Software Design Specification For Memora

Version 1.0

Prepared by

Sahil Kumar 2112232

Pershant Parkash 2112229

Faculty Of Computing and Engineering Sciences
Shaheed Zulfiqar Ali Bhutto
Institute Of Science and Technology
Karachi

Jan 26, 2025

Final Year Project

Table Of Content

Table Of Figures

Table Of Tables

1. Introduction

1.1 Purpose of this document

The purpose of this Software Design Specification (SDS) is to provide a comprehensive and detailed design framework for Memora, a digital time capsule application. This document elaborates on system architecture, components, user interface design, and data handling to ensure clarity and alignment among developers, testers, and stakeholders. It aims to guide the implementation of Memora's features, including secure time capsule creation, sharing, and milestone-based unlocking.

1.2 Scope of the development project

Memora is designed as a digital memory preservation platform where users can create, share, and unlock time capsules containing media and text. The app ensures secure data storage and retrieval while enhancing the user experience with features such as memory mapping, nested capsules, and emotional connections. It supports cross-platform deployment (Android and iOS) and integrates external tools like Google Maps and cloud storage solutions.

1.3 Definitions, acronyms, and abbreviations

- **Time Capsule**: A digital container for storing media, text, and memories, locked until a specified date.
- **Nested Capsules**: Sub-capsules grouped within a main capsule, unlocking on the same date.
- **Memory Calendar**: A feature highlighting upcoming capsule unlock dates.
- **API**: Application Programming Interface used for server and client communication.
- **UI**: User Interface, the graphical interface through which users interact with the application.

1.4 References

NA

1.5 Overview of document

This document details the system architecture, component design, user interface, reuse strategies, and key decisions for Memora. It concludes with pseudocode and supporting diagrams.

2. System architecture description

2.1 Section Overview

This section describes the architecture and design choices of Memora, focusing on data flow, system structure, and the rationale behind design decisions. The system architecture is designed to ensure scalability, security, and seamless user experience for creating, storing, and accessing digital time capsules.

2.2 General Constraints

The development and deployment of Memora are subject to the following constraints:

- **Operating Systems**: Must support Android 8.0+ and iOS 12.0+ to cater to a wide range of devices.
- **Hardware Requirements**: Devices must have at least 2GB of RAM and support for camera and GPS functionality.
- **Security Compliance**: The application must adhere to data privacy regulations, including GDPR and local laws.
- **Network Dependency**: Requires stable internet connectivity for capsule synchronization, map services, and notifications.
- **Integration Limits**: Relies on external APIs like Google Maps for location services and

2.3 Data Design

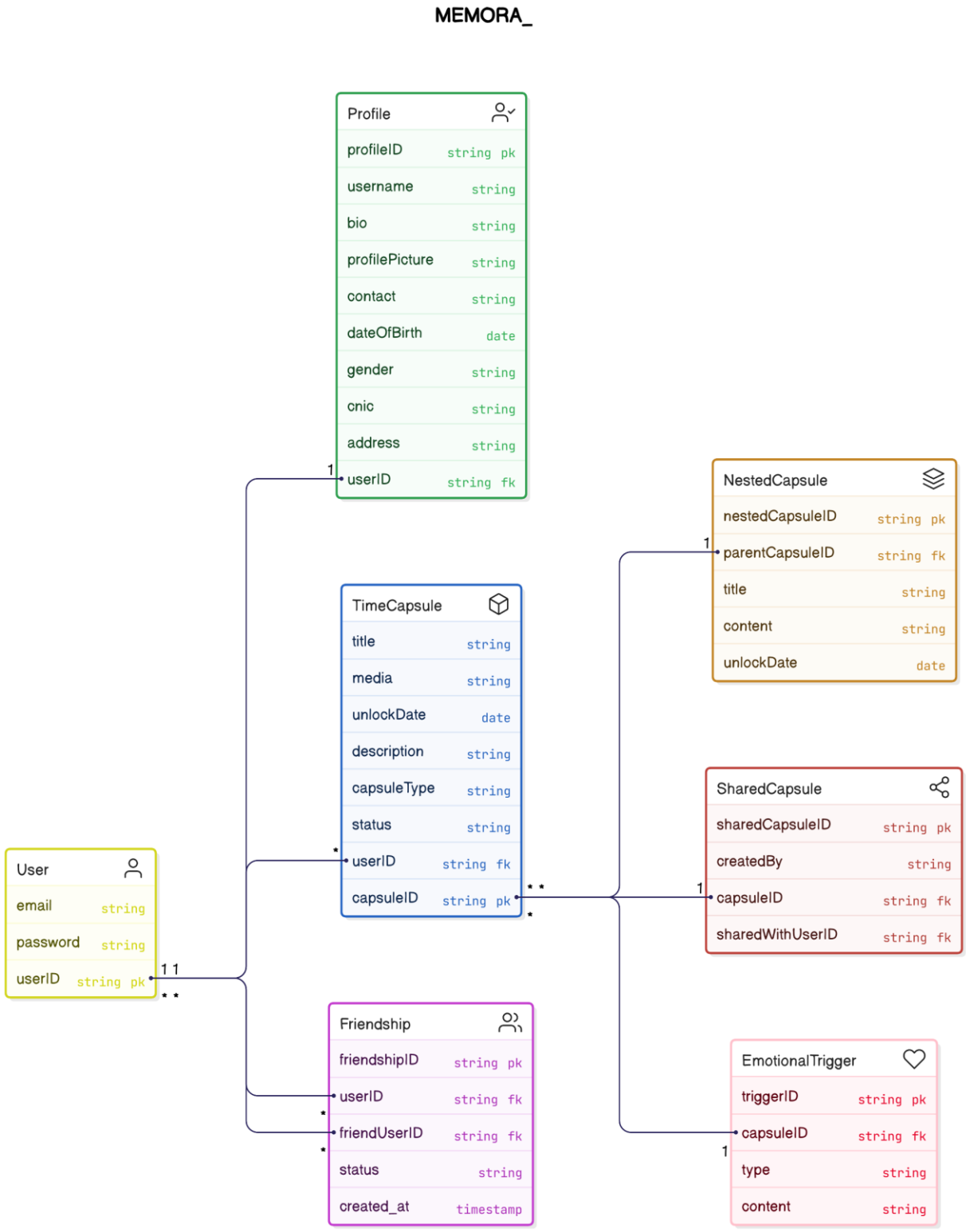

Figure 1:ERD

2.4 Program Structure

Memora uses a client-server architecture for seamless communication and functionality. The structure is as follows:

- **Frontend**: Developed in React Native to provide a cross-platform mobile experience. Handles user interactions and visual components.
- **Backend**: Built using Node.js, managing business logic, API endpoints, and database interactions.
- **Database**: MongoDB with Mongoose ORM for managing user and capsule data efficiently.
- **APIs**: RESTful APIs for communication between the frontend and backend.
- **External Integrations**:
- Google Maps API for memory locations.

2.5 AlternativesConsidered

- **Relational Database (e.g., MySQL)**: Rejected due to lack of flexibility for handling nested capsules and complex data relationships.
- **Flutter vs. React Native**: React Native was chosen for its large community support and ease of integration with existing libraries.
- **Custom Map Implementation**: Rejected in favor of Google Maps API for its reliability, feature set, and ease of integration.

3.Detailed description of components

3.1 SectionOverview

This section provides an in-depth description of Memora's key components. Each component is detailed in terms of its purpose, data structures, methods, and interactions with other components.

3.2 Component n Detail (include a sub-section for each component)

For each major feature in Memora, the components are outlined as follows:

3.2.1 Login, Logout, and User Registration

Description: Handles user authentication and account creation processes for the Memora platform.
Data Members:

- **Include Type:** User credentials (username, password, email).
- **Visibility:** Platform-wide access.
- **Methods:** Authenticate users, create accounts, log out.

Type	View & Controller.
Purpose	Enables secure user authentication and profile access.

Function	Validates user input, manages authentication, and session handling.
Subordinates	Handles session tracking, authentication, and error handling.
Dependencies	Requires internet, email verification, and password validation.
Interfaces	Connects with authentication services and user database.
Resources	Uses a database to store user credentials and tokens.
Processing	User submits credentials; system verifies, grants/denies access, manages sessions.
Data	Stores email, hashed passwords, profile data, and session tokens.

Table 1:User Registration Component

3.2.2 Profile Management

Description: Allows users to view, edit, and update their personal profiles on Memora.

Data Members:

- **Include Type:** User profile details (name, bio, profile picture, etc.).
- **Visibility:** Accessible by the user.
- **Methods:** Fetch, update, and manage profile details.

Identification	Profile Management Module
Type	Backend & UI Module
Purpose	Enables users to manage and customize their profiles.
Function	Provides functionalities for updating and viewing profile information.
Subordinates	Database modules and UI components.
Dependencies	Requires authentication and stable database connection.
Interfaces	Profile API and user dashboard interface.
Resources	Database storage and image upload services.
Processing	Updates profile data and synchronizes with the database.
Data	Stores profile details, including user images and metadata.

Table 2: Profile Component

3.2.3 Friend Requests and Community Groups

Description: Manages social connections through friend requests and group interactions.

Data Members:

- **Include Type:** Friend request statuses, group membership data.
- **Visibility:** Shared between users.
- **Methods:** Send/accept friend requests, create/join groups.

Identification	Social Interaction Module
Type	Backend and Frontend Module
Purpose	Facilitate user connections and community engagement.
Function	Manage friend requests, group invitations, and community activities.
Subordinates	Group and friend request management components.
Dependencies	Database access and notification system.
Interfaces	APIs for friend requests and group management.
Resources	Relational databases for user connections and group data.

Processing	Handles social interactions and synchronizes updates.
Data	Stores friend request and group statuses.

Table 3:Friend Requests and Community Groups

3.2.4 Time Capsule Creation

Description: Allows users to create digital time capsules with media and a future unlock date.

Data Members:

- **Include Type:** Capsule title, description, media files, unlock date.
- **Visibility:** Private; shared upon unlocking.
- **Methods:** Create, validate, save, retrieve.

Identification	Time Capsule Creation Module
Type	Backend & UI Module
Purpose	Enables users to store and schedule memories.
Function	Manages capsule creation, media uploads, and unlock scheduling.
Subordinates	Database module, media processing, authentication.
Dependencies	Requires authentication, storage, media validation.
Interfaces	Connects with storage API, media processing, and dashboard.
Resources	Uses database for capsule details and media storage.
Processing	Validates inputs, saves capsules, and schedules unlock.
Data	Stores metadata, media content, and unlock timestamps.

Table 4:Time Capsule Creation Component

3.2.5 Shared Capsules

Description: Allows users to share their time capsules with selected friends or family members, granting them access after the unlock date for collaborative or shared experiences.

Data Members:

- **Include Type**: User and recipient metadata.
- **Visibility**: User-specific.
- **Methods**: Share capsule with selected recipients, notify recipients, handle errors for invalid contacts.

Identification	Shared Capsule Module
Type	Backend Module
Purpose	Enable users to share capsules with friends or family.
Function	Share a time capsule with selected recipients, allowing access after unlock.
Subordinates	User management, notification system.
Dependencies	Relies on recipient validation and user contact management.
Interfaces	Capsule sharing and user management APIs.
Resources	Database for storing shared capsule information and recipient data.
Processing	Validates recipients, shares the capsule, and notifies recipients.
Data	Contains recipient and sharing metadata, including status of

	invitation.

Table 5:Shared Capsules Component

3.2.6 Media Upload/Create

Description: Allows users to create and upload various types of media, such as images, videos, and audio, to enrich their time capsules and preserve diverse memories.

Data Members:

- **Include Type**: Media file metadata (images, videos, audio).
- **Visibility**: User-specific.
- **Methods**: Create,Upload, validate, and save media to the capsule.

Identification	Media Upload/Create Module
Type	Backend Module
Purpose	Facilitate media creates and uploads into time capsules.
Function	save media (images, videos, audio) to capsules.
Subordinates	Media validation components (format and size check).
Dependencies	Relies on file validation for format, size, and secure storage.
Interfaces	Media capsule management APIs.
Resources	Database for storing media files .
Processing	Validates media files, ensuring proper format and size.
Data	Contains media file storage location.

Table 6:Media Upload/Create Component

3.2.7 Memory Capsules on Map

Description: Displays memory capsules on an interactive map, allowing users to explore location-based memories.

Data Members:

- **Include Type:** Capsule title, location data, unlock date.
- **Visibility:** Private or shared based on user settings.
- **Methods:** Load map, display capsules, retrieve capsule details.

Identification	Memory Capsules on Map Module
Type	UI & Backend Module
Purpose	Visualizes memory capsules geographically.
Function	Displays capsule locations, retrieves and presents details.
Subordinates	Map service, database, authentication.
Dependencies	Requires map API, geolocation data, and capsule metadata.
Interfaces	Connects with map services, capsule database, and user dashboard.
Resources	Uses geolocation API and database for capsule data.
Processing	Loads map, fetches locations, and displays capsule details.
Data	Stores coordinates, capsule metadata, and unlock timestamps.

Table 7 :Memory Capsules on Map Component

3.2.8 Memory Calendar

Description: Provides a calendar view to organize and view memory capsules by date.
Data Members:

- **Include Type:** Capsule dates and metadata.
- **Visibility:** User-specific.
- **Methods:** Fetch and display capsules on the calendar.

Identification	Calendar Module
Type	Frontend Module
Purpose	Allow users to organize capsules by date.
Function	Display capsules in a calendar interface.
Subordinates	Data fetching and rendering components.
Dependencies	Capsule metadata and user preferences.
Interfaces	Calendar UI integrated with capsule data.
Resources	JavaScript calendar libraries.
Processing	Synchronizes capsule data with the calendar view.
Data	Contains capsule metadata and associated dates.

Table 8:Memory Calendar Component

3.2.8 Nested Capsules

Description: Allows users to create capsules within other capsules for better organization.
Data Members:

- **Include Type:** Parent and child capsule metadata.
- **Visibility:** User-specific.
- **Methods:** Create and link nested capsules.

Identification	Nested Capsule Module
Type	Backend Module
Purpose	Organize capsules hierarchically.
Function	Link nested capsules to main capsules.
Subordinates	Capsule hierarchy management components.
Dependencies	Database relationships for parent-child mapping.
Interfaces	Capsule management APIs.
Resources	Database for storing relationships.
Processing	Maintain hierarchical consistency between capsules.
Data	Stores parent-child capsule relationships.

Table 9:Nested Capsules Component

3.2.9 Unlock Milestone Capsules

Description: Automatically unlocks capsules when specified milestones are reached.
Data Members:

- **Include Type:** Milestone criteria and capsule data.
- **Visibility:** User-defined.

- **Methods:** Check milestones, unlock capsules.

Identification	Milestone Module
Type	Backend Module
Purpose	Automatically manage milestone-based capsule unlocking.
Function	Track milestones and unlock capsules.
Subordinates	Milestone tracking and notification components.
Dependencies	Relies on milestone tracking data.
Interfaces	APIs for milestone tracking and capsule unlocking.
Resources	Milestone data and notification services.
Processing	Checks milestone criteria and unlocks capsules when met.
Data	Contains milestone conditions and unlock statuses.

Table 10:Unlock Milestone Capsules Component

3.2.10 Emotional Connection

Description: Enables users to add emotional content (text or audio) to time capsules, accessible before the official unlock to build anticipation.
Data Members:

- **Include Type:** Emotional text messages, recorded/uploaded audio.
- **Visibility:** Available to capsule participants before unlock.
- **Methods:** Add emotional content, validate inputs, store and retrieve data.

Identification	Emotional Connection in Capsule Module
Type	Backend & UI Module
Purpose	Allows users to add emotional teasers to time capsules.
Function	Enables adding and retrieving pre-unlock emotional content.
Subordinates	Database, media storage, UI components.
Dependencies	Requires time capsule existence, media storage, and user authentication.
Interfaces	Connects with capsule database and user dashboard.
Resources	Uses media storage for audio, database for text.
Processing	Validates, stores, and associates emotional content with capsules.
Data	Stores text messages, audio files, and metadata.

Table 11:Emotional Connection Component

4.User Interface Design

4.1 SectionOverview

The User Interface (UI) of Memora is designed to ensure a seamless and accessible experience for all users, including families and individuals who wish to preserve and share memories. The design emphasizes simplicity, clarity, and ease of navigation, particularly for non-technical users. Features like large buttons, intuitive workflows, and high-contrast color schemes are integral to the design philosophy.

4.2 Interface Design Rules

Memora's interface design follows these key principles:

- **Consistency**: Uniform layouts and styles are applied across all screens to create a cohesive user experience.
- **Accessibility**: Features include large buttons, clear fonts, and high-contrast themes to accommodate users with visual impairments.
- **Responsiveness**: The UI adapts to various screen sizes and orientations on Android and iOS devices.
- **Error Feedback**: Users receive clear, actionable feedback for errors, such as invalid inputs or unsupported actions.
- **Minimalism**: Unnecessary elements are avoided to keep the interface clean and focused.

4.3 GUIComponents

Key GUI components used in **React Native** include:

- **Navigation**: React Navigation for stack and tab management.
- **Media**: React Native Image Picker for uploads.
- **Calendar**:Calendars for memory scheduling.
- **Icons and Styling**: React Native Vector Icons and Styled Components.
- **Notifications**: React Native Push Notifications for reminders.

5.0 Reuse and relationships to other products

5.1 Role of Reuse in Product Design

The design of Memora incorporates reusable components to streamline development, ensure consistency, and reduce redundancy. The following aspects of product design leverage reuse:

- **UI Components**: Common interface elements, such as buttons, input fields, and modals, are designed as reusable React Native components for consistent UI design across screens.
- **Authentication Module**: The user authentication logic, including session management and password recovery, is implemented as a reusable backend service.
- **Media Validation**: A standardized module is used for validating and
- **Map Integration**: Google Maps API is reused for location services in various parts of the app, including the Memory Map and Capsule Details.

6.0 Design decisions and tradeoffs

6.1 Key Design Decisions

1. **Cross-Platform Development with React Native**

- **Decision**: React Native was chosen to build a single codebase for both Android and iOS.
- **Reason**: Simplifies development and maintenance, reducing the need for separate teams for each platform.
- **Impact**: Faster development and consistency across platforms.

2. **MongoDB for Database Management**

- **Decision**: MongoDB was selected for its flexible schema design and scalability.
- **Reason**: Nested data structures for capsules and media storage align well with MongoDB's document-based architecture.
- **Impact**: Simplified handling of hierarchical data like nested capsules.

3. **Integration of Google Maps API**

- **Decision**: Google Maps API was used to implement the Memory Map feature.
- **Reason**: Reliable map services and built-in support for interactive features like pins and tooltips.
- **Impact**: Enhanced user experience with minimal development effort for map functionalities.

4. **Firebase Cloud Messaging for Notifications**

- **Decision**: Firebase Cloud Messaging was integrated to handle push notifications.
- **Reason**: Simplifies notification management and ensures real-time updates.
- **Impact**: Reduced development time for notification systems.

5. **Multer for Media Uploads**

- **Decision**: Multer was integrated to handle media uploads efficiently.
- **Reason**: Provides middleware for handling multipart/form-data, ensuring secure and efficient file uploads.
- **Impact**: Improved file handling, validation, and storage management, reducing backend processing overhead.

6.2 Tradeoffs Made

1. **Performance vs. Development Speed**

- **Choice**: React Native was selected over native development for quicker deployment.
- **Tradeoff**: Slight performance lag on older devices compared to fully native apps.
- **Mitigation**: Optimized React Native code to minimize overhead.

2. **Flexibility vs. Complexity**

- **Choice**: MongoDB was preferred over relational databases for its flexibility with nested data structures.
- **Tradeoff**: More complex queries and indexing strategies compared to traditional relational databases.
- **Mitigation**: Pre-defined indexes and optimized query patterns were implemented.

3. **Third-Party APIs vs. Custom Implementation**

- **Choice**: Google Maps API was used instead of building custom solutions.
- **Tradeoff**: Dependency on external services for critical app functionalities.
- **Mitigation**: Ensured fallback mechanisms and API redundancy planning.

4. **Security vs. User Convenience**

- **Choice**: Strong encryption was prioritized for user data, such as password.
- **Tradeoff**: Slightly slower processing times for encryption and decryption.
- **Mitigation**: Efficient algorithms (e.g., AES) were used to balance security and performance.

5. **Feature Richness vs. Simplicity**

- **Choice**: The app included advanced features like nested capsules and milestone-based unlocking.
- **Tradeoff**: Increased complexity in the user interface and backend logic.
- **Mitigation**: Conducted user testing to ensure a seamless and intuitive experience.

7.0 Pseudocode for components

7.1 Register User

```
IF NOT is Empty(email) AND NOT is Empty(password) THEN
   normalizedEmail ← normalize(email)
   existingUser ← find User By Email(normalizedEmail)
   IF existingUser IS NOT FOUND THEN
```

```
    hashedPassword ← hash Password(password)
    user ← create User(normalizedEmail, hashedPassword)
    token ← generate JWT(user._id)
    DISPLAY "User registered successfully", token, user._id
  ELSE
    DISPLAY "User already exists"
  END IF
ELSE
  DISPLAY "Please provide all required fields."
END IF
END FUNCTION
```

7.2 User Login

```
user ← find User By Email(email)
IF user IS FOUND THEN
  isMatch ← compare Password(password, user.password)
  IF isMatch THEN
    token ← generate JWT(user._id)
    DISPLAY "Login successful", token, user._id
  ELSE
    DISPLAY "Invalid credentials"
  END IF
ELSE
  DISPLAY "User not found"
END IF
END FUNCTION
```

7.3 Create Profile

```
existingProfile ← find Profile By User(userId)
IF existingProfile IS NOT FOUND THEN
  profile ← create Profile(userId, profileData, profilePicture)
  DISPLAY "Profile created successfully", profile
ELSE
  DISPLAY "Profile already exists"
END IF
END FUNCTION
```

7.4 Update Profile

```
profile ← find Profile By User(userId)
IF profile IS FOUND THEN
  update Profile Fields(profile, updatedData, profilePicture)
  save Profile(profile)
  DISPLAY "Profile updated successfully", profile
ELSE
  DISPLAY "Profile not found"
END IF
```

```
END FUNCTION
```

7.5 Delete Profile

```
profile ← delete Profile By User(userId)
IF profile IS FOUND THEN
   DISPLAY "Profile deleted successfully"
ELSE
   DISPLAY "Profile not found"
END IF
END FUNCTION
```

7.6 Timecapsule Creation

```
IF NOT is Empty(title) AND NOT is Empty(description) AND NOT is Empty(unlockDate) THEN
   timeCapsule ← create New Time Capsule(userId, title, description, unlockDate, capsuleType, media)
   IF capsuleType IS "Shared" THEN
      sharedCapsule ← create Shared Capsule(timeCapsule._id, userId)
      DISPLAY "Time Capsule created and shared successfully"
   ELSE
      DISPLAY "Time Capsule created successfully"
   END IF
ELSE
   DISPLAY "Invalid data. Please provide all required fields."
END IF
END FUNCTION
```

7.7 Send Friend Request

```
IF userId = friendUserId THEN
   DISPLAY "You cannot send a friend request to yourself."
ELSE
   existingFriendship ← find Friendship (userId, friendUserId)
   IF existingFriendship IS FOUND THEN
      DISPLAY "Friendship already exists or pending."
   ELSE
      friendship ← create New Friendship(userId, friendUserId, status: 'pending')
      save Friendship(friendship)
      DISPLAY "Friend request sent successfully."
   END IF
END IF
END FUNCTION
```

7.8 Accept Friend Request

```
friendship ← find Friendship(friendshipId, loggedInUserId)
IF friendship IS FOUND THEN
```

```
    update Friendship Status(friendship, status: 'accepted')
    save Friendship(friendship)
    DISPLAY "Friend request accepted."
ELSE
    DISPLAY "Friend request not found."
END IF
END FUNCTION
```

7.9 Get Friend Request

```
friendships ← find Friendships(userId, status: 'accepted')
friends ← map Friendships To Friend List(friendships, userId)
DISPLAY "Friends list", friends
END FUNCTION
```

7.10 Remove Friend Request

```
delete Friendships(userId, friendUserId)
DISPLAY "Friend removed successfully."
END FUNCTION
```

7.11 Get Pending Friend Requests

```
pendingRequests ← find Friendships(friend_user_id: userId, status: 'pending')
DISPLAY "Pending friend requests", pendingRequests
END FUNCTION
```

7.12 Decline Friend Request

```
friendship ← find Friendship(friendshipId, loggedInUserId)
IF friendship IS FOUND THEN
    update Friendship Status(friendship, status: 'rejected')
    save Friendship(friendship)
    DISPLAY "Friend request declined."
ELSE
    DISPLAY "Friend request not found."
END IF
END FUNCTION
```

8.0 Appendices

8.1 Class Diagram

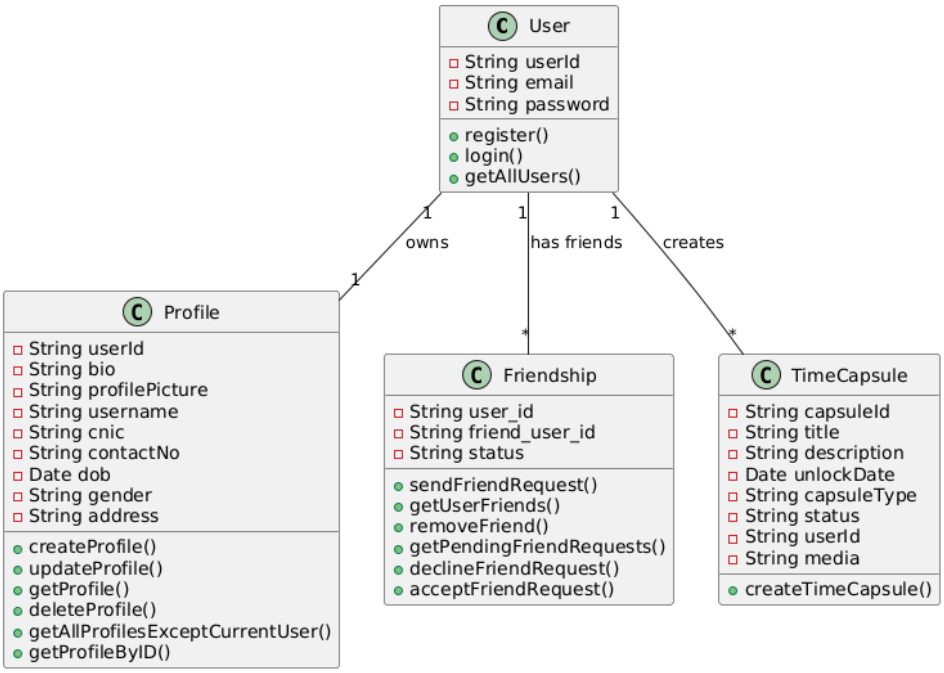

Figure 2:Class Diagram

8.2 Statechart Diagram

8.2.1 Emotional State Diagram

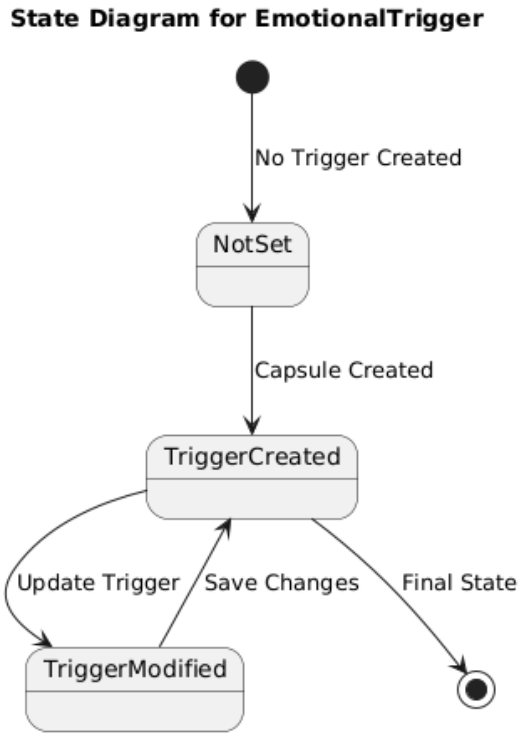

Figure 3:Emotional State Diagram

8.2.2 Friendship State Diagram

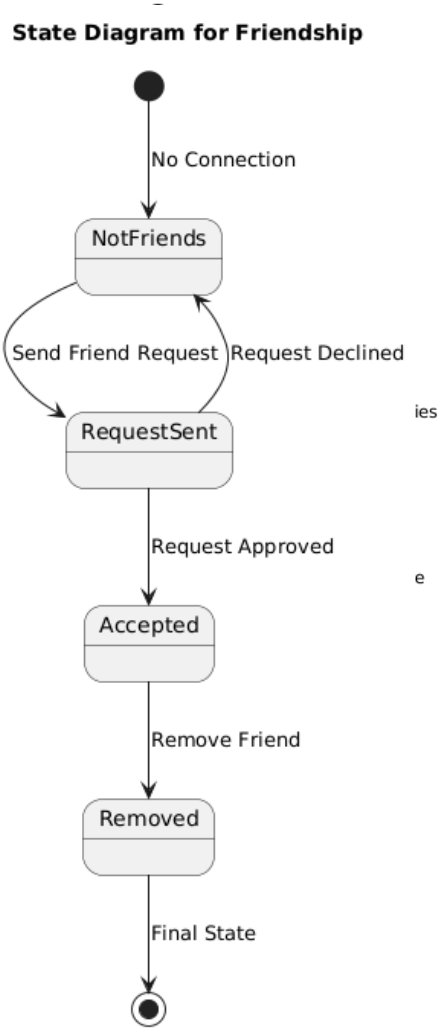

Figure 4:Friendship State Diagram

8.2.3 Media State Diagram

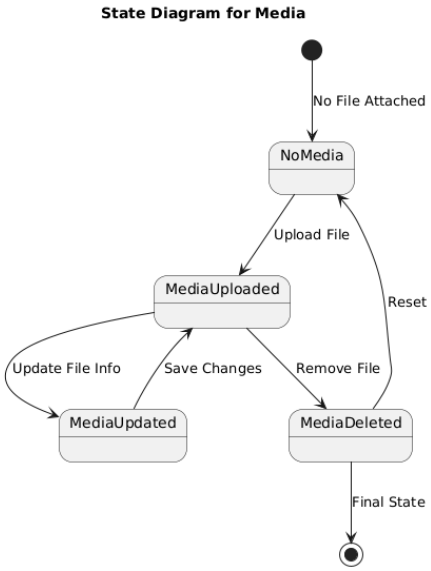

Figure 5:Media State Diagram

8.2.4 Memory Calendar State Diagram

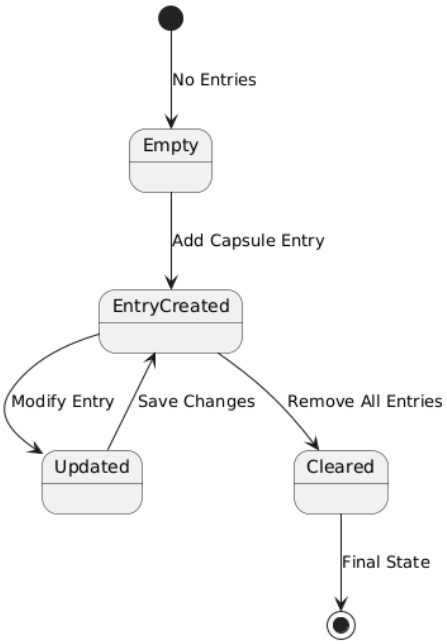

Figure 6: Memory Calendar State Diagram

8.2.5 Nested Capsule State Diagram

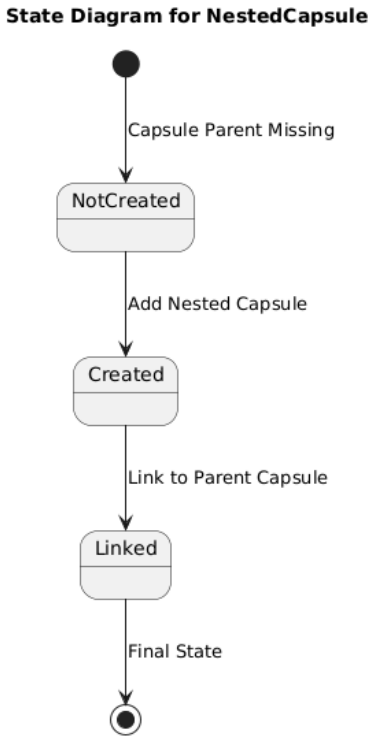

Figure 7: Nested Capsule State Diagram

8.2.6 Profile State Diagram

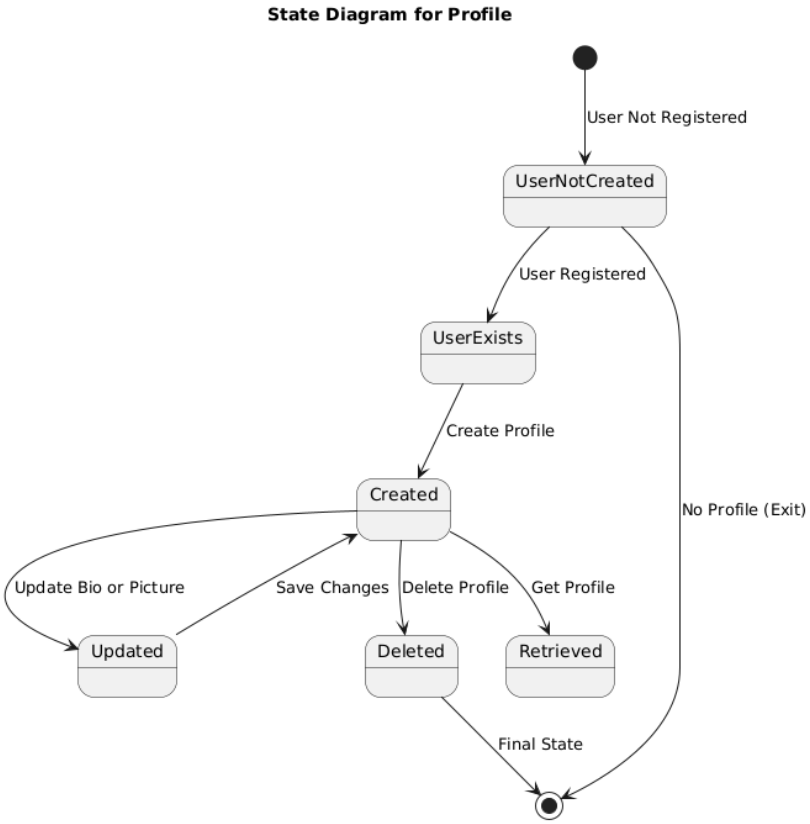

Figure 8: Profile State Diagram

8.2.7 Shared Capsule State Diagram

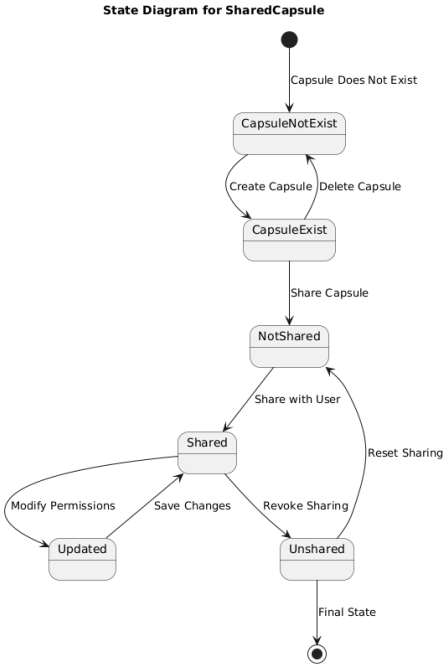

Figure 9: Shared Capsule State Diagram

8.2.8 TimeCapsule state Diagram

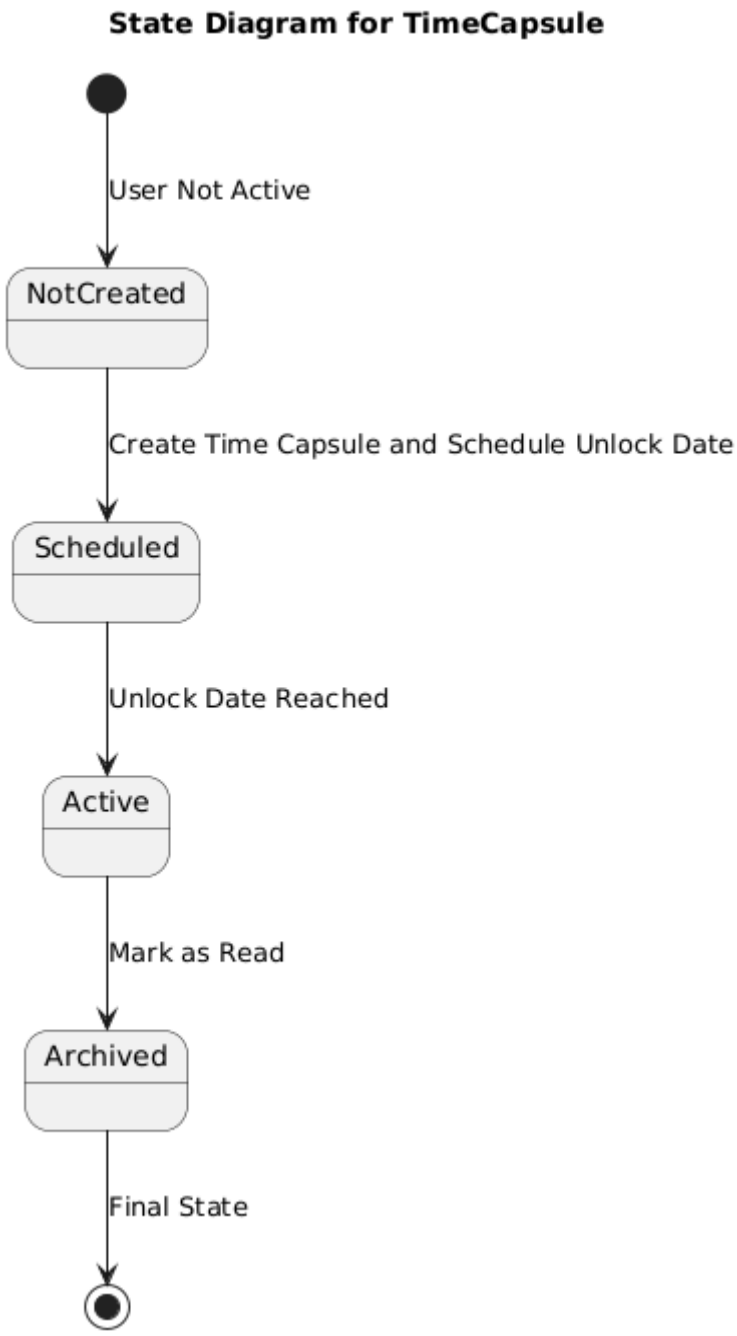

Figure 10:TimeCapsule state Diagram

8.2.9 User State Diagram

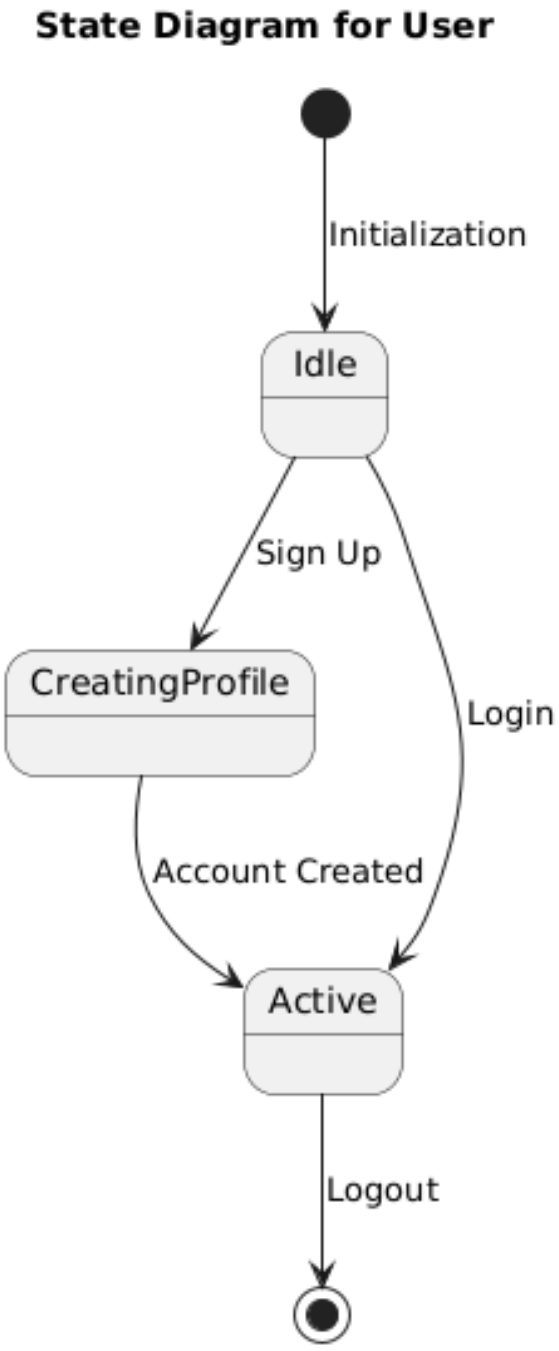

Figure 11: User State Diagram

8.3 Component Diagram

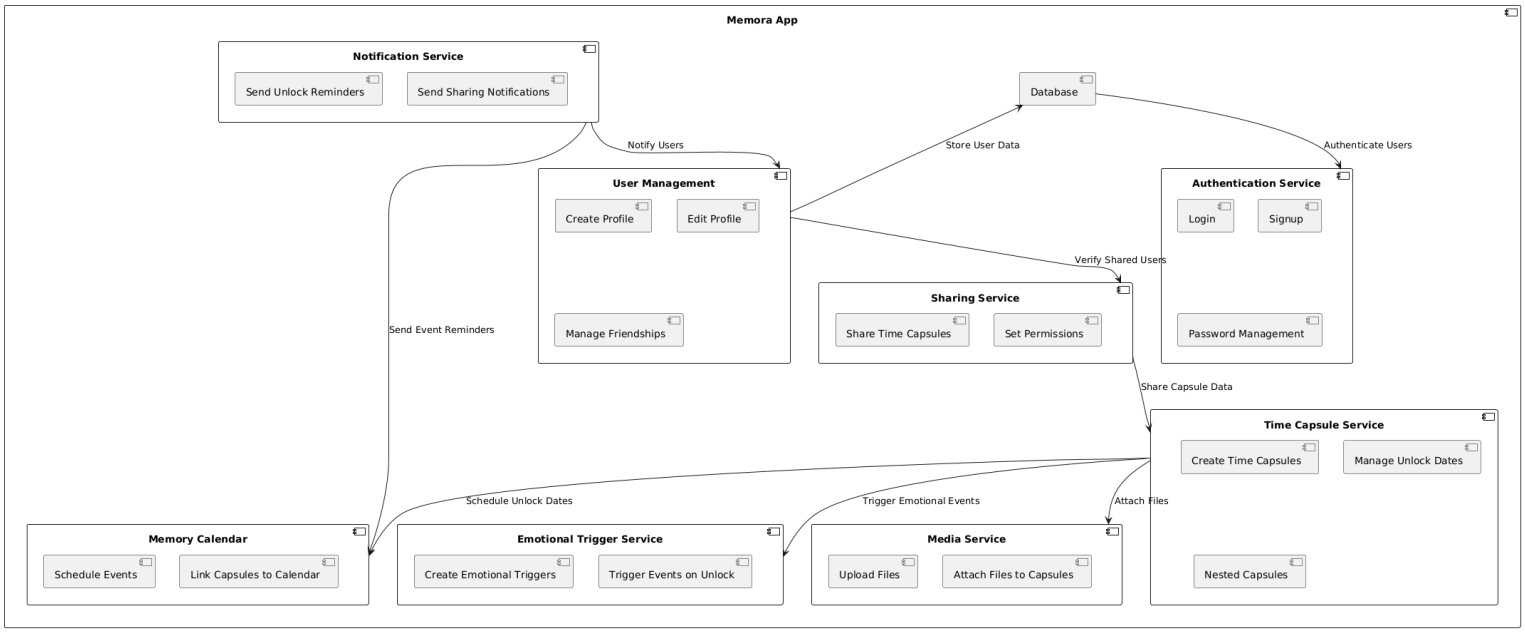

Figure 12: Component Diagram

8.4 Seqence Diagram

8.4.1 Capsule Type

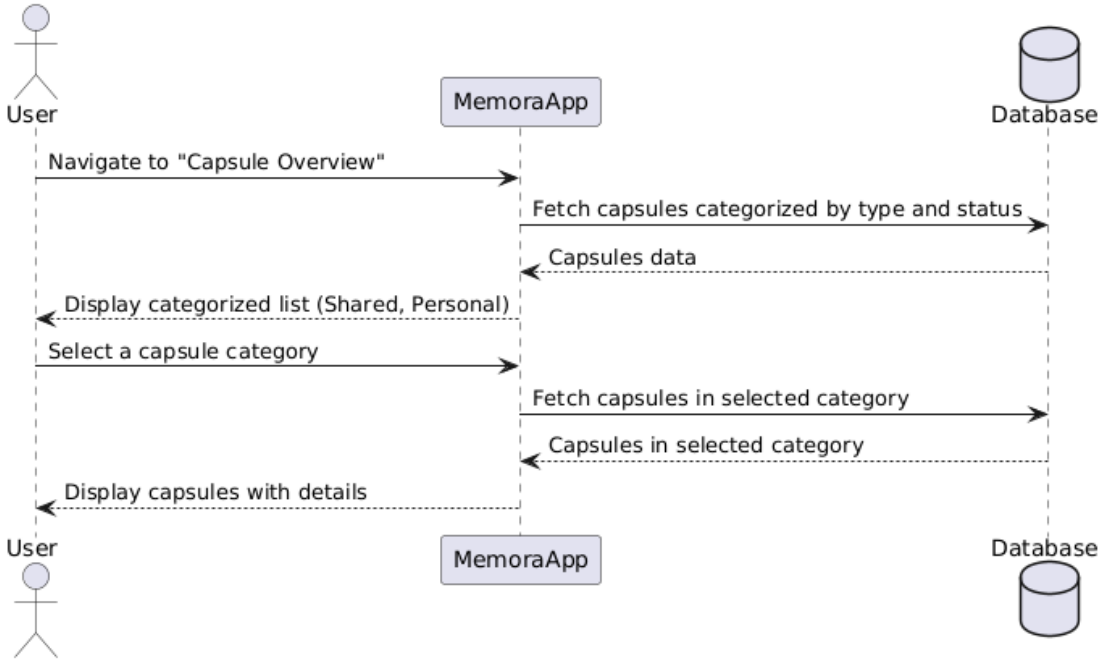

Figure 13:Capsule Type Seqence Diagram

8.4.2 Calendar on Map

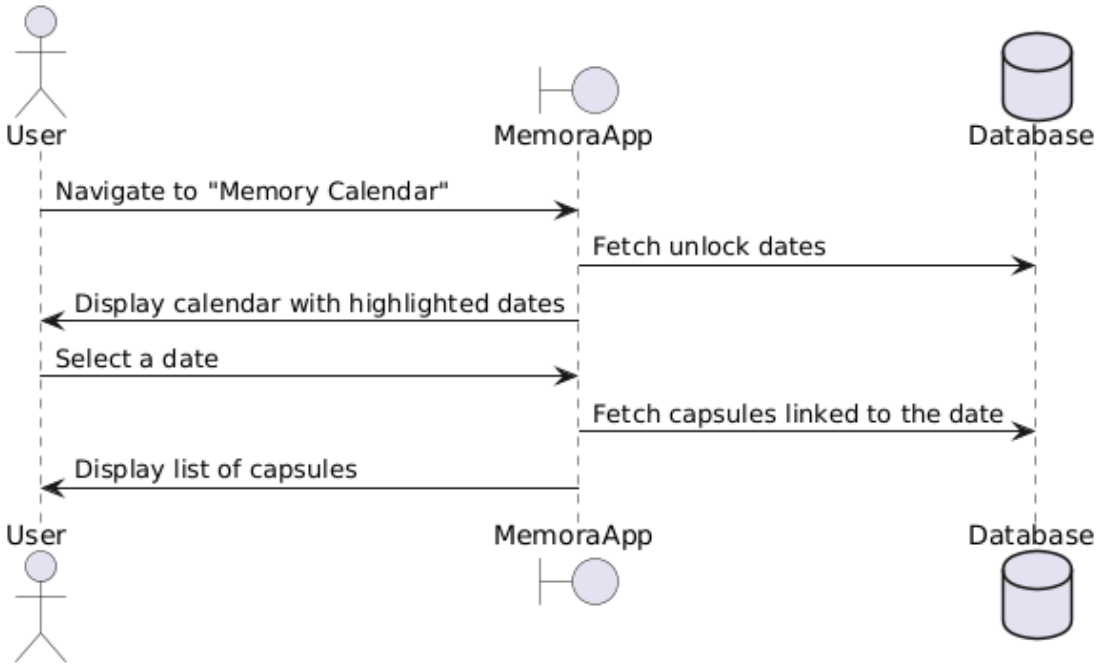

Figure 14: Calendar on Map Sequence Diagram

8.4.3 Emotional

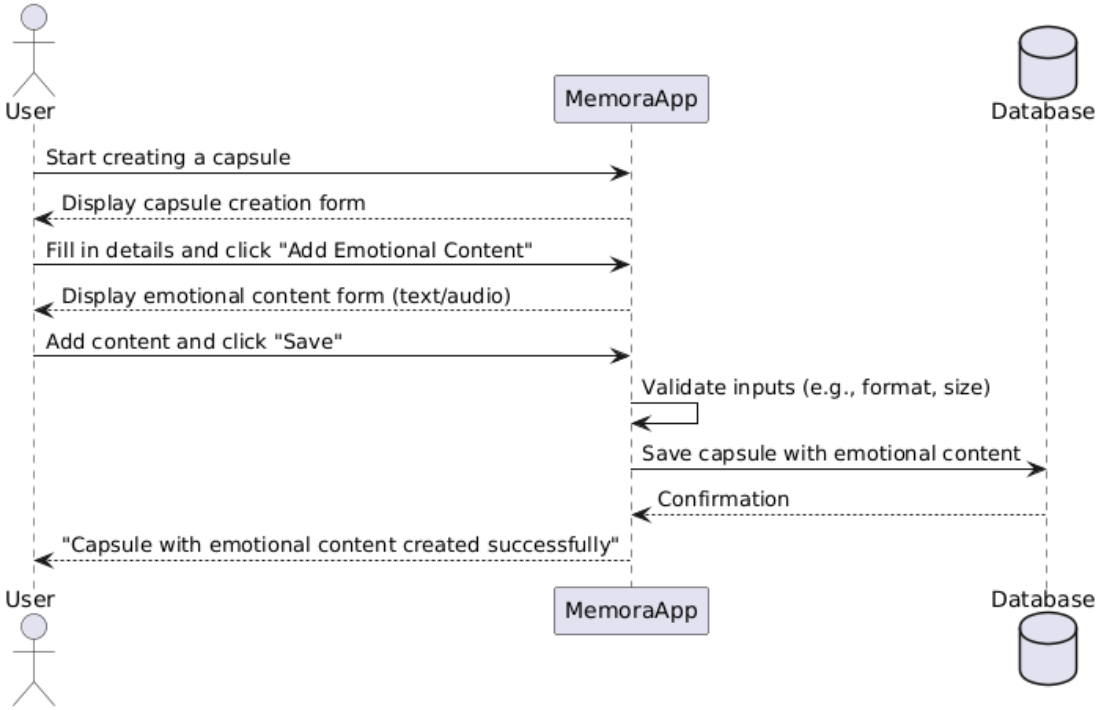

Figure 15: Emotional Sequence Diagram

8.4.4 Profile and Friends Management

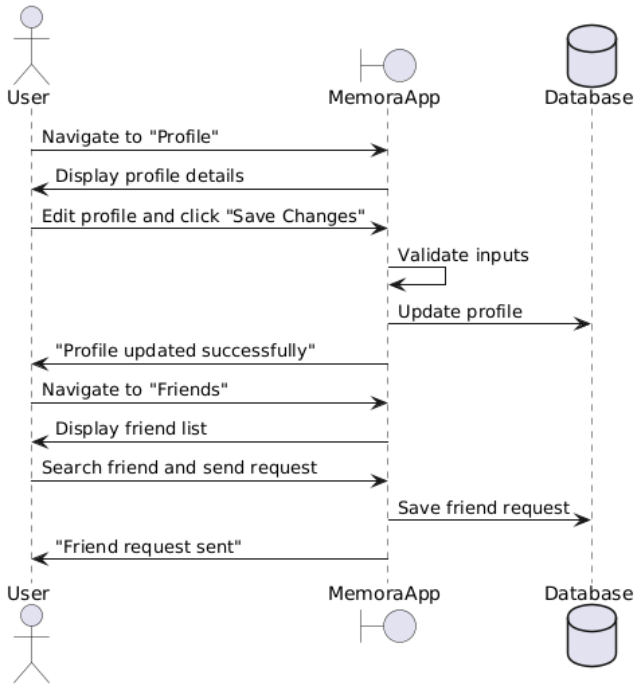

Figure 16:Friends Management Sequence Diagram

8.4.5 Login

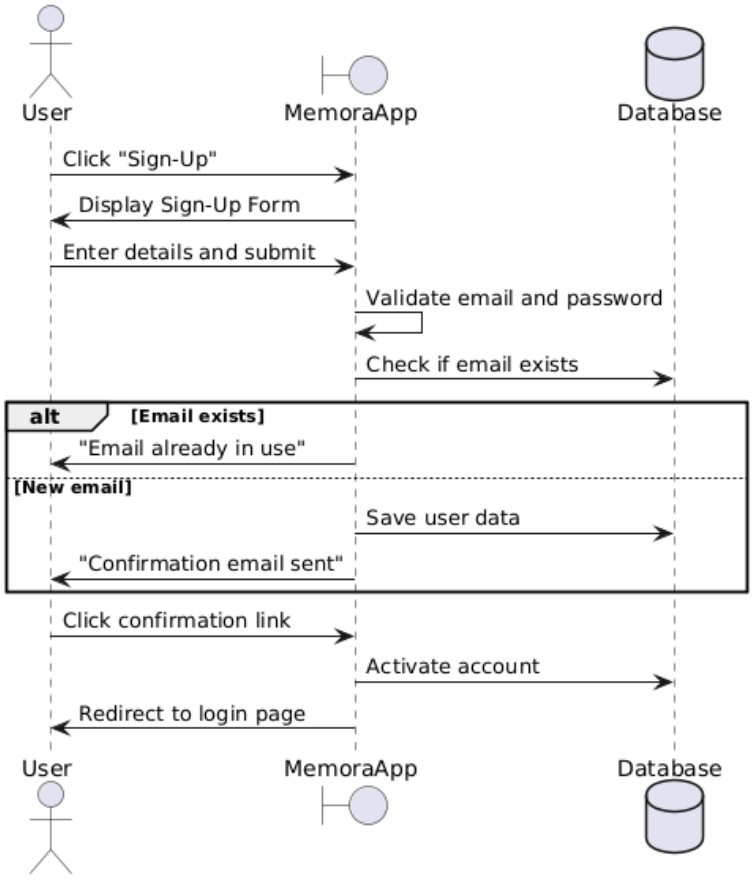

Figure 17: Login Sequence Diagram

8.4.6 Media

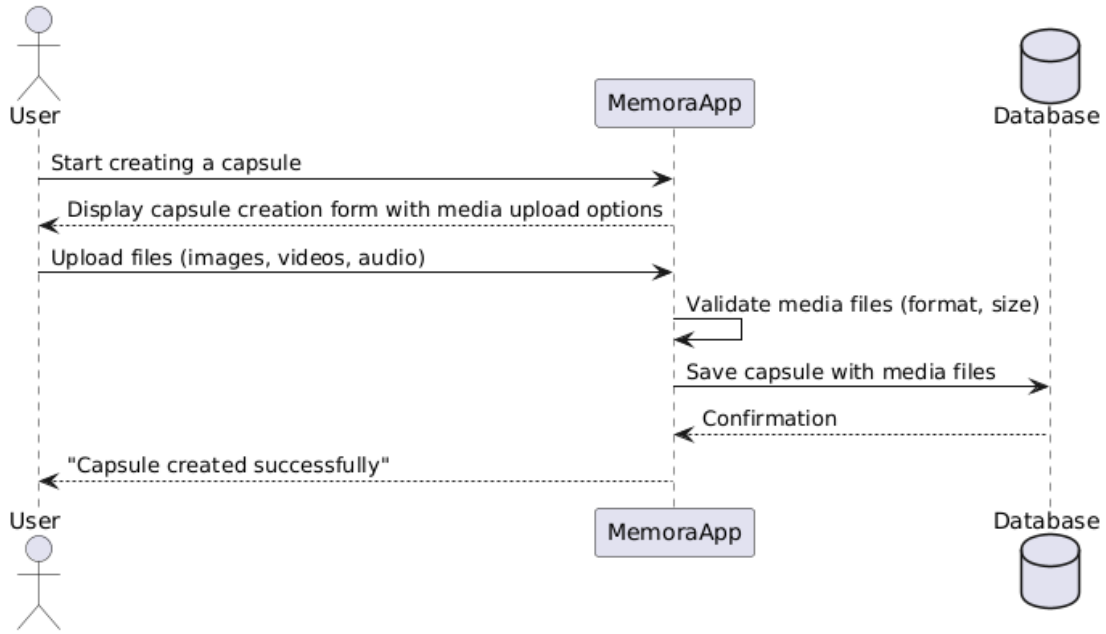

Figure 18:Media Sequence Diagram

8.4.7 Nested Capsule

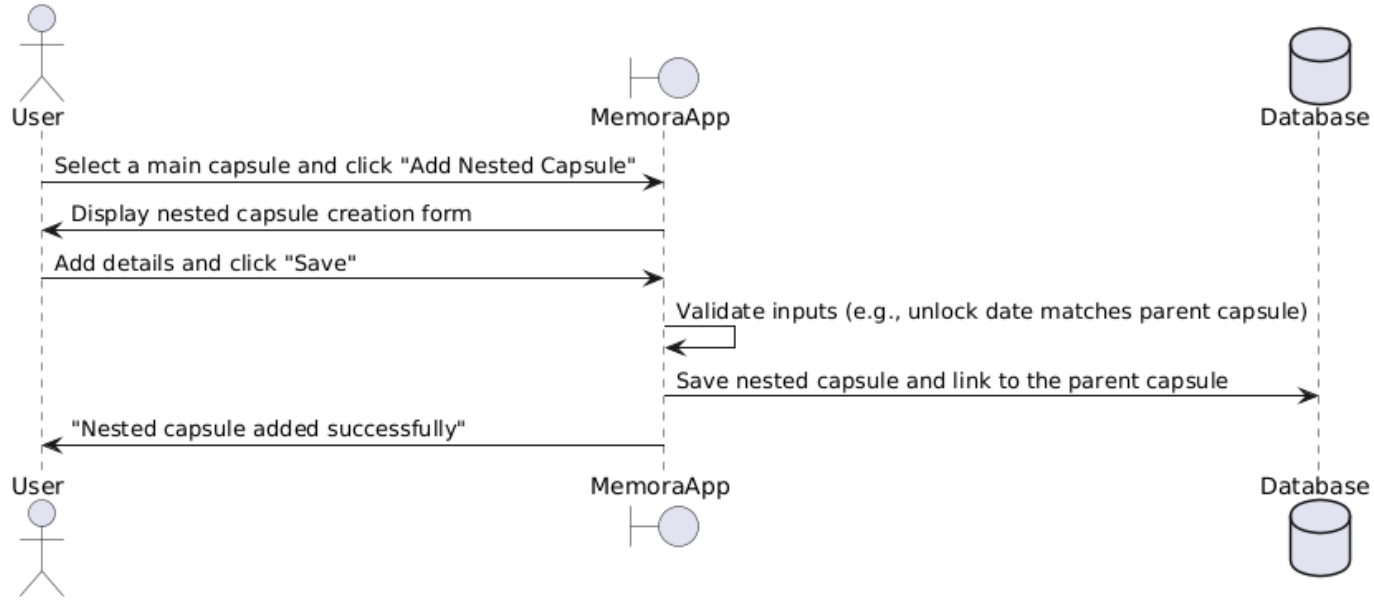

Figure 19:Nested Capsule Sequence Diagram

8.4.8 Shared Capsule

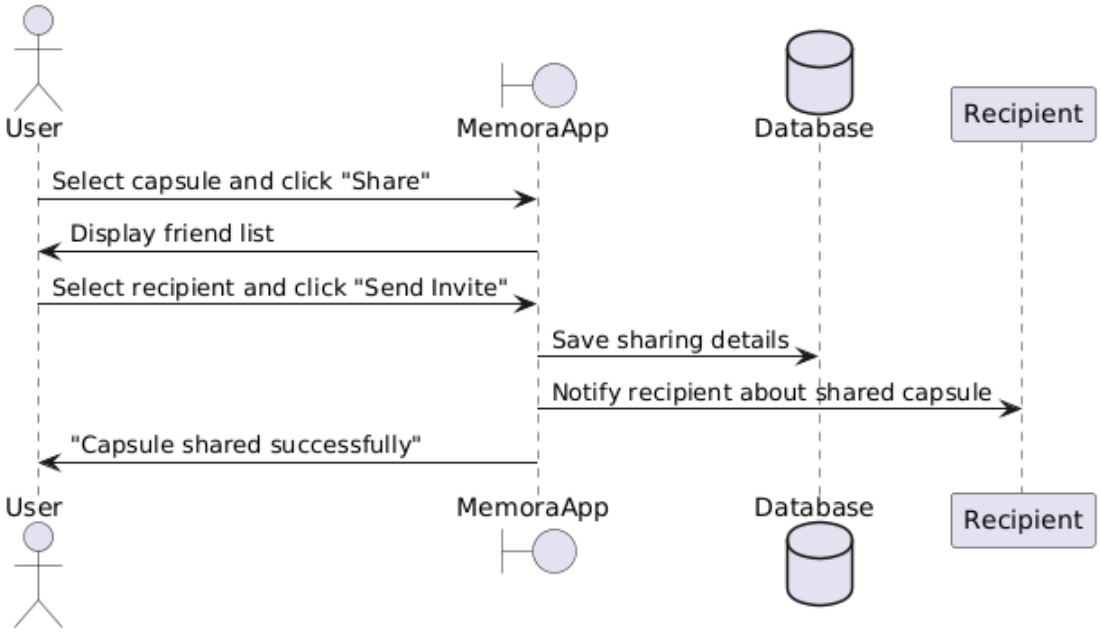

Figure 20:Shared Capsule Sequence Diagram

8.4.9 TimeCapsule Creation

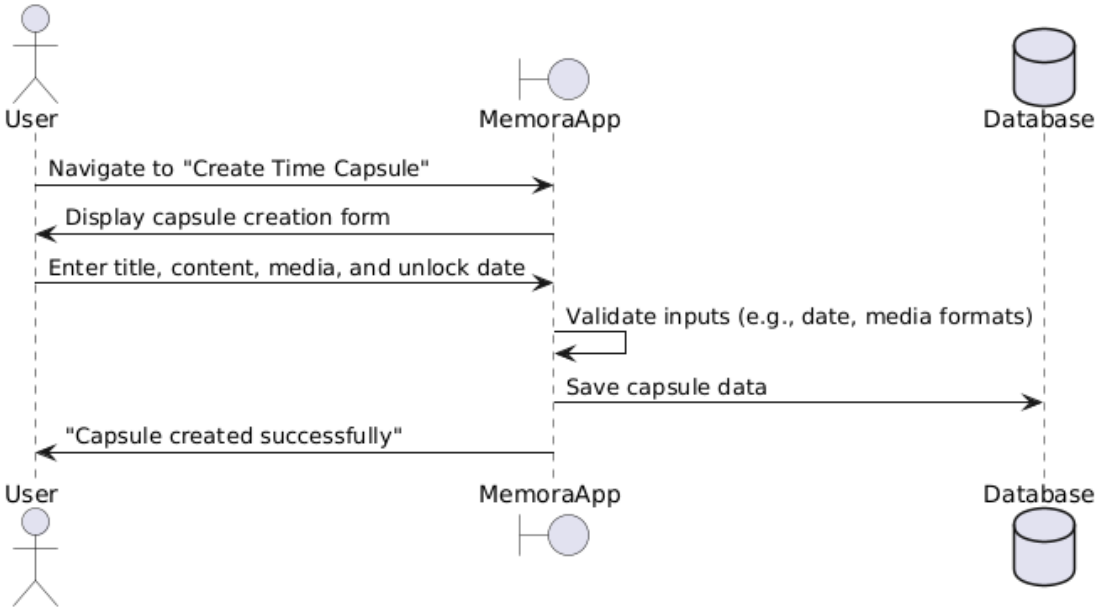

Figure 21:TimeCapsule Creation Sequence Diagram

8.5 Activity Diagram

8.5.1 Capsule Type

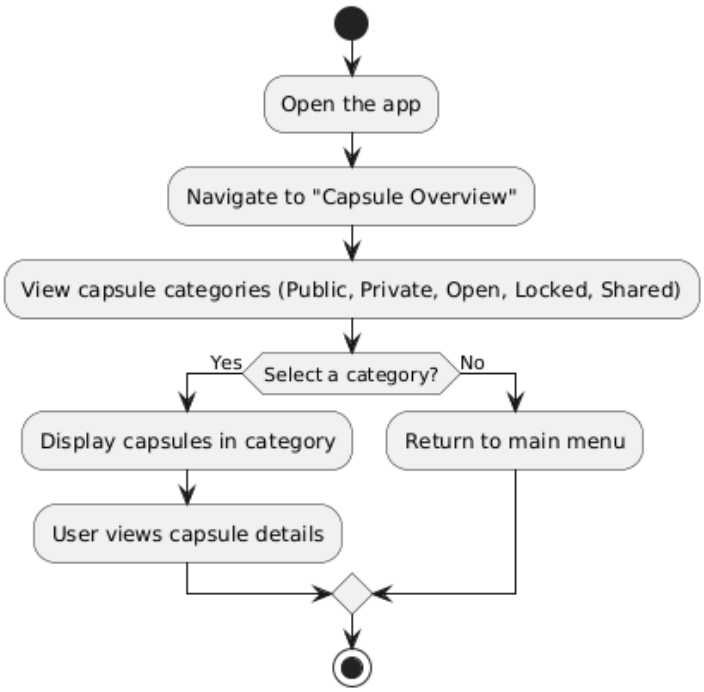

Figure 22: Capsule Type Activity Diagram

8.5.2 Login

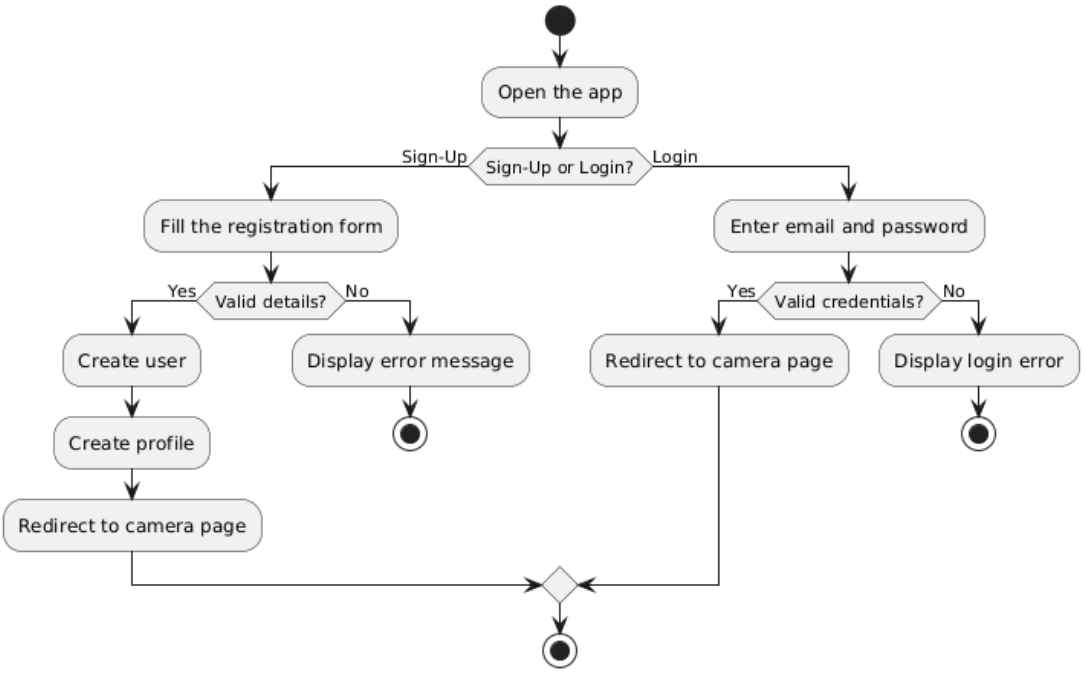

Figure 23:Login Activity Diagram

8.5.3 Nested Capsule

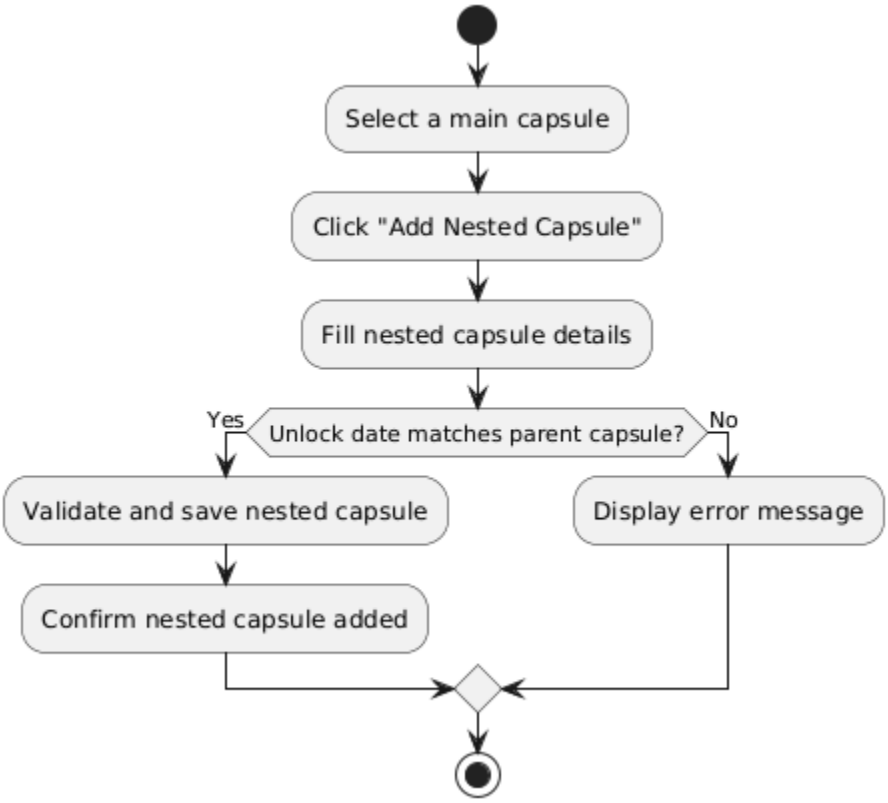

Figure 24: Nested Capsule Activity Diagram

8.5.4 Shared Capsule

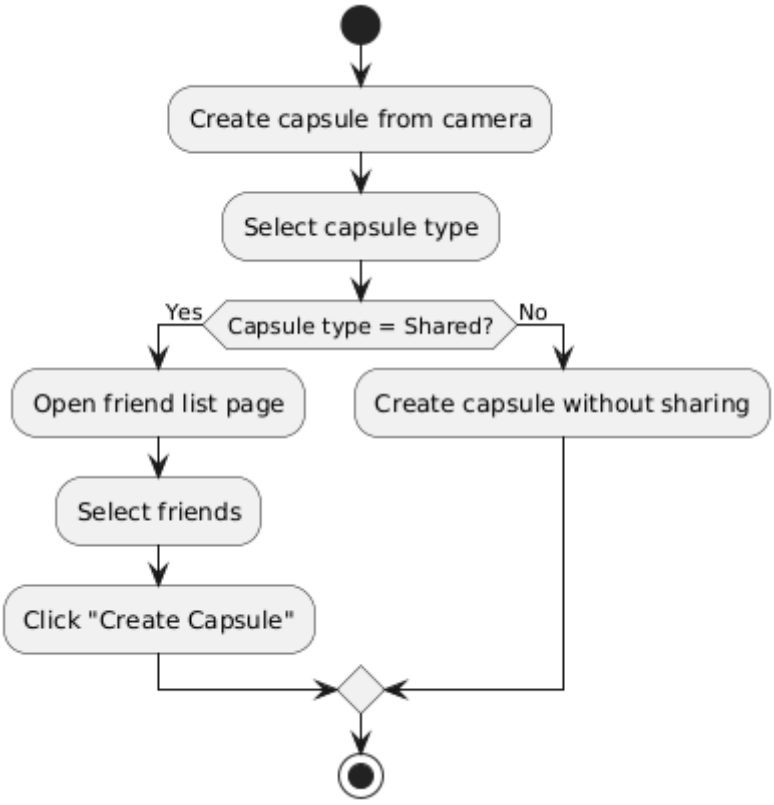

Figure 25:Shared Capsule Activity Diagram

8.6 Use-Case Diagram

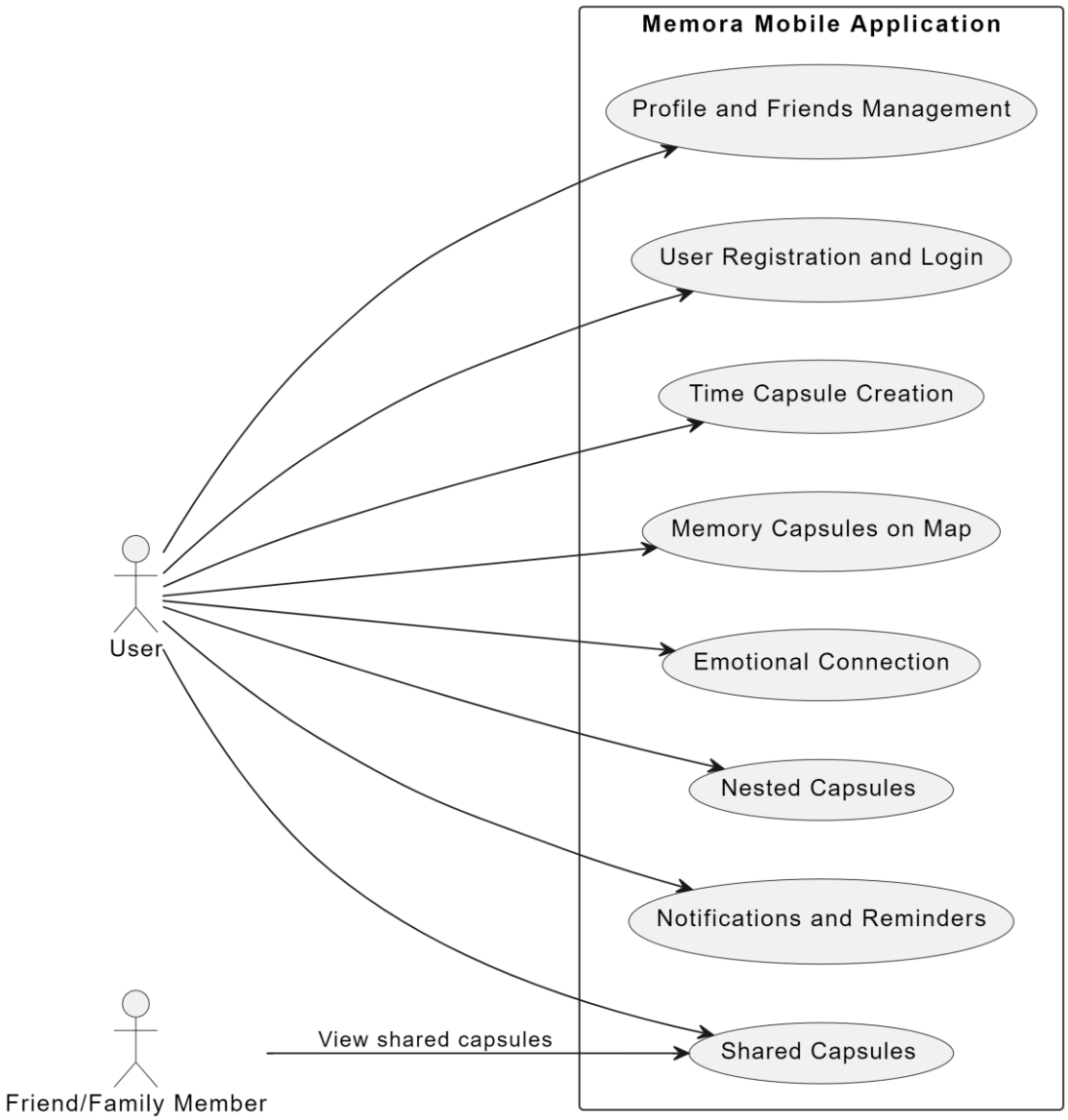

Figure 26:Use-Case Diagram