JSiI | Jurnal Sistem Informasi
Vol. 12 No. 1 | Maret 2025, Hal. 63-71

DOI 10.30656/jsii.v11i1.6936
p-ISSN: 2406-7768
e-ISSN: 2581-2181

# Implementation Of Firebase In The Development Of Android-Based Queue Reservation And Treatment Record Applications

**Audy Fitri Ariani[1], Agung Brastama Putra[2], Tri Luhur Indayanti Sugata[3]**

[1,2,3] *Information System Department , UPN "Veteran" East Java, Indonesia*
*Jl. Rungkut Madya, Gn. Anyar, Kec. Gn. Anyar, Surabaya, Jawa Timur, Indonesia*
[1]21082010005@student.upnjatim.ac.id
[2]agungbp.si@upnjatim.ac.id
[3]tri.luhur.fasilkom@upnjatim.ac.id

***Abstract***
*The development of an Android-based reservation and medical record management app is key to improving customer experience and operational efficiency in the beauty salon industry. At Salon Kezia Pasuruan, challenges such as long queues and inefficient manual recording of customer treatment histories have been identified, leading to customer dissatisfaction and operational bottlenecks. This study explores the use of Firebase, offering features like Authentication, Firestore, Cloud Messaging, and Cloud Functions, to create a reliable mobile solution. Firebase enables real-time data management, secure authentication, and efficient notifications, enhancing both salon operations and customer experience. Developed with the Waterfall model and MVVM architecture, the app showed positive results in reservation and medical record management. Users can easily book appointments and access treatment history, with fast data transfer powered by Firebase. With response times of 120 ms for reads and 140 ms for writes, Firebase ensures seamless performance. The app reduced booking time by 79.91% compared to manual methods. Further development is recommended to include staff management and real-time analytics for optimized service and better customer insights.*

*Keywords: Android, Customer Care Records, Firebase, Queue Reservation.*

## I. INTRODUCTION

Technology has a strategic role in supporting the improvement of service quality as well as business competitiveness, including in the beauty salon industry. One of the main challenges faced by beauty salons is the efficient management of customer queues and treatment records [1]. With the increasing demand for fast and organized services, mobile applications are becoming an indispensable solution to overcome this problem [2]. Therefore, the development of an Android-based queue reservation and treatment record application is important to improve customer experience and operational efficiency.

Based on data from the Central Statistics Agency (BPS) [3], mobile phone usage in Indonesia continues to increase, with 67.88% of the population owning or using at least one active mobile number in 2022. This figure shows an increase compared to the previous year of 65.87%. In addition, the proportion of the population accessing the internet also reached 66.48%, most of which was done through mobile devices. This trend indicates a strong public acceptance of technology, supporting the need for mobile-based solutions in various sectors, including beauty and healthcare. This data supports the relevance of Android-based application development as a solution for modern operational services in the health and beauty sector.

The development of this application uses the Waterfall method [4], which consists of structured stages such as Communication, Planning, Modeling, Construction, and Deployment, so that each stage can be evaluated before proceeding to the next stage. In addition, this application was built by applying MVVM (Model-View-ViewModel) architecture to ensure the separation of business logic, presentation logic, and data, which facilitates code management and future feature development [5].

Firebase, as an application development platform provided by Google, offers various services that can help developers build high-quality applications faster [6]. With features such as Firestore for real-time data storage, Authentication for user security, Cloud Functions for backend logic, and Firebase Cloud Messaging (FCM) for notifications, Firebase is an ideal choice to support Android-based application development [7]. The integration of these services not only speeds up the development process but also improves application functionality and security.

Previous research has shown how Firebase-based technologies can support application development in various sectors. For example, research by Kurniawan et al. (2021) on the implementation of Firebase in a sports facility rental application shows that the integration of Firebase Authentication, Realtime Database, and Cloud Messaging can improve the efficiency of data management and real-time

Received: Jan 21st, 2025 | Published: March 21st, 2025

DOI 10.30656/jsii.v11i1.6936
p-ISSN: 2406-7768
e-ISSN: 2581-2181

communication [8]. In addition, a study by Khawas & Shah (2018) discussed the advantages of Firebase as a NoSQL-based backend platform that can store data dynamically with better performance than traditional RDBMS [6].

This case study focuses on the implementation of Firebase in the development of a queue reservation and treatment record application at Salon Kezia Pasuruan. Through this research, it is expected to analyze how the application of this technology can improve the efficiency of salon operations and provide a better experience for customers. Thus, the purpose and benefit of this research is to design a queue reservation application and maintenance record management in the salon by applying Firebase technology, to improve the operational efficiency of the salon and provide a better customer experience. The problem limitation in this research is that the system is developed using Android-based and uses Firebase technology, which includes features for storage, cloud messaging, authentication, Firestore, and cloud functions.

## II. RESEARCH METHODOLOGY

The research method applied in this research uses the Waterfall approach. According to Pressman (2014) Waterfall is a structured and systematic method in software development [1]. This method consists of several sequential stages, namely Communication, Planning, Modeling, Construction, and Deployment. Each stage is designed to be completed thoroughly before proceeding to the next stage, allowing problems to be identified and resolved at an early stage of development [2].

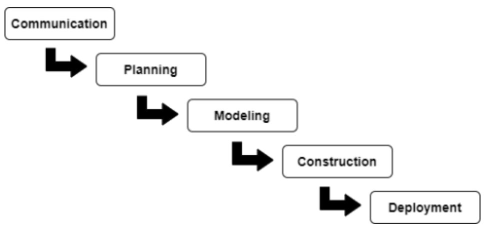


*Figure 1. Waterfall Research Method*

The following is an explanation of each stage of the research.

### A. *Communication*

This stage involved identifying needs through interviews with the owner and staff of Salon Kezia as well as direct observation of operational processes. Requirements data was collected to understand the existing problems and determine the main functions that the application should have, such as a queue reservation system, recording treatment records, and automatic notifications.

### B. *Planning*

Once the system requirements were identified, planning was done to determine the project scope, time estimation, and technology used. At this stage, Firebase was chosen as the backend platform as it offers various features that support more efficient application development. Firebase Authentication is used to ensure security and ease of user authentication, while Firestore provides a real-time data storage solution that enables fast and responsive data management. Cloud Messaging is used to efficiently send notifications to users, while Cloud Functions enables serverless execution of backend logic, reducing the burden on server management and facilitating feature development. MVVM architecture was chosen to support code modularity, simplify management, and enable further development that is more structured.

### C. *Modeling*

The modeling stage aims to design the system in depth through visual representations that describe the processes, interactions, and data structures in the application. In this research, use case diagrams, flowcharts, and class diagrams are used as the main tools in design [3]. Each diagram has a specific function in documenting the system requirements and process flow of the application being developed

The use case diagram developed only focuses on the customer as the main actor. This diagram illustrates how customers interact with the application to execute various functionalities [4]. The main features identified in this use case diagram are queue reservation, access to treatment history, and receiving notifications. This explanation is depicted visually in Figure 2, which shows the interaction between the customer and the system in the form of a use case diagram.

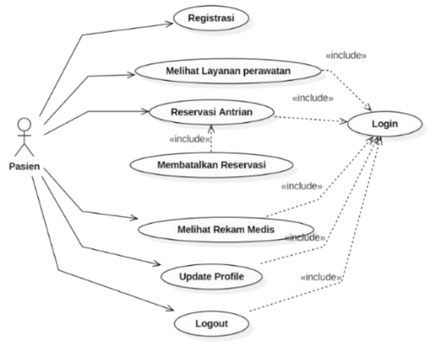


*Figure 2. Use Case Diagram*

Using use case diagrams, system requirements can be analyzed in a more structured manner, ensuring that the customer's interaction with the application is well covered in the design [5]. These diagrams also help to validate that the entire functionality required to fulfill the customer's needs has been considered in the development of the application.

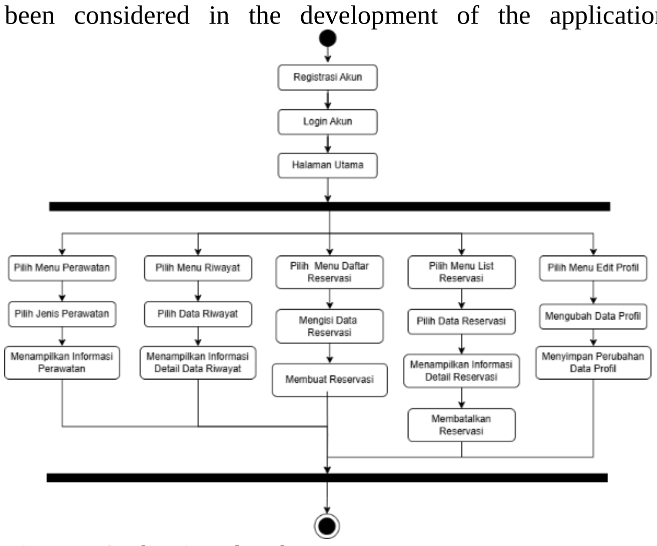


*Figure 3. Application Flowchart*

Received: Jan 21st, 2025 | Published: March 21st, 2025

DOI 10.30656/jsii.v11i1.6936
p-ISSN: 2406-7768
e-ISSN: 2581-2181

In addition, flowcharts are used to describe more detailed process flows of customer interactions with the system, such as the process of reservation booking [4]. This flowchart covers the steps that customers take, from selecting an available schedule and confirming to getting notifications regarding the reservation's status. It is depicted in Figure 3, which shows the steps that customers must take in booking a service and how the system responds to each stage

To model the customer-related data structure, a class diagram is also used, albeit a simpler one. This diagram depicts the "Customer" class that stores attributes such as customer ID, name, email address, and maintenance history. Each of these attributes supports functionality that will be used in the system, such as login, registration, and service history recording. Figure 4 shows the class diagram depicting the customer data structure in the application.

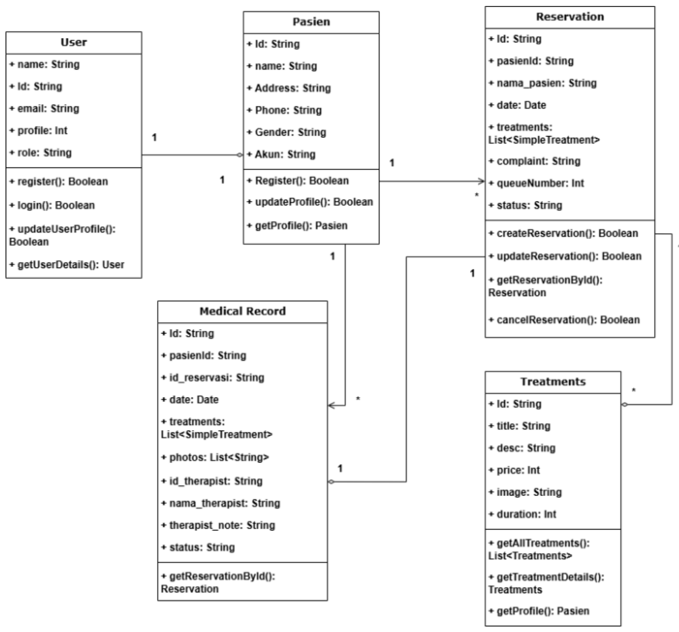


Figure 4. Class Diagram

## D. Construction

This stage involves developing program code based on the design, utilizing Firebase for key features such as Firestore for real-time data storage, Authentication for user security, and Firebase Cloud Messaging for notifications. The application adopts the Model-View-ViewModel (MVVM) architecture to enhance code modularity and maintainability.

The MVVM architecture is chosen for its ability to separate concerns effectively, where the Model handles data and business logic, the ViewModel manages and prepares data for the View, and the View focuses on presentation. This separation facilitates easier testing, reduces the risk of errors during updates, and supports real-time features through efficient data binding between the View and ViewModel [6].

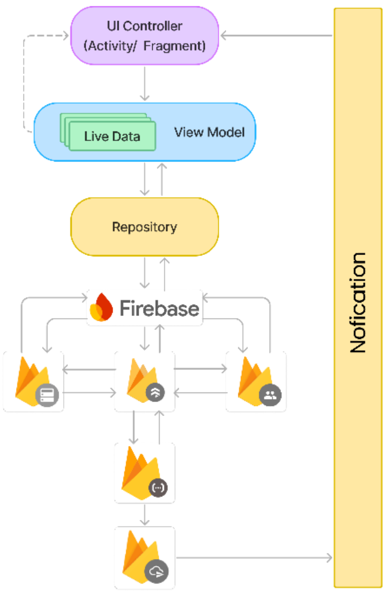


Figure 5. System Architecture

## E. Deployment

The application was tested and implemented for users. Testing is done using the *Blackbox* method to ensure each function of the application runs as needed [7], such as managing reservations, recording treatment records, and providing notifications. After testing was completed and all features were confirmed to be functioning properly, the application was launched for use by Salon Kezia.

## III. Result and Discussion

This chapter describes the implementation of Firebase technology in the development of an Android-based queue reservation and treatment record application. In addition, an evaluation of the application's performance is carried out to ensure its effectiveness and efficiency in meeting user needs

## A. Firebase Deployment

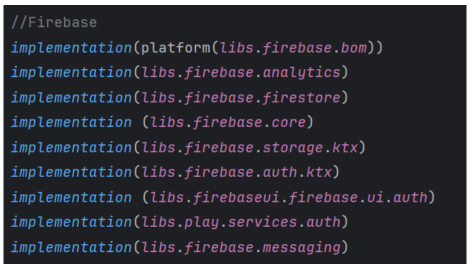
The application of Firebase in the development of an Android-based queue reservation and treatment record application focuses on utilizing the various services provided by this platform to support real-time data management, user authentication, and notification delivery. In this research, Firebase was chosen because it offers an efficient and easy-to-integrate solution for mobile applications, especially in terms of data management, security, and communication between users.

In this research, only five Firebase services are integrated, namely Firebase Authentication, Firestore, Storage, Cloud Messaging, and Cloud Functions. The process of integrating Firebase services in this application is done through the following steps:

a. Register your Google Account to the Firebase service at https://firebase.google.com/
b. Register the Android application (project) to the Firebase console using the SHA key of the Android project.
c. Integrate with Firebase SDK through Gradle as shown in Figure 6.

Received: Jan 21st, 2025 | Published: March 21st, 2025

DOI 10.30656/jsii.v11i1.6936
p-ISSN: 2406-7768
e-ISSN: 2581-2181

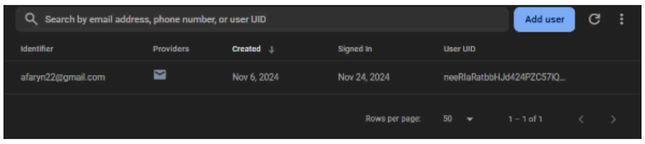


```
//Firebase
implementation(platform(libs.firebase.bom))
implementation(libs.firebase.analytics)
implementation(libs.firebase.firestore)
implementation (libs.firebase.core)
implementation(libs.firebase.storage.ktx)
implementation(libs.firebase.auth.ktx)
implementation (libs.firebaseui.firebase.ui.auth)
implementation(libs.play.services.auth)
implementation(libs.firebase.messaging)
```

*Figure 6. Firebase Dependency in Android Studio*

### 1) *Firebase Authentication*

The first feature implemented in the development of this application is Firebase Authentication, which is used to manage user authentication. Firebase Authentication provides various authentication methods, includingauthentication via email and password [8]. This service is implemented to ensure user security by facilitating efficient implementation of login, registration, and token verification systems. This enables protection of the confidentiality and integrity of user data accessed by the application [9].

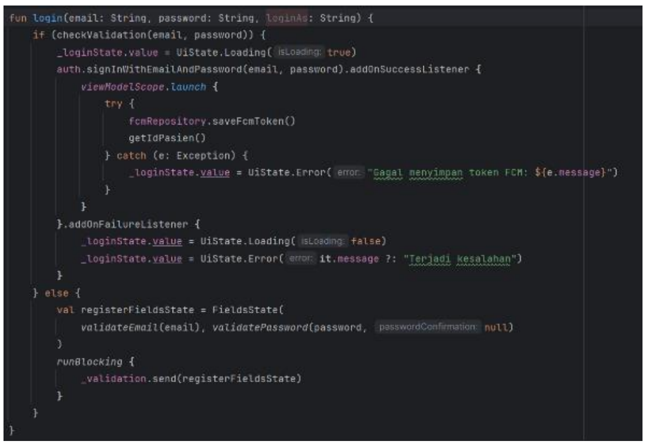


Search by email address, phone number, or user UID

| Identifier | Providers | Created ↓ | Signed In | User UID |
|---|---|---|---|---|
| afaryn22@gmail.com | ✉ | Nov 6, 2024 | Nov 24, 2024 | neeRlaRatbbHJd424PZC57lQ... |

Rows per page: 50    1 – 1 of 1

*Figure 7. Firebase Authentication*



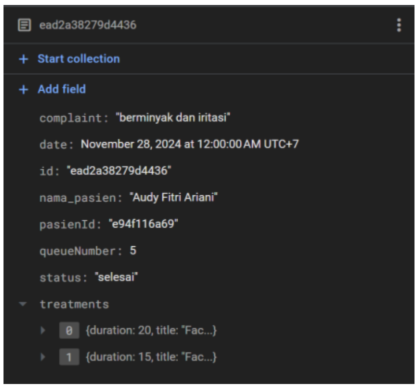
```
fun login(email: String, password: String, loginAs: String) {
    if (checkValidation(email, password)) {
        _loginState.value = UiState.Loading( isLoading: true)
        auth.signInWithEmailAndPassword(email, password).addOnSuccessListener {
            viewModelScope.launch {
                try {
                    fcmRepository.saveFcmToken()
                    getIdPasien()
                } catch (e: Exception) {
                    _loginState.value = UiState.Error( error: "Gagal menyimpan token FCM: ${e.message}")
                }
            }
        }.addOnFailureListener {
            _loginState.value = UiState.Loading( isLoading: false)
            _loginState.value = UiState.Error( error: it.message ?: "Terjadi kesalahan")
        }
    } else {
        val registerFieldsState = FieldsState(
            validateEmail(email), validatePassword(password,  passwordConfirmation: null)
        )
        runBlocking {
            _validation.send(registerFieldsState)
        }
    }
}
```

*Figure 8. Code Implementation for Authentication*

The authentication process starts with the user registering to create a new account. Once the user data is entered, the system verifies the data and grants access to the application. Customers who successfully login will be directed to the main page to start further interaction with the application, such as making queue reservations and accessing their treatment history. As seen in Figure 7 and Figure 8, the Firebase Authentication implementation can be seamlessly integrated into the Android app, ensuring a fast and secure login process for users.

### 2) *Firestore*

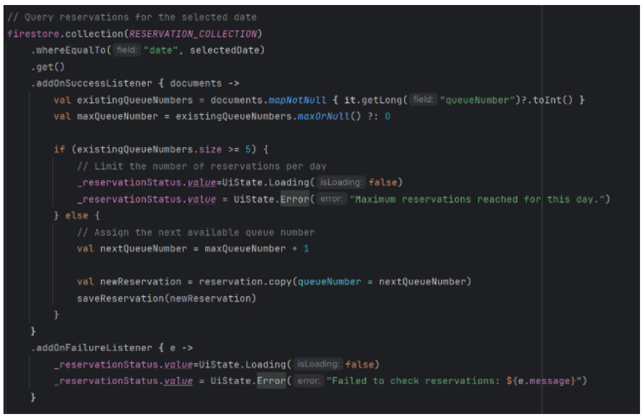
Firestore, as a real-time data storage solution, is used to store information related to reservations, treatment records, and user data. In this research, Firestore is used to ensure that the data managed in the application is always up-to-date and can be accessed quickly. Firestore enables efficient data storage as well as automatic synchronization of data between the application and the server, without the need for manual refresh [10].



```
ead2a38279d4436
+ Start collection
+ Add field
complaint: "berminyak dan iritasi"
date: November 28, 2024 at 12:00:00 AM UTC+7
id: "ead2a38279d4436"
nama_pasien: "Audy Fitri Ariani"
pasienId: "e94f116a69"
queueNumber: 5
status: "selesai"
treatments
    0 {duration: 20, title: "Fac..."}
    1 {duration: 15, title: "Fac..."}
```

*Figure 9. Firestore*

The data structure in Firestore is divided into collections and documents. For example, user data is stored in the "Users" collection, while reservation data is recorded in the "Reservations" collection. Each document in these collections contains relevant attributes, such as user ID, reservation schedule, and treatment details. The implementation of Firestore in this application allows customers to view their treatment history directly and update reservation data if needed.



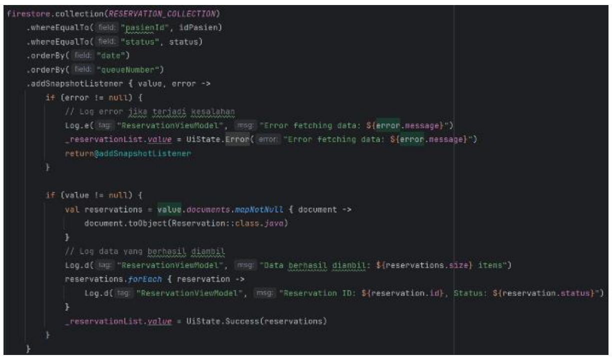
```
// Query reservations for the selected date
firestore.collection(RESERVATION_COLLECTION)
    .whereEqualTo( field: "date", selectedDate)
    .get()
    .addOnSuccessListener { documents ->
        val existingQueueNumbers = documents.mapNotNull { it.getLong( field: "queueNumber")?.toInt() }
        val maxQueueNumber = existingQueueNumbers.maxOrNull() ?: 0

        if (existingQueueNumbers.size >= 5) {
            // Limit the number of reservations per day
            _reservationStatus.value=UiState.Loading( isLoading: false)
            _reservationStatus.value = UiState.Error( error: "Maximum reservations reached for this day.")
        } else {
            // Assign the next available queue number
            val nextQueueNumber = maxQueueNumber + 1

            val newReservation = reservation.copy(queueNumber = nextQueueNumber)
            saveReservation(newReservation)
        }
    }
    .addOnFailureListener { e ->
        _reservationStatus.value=UiState.Loading( isLoading: false)
        _reservationStatus.value = UiState.Error( error: "Failed to check reservations: ${e.message}")
    }
```

*Figure 10. Code Implementation for Reading Data*

As shown in Figure 10 and Figure 11, the process of storing and retrieving data from Firestore is done using the API provided by Firebase, which allows the application to store and retrieve data from Firestore. Android communicates with Firestore efficiently without manually handling network communication. By utilizing the Firebase SDK (Software Development Kit), developers can access APIs that simplify interaction with Firestore through methods such as add(), get(), set(), and update(), ensuring real-time data updates and improving data management efficiency. This allows applications to provide a more responsive user experience without significant latency

Received: Jan 21st, 2025 | Published: March 21st, 2025

DOI 10.30656/jsii.v11i1.6936
p-ISSN: 2406-7768
e-ISSN: 2581-2181



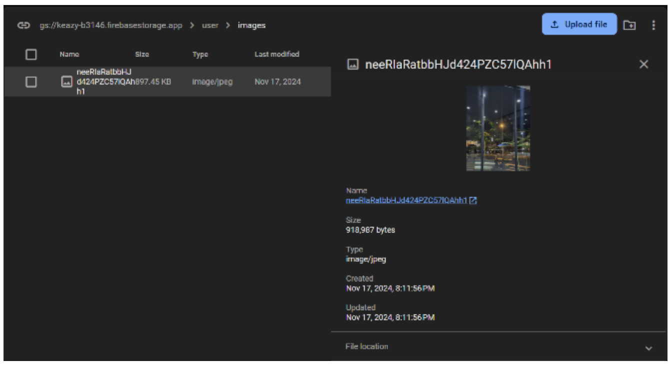
```
firestore.collection(RESERVATION_COLLECTION)
    .whereEqualTo( field: "pasienId", idPasien)
    .whereEqualTo( field: "status", status)
    .orderBy( field: "date")
    .orderBy( field: "queueNumber")
    .addSnapshotListener { value, error ->
        if (error != null) {
            // Log error jika terjadi kesalahan
            Log.e( tag: "ReservationViewModel", msg: "Error fetching data: ${error.message}")
            _reservationList.value = UiState.Error( error: "Error fetching data: ${error.message}")
            return@addSnapshotListener
        }

        if (value != null) {
            val reservations = value.documents.mapNotNull { document ->
                document.toObject(Reservation::class.java)
            }
            // Log data yang berhasil diambil
            Log.d( tag: "ReservationViewModel", msg: "Data berhasil diambil: ${reservations.size} items")
            reservations.forEach { reservation ->
                Log.d( tag: "ReservationViewModel", msg: "Reservation ID: ${reservation.id}, Status: ${reservation.status}")
            }
            _reservationList.value = UiState.Success(reservations)
        }
    }
```

Figure 11. Code Implementation for Data Input

### 3) *Firebase Storage*

In addition to Firestore, Firebase Storage is used to store multimedia files relevant to customer activities, such as photos of maintenance records or other documents. This file storage is especially important for applications that require the integration of images or supporting documents. Firebase Storage utilizes cloud to store files, which enables fast and secure access for users [8].

Each file uploaded to Firebase Storage is associated with a specific user, using the user ID as metadata to ensure that the file can only be accessed by its owner. In this application, customers can upload photos or documents related to the maintenance that has been carried out. This supports the feature of recording maintenance history with visual media, which makes it easier to manage maintenance data. The Firebase Storage implementation is shown in Figure 12 and Figure 13, which shows how the uploaded files can be stored and accessed through the Firebase API efficiently and securely.



gs://keazy-b3146.firebasestorage.app > user > images

| Name | Size | Type | Last modified |
|---|---|---|---|
| neeRlaRatbbHJd424PZC57lQAhh1 | 897.45 KB | image/jpeg | Nov 17, 2024 |

neeRlaRatbbHJd424PZC57lQAhh1

Name
neeRlaRatbbHJd424PZC57lQAhh1

Size
918,987 bytes

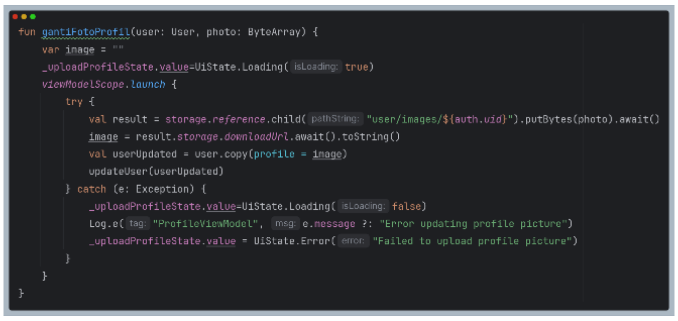
Type
image/jpeg

Created
Nov 17, 2024, 8:11:56 PM

Updated
Nov 17, 2024, 8:11:56 PM

File location

Figure 12. Firebase Storage

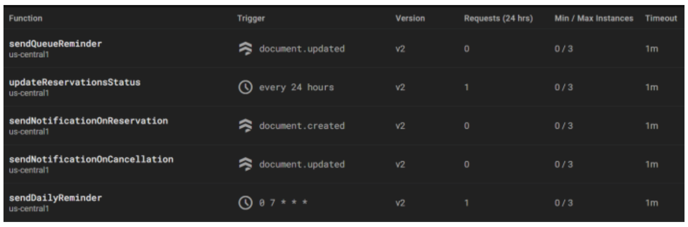


```
fun gantiFotoProfil(user: User, photo: ByteArray) {
    var image = ""
    _uploadProfileState.value=UiState.Loading( isLoading: true)
    viewModelScope.launch {
        try {
            val result = storage.reference.child( pathString: "user/images/${auth.uid}").putBytes(photo).await()
            image = result.storage.downloadUrl.await().toString()
            val userUpdated = user.copy(profile = image)
            updateUser(userUpdated)
        } catch (e: Exception) {
            _uploadProfileState.value=UiState.Loading( isLoading: false)
            Log.e( tag: "ProfileViewModel", msg: e.message ?: "Error updating profile picture")
            _uploadProfileState.value = UiState.Error( error: "Failed to upload profile picture")
        }
    }
}
```

Figure 13. Code Implementation for Input Storage

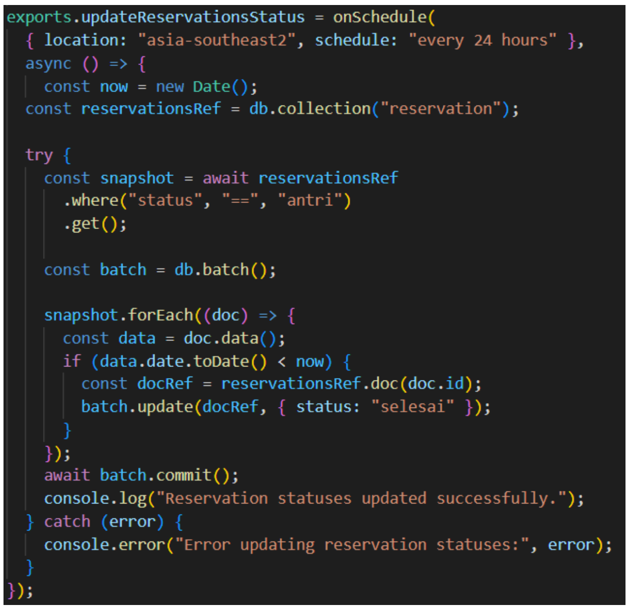
### 4) *Firebase Cloud Function*

To support more complex application logic and process automation, Firebase Cloud Functions are implemented in this system. Cloud Functions are used to handle various backend tasks, such as automatic updates of reservation status and management of notifications that should be sent to customers at a certain time [11]. These functions run on the Firebase server and can be programmed to respond to changes in data or requests made by users [12].



| Function | Trigger | Version | Requests (24 hrs) | Min / Max Instances | Timeout |
|---|---|---|---|---|---|
| sendQueueReminder<br>us-central1 | document.updated | v2 | 0 | 0 / 3 | 1m |
| updateReservationsStatus<br>us-central1 | every 24 hours | v2 | 1 | 0 / 3 | 1m |
| sendNotificationOnReservation<br>us-central1 | document.created | v2 | 0 | 0 / 3 | 1m |
| sendNotificationOnCancellation<br>us-central1 | document.updated | v2 | 0 | 0 / 3 | 1m |
| sendDailyReminder<br>us-central1 | 0 7 * * * | v2 | 1 | 0 / 3 | 1m |

Figure 14. Firebase Cloud Function



```
exports.updateReservationsStatus = onSchedule(
  { location: "asia-southeast2", schedule: "every 24 hours" },
  async () => {
    const now = new Date();
  const reservationsRef = db.collection("reservation");

  try {
    const snapshot = await reservationsRef
      .where("status", "==", "antri")
      .get();

    const batch = db.batch();

    snapshot.forEach((doc) => {
      const data = doc.data();
      if (data.date.toDate() < now) {
        const docRef = reservationsRef.doc(doc.id);
        batch.update(docRef, { status: "selesai" });
      }
    });
    await batch.commit();
    console.log("Reservation statuses updated successfully.");
  } catch (error) {
    console.error("Error updating reservation statuses:", error);
  }
});
```

Figure 15. Function for Reservation Update

An example of the application of Firebase Cloud Functions in this research is a function that automatically updates a customer's reservation status based on a predetermined time. For example, when a customer makes a reservation for a certain date, the reservation status will change to "queued" or "completed" automatically according to the predefined settings in the function. The implementation of Cloud Functions in this application helps to reduce the burden of managing the local server and improve the efficiency of the application. This can be seen in *Figure 15* and *Figure 16*, which illustrate the use of Cloud Functions for automatic reservation status updates

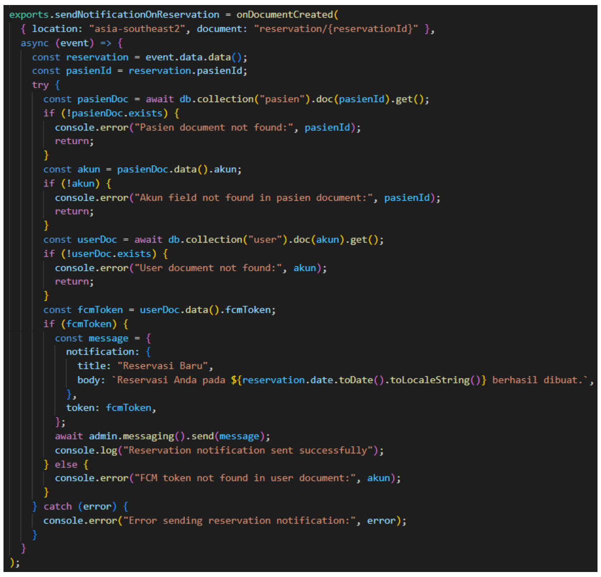
### 5) *Firebase Cloud Messaging*

Finally, Firebase Cloud Messaging (FCM) is used to send notifications to app users in real-time [8]. These notifications may include notifications about reservation status, maintenance schedule reminders, or relevant promotions. FCM is integrated into the app to provide a more interactive experience for customers [13], by ensuring that they are always up-to-date with the services they booked.

Received: Jan 21st, 2025 | Published: March 21st, 2025

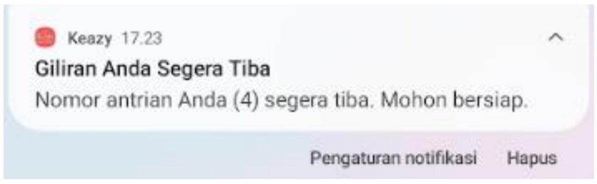
DOI 10.30656/jsii.v11i1.6936
p-ISSN: 2406-7768
e-ISSN: 2581-2181

The FCM implementation involves sending a device token to the Firebase server, which is then used to send messages directly to the customer's device. This feature is essential for improving communication between the app and the user, as well as ensuring that the customers always get notifications corresponding to their activities. FCM token storage and message delivery can be seen in Figure 17 and Figure 18, which show the implementation of notifications in the application.



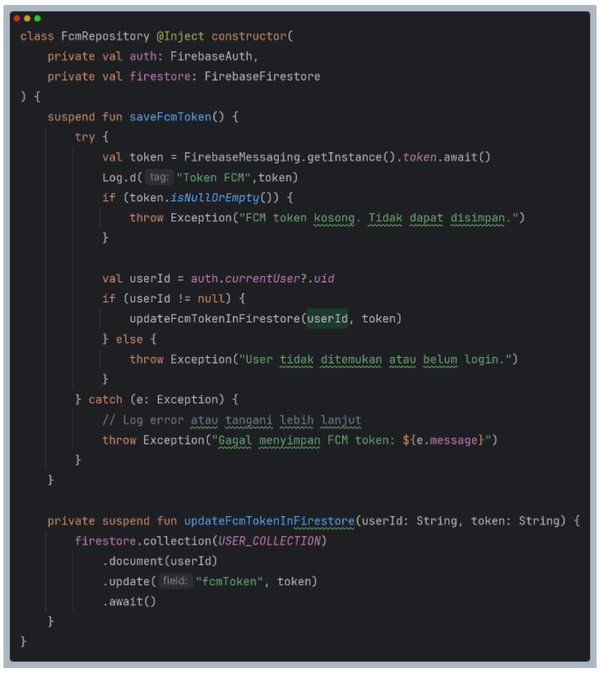
```
exports.sendNotificationOnReservation = onDocumentCreated(
  { location: "asia-southeast2", document: "reservation/{reservationId}" },
  async (event) => {
    const reservation = event.data.data();
    const pasienId = reservation.pasienId;
    try {
      const pasienDoc = await db.collection("pasien").doc(pasienId).get();
      if (!pasienDoc.exists) {
        console.error("Pasien document not found:", pasienId);
        return;
      }
      const akun = pasienDoc.data().akun;
      if (!akun) {
        console.error("Akun field not found in pasien document:", pasienId);
        return;
      }
      const userDoc = await db.collection("user").doc(akun).get();
      if (!userDoc.exists) {
        console.error("User document not found:", akun);
        return;
      }
      const fcmToken = userDoc.data().fcmToken;
      if (fcmToken) {
        const message = {
          notification: {
            title: "Reservasi Baru",
            body: `Reservasi Anda pada ${reservation.date.toDate().toLocaleString()} berhasil dibuat.`,
          },
          token: fcmToken,
        };
        await admin.messaging().send(message);
        console.log("Reservation notification sent successfully");
      } else {
        console.error("FCM token not found in user document:", akun);
      }
    } catch (error) {
      console.error("Error sending reservation notification:", error);
    }
  }
);
```

Figure 16. FCM Implementation through Function



Keazy 17.23
Giliran Anda Segera Tiba
Nomor antrian Anda (4) segera tiba. Mohon bersiap.

Figure 17. Queue Reminder Notification



```
class FcmRepository @Inject constructor(
    private val auth: FirebaseAuth,
    private val firestore: FirebaseFirestore
) {
    suspend fun saveFcmToken() {
        try {
            val token = FirebaseMessaging.getInstance().token.await()
            Log.d( tag: "Token FCM",token)
            if (token.isNullOrEmpty()) {
                throw Exception("FCM token kosong. Tidak dapat disimpan.")
            }

            val userId = auth.currentUser?.uid
            if (userId != null) {
                updateFcmTokenInFirestore(userId, token)
            } else {
                throw Exception("User tidak ditemukan atau belum login.")
            }
        } catch (e: Exception) {
            // Log error atau tangani lebih lanjut
            throw Exception("Gagal menyimpan FCM token: ${e.message}")
        }
    }

    private suspend fun updateFcmTokenInFirestore(userId: String, token: String) {
        firestore.collection(USER_COLLECTION)
            .document(userId)
            .update( field: "fcmToken", token)
            .await()
    }
}
```

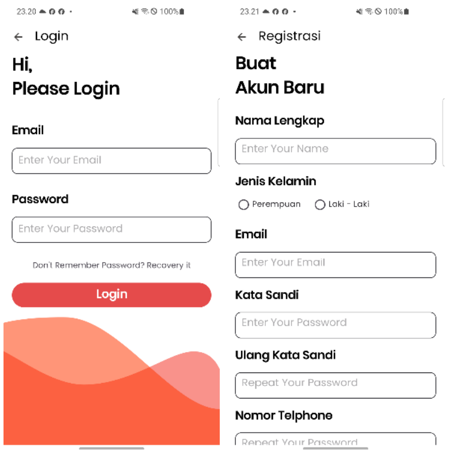
Figure 18. Code Implementation for Storing FCM Tokens

*B. Application View*

The application interface is designed to provide an intuitive and easy-to-use user experience, taking into account the main needs of customers in accessing various features [14], such as making reservations, managing treatment history, and updating profiles. The following is an explanation of the application interface that has been designed.

*1) Login and Registration Page*

The application interface starts with two main pages, namely the login and registration pages, which allow users to securely access the application. In Figure 19(a), we can see the login page designed with a simple interface, where users are required to enter their email and password to log into the app. This page ensures that only authenticated users can access important features in the app.



(a) (b)
Figure 19. Login (a) and Registration (b) page

Meanwhile, Figure 19(b) shows the registration page that allows new users to create an account. This page is designed to be easy to fill in, with a form that only requires basic information such as name, email, and password. This registration process is essential to initiate the user's interaction with the app, allowing them to make reservations and manage their personal data.

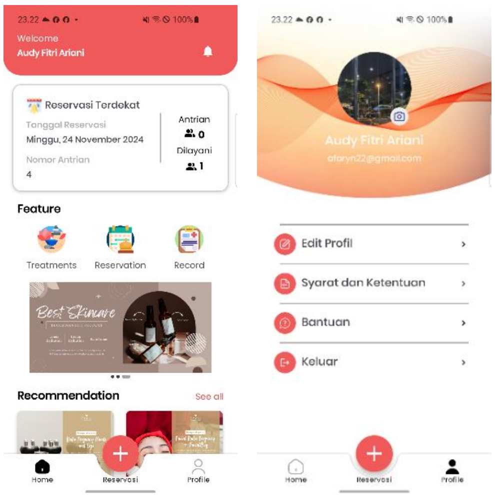
*2) Home and Profile Page*

Upon successful login, the user is directed to the main page of the application which provides an overview of the account status and recent activities. Figure 20(a) shows a view of the main page that features a clear and easy-to- understand navigation menu. This page includes quick access to the app's main features, such as reservation booking, treatment history, and profile settings.

In *Figure 20(b)*, a user profile view is shown that allows customers to manage their personal information. This page allows users to view and edit their data, such as name, email, and phone number, so customers can ensure that the information recorded in the app is always up to date. The user profile can also access the history of treatments that have been performed, providing deeper insight into the services received.

Received: Jan 21st, 2025 | Published: March 21st, 2025

DOI 10.30656/jsii.v11i1.6936
p-ISSN: 2406-7768
e-ISSN: 2581-2181

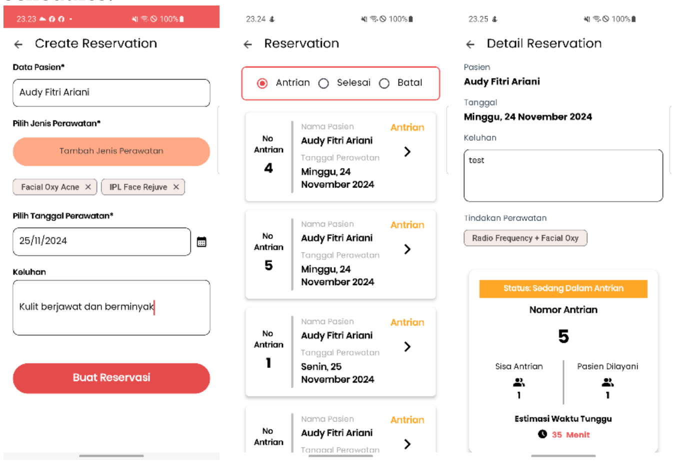


*(a)* *(b)*

*Figure 20. Main (a) and Profile (b) pages*

### *3) Reservation Page*

One of the main features of the app is the reservation system that allows customers to book services according to the available schedule. Figure 21(a) shows a view of the make reservation page, where customers can select the desired service and choose a suitable time. This interface is designed to be easy to use, with clear time selections and settings that minimize the possibility of making mistakes in selecting schedules.



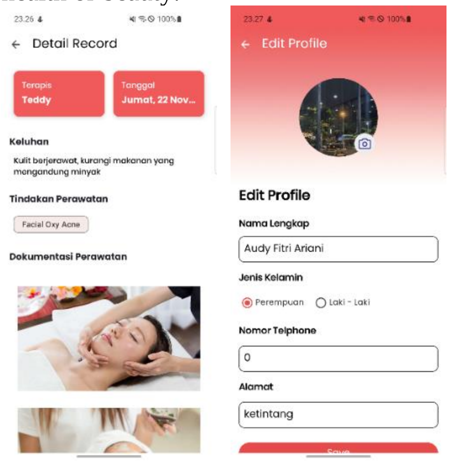
*(a)* *(b)* *(c)*

*Figure 21. Reservation creation page (a), Reservation list (b), and Reservation details (c)*

In *Figure 21(b)*, there is a reservation list page that displays all the reservations that have been made by customers, complete with reservation status and related details. This page makes it easy for customers to keep track of their reservations, so they can see the reservation history and current status of each booking made. *Figure 21(c)* shows a view of the reservation details page, which provides more information about a specific reservation, including the services ordered, the time, and the status of reservation. This page is designed to provide customers with full transparency regarding their reservations, as well as allowing them to make changes if needed.

### *4) Treatment Record and Edit Profile page*

In addition to reservations, the application also provides features to view and manage customer treatment records. Figure 22(a) shows the treatment record details page, which contains information about the treatments the customer has received, including the type of service and the date of treatment. This page provides detailed documentation of all procedures performed, which helps customers to keep track of their overall health or beauty.



*(a)* *(b)*

*Figure 22. Treatment Record Detail (a) and Edit profile (b) pages*

Finally, *Figure 22(b)* shows the profile edit page, which allows customers to update their personal information at any time. This page provides convenience in managing personal data, such as changing email addresses, phone numbers, or passwords, with the aim of improving convenience and flexibility for users in using the application

## *C. Application Evaluation*

The evaluation of the application was conducted using a black-box testing approach to assess the success rate of key features without examining the internal structure of the program. Black-box testing focuses on the output produced by the application based on various input conditions, ensuring that the system functions properly from the user's perspective [15]. The results showed that all features, including login, registration, reservation, notification, and profile management, performed as expected, with valid inputs processed correctly and error messages displayed for invalid ones

*Table 1. Blackbox Testing Results*

| No | Tested Component | Test Scenario | Expected Result | Test Result |
|---|---|---|---|---|
| 1 | Login Page | User enters a valid email and password | The system successfully logs in and navigates to the main page | Valid |
| | | User enters an invalid or unregistered email | An error message "Invalid email" appears | Valid |
| 2 | Registration Page | User enters complete registration data | Account is successfully created, and the system navigates to the main | Valid |

Received: Jan 21st, 2025 | Published: March 21st, 2025

DOI 10.30656/jsii.v11i1.6936
p-ISSN: 2406-7768
e-ISSN: 2581-2181

| | | | | |
|---|---|---|---|---|
| | | | page | |
| 3 | Reservation Page | User selects a service and an available time | Reservation is successfully saved in the system | Valid |
| | | User selects a service, but the time is unavailable | An error message "Time is not available" appears | Valid |
| 4 | Treatment History Page | User views the previous treatment history | The system displays the treatment history list | Valid |
| 5 | Notifications | Reservation time is close to the schedule | User receives a reservation reminder notification | Valid |
| 6 | Profile Page | User updates their name or email in the profile form | Profile data is successfully updated and saved | Valid |
| 7 | Logout | User selects the logout option | The system logs out and navigates back to the login page | Valid |

The average booking time using the application was 18.19 seconds, significantly faster than the manual method (90.24 second), representing a 79.91% improvement in efficiency. Firebase response times for reading and writing data were 120 ms and 140 ms, respectively, meeting real-time system criteria. These results demonstrate the application's positive impact on operational efficiency and user experience, with black-box testing confirming its functionality across all features. Figure 23 shows the improvements in booking time and efficiency before and after implementation.

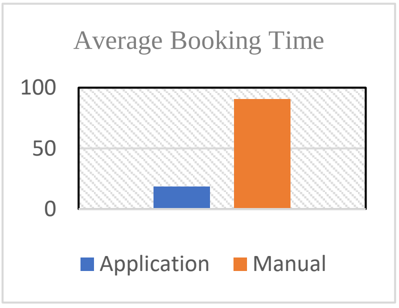


*Figure 23. Avarage Booking Time*

*Table 2. Avarage Firebase Response Times*

| Operations | Time (ms) |
|---|---|
| Writing data | 140 |
| Read Data | 120 |

## IV. Conclusion

Based on the discussion and test results, it can be concluded that the Android-based queue reservation and maintenance record management application developed using Firebase technology significantly improves operational efficiency and user experience. Firebase provides important features such as Authentication, Firestore, Cloud Messaging, and Cloud Functions, which facilitate real-time data management, secure user authentication, and efficient notification delivery. Based on the evaluation, this application shows an impressive improvement, with the booking time reduced by approximately 79.91% compared to the manual method. This shows an increase in speed and efficiency in the booking process. In addition, the system's response time for data handling, which is 120 ms for reading and 140 ms for writing, confirms its real-time capabilities

## V. Suggestions

In the design and implementation of this application, there are still several shortcomings and limitations. It is hoped that the developers of this application will consider adding a staff management module in the future, which would help streamline staff scheduling, ensure efficient allocation of resources, and improve overall operational flow, ultimately enhancing customer satisfaction. Moreover, implementing real-time user analytics could significantly improve the application by collecting data on customer preferences, such as frequently booked services and preferred reservation times. This would enable the salon to better tailor its services and promotions to meet customer needs. With these additional features, the application is expected to further contribute to improving salon operational efficiency and providing a better customer experience.

## Reference

[1] M. Serumpun, I. Ahmad, M. A. Assuja, And F. J. Hendri, "Sistem E-Booking Pelayanan Jasa Salon Menggunakan Metode Multilevel Feddback Queue (Study Kasus : Salon Griya)," *J. Teknol. Dan Sist. Inf.*, Vol. 3, No. 1, Pp. 21–25, 2022, Doi: Https://Doi.Org/10.33365/Tft.V3i1.2502.
[2] D. A. Dewi, L. E. Sudiati, And N. Haryani, "Implementation Of The Mobile Application For Facial Skin Beauty Using Sdlc Method," *J. Electro Luceat Jec*, Vol. 5, No. 2, Pp. 258–271, 2022, Doi: Https://Doi.Org/10.32531/Jsoscied.V5i2.531.
[3] B. P. S. Indonesia, "Statistik Telekomunikasi Indonesia 2022." Accessed: Nov. 25, 2024. [Online]. Available: Https://Www.Bps.Go.Id/Id/Publication/2023/08/31/131385d0253c6aae7c7a59fa/Statistik-Telekomunikasi-Indonesia-2022.Html
[4] M. D. Khairuddin And A. F. Waluyo, "Pengembangan Sistem Informasi Pasien Berbasis Mobile Pada Rsud Sunan Kalijaga Demak Dengan Metode Waterfall," *Rabit J. Teknol. Dan Sist. Inf. Univrab*, Vol. 9, No. 1, Pp. 79–90, Dec. 2023, Doi: 10.36341/Rabit.V9i1.4049.
[5] T. Lou, "A Comparison Of Android Native App Architecture – Mvc, Mvp And Mvvm," Oct. 2016, [Online]. Available: Https://Research.Tue.Nl/En/Studenttheses/A-Comparison-Of-Android-Native-App-Architecture
[6] C. Khawas And P. Shah, "Application Of Firebase In Android App Development-A Study," *Int. J. Comput. Appl.*, Vol. 179, No. 46, Pp. 49–53, Jun. 2018, Doi: 10.5120/Ijca2018917200.
[7] Khairun Nisa Meiah Ngafidin, A. Arista, And R. Nisa Sofia Amriza, "Implementasi Firebase Realtime Database Pada Aplikasi Feedbackme Sebagai Penghubung Guru Dan Orang

Received: Jan 21st, 2025 | Published: March 21st, 2025

DOI 10.30656/jsii.v11i1.6936
p-ISSN: 2406-7768
e-ISSN: 2581-2181

Tua," *J. Resti Rekayasa Sist. Dan Teknol. Inf.*, Vol. 5, No. 2, Pp. 327–334, Apr. 2021, Doi: 10.29207/Resti.V5i2.2909.

[8] T. Kurniawan, S. Samsudin, And T. Triase, "Implementasi Layanan Firebase Pada Pengembangan Aplikasi Sewa Sarana Olahraga Berbasis Android," *J. Inform. Univ. Pamulang*, Vol. 6, No. 1, P. 13, Mar. 2021, Doi: 10.32493/Informatika.V6i1.10270.

[9] Y. S. Rahayu, Y. Saputra, And D. Irawan, "Implementasi Metode Waterfall Pada Pengembangan Sistem Informasi Mobile E-Disarpus," *Zonasi J. Sist. Inf.*, Vol. 6, No. 2, Pp. 523–534, Jun. 2024, Doi: 10.31849/Zn.V6i2.20538.

[10] S. S. Ardian, A. Kurniadi, And M. A. Saputra, "Perancangan Sistem Antrian Sederhana Untuk Berbagai Layanan Berbasis Android," *Pros. Semin. Nas. Teknol. Dan Sains*, Vol. 2, No. 1, Pp. 255–260, Jan. 2023, Doi: Https://Doi.Org/10.29407/Stains.V2i1.2931.

[11] B. H. Rambe, R. Pane, D. Irmayani, M. Nasution, And I. R. Munthe, "Uml Modeling And Black Box Testing Methods In The School Payment Information System," Vol. 4, No. 36, 2020.

[12] S. Supriyono, "Software Testing With The Approach Of Blackbox Testing On The Academic Information System," *Ijistech Int. J. Inf. Syst. Technol.*, Vol. 3, No. 2, Pp. 227–233, 2020.

[13] D. Sharma And H. Dand, "Firebase As Baas For College Android Application," *Int. J. Comput. Appl.*, Vol. 178, No. 20, Pp. 1–6, Jun. 2019, Doi: 10.5120/Ijca2019918977.

[14] C. Informatika, Mohammad Bayu Anggara, And Fahrul Zaman, "Desain Arsitektur Server Google Cloud Untuk Mengoptimalkan Kinerja Aplikasi Daily Cloud Dalam Pemantauan Kesehatan Mental," *Comput. J. Inform.*, Vol. 11, No. 01, Pp. 14–21, Jun. 2024, Doi: 10.55222/Computing.V11i01.1480.

[15] B. Primin And A. P. Wibowo, "Implementasi Aplikasi Berbasis Mobile Untuk Pelayanan Jasa Kesehatan," *J. Inform. J. Pengemb. It*, Vol. 8, No. 2, Pp. 119–125, May 2023, Doi: 10.30591/Jpit.V8i2.5076.

[16] A. Romdhana, A. Merlo, M. Ceccato, And P. Tonella, "Deep Reinforcement Learning For Black-Box Testing Of Android Apps," *Acm Trans. Softw. Eng. Methodol.*, Vol. 31, No. 4, Pp. 1–29, Oct. 2022, Doi: 10.1145/3502868.

[17] A. B. Atmadjaja, "Rancang Bangun Aplikasi Kesehatan Berbasis Android: Integrasi Rekam Medis Elektronik Dan Komunikasi Dokter-Pasien," *J. Inform. Dan Sist. Inf.*, Vol. 9, No. 2, Pp. 54–62, Dec. 2023, Doi: 10.37715/Juisi.V9i2.4421.

[18] R. S. Pressman And D. Bruce R. Maxim, *Software Engineering: A Practitioner's Approach*. Mcgraw-Hill Education, 2014. [Online]. Available: Https://Books.Google.Co.Id/Books?Id=I8nmnaeacaaj

[19] K. U. Singh, N. Varshney, P. Gupta, G. Kumar, T. Singh, And S. R. Dogiwal, "Mobile Application Control With Firebase Cloud Messaging," In *Innovative Computing And Communications*, A. E. Hassanien, S. Anand, A. Jaiswal, And P. Kumar, Eds., Singapore: Springer Nature Singapore, 2024, Pp. 527–535.

[20] U. A. Madaminov And M. R. Allaberganova, "Firebase Database Usage And Application Technology In Modern Mobile Applications," In *2023 Ieee Xvi International Scientific And Technical Conference Actual Problems Of Electronic Instrument Engineering (Apeie)*, 2023, Pp. 1690–1694. Doi: 10.1109/Apeie59731.2023.10347828.

Received: Jan 21st, 2025 | Published: March 21st, 2025