**Complex Engineering Problem**

**(CEP) Report**

**Project Title:**

**Attendance Marking App**

**Student Name: Ikram Nabi**

**Roll No: 22SW059**

**Section: 2**

**Date: [27-10-2025]**

*Submitted for the subject Mobile Application Development.*

1. **Introduction**
   1. **Project Overview**

This project addresses a simple, common problem in our classes: the manual, time-consuming **process of taking attendance** and sharing it. Before this app, **Class Representatives (CRs)** had to manually write down lists and then send separate WhatsApp messages to teachers—a process prone to delay and errors.

The **goal** was to create **a mobile app** that allows CR to manage student lists, **mark attendance with a single tap**, and instantly generate a formatted text summary ready to be shared via WhatsApp or any other platform.

* 1. **Proposed Solution**

The **Attendance Marking App** provides a smart, efficient, and user-friendly solution to manage student attendance digitally.  
The app allows CRs or teachers to:

* Add and save students locally using **Room Database (SQLite)**.
* Mark attendance with a single tap (Present/Absent Mark).
* Automatically generate and send attendance summaries via **WhatsApp** or any messaging platform.
* Save attendance for a specific date (daily tracking).
* Send only absentees or presentees if required.

This app has already been tested and used by our current class CR, who found it very helpful in managing daily attendance efficiently.

1. **Tools and Technologies Used**

The following core components, technologies, and libraries were used to build the application:

|  |  |
| --- | --- |
| Component | Technology / Library |
| Programming Language | **Kotlin** |
| Architecture | **MVVM (Model–View–ViewModel)** |
| Database | **Room Database (SQLite)** |
| Concurrency | **Kotlin Coroutines** |
| State Management | **Kotlin Flows** |
| UI Framework | **Jetpack Compose** |
| IDE | **Android Studio** |
| Platform | **Android** |

1. **Core Features and Architecture**
   1. **Key Features**

The app is built as a dedicated tool for **attendance management** and includes these main capabilities:

**Student Management:** You can easily add, view, and manage the list of students for your class. This data is stored locally using the **Room Database (SQLite)**, ensuring quick, reliable access even when offline.

1. **One-Tap Attendance:** Marking a student as **Present** or **Absent** is done with a quick tap.
2. **Instant Sharing:** The app generates a clean text summary of the attendance (you can choose to list only present or absent students) and uses Android's built-in sharing features to send it immediately.
3. **Add Students:** Add students with their names and roll numbers, stored locally in the Room database.
4. **Mark Attendance:** Quickly mark students present or absent with one tap.
5. **Select All Option:** Option to mark all students as present or absent at once.
6. **Save Attendance:** Store attendance by date for later review or editing.
   1. **Technical Standards:** The app follows **MVVM architecture** to ensure a clean separation of concerns and maintainability.

* **ViewModel:** Handles UI logic, exposes state flows, and manages data updates.
* **Repository:** Manages database operations using Room.
* **UI Layer:** Built with Jetpack Compose, observing UI states through Kotlin Flows.

1. **Data Storage:**

The Attendance Marking App requires a reliable, lightweight, and fast local storage solution to store student details and attendance records. For this purpose, the app uses Room Database, which is an abstraction layer built over SQLite — the default embedded database in Android.

Room was chosen for this project due to the following reasons:

* **Local Data Persistence:**  
  Since the attendance data needs to be accessible even without an internet connection, a local database was the best choice. Room ensures all data (like student names, roll numbers, and attendance status) is stored persistently on the device.
* **Lightweight and Efficient:**  
  SQLite is built into Android, which makes Room extremely lightweight and fast for small datasets like student lists or attendance records.
* **Type-Safety and Compile-Time Checks:**  
  Room provides compile-time verification of SQL queries, reducing the chances of runtime errors and improving code reliability.
* **Easy Integration with Kotlin Coroutines and Flows:**  
  Room supports suspend functions and Kotlin Flow natively, which perfectly fits the app’s reactive and asynchronous data-handling approach. This allows seamless observation of data changes in real time.
* **MVVM Friendly:**  
  Room integrates naturally with the MVVM architecture and Hilt Module, where the Repository layer communicates with Room to fetch or update data, and the ViewModel exposes it as state to the UI.
* **Data Stored in Room Database:** The app’s Room database contains the following entities:

**Student Entity:** Contains fields like id, name, and rollNumber.

**Attendance Entity:** Stores attendance details with fields like date, studentId, and status (Present/Absent).

The structure allows the app to save attendance records for each student on a particular date and retrieve them efficiently when required.

1. **Technical Implementation: State Management**
   1. **Concurrency and Data Flow**

The app uses modern Kotlin practices to handle data efficiently:

* **Kotlin Coroutines:** Used to manage all background tasks, specifically the slow Input/Output (I/O) operations like saving and loading data from the Room database. This keeps the main screen smooth and responsive.
* **Kotlin Flows:** Utilized for reactive State Management. Data from the database is streamed directly to the user interface via Flows, so the screen automatically updates the moment data changes.
  1. **Major Challenge: Fixing List State Collusion**

The most difficult technical challenge **was related to state management within** the student list.

**The Issue:** When I first implemented the attendance marking, especially the **'Select All'** feature**, toggling one student's status would often cause the entire list state to momentarily reset** **or incorrectly update other student states**. This made the 'Select All' feature unusable and broke single-tap marking.

**The Fix (Parental State Management):** I resolved this by implementing a Parental State Management pattern:

1. Instead of the entire student object being the state, the core ViewModel was refactored to hold only a key map (Map<StudentId, AttendanceStatus>).
2. The UI component for each student was then updated to only observe its own status from this map, while the list structure remained static.
3. This change ensured that updating one student's status only triggers a minimal, localized update, and the 'Select All' action now works reliably with one clean, atomic state change.

4. **Other Issues Encountered**

During development, a few minor issues were debugged and fixed:

1. **Data Race Condition:** Initially, attendance data would sometimes fail to save if the user immediately exited the app after marking. This was fixed by making sure all database-writing Coroutines are completed using the secure viewModelScope before the app shuts down.
2. **Screen Overlap:** The software keyboard would cover input fields on smaller phones (like the Add Student form). This was resolved by adjusting the Android Manifest file to resize the screen when the keyboard appears.
3. **Sharing Ambiguity:** The original share text was too brief. I updated the sharing logic to provide a comprehensive summary that includes total students, total present, and total absentsS, making the report clearer for teachers.
4. **Testing and Evaluation**

The app was tested in real classroom scenarios. Our class CR used the app for several days to mark and send attendance.  
Feedback showed that:

* The app **reduced attendance time by over 70%**.
* Attendance messages were more **accurate and standardized**.
* **No manual writing or counting** was needed anymore.

1. **Future Enhancements**

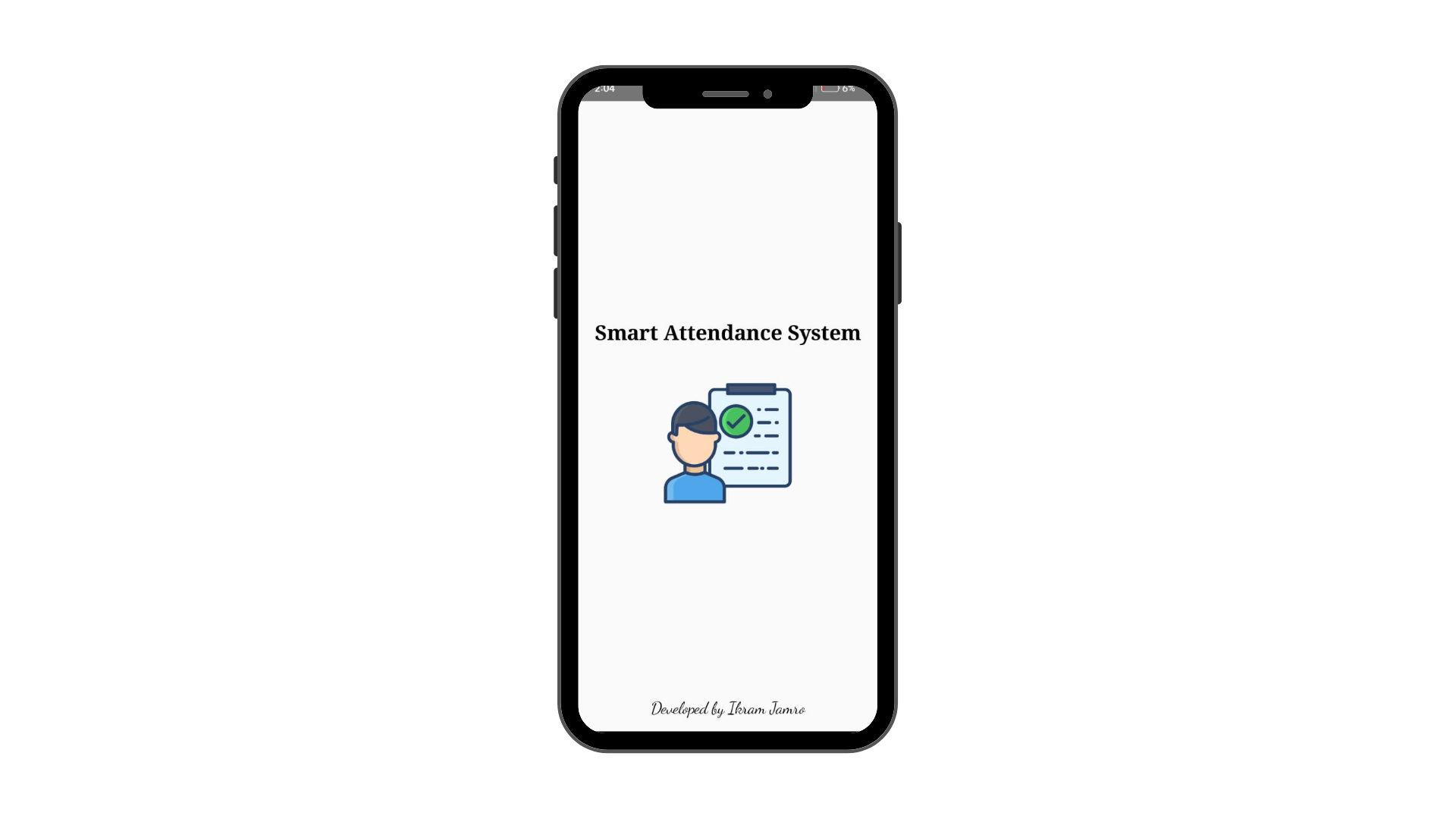
* Integration with **Firebase or Cloud Database** for syncing attendance across devices.
* Addition of **Excel/PDF export feature** for monthly attendance reports.
* **Teacher login module** for direct report access.
* **Dark mode and customizable themes**.
* **Push notifications** for attendance reminders.

1. **Conclusion**

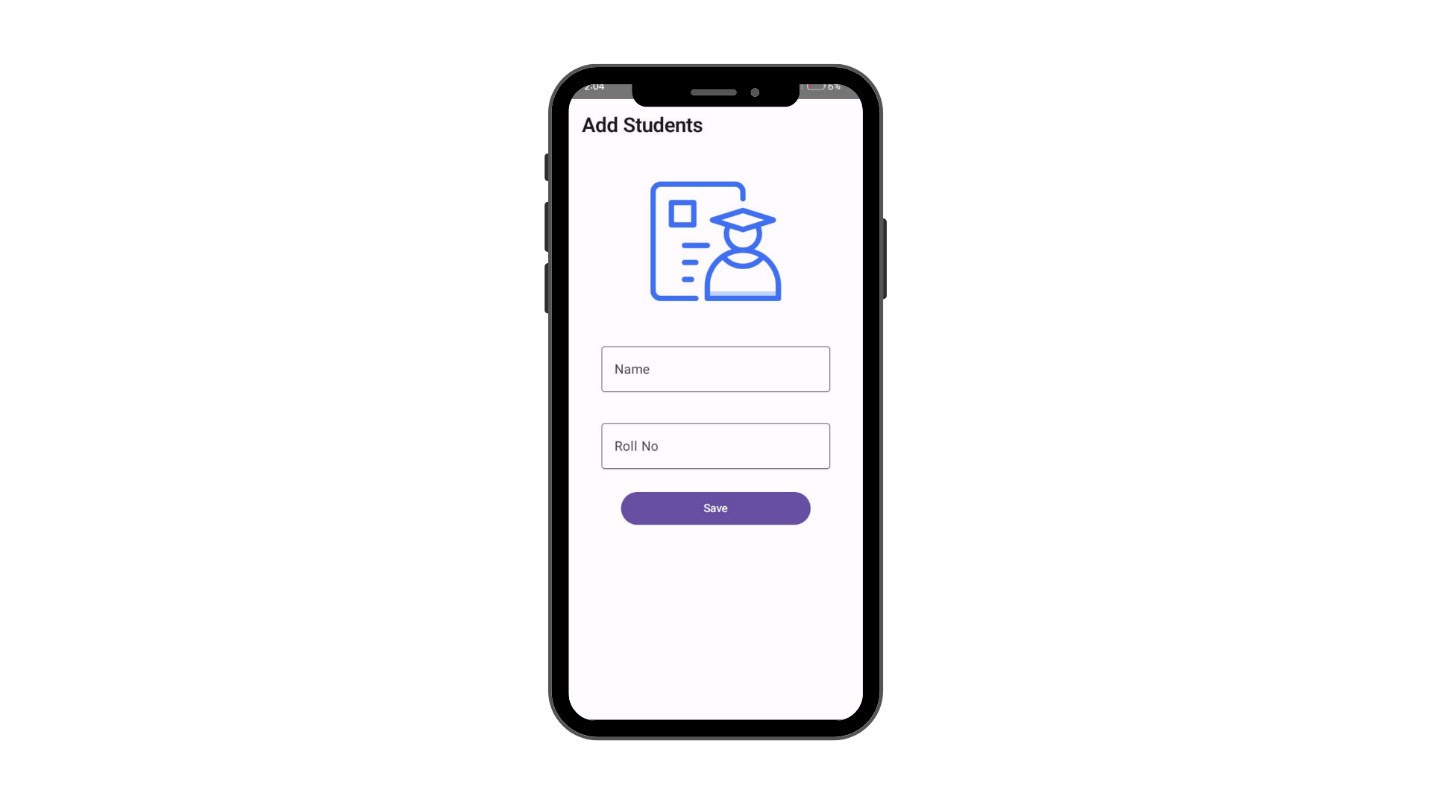
The **Attendance Marking App** successfully addresses the problem of manual attendance management. It simplifies the process, saves time, and ensures data accuracy.  
By using modern Android development standards like **MVVM**, **Room**, **Flows**, and **Coroutines**, the project demonstrates practical application of learned concepts and provides a useful real-world tool.

**—Application Screenshots—**

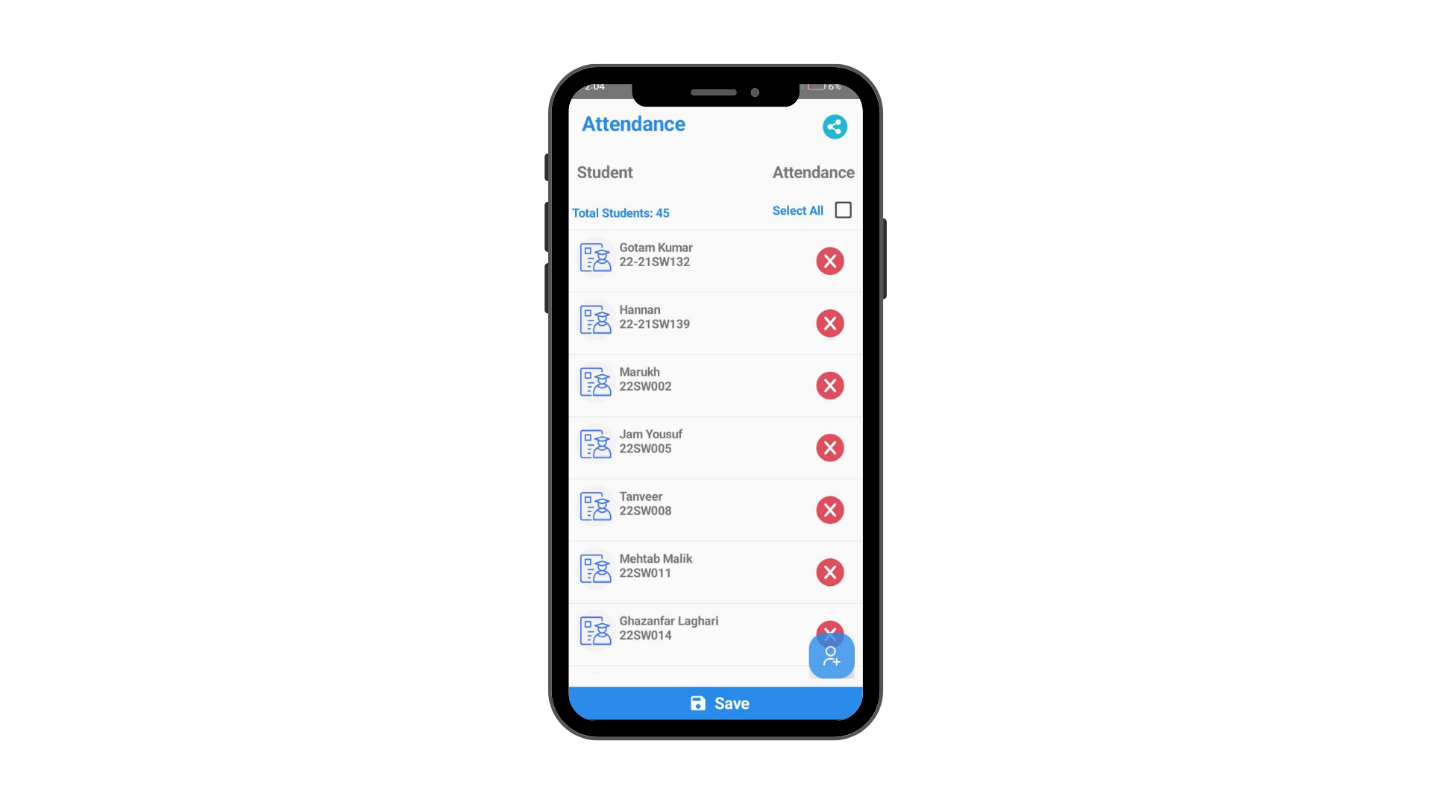
* 1. **Splash Screen**

****

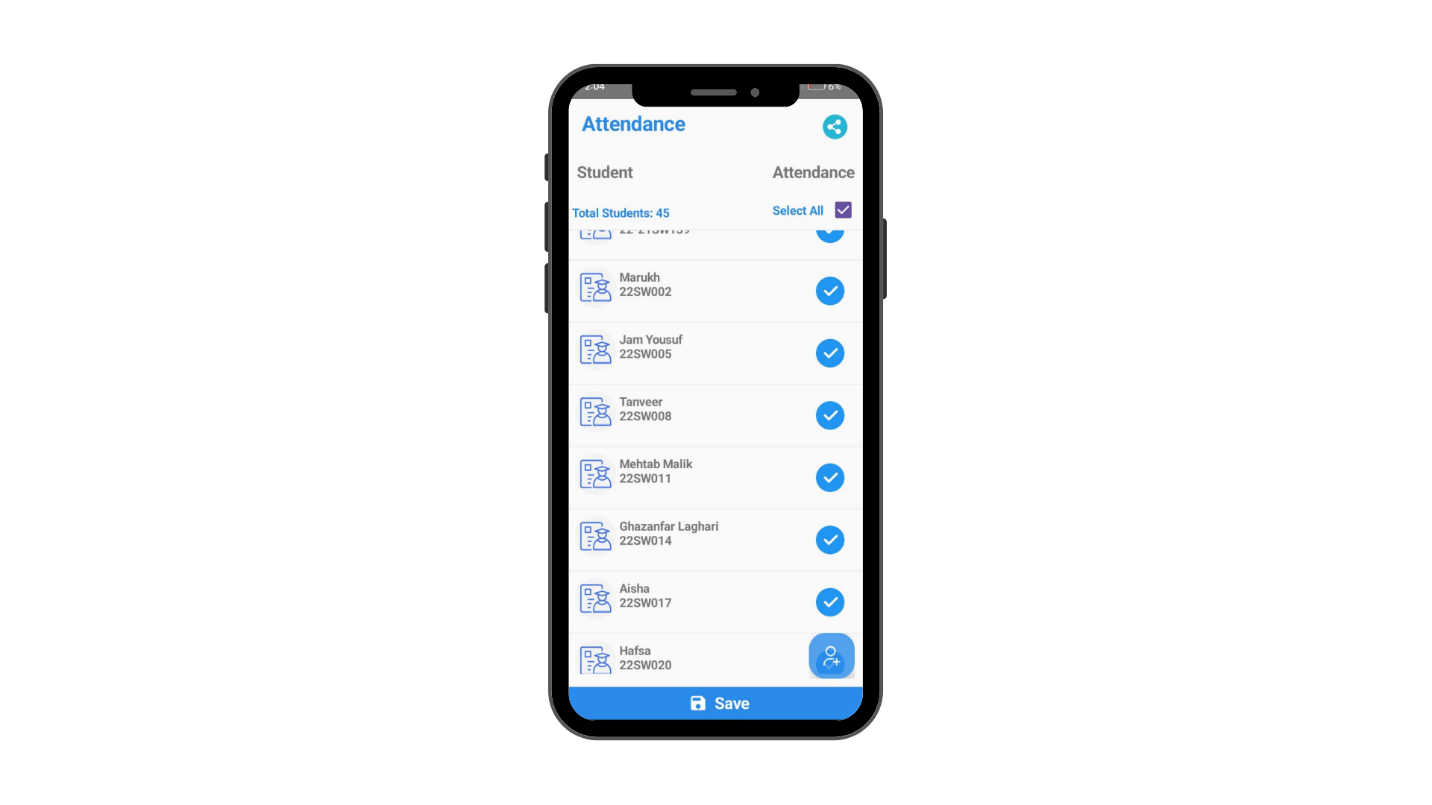
* 1. **Student Add Screen (SQL DATABASE)**

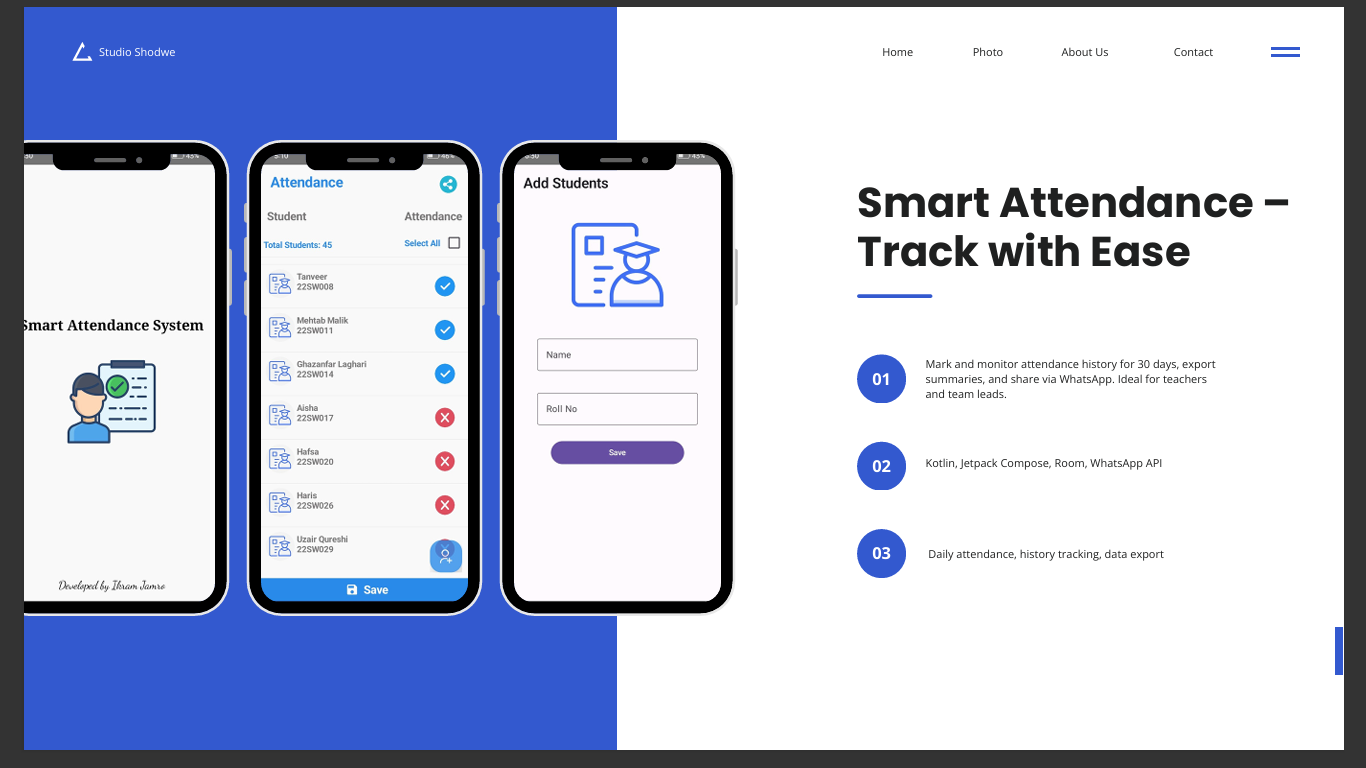
****

* 1. **Home Screen**



* 1. **All Marked**





**Note: Screen Shots only on mobile devices because emulators don’t work on my pc and in android there is no other option to run on other interfaces or platforms.**