International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. , No. (Mon-20..)

http://dx.doi.org/10.12785/ijcds/XXXXXX

Self-learning Topics for Multiple Activities Mobile Application with Multimedia Integration in Android Programming Learning Assistance System

Yan Watequlis Syaifudin[1], **Nobuo Funabiki**[2], **Andi Baso Kaswar**[3], **Triana Fatmawati**[1], **Mustika Mentari**[2], **Pramana Yoga Saputra**[1], **Yuri Ariyanto**[1] **and Abdul Rahman Patta**[3]

[1]*Department of Information Technology, Politeknik Negeri Malang, Malang, Indonesia*
[2]*Department of Electrical and Communication Engineering, Okayama University, Okayama, Japan*
[3]*Department of Computer Engineering, Universitas Negeri Makassar, Makassar, Makassar*

Received Mon. 20, Revised Mon. 20, Accepted Mon. 20, Published Mon. 20

Abstract:
The widespread use of Android smartphones underscores the growing need for skilled mobile developers and highlights the importance of effective Android programming education in IT curricula. The rapid evolution of technology and the demand for practical programming skills create significant challenges in delivering up-to-date and impactful Android programming classes. The *Android Programming Learning Assistance System (APLAS)* has been developed to address these challenges by supporting autonomous learning through *Test-Driven Development (TDD)* and automating the validation of code submissions. This study presents novel implementations within APLAS for two key learning topics: Multiple Activities and Multimedia Resources, integral to the Interactive Application stage. These topics are designed to foster self-directed learning and practical application of Android programming concepts. In the Multiple Activities topic, students develop an interactive application, learning to manage multiple Activities, use Intents, and apply advanced UI elements. The Multimedia Resources topic involves creating a dynamic application, incorporating various multimedia elements such as images, videos, and animations. The evaluations show a high success rate for both topics, with 90% of students successfully completing the Multiple Activities assignments and 100% succeeding on the first attempt for Multimedia Resources. Feedback highlights improved skills, problem solving capabilities, and overall satisfaction, despite suggestions for enhanced guidance and technical support. The consistent progress observed in student performance underscores the effectiveness of the APLAS framework in fostering robust Android development skills, while also pointing to areas needing further refinement for addressing complex topics and technical challenges.

Keywords: Self-learning system, Android, Test-driven development, Multiple Activities, Multimedia

1. Introduction

The widespread popularity of Android smartphones is reflected in Statista's data[1], which reports a substantial number of global users that reach 2.7 billion[2]. This vast user base highlights the increasing demand for mobile programmers who can develop applications for this dominant platform. Consequently, mobile programming has emerged as a main subject in IT departments across educational institutions, emphasizing its critical role in contemporary technology education. However, delivering effective *Android programming* classes presents challenges, such as keeping the curriculum up to date with rapid technological advancements and ensuring students acquire the practical skills[3]. Addressing these challenges is essential for optimizing the learning experience to meet the evolving needs of the mobile development industry.

The *Android Programming Learning Assistance System (APLAS)*[4] has been introduced in prior research as a platform designed to support autonomous learning in Android programming by incorporating the *Test-Driven Development (TDD)* methodology[5]. It helps students to write Java, XML, and Gradle DSL source codes within Android Studio and validates these codes using *JUnit*[6] and *Robolectric*[7], thus eliminating the need for teacher verification. APLAS also features an online web platform that distributes learning materials and collects student submissions through a server-side validator program[8]. To offer a structured learning experience on diverse topics, APLAS organizes its content into four stages[9].

The user interface (UI), along with *multiple activities* and *multimedia resources*, significantly influences the user experience and functionality of an Android application[10]. The UI provides the visual representation that guides user interactions, making it essential to design an intuitive and

E-mail address: qulis@polinema.ac.id, funabiki@okayama-u.ac.jp, a.baso.kaswar@unm.ac.id, triana@polinema.ac.id, pqt85hm5@s.okayama-u.ac.jp, pramanay@polinema.ac.id, yuri@polinema.ac.id, abd.rahman.patta@unm.ac.id*

engaging layout. Multiple activities organize application functions into separate screens, enhancing navigation and task execution. The Activity class provides the screens or windows, with each Activity having its own lifecycle. The Intent class facilitates navigation between Activities, while the Fragment class allows for modular UI components to be embedded within an Activity. Multimedia resources, such as images, videos, audio, and animations, enrich the application's content and interaction, creating a more immersive experience. Android applications leverage these components to develop interactive, user-centered apps[11].

This study presents novel implementations for two key learning topics in APLAS: Multiple Activities and Multimedia Resources. These topics are integral to the Interactive Application stage, a critical component of foundational Android programming education[9]. Designed to support the self-learning process in mobile programming classes, these topics allow students to independently learn essential concepts. In practical case studies, the Multiple Activities topic requires students to develop a SoccerMatch application, while the Multimedia Resources topic involves creating an AnimalTour application. These projects aim to enhance students' hands-on experience and self-directed learning by applying concepts in real-world scenarios, thereby supporting a more effective and engaging mobile programming education. Students can submit their Android projects to the server to receive verification and validation results for their code, ensuring accuracy and providing feedback to enhance their learning experience.

The Multiple Activities topic explores the development of Android applications that utilize multiple Activities through the use of Intent and Fragment components. This lesson plan is structured around six key learning objectives, including mastering application resources, managing multiple Activities, designing UI layouts, and implementing Java and Android classes, with a focus on event listeners for interactive features. The assignment is divided into eight tasks, guiding students through creating the application from initial setup and UI design to finalizing user interactions. To complete the assignment for hands-on experience, students develop SoccerMatch application that is designed with four Activities and various UI elements. The application integrates various widgets like *CardView*, *RecyclerView*, and *Chronometer*, and implementing three main Activities (*MainActivity*, *PlayActivity*, and *LogActivity*).

The Multimedia Resources topic is designed to assist students developing Android applications that integrate multimedia elements effectively. It is structured around five key learning objectives: using application resources, designing multimedia user interfaces, incorporating multimedia components like images, videos, and YouTube videos, applying animations, and handling user events. To complete the assignment hands-on experience, students follow eight specific tasks to develop AnimalTour application that is designed with four Activities and various UI elements. The application utilizes multimedia elements such as images, video files, animations, and YouTube videos, and incorporates animations, animated transitions, and gesture interactions.

The evaluation of the Multiple Activities and Multimedia Resources topics in Android programming involved a structured assessment of student proficiency through practical assignments. For the Multiple Activities topic, 90% of the students completed the assignments successfully within the allotted three days. Despite a few students needing additional time to address specific issues. In contrast, the Multimedia Resources topic resulted in a 100% success rate on the first attempt by all students, reflecting a high level of competence in integrating multimedia elements into Android applications. Feedback from students was predominantly positive, highlighting satisfaction with the learning experience and assignments, though some suggested improvements in areas such as UI design and technical issues. The creative modifications students made to the assignments further demonstrate their growing skills and confidence.

The paper content is organized into six sections: Section 2 presents some related theoretical approaches. Section 3 reviews the Android Programming Learning Assistance System containing automatic source code validation and online web platform. Section 4 explains the structure of learning materials in APLAS consisting of the lesson plan and package of learning materials. Section 5 delineates the fundamental concept of multiple Activities and multimedia resources in Android applications for interactive applications. Section 6 presents the implementation of *Multiple Activities* topic. Section 7 presents the implementation of *Multimedia Resources* topic. Section 8 presents the evaluation results of the two topics implementations. Section 9 discusses related findings based on evaluation results. Finally, Section 10 concludes this paper with future works.

2. Literature Review

This section presents some theoretical approaches related to the fundamentals of Android applications, test-driven development in Android applications, and related research on Android programming learning tools.

A. *Android Application Development*

Android is an open-source and Linux-based operating system that is specifically designed to be installed on smartphones and tablet devices[12], and also adjusts from low-end specifications to high-end specifications[13]. Currently, Android now belongs to Google. Due to the open-source license, many parties or companies use this operating system. To develop Android apps, some programming languages, such as Kotlin, Java, and C++, can be used to write code[14]. It is supplemented by various resources provided in terms of XML layout and data files, image files, raw data files, and so on. The Android SDK tools compile the source code along with any data and resource files into an APK, an Android package, which is an archive file with a '.apk' suffix[15].

B. *Tools for Learning Android Programming*

During the initial stages of the Android smartphone's popularity, Hanafi et al. discussed the development of a mobile application aimed at aiding the education of undergraduate students in Malaysia [16]. Their work highlighted essential elements of designing and implementing a learning system based on the Android platform.

The development of learning platforms for Android programming has been explored in various studies. Kang et al.[15] investigated educational methods for Android programming through the use of *Multi Android Development Tools*, incorporating MIT App Inventor and Eclipse. For beginners, MIT App Inventor's puzzle model is user-friendly, but creating fully functional applications necessitates using Eclipse. Additionally, Rekhawi et al.[17] designed a web-based intelligent tutoring system to teach Android application development. This system offers lessons on the fundamentals of Android programming, the basic user interface, and application design.

C. *Test-Driven Development Method*

Test-driven development (TDD) is a software development approach that relies on iterative, short development cycles[18]. This approach is a key component of the agile software development methodology. Within TDD, developers produce new code only when a test case has failed. The requirements outlined in the source code are converted into specific test cases within the test code. As depicted in Figure 1, the process begins with the programmer writing *test code*, even for minor functionalities. Subsequently, the necessary code is written solely to fulfill the test requirements, addressing any immediate test failures. The source code is iteratively refined until all tests are successfully passed [19]. In the context of programming education, this TDD model is employed to create an automated self-learning assistance system. This system provides learning support through feedback or comments that are automatically generated when a test fails, aiding in the learning process.

As shown in Figure 1, the programmer writes a *test code* first, even for implementing the smallest functionality. Then, he/she writes the code only to meet the test requirements, when he/she runs it and gets an obvious fail. The source codes are improved until all the tests are passed [19]. For learning programming, this TDD model is adopted to realize an automated assistance self-learning system. The learning assistance feature is represented by feedback or comments that can be generated automatically when a failed test occurs.

Various studies have implemented the TDD approach to offer automatic feedback, grading, and support within programming education systems. Funabiki et al.[20] developed a web-based platform known as the *Java Programming Learning Assistant System (JPLAS)*, designed for self-directed Java programming learning using the TDD methodology. JPLAS employs *JUnit* for unit testing of students' source codes, which enhances the learning experience in Java programming courses by facilitating student self-study and lightening the instructors' workload. In another study, Almeida et al.[21] shared their experiences in revamping an introductory lab course to teach functional programming concepts and sound software development practices through hands-on training.

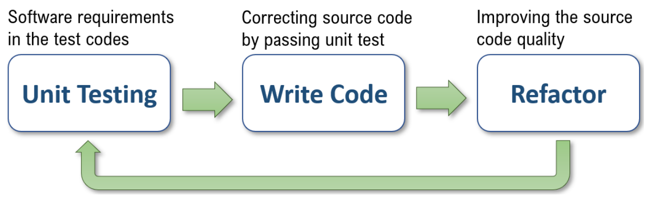

Software requirements in the test codes
Correcting source code by passing unit test
Improving the source code quality
Unit Testing
Write Code
Refactor

Figure 1. Process in test-driven development method.

D. *Automated Testing for Android Application*

To implement the Test-Driven Development (TDD) approach for Android applications, creating an automated testing process is crucial. This involves developing test cases within the test code derived from the source code or the design model, which act as pre-established benchmarks. Following this, the application is automatically tested using a specific tool or framework [22]. For Android, testing frameworks are divided into three scale categories: small (unit tests), medium (integration tests), and large (UI tests)[23].

Researchers have carried out a few studies on automated testing methods designed for Android applications. For example, Sadeh et al.[24] analyzed the effectiveness of of Robolectric which can enhance the speed of testing by offering essential tools designed for testing user interface code. Vásquez et al.[25] surveyed Android developers regarding their testing experiences. The purpose of this study was to gather data regarding the testing tool techniques used in the creation of Android applications. According to the results, JUnit is the most popular automated testing tool, followed by Roboelectic, Robotium, and Kochhar et al.[26] and Lin et al.[27]. Robolectric offers the Java Virtual Machine-based framework for unit testing by mimicking the Android execution environment on real devices or emulators, thus Robolectric operates much faster than other testing instruments[28].

3. Self-learning Assistance System for Android Programming

This section outlines the self-learning support system designed for Android programming, building upon insights gained from our earlier research on the Android Programming Learning Assistance System (APLAS).

A. *Overview of APLAS*

APLAS serves as a platform that fosters a self-directed learning environment for studying Android application development more autonomously[4], as described in Figure 2. It aids students in composing Java, XML, and Gradle DSL source codes within Android Studio by incorporating the TDD method into the software development process. By integrating guide documents and test codes for specific assignments, students are directed toward the development of Android applications. Additionally, test codes are used to validate the written source codes by executing them, thereby eliminating the need for teacher verification to confirm their accuracy.

B. Automatic Source Code Validation

In the APLAS system, the validation of answers was achieved through the implementation of test codes combining two automated testing tools: JUnit for unit testing[29], a widely adopted Java testing framework, and Robolectric for integration testing[30]. Robolectric, which emulates the Android application environment on the Java Virtual Machine (JVM), enables thorough testing by integrating components and generating Java objects for simulation. This approach allows JUnit to conduct rapid unit tests without the need for emulators or physical devices, streamlining the testing process and eliminating the time-consuming APK file production step.

The efficacy of Robolectric in integration testing has been corroborated in previous studies by Sadeh et al.[24] and Hussain et al[31]. A test case is a method in a test code that is noticed by the '@Test' annotation. Figure 3 shows a test code where an initiation method *'setup()'* containing the Robolectric building process is executed first to instantiate the 'activity' object. The assertion method is used to compare two values and determine the *Pass* or *Fail* result.

C. Online Web Platform

As an interactive learning system in Figure 2, APLAS provides an online platform that facilitates the distribution of learning materials and the collection and validation of student submissions via a web-based application[8], [32]. It employs a server-side validator program, implemented in Java using Gradle, which automatically compiles and tests submitted source codes in the background, using the Android SDK. The system proved effective in reducing teachers' workload and addressing primary challenges in the APLAS learning process.

4. Structure of Learning Materials

In APLAS, educational content (learning materials) is structured into various learning topics to cater to the diverse array of subjects within Android programming, where each learning topic provides lesson plans and a package of learning materials.

A. Lesson Plan

A lesson plan is a document prepared to make sure that a *learning topic* goal can be achieved. In APLAS, it consists of guide documents, supplement files, and test codes. It will make up the learning steps of students. As illustrated in Figure 4, the lesson plan for a learning topic contains:

1) Learning objectives

These objectives outline the key concepts and skills that students are expected to grasp or demonstrate by the conclusion of the learning experience.

2) Application for assignment

This refers to an Android application utilized as a case study for students during the development of their own Android applications. It encompasses the content necessary for understanding a specific learning topic.

3) Tasks

They are a set of step-by-step instructions for creating an entire application for an assignment. Every work is a basic assignment towards creating an Android application from scratch. Usually, students are allowed to create the UI layouts in order to complete the application development. The test codes for every task are designed to ensure that the *mandatory functions* are verified and that the UI components that are going to be used are complete. They do not examine other characteristics like font size, background color, or widget location. Students can also make the application better by including other features without compromising the required features.

B. Package of Learning Materials

Students use learning resources related to the learning topic to complete an Android application for assignment, as shown in Figure 4. Every one of them has a task correlation and helps students complete the project. They include the following to create a setting for self-learning:

1) Guide Documents

The guide documents are used to assist students accomplishing their programming assignments in APLAS. A document contains the description of the task, the learning goal, the list of learning resources, the hardware and software specifications, the explanation of the components of the Android application, the instructions to write the source codes, and the instructions to validate the source codes. Figure 5 shows a part of guide document that directs writing Java codes to define Android variables.

2) Test codes

Test codes are Java codes designed to help students write source codes and validate their accuracy using instructions found in guide papers. Once a task is completed, the student can use Android Studio to run the test codes to verify that the answer codes are valid. As seen in Figure 6, red alerts indicating faulty codes that require adjustments will be displayed if any tests fail. The test codes' reply will direct the student in making the necessary modifications. The green check, which indicates that all of the written answer codes are accurate, will appear if every test passes.

3) Supplement files

They are support files that must be added to the Android project before starting to write source codes. They consist of various file types, such as images, videos, drawable files, libraries, and APIs.

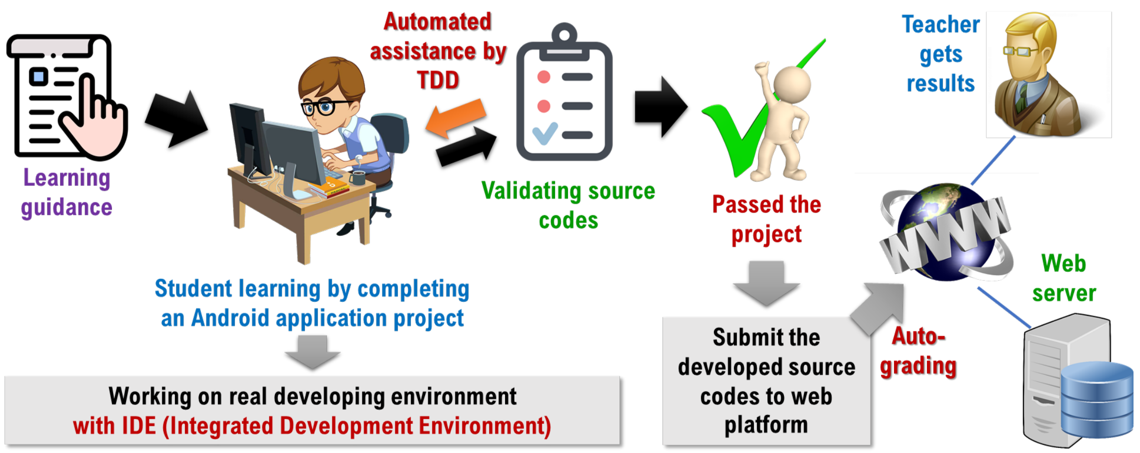
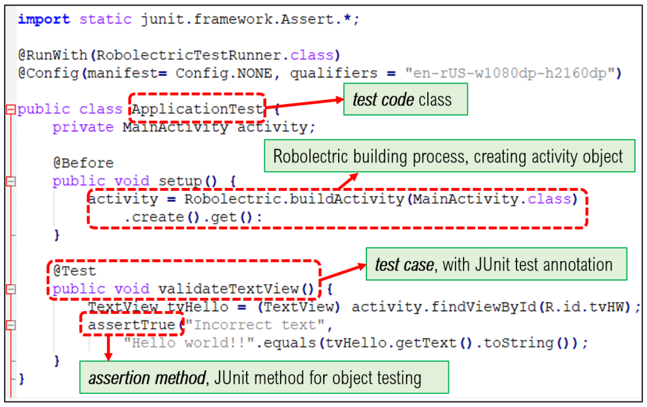

Learning guidance
Automated assistance by TDD
Validating source codes
Passed the project
Teacher gets results
Student learning by completing an Android application project
Working on real developing environment with IDE (Integrated Development Environment)
Submit the developed source codes to web platform
Auto-grading
Web server

Figure 2. Learning model of student and teacher in APLAS.



```
import static junit.framework.Assert.*;

@RunWith(RobolectricTestRunner.class)
@Config(manifest= Config.NONE, qualifiers = "en-rUS-w1080dp-h2160dp")

public class ApplicationTest {
    private MainActivity activity;

    @Before
    public void setup() {
        activity = Robolectric.buildActivity(MainActivity.class)
            .create().get():
    }

    @Test
    public void validateTextView() {
        TextView tvHello = (TextView) activity.findViewById(R.id.tvHW);
        assertTrue("Incorrect text",
            "Hello world!!".equals(tvHello.getText().toString());
    }
}
```

test code class
Robolectric building process, creating activity object
test case, with JUnit test annotation
assertion method, JUnit method for object testing

Figure 3. A Robolectric-based test code.

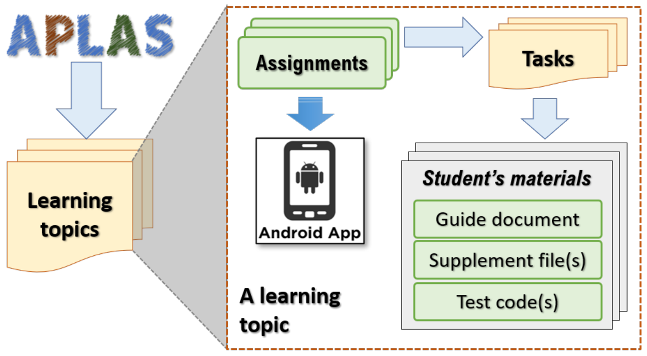


Figure 4. Structure of learning materials.

5. Multiple Activities and Multimedia Resources in Android Applications

This section reviews the components to build multiple Activities with multimedia integration in Android application.

A. *Overview*

The user interface (UI), in conjunction with multiple activities and multimedia resources, plays a crucial role in shaping both the user experience and the operational functionality of an Android application. The UI functions as the application's visual representation, guiding user interactions with its features and content. It is essential to design an intuitive and visually appealing layout that enhances user engagement and usability. Multiple activities facilitate the organization of application functions into separate screens or segments, thus improving navigation and task execution. The integration of multimedia elements, including images, videos, and audio files, allows developers to enrich the application's content and increase user interaction, creating a more immersive experience. The effective combination of a thoughtfully designed UI, well-implemented multiple activities, and engaging multimedia resources contributes to the development of an interactive application.

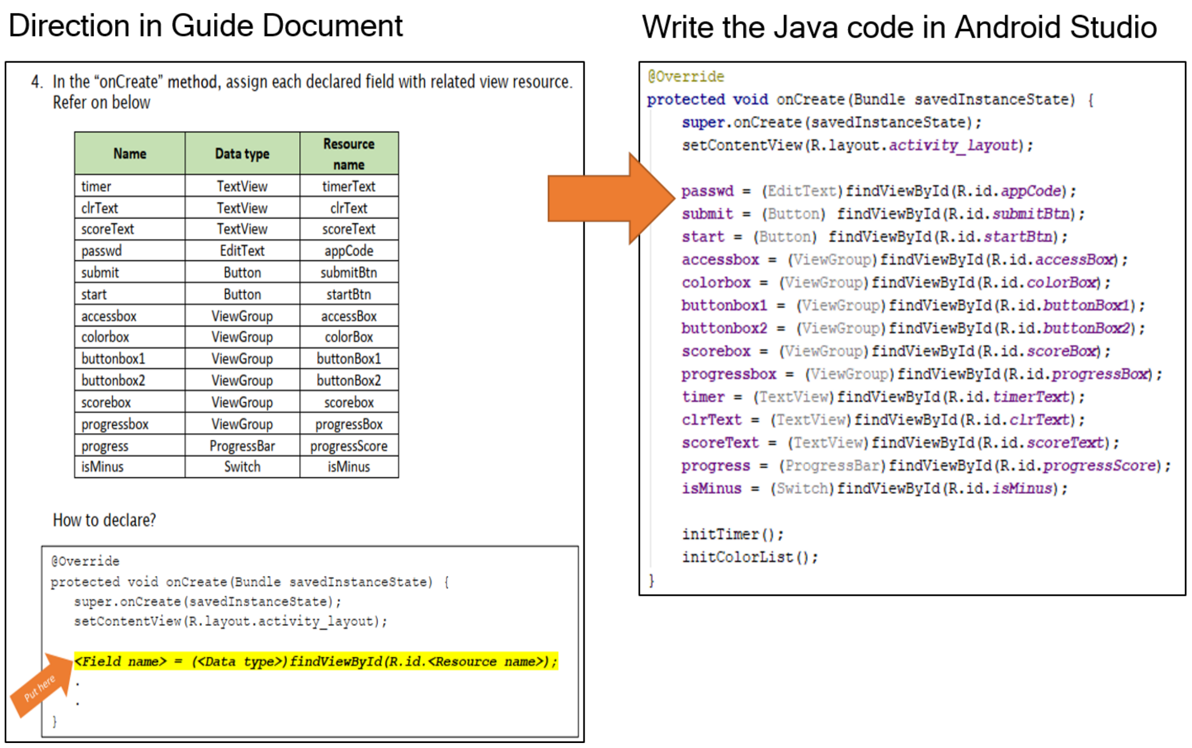
B. *Multiple Activities*

The user interface (UI) functions as the principal means of user interaction in Android applications. Currently, Android provides a range of components for crafting rich and aesthetically pleasing UIs, encouraging developers to build applications with progressively intricate features. Many modern Android applications utilize multiple screens to handle these complex functionalities, thus incorporating multiple Activities. Chrome Mobile and Google Translate are the two well-known Android applications which employ multiple Activities.

To effectively develop Android applications with multiple Activities, programmers must be proficient in using three key classes within the Android project:

1) *Activity*

In Android applications, the Activity class is fundamental for launching the application and displaying the UI[33]. It functions similarly to the *'main()'* method in Java, which is executed first by the Java Virtual Machine to initiate



Direction in Guide Document

4. In the "onCreate" method, assign each declared field with related view resource. Refer on below

Name	Data type	Resource name
timer	TextView	timerText
clrText	TextView	clrText
scoreText	TextView	scoreText
passwd	EditText	appCode
submit	Button	submitBtn
start	Button	startBtn
accessbox	ViewGroup	accessBox
colorbox	ViewGroup	colorBox
buttonbox1	ViewGroup	buttonBox1
buttonbox2	ViewGroup	buttonBox2
scorebox	ViewGroup	scorebox
progressbox	ViewGroup	progressBox
progress	ProgressBar	progressScore
isMinus	Switch	isMinus

How to declare?

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_layout);

   <Field name> = (<Data type>)findViewById(R.id.<Resource name>);
   .
   .
}
```

Write the Java code in Android Studio

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_layout);

    passwd = (EditText)findViewById(R.id.appCode);
    submit = (Button) findViewById(R.id.submitBtn);
    start = (Button) findViewById(R.id.startBtn);
    accessbox = (ViewGroup)findViewById(R.id.accessBox);
    colorbox = (ViewGroup)findViewById(R.id.colorBox);
    buttonbox1 = (ViewGroup)findViewById(R.id.buttonBox1);
    buttonbox2 = (ViewGroup)findViewById(R.id.buttonBox2);
    scorebox = (ViewGroup)findViewById(R.id.scoreBox);
    progressbox = (ViewGroup)findViewById(R.id.progressBox);
    timer = (TextView)findViewById(R.id.timerText);
    clrText = (TextView)findViewById(R.id.clrText);
    scoreText = (TextView)findViewById(R.id.scoreText);
    progress = (ProgressBar)findViewById(R.id.progressScore);
    isMinus = (Switch)findViewById(R.id.isMinus);

    initTimer();
    initColorList();
}
```

Figure 5. Example of guidance to write Java code.

Figure 6. Test result on Android Studio.

a Java program. For Android applications, the Android system begins by executing code within the Activity class via specific callback methods. Moreover, the Android SDK provides the Activity class to manage the logic of a user interface screen. In Android Studio, this must be written in either Java or Kotlin[34]. Typically, a single Activity handles one UI screen. Figure 7 illustrates the Google Translate application, which uses four activities.

In an Android application, every Activity has a unique lifecycle. Developing an Android application with several Activities requires careful consideration on how to handle each Activity's lifecycle. Activities in the Android system are arranged in an Activity stack, with the top Activity being shown on the device's screen. When an Activity is launched, the previous Activity remains underneath it and resurfaces once the new Activity is deleted. The newly launched Activity is positioned at the top of the stack and becomes the foreground Activity. There are six callbacks or lifecycle methods in the Activity class: *onCreate*, *onDestroy*, *onStart*, *onResume*, *onStop*, and *onPause*.

2) *Intent*

The Intent class [35] is used to request actions from components of other applications. It primarily serves three functions for intercomponent communication: starting an Activity, initiating a service, and sending a broadcast. The most crucial function, particularly for applications with multiple Activities, is the initiation of an Activity, which allows users to navigate between different screens through interactions such as clicking, swiping, and touching. Furthermore, the Intent class also allows other applications or services to initiate Activities.

To initiate an Activity, an Intent must be created by specifying the target Activity and including relevant variables, known as *Extras*, as illustrated in Figure 8. Extras are variables comprising a name and a value that are transmitted as messages by the Intent to the target Activity. The 'putExtra()' or 'putExtras()' methods can be used to add the Extras. Then, by calling 'startActivity()' method that has been passed by the Intent object, the target Activity will be started.

As illustrated in Figure 7, there are two types of Intent:

1) *Explicit Intent* facilitates communication with a specific component or Activity within the same application by requiring the name of the target Activity or component. For example, as shown in Figure 8, the target Activity's class name must be specified. It can

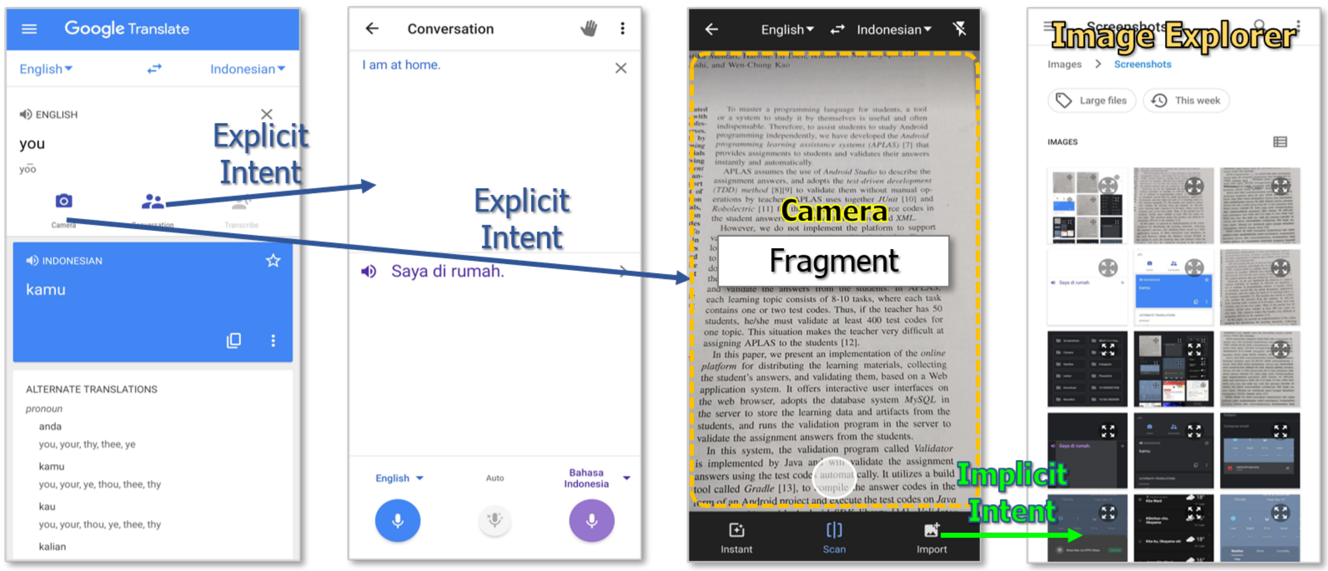


Figure 7. Google Translate application with four Activities

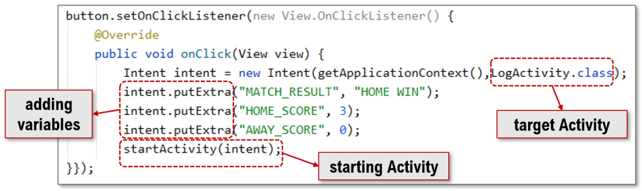

```
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        Intent intent = new Intent(getApplicationContext(),LogActivity.class);
        intent.putExtra("MATCH_RESULT", "HOME WIN");
        intent.putExtra("HOME_SCORE", 3);
        intent.putExtra("AWAY_SCORE", 0);
        startActivity(intent);
}});
```

Figure 8. Source code of using Intent to start an Activity

also be used to start a service, such as background image downloading.

2) *Implicit Intent* is used to start a component from another application without specifying its name, but by defining a general action. For example, to open a camera screen, an Implicit Intent requests the Camera application, which handles this function by default. Android includes several default applications for tasks like taking photos, exploring files, mapping, and browsing the web.

3) *Fragment*

A section of an application's user interface that can be included into an activity is called a fragment[36]. It simplifies reuse between activities by encapsulating the UI layout and logic. By enabling Fragments to dynamically populate UI parts, this technique improves UI modularity and reusability. As demonstrated in Figure 7, the third Activity, for instance, has a Fragment that has a camera in it. Because a Fragment is modular, it may be used by more than one Activity, which means that programmers don't have to write several Activities for similar tasks. Similar to an Activity, a Fragment consists of an XML UI layout file, a Java program acting as the Fragment controller, and its own lifecycle. An Activity's UI layout has to use *'FrameLayout'* to set aside space for a Fragment in order to construct a dynamic UI screen.

C. *Multimedia Resources*

When creating Android applications, multimedia assets are essential since they significantly improve user interaction, interface design, and overall experience. The application's visual appeal and intuitiveness are enhanced with high-quality graphics, animations, and videos. Additionally, audio elements serve as crucial feedback and can support voice-activated functions. These components contribute significantly to accessibility as well, which increases the application's usefulness. Incorporating multimedia elements into an application guarantees that it not only performs effectively but also draws and holds the attention of users through interactive learning and efficient content delivery. The following multimedia components are applicable:

1) *VideoView*

A higher-level View that offers a direct method of displaying video material inside an application is called *VideoView*. It makes it easier to integrate and manage video playback in user interfaces that load videos from several sources (like resources or content providers). To display the movie, the VideoView class combines a *SurfaceView* with a media player (*MediaPlayer* class). It incorporates many of the fundamental needs needed to play a video in an Android application.

2) *YouTubePlayer API*

Google provides a set of functions in the *YouTubePlayer API* that allow YouTube videos to be played in Android applications[37]. It offers library methods for playing, loading, and adjusting the playback of YouTube videos. Several classes of 'YouTubePlayer.Provider', like *YouTubePlayerFragment* or *YouTubePlayerView*, must be used to obtain an instance for this playback in order to embed a YouTube player video on the application's user interface. Programmatically managing YouTube videos—for example, by using *play*, *pause*, or *seek* to a certain point in the loaded video—is possible using this API.

3) *Animations*

The Android platform offers a variety of APIs for incorporating animations into applications. Animations serve as visual indicators within the user interface during application runtime [38]. They are commonly utilized for transitions between screens, interactions with various widgets, and displaying new content on the screen. Additionally, animations enhance the overall aesthetic and user experience of the application's interface. Android includes several animation types to increase interactivity, such as *Drawable Animation*, *Shared Element Transition*, and *Transition Animation*.

6. Implementation of Multiple Activities Topic

This section outlines the implementation of the Multiple Activities topic, which includes the learning objectives, application for assignments, and associated tasks complete with test codes.

A. *Learning Objectives*

Developing Android applications with multiple Activities requires students to master several key components. The focus of Multiple Activities topic is on exploring the development of an Android application that features multiple Activities, utilizing Intent and Fragment for effective management. Table I outlines six learning objectives that guide students in mastering these components to create Android applications with multiple Activities.

B. *Application for Assignment*

The SoccerMatch application has been created as the assignment for the Multiple Activities topic. This application not only emphasizes the use of multiple Activities but also showcases a variety of widgets and Android features, including *ListView*, *CardView*, *Chronometer*, *Handler*, *RecyclerView*, and *AlertDialog*.

1) *Mandatory Functions*

The SoccerMatch is a real-time reporting application tailored for SoccerMatches. Its primary purpose is to document significant events occurring in a match involving two teams (the home team and the away team). Data for each team include details such as the team name, logo, and player names. The essential match events that must be captured include goals, yellow card infractions, and red card penalties. Beyond these core functionalities, students have the freedom to enhance the application by incorporating additional features aligned with their unique ideas and creativity.

2) *Application Specifications*

Every significant event that occurs during a soccer match is captured by the SoccerMatch app, including player

TABLE I. Six learning objectives in Multiple Activities topic.

objective (no.:name)	**description**
LO1: Application Resources	Students have skills to define essential resources in an Android project, encompassing colors, strings, styles, and various other assets.
LO2: Application Activity	Students have capability to handle multiple Activities by employing Activity, Intent, and Fragment classes.
LO3: User Interface	Students are able to create UI layouts for Android applications using a variety of widgets to build the application's user interface.
LO4: Java Class	Students understand to handle and work with principal Java classes such as Array, Runnable, and Handler.
LO5: Android Class	Students have adapted in using fundamental Android classes, including ListView, and PopupMenu.
LO6: Event Listener	Students are able to apply event listeners to enable user interactions with the application.

names, team logos, goals, yellow and red cards, and player counts. As seen in Figures 9 and 10, this application uses three Activities with one Explicit Intent and one Fragment for multiple activities learning. By completing the eight objectives, this application introduces a number of widgets and Android technologies, including *Chronometer*, *ListView*, *RecyclerView*, *CardView*, *Handler*, and *Runnable*. With Java programming, the students can construct the Soccer Match Android application by completing the eight tasks.

C. *Implemented Android Components*

The SoccerMatch application implements 13 main components with some common components that are organized:

1) *Application Resources*

Table II presents the four implemented application resources in SoccerMatch with their correlation with learning objectives.

2) *Main Classes for Multiple Activities*

Table III presents the four main classes implemented for multiple activities in SoccerMatch with their correlation with learning objectives.

3) *Widgets*

Table IV presents the five implemented widgets in SoccerMatch with their correlation to learning objectives. They are newly introduced widgets on this topic.

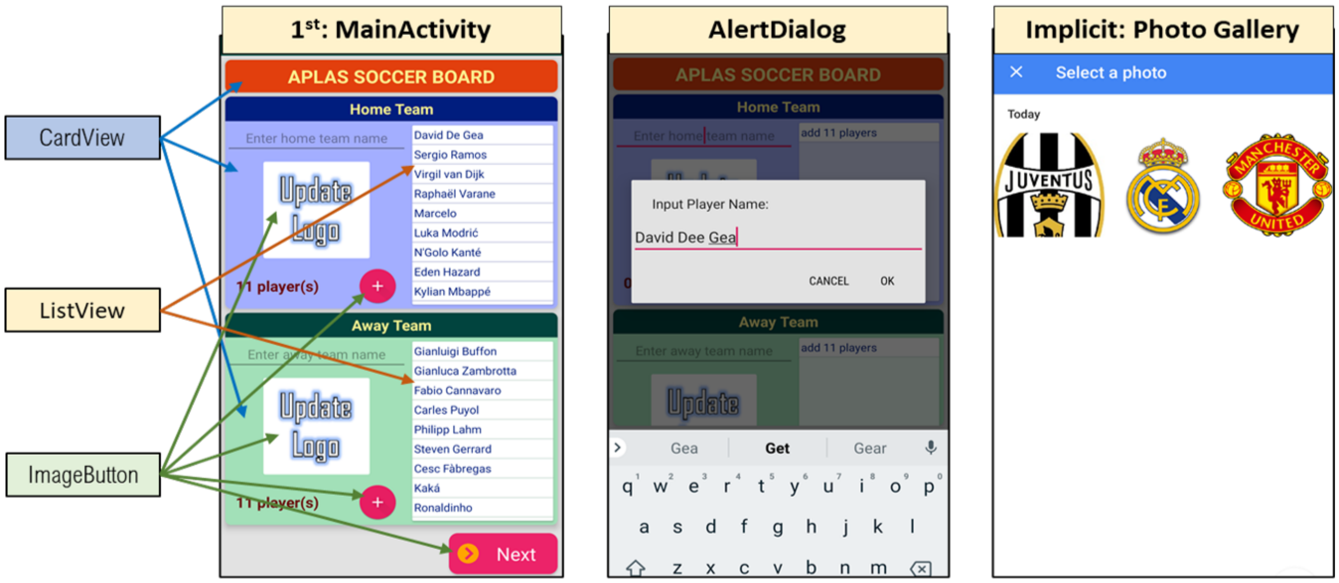


Figure 9. First *Activity* of *SoccerMatch* application.

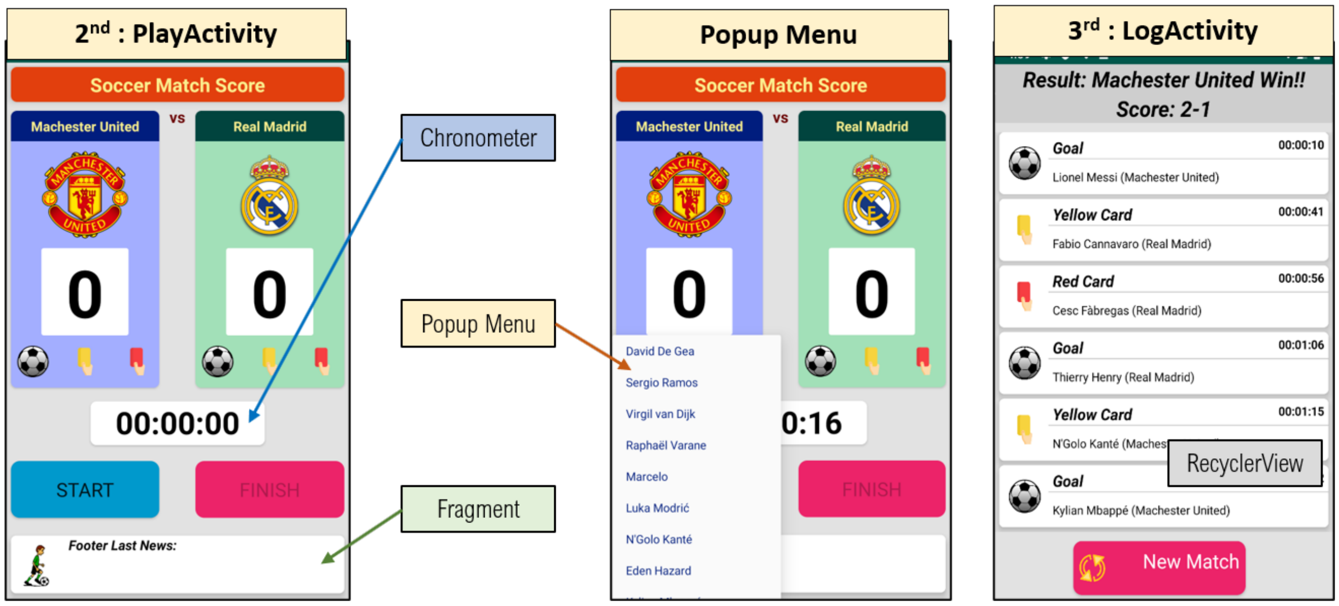


Figure 10. Second and third *Activities* of *SoccerMatch* application.

D. Tasks

In the context of the *Multiple Activities* topic, eight tasks have been designed to assist students in the step-by-step development of the SoccerMatch application. Table V outlines these eight tasks and their relationships to the learning objectives.

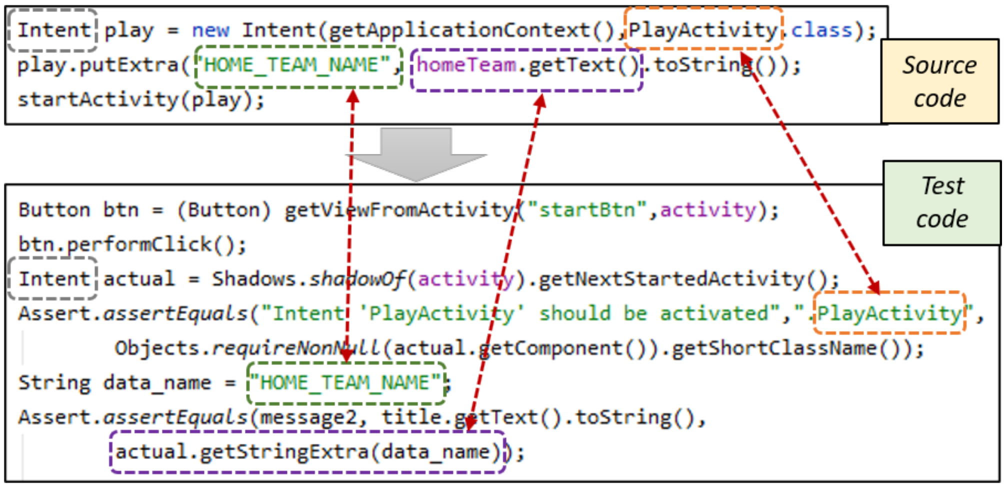
Figure 11 depicts the source code used to launch PlayActivity via Intent, as well as to pass data by adding an *Extra* when the '*Next*' button is clicked. The testing code is designed to confirm that the Activity that opens is PlayActivity and that the string value associated with 'HOME_TEAM_NAME' from the Extra data serves as the title for PlayActivity.

7. Implementation of Multimedia Resources Topic

This section outlines the development of the educational materials pertaining to the *Multimedia Resources* topic.

A. Learning Objectives

This topic provides six learning objectives in Table VI for developing an Android application that applies multimedia resources.


```
Intent play = new Intent(getApplicationContext(),PlayActivity.class);
play.putExtra("HOME_TEAM_NAME", homeTeam.getText().toString());
startActivity(play);
```

```
Button btn = (Button) getViewFromActivity("startBtn",activity);
btn.performClick();
Intent actual = Shadows.shadowOf(activity).getNextStartedActivity();
Assert.assertEquals("Intent 'PlayActivity' should be activated","PlayActivity",
        Objects.requireNonNull(actual.getComponent()).getShortClassName());
String data_name = "HOME_TEAM_NAME";
Assert.assertEquals(message2, title.getText().toString(),
        actual.getStringExtra(data_name));
```

Figure 11. Source code to use Intent and test code to confirm its correctness.

TABLE II. Four application resources in Multiple Activities topic.

no.	component(s) & description	LO
1	**String resource**: a collection of strings used for application.	LO1
2	**Color resource**: a collection of used colors for application where each of them can be defined as a hexadecimal value.	LO1
3	**Style resource**: a definition of UI style using XML that can be applied to Activities or widgets.	LO1
4	**Drawable resource**: a general definition of graphics that can be shown on screen. This component uses several bitmap graphics and shape drawables to make rounded buttons.	LO1

TABLE III. Four main classes in Multiple Activities topic.

no.	component(s) & description	LO
1	**Activity**: three Activities are applied in this application, including *MainActivity*, *PlayActivity*, and *LogActivity*	LO2
2	**Intent**: explicit Intents are used to open the three Activities.	LO2
3	**Fragment**: one Fragment is applied namely *FooterFragment* that is used by *PlayActivity*.	LO2
4	**UI Layout**: six UI layouts are applied to define the layout for three *Activities*, a *Fragment*, a *ListView*, and an *AlertDialog*.	LO3

TABLE IV. Five new widgets in Multiple Activities topic.

no.	component(s) & description	LO
1	**CardView**: a card-shaped UI container with a drop shadow called elevation and corner radius.	LO3
2	**RecyclerView**: a UI container to display large sets of dynamic data. The display of data items can be dynamically defined as UI layout.	LO3
3	**ListView**: a widget to display a vertical list of several items. The list items are managed by Adapters as an array of data sources.	LO3
4	**Chronometer**: a subclass of TextView that can be used for showing clock or simple timer.	LO3
5	**ImageButton**: a button with an image foreground (instead of text) that can be pressed or clicked by the user.	LO3

B. Application for Assignment

The *AnimalTour* application is designed for the assignments in this topic.

1) Mandatory Functions

The *AnimalTour* application serves as an informative platform that explores the classifications of animals across the globe. It encompasses six main categories: mammals, birds, reptiles, amphibians, fish, and invertebrates. For each category, users can access a variety of related images, video files, YouTube links, and informative text. The application offers an interactive user interface, enriched with multimedia elements, animations, dynamic transitions, and gesture controls, achieved through a combination of different Activity classes, event listeners, and popups. In addition to these foundational features, students are encouraged to innovate by integrating extra functionalities that reflect their personal ideas and creative visions.

2) Application Specification

To integrate multimedia features, the application utilizes four key Activities, as depicted in Figure 12. Upon selecting the invertebrates category within MainActivity, users are directed to InvertActivity, which showcases a GridView. When a user taps on an image of an invertebrate in InvertActivity, the SubInvertActivity emerges, utilizing a

TABLE V. Eight tasks in Multiple Activities topic.

no.	description	LO
1	starting an SoccerMatch project and configuring necessary resources (XML and Gradle)	LO1
2	designing UI (XML) for first Activity (MainActivity)	LO3
3	designing UI (XML) for second Activity (PlayActivity)	LO3
5	designing UI (XML) for additional layouts (LogActivity and Fragments)	LO3
6	developing logic (Java/Kotlin) for first Activity (MainActivity)	LO2,LO5,LO6
7	developing logic (Java/Kotlin) for second Activity (PlayActivity)	LO2,LO4, LO5,LO6
8	developing logic (Java/Kotlin) for last Activity (LogActivity)	LO4,LO5, LO6

TABLE VI. Five learning objectives in Multimedia Resources topic.

objective (no.:name)	description
LO1: Application Resources	Students are able to utilize application resources within an Android project.
LO2: UI Components	Students comprehend the process of defining layouts for multimedia user interfaces by incorporating diverse UI components.
LO3: Multimedia Components	Students have skills in employing multimedia components to present images, videos, and YouTube content.
LO4: Animations	Students understand the techniques to implement animations in Android applications.
LO5: User Events	Students become adept at managing various user events such as touch and swipe interactions.

shared element transition to create a zoom effect during the transition between screens.

C. Implemented Android Components

The SoccerMatch application implements 13 main components with some common components that are organized:

1) Application Resources

Table VII presents the six implemented application resources in SoccerMatch with their correlation to learning objectives.

TABLE VII. Six application resources in Multimedia Resources topic.

no.	component(s) & description	LO
1	**String resource**: a collection of strings used for application.	LO1
2	**Color resource**: a collection of used colors for application where each of them can be defined as a hexadecimal value.	LO1
3	**Style resource**:a definition for the UI style using XML that can be applied to Activities or widgets.	LO1
4	**Drawable resource**: a general definition of graphics that can be shown on screen.	LO1
5	**Mipmap resource**: a type of drawable resource that is specifically intended for different versions of an application's launcher icon.	LO1
6	**Anim resource**: animation files that define a series of changes or transformations that can be applied to UI components over time.	LO1

2) Main Classes for Multimedia Resources

Table VIII presents the seven main classes implemented for multimedia resources in AnimalTour with their correlation with learning objectives.

TABLE VIII. Seven main classes in Multimedia Resources topic.

no.	component(s) & description	LO
1	**Activity**: four Activities are applied in this application, including *MainActivity*, *MediaActivity*, *InvertActivity*, and *SubInvertActivity*.	LO2
2	**DataAdapter**: a type of adapter used to bind data animals to views, specifically in RecyclerView.	LO2
3	**ItemMoveCallback**: a class typically used in conjunction with RecyclerView in Android development.	LO2
4	**GridLayout**: a layout manager that allows developers to arrange UI components in a two-dimensional grid.	LO2
5	**Drawable Animation**: a type of animation that relies on a series of drawable resources (animal images) displayed in sequence, similar to a traditional flipbook animation.	LO4
6	**Transition Animation**: a visual effect applied when the user navigates from one screen to another within an Android application (MainActivity to Media Activity).	LO4
7	**Shared Element Transition**: a feature to provide a smooth visual transition between two activities or fragments by animating a common UI element.	LO4

3) Widgets

Table IX presents the six widgets implemented in SoccerMatch with their correlation with the learning objectives.

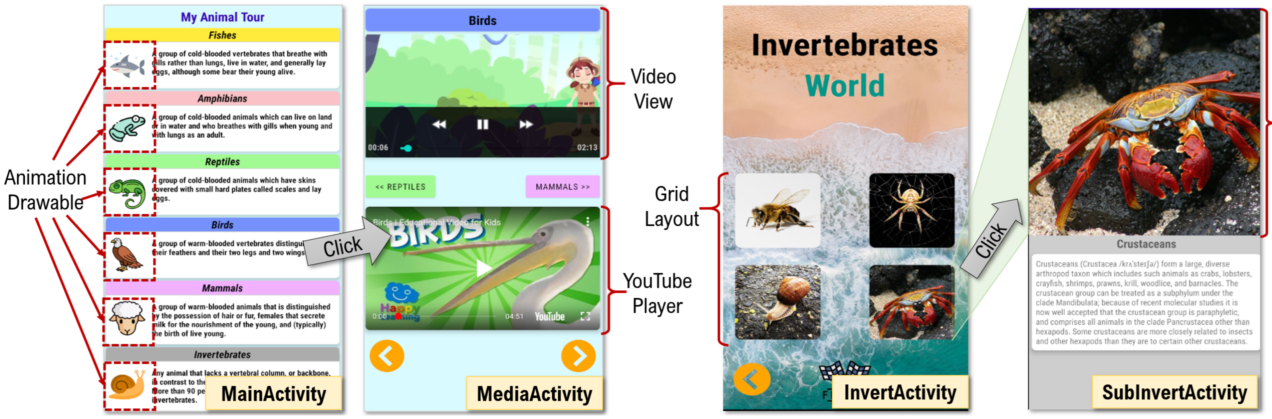


Figure 12. Four Activities of *AnimalTour* application.

They are newly introduced widgets on this topic.

TABLE IX. Six new widgets in Multiple Activities topic.

no.	component(s) & description	LO
1	**VideoView**: one ViewView is used to display and play related video file with mp4 format on MediaActivity as raw resources.	LO2
2	**YouTubePlayerFragment**: a part of the YouTube Android Player API, which allows developers to embed and control YouTube videos directly within their Android applications using a fragment-based approach.	LO2
3	**CardView**: a card-shaped UI container with a drop shadow called elevation and corner radius.	LO2
4	**RecyclerView**: a UI container to display large sets of dynamic data. The display of data items can be dynamically defined as UI layout.	LO2
5	**VideoFlipper**: a handy UI component that allows application to switch between different views with animation.	LO3
6	**ImageView**: a view that is used to display images or drawables.	LO2

D. Tasks

Under the topic of Multimedia Resources, it is essential to complete all eight tasks to develop the application in alignment with the five learning objectives outlined in Table X.

Figure 13 shows the source code to verify that the defined transition animaton working well when moving to another Activity. Also, the last function defines verfication code to check the defined transition animation working well when entering an Activity.

8. Evaluation

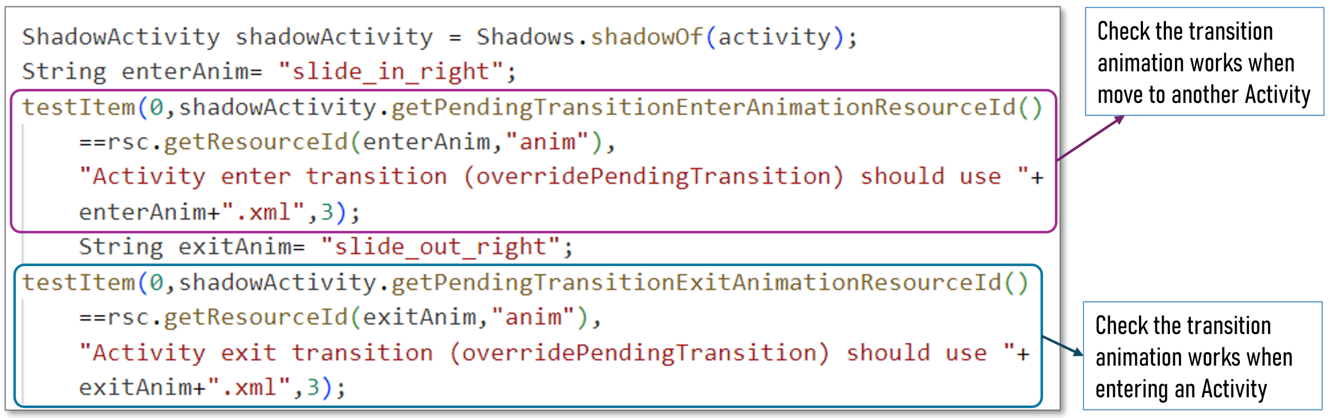
Evaluation of the implementation of both the Multiple Activities topic and the Multimedia Resources topic is discussed in this section.

TABLE X. Eight tasks in Multimedia Resources topic.

no.	description	LO
1	starting an AnimalTour project and configuring necessary resources (XML and Gradle)	LO1
2	designing UI for first Activity (MainActivity)	LO2
3	designing UI for second Activity (MediaActivity)	LO2,LO3
4	designing UI for third Activity (InvertActivity)	LO2
5	designing UI for last Activity (SubInvertActivity)	LO2,LO3
6	developing logic for first Activity (MainActivity)	LO4,LO5
7	developing logic for second Activity (MediaActivity)	LO3,LO4,LO5
8	developing logic for third Activity (MediaActivity) and last Activity (SubInvertActivity)	LO3,LO4

A. Evaluation Scenario

In a comprehensive evaluation strategy aimed at gauging proficiency in the Multiple Activities and Multimedia Resources topics, a cohort of 50 students from the Information Technology Department at an esteemed university in Indonesia embarked on a challenging academic endeavor. Equipped with foundational knowledge of the Android Studio platform, these students were primed to showcase their understanding and application of key concepts in Android programming. The class conducted lab featuring computers with Windows 10 OS, Intel Core i5-3470 3.2GHz CPUs, and 8GB of RAM. Over a structured timeline of three days, students solve the two topics sequentially, then they comprehensively solved all tasks in sequence. Once the students successfully tackled the assignments, the next



```
ShadowActivity shadowActivity = Shadows.shadowOf(activity);
String enterAnim= "slide_in_right";
testItem(0,shadowActivity.getPendingTransitionEnterAnimationResourceId()
    ==rsc.getResourceId(enterAnim,"anim"),
    "Activity enter transition (overridePendingTransition) should use "+
    enterAnim+".xml",3);
    String exitAnim= "slide_out_right";
testItem(0,shadowActivity.getPendingTransitionExitAnimationResourceId()
    ==rsc.getResourceId(exitAnim,"anim"),
    "Activity exit transition (overridePendingTransition) should use "+
    exitAnim+".xml",3);
```

Check the transition animation works when move to another Activity
Check the transition animation works when entering an Activity

Figure 13. Source code to verify the transition animation works.

step involved submitting their source codes for evaluation on a specialized server. In the event of a student receiving a ”FAILED” result from the server, they were granted an opportunity to address any errors and resubmit their code for a second chance at assessment.

B. *Results on Multiple Activities Topic*

The evaluation of Multiple Activities topic resulted in the following aspects:

1) *Students' Solving Assignments*

Out of the initial group of 50 students, a notable achievement was observed as 90% of the cohort, numbered 45 students, successfully navigated and completed the Multiple Activities topic within the three allocated days, demonstrating a high level of proficiency in this area. The ability of these students to effectively tackle the challenges posed by this topic in their first attempt indicates a strong grasp of the concepts and skills involved in the development of Android applications.

The remaining five students who required a second opportunity to tackle the Multiple Activities topic, showcased commendable growth and determination by successfully resolving the tasks within an additional three-day period. This not only indicates their resilience and dedication, but also highlights the effectiveness of the learning process in enabling students to enhance their proficiency in developing more intricate Android applications.

2) *Solving Time Results*

Table XI provides the minimum, average, and maximum times students took to complete each task in the Multiple Activities topic. The time required to solve a single task varies widely, from 6 to 150 minutes. The total time to complete all tasks ranges from 89 to 660 minutes, with an average time of 174 minutes.

3) *Failed Task Results*

Table XII outlines the difficulties experienced by five students during their attempts. Specifically, two of them had trouble with task 1, which focused on setting up the string resource. Meanwhile, three students encountered issues with tasks 2 and 5, both of which pertain to layout design. Additionally, one student faced challenges in writing Java code, especially in utilizing Intent to launch LogActivity. Although there were errors in resource configuration and layout design, these issues are relatively minor, as the application remains functional despite them.

TABLE XI. Solving time of each task in Multiple Activities topic.

task no.	fastest	average	longest
1	7.5 minutes	15.86 minutes	32 minutes
2	9 minutes	19.72 minutes	55 minutes
3	9 minutes	19.48 minutes	60 minutes
4	6 minutes	13.76 minutes	62 minutes
5	7 minutes	17.85 minutes	58 minutes
6	16 minutes	31.75 minutes	130 minutes
7	15 minutes	26.84 minutes	115 minutes
8	14 minutes	24.55 minutes	100 minutes

4) *Task Difficulties*

The average amount of time students take to complete each task is used to determine how tough it is. The tasks are divided into three categories: simple, medium, and difficult, as indicated in Table XI. With an average completion time of less than 15 minutes, assignment 4 falls into the easy category. Its basic needs center around building the layout for LogActivity. Tasks 1, 2, 3, and 5 fall into the medium group; their typical completion times range from 15 to 20 minutes. The assignment begins with Task 1, with increasingly complex layout designs found in the remaining tasks. The difficult category includes tasks 6, 7, and 8, all of which require over 20 minutes to complete as they involve developing three Activities using Java.

TABLE XII. Most occurred failed messages.

task	#std	descriptions
1	2	**message**: Not valid value of app_name. **params**: expectation <SoccerMatch >, but has <MY SOCCER GAME>
2	2	**message**: The scaleType (id=addHomePlayer) in AppCompatImageButton must be FIT_END. **params**: expectation <FIT_END>, but has <FIT_CENTER>
2	1	**message**: Item text in AppCompatTextView (id=mainTitleTxt) is incorrect. **params**: expectation <APLAS SOCCER [BOARD]>but has <APLAS SOCCER [MANAGER]>
5	1	**message**: The layout of 'fragment_footer.xml' needs to be LinearLayout **params**: expectation <[Linear]Layout>but has:<[Grid]Layout>
8	1	**message**: an Intent can't reach Activity **params**: { cmp=.LogActivity (must have extras) }

TABLE XIII. Task difficulty levels in Multiple Activites topic.

task no.	level	time to solve
4	easy	10-14 minutes
1, 2, 3, 5	medium	15-20 minutes
6, 7, 8	difficult	21-35 minutes

5) *Students' Feedback*

Upon submission of the answers, the students collected comments on the Multiple Activities topic, and no negative comments were reported.

Upon submission of the answers, the students collected comments on the Multiple Activities topic and no negative comments were reported.

1) **Positive feedback**:
   - *useful platform to study Android programming*
   - *excellent e-learning tool*
   - *praising can pass assignments*
   - *success to pass all tasks*
2) **Suggestive feedback**:
   - *must be attentive in tasks 2-5 designing UI,*
   - *getting hard when reaching tasks 6-8*
   - *configuring Gradle makes error*
   - *guidance didn't explain RestartBtn creation.*
   - *constructing RecyclerView with Adapter is problem*
   - *guidance didn't notice few styling requirements.*

The notable comments serve as essential feedback from the students. They indicated that most failures were due to errors in designing the user interface, which requires careful attention to the guide documents. Furthermore, the students confirmed that tasks 6, 7, and 8 are particularly challenging to solve.

6) *Developed Applications*

A student-developed SoccerMatch application is shown in Figure 14; it has a different user interface than Figures 9 and 10. Although it wasn't stated in the guide docs, this student developed a method for entering each player's jersey number using an AlertDialog. Some students have added functions including a penalty event function, a venue information function, personalized icons and graphics, and a welcome screen.

C. *Results on Multimedia Resources*

The evaluation of Multimedia Resources topic resulted in the following aspects:

1) *Students' Solving Assignment*

The evaluation revealed that all 50 students successfully solved the Multimedia Resources topic on their first attempt, achieving a 100% success rate within one week. This universal proficiency underscores the students' ability to comprehend and apply the principles of multimedia resource integration in Android applications, showcasing their mastery of more sophisticated and challenging aspects of the curriculum. The result is particularly noteworthy given the complexity of the subject matter, which often poses significant challenges even for seasoned developers.

2) *Solving Time Results*

Students' needed times to finish each activity are broken down into minimum, average, and maximum values in Table XIV. The time required to finish each job ranged from 82 to 510 minutes, with an average of 163 minutes. job lengths varied from 5 to 120 minutes. After completing these four topics, students improved their knowledge of the guide papers and their ability to program Android using Android Studio.

TABLE XIV. Solving time of each task in *Multimedia Resources* topic.

task no.	fastest	average	longest
1	7.5 minutes	17.46 minutes	32 minutes
2	9 minutes	18.17 minutes	45 minutes
3	9 minutes	17.88 minutes	42 minutes
4	6.5 minutes	16.86 minutes	42 minutes
5	5.5 minutes	15.92 minutes	33 minutes
6	16 minutes	28.16 minutes	110 minutes
7	14 minutes	26.16 minutes	100 minutes
8	13 minutes	21.24 minutes	85 minutes

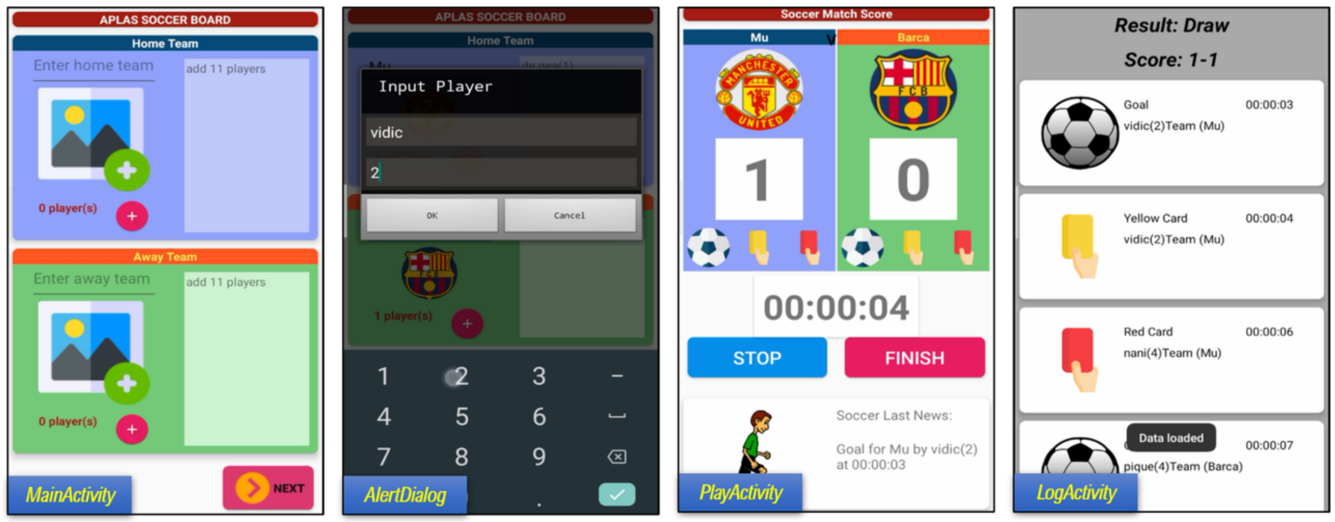


Figure 14. *SoccerMatch* application interfaces by a student.

3) *Task Difficulties*

The average amount of time students take to finish an assignment can be used to determine how tough it is. Tasks are divided into easy and tough categories, as indicated in Table XV, as explained in Table XIV. The first five tasks at the easy level, which concentrated on starting the Android project and creating the user interface, were finished in an average of less than 20 minutes. On the other hand, tasks 6 through 8 in the challenging level took an average of more than 20 minutes and focused on creating four Java activities.

TABLE XV. Task difficulty levels in Multimedia Resources topic.

task no.	level	time to solve
1,2,3,4,5	easy	15-20 minutes
6, 7, 8	difficult	21-30 minutes

4) *Students' Feedback*

Students were asked to provide feedback on the topic after submitting their answers. No negative comments were received. Nonetheless, we will revise the guide documents to enhance the learning experience for students, taking their feedback into account.

1) **Positive feedback**:
   - *useful platform to study Android programming,*
   - *success to pass all tasks,*
   - *delighted making Android app,*
   - *the application has excellent animations.*
2) **Suggestive feedback**:
   - *creating layout provides minimum guidance.*
   - *loading YouTube video takes a long time,*
   - *configuring Gradle makes error*

5) *Developed Applications*

Figure 15 displays an AnimalTour application developed by a student. In contrast to Figure 12, it features different user interfaces. Additionally, some students incorporated a welcome screen, used custom icons and images, introduced a new animal type, and implemented a drag-and-drop function in InvertActivity.

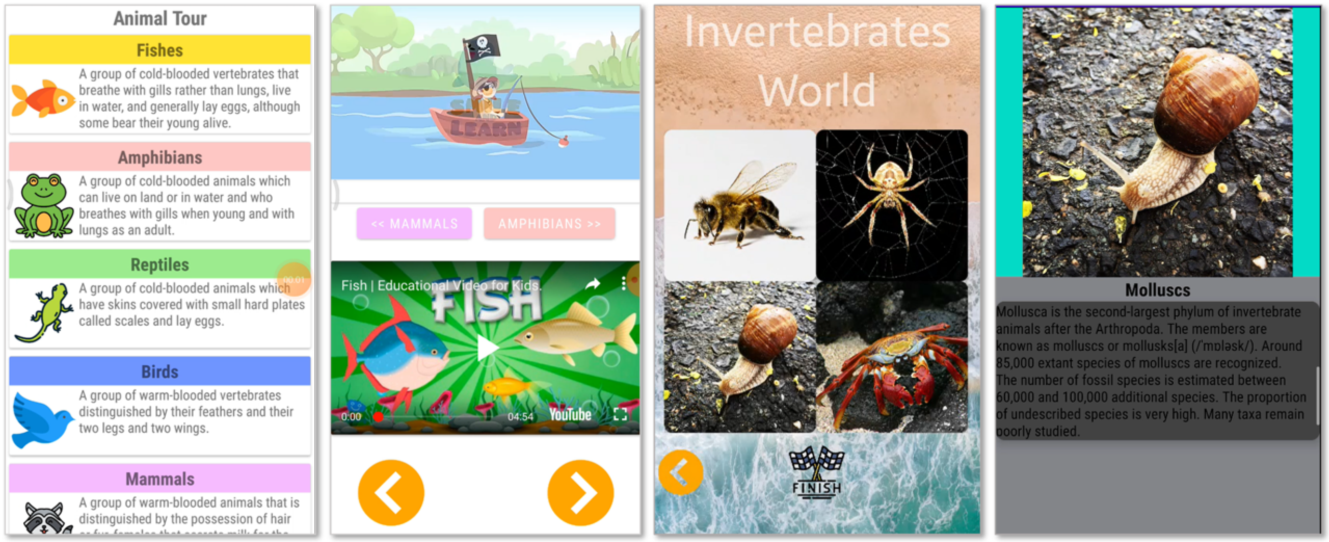
9. Analysis on Evaluation Results

This section discusses related findings based on evaluation results consisting of learning performance, learning effectiveness, students' opinion, and their skills in Android application development.

A. *Students' Learning Performance*

In solving the assignments on Multiple Activities topic, 45 of 50 students successfully completed the assignment on their first submission attempt, indicating a 90% success rate within the initial three-day period. The remaining five students managed to solve the topic within the next three days on their second attempt. These results highlight a significant improvement in the student's ability to develop more complex Android applications compared to previous topics. The average solving time for the Multiple Activities topic was 174 minutes, notably better than the times recorded for Basic UI, Basic Activity, and Advanced Widgets, which took 216 minutes, 216 minutes, and 225 minutes on average, respectively. This improvement suggests that students have become more adept at understanding guide documents and navigating the Android Studio environment.

Following the success in the Multiple Activities topic, all 50 students successfully completed the assignments in Multimedia Resources topic on their first submission attempt, achieving a 100% success rate within one week. This outcome underscores the improvements in students' ability to handle increasingly complex Android applications. The average solving time for the Multimedia Resources topic was 163 minutes, the best performance compared to the four previous topics. After mastering the earlier topics,



Fishes

A group of cold-blooded vertebrates that breathe with gills rather than lungs, live in water, and generally lay eggs, although some bear their young alive.

Amphibians

A group of cold-blooded animals which can live on land or in water and who breathes with gills when young and with lungs as an adult.

Reptiles

A group of cold-blooded animals which have skins covered with small hard plates called scales and lay eggs.

Birds

A group of warm-blooded vertebrates distinguished by their feathers and their two legs and two wings.

Mammals

A group of warm-blooded animals that is distinguished by the possession of hair

Molluscs

Mollusca is the second-largest phylum of invertebrate animals after the Arthropoda. The members are known as molluscs or mollusks[a] (/ˈmɒləsk/). Around 85,000 extant species of molluscs are recognized. The number of fossil species is estimated between 60,000 and 100,000 additional species. The proportion of undescribed species is very high. Many taxa remain poorly studied.

Figure 15. *AnimalTour* application interfaces by a student.

students found it easier to comprehend the guide documents and navigate the Android Studio environment, contributing to their improved performance in the Multimedia Resources topic.

B. *Learning Effectiveness*

Overall, the five implemented learning topics in APLAS provide a comprehensive learning journey that enables students to learn Android programming independently from basic to advanced levels. The consistent reduction in solving time from 216 minutes for Basic UI and Basic Activity, 225 minutes for Advanced Widgets, 174 minutes for Multiple Activities, and 163 minutes for Multimedia Resources reflects the students' enhanced efficiency and familiarity with the Android programming environment. This structured approach not only sparks their curiosity but also motivates them to progressively build their skills in Android application development. The students' success across these topics demonstrates their growing expertise and the effectiveness of the APLAS framework in fostering a robust understanding of Android programming. The structured approach of the APLAS framework has effectively supported students' progressive learning from basic to advanced levels, ensuring their readiness for more sophisticated development tasks.

C. *Skills In Application Development*

Both Multiple Activities and Multimedia Resources topics provide students with opportunities to showcase their creativity and technical skills by allowing them to design user interfaces and modify applications with fewer restrictions compared to previous assignments. In the Multiple Activities topic, students demonstrated their ability to innovate beyond the provided guidelines. For instance, one student enhanced the SoccerMatch application by incorporating an AlertDialog feature to input player jersey numbers—an addition not specified in the guide documents. These personalized touches not only reflect the students' growing familiarity with Android development tools but also their willingness to explore and extend beyond basic requirements. Similarly, the Multimedia Resources topic further illustrates students' ability to apply their creativity within a flexible framework. The freedom to modify user interfaces and incorporate personal design elements appears to foster a deeper engagement with the material and a stronger grasp of Android programming concepts.

10. Conclusion

This paper presented novel implementations for two learning topics within the APLAS framework: Multiple Activities and Multimedia Resources, both of which are essential to the Interactive Application stage of foundational Android programming education. These topics are designed to facilitate self-directed learning by allowing students to independently acquire and apply key concepts in mobile programming. The Multiple Activities topic involves developing a SoccerMatch application, while the Multimedia Resources topic focuses on creating an AnimalTour application. The evaluation demonstrated substantial progress in student proficiency. For the Multiple Activities topic, 90% of students successfully completed the assignments. In contrast, the Multimedia Resources topic saw a 100% success rate on the first attempt by all students, reflecting high competence in integrating multimedia elements. The feedback from the students was predominantly positive, although there were suggestions for additional guidance on technical issues and UI design.

In future research efforts, the implementation of next topics will be continued, thereby broadening the educational scope of APLAS. The analytical capabilities functions to systematically assess and interpret students' feedback will be improved. The integration of intelligent assistance and auto-grading features will further augment the learning experience.

References

[1] J. Degenhard, "Number of smartphone users worldwide from 2014 to 2029," 2 2024. [Online]. Available: https://www.statista.com/forecasts/1143723/smartphone-users-in-the-world

[2] K. Hsiao, "Android smartphone adoption and intention to pay for mobile internet," *Library Hi Tech*, vol. 31, 2013.

[3] K. McCullen, "An android application development class," *Journal of Computing Sciences in Colleges*, vol. 31, 2016.

[4] Y. W. Syaifudin, N. Funabiki, M. Kuribayashi, and W. C. Kao, "A proposal of android programming learning assistant system with implementation of basic application learning," *International Journal of Web Information Systems*, vol. 16, 2020.

[5] D. Staegemann, M. Volk, M. Perera, C. Haertel, M. Pohl, C. Daase, and K. Turowski, "A literature review on the challenges of applying test-driven development in software engineering," *Complex Systems Informatics and Modeling Quarterly*, vol. 2022, 2022.

[6] T. J. Team, "The 5th major version of the programmer-friendly testing framework for java and the jvm," 2024. [Online]. Available: https://junit.org/junit5/

[7] "Robolectric — robolectric.org," https://robolectric.org/, [Accessed 29-07-2024].

[8] Y. W. Syaifudin, N. Funabiki, M. Mentari, H. E. Dien, I. Mu'Aasyiqiin, M. Kuribayashi, and W. C. Kao, "A web-based online platform of distribution, collection, and validation for assignments in android programming learning assistance system," *Engineering Letters*, vol. 29, 2021.

[9] Y. W. Syaifudin, N. Funabiki, M. Kuribayashi, and W. chung Kao, "A proposal of advanced widgets learning topic for interactive application in android programming learning assistance system," *SN Computer Science*, vol. 2, 2021.

[10] J. Yan, H. Liu, L. Pan, J. Yan, J. Zhang, and B. Liang, "Multiple-entry testing of android applications by constructing activity launching contexts," in *Proceedings - International Conference on Software Engineering*, 2020.

[11] M. Schnieder and S. Williams, "Educational mobile apps for programming in python: Review and analysis," *Education Sciences*, vol. 13, 2023.

[12] P. Khanna and A. Singh, "Google android operating system: A review," *International Journal of Computer Applications*, vol. 147, 2016.

[13] C. Easttom, *Android Operating System*, 2021.

[14] R. Payne, *Beginning App Development with Flutter: Create Cross-Platform Mobile Apps*, 2019.

[15] H. Kang and J. Cho, "Case study on efficient android programming education using multi android development tools," *Indian Journal of Science and Technology*, vol. 8, 2015.

[16] H. F. Hanafi and K. Samsudin, "Mobile learning environment system (mles): The case of android-based learning application on undergraduates' learning," *International Journal of Advanced Computer Science and Applications*, vol. 3, no. 3, 2012. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2012.030311

[17] H. A. A. Rekhawi and S. S. Abu-Naser, "Android applications ui development intelligent tutoring system," *International Journal of Engineering and Information Systems (IJEAIS)*, vol. 2, 2018.

[18] A. Roman and M. Mnich, "Test-driven development with mutation testing – an experimental study," *Software Quality Journal*, vol. 29, 2021.

[19] P. Blundell and D. Milano, *Learning Android Application Testing*, ser. Community experience distilled. Packt Publishing, 2015.

[20] N. Funabiki, Y. Matsushima, T. Nakanishi, K. Watanabe, and N. Amano, "A java programming learning assistant system using test-driven development method," *IAENG International Journal of Computer Science*, vol. 40, 2013.

[21] J. B. Almeida, N. MacEdo, and J. Proenca, "Teaching how to program using automated assessment and functional glossy games (experience report)," *Proceedings of the ACM on Programming Languages*, vol. 2, 2018.

[22] N. G. Berihun, C. Dongmo, and J. A. V. der Poll, "The applicability of automated testing frameworks for mobile application testing: A systematic literature review," *Computers*, vol. 12, 2023.

[23] M. Hirzel and H. Klaeren, "Code coverage for any kind of test in any kind of transcompiled cross-platform applications," in *INTUITEST 2016 - Proceedings of the 2nd International Workshop on User Interface Test Automation, Co-located with ISSTA 2016*, 2016.

[24] B. Sadeh and S. Gopalakrishnan, "A study on the evaluation of unit testing for android systems," *International Journal of New Computer Architectures and their Applications (IJNCAA)*, vol. 4, 2011.

[25] M. Linares-Vásquez, C. Bernal-Cárdenas, K. Moran, and D. Poshyvanyk, "How do developers test android applications?" in *Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017*, 2017.

[26] P. S. Kochhar, F. Thung, N. Nagappan, T. Zimmermann, and D. Lo, "Understanding the test automation culture of app developers," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation, ICST 2015 - Proceedings*, 2015.

[27] J. W. Lin, N. Salehnamadi, and S. Malek, "Test automation in open-source android apps: A large-scale empirical study," in *Proceedings - 2020 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020*, 2020.

[28] P. Kong, L. Li, J. Gao, K. Liu, T. F. Bissyandé, and J. Klein, "Automated testing of android apps: A systematic literature review," *IEEE Transactions on Reliability*, vol. 68, 2019.

[29] J. Wu and J. Clause, "Automated identification of uniqueness in junit tests," *ACM Transactions on Software Engineering and Methodology*, vol. 32, 2023.

[30] G. K. Mostefaoui and F. Tariq, *Mobile apps engineering : design, development, security, and testing*. Boca Raton, FL: CRC Press, 2019.

[31] A. Hussain, H. A. Razak, and E. O. Mkpojiogu, "The perceived

usability of automated testing tools for mobile applications," *Journal of Engineering Science and Technology*, vol. 12, 2017.

[32] Y. W. Syaifudin, I. Siradjuddin, N. Funabiki, D. Y. Liliana, A. B. Kaswar, and M. Mentari, "An interactive learning system with automated assistance for self-learning user interface design on android applications," *International Journal of Computing and Digital Systems*, vol. 13, pp. 1397–1407, 5 2023.

[33] Google for Developers — Activity, "Activity," 6. [Online]. Available: https://developer.android.com/reference/android/app/Activity

[34] L. Ardito, R. Coppola, G. Malnati, and M. Torchiano, "Effectiveness of kotlin vs. java in android app development tasks," *Information and Software Technology*, vol. 127, 2020.

[35] Google for Developers — Intent, "Intent," 6. [Online]. Available: https://developer.android.com/reference/android/content/Intent

[36] Google for Developers — Fragment, "Fragment," 6. [Online]. Available: https://developer.android.com/reference/android/app/Fragment

[37] Google for Developers — YouTubePlayer API, "Youtube player api reference for iframe embeds," 6. [Online]. Available: https://developers.google.com/youtube/iframe_api_reference

[38] Google for Developers — Animations, "Introduction to animations," 6. [Online]. Available: https://developer.android.com/develop/ui/views/animations/overview