the elements, despite providing instructions and being able to see it in the screenshots. Students are required to read and comprehend these modifications. Thus, the instruction can be improved for better understanding.

In ID = 3, 5, and 6, there is an element that requires the use of *ternary conditional operators*. Students did not understand how to utilize this operator.

In ID = 16, 17, and 18, there is an element that requires the use of the provided *Maps*. The *Maps* type in *Flutter* is similar to the *dictionary* type in other programming languages. Students often struggle with how to utilize them in *Flutter*. To help students understand them, additional problems on *collection data types* combined with *data structure* and *ternary conditional operators* in *Dart* should be provided with guidance, which will be carried out in future works.

5.5.3. Comparing of Learning Platforms

Compared with other platforms such as the *mobile-based application* developed by Al-Hakim et al. [9], *cross-platform solution* presented by Yassine et al. [12], and the blended learning approach discussed by Hu et al. [13], our approach focuses on enhancing problem-solving skills through code modification problems. While these platforms provide structured learning experiences or integrate various learning methods. Our system offers a distinct method by inviting students to modify code, which helps students practice their observation and problem-solving abilities in a controlled environment. This method supports students in developing their coding skills by engaging them in practical coding challenges where they need to understand and adjust code to meet specific interface requirements.

Our findings align with previous research indicating that students benefit from engaging in problem-solving activities [10,11]. By focusing on coding modification tasks that require students to analyze and adapt code based on interface observations, our system supports the idea that engaging students in practical, hands-on activities can improve their comprehension and retention of programming concepts. This approach aligns with the view that active involvement in solving coding problems enhances learning outcomes compared with more passive methods.

5.6. Feedback

After solving the CMP instances, the students gave the following opinions as feedback:

- The screenshots are unclear, and certain text and word styles are difficult to see.
- The instructions are challenging to read.
- Some problems are quite tricky.
- The transition from *basic topics* to *multimedia and storage topics* makes a significant difference.
- The hint button is useful if available.

To support independent learning by the proposal, we will address these problems in our future works.

5.7. Limitations

While this study provides valuable insights, several limitations should be considered. First, the sample size of 21 novice students from a single university may limit the generalizability of the findings. This participant pool may also introduce the subject selection bias, since students from different backgrounds or institutions may produce varied results. Although 38 CMP instances were developed to ensure the thorough coverage of fundamental concepts of *Flutter/Dart* programming, some advanced topics still need to be implemented. Besides, the current system allows the modifications of the parameters such as color, size, position, and alignment to avoid the issues related to *string matching* at the answer verification. As the next step, this limitation should be alleviated to allow modifica-

tions of functions or classes. Further integration across various educational contexts will be necessary in future works or studies.

5.8. Implications

This study has several implications for both researchers and practitioners in education technology and mobile application programming learning. For researchers, the findings will provide a foundation for further investigations of adaptive learning systems that automatically adjust the problem difficulty based on student performances. For practitioners, and particularly for educators, this system will serve as a practical tool to support independent learning and supplement classroom teaching of *Flutter/Dart* programming. It offers an interactive method to enhance student engagement. Integrating this system into both formal and informal learning environments could offer students a more flexible and self-paced way to develop their programming skills.

6. Conclusions

This paper presented the *programming learning assistant system (PLAS)* with the *code modification problem (CMP)* for *Flutter/Dart cross-platform mobile application* programming. Since the answer interface is implemented on a web browser, students can easily access this independent learning environment without needing additional tools or setups. A key feature of our system is the use of user interface screenshots, which helps students visualize the expected output and understand the relationship between the code and interface changes. For the evaluations, 38 CMP instances were generated and assigned to 21 novice students at Okayama University, Japan. Without the formal classes of *Flutter/Dart* programming, their solution results confirm that novice students can understand and solve the CMPs efficiently. It is implied that *PLAS with CMP* is a valuable system for providing a structured platform for independent learning. In future works, we will enhance the *PLAS with CMP* for independent learning with additional topics in the *cross-platform mobile application programming* and evaluate the effectiveness with a larger sample size. We will also focus on expanding the areas of code modifications, such as enabling edits to entire functions or classes to further improve the system's flexibility and effectiveness. They will help us further assess its effectiveness in supporting independent learning across a wider range of programming skills.

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

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

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

References

1. Statista. Smartphone Users Worldwide—Forecast to 2027. Available online: https://www.statista.com/forecasts/1143723/smartphone-users-in-the-world (accessed on 20 July 2024).
2. data.ai. State of Mobile 2023. Available online: https://www.data.ai/en/go/state-of-mobile-2023/ (accessed on 20 July 2024).
3. Pinto, C.M.; Coutinho, C. From native to cross-platform hybrid development. In Proceedings of the 2018 International Conference on Intelligent Systems (IS), Madeira, Portugal, 25–27 September 2018.
4. Hiwale, P.R. Review on cross-platform Mobile Application Development. *Int. J. Res. Appl. Sci. Eng. Technol.* **2022**, *10*, 1433–1439. [CrossRef]
5. Flutter. Build Apps for Any Screen. Available online: https://flutter.dev/ (accessed on 20 July 2024).
6. Simplilearn. Kotlin vs. Flutter: The Best Guide to Choose between Them. Available online: https://www.simplilearn.com/tutorials/kotlin-tutorial/kotlin-vs-flutter (accessed on 20 July 2024).

7. Patta, A.R.; Funabiki, N.; Lu, X.; Syaifudin, Y.W. A study of grammar-concept understanding problem for flutter cross-platform mobile programming learning. In Proceedings of the 2023 Sixth International Conference on Vocational Education and Electrical Engineering (ICVEE), Surabaya, Indonesia, 25–26 October 2023.
8. 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. [CrossRef]
9. Al Hakim, R.R.; Kisworini, R.Y.; Hamid, A.P.; Pangestu, A.; Jaenul, A.; Arief, Y.Z. Design and Development of Android-Based Learning Media for Learning Algorithm and Data Structure. *Semnasfkip* **2021**, *3*, 321–329.
10. Septiana, D.; Gunarhadi, G.; Akhyar, M. Mobile application with 3D to improve self-determined learning interest: Student's response and challenge in biology class. In Proceedings of the First International Conference on Progressive Civil Society (ICONPROCS 2019), Surakarta, Indonesia, 27–28 August 2019.
11. Huynh, T. *A Flashcard Mobile Application Development with Flutter*; Theseus: Helsinki, Finland, 2021.
12. Yassine, A.; Berrada, M.; Tahiri, A.; Chenouni, D. A cross-platform mobile application for Learning Programming Basics. *Int. J. Interact. Mob. Technol. (IJIM)* **2018**, *12*, 139. [CrossRef]
13. Hu, M.; Assadi, T.; Baliuag, C. Using online and blended learning method for teaching novices in mobile application development. In Proceedings of the 2021 World Engineering Education Forum/Global Engineering Deans Council (WEEF/GEDC), Madrid, Spain, 15–18 November 2021.
14. Dart. Dart Overview. Available online: https://dart.dev/overview (accessed on 20 July 2024).
15. Wai, K.H.; Funabiki, N.; Qi, H.; Xiao, Y.; Mon, K.T.; Syaifudin, Y.W. Code modification problems for multimedia use in JavaScript-based web client programming. In *Complex, Intelligent and Software Intensive Systems*; Springer: Cham, Switzerland, 2022; pp. 548–556.
16. Malik, S.I.; Mathew, R.; Hammood, M.M. PROBSOL: A web-based application to develop problem-solving skills in introductory programming. In *Smart Technologies and Innovation for a Sustainable Future*; Springer: Cham, Switzerland, 2019; pp. 295–302.
17. McCall, D.; Kölling, M. A new look at novice programmer errors. *ACM Trans. Comput. Educ.* **2019**, *19*, 1–30. [CrossRef]