Vocational Training Authority of Sri Lanka

District Center – Inamaluwa

Software Requirements Specification (SRS) for

Contact Mobile App

Android App Development

Table of Content

INTRODUCTION

Purpose Of This Document

This document outlines the software requirements specification for the "Contact" mobile application. The application allows users to add, update, and delete contacts with associated images, names, and mobile numbers. This document serves as a guide for developers, testers, and stakeholders.

Scope Of This Document

The document covers all functional and non-functional requirements, design considerations, and system specifications. It also includes diagrams and appendices for further clarification.

SOFTWARE LIFE CYCLE MODELS

Methodology used

The chosen software development life cycle (SDLC) model for this application is **Waterfall**. This model ensures clear phases and structured progress for the development process.

Week 1-2: Project Initiation and Planning

Define Project Scope and Objectives:

- Identify the functionalities and features required in the Contact app.
- Define the goals and objectives of the system.

Stakeholder Analysis:

- Identify and list all stakeholders (e.g., app users, developers).
- Gather requirements and expectations from each stakeholder group.

Risk Analysis:

- Identify potential risks associated with the project and its development.
- Develop a risk management plan.

Week 3-4: Requirements Gathering

Functional Requirement Documentation:

- Document detailed functional requirements such as add, update, delete, and image upload functionality.

Non-Functional Requirement Documentation:

- Identify and document performance, reliability, and maintainability goals.

Use Case Scenarios:

- Create use case diagrams and scenarios for each functionality.

Week 5-6: Design Phase

User Interface Design:

- Create mockups for screens such as the add contact, edit contact, and delete contact screens.

Database Schema Design:

- Plan and design the SQLite database schema for contact storage.

Week 7-10: Development Phase

Frontend Development:

- Implement UI using Kotlin for native Android development.

Database Integration:

- Integrate SQLite for contact data management.

Feature Implementation:

- Code the add, update, delete, and image upload functionalities.

Week 11-12: Testing Phase

Unit Testing:

- Test individual components for correctness.

Integration Testing:

- Test the interaction between the SQLite database and the frontend.

Usability Testing:

- Ensure the application is user-friendly and meets user requirements.

PROBLEM STATEMENT

In today’s fast-paced world, managing personal and professional contacts efficiently is an essential part of everyday life. With the increasing number of mobile applications available for contact management, many users struggle to find an intuitive and user-friendly solution that caters to their needs.

While many existing contact management applications offer basic features like adding, editing, and deleting contacts, they often come with a cluttered interface, unnecessary complexity, or fail to offer specific features that would enhance user experience. Additionally, some apps do not provide an easy way to attach images to contacts, a feature many users find crucial for visual identification. Furthermore, there are applications that depend on cloud services, creating concerns about data privacy and security, while others lack offline functionality, rendering them ineffective in areas with limited internet connectivity.

The lack of a lightweight, offline-capable contact management solution leads to several issues:

1. **Cluttered User Interfaces:** Many apps present a complex and overburdened user interface with an overwhelming number of options and settings, which creates confusion and hinders ease of use. The user interface is often not optimized for a seamless and intuitive experience, leading to unnecessary complexity.
2. **Inefficient Image Management:** Most existing contact management apps do not allow users to associate images with contacts or have cumbersome methods for adding them. Users want a quick, simple way to upload and store images with contacts for visual identification, but current solutions may either lack this feature or make it difficult to manage these images effectively.
3. **Limited Offline Functionality:** Cloud-based contact management solutions often require a stable internet connection to function correctly. This is not feasible in all

environments, particularly in regions with unreliable or no internet access. As a result, users are unable to access or update their contacts when they are offline.

4. **Lack of Data Privacy:** Many contact management apps store user data on external servers or in the cloud, raising concerns over privacy and security. Users may not feel comfortable uploading their personal contact information and associated images to third-party servers due to the potential risks of data breaches and unauthorized access.
5. **Performance Issues:** Some apps can be sluggish, especially on older or less powerful devices. This can be frustrating for users, particularly when managing large contact lists or uploading images. The performance and responsiveness of the app can be hindered by inefficient coding practices or heavy reliance on external cloud services.

Objective:

The **Contact** mobile application aims to address these issues by offering a simple, efficient, and offline-capable solution for managing contacts on Android devices. The app will be designed with a user-friendly interface that makes it easy to add, update, delete, and view contacts. Users will be able to associate images with each contact, ensuring that visual identification is both easy and intuitive. Furthermore, the app will function entirely offline, storing all data locally on the device to ensure that users have full control over their information and can access their contacts at any time, even without an internet connection.

By focusing on a clean and intuitive interface, lightweight performance, and offline functionality, the **Contact** mobile application will help users organize their contacts effectively without unnecessary complications or privacy concerns. This app will provide users with a straightforward and reliable solution to manage their personal and professional contact information with the added benefit of attaching images to contacts for better identification.

SOFTWARE DESIGN DOCUMENT

1. Purpose:

To provide a clear structure for the design and development of the Contact application.

2. Scope:

The application will be available for Android platforms only. Users can perform CRUD (Create, Read, Update, Delete) operations on their contact list.

3. System Overview:

- **Frontend:** Developed using Kotlin for native Android development.
- **Database:** SQLite for local storage.

4. Module Decomposition:

Module Name	Description
User Interface	Provides screens for user interactions.
Data Handling	Manages CRUD operations for contact data.
Media Handling	Manages image uploads and retrieval.

5. Concurrent Process Decomposition:

- Image selection runs concurrently with data entry.
- Local data storage with SQLite ensures seamless offline access.

6. Data Decomposition:

Data Entity	Attributes
Contact	Name, Mobile Number, Image

Overall Description:

The **Contact** mobile application is designed to provide a simple, user-friendly interface for managing personal contacts on Android devices. The app aims to allow users to perform common contact management tasks such as adding, updating, and deleting contacts, while offering a seamless experience for storing associated information like names, mobile numbers, and images. The app is built using native Android development technologies and relies on an SQLite local database for efficient and offline contact management.

Key Features and Functionalities

1. **User Interface (UI)**
   The UI is designed to be intuitive, with easy navigation between key screens: the contact list, individual contact details, and the screens for adding, editing, and deleting contacts. Key UI features include:
   - **Contact List View:** A scrollable list displaying all stored contacts with

essential information such as the contact name and mobile number.

- **Add Contact:** A screen for adding new contacts, allowing users to input the contact's name, mobile number, and upload an image from the device's gallery or camera.
- **Edit Contact:** A screen for editing the details of existing contacts, with similar fields to the add contact screen and a simple action for deleting contacts from the list, with a confirmation prompt to avoid accidental deletions.

2. **Data Storage and Management**

   The app utilizes an **SQLite database** for storing contact data locally on the device. The SQLite database is chosen for its lightweight, efficient handling of relational data, enabling the app to function offline without relying on network connectivity. The data model includes:

   - **Contacts Table:** A primary table containing contact details, including columns for name, phone number, and the path to the image.
   - **Image Storage:** Each contact can have an associated image stored locally. The images are stored in a separate folder in the app's internal storage, and their file paths are linked to the respective contact in the database.

3. **CRUD Operations**
   - **Create (Add Contact):** Users can add new contacts, with a full set of details including a name, mobile number, and an image.
   - **Read (View Contact):** Users can view detailed information about a contact by clicking on it in the contact list. This includes the contact's name, phone number, and associated image.
   - **Update (Edit Contact):** Users can edit the details of an existing contact. The changes are updated in the SQLite database, and the associated

image can also be replaced.
   - **Delete (Remove Contact):** Users can delete a contact from the database.

4. **Image Handling**
   The **Media Handling** module enables the user to upload and manage images for contacts. When a user adds or edits a contact, they can choose to attach an image either from their gallery. The images are stored in the app's local file system, and the file path is stored in the SQLite database. The app will also handle basic image manipulation, such as auto scaling the image for optimized storage and display.

5. **Offline Functionality**
   Since the application is designed to function offline, all data is stored locally in the SQLite database, and no network connection is required to perform CRUD operations. This ensures that the app can be used in any environment without the need for an internet connection, providing a responsive and fast user experience.

6. **Data Integrity and Consistency**
   The application ensures that the data in the database is consistent and intact by using SQLite's built-in ACID (Atomicity, Consistency, Isolation, Durability) properties. This ensures that even in the event of an unexpected app shutdown or crash, data corruption is minimized, and changes are reliably committed to the database.

7. **Performance Considerations**

   The app is designed to operate efficiently with minimal resource consumption. The following techniques are implemented to ensure high performance:

   - **Efficient Database Queries:** Only the necessary fields are queried when displaying contacts, reducing memory usage and improving load times.
   - **Image Optimization:** Uploaded images are compressed and resized before being stored to minimize file size and storage space usage.
   - **Caching Mechanisms:** Frequently accessed data, like the contact list, is cached to reduce the need for repeated database queries, improving response times.

OTHER NON-FUNCTIONAL ATTRIBUTES

Reliability:

- The app will ensure data consistency and integrity within the SQLite database.
- Reliable operation during CRUD processes.

Maintainability:

- Code will follow modular design principles.
- Comprehensive documentation will be provided.

Portability:

- The app will run on Android devices with API 21 and above.

Extensibility:

- Features like contact grouping may be added in future updates.

Reusability:

- Common components, such as form validations, will be reusable across screens.

Application Affinity/Compatibility:

- Compatible with native device features like the gallery and file manager.

Flexibility:

- The app design will support additional fields like email or address in the future.

System Specifications

The **Contact** mobile application is designed to be lightweight and efficient, ensuring it performs well across a wide range of Android devices. The system specifications outline the hardware, software, and environmental requirements needed for the app to function smoothly and reliably.

1. Hardware Requirements

- **Processor:**
  - Minimum: ARM Cortex-A7 or equivalent
  - Recommended: ARM Cortex-A53 or higher (quad-core or higher for optimal performance)
  - The processor must support basic Android operations and handle SQLite database queries efficiently.
- **Memory (RAM):**
  - Minimum: 1 GB of RAM
  - Recommended: 2 GB or more
  - The app is designed to run with low memory usage, but having at least 1 GB of RAM will ensure smooth performance, especially when managing a large contact list.
- **Storage Space:**
  - Minimum: 10 MB of free storage space
  - Recommended: 20 MB or more
  - The storage requirement is minimal, primarily for storing contact data and images. The storage space will be used by the SQLite database, app data, and cached images. Users with devices having higher storage capacity will have a smoother experience, especially when storing images.

- **Display:**
  - Minimum: 4.0-inch screen size
  - Recommended: 5.0-inch screen size or larger
  - The app is optimized for small and medium-sized Android screens, but larger screens provide a better user experience for viewing contacts and images.
  - The app is responsive, with layouts that adapt to different screen sizes and resolutions, ensuring it works well on smartphones and tablets.
- **Gallery:**
  - The device should have access to a functional gallery (file manager) for image selection. The gallery will be used to select images from the device's storage, which will then be associated with contacts.

2. Software Requirements

- **Operating System:**
  - Minimum: Android 5.0 Lollipop (API Level 21)
  - Recommended: Android 8.0 Oreo (API Level 26) or higher
  - The app is compatible with Android 5.0 and above to ensure it can run on a wide range of Android devices. Devices with Android 8.0 or newer will benefit from enhanced performance and security features.
- **Programming Language:**
  - **Kotlin:** The primary programming language for the app's development. Kotlin is fully supported by Google for Android development, providing a modern, concise, and efficient syntax for building Android apps.
- **Database:**
  - **SQLite:** The app uses SQLite for local data storage. SQLite is a lightweight, file-based relational database that is ideal for mobile apps with offline functionality. It is integrated natively into Android, providing a fast,

efficient, and reliable database solution.
    - The database stores contact information, including names, mobile numbers, and image paths.

- **Development Tools:**
    - **Android Studio:** The official IDE for Android development, used to build, test, and debug the app. Android Studio includes tools for designing UI layouts, managing resources, and compiling the app into APK files.
    - **Gradle:** The build system used by Android Studio to automate tasks like compiling code, managing dependencies, and building APKs.

3. Network Requirements

- **Network Connectivity:**
    - The app does not require an active network connection to perform its core functionalities, as all data is stored locally in the SQLite database.
    - However, if future features such as cloud backup or sync are added, a network connection (Wi-Fi or mobile data) will be required to perform these operations.
- **No Cloud Storage:**
    - The application does not rely on any online or cloud-based storage services, ensuring it works in offline mode. Users' contacts and images are stored locally on the device.

4. Performance and Load Considerations

- **Database Performance:**
    - SQLite queries should be optimized for quick retrieval and modification of contact data. Indexing may be applied on columns frequently queried (e.g., `name`, `phone number`) to enhance performance.
    - As the app is designed to be lightweight, database operations are performed on a separate thread to prevent blocking the UI thread and causing UI lag.

- **Image Optimization:**
  - Images are resized before being stored in the device's storage to reduce file sizes. Image compression will be applied during the upload process to save storage space and improve performance.

5. Security Considerations

- **Data Storage Security:**
  - The SQLite database can be encrypted to protect sensitive contact information. While the app does not use cloud storage or require user authentication, sensitive data (e.g., phone numbers) should still be secured locally.
- **Permissions:**
  - The app will request the necessary permissions for accessing the gallery (storage) and reading/writing to storage.
  - These permissions will be handled at runtime to ensure privacy and security compliance, especially for newer versions of Android (Android 6.0 and above).

6. Usability Requirements

- **User Interface:**
  - The app will provide a simple, intuitive, and responsive UI with clear navigation between the main features: adding, editing, deleting, and viewing contacts.
  - The UI should adhere to Material Design principles to ensure consistency with the Android platform and improve user experience.

7. Backup and Restore (Future Enhancement)

- **Backup Feature (Optional):**
  - Users may be able to back up their contacts to cloud storage (e.g., Google

Drive) in future versions of the app, allowing for seamless restoration of contacts if the app is reinstalled or if users switch devices.

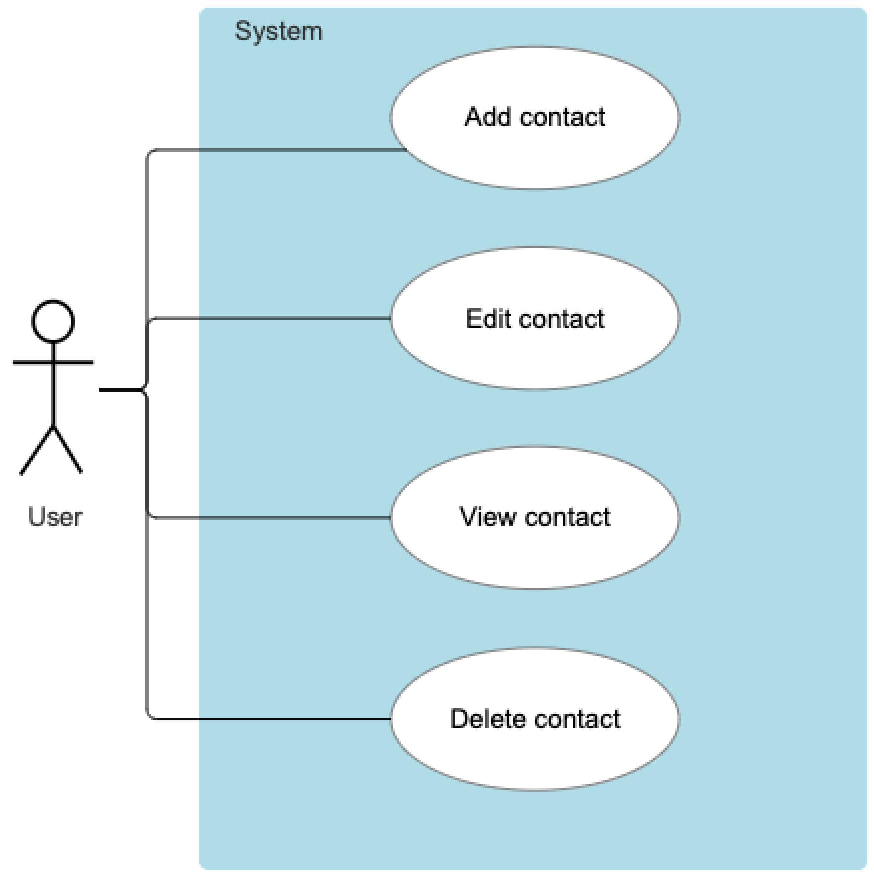
- **Data Synchronization (Future Enhancement):**
  - Future updates may allow synchronization of contacts across multiple devices, providing users with the ability to manage contacts from different platforms or devices.

8. Extensibility and Future Updates

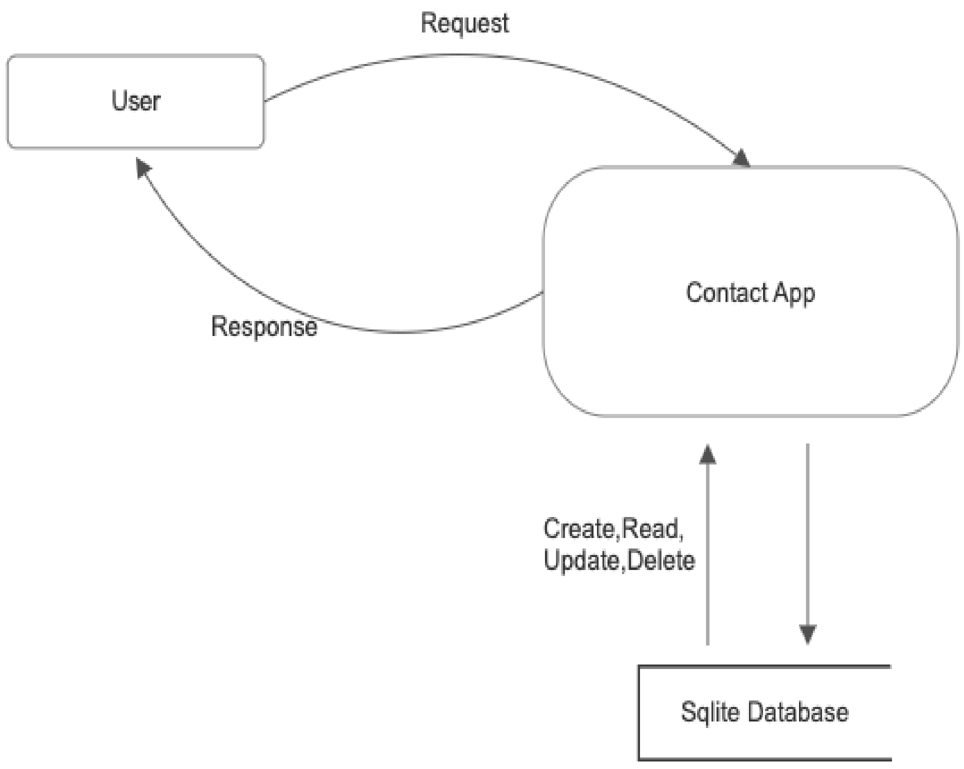
- **Feature Expansion:**
  - Future versions of the app may support additional fields for contacts, such as email addresses, home addresses, and social media profiles.
  - A contact grouping feature could allow users to categorize their contacts (e.g., Family, Work, Friends) for easier management.
- **Integration with Other Apps:**
  - In the future, the app could integrate with other native Android applications (e.g., messaging apps, email apps) for enhanced functionality, such as making calls directly from the contact list.

DIAGRAMS

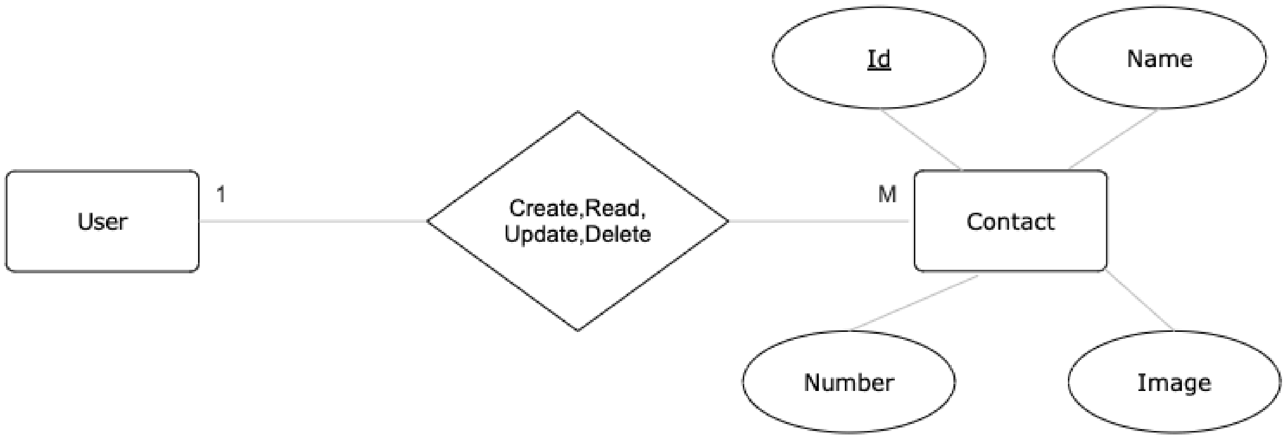
a) Use Case Diagram



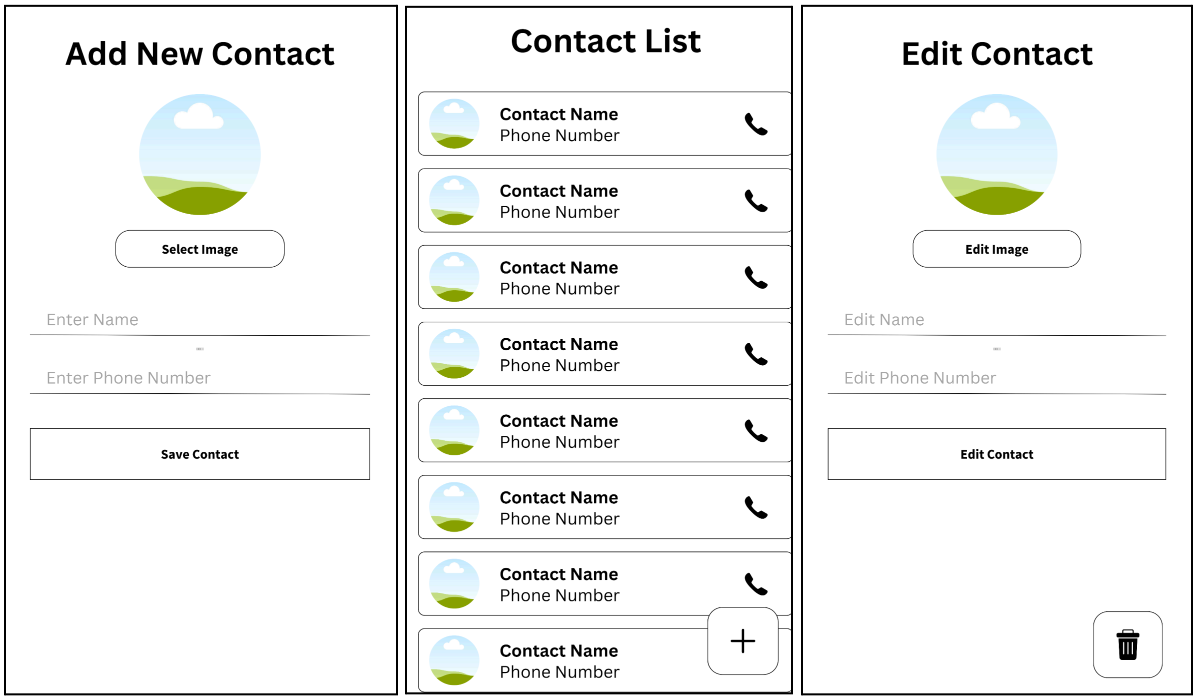
b) Data Flow Diagram



c) E-R Diagram


Id
Name
User
1
Create,Read,
Update,Delete
M
Contact
Number
Image

OUTPUT SCREENS



APPENDICES

Definitions:

- **CRUD:** Create, Read, Update, Delete operations.
- **SDLC:** Software Development Life Cycle.

References:

- [SQLite Documentation]
- [Kotlin Official Guide]