VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**DAO TUONG LAM QUYEN ANH – 520H0509**

**KIEU THANH PHAT – 521H0125**

**NGUYEN DINH VIET HOANG - 522H0120**

**DEVELOPING A REALTIME APP FOR STUDENT INFORMATION MANAGEMENT**

**MIDTERM REPORT**

**MOBILE APPS DEVELOPMENT**

**HO CHI MINH CITY, YEAR 2024**

VIETNAM GENERAL CONFEDERATION OF LABOUR

**TON DUC THANG UNIVERSITY**

**FACULTY OF INFORMATION TECHNOLOGY**



**DAO TUONG LAM QUYEN ANH – 520H0509**

**KIEU THANH PHAT – 521H0125**

**NGUYEN DINH VIET HOANG - 522H0120**

**DEVELOPING A REALTIME APP FOR STUDENT INFORMATION MANAGEMENT**

**MIDTERM REPORT**

**MOBILE APPS DEVELOPMENT**

Advised by

**MsC. VU DINH HONG**

**HO CHI MINH CITY, YEAR 2024**

**ACKNOWLEDGEMENT**

We sincerely thank MsC. Vu Dinh Hong for teaching us the Mobile Apps Development course with great enthusiasm. We want to express our deep appreciation for the dedication and professional knowledge that you shared with us. Through your classes, we gained a better understanding of the fundamental aspects of the Mobile Apps Development, thanks to your detailed explanations and practical applications. You helped us grasp the knowledge and apply it effectively. Finally, we extend our heartfelt gratitude to MsC. Vu Dinh Hong for your commitment and invaluable support throughout our learning journey in this course. The skills and knowledge we acquired will continue to impact our future development. We sincerely thank you and wish your health, success, and happiness.

*Ho Chi Minh City, November 14, 2024*

*Authors:*

*Dao Tuong Lam Quyen Anh*

*Kieu Thanh Phat*

*Nguyen Dinh Viet Hoang*

**DECLARATION OF AUTHORSHIP**

We hereby declare that this thesis was carried out by ourselves under the guidance and supervision of MsC. Vu Dinh Hong; and that the work and the results contained in it are original and have not been submitted anywhere for any previous purposes. The data and figures presented in this thesis are for analysis, comments, and evaluations from various resources by our own work and have been duly acknowledged in the reference part.

In addition, other comments, reviews and data used by other authors, and organizations have been acknowledged, and explicitly cited.

We will take full responsibility for any fraud detected in my thesis. Ton Duc Thang University is unrelated to any copyright infringement caused on my work (if any).

*Ho Chi Minh City, November 14, 2024*

*Authors:*

*Dao Tuong Lam Quyen Anh*

*Kieu Thanh Phat*

*Nguyen Dinh Viet Hoang*

**DEVELOPING A REALTIME APP FOR STUDENT INFORMATION MANAGEMENT**

**ABSTRACT**

The Student Information Management (SIM) system is a real-time mobile application designed to streamline the management of student data by allowing administrators, managers, and employees to access and update information efficiently. By integrating Firebase Firestore, the SIM system leverages NoSQL document-based storage to enable secure, synchronized, and scalable management of student records, academic details, and certifications. This role-based platform supports secure login, profile management, data import/export, and multi-criteria search capabilities, ensuring that each user role has tailored permissions for optimal data handling. Admins manage user and student data comprehensively, while managers have rights to update student details and employees can access information within their scope. Real-time data synchronization enhances data consistency across connected devices, enabling instant updates. This application offers a user-centric, efficient approach to student data management, ensuring data integrity, accessibility, and adaptability within educational institutions.

**TABLE OF CONTENTS**

[LIST OF FIGURES viii](#_Toc182677772)

[ABBREVIATIONS ix](#_Toc182677773)

[CHAPTER 1. PROJECT INTRODUCTION 1](#_Toc182677774)

[1.1 Student Information Management 1](#_Toc182677775)

[1.2 System Scope 1](#_Toc182677776)

[1.3 Detailed System Description 2](#_Toc182677777)

[CHAPTER 2. THEORETICAL SURVEY 4](#_Toc182677778)

[2.1 What is Firebase Firestore? 4](#_Toc182677779)

[2.2 What Is Firebase Realtime Database? 6](#_Toc182677781)

[2.3 Firestore Database Architecture 6](#_Toc182677782)

[2.4 Firebase Authentication 7](#_Toc182677784)

[2.5 Realtime Database vs. Firestore 7](#_Toc182677785)

[2.6 Query and Filter Functions in Firestore 8](#_Toc182677787)

[2.7 Real-Time Data Update and Synchronization 9](#_Toc182677788)

[2.8 Security and Access Control in Firestore 9](#_Toc182677789)

[2.9 Data Import and Export in Firestore 10](#_Toc182677790)

[2.10 Firestore SDK and REST API 10](#_Toc182677791)

[2.11 Firestore Indexing 10](#_Toc182677792)

[2.12 Development and Management Tools for Firestore 11](#_Toc182677793)

[CHAPTER 3. STUDENT MANAGEMENT APPLICATION (SIM) ON MOBILE PLATFORM 12](#_Toc182677794)

[3.1 Requirements Analysis 12](#_Toc182677795)

[3.2 System Scope 12](#_Toc182677796)

[3.3 UML Use Case Diagram 14](#_Toc182677797)

[3.4 UML Class Diagram 20](#_Toc182677799)

[3.5 UML Entity Relationship Diagram 22](#_Toc182677801)

[3.6 Implementation 27](#_Toc182677803)

[3.6.1 The Login Form 27](#_Toc182677804)

[3.6.2 The User Management Form 28](#_Toc182677805)

[3.6.3 The Student Management Form 29](#_Toc182677806)

[3.6.4 The Setting Form 30](#_Toc182677807)

[CHAPTER 4. CONCLUSION 31](#_Toc182677808)

[4.1 Achieved Outcomes 31](#_Toc182677809)

[4.2 Limitations 31](#_Toc182677810)

[4.3 Suggestions for Future Enhancements 31](#_Toc182677811)

[REFERENCES 32](#_Toc182677812)

# LIST OF FIGURES

[Figure 2.1: Document-based architecture 5](#_Toc182522302)

[Figure 2.2: Firestore Database Architecture in real life 6](#_Toc182522305)

[Figure 2.4: Compare Realtime Database and Firestore 8](#_Toc182522308)

[Figure 3.3: Use Case Diagram 14](#_Toc182522320)

[Figure 3.4: Class Diagram 20](#_Toc182522322)

[Figure 3.5: Entity Relationship Diagram 22](#_Toc182522324)

# ABBREVIATIONS

|  |  |
| --- | --- |
| SIM | Student Information Management |
| CSV | Comma-Separated Values |
| SDK | Software Development Kit​ |
| API | Application Programming Interface​ |
| I/O | Input/Output |
| CI/CD | Continuous Integration/Continuous Delivery​ |
| UTS | UNIX Time-Sharing |
| PID | Process Identifier |
| HTTP | Hypertext Transfer Protocol​ |

# CHAPTER 1. PROJECT INTRODUCTION

1.1 Student Information Management

The Student Information Management (SIM) application is a meticulously designed system aimed at facilitating the management of student data in an easy and efficient way. This system allows for the management of students' personal information, grades, and certificates, while emphasizing user-friendliness and convenience. SIM enables quick additions and updates to student information and supports user profile management along with role assignments based on specific responsibilities. The system includes three main user roles: Student, Manager, and Admin, ensuring a customized and seamless experience for each user group.

1.2 System Scope

The SIM system includes the following components:

1. Login
2. User Management
3. Student Management
4. Import/Export of Certificates
5. Role Separation

SIM incorporates a secure and user-friendly login system, ensuring a role-based access hierarchy for Admin, Manager, and Employee. This fundamental feature forms the foundation for the user management functionalities within the system. The Admin has access to powerful tools for user management, including the ability to change profile pictures for all users and monitor a comprehensive user list. They can add new users and collect essential information such as name, age, phone number, and status (normal or locked). Admins also have the authority to delete and modify user information, creating a flexible and easily adjustable management environment.

In terms of student management, SIM provides both Admin and Managers with intuitive tools to easily handle student data. The system allows for a detailed view of student lists, as well as the ability to add or update information. Users can sort the student list based on various criteria and conduct multi-criteria searches, offering a comprehensive view of each student’s information, including academic certificates.

Beyond student management, SIM supports the import and export of student data and certificates, enhancing data accessibility and compatibility. The import functions allow Admins and Managers to upload lists of students and certificates from the database, while the export functions ensure data is presented in a user-friendly Excel or CSV format.

Additionally, SIM has a well-defined role separation feature for three main user roles: Admin, Manager, and Employee. Admin accounts are directly integrated into the system without additional configuration, giving them full control over all system functions. Managers are granted permission to perform tasks related to student management, while Employees have limited access to view system content and change their profile picture.

1.3 Detailed System Description

The Student Information Management System (SIM) was developed to make student information management more straightforward and efficient. Below is an outline of each user role and their respective permissions:

* Admin: Admins have control over the entire system’s operations. They can track login activity, perform updates, delete or add new user and student information. In addition, Admins can change their profile picture, as well as import and export student data and certificates.
* Manager: Managers, whose accounts are created by the Admin, are authorized to manage student-related information. They can import and export data, add or

delete student records, and edit student details. Managers also have permission to change their own profile picture.

* Employee: Employees, whose accounts are also created by the Admin, mainly have viewing rights within the system without the ability to modify data. Their actions are primarily focused on accessing information, though they are still allowed to change their profile picture.

# CHAPTER 2. THEORETICAL SURVEY

2.1 What is Firebase Firestore?

Cloud Firestore is an innovative, cloud-hosted NoSQL database from Google that allows developers to manipulate their data with ease. It uses intuitive and familiar features for storing, syncing, and querying data for web and mobile app development.

Firestore offers convenience as well as scalability with its intelligent caching capability that stores the most frequently used queries, thus minimizing latency when accessing data. Development teams can also benefit from intuitive usage through the integration of both real-time capabilities (sync) and offline support in queries.

Its real-time capabilities keep data updated across all connected clients as soon as changes are made. This means that users will always have access to the most up-to-date available version of their documents no matter where they’re stored.

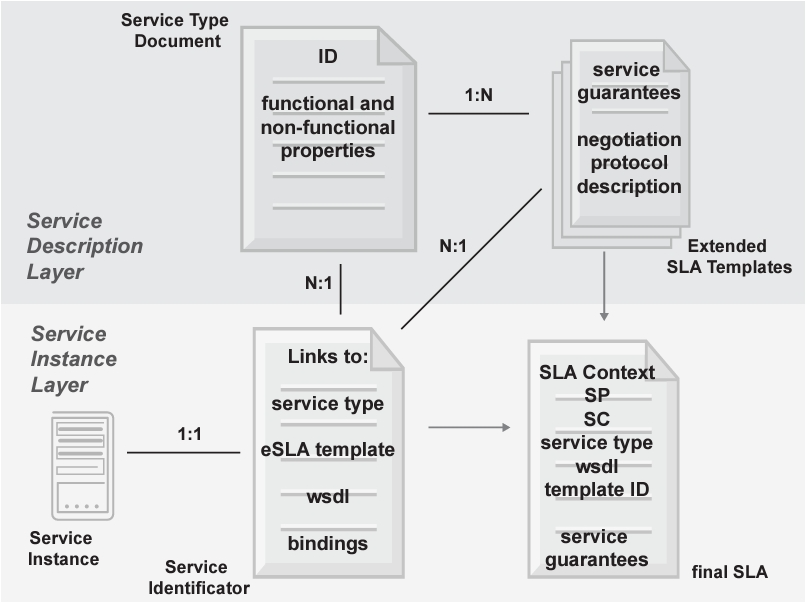


Figure 2.1: Document-based architecture

+ In the Figure 2.1, we can see that:

* Collections: Collections contain documents of the same type. For example, the "users" collection contains documents representing users.
* Documents: Each document is a data record, which can contain many different fields. For example, a document in the "users" collection can contain fields such as "name", "email", "address", etc.
* Fields: Fields contain data values. For example, the "name" field contains the name of the user.
* Nested documents: Documents can be nested to create more complex data structures.

2.2 What Is Firebase Realtime Database?

Realtime Database is a cloud-hosted database service provided by Google Firebase that synchronizes data between the user’s device and the cloud without manually refreshing the page.

It allows you to build powerful and complex applications without having to worry about networking or data storage. It also facilitates interactive web and mobile apps, making it easier to manage data and create real-time experiences for users.

This helps businesses easily integrate real-time updates into their applications and enables them to keep customers informed if anything changes with their products or services.

Realtime Database simplifies the way we access and update information from anywhere, creating efficient user experiences across devices.

2.3 Firestore Database Architecture

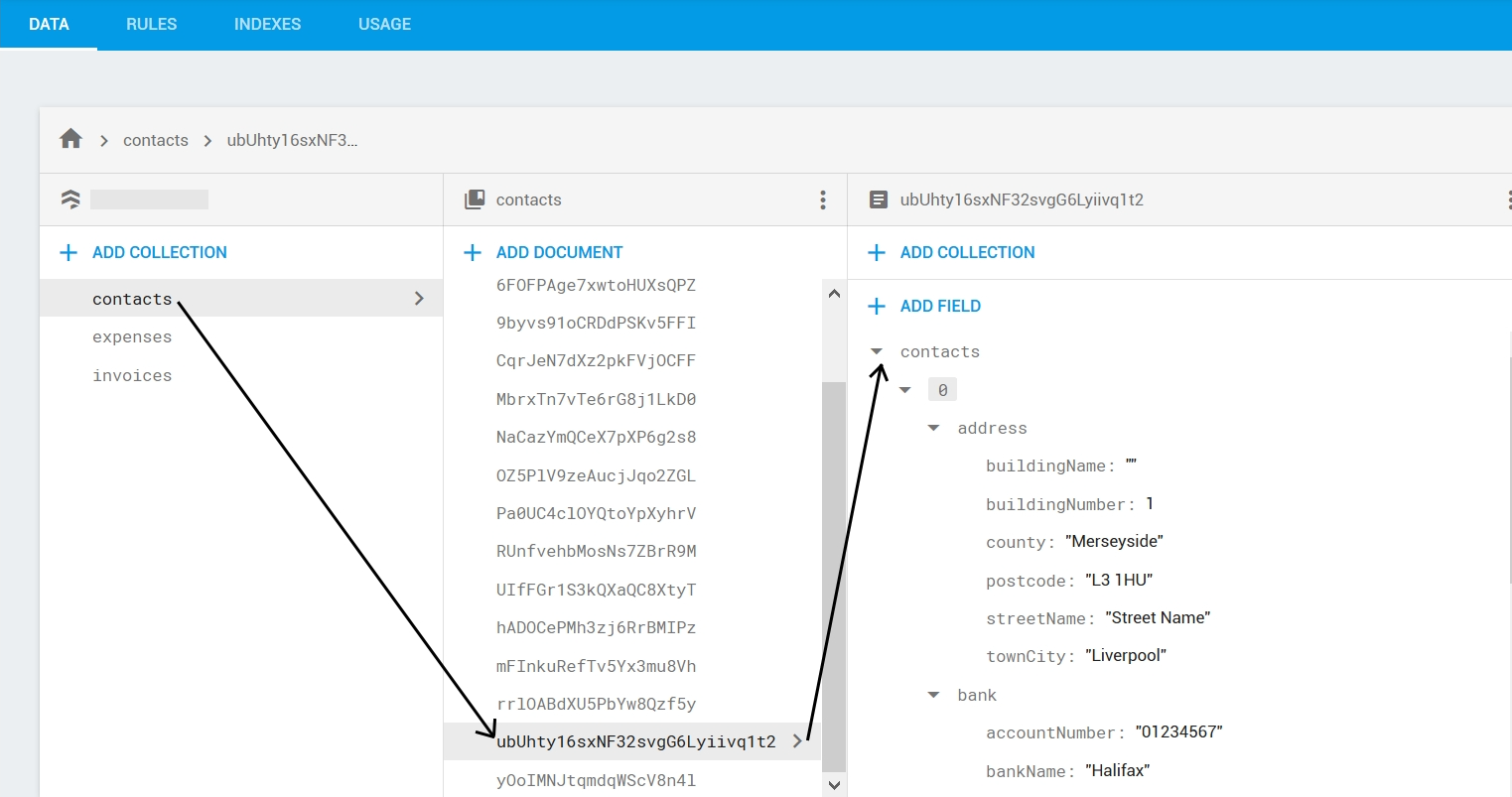


Figure 2.3: Firestore Database Architecture in real life

In the Figure 2.2, we can see each Firebase project can have multiple Firestore databases, each database contains collections, collections contain documents and each document is a data record.

Additional definitions of the keywords:

* + Collections: A group of related documents. Each document belongs to a collection, and a collection can contain many documents. Collections are the primary data organization unit in Firestore.
  + Documents: The main storage units in Firestore, containing application data as key-value pairs. Each document has a unique ID and can contain multiple fields as well as sub-collections.
  + Fields: Attributes within each document, similar to columns in traditional databases. A field can contain basic data types (strings, numbers, booleans) or complex types (arrays, maps).
  + Sub-collections: Collections within a document, creating a hierarchical structure between collections and documents. Firestore supports sub-collections to store detailed data related to each document.

2.4 Firebase Authentication

Firebase Authentication is a user authentication service provided by Firebase that helps applications manage and secure user accounts easily. While the project does not require Firebase Authentication, it is a popular solution suitable for implementing login, registration, and session management functions.

Firebase Authentication supports email/password, phone number, Google account, Facebook, and other methods, providing flexibility and security for user access to the application.

2.5 Realtime Database vs. Firestore

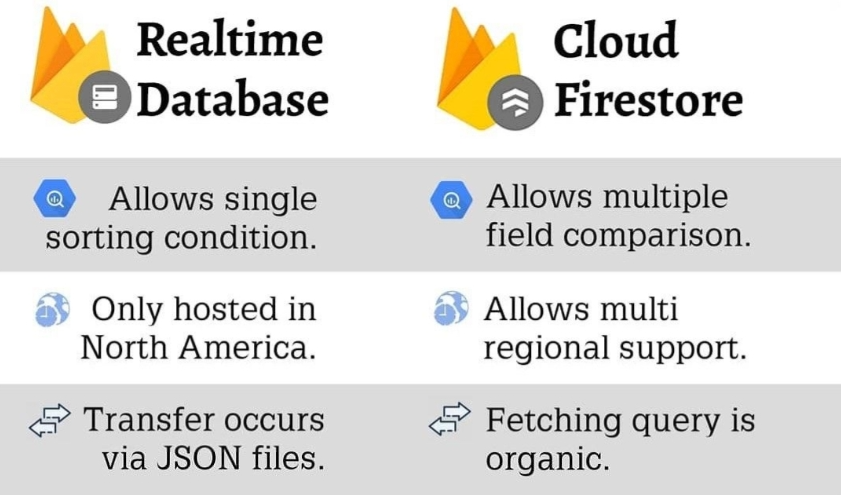


Figure 2.4: Compare Realtime Database and Firestore

From Figure 2.4, we can indicate that Firestore uses a document-collection structure, whereas Realtime Database uses a JSON tree structure. Moreover, Firestore supports stronger queries, handles large-scale data with a hierarchical structure, whereas Realtime Database is mainly suited for applications with simple data structures. In addition, Firestore allows filtering and sorting data based on multiple conditions without downloading the entire dataset to the client.

2.6 Query and Filter Functions in Firestore

Firestore supports complex queries, making it easy to filter and sort data based on different criteria. This is especially useful for managing student lists, searching for students, and sorting lists based on various criteria.

Firestore allows:

* + **Conditional Queries**: Filter data based on specific conditions (e.g., finding students with a specific name or ID).
  + **Sorting Data**: Sort results by one or more fields.
  + **Combined Queries**: Combine multiple conditions to produce the desired result.

Firestore supports real-time queries, meaning that when data changes, connected clients are immediately notified and can update the data.

2.7 Real-Time Data Update and Synchronization

Firestore offers real-time synchronization, allowing updates to be instantly reflected on all connected devices. This feature is ideal for building a real-time student management application, as data will synchronize as soon as changes are made, ensuring that administrators, managers, and staff always have the latest data.

Firestore supports Listeners, which allow the application to listen for changes in a specific collection or document. When data changes, the Listener will automatically update the user interface with the new data.

2.8 Security and Access Control in Firestore

Firestore Security Rules: Firestore provides Security Rules to protect data, allowing you to define access rights for each document and collection based on specific conditions. These rules are applied in real time and can be customized based on the application’s requirements.

To implement the system requirements for the three user roles (Admin, Manager, and Employee), Firestore Security Rules can be set up to allow only:

* + Admin full access to data, including add, delete, and update permissions.
  + Manager restricted access to add, modify, and view student information but no permissions to alter system settings or security configurations.
  + Employee view-only access to data without modification rights, except for updating their own profile picture.

Custom Claims: Combined with Firebase Authentication, Firestore supports Custom Claims to grant access rights based on user roles. For example, Custom Claims can label users as Admin, Manager, or Employee and apply corresponding access permissions.

2.9 Data Import and Export in Firestore

Data Import: Firestore allows data import from JSON or CSV files through support tools or programmatically. For a student management application, Admins can import student lists or certificates from files for easier management.

Data Export: Firestore allows data export in formats such as JSON or CSV. This is useful for creating reports, generating statistics, or storing data offline when needed. For the requirement to export student and certificate data, Firestore can be combined with libraries like papaparse (for CSV) or exceljs to export data to Excel or CSV files easily.

2.10 Firestore SDK and REST API

Firestore provides Firebase SDKs for multiple platforms like web, iOS, and Android, enabling easy integration and development with Firestore. Firestore SDKs support full functionality such as add, edit, delete, and real-time data synchronization.

Additionally, Firestore also offers a REST API, allowing data access and management from server applications or environments without SDK support.

2.11 Firestore Indexing

Firestore automatically indexes each field in documents to support efficient queries. However, custom indexes are needed for complex queries. This optimizes the application’s performance when querying large datasets like student lists or certificates.

Composite Index: Firestore allows creating composite indexes to support multi-condition queries, such as sorting students by name and age.

2.12 Development and Management Tools for Firestore

Firebase Console: An online Firestore administration interface, allowing developers to manage the database, view and edit data, set security rules, and monitor system metrics.

Firebase Emulator Suite: Firebase’s emulation toolset allows development and testing of Firestore without connecting to the live database. This is particularly useful for testing security rules and authentication before deploying to production.

# CHAPTER 3. STUDENT MANAGEMENT APPLICATION (SIM) ON MOBILE PLATFORM

3.1 Requirements Analysis

The Student Information Management (SIM) mobile application is designed to enable efficient and straightforward management of student data. This application focuses on handling students' personal information, including academic records and certificates, with an emphasis on ease of use and user-friendliness. SIM allows for quick updates and additions to student information and supports user profile management and role-based access control across three roles: Admin, Manager, and Employee, ensuring a customized experience for each user group.

3.2 System Scope

The SIM mobile system provides the following key features:

1. Login
2. User Management
3. Student Management
4. Certificate Import/Export
5. Role-based Access Control

SIM integrates a secure, user-friendly login system, establishing a clear access hierarchy for each role. **Admin** users have powerful management tools, including the ability to update user profile pictures, monitor a comprehensive user list, add new users, and manage personal details such as name, age, phone number, and status (Normal or Locked). The SIM system creates a flexible, easily adjustable environment for management tasks.

For student management, Admins and Managers have access to intuitive tools for handling student data. Users can view detailed student lists, add or update student information, and use sorting and multi-criteria search functions to retrieve comprehensive information about each student, including grades and certificates.

In addition, SIM supports the import and export of student data and certificates, enhancing data accessibility and integration. Admins and Managers can upload student and certificate lists from the database using the import function, while the export function formats data into Excel or CSV files for user convenience.

Lastly, SIM sets up role-based access control for three primary user roles. The Admin account is integrated directly into the system, granting full control over all functions. Managers have permission to perform student-related tasks, while Employees primarily have view-only rights, allowing them to access content and update only their profile pictures.

3.3 UML Use Case Diagram

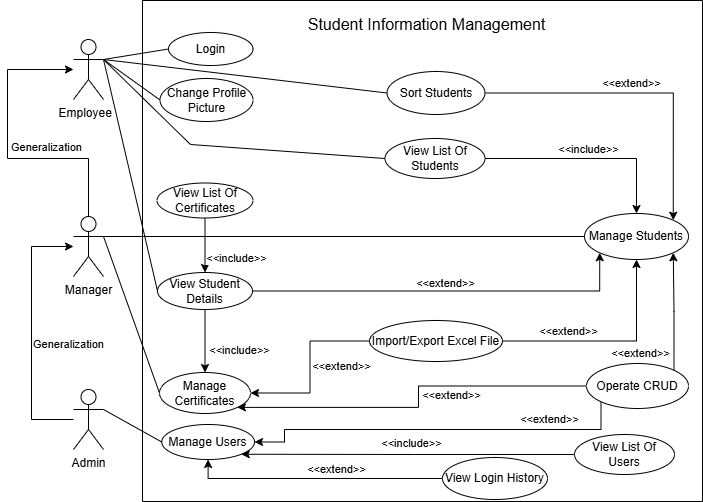


Figure 3.3: Use Case Diagram

+ Use Case 1: User Authentication

* Actors: Administrator, Manager, Employee
* Description: Users log into the system, with their assigned roles determined by their login credentials.
* Preconditions: The system is operational, and users possess valid credentials.
* Postconditions: Users gain access to the system upon successful login.
* Main Flow: Users begin the login process. The system requests a username and password. Users enter their credentials, which the system then verifies. If authentication succeeds, access is granted; if not, an error message is displayed.

+ Use Case 2: Profile Picture Update

* Actors: Administrator, Manager, Employee
* Description: Logged-in users can update their profile pictures.
* Preconditions: Users are signed into the system.
* Postconditions: The user's profile picture is updated to the selected image.
* Main Flow: Users go to their profile settings, where the current profile picture is displayed. They select an option to upload a new image, choose a file, and the system updates their profile picture.

+ Use Case 3: View List Of Students

* Actors: Administrator, Manager, Employee
* Description: Users access a complete list of students within the system.
* Preconditions: Users are logged in.
* Postconditions: Users view an overview of all students.
* Main Flow: Users enter the student management section, where the system retrieves and presents the student list in a table.

+ Use Case 4: Access Student Details Page

* Actors: Administrator, Manager, Employee
* Description: Users open a detailed page showing comprehensive information about a specific student.
* Preconditions: Users are logged into the system.
* Postconditions: Users see full details about the selected student.
* Main Flow: Users navigate to the student management section and access a list of students. After selecting a student, the system displays a page with all pertinent details. Employees, as restricted users, have view-only access to this information.

+ Use Case 5: Manage Students

* Actors: Manager, Administrator
* Description: Users manage student records by adding, updating, deleting, and importing/exporting student data, as well as managing their academic qualifications.
* Preconditions: Managers or Administrators are authenticated and logged in to the Student Management application.
* Postconditions: Student data is accurately modified, added, or exported as per the Manager’s or Admin’s actions.
* Main Flow:

1. Update Student Information: The Manager or Admin enters the "Student" section, selects the student to be updated, modifies the relevant details, and submits the changes. The system verifies and updates the student record accordingly.
2. Delete Student: The Manager or Admin navigates to the "Student" section, identifies the student to be removed, and confirms the deletion. The system then removes the student’s record from the database.
3. Add New Student: The Manager or Admin accesses the "Student" section, inputs the necessary details for a new student, and submits the entry. The system validates the provided data and adds the new student to the database.
4. Import/Export Student Data: The Manager or Admin goes to the "Student" section and selects the option for "Import/Export Student Data." If importing, they upload an Excel file containing student information. If exporting, the system generates an Excel file containing student records and makes it available for download. The system confirms the validity of the imported data (for import operations) and updates the student records accordingly, or completes the export and provides the file for download (for export operations).

+ Use Case 6: Manage Certificates

* Actors: Manager, Admin
* Description: Users manage student certificates, including tasks like adding, updating, deleting, and importing/exporting certificate data.
* Preconditions: The Manager or Admin is logged into the Certificate Management application, with access to the student details page.
* Postconditions: Certificate data is modified, added, or exported as appropriate based on the actions taken by the Manager or Admin.
* Main Flow:

1. Update Certificate Information: The Manager or Admin navigates to the "Student Details" section, selects the certificate to be updated, modifies the necessary information, and submits the updates. The system validates the input and updates the certificate information.
2. Delete Certificate: The Manager or Admin enters the "Student Details" section, identifies the certificate to be deleted, and confirms the action. The system then removes the certificate record from the database.
3. Add New Certificate: Within the "Student Details" section, the Manager or Admin fills in the necessary information for a new certificate and submits the form. The system verifies the details and adds the certificate to the student's records.
4. Import/Export Certificate Data: The Manager or Admin goes to the "Student Management" section, selects a student, and then navigates to "Import/Export Certificate Data" in the student’s detail section. If importing, they upload an Excel file containing certificate data. If exporting, the system provides two options: Option 1 allows exporting certificates for the selected student, while Option 2 allows exporting all certificates in the system. For Option 1, the system generates an Excel file with certificates specific to that student; for Option 2, the system prepares a file with all certificate records. The system validates imported data and updates the student’s certificate records as needed.

+ Use Case 7: Manage Users

* Actor: Admin
* Description: The Admin manages user accounts, including updating, deleting, adding new users, viewing a list of users in the system, and accessing login history.
* Preconditions: The Admin is authenticated and has logged into the User Management application.
* Postconditions: User data is adjusted, added, or removed as per the Admin’s actions.
* Main Flow:

1. Update User Information: The Admin goes to the "User" section, selects the user whose details need updating, makes the necessary changes, and submits them. The system verifies and applies the updates.
2. Delete User: In the "User" section, the Admin identifies the user to delete, confirms the action, and the system removes the user from the system.
3. Add New User: The Admin accesses the "User" section, fills out the required details for a new user, and submits the form. The system validates the input and adds the user to the database.
4. View List of System Users: The Admin goes to the "User" section, where the system retrieves and displays a complete list of users in the system.
5. View Login History of a User: In the "User" section, the Admin selects a user to view their login history. The system displays a record of the selected user’s login activity.

+ Use Case 8: Sort Students Using Multiple Criteria

* Actors: Admin, Manager, Employee
* Description: Users initiate sorting of student records by selecting the desired column header, and the system then displays the sorted results based on the selected column.
* Preconditions: Users are logged into the system.
* Postconditions: Users receive sorted results in the student table according to the chosen column.
* Main Flow: Users access the student management section, click on the column header by which they wish to sort the list, and the system rearranges and displays the student records based on that column.

3.4 UML Class Diagram

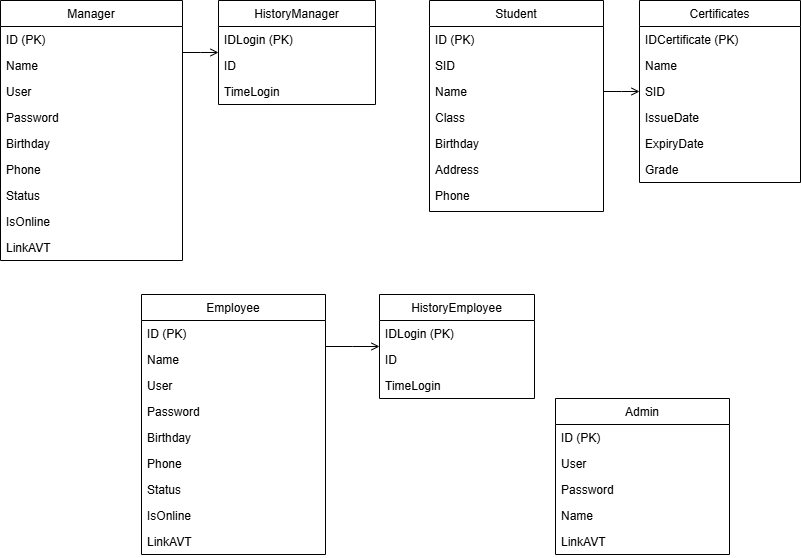


Figure 3.4: Class Diagram

+ Manager: Attributes for this class include: ID, Name, Username, Password, BirthDate, Phone Number, Status, Online Status (isOnline), and Avatar Link (LinkAVT).

* Related Class: HistoryManager – This class inherits properties from the Manager class and adds attributes for Login and Login Time.
* Relationship: There is a one-to-many association between Manager and HistoryManager, signifying that a single manager may possess multiple login records.

+ Student: Attributes encompass: ID, Student ID (SID), Name, BirthDate, Class, Address, and Phone Number.

* Related Class: Certification – This class includes attributes such as Certificate ID, Certificate Name, Issue Date, Expiry Date, and Grade.
* Relationship: A one-to-many relationship exists between Student and Certification, allowing a single student to hold multiple certifications.

+ Employee: Attributes include: ID, Name, Username, Password, BirthDate, Phone Number, Status, Online Status (isOnline), and Avatar Link (LinkAVT).

* Related Class: HistoryEmployee – This class records the ID, Action, and Action Time for employees, documenting activities as history.
* Relationship: The Employee and HistoryEmployee classes share a one-to-many relationship, meaning that each employee may have several entries documenting past actions.

+ Admin: Attributes are: ID, Name, Username, Password, and Avatar Link (LinkAVT).

* Special Status: Administrators possess the highest level of access within the system, granting them full system privileges.
* Summary: The system comprises distinct classes, each with specialized attributes and interrelationships. The Manager class records login data using the HistoryManager class; Student objects can be associated with multiple Certification records; Employee actions are recorded through the HistoryEmployee class; and Admin users hold the top level of access privileges within the system.

3.5 UML Entity Relationship Diagram

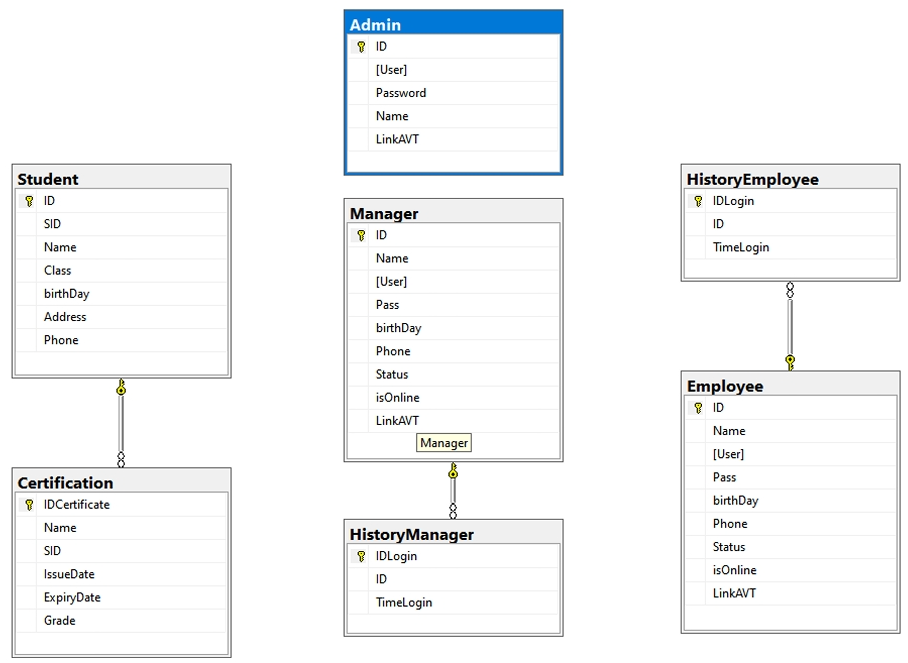


Figure 3.5: Entity Relationship Diagram

1. Admin Table:

+ Attributes:

* + ID: Serves as the primary key, uniquely identifying each admin.
  + User: Stores the administrator’s username.
  + Password: Holds the admin’s password.
  + Name: Keeps the admin's name.
  + LinkAVT: Defaults to 'user.png', storing the path to the admin’s avatar image.

+ Rationale: The Admin table is built to store essential information about each admin account. The ID functions as a unique primary key to ensure distinct identification. For enhanced security, it is recommended that the password be stored using secure methods like hashing and salting. The default avatar image path provides a placeholder image when no custom avatar is uploaded.

1. Manager Table:

+ Attributes:

* ID: Primary key ensuring unique identification for each manager.
* Name: Stores the manager's full name.
* User: A unique constraint ensures no duplicate usernames.
* Password: Holds the manager’s password.
* Birthday: Stores the manager's date of birt
* Phone: Saves the manager’s contact number
* Status: Default is 0 (active), with values like 1 (blocked) and -1 (deleted) indicating account states.
* IsOnline: Default is 0, where 1 means online, and 0 means offline.
* LinkAVT: Defaults to 'user.png', linking to the manager's avatar image.

+ Rationale: The Manager table is intended to hold detailed records for each manager. The primary key (ID) ensures that each record is unique, and the unique username constraint maintains individuality. Default values for Status and isOnline set initial states, while the avatar link uses a default image when no custom upload is provided.

1. Employee Table:

+ Attributes:

* ID: Primary key to uniquely identify each employee.
* Name: Stores the name of the employee.
* User: Unique constraint ensuring a unique login name for each user.
* Password: Holds the employee’s password.
* Birthday: Records the birth date of the employee.
* Phone: Stores the employee's contact number.
* Status: Default is 0 (active); other states are 1 (blocked) and -1 (deleted)
* IsOnline: Default is 0; this field tracks if the employee is online.
* LinkAVT: Defaults to 'user.png', storing a link to the employee’s avatar image.

+ Rationale: This table holds comprehensive employee information. The unique primary key (ID) prevents duplicate records, and the unique username ensures a distinct login. To improve security, passwords should be securely stored with hashing and salting. Default values for Status and isOnline set initial conditions, while LinkAVT defaults to a standard image if no custom avatar is added.

1. Student Table:

+ Attributes:

* ID: Primary key, uniquely identifying each student.
* SID: Unique constraint ensures each student ID is distinct.
* Name: Holds the student’s full name.
* Class: Captures the student’s class or section.
* Birthday: Records the birth date of the student.
* Address: Stores the student’s address.
* Phone: Saves the student’s phone number.

+ Rationale: This table captures vital details about each student. The primary key (ID) ensures unique identification, while the unique SID enforces distinct student IDs. Essential fields such as name, class, birth date, address, and phone are stored to maintain a detailed student profile.

1. Certificates Table:

+Attributes:

* + IDCertificate: Primary key to uniquely identify each certification record.
  + Name: Stores the name of the certification.
  + SID: Links to the student ID associated with this certification
  + IssueDate: Records when the certification was issued.
  + ExpiryDate: Indicates the certification’s expiry date.
  + Grade: Contains information on the grade, score, or level achieved in the certification.

+ Constraints:

* + FK\_StudentCertificate: Foreign key linking to the SID field in the Student table, ensuring each certification is tied to an existing student.

+ Rationale: The Certificates table holds details about certifications, uniquely identified by IDCertificate. The foreign key FK\_StudentCertificate links certifications to students, enforcing data integrity by preventing associations with non-existent students. Fields for issue date, expiry date, and grade provide a complete profile of each certification.

1. HistoryManager Table:

+ Attributes:

* + IDLogin: Primary key, uniquely identifying each login record.
  + ID: References the manager’s ID associated with the login event.
  + TimeLogin: Stores the timestamp of the login event.

+ Constraints:

* + FK\_HisLoginM: Foreign key linking to the ID field in the Manager table, ensuring each login record is connected to a valid manager.

+ Rationale: The HistoryManager table tracks login events for managers, with each record uniquely identified by IDLogin. The foreign key FK\_HisLoginM connects login records to manager accounts, upholding data consistency by ensuring all login entries correspond to existing managers.

1. HistoryEmployee Table:

+ Attributes:

* + IDLogin: Primary key, uniquely identifying each login history record for employees.
  + ID: References the employee’s ID associated with the login event.
  + TimeLogin: Stores the date and time of each login.

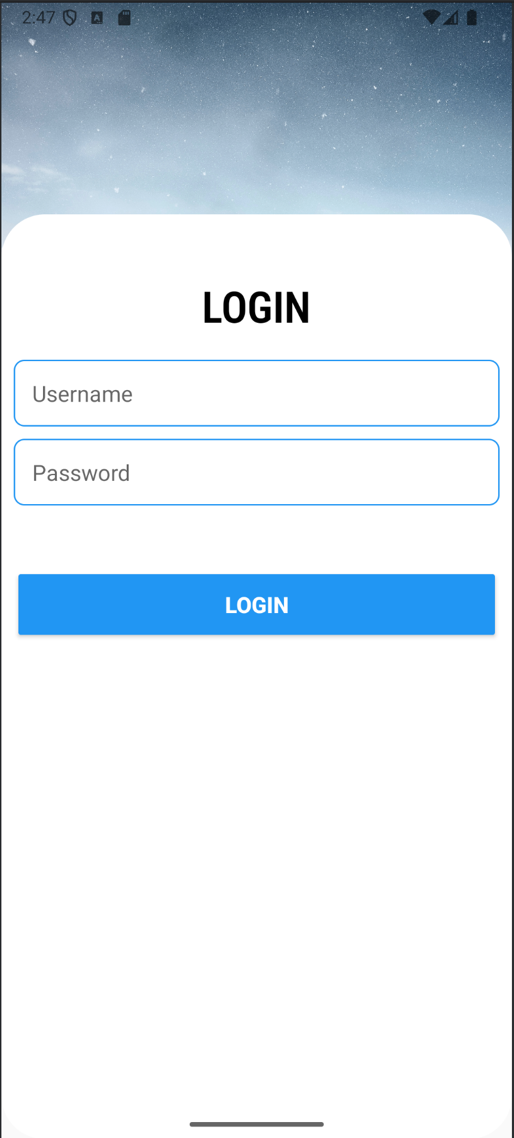
+ Constraints:

* + FK\_HisLoginE: Foreign key linking to the ID field in the Employee table, ensuring each login history record is linked to a valid employee.

+ Rationale: The HistoryEmployee table is structured to log each employee’s login events, with IDLogin as the unique identifier for each entry. The foreign key FK\_HisLoginE guarantees data integrity by associating each record with an existing employee account, thereby ensuring accurate historical tracking.

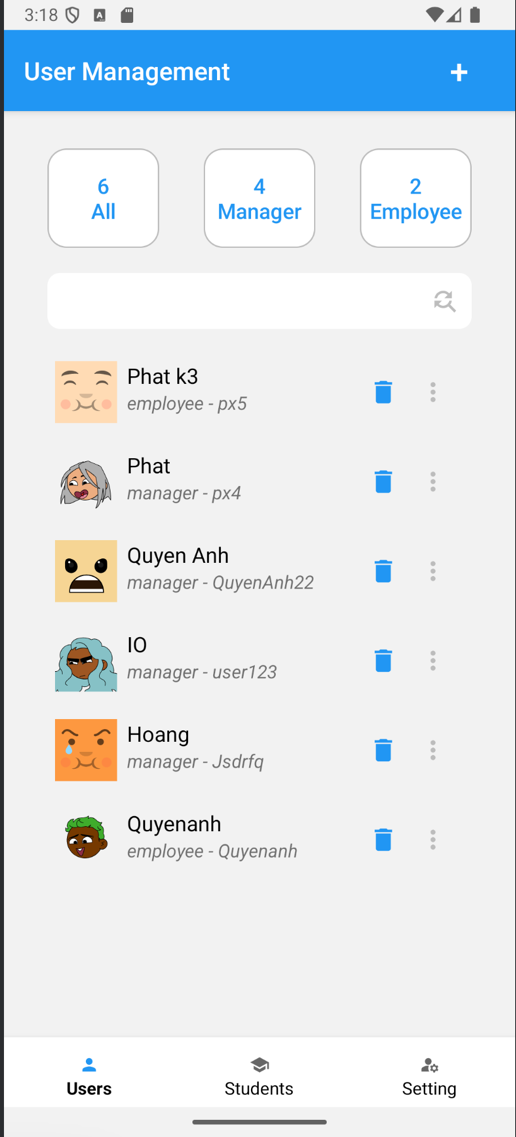
3.6 Implementation

3.6.1 The Login Form



The “frmLogin” serves as the initial interface upon launching the system. This login form prompts users to input their username and password. Upon entering the correct credentials, users are assigned a specific role corresponding to their username and password. The available roles include Admin, Manager, and Staff (Employee).

3.6.2 The User Management Form



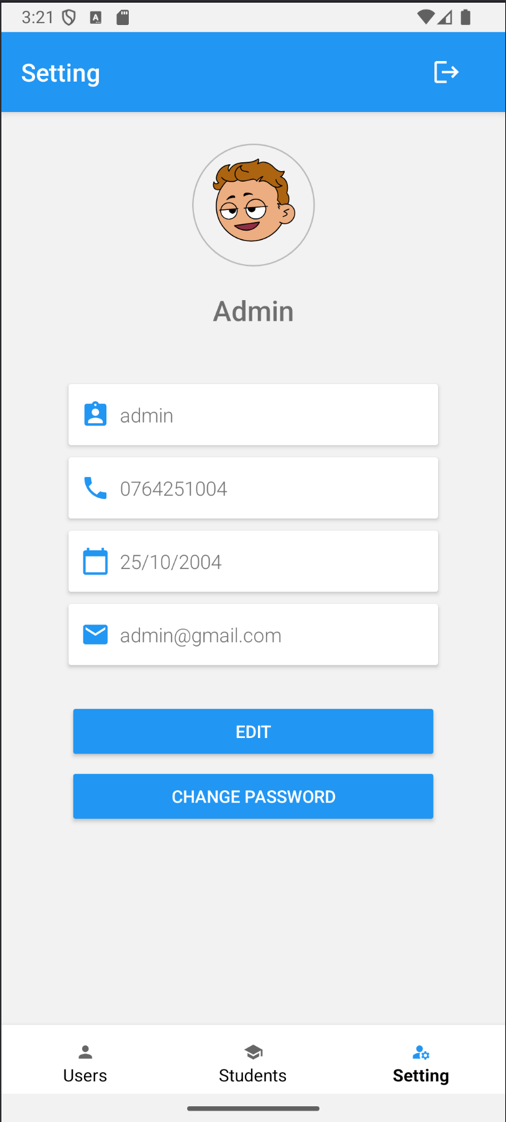
The “User Management” form serves as the main interface for managing user accounts in the system. The form has buttons at the top to display statistics, allowing users to view the total number of accounts or filter by role (Manager or Staff). The form provides a searchable list of users, complete with avatars, roles, and IDs for easy identification. Actionable icons allow for quick management tasks, including editing user details,

viewing their history, or deleting accounts. The “+” button facilitates adding new users with two functions: quick add and detailed add.

3.6.3 The Student Management Form

**###.**

3.6.4 The Setting Form



The "Setting" form provides users with the ability to view and manage their personal account information. At the top, the user's avatar and name are prominently displayed for easy identification. Below, key account details, such as username, phone number, date of birth, and email address, are shown in editable text fields. The form includes two primary actions: an "Edit" button that allows users to update their personal information and a "Change Password" button for enhancing account security. A logout icon in the top-right corner enables users to exit their account.

# CHAPTER 4. CONCLUSION

4.1 Achieved Outcomes

The system has been successfully developed to meet the outlined requirements, providing a structured framework for managing user data effectively. Managers are empowered to manage and update user information within their scope, Admins maintain full control over all aspects of user data, and Employees are limited to viewing and updating their profile pictures only. A role-based authentication mechanism has been implemented to establish secure, defined access levels, which enforces a well-structured access hierarchy within the system.

4.2 Limitations

The primary limitation of this system is its streamlined design, which lacks additional features that could improve its overall functionality. Notably, it does not include an advanced search function to allow retrieval of student information based on multiple or complex criteria. The absence of a visual data section, such as for presenting information through charts or graphs, limits the system’s capacity to display and analyze student data in an easily interpretable format. These gaps restrict the system’s potential to offer a more in-depth and versatile user experience in managing and visualizing data.

4.3 Suggestions for Future Enhancements

To address these limitations, future improvements could focus on expanding the system’s capabilities with more sophisticated features. Integrating an advanced search tool capable of handling complex search parameters would streamline student data retrieval. Additionally, a section dedicated to visualizing data through charts or graphs could improve how the system presents and interprets student information. Such enhancements would strengthen the system’s overall functionality, offering users a more comprehensive and engaging experience in student data management.

# REFERENCES

D. Banerjee, *Cloud Firestore: A Comprehensive Guide*, 1st ed. Cambridge, MA: O'Reilly Media, 2020.

A. K. Sinha, "Understanding Firebase Authentication and Cloud Firestore," *International Journal of Computer Science and Mobile Computing*, vol. 8, no. 3, pp. 45-54, Mar. 2019.

Google LLC, "Cloud Firestore Documentation," *Google Firebase Documentation*, 2023. [Online]. Available: <https://firebase.google.com/docs/firestore>

T. H. Davenport, *Big Data at Work: Dispelling the Myths, Uncovering the Opportunities*, 2nd ed. Boston, MA: Harvard Business Review Press, 2017.

M. Glendenning and B. Gilbert, "Firebase and Firestore: Real-Time NoSQL Databases," *International Conference on Web and Cloud Computing*, pp. 233-245, 2020.

M. Stonebraker and U. Cetintemel, "One Size Fits All: An Idea Whose Time Has Come and Gone," in *Proceedings of the 21st International Conference on Data Engineering*, 2005, pp. 2-11.

A. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Reading, MA: Addison-Wesley, 2019.

Firebase Inc., "Firebase Realtime Database Guide," *Firebase Documentation*, 2022. [Online]. Available: <https://firebase.google.com/docs/database>

S. F. Ceri, F. Garett, and D. Stalder, "NoSQL Databases for Big Data Applications," *Journal of Cloud Computing*, vol. 5, no. 2, pp. 95-102, 2021.

R. F. Woolf, *Mastering Firebase for Real-Time Applications*, New York, NY: Apress, 2021.