**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **CONTENT** | **PAGE NO** |
| 1 | INTRODUCTION  1.1 The Employee Management System  1.2 Project Objectives  1.3 Scope of the Project  1.4 Front End – Flutter  1.5 Backend – Microsoft Sql Server  1.6 Dart  1.7 Android Studio  1.8 System Specification  1.8.1 Hardware specification  1.8.2 Software specification | 3  3  4  5  6  6  7  9  10  10  10 |
| 2 | MODULES  2.1 WIDGETS  2.1.1 Appbar  2.1.2 Column | 11  12  12  13 |
|  | 2.1.3 Container  2.1.4 Scaffold  2.1.5 Text | 13  13  14 |
| 3 | ER DIAGRAM  3.1 Entities | 15  15 |
| 4 | SCHEMA DIAGRAM | 18 |
| 5 | NORMALIZATION | 21 |
| 6 | APPENDIX  6.1 Source Code  6.1.1 Sign in tab  6.1.2 Create tab  6.1.3 View tab  6.1.4 Edit tab  6.1.5 Delete tab | 23  23  34  34  36  37  38 |
| 7 | SCREENSHOTS | 39 |
| 8  9  10 | CONCLUSION    FUTURE WORK  REFERENCES | 42  42  44 |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **Figure No** | **Figure Caption** | **Page Number** |
| 1 | LOGIN OR SIGN IN | 39 |
| 2 | CREATE | 39 |
| 3 | VIEW | 40 |
| 4 | DETAILS | 40 |
| 5 | EDIT OR UPDATE | 40 |
| 6 | DELETE | 41 |

**CHAPTER 1**

# INTRODUCTION

# The Employee Management System (EMS)

The Employee Management System (EMS) is a cutting-edge application developed using the Flutter framework, designed to simplify and streamline the management of employee data within an organization. With its user-friendly interface and comprehensive features, the EMS project aims to revolutionize employee data management, enabling organizations to efficiently handle their workforce.

Managing employee information is a crucial task for any organization, and traditional manual methods often prove to be time-consuming and error-prone. The EMS project addresses these challenges by providing a digital platform that empowers users to perform CRUD operations on employee records effortlessly.

The project's primary objective is to enhance the efficiency of employee data management, allowing organizations to focus on their core activities rather than grappling with cumbersome paperwork and manual processes. It includes user authentication mechanisms to ensure data security and restrict access to authorized personnel. Additionally, the system enables users to create new employee records by providing essential details such as names, contact information, positions, and departments. Effortless retrieval of employee records is another crucial feature of the EMS project. Users can search and filter records based on specific criteria, such as employee names, departments, positions, or unique employee IDs. This functionality empowers organizations to access and analyze employee data swiftly and accurately.

Updating employee information is made hassle-free through the EMS application. Users can modify details such as contact information, job positions, department assignments, and other relevant data, ensuring that the employee records remain up to date and accurate.

**Key Features:**

1. User Authentication: The system includes secure user authentication mechanisms to ensure that only authorized individuals can access and manage employee data.
2. Employee Record Creation: Users can create new employee records by providing relevant details such as name, contact information, position, department, and other pertinent information.
3. Employee Record Retrieval: Users can search and retrieve employee records based on various criteria such as name, department, position, or employee ID. This functionality allows for efficient data retrieval and analysis.
4. Employee Record Update: Users can update employee information, including contact details, job position, department assignment, and any other relevant data. This ensures that employee records remain accurate and up to date.
5. Employee Record Deletion: Users have the ability to remove obsolete or inactive employee records from the system. This functionality helps maintain a clean and organized database.
6. Data Visualization: The EMS project offers data visualization features to present employee data in a meaningful and informative manner. Users can view charts, graphs, or other visual representations to gain insights into employee statistics, such as department distribution, age demographics, or salary ranges.

# PROJECT OBJECTIVES

The primary objectives of the project were as follows:

* Develop a Flutter application for managing employee data.
* Implement CRUD operations (Create, Read, Update, Delete) for employee records.
* Integrate a suitable database solution for storing employee data.
* Design a user-friendly interface for easy navigation and data input.
* Ensure proper validation and error handling during CRUD operations.

# SCOPE OF THE PROJECT:

The scope of an Employee Management System project using Flutter for CRUD (Create, Read, Update, Delete) operations can vary depending on the specific requirements and goals of the project. However, here are some common functionalities and features that you can consider for such a system:

1. User Authentication: Implement a login/signup mechanism to ensure that only authorized users can access the system.
2. Employee Records Management: Provide features to add, view, update, and delete employee records. Users should be able to enter details such as name, employee ID, department, position, contact information, etc.
3. Search and Filtering: Include functionality to search and filter employee records based on different criteria, such as name, department, position, or any other relevant attributes.
4. Employee Profile: Allow users to view detailed profiles of individual employees, including their personal information, job history, performance reviews, and any other relevant details.
5. Access Control and Permissions: Define user roles and permissions to restrict access to certain functionalities or data based on the user's position or responsibilities within the organization.
6. Responsive Design: Ensure the system is responsive and compatible with different screen sizes and devices to allow users to access it from desktops, tablets, or mobile devices.
7. Data Security: Implement necessary security measures to protect sensitive employee data, including encryption, user access controls, and secure communication protocols.
8. Scalability and Performance: Design the system to handle a growing number of employees and data efficiently, considering factors like database optimization and code optimization.

# FRONT END - FLUTTER

Flutter is Google’s portable UI toolkit for crafting beautiful, natively compiled applications for mobile, web, and desktop from a single codebase. Flutter works with existing code, is used by developers and organizations around the world, and is free and open source.

For users, Flutter makes beautiful apps come to life.

For developers, Flutter lowers the bar to entry for building apps. It speeds app development and reduces the cost and complexity of app production across platforms.

For designers, Flutter provides a canvas for high-end user experiences. Fast Company described Flutter as [one of the top design ideas of the decade](https://www.fastcompany.com/90442092/the-14-most-important-design-ideas-of-the-decade-according-to-the-experts) for its ability to turn concepts into production code without the compromises imposed by typical frameworks. It also acts as a productive prototyping tool, with [CodePen](https://codepen.io/topic/flutter) support for sharing your ideas with others.

For engineering managers and businesses, Flutter allows the unification of app developers into a single mobile, web, and desktop app team, building branded apps for multiple platforms out of a single codebase. Flutter speeds feature development and synchronizes release schedules across the entire customer base.

**1.5 BACKEND – MICROSOFT SQL SERVER**

Microsoft SQL Server is a relational database management system (RDBMS) that supports a wide variety of transaction processing, business intelligence and analytics applications in corporate IT environments. Microsoft SQL Server is one of the three market-leading database technologies, along with Oracle Database and IBM's [DB2](https://www.techtarget.com/searchdatacenter/definition/DB2).

Like other [RDBMS](https://www.techtarget.com/searchdatamanagement/definition/RDBMS-relational-database-management-system) software, Microsoft SQL Server is built on top of [SQL](https://searchsqlserver.techtarget.com/definition/SQL), a standardized programming language that database administrators ([DBAs](https://searchsqlserver.techtarget.com/definition/database-administrator)) and other IT professionals use to manage databases and query the data they contain. SQL Server is tied to Transact-SQL ([T-SQL](https://searchsqlserver.techtarget.com/definition/T-SQL)), an implementation of SQL from Microsoft that adds a set of proprietary programming extensions to the standard language

**SQL Server Management Studio**

[SQL Server Management Studio](https://en.wikipedia.org/wiki/SQL_Server_Management_Studio) is a [GUI](https://en.wikipedia.org/wiki/GUI) tool included with SQL Server 2005 and later for configuring, managing, and administering all components within Microsoft SQL Server. The tool includes both script editors and graphical tools that work with objects and features of the server. SQL Server Management Studio replaces [Enterprise Manager](https://en.wikipedia.org/w/index.php?title=Enterprise_Manager&action=edit&redlink=1) as the primary management interface for Microsoft SQL Server since SQL Server 2005. A version of SQL Server Management Studio is also available for SQL Server Express Edition, for which it is known as SQL Server Management Studio Express (SSMSE).

A central feature of SQL Server Management Studio is the Object Explorer, which allows the user to browse, select, and act upon any of the objects within the server. It can be used to visually observe and analyze query plans and optimize the database performance, among others. SQL Server Management Studio can also be used to create a new database, alter any existing database schema by adding or modifying tables and indexes, or analyze performance. It includes the query windows which provide a GUI based interface to write and execute queries.

# 1.6 DART

Dart is a programming language developed by Google. It is primarily used for building mobile, web, and desktop applications. Dart is designed to be efficient, fast, and scalable, with a strong focus on developer productivity.

Here are some key features and aspects of the Dart programming language:

Object-Oriented: Dart is an object-oriented language, which means it supports concepts such as classes, objects, inheritance, and encapsulation.

Strong Typing: Dart is a statically typed language, meaning that variables have specific types that are checked at compile-time. However, Dart also has type inference, which allows you to omit explicit type declarations in certain cases.

Garbage Collection: Dart uses automatic garbage collection to manage memory, relieving developers from manual memory management tasks.

Asynchronous Programming: Dart provides built-in support for asynchronous programming using features like asynchronous functions, async and await keywords, and futures. This makes it easier to write efficient and responsive applications, especially for tasks such as network requests or file operations.

Just-In-Time (JIT) and Ahead-of-Time (AOT) Compilation: Dart offers both JIT and AOT compilation. During development, Dart applications can be executed in JIT mode, which allows for fast development cycles and hot-reloading. For deployment, Dart code can be compiled to native machine code using AOT compilation for improved performance and reduced startup time.

Flutter Framework: Dart is widely known for its association with the Flutter framework. Flutter, a UI toolkit for building cross-platform applications, uses Dart as its primary language. With Flutter and Dart, developers can create high-performance, visually appealing applications for mobile, web, and desktop platforms using a single codebase.

Tooling and Ecosystem: Dart has a rich set of development tools and libraries. The Dart SDK provides a command-line interface for compiling and running Dart code. Additionally, Dart has a robust package manager called Pub, which hosts a wide range of packages and libraries contributed by the Dart community.

Dart offers a combination of productivity, performance, and versatility, making it a popular choice for developers building applications across different platforms. Whether you're working on mobile apps with Flutter or developing server-side applications, Dart provides a cohesive and consistent language experience.

# 1.7 ANDROID STUDIO

Android Studio is the official integrated development environment (IDE) for developing Android applications. It is built on top of the IntelliJ IDEA platform and provides a comprehensive set of tools and features specifically tailored for Android development.

Here are some key aspects and features of Android Studio:

1. Project Structure: Android Studio provides a project structure that organizes your source code, resources, and assets in a convenient and manageable manner. You can create modules, libraries, and dependencies within your project structure.
2. Code Editor: Android Studio offers a powerful code editor with features such as syntax highlighting, code completion, code navigation, and refactoring tools. It supports multiple programming languages, including Java and Kotlin (the recommended language for Android development).
3. Layout Editor: Android Studio includes a visual Layout Editor that allows you to design and preview the user interface (UI) of your Android app. It supports drag-and-drop functionality, XML editing, and real-time UI rendering.
4. Gradle Build System: Android Studio utilizes the Gradle build system to automate the build, testing, and packaging processes of your Android app. It provides flexibility in managing dependencies, running tasks, and generating different build variants (e.g., debug, release).
5. Emulator and Device Testing: Android Studio includes an emulator that enables you to run and test your Android applications on virtual devices. It also supports USB debugging for connecting physical Android devices to test your apps directly.
6. Profiling and Debugging: Android Studio offers robust tools for profiling and debugging your app's performance and behavior. You can analyze CPU and memory usage, view logcat messages, and track app performance to optimize your code and identify issues.
7. Android SDK Manager: Android Studio includes the Android SDK Manager, which allows you to download, update, and manage the Android SDK components, platform versions, and additional libraries required for app development.
8. Integrated Support: Android Studio provides integration with various tools and services to streamline development workflows. It supports version control systems like Git, integration with Google Cloud services, Firebase integration for backend services, and more.
9. Play Store Publishing: Android Studio facilitates the process of publishing your app to the Google Play Store. It provides tools for generating app signing keys, managing release versions, and creating optimized APK files for distribution.
10. Android Studio is continuously updated with new features and improvements to support the latest Android platform updates and development practices. It offers a comprehensive development environment that enhances productivity and simplifies the process of building high-quality Android applications.

# 1.8 SYSTEM SPECIFICATION

# 1.8.1 Hardware specification

* Processor : Intel dual core
* Processor speed: 1.04GHZ
* Ram : 1GB
* Monitor
* Keyboard
* Mouse

**1.8.2 Software specification**

* OS
* Language : Dart
* IDE : Android Studio

# CHAPTER 2

# MODULES

1. http or dio: These modules allow you to make HTTP requests to communicate with the backend server and perform CRUD operations on employee data.
2. provider: The provider package is useful for managing state in Flutter applications. It helps with data sharing and updating across different screens or widgets.
3. intl: The intl package provides localization and internationalization support, which can be helpful if your Employee Management System needs to support multiple languages or regions.
4. flutter\_form\_builder: This module simplifies form creation and validation in Flutter applications. It provides pre-built form fields and validators for capturing and validating employee data.
5. fluttertoast or snackbar: These modules provide user feedback in the form of toasts or snack bars to display success messages, error messages, or notifications during CRUD operations.
6. flutter\_slidable: This module allows you to create swipeable lists, which can be useful for implementing actions like deleting or updating employee records with a swipe gesture.
7. flutter\_spinkit: This module provides a collection of loading spinners or indicators that can be used to indicate background processes during CRUD operations, such as loading or saving employee data.
8. path\_provider: The path\_provider package helps with accessing the device's local storage to store or retrieve data related to the Employee Management System, such as user preferences or cached data.

**2.1 WIDGETS**

The basic component in a Flutter program is a "widget", which can in turn consist of other widgets. A widget describes the logic, interaction, and design of a UI element with an implementation similar to [React](https://en.wikipedia.org/wiki/React_(JavaScript_library)). Unlike other cross-platform toolkits such as [React Native](https://en.wikipedia.org/wiki/React_Native) and [Xamarin](https://en.wikipedia.org/wiki/Xamarin) which draw widgets using native platform components, Flutter renders widgets itself on a per-pixel basis.

There are two fundamental types of widgets: stateless and stateful. Stateless widgets only update if their inputs change, making them very efficient, while stateful widgets can call the setState() method to update an internal state and redraw.

Although widgets are the primary method of constructing Flutter applications, they can also be bypassed in favor of directly drawing on a canvas. This feature has been occasionally used to implement game engines in Flutter.

**2.1.1 APPBAR**

A Material Design app bar.

An app bar consists of a toolbar and potentially other widgets, such as a [TabBar](https://api.flutter.dev/flutter/material/TabBar-class.html) and a [FlexibleSpaceBar](https://api.flutter.dev/flutter/material/FlexibleSpaceBar-class.html). App bars typically expose one or more common [actions](https://api.flutter.dev/flutter/material/AppBar/actions.html) with [IconButton](https://api.flutter.dev/flutter/material/IconButton-class.html)s which are optionally followed by a [PopupMenuButton](https://api.flutter.dev/flutter/material/PopupMenuButton-class.html) for less common operations (sometimes called the "overflow menu").

App bars are typically used in the [Scaffold.appBar](https://api.flutter.dev/flutter/material/Scaffold/appBar.html) property, which places the app bar as a fixed-height widget at the top of the screen. For a scrollable app bar, see [SliverAppBar](https://api.flutter.dev/flutter/material/SliverAppBar-class.html), which embeds an [AppBar](https://api.flutter.dev/flutter/material/AppBar-class.html) in a sliver for use in a [CustomScrollView](https://api.flutter.dev/flutter/widgets/CustomScrollView-class.html).

**2.1.2 COLUMN**

A widget that displays its children in a vertical array.

To cause a child to expand to fill the available vertical space, wrap the child in an [Expanded](https://api.flutter.dev/flutter/widgets/Expanded-class.html) widget.The [Column](https://api.flutter.dev/flutter/widgets/Column-class.html) widget does not scroll (and in general it is considered an error to have more children in a [Column](https://api.flutter.dev/flutter/widgets/Column-class.html) than will fit in the available room). If you have a line of widgets and want them to be able to scroll if there is insufficient room, consider using a [ListView](https://api.flutter.dev/flutter/widgets/ListView-class.html).For a horizontal variant, see [Row](https://api.flutter.dev/flutter/widgets/Row-class.html).

If you only have one child, then consider using [Align](https://api.flutter.dev/flutter/widgets/Align-class.html) or [Center](https://api.flutter.dev/flutter/widgets/Center-class.html) to position the child

**2.1.3 CONTAINER**

A container first surrounds the child with [padding](https://api.flutter.dev/flutter/widgets/Container/padding.html) (inflated by any borders present in the [decoration](https://api.flutter.dev/flutter/widgets/Container/decoration.html)) and then applies additional [constraints](https://api.flutter.dev/flutter/widgets/Container/constraints.html) to the padded extent (incorporating the width and height as constraints, if either is non-null). The container is then surrounded by additional empty space described from the [margin](https://api.flutter.dev/flutter/widgets/Container/margin.html).

During painting, the container first applies the given [transform](https://api.flutter.dev/flutter/widgets/Container/transform.html), then paints the [decoration](https://api.flutter.dev/flutter/widgets/Container/decoration.html) to fill the padded extent, then it paints the child, and finally paints the [foregroundDecoration](https://api.flutter.dev/flutter/widgets/Container/foregroundDecoration.html), also filling the padded extent.

Containers with no children try to be as big as possible unless the incoming constraints are unbounded, in which case they try to be as small as possible. Containers with children size themselves to their children. The width, height, and [constraints](https://api.flutter.dev/flutter/widgets/Container/constraints.html) arguments to the constructor override this.

By default, containers return false for all hit tests. If the [color](https://api.flutter.dev/flutter/widgets/Container/color.html) property is specified, the hit testing is handled by [ColoredBox](https://api.flutter.dev/flutter/widgets/ColoredBox-class.html), which always returns true. If the [decoration](https://api.flutter.dev/flutter/widgets/Container/decoration.html) or [foregroundDecoration](https://api.flutter.dev/flutter/widgets/Container/foregroundDecoration.html) properties are specified, hit testing is handled by [Decoration.hitTest](https://api.flutter.dev/flutter/painting/Decoration/hitTest.html).

**2.1.4 SCAFFOLD**

Implements the basic Material Design visual layout structure.This class provides APIs for showing drawers and bottom sheets.To display a persistent bottom sheet, obtain the [ScaffoldState](https://api.flutter.dev/flutter/material/ScaffoldState-class.html) for the current [BuildContext](https://api.flutter.dev/flutter/widgets/BuildContext-class.html) via [Scaffold.of](https://api.flutter.dev/flutter/material/Scaffold/of.html) and use the [ScaffoldState.showBottomSheet](https://api.flutter.dev/flutter/material/ScaffoldState/showBottomSheet.html) functionThe scaffold will expand to fill the available space. That usually means that it will occupy its entire window or device screen.

When the device's keyboard appears the Scaffold's ancestor [MediaQuery](https://api.flutter.dev/flutter/widgets/MediaQuery-class.html) widget's [MediaQueryData.viewInsets](https://api.flutter.dev/flutter/widgets/MediaQueryData/viewInsets.html) changes and the Scaffold will be rebuilt. By default the scaffold's [body](https://api.flutter.dev/flutter/material/Scaffold/body.html) is resized to make room for the keyboard. To prevent the resize set [resizeToAvoidBottomInset](https://api.flutter.dev/flutter/material/Scaffold/resizeToAvoidBottomInset.html) to false. In either case the focused widget will be scrolled into view if it's within a scrollable container.The [MediaQueryData.padding](https://api.flutter.dev/flutter/widgets/MediaQueryData/padding.html) value defines areas that might not be completely visible, like the display "notch" on the iPhone X. The scaffold's [body](https://api.flutter.dev/flutter/material/Scaffold/body.html) is not inset by this padding value although an [appBar](https://api.flutter.dev/flutter/material/Scaffold/appBar.html) or [bottomNavigationBar](https://api.flutter.dev/flutter/material/Scaffold/bottomNavigationBar.html) will typically cause the body to avoid the padding. The [SafeArea](https://api.flutter.dev/flutter/widgets/SafeArea-class.html) widget can be used within the scaffold's body to avoid areas like display notches..

**2.1.5 TEXT**

A run of text with a single style.

The [Text](https://api.flutter.dev/flutter/widgets/Text-class.html) widget displays a string of text with single style. The string might break across multiple lines or might all be displayed on the same line depending on the layout constraints.

The [style](https://api.flutter.dev/flutter/widgets/Text/style.html) argument is optional. When omitted, the text will use the style from the closest enclosing [DefaultTextStyle](https://api.flutter.dev/flutter/widgets/DefaultTextStyle-class.html). If the given style's [TextStyle.inherit](https://api.flutter.dev/flutter/painting/TextStyle/inherit.html) property is true (the default), the given style will be merged with the closest enclosing [DefaultTextStyle](https://api.flutter.dev/flutter/widgets/DefaultTextStyle-class.html). This merging behavior is useful, for example, to make the text bold while using the default font family and size.

# CHAPTER 3

# ER DIAGRAM

An Entity-Relationship (ER) diagram is a visual representation of the entities and their relationships in a database system. It is a modeling technique used to design and communicate the structure and organization of data within an application or system. The ER diagram consists of entities (objects or concepts), attributes (properties or characteristics), and relationships (associations between entities).

# 

**3.1 ENTITIES**

Entities: Entities represent the objects or concepts in the system. In an Employee Management System, entities can include Employee, Department, Role, Attendance, Leave, etc. Each entity is represented by a rectangle in the ER diagram.

Attributes: Attributes are the properties or characteristics of entities. They describe the data associated with each entity. For example, in the Employee entity, attributes can include EmployeeID, Name, Position, ContactInfo, etc. Attributes are depicted as ovals connected to their respective entities in the ER diagram.

Relationships: Relationships represent the associations between entities. They define how entities interact or relate to each other. Relationships can be one-to-one, one-to-many, or many-to-many. In an ER diagram, relationships are depicted using lines connecting the entities involved, with labels indicating the cardinality or nature of the relationship (such as 1:M or M:N)**.**

1. Department: The Department entity represents different departments or units within the organization. It contains attributes such as DepartmentID (unique identifier for each department) and DepartmentName.
2. Employee: The Employee entity represents individual employees within the organization. It includes attributes such as EmployeeID (unique identifier for each employee), Name, Position, ContactInfo, and other relevant employee details. The relationship between Department and Employee is a one-to-many relationship, as one department can have multiple employees, but an employee belongs to only one department.
3. Role: The Role entity represents the different roles or job positions within the organization. It contains attributes like RoleID (unique identifier for each role) and RoleName. The relationship between Employee and Role is a many-to-one relationship, as multiple employees can have the same role, but each employee has only one role.

In this Employee Management System, there are three main entities: Department, Employee, and Role. The Department entity represents the different departments within the organization. The Employee entity represents individual employees, and each employee belongs to one department. The Role entity represents the various job roles within the organization, and multiple employees can have the same role.

The relationships between the entities are as follows:

* Employee has a one-to-many relationship with Department, as one employee belongs to only one department, but a department can have multiple employees.
* Employee has a one-to-one relationship with Role, as each employee holds only one role, and each role belongs to only one employee.

# 

# 

# CHAPTER 4

# SCHEMA DIAGRAM

Schema diagram for the Employee Management System project using Flutter and Microsoft SQL for CRUD operations, with the entities Department, Employee, and Role:

1. Department: The Department entity represents different departments or units within the organization. It contains the following columns:

* DepartmentID (Primary Key): A unique identifier for each department.
* DepartmentName: The name of the department.

1. Employee: The Employee entity represents individual employees within the organization. It includes the following columns:

* EmployeeID (Primary Key): A unique identifier for each employee.
* Name: The name of the employee.
* Position: The job position or role of the employee.
* ContactInfo: Contact information of the employee, such as phone number or email.
* DepartmentID (Foreign Key): A reference to the DepartmentID from the Department entity, indicating the department to which the employee belongs.

1. Role: The Role entity represents the job roles or positions held by employees. It contains the following columns:

* RoleID (Primary Key): A unique identifier for each role.
* RoleName: The name or title of the role.

The relationships between the entities are as follows:

* Employee (DepartmentID) --> Department (DepartmentID): This is a one-to-many relationship, where an employee belongs to one department, and a department can have multiple employees.
* Employee (RoleID) --> Role (RoleID): This is a one-to-many relationship, where an employee holds one role, and a role can be associated with multiple employees.

In this schema diagram, the Department entity represents the different departments within the organization. The Employee entity represents individual employees and includes a foreign key (DepartmentID) to establish a relationship with the Department entity. The Role entity represents the various job roles or positions and is associated with employees through the RoleID foreign key in the Employee entity.

# SCHEMA DIAGRAM

login

|  |  |
| --- | --- |
| email | password |

Department

|  |  |
| --- | --- |
| dept\_name | dept\_id |

Now, let's look at the attributes for the "Department" entity:

**TABLE NAME :** Department

**Columns:**

1. **Dept\_ID**: This attribute serves as the primary key of the "Department" table, ensuring each department has a unique identifier.
2. **Dept\_Name**: Represents the name of the department.

role

|  |  |
| --- | --- |
| Role | role\_id |

**TABLE NAME :** role

**Columns:**

1. **Des\_ID**: This attribute serves as the primary key of the "Designation" table, ensuring each designation has a unique identifier.
2. **Des\_Name**: Represents the name or title of the designation.

employee

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | name | email | age | exp | sal | dept\_id | dept\_  name | des\_ id | des\_name |

**TABLE NAME :** employee

**Columns:**

1. **ID**: This attribute serves as the primary key of the "Employee" table, ensuring each employee has a unique identifier.
2. **Name**: Represents the name of the employee.
3. **Email**: Represents the email address of the employee.
4. **Age**: Stores the age of the employee.
5. **Experience**: Represents the number of years of experience the employee has.
6. **Salary**: Stores the salary of the employee.
7. **DepartmentID**: This attribute is a foreign key referencing the "Department" entity. It establishes a relationship between the employee and the department they belong to.

# CHAPTER 5

# NORMALIZATION

Normalization is the process of organizing data in a database to eliminate redundancy and improve data integrity. It involves decomposing a database schema into multiple tables and defining relationships between them. Here's an example of how normalization can be applied to the Employee Management System project, resulting in the final normalized tables:

1. Initial Table Design:

* Employee (EmployeeID, Name, Position, ContactInfo, DepartmentID)
* Department (DepartmentID, DepartmentName).

1. First Normal Form (1NF):

* Employee (EmployeeID, Name, Position, ContactInfo)
* Department (DepartmentID, DepartmentName)

In 1NF, each column contains only atomic values, and there are no repeating groups or arrays within a column.

1. Second Normal Form (2NF):

* Employee (EmployeeID, Name, Position, ContactInfo)
* Department (DepartmentID, DepartmentName)

In 2NF, we ensure that non-key attributes depend on the whole primary key, and there are no partial dependencies. Since there are no composite primary keys in our initial design, it is already in 2NF.

1. Third Normal Form (3NF):

* Employee (EmployeeID, Name, Position, ContactInfo)
* Department (DepartmentID, DepartmentName)

In 3NF, we remove transitive dependencies. Since there are no dependencies between non-key attributes in our initial design, it is already in 3NF.

The final tables remain the same as the initial design, as they are already in 3NF and meet the requirements of the Employee Management System. The Employee table contains employee-specific information, while the Department table stores department-related information.

Final Table used

employee

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | name | email | age | exp | sal | dept\_id | dept\_  name | des\_ id | des\_name |

**TABLE NAME :** employee

1. **ID**: This attribute serves as the primary key of the "Employee" table, ensuring each employee has a unique identifier.
2. **Name**: Represents the name of the employee.
3. **Email**: Represents the email address of the employee.
4. **Age**: Stores the age of the employee.
5. **Experience**: Represents the number of years of experience the employee has.
6. **Salary**: Stores the salary of the employee.
7. **DepartmentID**: This attribute is a foreign key referencing the "Department" entity. It establishes a relationship between the employee and the department they belong to.

**CHAPTER 6**

**APPENDIX**

**6.1 SOURCE CODE**

Widget build(BuildContext context) {  
 return Scaffold(  
 backgroundColor: Colors.*greenAccent*,  
 appBar: AppBar(  
 title: Text("Employee Management System"),  
 backgroundColor: Colors.*black*,  
 bottom: TabBar(  
 controller: \_controler1, tabs: [  
 Row(  
 children:[  
  
 Text("Create",style: TextStyle(fontSize: 20),),  
 ]  
 ),  
 Row(  
 children:[  
 Icon(Icons.*edit\_attributes*),  
 Text(" Edit",style: TextStyle(fontSize: 20),),  
 ]  
 ),  
 Row(  
 children:[  
 Icon(Icons.*view\_array\_outlined*),  
 Text("View",style: TextStyle(fontSize: 20),),  
 ]  
 ),  
 Row(  
 children:[  
  
 Text("Details",style: TextStyle(fontSize: 20),),  
 ]  
 ),  
  
 ],  
 ),  
 ),  
 body: TabBarView(  
 controller: \_controler1,  
 children: [  
 ListView(  
 shrinkWrap: true,  
 children: [  
 Form(  
 key: formKey2,  
 child:Column(  
 children: [  
 Row(  
 children: [  
 Padding(  
 padding: EdgeInsets.symmetric(horizontal:20,vertical:40),  
 child:Text("Id ",style: TextStyle(fontSize: 20),),  
 ),  
 Expanded(  
 child:TextFormField(  
 controller:id\_control,  
 validator: (value){  
 if (value!.isEmpty){  
 return "Please enter your Id";  
 }  
 },  
 style: TextStyle(color: Colors.*black*),  
 decoration: InputDecoration(  
 hintText: 'Enter your Id',hintStyle: TextStyle(color: Colors.*grey*[20]),  
 enabledBorder: OutlineInputBorder(  
 borderSide: BorderSide(  
 width: 3, color: Colors.*black*),  
 ),  
 ),  
 ),  
 ),  
 ],  
 ),  
 Row(  
 children: [  
  
 Padding(  
 padding: EdgeInsets.symmetric(horizontal:20,vertical:00),  
 child:Text("Name ",style: TextStyle(fontSize: 20),),  
 ),  
 Expanded(  
 child:TextFormField(  
 controller: name\_control,  
 validator: (value){  
 if (value!.isEmpty){  
 return "Please enter your name";  
 }  
 },  
 style: TextStyle(color: Colors.*black*),  
 decoration: InputDecoration(  
 hintText: 'Enter your name',hintStyle: TextStyle(color: Colors.*grey*[20]),  
 enabledBorder: OutlineInputBorder(  
 borderSide: BorderSide(  
 width: 3, color: Colors.*black*),  
 ),  
 ),  
 ),  
 ),  
 ],  
 ),  
 Row(  
 children: [  
  
 Padding(  
 padding: EdgeInsets.symmetric(horizontal:20,vertical:40),  
 child:Text("Email ",style: TextStyle(fontSize: 20),),  
 ),  
 Expanded(  
 child:TextFormField(  
 controller: email\_control,  
 validator: (value){  
 if (EmailValidator.*validate*(email\_control.text)==false || email\_control.text!=em){  
 return "Please enter the email used for login";  
 }  
 },  
 style: TextStyle(color: Colors.*black*),  
 decoration: InputDecoration(  
 hintText: 'Enter your email',hintStyle: TextStyle(color: Colors.*grey*[20]),  
 enabledBorder: OutlineInputBorder(  
 borderSide: BorderSide(  
 width: 3, color: Colors.*black*),  
 ),  
 ),  
 ),  
 ),  
 ],  
 ),  
 Row(  
 children: [  
  
 Padding(  
 padding: EdgeInsets.symmetric(horizontal:20,vertical:20),  
 child:Text("Age ",style: TextStyle(fontSize: 20),),  
 ),  
 Expanded(  
 child:TextFormField(  
 controller: age\_control,  
 validator: (value){  
 if (value!.isEmpty){  
 return "Please enter your age";  
 }  
 },  
 style: TextStyle(color: Colors.*black*),  
 decoration: InputDecoration(  
 hintText: 'Enter your age',hintStyle: TextStyle(color: Colors.*grey*[20]),  
 enabledBorder: OutlineInputBorder(  
 borderSide: BorderSide(  
 width: 3, color: Colors.*black*),  
 ),  
 ),  
 ),  
 ),  
 ],  
 ),  
 Row(  
 children: [  
  
 const Padding(  
 padding: EdgeInsets.symmetric(horizontal:20,vertical:40),  
 child:Text("Experience",style: TextStyle(fontSize: 20),),  
 ),  
 Expanded(  
 child:TextFormField(  
 controller: exp\_control,  
 validator: (value){  
 if (value!.isEmpty){  
 return "Please enter your experiance";  
 }  
 },  
 style: TextStyle(color: Colors.*black*),  
 decoration: InputDecoration(  
 hintText: 'Enter your experience(in years)',hintStyle: TextStyle(color: Colors.*grey*[20]),  
 enabledBorder: OutlineInputBorder(  
 borderSide: BorderSide(  
 width: 3, color: Colors.*black*),  
 ),  
 ),  
 ),  
 ),  
 ],  
 ),  
 Row(  
 children: [  
  
 const Padding(  
 padding: EdgeInsets.symmetric(horizontal:20,vertical:20),  
 child:Text("Salary ",style: TextStyle(fontSize: 20),),  
 ),  
 Expanded(  
 child:TextFormField(  
 controller: sal\_control,  
 validator: (value){  
 if (value!.isEmpty){  
 return "Please enter your Salary";  
 }  
 },  
 style: TextStyle(color: Colors.*black*),  
 decoration: InputDecoration(  
 hintText: 'Enter your Salary',hintStyle: TextStyle(color: Colors.*grey*[20]),  
 enabledBorder: OutlineInputBorder(  
 borderSide: BorderSide(  
 width: 3, color: Colors.*black*),  
 ),  
 ),  
 ),  
 ),  
 ],  
 ),  
 Row(  
 children: [  
  
 const Padding(  
 padding: EdgeInsets.symmetric(horizontal:20,vertical:40),  
 child:Text("Dep\_Id ",style: TextStyle(fontSize: 20),),  
 ),  
 Expanded(  
 child:DropdownButton<String>(  
 value: dropdownValue1,  
 items: <String>['Choose the department\_id','1', '2', '3', '4','5','6','7','8']  
 .map<DropdownMenuItem<String>>((String value) {  
 return DropdownMenuItem<String>(  
 value: value,  
 child: Text(value, style: TextStyle(fontSize: 20),),);  
 }).toList(),  
 underline: Container( height: 2, color: Colors.*black*),  
 onChanged: (String? newValue) {  
 setState(() {  
 dropdownValue1 = newValue!;  
 });  
 },  
 ),  
 ),  
 ],  
 ),  
 Row(  
 children: [  
  
 const Padding(  
 padding: EdgeInsets.symmetric(horizontal:20,vertical:20),  
 child:Text("Dep\_Name",style: TextStyle(fontSize: 20),),  
 ),  
 Expanded(  
 child: DropdownButton<String>(  
 value: dropdownValue,  
 items: <String>['Choose the department','CSE', 'EEE', 'ECE', 'CIVIL','MECH','IT','AIDS','CSBS']  
 .map<DropdownMenuItem<String>>((String value) {  
 return DropdownMenuItem<String>(  
 value: value,  
 child: Text(value, style: TextStyle(fontSize: 20),),);  
 }).toList(),  
 underline: Container( height: 2, color: Colors.*black*),  
 onChanged: (String? newValue) {  
 setState(() {  
 dropdownValue = newValue!;  
 });  
 },  
 ),  
 ),  
 ],  
 ),  
 Row(  
 children: [  
  
 const Padding(  
 padding: EdgeInsets.symmetric(horizontal:20,vertical:40),  
 child:Text("Des\_Id ",style: TextStyle(fontSize: 20),),  
 ),  
 Expanded(  
 child:DropdownButton<String>(  
 value: dropdownValue2,  
 items: <String>['Choose the designation\_id','1','2','3','4']  
 .map<DropdownMenuItem<String>>((String value) {  
 return DropdownMenuItem<String>(  
 value: value,  
 child: Text(value, style: TextStyle(fontSize: 20),),);  
 }).toList(),  
 underline: Container( height: 2, color: Colors.*black*),  
 onChanged: (String? newValue) {  
 setState(() {  
 dropdownValue2 = newValue!;  
 });  
 },  
 ),  
 ),  
 ],  
 ),  
 Row(  
 children: [  
  
 const Padding(  
 padding: EdgeInsets.symmetric(horizontal:20,vertical:20),  
 child:Text("Des\_Name",style: TextStyle(fontSize: 20),),  
 ),  
 Expanded(  
 child:DropdownButton<String>(  
 value: dropdownValue3,  
 items: <String>['Choose the designation','Head of the Department', 'Associate Professor', 'Assistant Professor','Technician']  
 .map<DropdownMenuItem<String>>((String value) {  
 return DropdownMenuItem<String>(  
 value: value,  
 child: Text(value, style: TextStyle(fontSize: 20),),);  
 }).toList(),  
 underline: Container( height: 2, color: Colors.*black*),  
 onChanged: (String? newValue) {  
 setState(() {  
 dropdownValue3 = newValue!;  
 });  
 },  
 ),  
 ),  
 ],  
 ),  
 ],  
 ),  
 ),

**6.1.1 Sign in tab**

**Description:**

The "Sign In" tab in an employee management system is a feature that allows users to authenticate themselves and gain access to the system. It provides a secure mechanism for users to verify their identity and access their account or the system's functionalities. Here's a description of the "Sign In" tab and its functionality:

1. Username/Email and Password Fields: The "Sign In" tab typically consists of two input fields: one for the username or email and another for the password. Users need to enter their registered username or email and the corresponding password associated with their account.
2. Sign In Button: The "Sign In" tab includes a "Sign In" or "Login" button. Once users have entered their username/email and password, they can click this button to initiate the authentication process.
3. Authentication and Error Handling: Upon clicking the "Sign In" button, the system verifies the entered credentials by comparing them with the stored user data in the database. If the credentials match, the user is granted access to their account or the system. If the credentials are incorrect, an appropriate error message is displayed, indicating that the authentication failed. Error handling should ensure that sensitive error details are not exposed, enhancing the security of the login process.

The "Sign In" tab is crucial for ensuring the security and controlled access to the employee management system. It requires users to provide valid credentials, and the system performs authentication to verify their identity before granting access to the system's resources.

**6.1.2 Create tab**

**Description:**

The "Create" tab in an employee management system is a feature that allows users to add new employee records to the system. It provides a user-friendly interface to input and store all the necessary information about a new employee. Here's a description of the "Create" tab and its functionality:

1. Form Fields: The "Create" tab typically consists of a form with various fields where users can enter the details of the new employee. These fields may include:
   * Employee ID: A unique identifier for the employee (automatically generated or manually entered).
   * Name: The employee's full name.
   * Email: The employee's email address.
   * Age: The age or date of birth of the employee.
   * Experience: The number of years of experience the employee has.
   * Salary: The employee's salary or wage.
   * Department: A drop-down menu or selection field to choose the department the employee belongs to.
   * Other relevant fields: Additional fields may be included based on the specific requirements of the system, such as address, phone number, job title, etc.
2. Data Validation: The "Create" tab should perform data validation to ensure that the entered information is accurate and complete. It may include checks for valid email formats, age restrictions, salary ranges, and any other relevant validation rules.
3. Save/Submit Button: The "Create" tab typically includes a "Save" or "Submit" button to store the entered employee data in the system. Upon clicking the button, the system will process and save the information as a new employee record in the database.
4. Error Handling: If any errors or validation issues occur during the data entry process, the "Create" tab should display appropriate error messages to the user, highlighting the fields that require attention or correction.
5. Success Message: After successfully creating a new employee record, the "Create" tab may display a success message to confirm that the employee has been added to the system.

Overall, the "Create" tab simplifies the process of adding new employees to the employee management system by providing an organized and intuitive interface for entering their details, validating the data, and storing it securely in the database.

**6.1.3 View tab**

**Description:**

The "Employee Details" tab in an employee management system is a feature that allows users to view and access detailed information about a specific employee. It provides a comprehensive overview of an employee's profile, including personal details, employment information, and any other relevant information stored in the system. Here's a description of the "Employee Details" tab and its functionality:

1. **Employee Selection**: The "Employee Details" tab typically includes a search or selection mechanism where users can choose a specific employee whose details they want to view. This can be done by entering the employee's ID, name, or using a dropdown menu to select from a list of available employees.
2. **Employee Profile**: Once an employee is selected, the "Employee Details" tab displays a profile section that presents a summary of the employee's information. This may include the employee's name, ID, position, department, and other key details that provide a quick overview of the employee's role within the organization.
3. **Personal Information**: The tab provides a section to display personal information about the employee, such as their date of birth, gender, contact details (email, phone number, address), emergency contact information, and any other relevant personal details.
4. **Employment Details**: The "Employee Details" tab includes a section dedicated to the employee's employment information. This typically includes details such as the date of joining the company, current employment status, job title or position, department, reporting manager or supervisor, and any other pertinent employment-related information.
5. **Salary and Compensation**: If applicable, the tab may include a section that provides information about the employee's salary and compensation. This could include details about the salary structure, pay scale, allowances, bonuses, and any other components of the employee's compensation package.
6. **Performance and Reviews**: In some cases, the "Employee Details" tab may also display information related to the employee's performance, such as performance reviews, ratings, or feedback. This section can provide insights into the employee's work performance and progress within the organization.
7. **Additional Information**: Depending on the specific requirements of the employee management system, there may be additional sections or tabs within the "Employee Details" tab to display other relevant information. This can include details about training and development, certifications, qualifications, work history, or any custom fields that are relevant to the organization.

The "Employee Details" tab provides a consolidated view of an employee's information, allowing users to quickly access and review the comprehensive details associated with a specific employee. It ensures that relevant employee information is readily available to authorized users within the employee management system

**6.1.4 Edit tab**

**Description:**

**Edit Tab**:

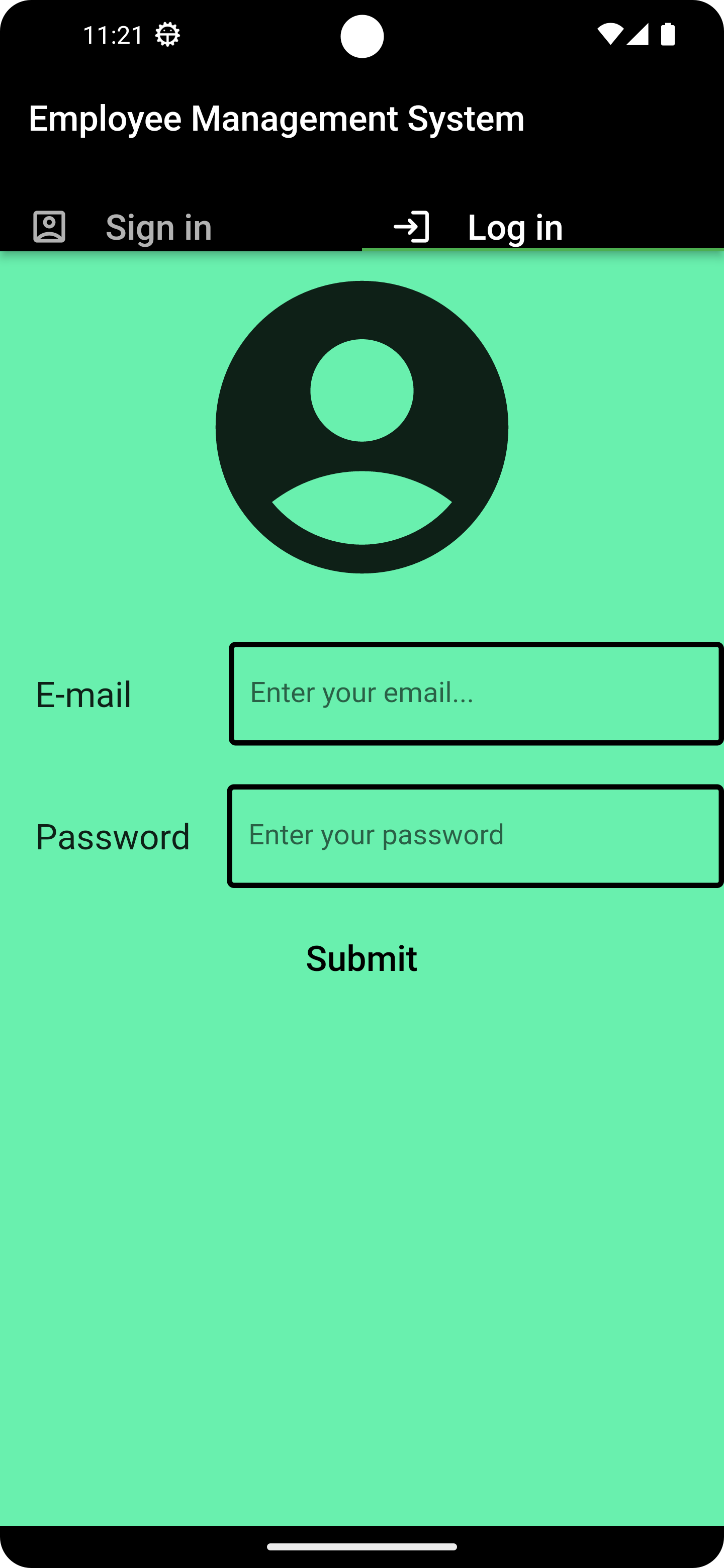
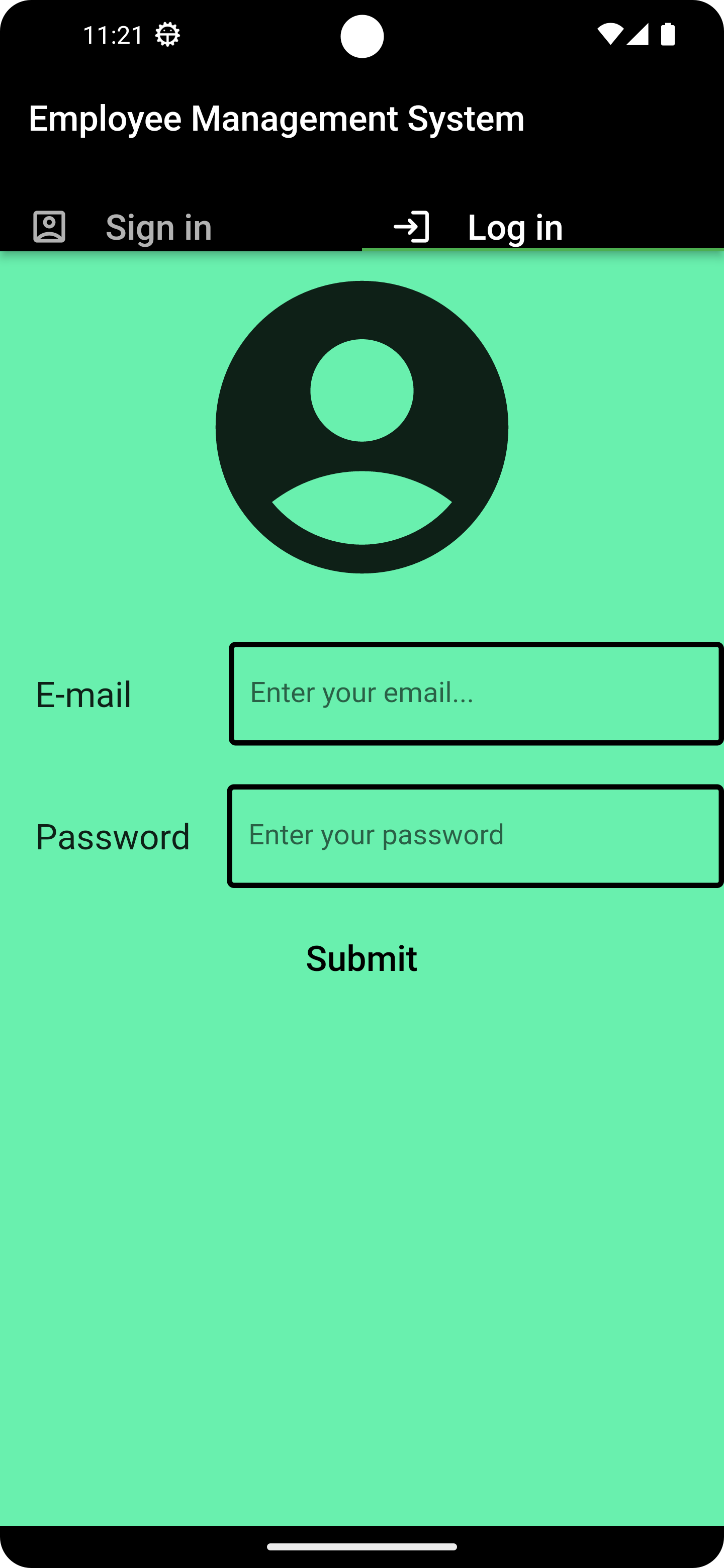
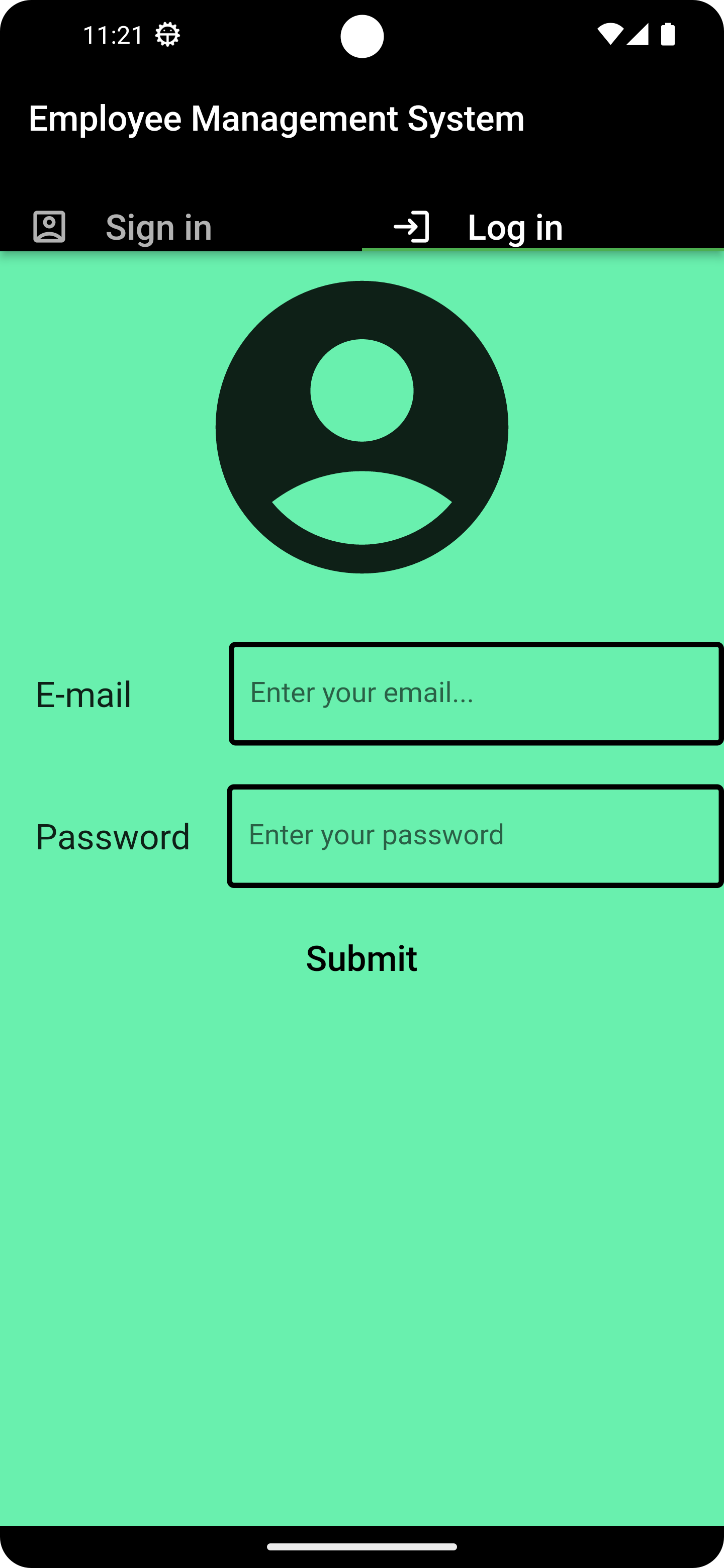
* + Employee Selection: The "Edit" tab typically includes a search or selection mechanism to choose the employee whose information needs to be edited. Users can search for employees based on their ID, name, or other relevant criteria.
  + Display Existing Information: Once an employee is selected, the "Edit" tab displays the current information associated with that employee, such as personal details, employment information, and any other relevant data.
  + Edit Fields: Users can modify the displayed fields and make changes to the employee's information. This can include updating personal details, employment details, salary, position, department, or any other editable fields.
  + Validation and Error Handling: The "Edit" tab should perform validation checks to ensure that the entered data is accurate and complete. If any errors or inconsistencies are found, appropriate error messages should be displayed, guiding users to correct the issues before saving the changes.
  + Save Changes: Once the necessary modifications have been made, users can click a "Save" or "Update" button to save the edited information. The system should then update the employee record with the new data.

**6.1.5 Delete tab**

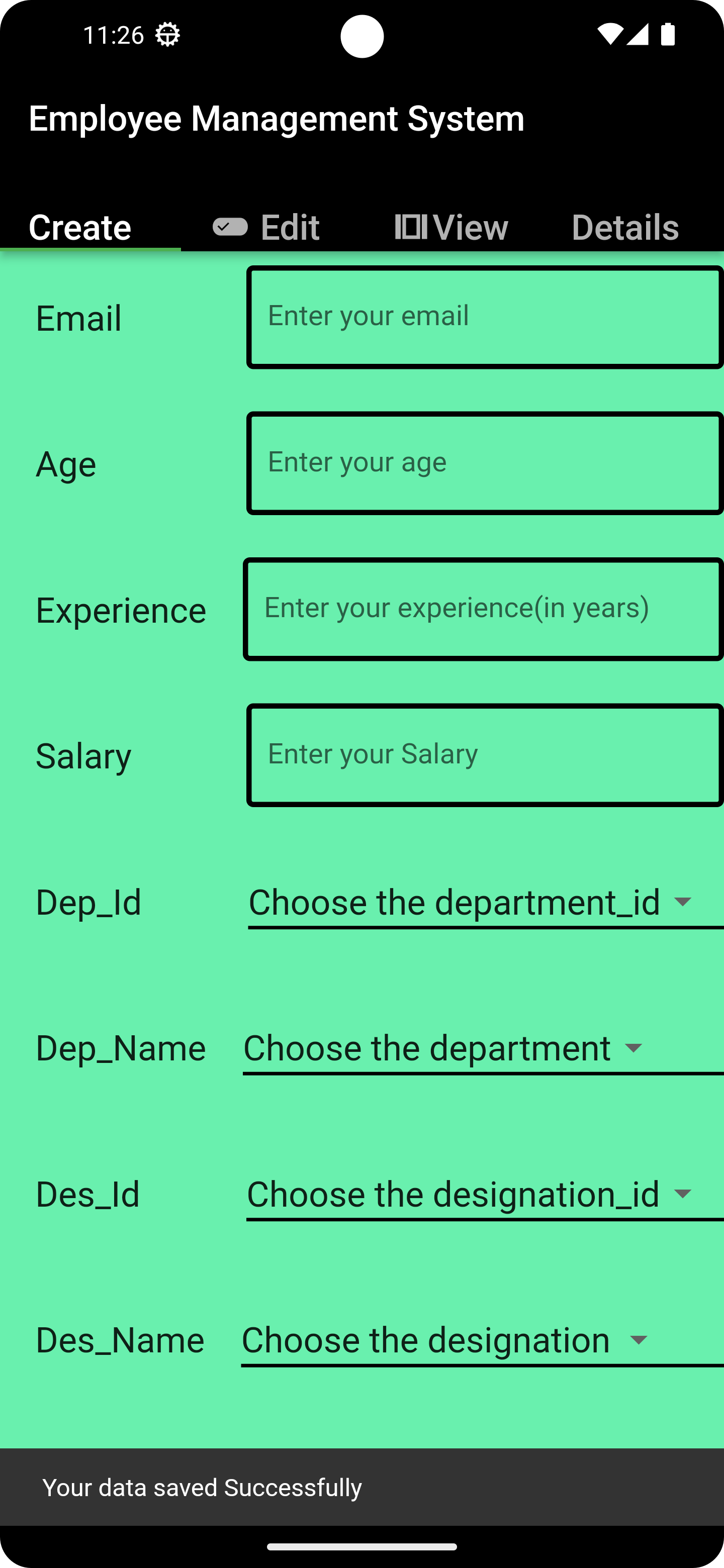
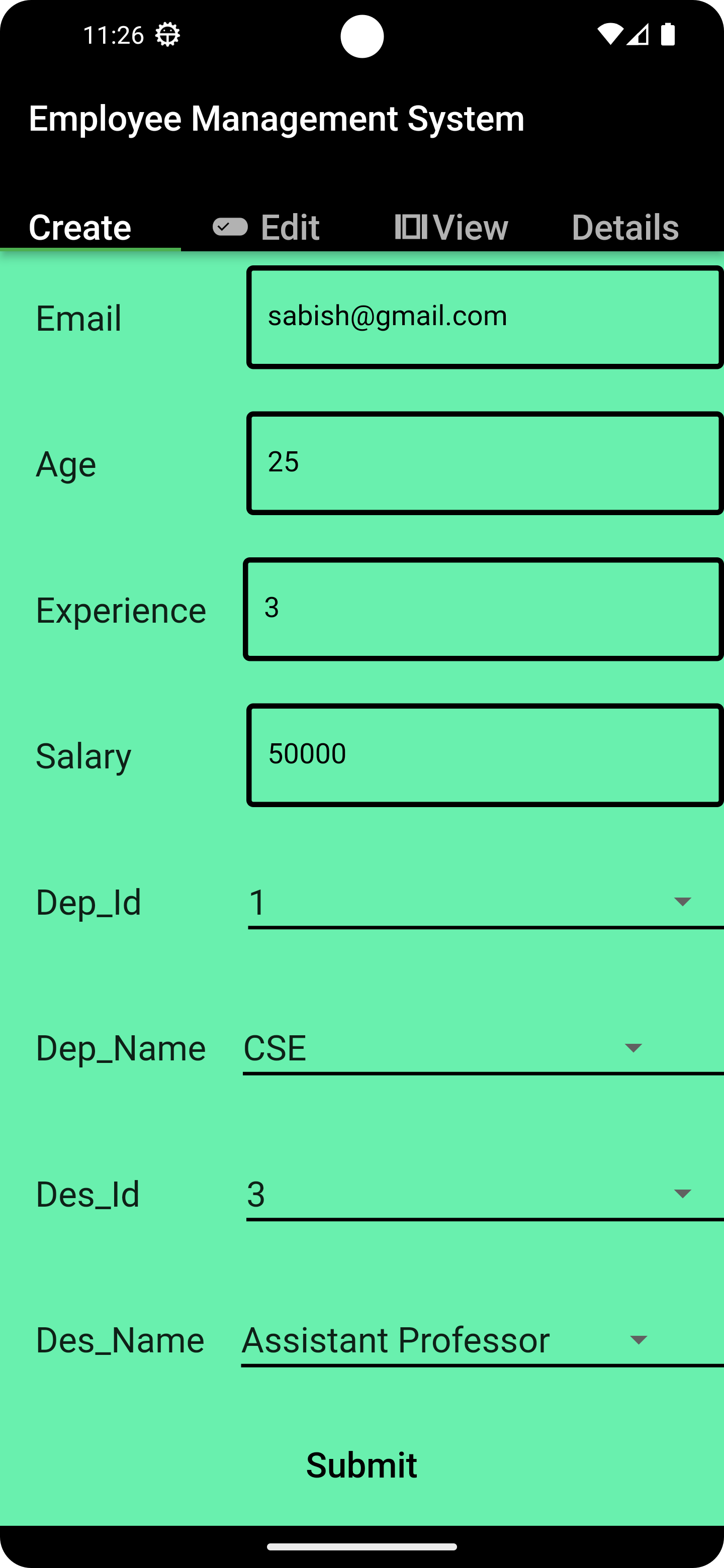
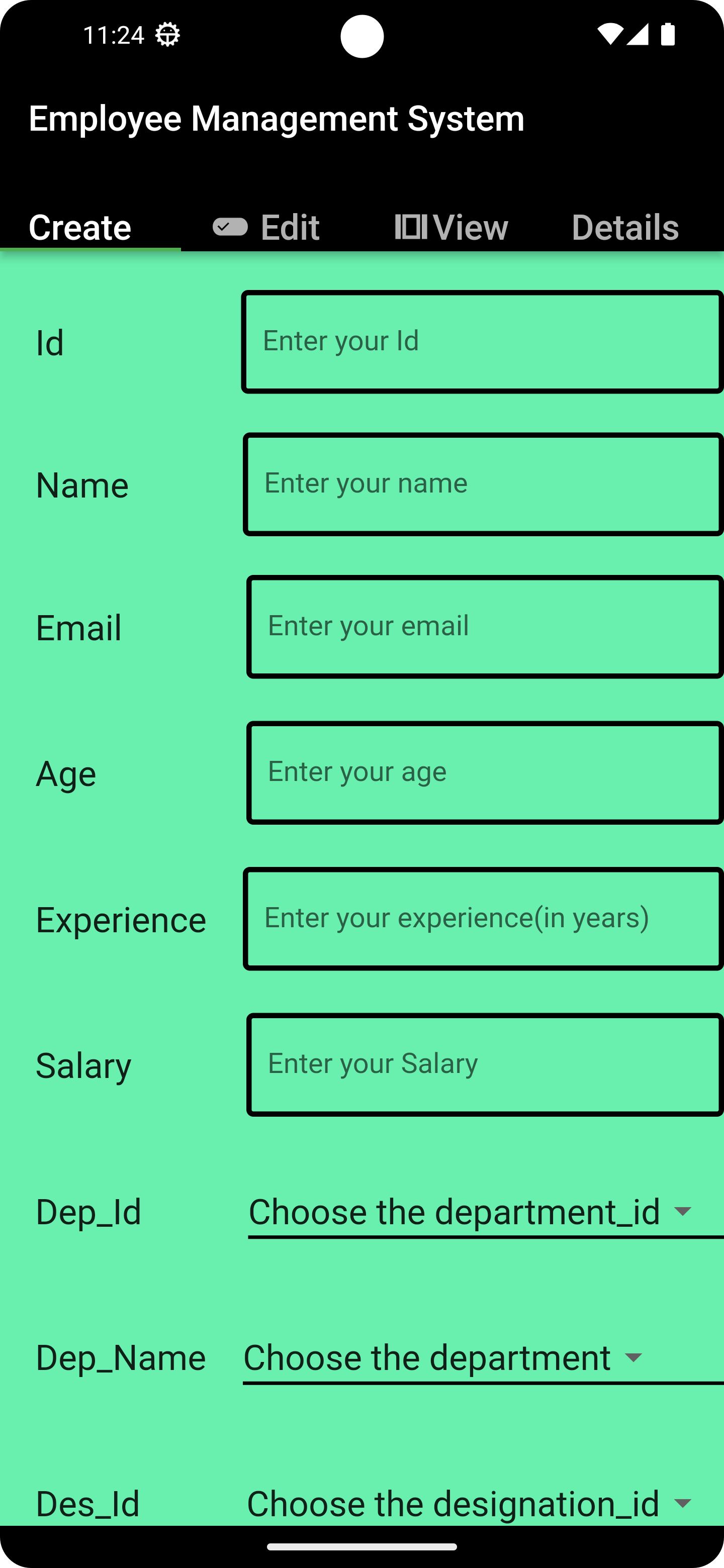
**Description:**

1. **Delete Tab**:
   * Employee Selection: The "Delete" tab allows users to select the employee they wish to remove from the system. Similar to the "Edit" tab, this can be done by searching for the employee based on their ID, name, or other relevant criteria.
   * Confirmation Prompt: After selecting the employee to be deleted, the "Delete" tab typically displays a confirmation prompt to ensure the user's intention. This is done to prevent accidental deletions. The prompt may ask the user to confirm the deletion or provide additional information about the consequences of deleting the employee record.
   * Delete Action: If the user confirms the deletion, clicking a "Delete" or "Remove" button will initiate the process of removing the selected employee record from the system's database. Once deleted, the employee's information will no longer be accessible

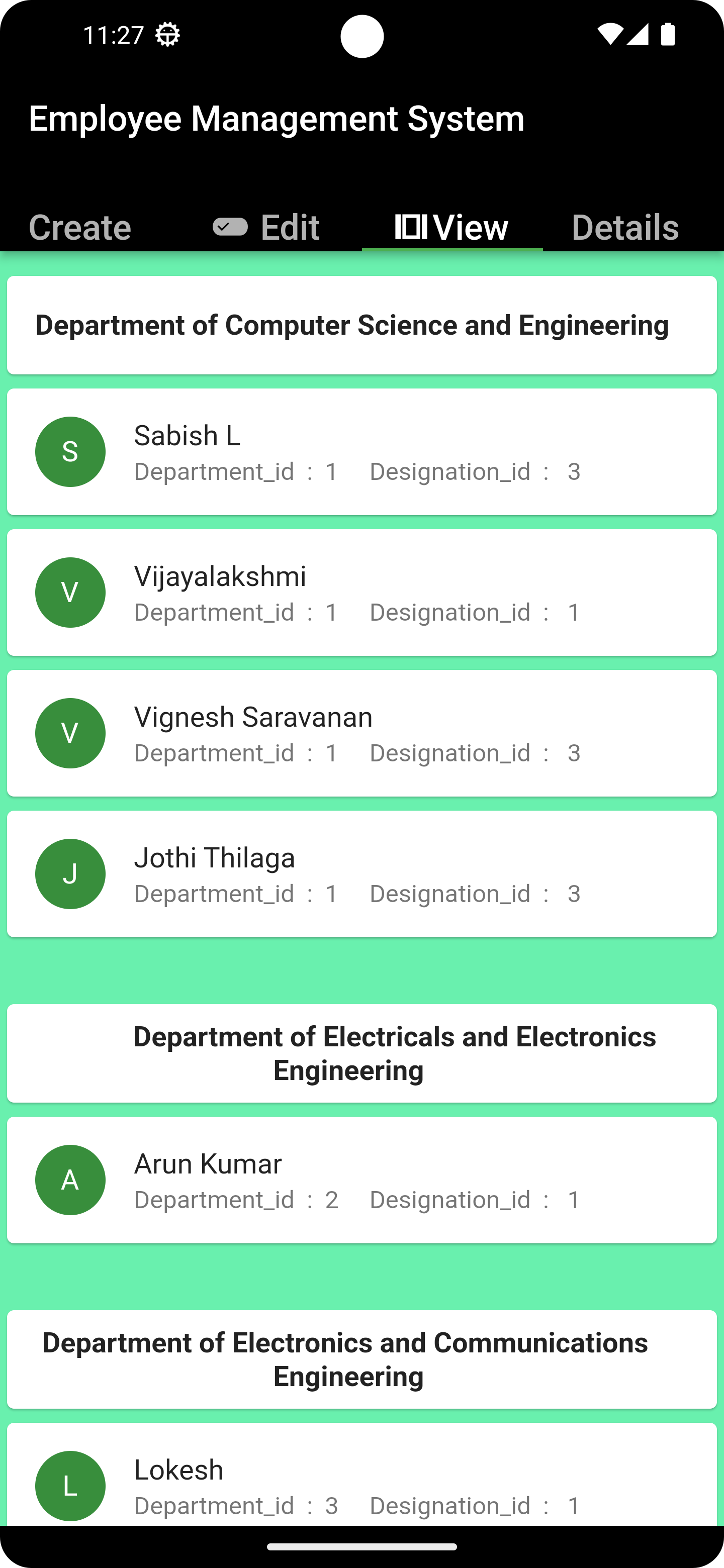
**CHAPTER 7 SCREENSHOTS**

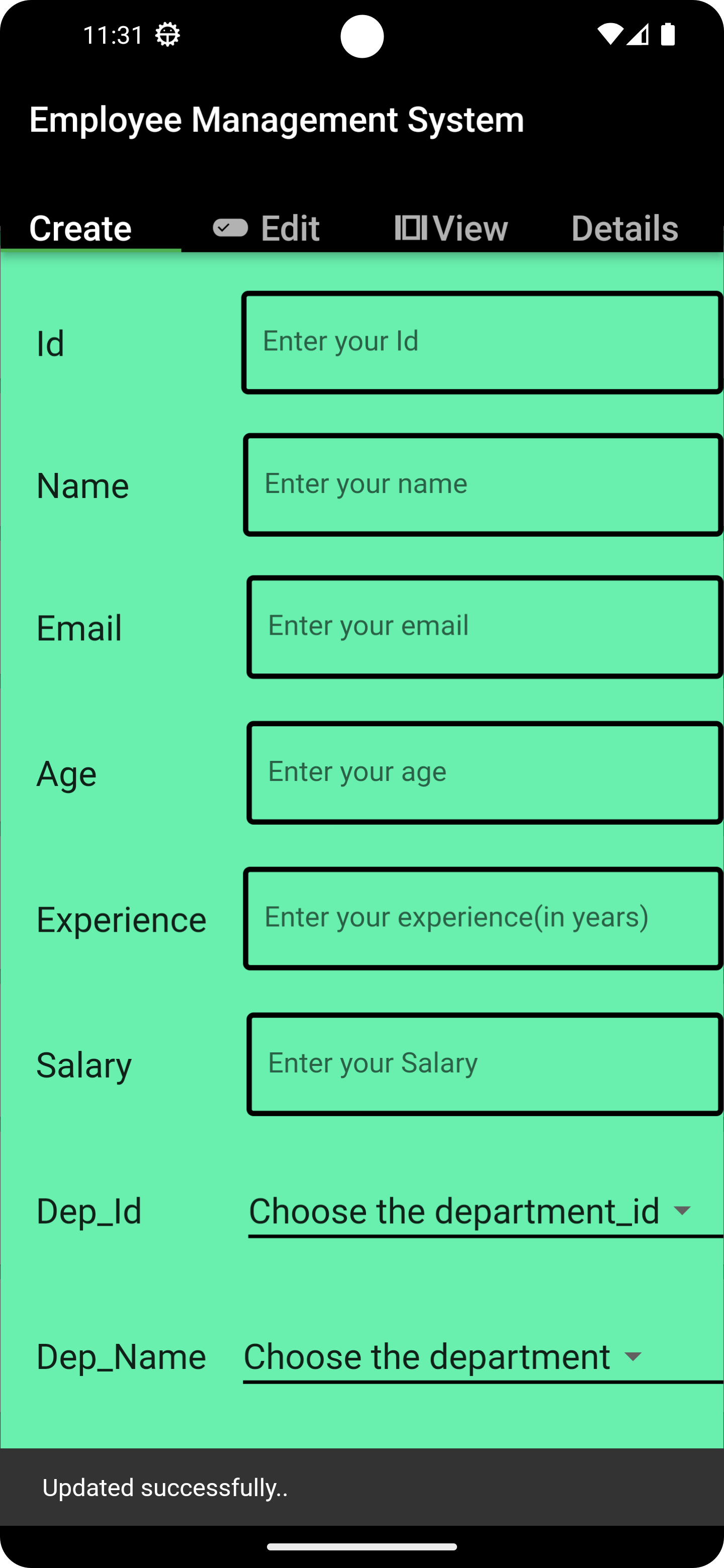
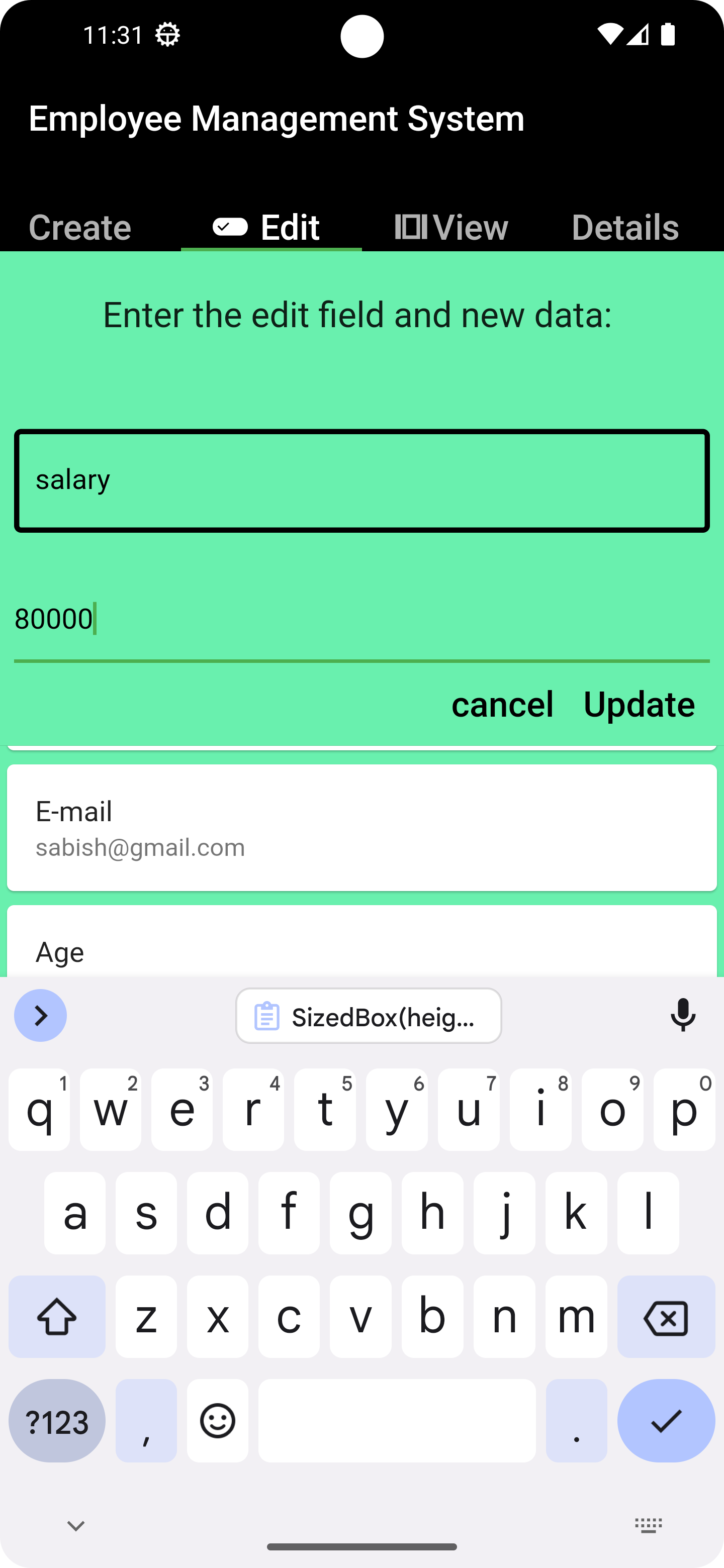
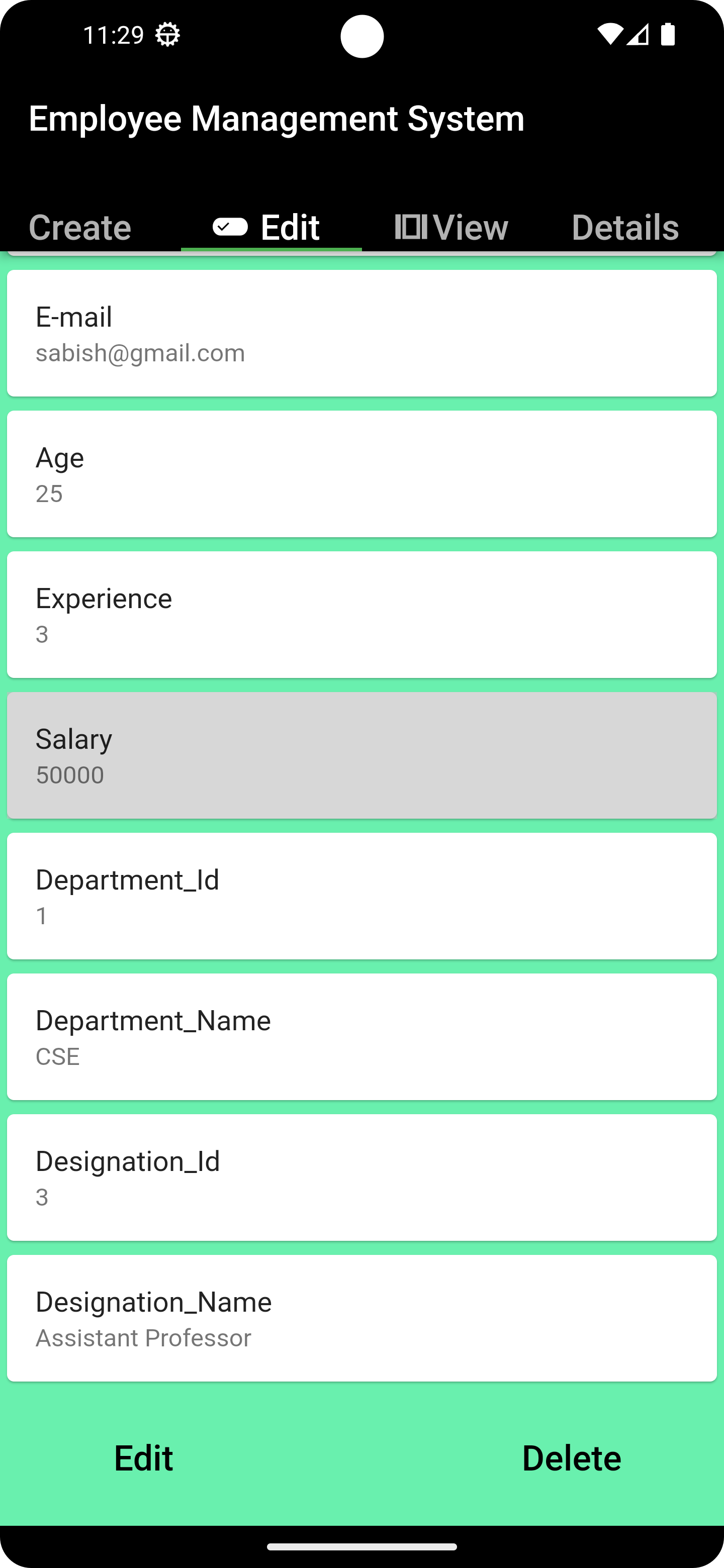
**FIGURE – 1 – LOGIN OR SIGN IN**



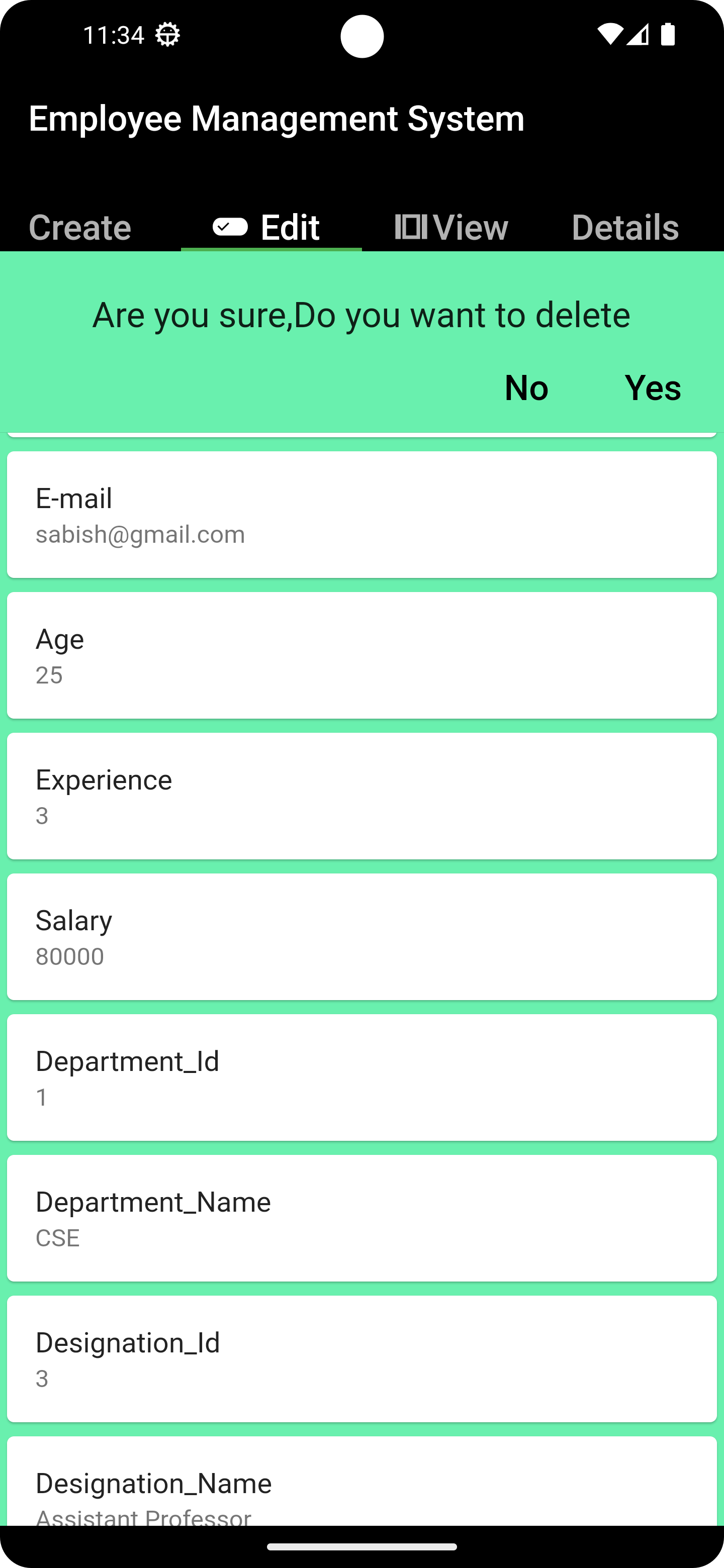
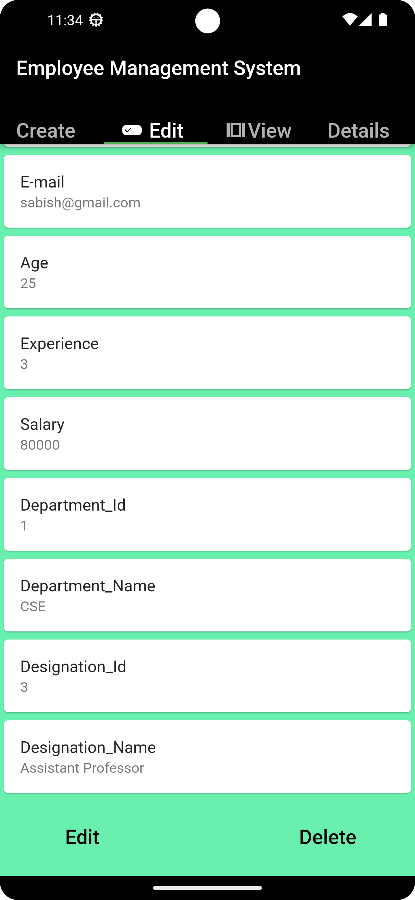
**FIGURE – 2 – CREATE**

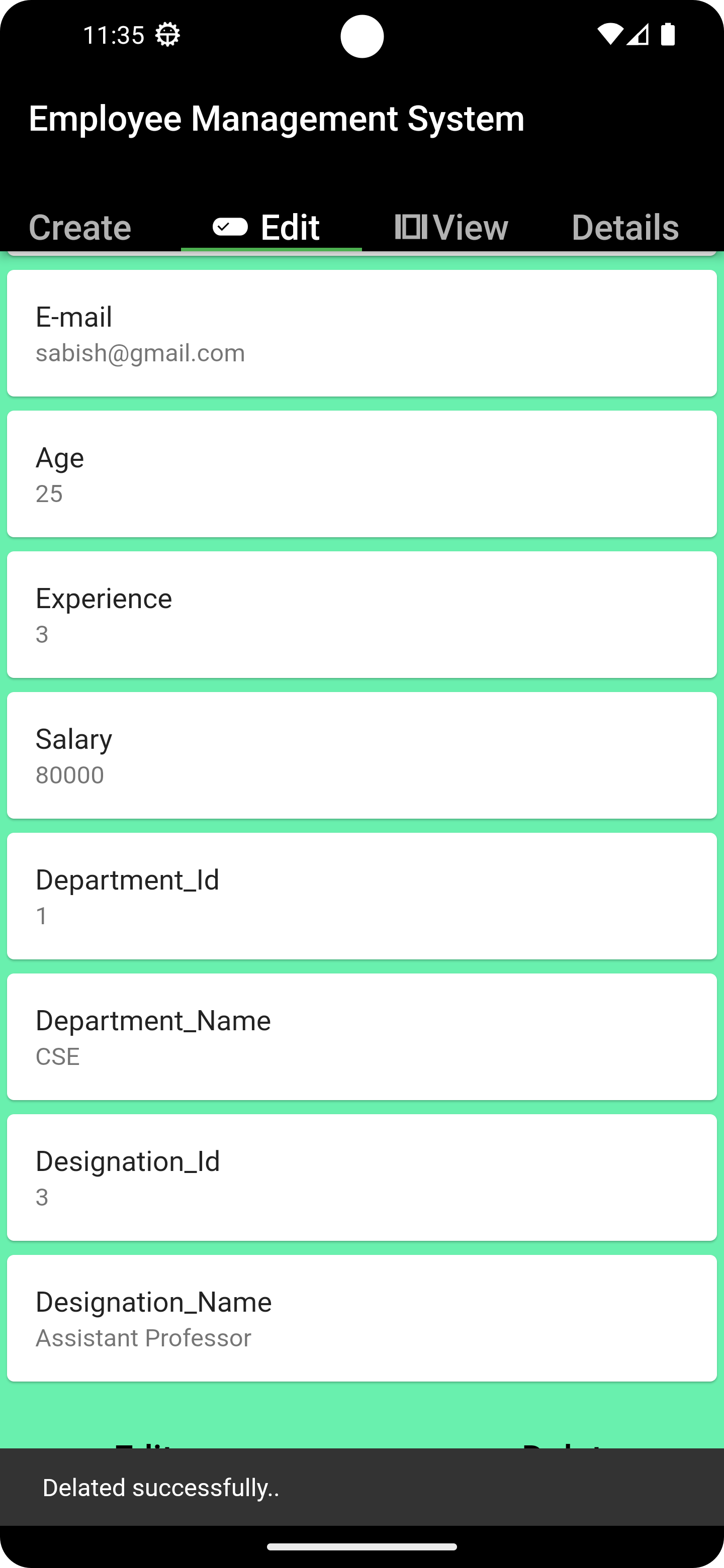
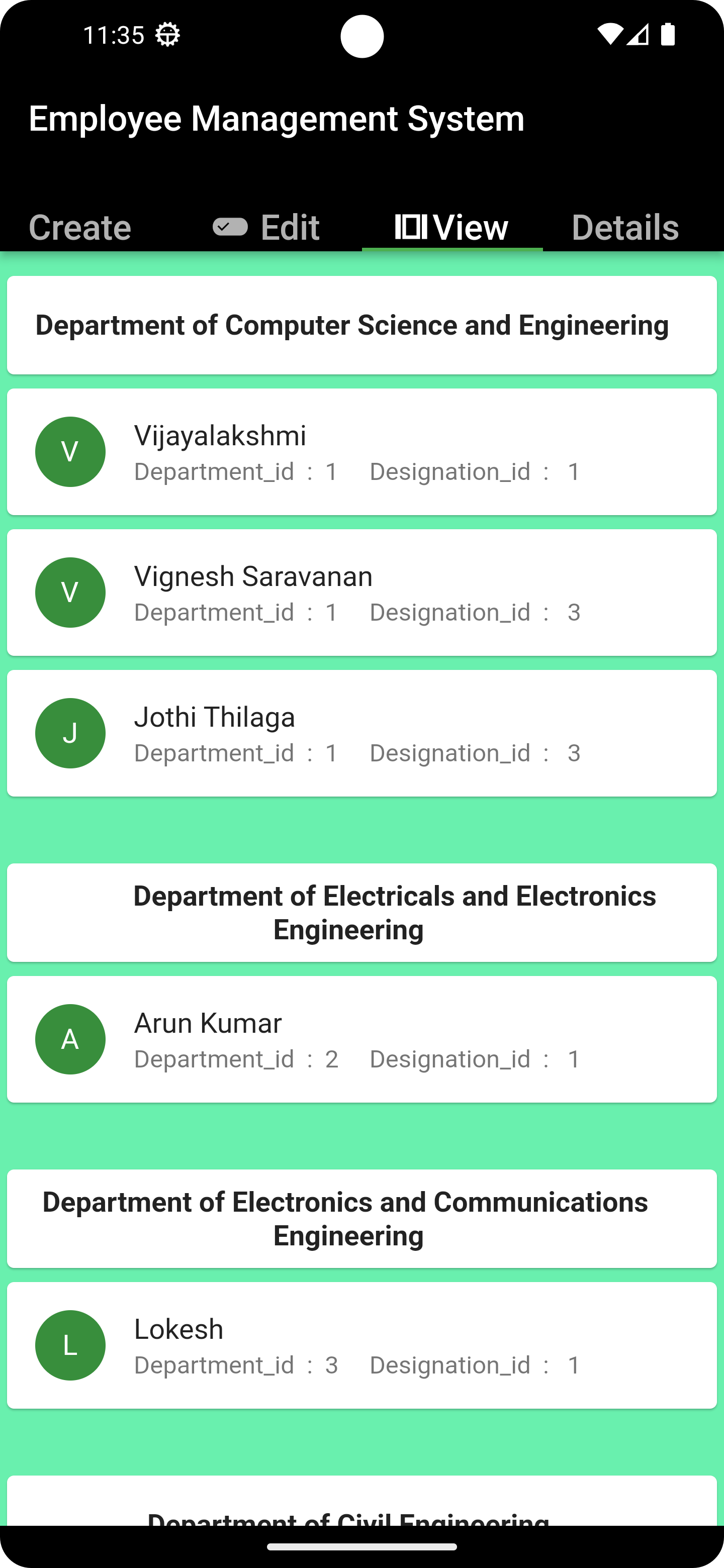
 

**FIGURE – 3 –VIEW FIGURE – 4–DETAILS**



**FIGURE – 5 –EDIT OR UPDATE**



**FIGURE – 6–DELETE**

**CHAPTER 8**

**CONCLUSION**

In conclusion, the Employee Management System (EMS) developed using Flutter is an advanced application that empowers organizations to efficiently manage their employee data. By providing CRUD operations, user authentication, and data visualization features, the EMS project simplifies the complexities associated with employee management. With this innovative solution, organizations can streamline their operations, ensure data accuracy, and make informed decisions based on comprehensive employee insights.

**CHAPTER 9**

**FUTURE WORK**

In the future, employee management systems using Flutter can be enhanced by incorporating the following CRUD (Create, Read, Update, Delete) operations:

1. Create: Allow administrators or managers to create new employee records by capturing relevant information such as name, contact details, position, department, etc. Implement form validation to ensure data integrity and prevent erroneous entries.
2. Read: Provide a comprehensive view of all employee records, allowing authorized users to search, sort, and filter employees based on various criteria. Implement pagination or lazy loading for efficient handling of large datasets.
3. Update: Enable authorized users to modify employee details as required. Implement an intuitive user interface that allows for easy editing of individual fields or a complete record. Apply validation checks to ensure data consistency during updates.
4. Delete: Allow administrators or managers to remove employee records from the system. Implement confirmation dialogs to prevent accidental deletions and provide appropriate feedback on successful deletion.

In addition to these basic CRUD operations, you can consider the following future enhancements for an employee management system:

1. Authentication and Authorization: Implement user authentication and role-based access control to ensure that only authorized users can perform CRUD operations on employee records. This can be achieved using technologies such as Firebase Authentication or OAuth.
2. Image Upload: Enable the ability to upload employee profile pictures or other relevant images. Implement image cropping and resizing to maintain consistency within the system.
3. Notifications and Reminders: Integrate push notifications or email notifications to inform employees or managers about important updates or reminders related to employee management. This can include notifications for upcoming appraisals, birthdays, or other significant events.
4. Reporting and Analytics: Provide data visualization and reporting capabilities to help managers gain insights into employee performance, attendance, or any other relevant metrics. Implement charts, graphs, or dashboards to represent the data effectively.
5. Integration with HR Systems: Integrate the employee management system with other HR systems such as payroll, leave management, or performance management systems. This enables seamless data exchange and reduces data entry duplication.
6. Offline Support: Implement offline capabilities, allowing users to access and modify employee data even when there is no internet connectivity. Sync the changes automatically when the device is back online.
7. Cross-Platform Support: Consider making the employee management system compatible with multiple platforms, such as iOS, Android, and web. Flutter's multi-platform capabilities make it easier to achieve this.

Remember to prioritize data security and privacy by implementing appropriate encryption and access controls. Regularly update the system to address any security vulnerabilities and incorporate user feedback for continuous improvement.

**CHAPTER 10**

**REFERENCE**

[**https://github.com/Poonam0311/employeemanagementapp**](https://github.com/Poonam0311/employeemanagementapp)

[**https://flutterawesome.com/sample-project-for-an-employee-management-application/**](https://flutterawesome.com/sample-project-for-an-employee-management-application/)

[**https://codecanyon.net/category/mobile/flutter?term=employee%20management**](https://codecanyon.net/category/mobile/flutter?term=employee%20management)

[**https://lnu.diva-portal.org/smash/get/diva2:204828/FULLTEXT01.pdf**](https://lnu.diva-portal.org/smash/get/diva2:204828/FULLTEXT01.pdf)