**Project Documentation**

**Education Visualizer**

**Team Members**

Mounica Seelam – S559228

Sri Harshika Sattor – S558566

Katta Chandi Priya – S558615

Bodapati Supriya - S562040

Kalla Pavan Kumar Reddy – S560196

Kandimalla Manoj Kumar – S561637

**Table of Contents**

1. Introduction
   1. Purpose
   2. Scope
   3. Definitions, Acronyms, and Abbreviations
   4. Overview
2. Installation process
   1. Software Requirements
   2. Hardware Requirements
   3. Software Installations
   4. Running the application

1. Code utilization
   1. Login
   2. LogOut
   3. Super admin dashboard
   4. Visualization
   5. Admin History
   6. Admin History Details
   7. Visualization Details
   8. Admin Dashboard
   9. Admin Account
   10. Faculty Account
   11. Faculty Dashboard
2. System Architecture
3. Environmental Setup
   1. Authentication
   2. Database
4. FlowChart
5. Conclusion
6. Git link

**1. Introduction**

**1.1. Purpose**

The goal of this project is to create an iOS application specifically for educational institutions, with easy access for admins, super admins, and faculty members. The application also attempts to make it easier to visualize students’ achievements using graphical representations. The implementation of role-based authentication is critical to its operation, as it ensures safe access management for different kinds of users in the educational environment.

**1.2. Scope**

The program will support three primary user roles: admins, super admins, and lecturers. Each job will have unique rights and capabilities based on their roles within the educational environment. Charts, graphs, and progress indicators will be utilized to present achievement data in an easy-to-understand style. Role-based access control measures will be used to prevent unauthorized access to data inside the application.

**1.3. Definitions, Acronyms, and Abbreviations**

* Admins: Users with administrative privileges who have access to the courses in their respective departments.
* Super Admins: Users with elevated administrative privileges who have access to all the courses and all the departments in the university.
* Faculty: Academic staff members who have access to the courses which they teach.
* iOS: Operating system developed by Apple Inc.

**1.4. Overview**

The overview of this project entails the development of an iOS application aimed at facilitating seamless access for admins, super admins, and lecturers within educational settings. Additionally, the application will feature graphical representations of students’ achievements, enhancing data visualization and analysis. Key to its functionality is the implementation of robust role-based authentication mechanisms to ensure secure user access and management.

**2. Installation Process**

**2.1. Software Requirements**

* Tools: Xcode 15 or higher with in-built simulator
* OS: IOS 17 or higher versions

**2.2. Hardware Requirements**

* IOS Devices: Macbook, Iphone, Ipad etc

.

**2.3. Software Installations**

**Xcode:**

* **Check System Requirements:**
* Before installing Xcode, ensure that your Mac meets the system requirements.
* **Open the App Store:**
  + Launch the App Store application on your Mac.
  + In the App Store, use the search bar at the top-right corner and search for "Xcode".
  + On the Xcode page, you'll see an "Install" button. Click on it to begin the installation process.
  + You may be prompted to enter your Apple ID credentials. Provide your credentials to proceed.
* **Launch Xcode:**
* Once the download and installation are complete, Xcode will appear in your Applications folder.
* You can launch Xcode by clicking on its icon in the Applications folder.
* **Verify Installation:**
* After launching Xcode, you can verify that it's installed correctly by checking the version number in the "About Xcode" section under the "Xcode" menu.
* Additionally, you can create a new project or open an existing project to ensure that Xcode is functioning properly.

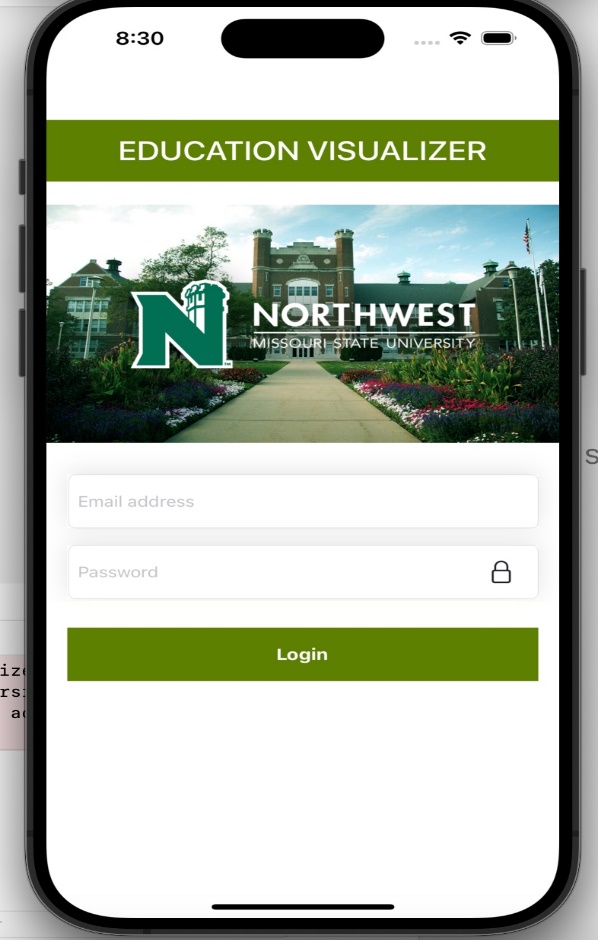
**2.4. Running the application**

* Open Xcode:
  + Launch Xcode on your Mac.
* Open the Project:
  + Open the project of the app to run.
* Select the Target Device:
  + Choose the target device you want to run the app on. You can choose between a physical iOS device connected to your Mac or a simulator.
* Build the Project:
  + Click on the "Build" button (a play button symbol) in the Xcode toolbar to build your project. This step compiles your code and prepares it for running.
* Run the App:
  + Click on the "Run" button (a play button symbol) in the Xcode toolbar. If you're running the app on a physical device, make sure it's connected to your Mac via USB.
* Monitor the Build and Run Progress:
  + Xcode will compile your code, launch the simulator (if you're using one), and deploy the app on the selected device. You'll see the progress in the Xcode interface.
* Interact with the App:
  + Once the app is running, you can interact with it just like any other app on your device or simulator.
* Stop the App:
  + When you're done testing the app, you can stop it by clicking on the "Stop" button (a square symbol) in the Xcode toolbar.

**3. Code Utilization**

**3.1. Login**

* **clickOnLogin Action**: This method is called when the user taps on the login button. It performs the following tasks:
  + Validates the email format and displays an alert if it's invalid.
  + Validates if the password field is empty and displays an alert if it is.
  + If the entered email matches a predefined super admin email, it checks if the password matches. If it does, it saves user data and navigates to the next screen.
  + Otherwise, it calls a function to handle login using Firebase authentication.
* **setLockImage Function**: This function toggles the visibility of the password in the password text field and updates the lock button's image accordingly.
* **clickOnLockUnlock Action**: This method is called when the user taps on the lock/unlock button. It toggles the visibility of the password using the setLockImage function.
* This code represents the login functionality of an iOS application, including email and password validation, toggling password visibility, and handling login actions using Firebase authentication.



**3.2. LogOut**

* **clickOnLogout Action**: This method is called when the user taps on the logout button. It performs the following tasks:
  + Clears user defaults using UserDefaultsManager to log out the user.
  + Calls the checkLogin method of SceneDelegate to navigate back to the login screen.

A screenshot of a phone

Description automatically generated

**3.3. Super admin dashboard:**

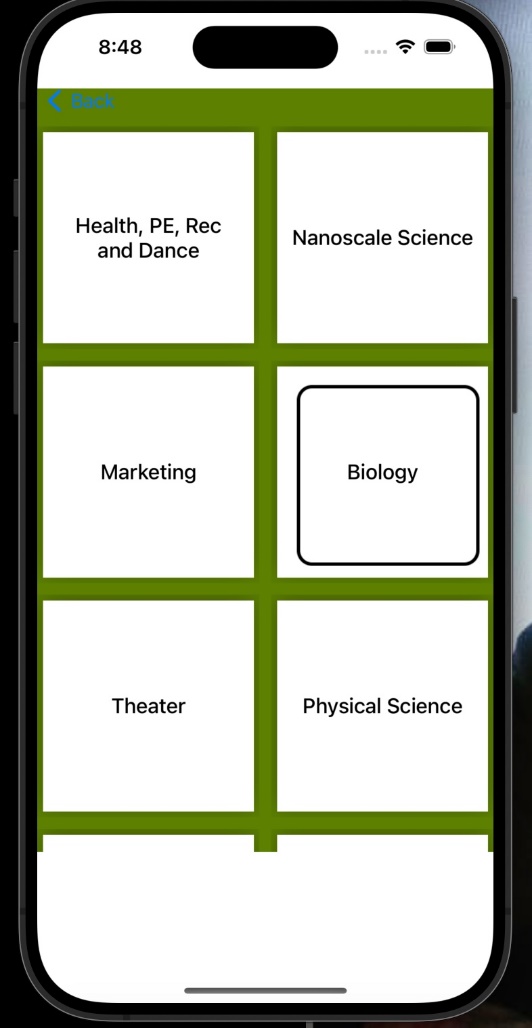
* **clickOnUploadCSV Action**: This method is called when the user taps on the upload CSV button. It initiates a document picker view controller (UIDocumentPickerViewController) configured to allow the selection of comma-separated text files (UTType.commaSeparatedText). Once the user selects a file, the documentPicker delegate method is invoked.
* **documentPicker delegate Methods**:
  + **documentPicker(\_:didPickDocumentsAt:)**: This method is called when the user picks a document. It retrieves the URL of the selected file, reads its contents as a CSV string, and processes each row of data using a CSV reader (CSVReader). It then decodes each row into a custom data structure (CSVData) using a CSV row decoder (CSVRowDecoder) and saves the records to CoreData. Finally, it displays a success message in an alert.
  + **documentPickerWasCancelled**: This method is called when the user cancels the document picker. It simply prints a message indicating that the document picker was canceled.
* **Core Data Operations**: Within the **documentPicker(\_:didPickDocumentsAt:)** method, the CSV data is processed and saved into CoreData. Each record represents student data, including account name, course name, assessment title, learning outcome name, and learning outcome rating. Additionally, a timestamp string is generated and used as the file name for identification and association with the uploaded data.
* This code enables super admin users to upload CSV files containing student data, processes the data, saves it into CoreData, and provides feedback to the user through alerts. It leverages iOS frameworks such as UIKit, UniformTypeIdentifiers, and CoreData for file handling and data management operations.

A cell phone with a screen

Description automatically generated

**3.4. Visualization**

* **VisualizationRecordVC Class:**
* **Properties**: It declares properties to store data related to departments, courses, learning outcomes, assignments, and currently selected data. It also declares properties to store the selected department, course, and learning outcome.
* **ViewDidLoad**: This method is called after the view controller's view is loaded into memory. It sets up the collection view's delegate, data source, and layout. It fetches files from CoreData and populates department data based on the user's role (admin, faculty, or super admin).
* **CollectionView Delegate and Data Source Methods**: These methods implement the necessary functionality for the collection view, such as determining the number of items, configuring cells, and handling item selection. They also handle the logic for transitioning between different levels of data (department, course, learning outcome, and assignment) based on user selection.
* **showAssignmentList Function**: This function fetches assignments for the selected course from CoreData and updates the collection view's data accordingly.

 A cell phone screen with a cell phone screen

Description automatically generated

A screen shot of a cell phone

Description automatically generated A screenshot of a cell phone

Description automatically generated

* **CollectionViewFlowLayout Class:**
* This class represents a custom UICollectionViewFlowLayout subclass.
* It overrides the prepare method to configure the layout of the collection view.
* The configureLayout method calculates the item size based on the available width of the collection view and sets it accordingly.
* This code implements a dynamic collection view interface for visualizing records within an iOS application, allowing users to navigate through different levels of data (departments, courses, learning outcomes, and assignments) and view detailed information for each record. It leverages CoreData for data storage and retrieval and utilizes custom cell and layout classes for UI customization.

**3.5. Admin History**

* **UITableViewDelegate and UITableViewDataSource Methods**:
  + **numberOfSections(in:)**: Returns the number of sections in the table view. In this case, it always returns 1.
  + **tableView(\_:numberOfRowsInSection:)**: Returns the number of rows in the table view, which is equal to the count of filesRecord.
  + **tableView(\_:cellForRowAt:)**: Configures and returns a cell for the specified index path. It dequeues a reusable cell using the identifier "CellIdentifier" and populates it with information about the file at the corresponding index in filesRecord.
  + **tableView(\_:didSelectRowAt:)**: Handles row selection events. It instantiates and presents the AdminHistoryDetailsVC view controller for displaying details of the selected historical record.
  + **tableView(\_:commit:forRowAt:)**: Handles row deletion events. If the user swipes left on a row and selects the delete option, it deletes all records associated with the file and removes the file from CoreData and the filesRecord array. The table view is then updated to reflect the deletion.
* This code implements a view controller that manages a table view displaying historical records uploaded by admins within an iOS application. It interacts with CoreData for data storage and retrieval and provides functionalities for viewing and deleting historical records.

**3.6. Admin History Details**

* **ViewDidLoad**:
  + It adds a right bar button item (**btnFilter**) to the navigation bar, which triggers the **filterRecord** method when tapped.
  + It fetches student data records associated with the specified file name from CoreData. Upon completion, it updates studentDataRecord and studentAssignment arrays accordingly and reloads the table view.
* **UITableView Delegate and Data Source Methods**:
  + These methods implement the necessary functionality for the table view, such as determining the number of rows and sections, configuring cells, and handling cell selection.
  + In **cellForRowAt**, it configures each cell to display detailed information about each student record, including account name, course name, assessment title, learning outcome name, learning outcome rating, and created timestamp.
* **filterRecord Method**:
  + This method is called when the filter button in the navigation bar is tapped.
  + It presents a **PickerView** (presumably a custom picker view) with the list of unique student assignments (**studentAssignment**).
  + When an assignment is selected, it filters the **studentDataRecord** array to display only records associated with the selected assignment and reloads the table view accordingly.
* **attributedText Method**:
  + This private method formats the display text for each cell in the table view.
  + It adds a custom attribute (green text color) to the "Learning Outcome Rating" part of the text to emphasize it.
* This code implements a view controller that displays detailed records associated with a specific file uploaded by an admin within an iOS application. It provides functionalities for filtering records based on student assignments and presents detailed information in a table view.

Top of Form

**3.7. Visualization Details**

* **Properties**:
  + It declares properties to store information about the department, course, learning outcome, and assignment.
  + It declares an instance of CoreDataHelper for fetching learning outcome ratings.
* **ViewDidLoad:**
  + It sets the delegate and data source of both the pie chart and bar chart views to the view controller.
  + It adds a right bar button item (btnFilter) to the navigation bar, which triggers the filterRecord method when tapped.
  + It fetches learning outcome ratings associated with the specified assignment and updates the visualizations accordingly.
* **pieChartUpdateRecord Method:**
  + This method updates the pie chart with proficiency level data.
  + It creates an array of PieChartDataEntry objects representing each proficiency level along with its count.
  + It configures a PieChartDataSet with the entries and assigns colors to the dataset.
  + It creates a PieChartData object with the dataset and assigns it to the pie chart view.
  + It sets additional properties such as center text, hole color, and data availability text.
* **barChartUpdateRecord Method:**
  + This method updates the bar chart with proficiency level data.
  + It creates an array of BarChartDataEntry objects representing each proficiency level along with its count.
  + It configures a BarChartDataSet with the entries and assigns colors to the dataset.
  + It creates a BarChartData object with the dataset and assigns it to the bar chart view.
  + It sets additional properties such as data availability text.
* This code implements a view controller that displays detailed visualizations of proficiency levels associated with a specific assignment's learning outcome within an iOS application. It utilizes the DGCharts framework to create pie and bar charts dynamically based on fetched data from CoreData.

A cell phone with a graph on it

Description automatically generated

**3.8. Admin Dashboard**

* **clickOnVisualization**:
  + This method is connected to an action in the user interface, likely a button tap event.
  + When the user taps on a visualization-related button in the admin dashboard, this method is triggered.
  + Inside the method:
    - It instantiates a view controller (VisualizationRecordVC) from the storyboard using its identifier.
    - It sets the modal presentation style of the view controller to full screen.
    - It pushes the instantiated view controller onto the navigation stack of the navigation controller associated with the current view controller. This navigation action will animate the transition to the visualization record view controller (**vc**).
    - By pushing the view controller onto the navigation stack, the user is taken to the visualization record screen, allowing them to view and interact with visualization-related functionalities.
* This code enables navigation from the admin dashboard to a screen where visualizations of records can be viewed and interacted with, enhancing the user experience, and providing access to relevant data representations.

A cell phone with a screen

Description automatically generated

**3.9. Admin Account**

* **clickOnDepartment**:
  + This method is connected to an action triggered by selecting the department field.
  + When the user taps on the department field, a picker view is presented with a list of department names obtained from the courses dictionary.
  + When a department is selected from the picker, its value is set to the department text field.
* **clickOnCreate**:
  + This method is connected to an action triggered by tapping a "Create" button.
  + It first dismisses the keyboard by calling endEditing(true) on the view.
  + It validates the input fields using the validate() method:
    - Checks if the name field is empty or invalid.
    - Checks if the email field is empty or invalid.
    - Checks if the department field is empty.
  + If validation succeeds, it generates a random password and calls the signUp method of FireStoreManager to create a new admin account with the provided information.
* **validate Method**:
  + This method validates the input fields:
    - Checks if the name field is empty or contains invalid characters.
    - Checks if the email field is empty or contains an invalid email format.
    - Checks if the department field is empty.
  + If any validation fails, it displays an alert with an appropriate message and returns false. Otherwise, it returns true.
* This code implements the functionality for creating admin accounts in the application. It ensures that the user provides valid input for the required fields and utilizes a picker view for selecting the department. Once validated, it creates a new admin account using the provided information.

A screenshot of a cell phone

Description automatically generated

**3.10. Faculty Account**

* **clickOnSignup**:
  + This method is connected to an action triggered by tapping a "Signup" button.
  + It dismisses the keyboard by calling endEditing(true) on the view.
  + It validates the input fields using the validate () method:
    - Checks if the name field is empty or contains invalid characters.
    - Checks if the email field is empty or contains an invalid email format.
    - Checks if the course label is empty or displays the default text "Select Courses".
  + If validation succeeds, it generates a random password and calls the **signUp** method of **FireStoreManager** to create a new faculty account with the provided information.
* **validate Method**:
  + This method validates the input fields:
    - Checks if the name field is empty or contains invalid characters.
    - Checks if the email field is empty or contains an invalid email format.
    - Checks if the course label is empty or displays the default text "Select Courses".
  + If any validation fails, it displays an alert with an appropriate message and returns false. Otherwise, it returns true.
* **clickOnCourses**:
  + This method is connected to an action triggered by tapping the "Select Courses" button.
  + It retrieves the courses associated with the faculty's department from the **courses** dictionary.
  + It constructs table data from the retrieved courses.
  + It initializes a **CheckboxDialogViewController** instance with the constructed table data and presents it modally over the current context.
* **CheckboxDialogViewDelegate Extension**:
  + This extension conforms to the **CheckboxDialogViewDelegate** protocol.
  + It implements the **pickerValueChanged** method, which is called when the user selects courses in the checkbox dialog.
  + It updates the **selectedValues** array with the selected course values and updates the course label (**courseLbl**) to display the selected courses.
* This code implements the functionality for creating faculty accounts in the iOS application. It ensures that the user provides valid input for the required fields and allows them to select courses from a list of options. Once validated, it creates a new faculty account using the provided information.

A screenshot of a cell phone

Description automatically generated

Top of Form

**3.11. Faculty Dashboard**

* **clickOnVisualization:**
  + This method is connected to an action triggered by tapping on a visualization button or similar UI element.
  + When the button is tapped, it creates an instance of VisualizationRecordVC using the instantiateViewController(withIdentifier:) method, which retrieves the view controller with the specified storyboard identifier "VisualizationRecordVC".
  + It sets the modal presentation style of the view controller to .fullScreen.
  + It pushes the VisualizationRecordVC onto the navigation stack using pushViewController(\_:animated:) method of the navigation controller.
  + As a result, the application transitions to the visualization screen where students can view records and charts related to their achievements or other data.
* This code enables students to navigate from the student dashboard to the visualization screen within the iOS application by tapping on the visualization button.

A cell phone with a picture of a building

Description automatically generated

**4. System Architecture**

A diagram of a cell phone

Description automatically generated

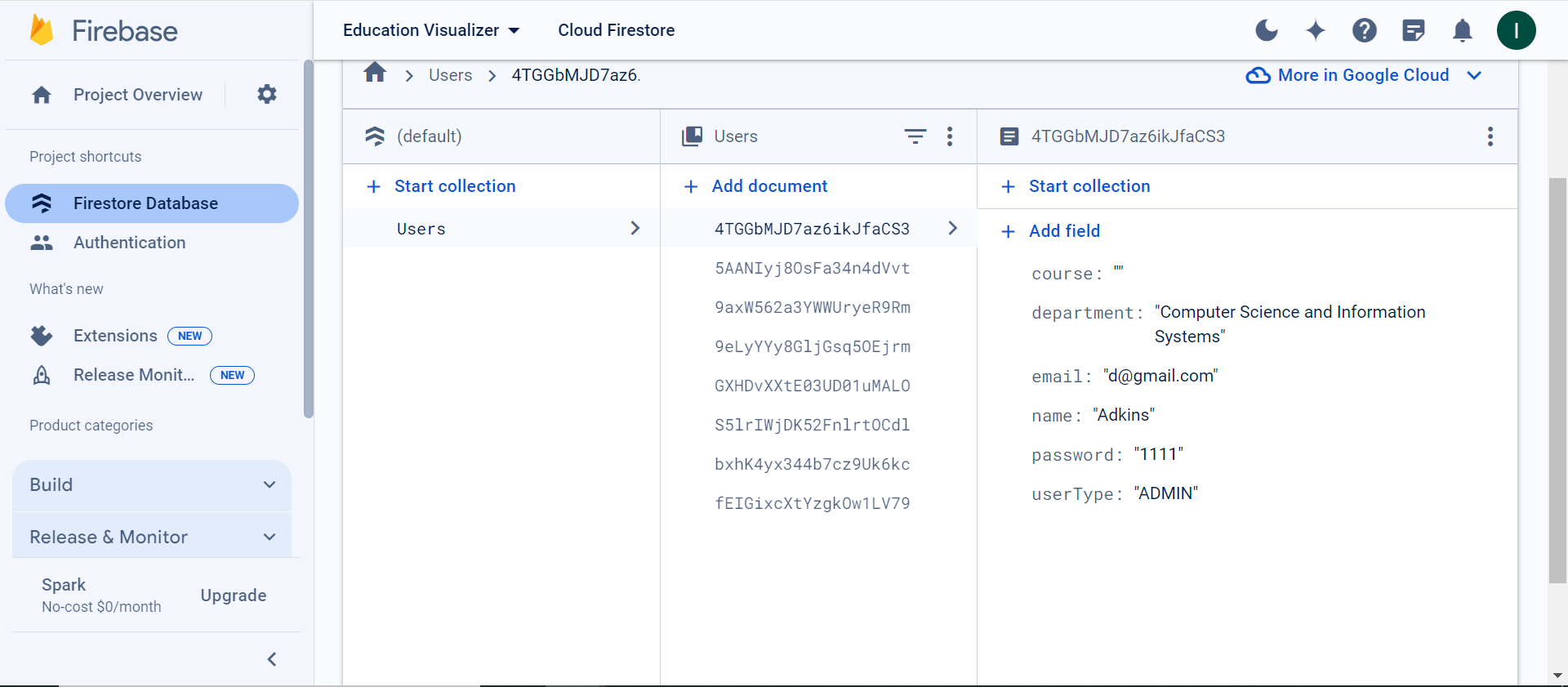
**5. Environmental Setup**

**5.1. Authentication**

* Go to the Firebase Console (<https://console.firebase.google.com/>).
* Create a new project or select an existing one.
* Navigate to "Authentication" from the left-hand menu.
* Enable the authentication methods you want to use i.e., email/password.
* Follow the provided instructions to integrate Firebase Authentication SDK into your iOS project.

**5.2. Database**

* **Setting up firebase:** To set up the database, we use “Google Firebase” software. It can be accessed using the link: <https://firebase.google.com/>
* To create a project in firebase, we need to follow the below steps:
* Create a project with your app name.
* Click on the iOS+ logo to add an ios app to the firebase project.
* Register the app using the bundle id of the app and click on the Register app button.
* Then download the info list and drag drop the downloaded file into the project in xcode.
* Copy the GitHub link given below ( <https://github.com/firebase/firebase-ios-sdk>) and paste it in the “add packages” option in Xcode.
* Copy the code given in firebase and paste it in the AppDelegate file of the app in Xcode.



**6. FlowChart**

**A diagram of a company

Description automatically generated**

**7. Conclusion**

The development of the iOS application for educational institutions marks a significant step towards enhancing administrative efficiency and fostering a data-driven approach to student achievement analysis. Throughout this project, our primary focus has been on creating a user-friendly platform that provides easy access for admins, super admins, and faculty members, while also prioritizing the security and integrity of sensitive educational data.

By implementing role-based authentication, we have ensured that each user role has appropriate access rights tailored to their responsibilities within the educational environment. The incorporation of graphical representations for visualizing student achievements adds another layer of depth to the application. Whether it's tracking academic progress, identifying areas for improvement, or recognizing outstanding performance, the visualizations provided by the application offer valuable insights into student performance trends.

**8. Git link:** <https://github.com/S559228-Mounica/GDPProject.git>