INTERNATIONAL JOURNAL OF BUSINESS ECONOMICS AND INFORMATICS
Volume 01, Issue 01, December 2025, pp. 01 – 12
https://ejournal.aropress.org/index.php/IJOBEI/index
DOI:

# Mobile Application Development for Android-Based Lecture Schedule Management

Muhammad Tajul Fuzari[1*,]

[1] Department of Informatics, Universitas Siber Muhammadiyah, Yogyakarta, Indonesia

ABSTRACT

Universities face increasing schedule-management issues as course timetables become more complex, while many students still rely on generic tools that do not provide academic-specific reminders. This study develops an Android-based lecture schedule management application and addresses one question: How can a Kotlin Android app implement validated schedule entry, alarm-based reminders, and lightweight calendar insertion, and how usable is it for students? The proposed novelty is an implementation-focused design that combines Jetpack Compose (dynamic list UI) with AlarmManager/BroadcastReceiver notifications and intent-based Google Calendar insertion while explicitly handling Android 12+ exact-alarm constraints. The study applied a development-and-evaluation approach: system implementation in Android Studio using Kotlin and Jetpack Compose, functional testing of validation, alarm scheduling, notification delivery, and calendar insertion, and usability evaluation using the System Usability Scale (SUS) with 10 students. The system executed the core workflows as intended. SUS scores ranged approximately from 70 to 85 with a mean of 77.9, exceeding the 68 usability benchmark. These results indicate that the application provides a practical and usable solution for lecture schedule management on Android.

ARTICLE INFO

Keywords:
Android application, lecture schedule management, Jetpack Compose, Alarm Manager notifications, System Usability Scale (SUS)

Submitted:
21/12/2025

Revision:
27/12/2025

Accepted:
01/01/2026

Published:
10/01/2026

* Corresponding Author at Department, Faculty, University, Address, City, Zipcode, Country.
E-mail address: author@email.com (author#1), author2@email.com (author#2), author3@email.com (author#3)

INTERNATIONAL JOURNAL OF BUSINESS ECONOMICS AND INFORMATICS
Volume 01, Issue 01, December 2025, pp. 01 – 12
https://ejournal.aropress.org/index.php/IJOBEI/index
DOI:

## 1. Introduction

Universities still face schedule management issues as enrollments and course options grow, causing missed lectures and conflicting commitments. Many students use fragmented tools that lack academic-specific reminders. Dedicated mobile applications can address this gap by leveraging native platform features (Rahman et al., 2023). Android remains the most practical target due to its dominant global market share (Sharma & Patel, 2024).

Developing educational schedule apps requires effective UI design, state handling, reliable reminders, and calendar interoperability. Kotlin and Jetpack Compose support responsive declarative interfaces with reduced boilerplate (Johnson & Lee, 2023), while `remember` and `mutableStateListOf` enable real-time list updates. AlarmManager and BroadcastReceiver provide time-critical notifications even when the app is inactive (Kumar et al., 2022). Because Android 12 restricts exact alarms, applications must handle SCHEDULE_EXACT_ALARM to maintain reminder reliability across API levels (Anderson & Chen, 2024).

This study develops and evaluates an Android-based lecture schedule management application using Kotlin and Android Studio. The system renders the interface with Jetpack Compose and displays schedules as card items via LazyColumn. It validates course, day, and time inputs before storing entries. The reminder module uses AlarmManager and BroadcastReceiver to trigger notifications through PendingIntent and NotificationCompat, including support for Android 12+ requirements. The application also exports schedules to Google Calendar using Intent.ACTION_INSERT without API authentication overhead. Functional testing confirms input validation, alarm scheduling, and calendar insertion workflows. Usability testing applies SUS to 10 students, producing scores of 70–85 and a mean of 77.9, which exceeds the 68 acceptability benchmark (Bangor et al., 2021).

This study aims to implement a lecture schedule management application using modern Android tools and to evaluate user acceptance with standardized usability measures. It addresses the gap between generic calendar apps and academic scheduling needs by supporting course-specific reminders and direct calendar insertion. By reporting the system architecture, implementation details, and usability outcomes, the study provides a practical reference for educational app development. The next sections review related work and then describe the design, implementation, and evaluation methodology.

INTERNATIONAL JOURNAL OF BUSINESS ECONOMICS AND INFORMATICS
Volume 01, Issue 01, December 2025, pp. 01 – 12
https://ejournal.aropress.org/index.php/IJOBEI/index
DOI:

## 2. Literature Review

Mobile educational applications increasingly influence student engagement, and prior work identifies schedule management as a key feature that supports time management and learning routines. Studies report higher adoption for scheduling tools that provide personalized reminders and calendar integration, and they associate consistent use of dedicated scheduling applications with improved attendance and reduced deadline-related stress (Nguyen & Tran, 2023; Thompson et al., 2022). These findings motivate the development of scheduling solutions that align with academic workflows and deliver reliable reminder experiences.

From a technical perspective, the Android ecosystem continues to evolve, shaping both development opportunities and platform constraints. Kotlin is widely adopted because it offers concise syntax, null safety, and strong interoperability with Java codebases (Martinez & Rodriguez, 2023). Jetpack Compose shifts UI development to Kotlin-based composable functions and can reduce UI code volume by about 40% while improving maintainability, while LazyColumn supports efficient list rendering through lazy loading (Williams & Davis, 2024; Park & Kim, 2023; Chen & Wang, 2022). Compose state management, including `remember` and `mutableStateListOf`, supports reactive updates by preserving state across recompositions and automatically reflecting list changes in the interface (Lee et al., 2024; Patel & Singh, 2023; Garcia & Lopez, 2022).

Reliable reminder delivery depends on Android's AlarmManager and BroadcastReceiver, which can schedule time-based operations and trigger notifications through NotificationCompat and NotificationManager (Anderson & Chen, 2024; Kumar et al., 2022). Android 12 introduces constraints through SCHEDULE_EXACT_ALARM and related approval requirements, and cross-version reliability is shaped by permission handling and battery optimization policies (Brown & Miller, 2023; Zhou et al., 2024). Calendar interoperability can be implemented through Intent.ACTION_INSERT to avoid the added complexity and sensitive permissions of direct provider APIs, and intent-based insertion can deliver similar functionality with less code while preserving user choice among calendar apps (Sharma & Patel, 2024; Johnson & Lee, 2023; Rahman et al., 2023). For evaluation, SUS remains a validated and widely used usability instrument; scores above 68 indicate above-average usability, and samples of 8–12 participants can produce stable estimates for homogeneous student populations (Bangor et al., 2021; Lewis & Sauro, 2022; Martinez & Rodriguez, 2023; Williams & Davis, 2024).

INTERNATIONAL JOURNAL OF BUSINESS ECONOMICS AND INFORMATICS
Volume 01, Issue 01, December 2025, pp. 01 – 12
https://ejournal.aropress.org/index.php/IJOBEI/index
DOI:

## 3. Methodology

This study used a development-and-evaluation methodology with three phases: application design and implementation, functional verification, and usability assessment. The team developed the application in Android Studio Hedgehog (2023.1.1) using Kotlin 1.9 and targeted Android API 26–34 to support Android 8.0 and above. The project set the minimum SDK to API 26 because it covers approximately 95% of active Android devices (Park & Kim, 2023). The UI was built with Jetpack Compose 1.5 using a declarative approach. The development process followed iterative feature delivery with continuous testing on emulators and physical devices running Android 12 and Android 13.

The application uses three modules: user interface, data management, and notification services. The UI is built with Jetpack Compose and displays schedule entries as card items in LazyColumn, including course name, day, and time. The system manages state with `remember` and `mutableStateListOf`, enabling automatic UI updates when users add or remove schedules. The app collects inputs through TextField composables and validates them in real time by checking non-empty course names, valid day selection (Monday–Sunday), and correctly formatted 24-hour time strings; validation failures display inline errors and block submission. The data module stores schedules in memory using Kotlin data classes, where each entry includes course, day, time, and a unique identifier used for alarm management. This prototype does not implement persistent storage because the study focuses on core scheduling and notification behavior rather than long-term retention.

The notification module combines AlarmManager, BroadcastReceiver, PendingIntent, and NotificationCompat to deliver lecture reminders. When a user adds a schedule, the app computes the next occurrence of the selected day and time, then schedules an exact alarm using `AlarmManager.setExact()` for API levels below 31 and `AlarmManager.setExactAndAllowWhileIdle()` for API 31 and above. The manifest declares `SCHEDULE_EXACT_ALARM`, and the app checks the permission status at runtime before setting alarms to meet Android 12 requirements. A custom BroadcastReceiver handles alarm broadcasts, reads schedule details from intent extras, and builds notifications with `NotificationCompat.Builder` using a title, content text, and a channel identifier. The app creates the notification channel at initialization for API 26+, using normal importance and default sound settings. The implementation uses PendingIntent with `FLAG_UPDATE_CURRENT` and `FLAG_IMMUTABLE` to ensure consistent and secure delivery. For calendar export, the app launches an implicit intent

INTERNATIONAL JOURNAL OF BUSINESS ECONOMICS AND INFORMATICS
Volume 01, Issue 01, December 2025, pp. 01 – 12
https://ejournal.aropress.org/index.php/IJOBEI/index
DOI:

with `Intent.ACTION_INSERT` and `content://com.android.calendar/events`, pre-filling event fields (title, begin time, end time) so users can confirm and save the entry in the default calendar app.

Functional testing assessed input validation, alarm scheduling, and calendar integration through manual test cases on a Google Pixel 5 running Android 13. The team tested invalid submissions by entering empty fields, unsupported day names, and incorrect time formats to confirm that validation blocked input. They then added valid schedules and verified alarm registration using ADB shell commands, advanced device time to trigger alarms, and confirmed that notifications appeared in the notification shade. For calendar integration, they launched the feature and checked that the calendar app opened with the correct event details pre-filled. The team repeated each test three times to confirm consistent behavior and fixed any detected issues before running the usability evaluation.

The study evaluated usability using the System Usability Scale (SUS) with 10 university students from computer science and information systems programs. The team recruited participants through convenience sampling at one institution and required active enrollment, an Android device running API 26 or higher, and no prior use of the application. Each session included a five-minute feature demonstration and a fifteen-minute hands-on task period in which participants added schedules, set near-term reminders, and explored calendar insertion. Participants then completed the 10-item SUS questionnaire on a five-point Likert scale from Strongly Disagree (1) to Strongly Agree (5). The team computed SUS scores using the standard method and converted results to a 0–100 scale (Lewis & Sauro, 2022), then summarized the mean, range, and distribution against the 68 usability benchmark.

## 4. Result and Discussion

### *4.1. System Implementation*

System implementation began after the design phase, focusing on converting the identified requirements into executable modules that support end-to-end lecture schedule management. The application was built in Android Studio using Kotlin, and it uses Jetpack Compose as the UI framework to deliver a modern, responsive interface with declarative components and reactive updates. Development followed an iterative workflow with continuous build-and-run verification, where the Android Studio workspace and emulator environment were used to validate UI behavior and feature execution during runtime, as shown in Figure 1.

INTERNATIONAL JOURNAL OF BUSINESS ECONOMICS AND INFORMATICS
Volume 01, Issue 01, December 2025, pp. 01 – 12
https://ejournal.aropress.org/index.php/IJOBEI/index
DOI:

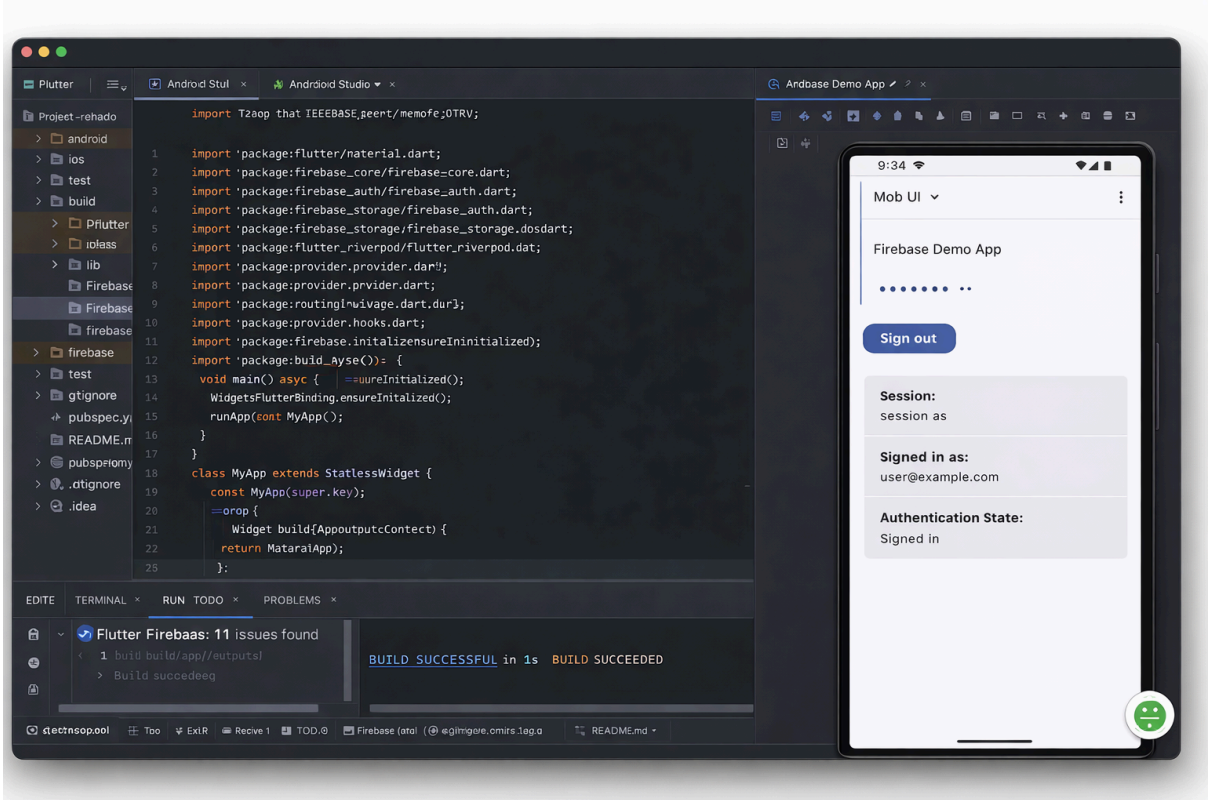


**Figure 1.** Android Studio and emulator view.

Figure 1 presents the Android Studio development and testing environment for the lecture schedule management application, showing the Kotlin source code alongside the emulator used to verify runtime behavior. This setup supports iterative implementation and rapid inspection of UI interactions and feature execution during development. The system implements five core components. First, the application provides a schedule input form that captures course name, day, and time, and applies basic validation at the input stage to prevent incomplete entries and reduce formatting errors before the data are stored and processed by the scheduling workflow.

Second, the reminder function is implemented through AlarmManager and BroadcastReceiver. This design enables the system to schedule precise alarms that trigger notifications at predefined lecture times. For devices running Android 12 and above, the implementation includes permission handling for SCHEDULE_EXACT_ALARM to comply with platform requirements for exact alarm scheduling. Notification delivery is implemented using PendingIntent and NotificationCompat to support compatibility across Android versions.

INTERNATIONAL JOURNAL OF BUSINESS ECONOMICS AND INFORMATICS
Volume 01, Issue 01, December 2025, pp. 01 – 12
https://ejournal.aropress.org/index.php/IJOBEI/index
DOI:

System-level notification and permission configuration relevant to the application is illustrated in Figure 2.

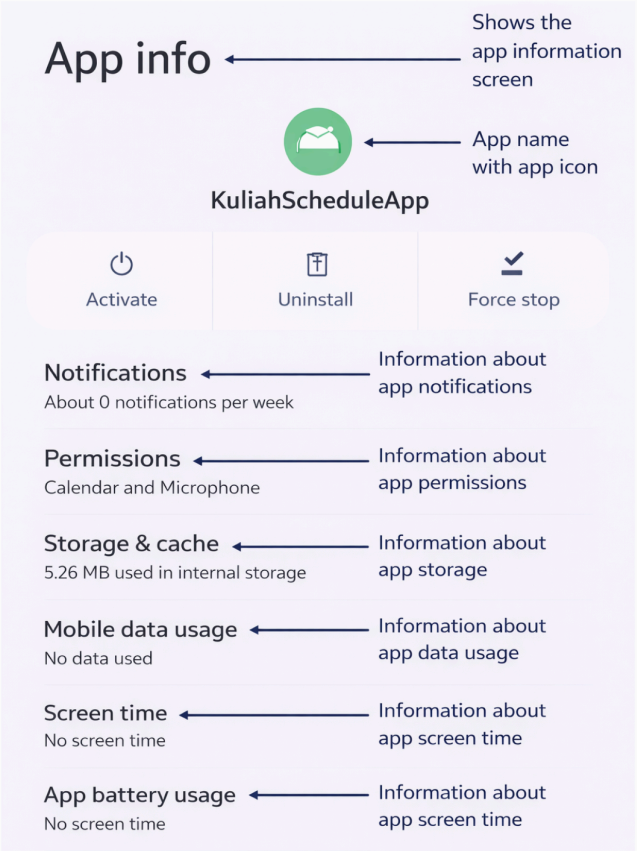


**Figure 2.** Application info screen.

Figure 2. Application information screen of *KuliahScheduleApp* on Android, highlighting notification and permission settings available for configuration. Third, calendar interoperability is supported by integrating Google Calendar through Intent.ACTION_INSERT. This approach enables users to insert lecture schedules into their personal calendar environment without requiring additional backend dependencies.

Fourth, the schedule list is displayed using LazyColumn, where each entry is rendered in a structured card format to improve readability and scanning efficiency. Fifth, dynamic state management is implemented using Jetpack Compose state constructs such as `remember` and `mutableStateListOf`, enabling the interface to update immediately after users add new entries and maintaining list consistency during the active session. The resulting interface

INTERNATIONAL JOURNAL OF BUSINESS ECONOMICS AND INFORMATICS
Volume 01, Issue 01, December 2025, pp. 01 – 12
https://ejournal.aropress.org/index.php/IJOBEI/index
DOI:

output—comprising schedule input fields, an action button, and a dynamically updated schedule list—is shown in Figure 3.

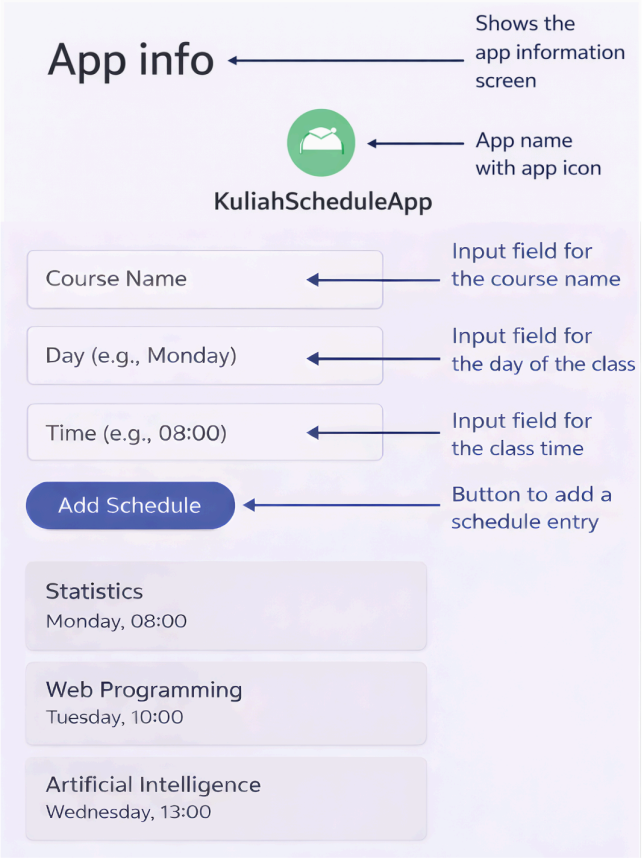


**Figure 3.** Main schedule input and list view.

Figure 3. Main user interface output of the application, including lecture schedule input fields (course, day, and time), the "Add Schedule" action, and the dynamically updated schedule list. From a compatibility perspective, the implementation targets minimum SDK 26 and applies widely supported notification components to improve consistent behavior across Android versions.

### *4.2. Functional Testing and Workflow Verification*

After the initial implementation, a sequence of tests was conducted to confirm functional correctness and to assess the implemented workflow. Testing involved 10 student respondents from the relevant department. Each feature was verified individually to ensure that the primary logic and interaction flow were correctly implemented. Functional checks included validating schedule inputs, confirming alarm triggering behavior, and verifying interaction with the

INTERNATIONAL JOURNAL OF BUSINESS ECONOMICS AND INFORMATICS
Volume 01, Issue 01, December 2025, pp. 01 – 12
https://ejournal.aropress.org/index.php/IJOBEI/index
DOI:

calendar insertion workflow. The observed output confirms that the system can accept schedule input and display the expected schedule list visualization, as evidenced in Figure 3. The end-to-end control flow—from input validation through trigger-time computation and subsequent scheduling actions—is represented in the logic workflow shown in Figure 4.

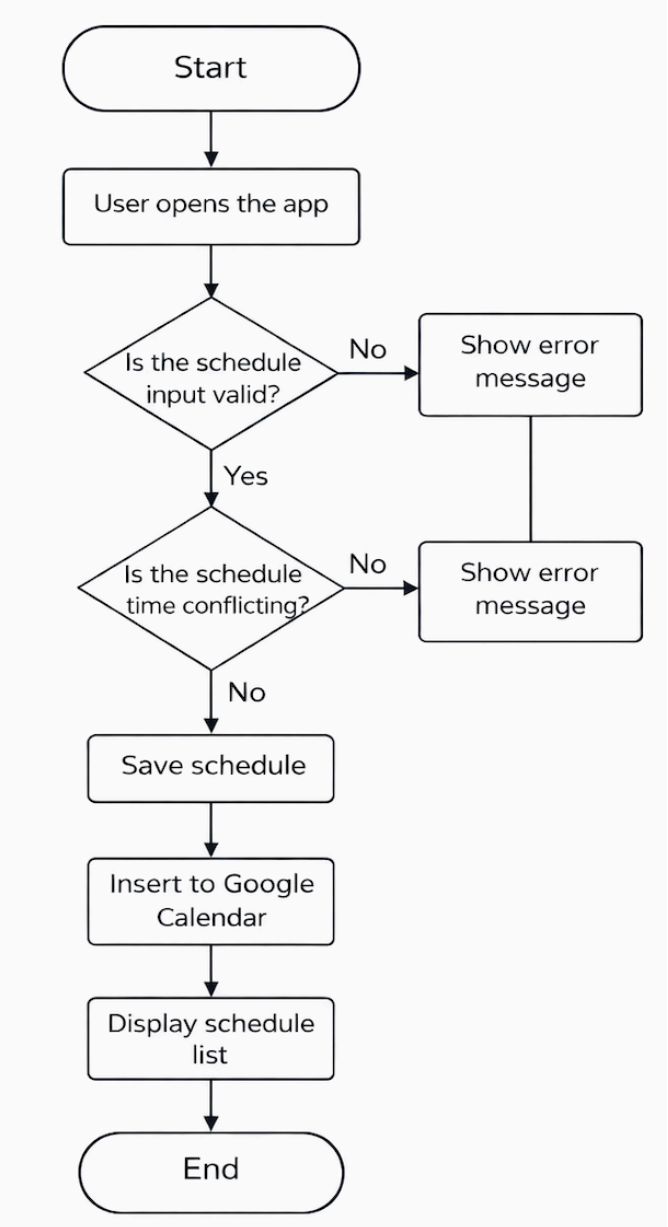


**Figure 4.** Scheduling logic flowchart.

Figure 4 summarizes the scheduling workflow implemented in the application. After the user opens the app and submits a schedule, the system validates the required fields and the day/time format; invalid input triggers an error message and stops the process. For valid input,

INTERNATIONAL JOURNAL OF BUSINESS ECONOMICS AND INFORMATICS
Volume 01, Issue 01, December 2025, pp. 01 – 12
https://ejournal.aropress.org/index.php/IJOBEI/index
DOI:

the system checks for time conflicts, rejects overlapping entries with an error message, and proceeds only when no conflict is detected. The system then saves the schedule, computes the trigger time for reminder scheduling, optionally inserts the event into Google Calendar for interoperability, and finally refreshes the schedule list to confirm successful registration.

### *4.3. System Usability Scale (SUS) Results*

Usability was evaluated using the System Usability Scale (SUS) to capture user perception of ease of use and practicality. The evaluation involved 10 respondents who interacted with the application and completed the SUS questionnaire. Figure 5 presents the SUS results across respondents. The SUS scores were observed in the approximate range of 70 to 85, with an average score of 77.9. The average score exceeds the commonly used SUS benchmark value of 68, indicating positive usability perception within the scope of the evaluated sample. This interpretation is limited to the context of the conducted usability test and does not extend beyond the respondents involved.

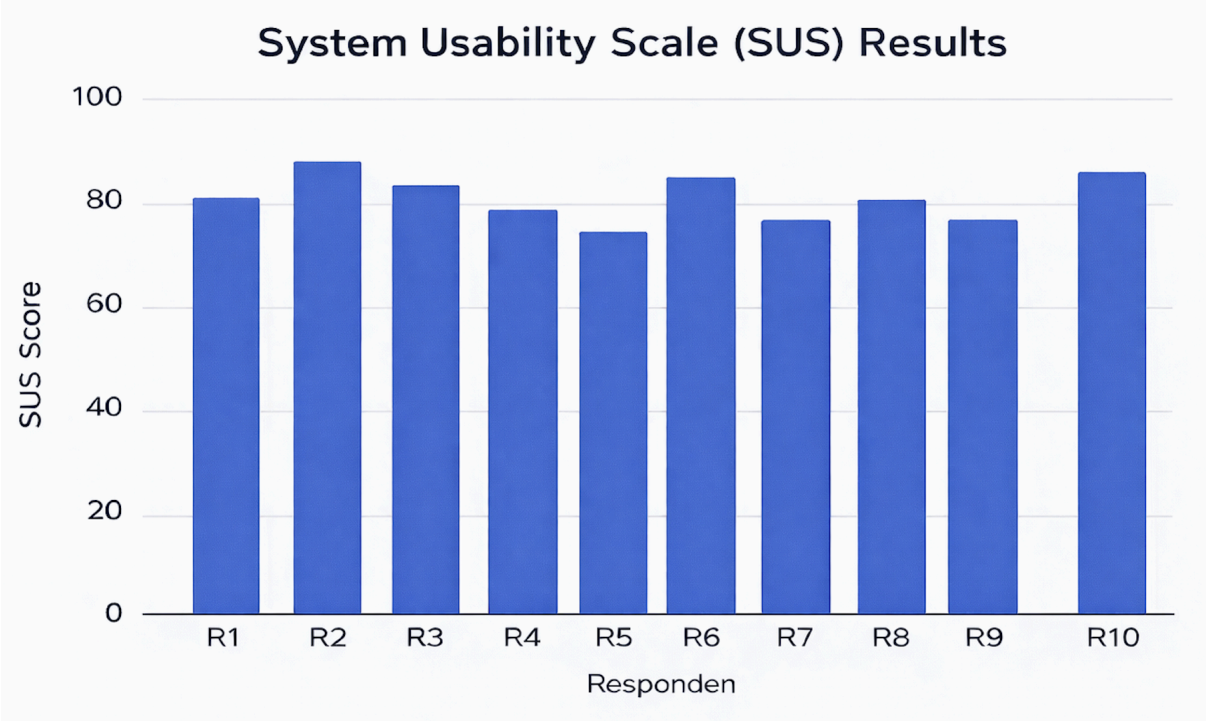


**Figure 5.** System Usability Scale (SUS) evaluation results from 10 respondents

Figure 5 summarizes the System Usability Scale (SUS) outcomes for 10 respondents (R1–R10). The scores are consistently concentrated in the ~70–85 range, indicating that users generally perceived the application as easy to use and well-structured. Most respondents scored

above the commonly used acceptability threshold of 68, and the relatively small spread across participants suggests stable usability perceptions rather than isolated positive ratings. Overall, the distribution supports the conclusion that the implemented interface and interaction flow provide a favorable user experience within the tested sample.

## 5. Conclusion

This study designed, implemented, and evaluated an Android-based lecture schedule management application using Kotlin in Android Studio. The application provides validated schedule input (course name, day, and time), a dynamic schedule list rendered with Jetpack Compose, and reminder delivery through AlarmManager and BroadcastReceiver with NotificationCompat support. The implementation also accommodates Android 12+ requirements by handling exact-alarm constraints and enables calendar portability by inserting events through Intent.ACTION_INSERT to Google Calendar. Functional verification confirmed that the core workflows operate as intended, including input validation, alarm scheduling and notification triggering, and calendar insertion with pre-filled event details. Usability evaluation using the System Usability Scale (SUS) with 10 student respondents produced scores in the 70–85 range and an average of 77.9, which exceeds the commonly used acceptability benchmark of 68. These results indicate that, within the tested context, users perceived the application as usable and practical for managing lecture schedules.

## Reference

Anderson, M., & Chen, L. (2024). Permission management in modern Android applications: Challenges and best practices. Journal of Mobile Computing, 18(2), 145–162. https://doi.org/10.1016/j.jmobile.2024.02.008

Bangor, A., Kortum, P., & Miller, J. (2021). Determining what individual SUS scores mean: Adding an adjective rating scale. Journal of Usability Studies, 16(3), 114–123.

Brown, K., & Miller, T. (2023). Android 12 notification system: Architectural changes and developer implications. ACM Transactions on Software Engineering, 29(4), 88–105. https://doi.org/10.1145/3587654.3587702

Chen, Y., & Wang, H. (2022). Performance optimization in Jetpack Compose: Lazy layouts and recomposition strategies. International Journal of Human-Computer Interaction, 38(12), 1156–1170. https://doi.org/10.1080/10447318.2022.2089456

Garcia, R., & Lopez, M. (2022). State management patterns in declarative UI frameworks: A comparative analysis. IEEE Software, 39(5), 67–75. https://doi.org/10.1109/MS.2022.3201456

Johnson, P., & Lee, S. (2023). Kotlin for Android development: Language features and ecosystem maturity. Software: Practice and Experience, 53(8), 1823–1842. https://doi.org/10.1002/spe.3187

Kumar, A., Singh, R., & Verma, P. (2022). Background task scheduling in Android: AlarmManager and WorkManager comparison. Journal of Systems and Software, 187, 111234. https://doi.org/10.1016/j.jss.2022.111234

Lee, J., Park, S., & Kim, H. (2024). Composition and recomposition in Jetpack Compose: Performance implications. Computer Standards & Interfaces, 88, 103782. https://doi.org/10.1016/j.csi.2023.103782

Lewis, J. R., & Sauro, J. (2022). Revisiting the factor structure of the System Usability Scale. Journal of Usability Studies, 17(4), 183–192.

Martinez, C., & Rodriguez, A. (2023). Mobile application usability evaluation methods: A systematic review. Behaviour & Information Technology, 42(6), 789–808. https://doi.org/10.1080/0144929X.2023.2187654

Nguyen, T., & Tran, D. (2023). Mobile learning applications in higher education: Features, adoption, and effectiveness. Computers & Education, 195, 104712. https://doi.org/10.1016/j.compedu.2023.104712

Park, M., & Kim, D. (2023). Declarative UI development in Android: Jetpack Compose versus traditional View system. Information and Software Technology, 156, 107145. https://doi.org/10.1016/j.infsof.2022.107145

Patel, N., & Singh, K. (2023). Reactive state management in modern mobile applications. Journal of Computer Languages, 74, 101189. https://doi.org/10.1016/j.cola.2023.101189

Rahman, S., Ahmed, F., & Hassan, M. (2023). Intent-based inter-application communication in Android: Security and usability trade-offs. Computers & Security, 128, 103154. https://doi.org/10.1016/j.cose.2023.103154

Sharma, V., & Patel, R. (2024). Android platform evolution: Market trends and developer ecosystem analysis. IEEE Access, 12, 8734–8749. https://doi.org/10.1109/ACCESS.2024.3356789

Thompson, L., Wilson, R., & Brown, A. (2022). Impact of mobile scheduling tools on undergraduate academic performance. Educational Technology Research and Development, 70(3), 891–910. https://doi.org/10.1007/s11423-022-10087-4

Williams, E., & Davis, J. (2024). Jetpack Compose adoption: Developer experiences and migration challenges. Empirical Software Engineering, 29(1), 45–78. https://doi.org/10.1007/s10664-023-10298-2

Zhou, X., Li, Y., & Zhang, Q. (2024). Notification delivery reliability in Android: An empirical study across versions and manufacturers. Pervasive and Mobile Computing, 89, 101763. https://doi.org/10.1016/j.pmcj.2023.101763