



FACULTY OF COMPUTING
UTM Johor Bahru

SECJ3623: Mobile Application Programming

Technical Report

< Study Companion + >

Lecturer: AP Dr. Mohd Shahizan Othman

Prepared by: <Team name: Asal Boleh>

Team Member Name	Matric Number
Noor Amera Shafinaz	SX190392CSJS04
Nor Hazida	SX190856CSJS04
Syaiful Alfraiza	SX190855CSJS04

Table of Contents

No.	Item
1.0	Introduction
2.0	Development Approach and Workflow
2.1	Development Methodology
2.2	Development Workflow
3.0	Software, Tools, and Frameworks Used
3.1	Frontend Development
3.2	Development Workflows
3.3	Development Tools
4.0	Database and Data Handling Design
4.1	Overview of Data Management
4.2	Data Components and Structure
4.3	Data Access by User Roles
5.0	CRUD Operations and System Implementation
5.1	Overview of CRUD Operations
5.2	CRUD Capability by System Module
5.3	User Role Responsibilities and CRUD Scope
5.4	Example CRUD Implementation (Task Module)
6.0	Technical Challenges and Solutions
7.0	Conclusion

1. INTRODUCTION

In a school environment, academic activities such as assignments, announcements, and progress monitoring are often managed using multiple platforms or manual processes. This fragmented approach can result in missed deadlines, unclear communication, and limited visibility of student performance. Students may find it difficult to keep track of their academic tasks, teachers face challenges in monitoring class progress efficiently, and parents often have limited access to timely and accurate information about their children's learning activities. As educational institutions increasingly adopt digital solutions, there is a clear need for a centralized mobile application that supports effective coordination, improves information visibility, and ensures consistent communication among all stakeholders.

StudyCompanion+ is a role-based mobile application developed to address these challenges by providing a unified academic support platform. The system is designed for students, teachers, parents, and administrators, with each role accessing features relevant to their responsibilities. Core functionalities include task and assignment management, announcements, progress tracking, calendar scheduling, and real-time notifications. Developed using Flutter and supported by Firebase services, StudyCompanion+ enables secure authentication, real-time data synchronization, and scalable cloud-based data management. By centralizing academic information and communication, the application aims to improve efficiency, transparency, and engagement within the school ecosystem. This technical report explains how StudyCompanion+ was designed and implemented, covering the development workflow, system architecture, and database structure.

2. DEVELOPMENT APPROACH AND WORKFLOW

2.1 Development Methodology

The development of StudyCompanion+ followed an incremental and modular approach to ensure that system requirements were clearly defined and implemented in a structured manner. The project began with identifying user roles and core system functions, followed by gradual development of features based on priority and dependency. This approach allowed continuous refinement of the system while maintaining flexibility to incorporate feedback during development.

The methodology emphasizes clear separation between system components, including user interface, data handling, and business logic. By developing and validating each component in stages, the system could be tested and improved progressively, reducing integration risks and improving overall system stability.

2.2 Development Workflows

1. Requirement Analysis

The development process began with requirement analysis to identify the overall objectives and scope of the system. During this stage, the main user roles of the system, namely students, teachers, parents, and administrators, were identified. General functional expectations such as academic task management, information sharing, progress monitoring, and notifications were analyzed. This stage ensured that the system requirements were clearly defined before any design or implementation work was carried out.

2. System and Interface Design

After the requirements were established, the system and interface design stage was conducted. This stage focused on organizing the system into logical modules and defining how different users would interact with the application. User interface flows were designed to ensure ease of navigation and consistency across different roles. Emphasis was placed on creating a mobile-friendly interface that supports clear information presentation and smooth user interaction.

3. Implementation of Core System Components

Once the design was completed, system implementation was carried out in stages. Development started with core system components such as user authentication and role-based access control, which form the foundation of the application. After the core components were stable, additional functional modules were implemented incrementally. This staged implementation approach helped ensure that newly developed components could be integrated without affecting existing system functionality.

4. Integration and Testing

As development progressed, system components were continuously integrated and tested. Functional testing was performed to verify that each component operated as expected, while integration testing ensured that different modules worked together correctly. This stage helped identify errors, inconsistencies, and performance issues early in the development process, allowing corrective actions to be taken promptly.

5. Refinement and Final Validation

The final stage of the workflow involved refining the system based on testing outcomes and overall system review. Improvements were made to enhance usability, system performance, and interaction flow where necessary. Final validation was then carried out to ensure that the system met the defined requirements and was ready for deployment as a complete mobile application.

3 SOFTWARE, TOOLS, AND FRAMEWORKS USED

3.1 Frontend Development

StudyCompanion+ was developed using Flutter as the frontend framework, enabling cross-platform mobile application development from a single codebase. Flutter provides a responsive user interface and supports consistent behavior across Android and other supported platforms. The application logic is written using the Dart programming language.

Several Flutter packages were used to support application functionality, including state management, file handling, and local storage. These packages help manage application state, enable document access, and improve overall user experience.

3.2 Development Workflows

Firebase services were used to support backend functionality for StudyCompanion+. Firebase Authentication provides secure user login and role identification, while Firebase Cloud Firestore serves as the primary cloud database for storing academic and system-related data. Firebase Cloud Messaging is used to deliver notifications to users in real time.

3.3 Development Tools

Development and testing were carried out using standard tools such as Android Studio and Visual Studio Code. Version control was managed using Git and GitHub to support collaborative development and code tracking. Firebase Console was used for database management, authentication configuration, and monitoring application usage.

4 DATABASE AND DATA HANDLING DESIGN

4.1 Overview of Data Management

The system flow of StudyCompanion+ illustrates the sequence of interactions between users and the mobile application based on role-specific activities. Each flow begins with user authentication, where credentials are validated before access to the respective dashboard is granted. Upon successful login, the system routes the user to functions permitted by their assigned role.

Teachers perform task and announcement management, where created or updated data are stored in the database and notifications are dispatched to relevant users. Students retrieve assigned tasks, update completion status, and access announcements, with all actions recorded to maintain accurate progress tracking. Parents access monitoring functions to review their child's tasks, progress, and announcements without modifying academic records.

All user actions are processed by the application layer and synchronized with the database to ensure data consistency and real-time updates. The activity diagrams provide a visual representation of these operational workflows and depict the control flow between user actions and system responses.

4.2 Data Components and Structure

The data structure of StudyCompanion+ is designed around the core entities required to support academic management and communication within a school environment. The system uses Firebase Cloud Firestore, where data is organized into collections. Each collection represents a specific entity and stores related documents that support system functionality.

The data components are structured to ensure clear separation between user information, academic content, and system-related records. This structure improves data consistency, simplifies data retrieval, and supports scalable system growth as new features are introduced.

The main Firestore collections used in StudyCompanion+ are described in Table 4.1.

Table 4.1: Firestore Collections in StudyCompanion+

Collection Name	Description
users	Stores general user account information, including role type such as student, teacher, parent, or administrator
students	Stores student-specific profile and academic-related information
teachers	Stores teacher profile information and classroom associations
parents	Stores parent profile information
children	Stores the relationship between parent accounts and student accounts
classrooms	Stores classroom or class group information
subjects	Stores subject details associated with classrooms
assignments	Stores assignments created by teachers for students
tasks	Stores task-level records and student progress information
announcements	Stores announcements created by teachers for communication purposes
calendar_events	Stores academic calendar events and schedules
reports	Stores student performance and progress-related records

Figure 4.1 provides an example of how assignment-related data is stored and organized within the Firestore database.

```

{
  "assignmentId": "a1",
  "className": "cl1",
  "createdDate": "8 February 2026 at 10:00:00 UTC+8",
  "description": "Complete all practice problems from Chapter 4. Focus on quadratic equations.",
  "dueDate": "14 February 2026 at 23:59:59 UTC+8",
  "isPublished": true,
  "maxScore": "100",
  "subject": "Mathematics",
  "submittedCount": "4",
  "teacherId": "t1",
  "title": "Chapter 4 Review - Algebra",
  "totalStudents": "5"
}

```

Figure 4.1: Example Firestore collection structure for assignments in StudyCompanion+

This structured organization allows the system to manage data efficiently while supporting different user roles and academic activities. The separation of collections also helps maintain data clarity and simplifies future system maintenance and enhancement.

4.3 Data Access by User Roles

StudyCompanion+ applies a role-based data access approach to ensure that users can only access information relevant to their responsibilities. User roles are assigned during authentication and are used to control how academic and system data is accessed throughout the application. This approach supports data privacy, security, and clear separation of responsibilities.

The access scope for each user role in StudyCompanion+ is summarized in Table 4.2.

Table 4.2: Data Access by User Roles

User Role	General Data Access Scope
Student	Create, update, and delete own task-related records, and view assignments, announcements, and academic calendar information
Teacher	Create, read, update, and delete academic content such as assignments and announcements, and view student progress for assigned classes
Parent	Create, read, update, and delete parent-owned records such as calendar reminders or notification items, and view academic information related to their children
Administrator	Create, read, update, and delete user accounts and system-related records, and oversee overall system configuration

This role-based data access structure ensures that all user roles fulfil the required CRUD operations while maintaining controlled access to academic data. Detailed CRUD operations for each system module are presented in the subsequent chapter.

5 CRUD OPERATIONS AND SYSTEM IMPLEMENTATION

5.1 Overview of CRUD Operations

StudyCompanion+ implements Create, Read, Update, and Delete (CRUD) operations across its core system modules to support academic management and user interaction. CRUD functionality is implemented based on system requirements and controlled through role-based access. Each system module supports the necessary data operations, while individual user roles are granted access only to modules relevant to their responsibilities.

The CRUD implementation is designed to maintain data integrity and prevent unauthorized modification of academic records. Access to data operations is distributed across different modules and user roles in accordance with the functional responsibilities defined for the system.

5.2 CRUD Capability by System Module

Table 5.1 presents the CRUD capability supported by each major system module in StudyCompanion+. This table reflects system-level functionality and confirms that each module is designed to support appropriate data operations.

Table 5.1: CRUD Capability by System Module

Module	Create	Read	Update	Delete
User Management	✓	✓	✓	✓
Classroom	✓	✓	✓	✓
Subject	✓	✓	✓	✓
Assignment	✓	✓	✓	✓
Task	✓	✓	✓	✓
Announcement	✓	✓	✓	✓
Calendar Event	✓	✓	✓	✓
Notification / Reminder	✓	✓	✓	✓
Parent–Child Association	✓	✓	✓	✓
Reports / Performance	✓	✓	✓	—

Note: Delete operations for reports and performance records are restricted to preserve academic records and reporting integrity.

5.3 User Role Responsibilities and CRUD Scope

While full CRUD capability is supported at the system module level, access to these operations is restricted according to user roles. Table 5.2 summarizes how CRUD operations are distributed across different user roles in StudyCompanion+, ensuring that each role is provided with meaningful data interaction aligned with its responsibilities.

Table 5.2: User Role Responsibilities and CRUD Scope

User Role	Modules Involved	CRUD Scope
Student	Tasks	Create, Read, Update, Delete own task records
	Calendar Reminders	Create, Read, Update, Delete personal reminders
	Assignments	Read assignment details and update submission or completion status
	Notifications	Read and delete notification items
Teacher	Assignments	Create, Read, Update, Delete assignments
	Announcements	Create, Read, Update, Delete announcements
	Classroom & Subject	Create, Read, Update, Delete classroom and subject information
	Tasks	Read and update student task status
Parent	Calendar Reminders	Create, Read, Update, Delete parent-owned reminders
	Notifications	Read and delete notification items
	Child Academic Information	Read-only access
Administrator	User Management	Create, Read, Update, Delete user accounts
	Reports / Performance	Create, Read, Update reports through the Generate Report use case
	System Records	Create, Read, Update, Delete system-related data

Each user role is assigned specific data operations based on functional responsibilities within the system. This role-based CRUD distribution ensures effective system operation while maintaining appropriate access control, data integrity, and security across the application.

5.4 Example CRUD Implementation (Task Module)

This section presents an example of CRUD implementation for the Task module in StudyCompanion+. The Task module allows students to manage personal academic tasks, including creating tasks, viewing task lists, updating task status, and deleting tasks. The implementation uses Firebase Cloud Firestore and is handled through a dedicated service class.

Create Task

The create operation allows a student to add a new task record to the Firestore database.

```
class TaskService {  
    final FirebaseFirestore _firestore = FirebaseFirestore.instance;  
    final String _collectionName = 'tasks';  
  
    Future<String> createTask({  
        required String studentId,  
        required String title,  
        required DateTime dueDate,  
    }) async {  
        final taskData = {  
            'studentId': studentId,  
            'title': title,  
            'dueDate': Timestamp.fromDate(dueDate),  
            'status': 'pending',  
            'createdAt': Timestamp.now(),  
        };  
  
        final docRef = await _firestore  
            .collection(_collectionName)  
            .add(taskData);  
  
        return docRef.id;  
    }  
}
```

Read Tasks

The read operation retrieves all tasks associated with a specific student.

```
Future<List<QueryDocumentSnapshot>> getStudentTasks(String studentId) async {  
    final querySnapshot = await _firestore  
        .collection(_collectionName)  
        .where('studentId', isEqualTo: studentId)  
        .orderBy('createdAt', descending: true)  
        .get();  
  
    return querySnapshot.docs;  
}
```

Update Task Status

The update operation allows a student to mark a task as completed or update its status.

```
Future<void> updateTaskStatus(String taskId, String newStatus) async {  
    await _firestore  
        .collection(_collectionName)  
        .doc(taskId)  
        .update(  
            {'status': newStatus,  
             'updatedAt': Timestamp.now(),  
        });  
}
```

Delete Task

The delete operation removes a task record from the Firestore database.

```
Future<void> deleteTask(String taskId) async {
    await _firestore
        .collection(_collectionName)
        .doc(taskId)
        .delete();
}
```

This example demonstrates how CRUD operations are implemented within the StudyCompanion+ system using Firebase Cloud Firestore. Similar implementation patterns are applied across other system modules to ensure consistent data handling and maintainability.

6 TECHNICAL CHALLENGES AND SOLUTIONS

During the development of StudyCompanion+, several technical challenges were encountered due to the use of a cross-platform mobile framework, real-time cloud database, and role-based access requirements. This chapter discusses the key challenges faced and the solutions applied to ensure stable system functionality and data integrity.

6.1 Challenge: Role-Based Access Control Across Multiple User Types

Issue:

StudyCompanion+ supports multiple user roles, including students, teachers, parents, and administrators. Ensuring that each role could access only the relevant data and system functions was challenging, particularly when using a shared Firestore database.

Solution:

Role-based access control was implemented using Firebase Authentication combined with structured Firestore queries. User roles are identified during login and stored in the user profile. Application-level logic restricts access to specific features based on role, while Firestore queries are filtered to ensure users retrieve only authorized data.

6.2 Challenge: Structuring Firestore Data for Academic Relationships

Issue:

Managing relationships between students, teachers, parents, classrooms, and subjects in a NoSQL database required careful data modeling to avoid redundancy and complex queries.

Solution:

The system was designed using clearly separated Firestore collections such as users, classrooms, assignments, tasks, and parent-child associations. Reference IDs are used to link related records, allowing efficient data retrieval while maintaining flexibility and scalability.

6.3 Challenge: Real-Time Data Synchronization Across User Interfaces

Issue:

Academic data such as assignments, announcements, and task updates must be reflected immediately across different user dashboards. Manual refresh mechanisms would reduce usability and cause inconsistent views.

Solution:

Firestore's real-time snapshot listeners were used to synchronize data automatically. Stream-based data handling was integrated with Flutter widgets to ensure that changes in the database are immediately reflected in the user interface without requiring manual refresh actions.

6.4 Challenge: Managing Notification Lifecycle and Data Accumulation

Issue:

Frequent academic notifications can accumulate over time, leading to cluttered interfaces and unnecessary data storage if not properly managed.

Solution:

Notifications were implemented as user-owned records. Users are allowed to read and delete notification items once they are no longer needed. This approach improves usability and prevents excessive growth of notification data in the database.

6.5 Challenge: Protecting Academic Records from Accidental Deletion

Issue:

Academic records such as assignments and performance reports must be protected to maintain data accuracy and prevent accidental or unauthorized deletion.

Solution:

Delete operations for critical academic data were restricted to authorized roles such as teachers and administrators. Students and parents are permitted to update only status-related or personal records. This controlled deletion strategy ensures data integrity while supporting required system interactions.

6.6 Challenge: Supporting User-Specific Calendar Management

Issue:

The system includes both academic calendar events created by teachers and personal reminders created by students or parents. Differentiating access rights between these two types of calendar entries was necessary to avoid unintended data modification.

Solution:

Academic calendar events are treated as read-only for students and parents, while personal calendar reminders are stored as user-owned records. Users are allowed to create, update, and delete only their own reminder entries, ensuring clear separation between system-managed and user-managed data

7 CONCLUSION

StudyCompanion+ was developed to address common challenges in managing academic activities within a school environment, particularly issues related to task tracking, communication, and visibility of student progress. By providing a centralized mobile application, the system supports students, teachers, parents, and administrators through clearly defined roles and responsibilities. Core features such as assignment management, task tracking, announcements, calendar scheduling, notifications, and reporting enable more organized and transparent academic coordination.

The system was implemented using Flutter for cross-platform mobile development and Firebase services for authentication and cloud-based data management. A structured database design and role-based access control were applied to ensure data security, integrity, and scalability. CRUD operations were successfully implemented across system modules, with access appropriately restricted based on user roles. Overall, StudyCompanion+ demonstrates a practical and effective approach to supporting academic management through a modern mobile application, fulfilling the project objectives and providing a solid foundation for future enhancements..