LETI-DSSMV Project

Technical Report - Tick it (Revamped)

GROUP 5 - 2DD

Project by:

André Moreira (1240567)
Diogo Nogueira (1241692)

January 2026

Contents

List of Figures

List of Tables

Chapter 1

Context

The **Tick it (Revamped)** is a mobile note-taking application built with `React Native` (and `Expo`) that provides offline-first note management with optional synchronization to **Todoist**. The application implements biometric security, weather integration, and calendar-based note organization.

The motivation for this project arose from the need for a practical tool to manage personal tasks and thoughts in a single space. **Tick it (Revamped)** was designed with the aim of providing a clean interface.

The application serves users who need:

- Local-first note storage with optional cloud sync
- Location-based weather display
- Calendar-based note scheduling
- Biometric-protected access

The application is built on the following technology stack:

Table 1.1: Technology Stack

Technology	**Purpose**
React Native	Mobile application framework
react-native-calendars	Calendar UI components
TypeScript	Type safety and developer experience
axios	HTTP client for API requests
Expo	Development platform and SDK
expo-router	File-based navigation system
expo-sqlite	Local database (SQLite)
expo-secure-store	Encrypted credential storage
expo-local-authentication	Biometric authentication (Face ID/Fingerprint)
expo-web-browser	OAuth browser for Todoist login
expo-location	Location services for weather

Chapter 2

Analysis

2.1 Domain Model

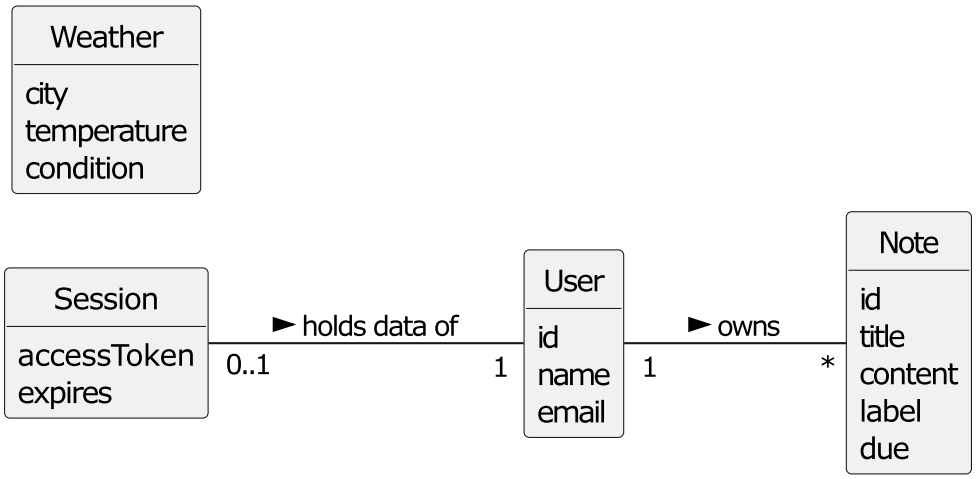

Figure 2.1: Domain Model

2.2 Non-Functional Requirements

The requirements analysis follows the **FURPS+** model. While the **Functional** (F) requirements are detailed in section 2.3, this section focuses on the non-functional attributes: **Usability, Reliability, Performance, Supportability**, and **Design Constraints** (+).

2.2.1 Usability

- **Minimalist Aesthetic:** The user interface shall prioritize a clean design.
- **Learnability:** First-time users should be able to use the app without requiring a tutorial or external documentation.

2.2.2 Reliability

- **Offline Persistence:** The application must guarantee that all notes created or edited while offline are saved locally to the database and are available immediately upon restarting the app.

2.2.3 Performance

- **Responsiveness:** Transitions between screens should appear instantaneous to the user, avoiding noticeable lag or "jank" during navigation.

2.2.4 Supportability

- **Code Quality:** The source code shall adhere to standard `TypeScript` best practices, including proper interface/type definitions for all data models.
- **Project Structure:** The codebase must be organized by feature or type to facilitate peer review and grading.

2.2.5 + (Constraints)

- **Development Timeline:** The Minimum Viable Product (MVP) must be completed and ready for demonstration by the project deadline (January 2026).
- **Technology Stack:** The solution is constrained to the `React Native` framework as defined in the project proposal.
- **Free Tier Services:** Any external APIs (Weather, Todoist) used must operate within their free-tier usage limits to avoid project costs.

2.3 Functional Requirements

2.3.1 Use-Case Diagram

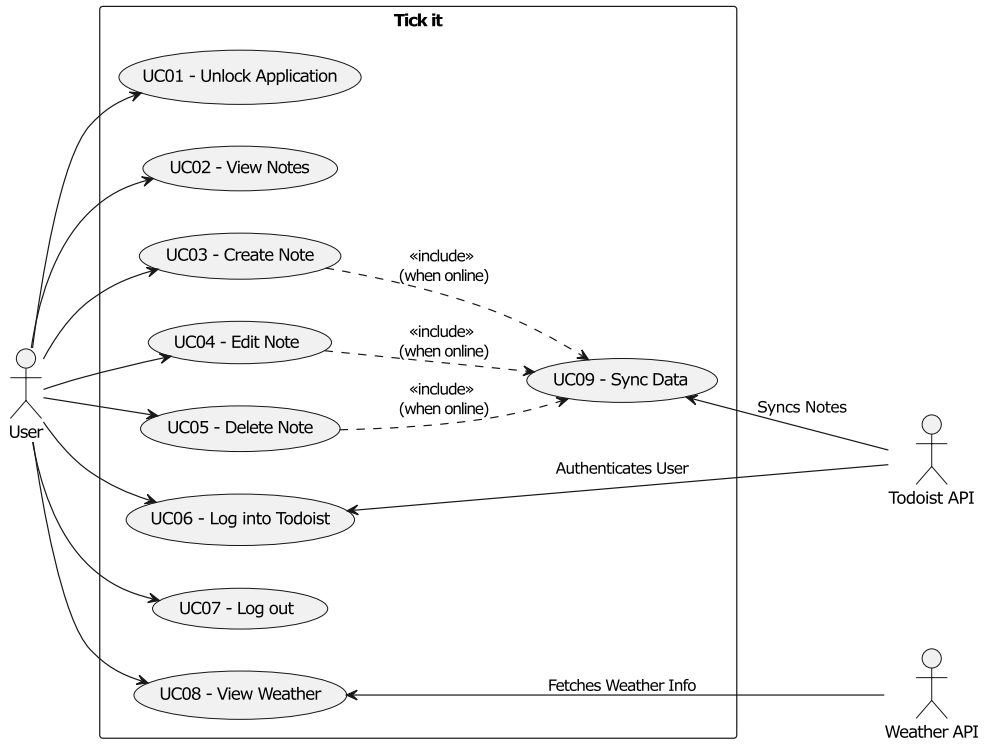

Figure 2.2: Use-Case Diagram

2.3.2 System Sequence Diagrams

UC01 - Unlock Application

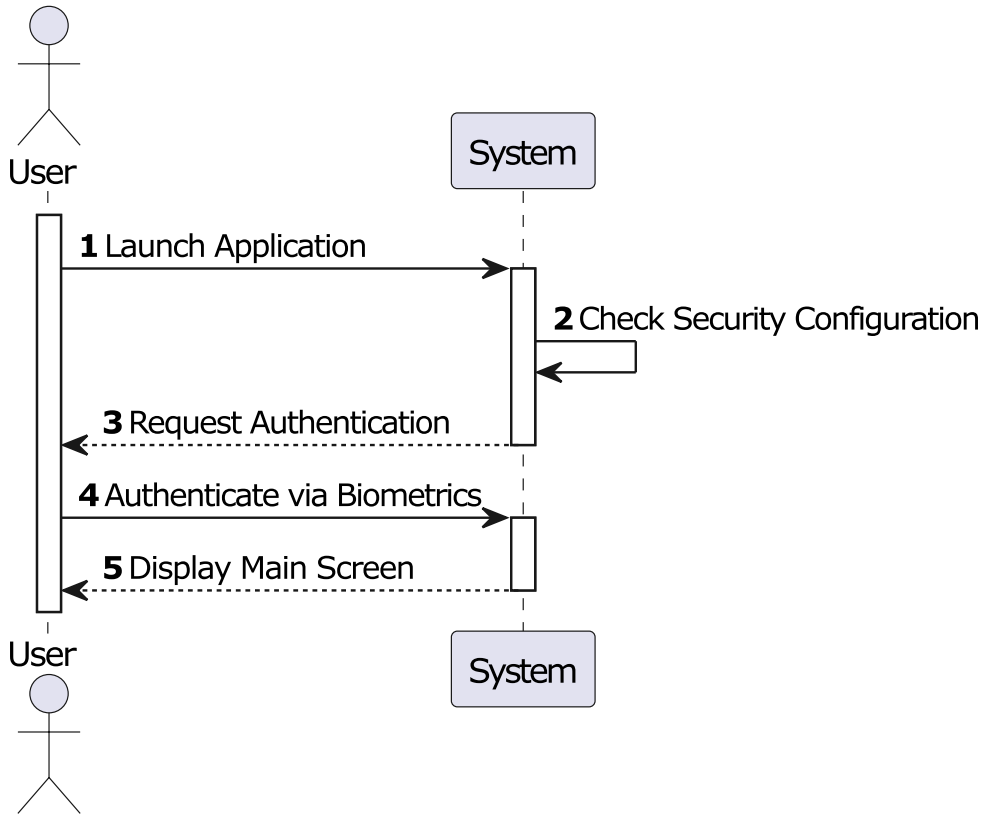

Figure 2.3: System Sequence Diagram (UC01)

UC02 - View Notes

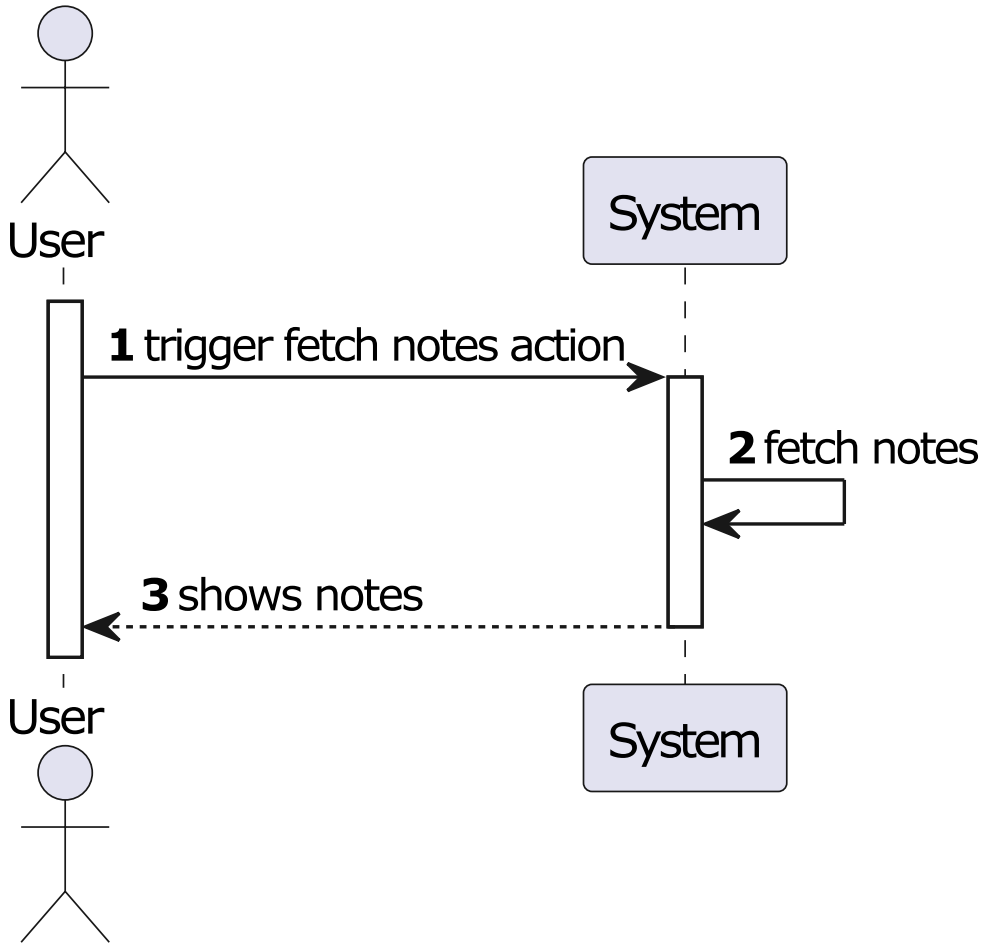

Figure 2.4: System Sequence Diagram (UC02)

UC03 - Create Note

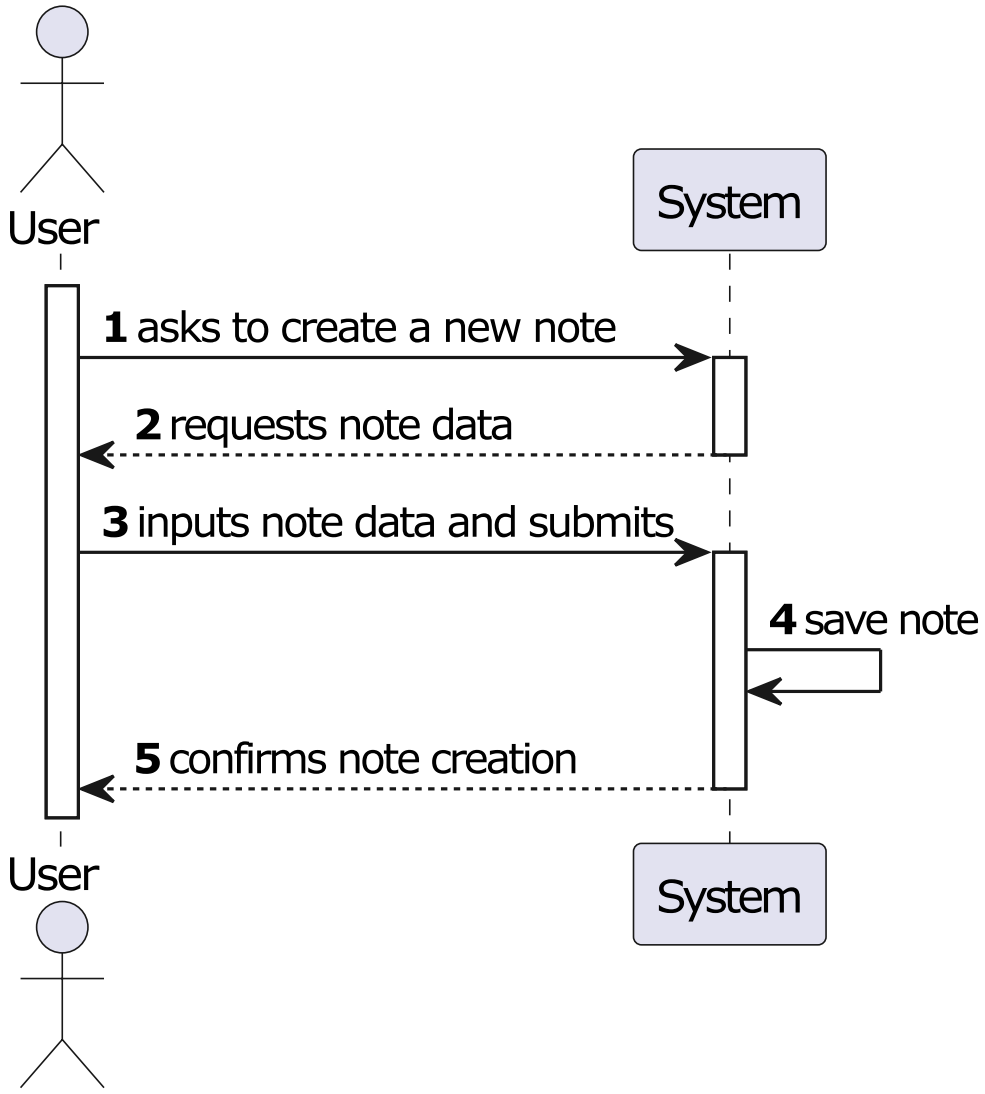

Figure 2.5: System Sequence Diagram (UC03)

UC04 - Edit Note

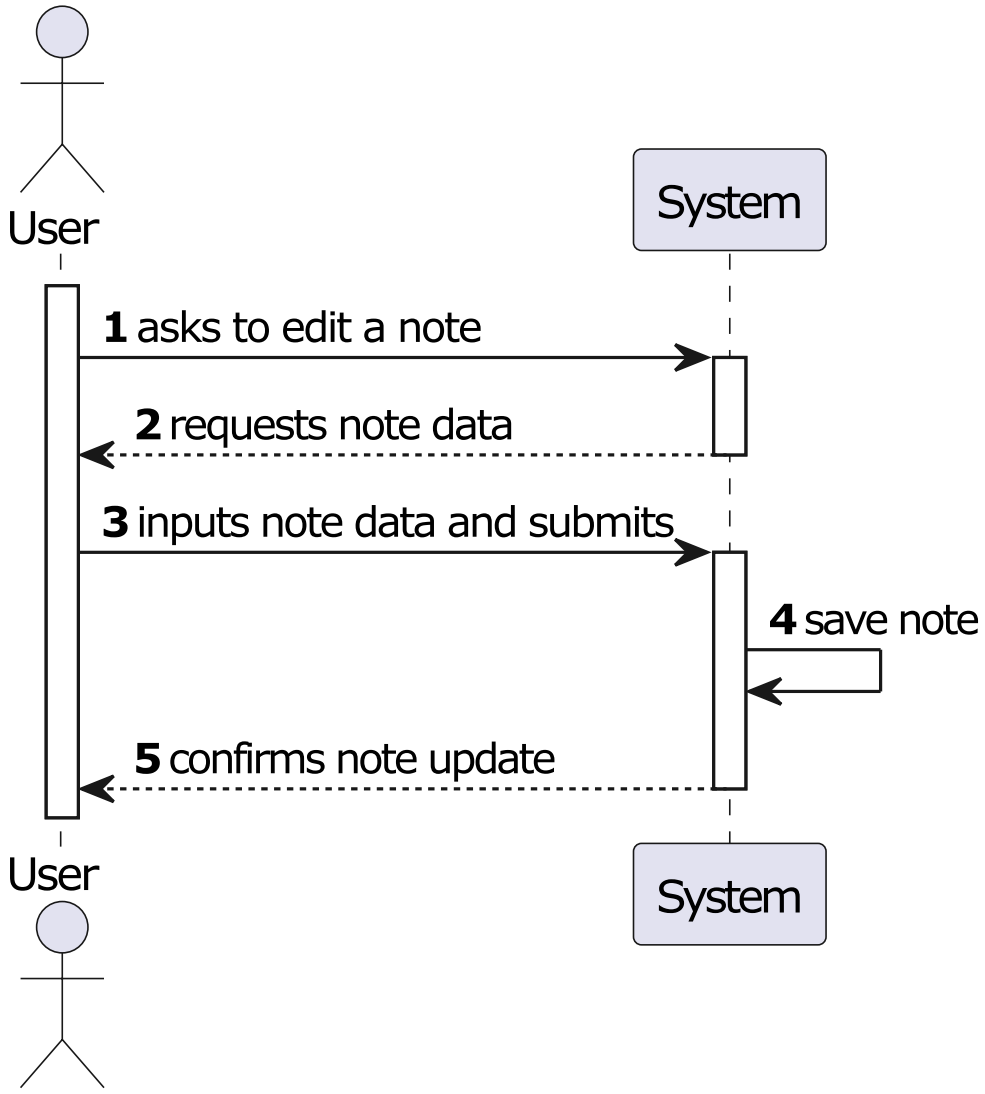

Figure 2.6: System Sequence Diagram (UC04)

UC05 - Delete Note

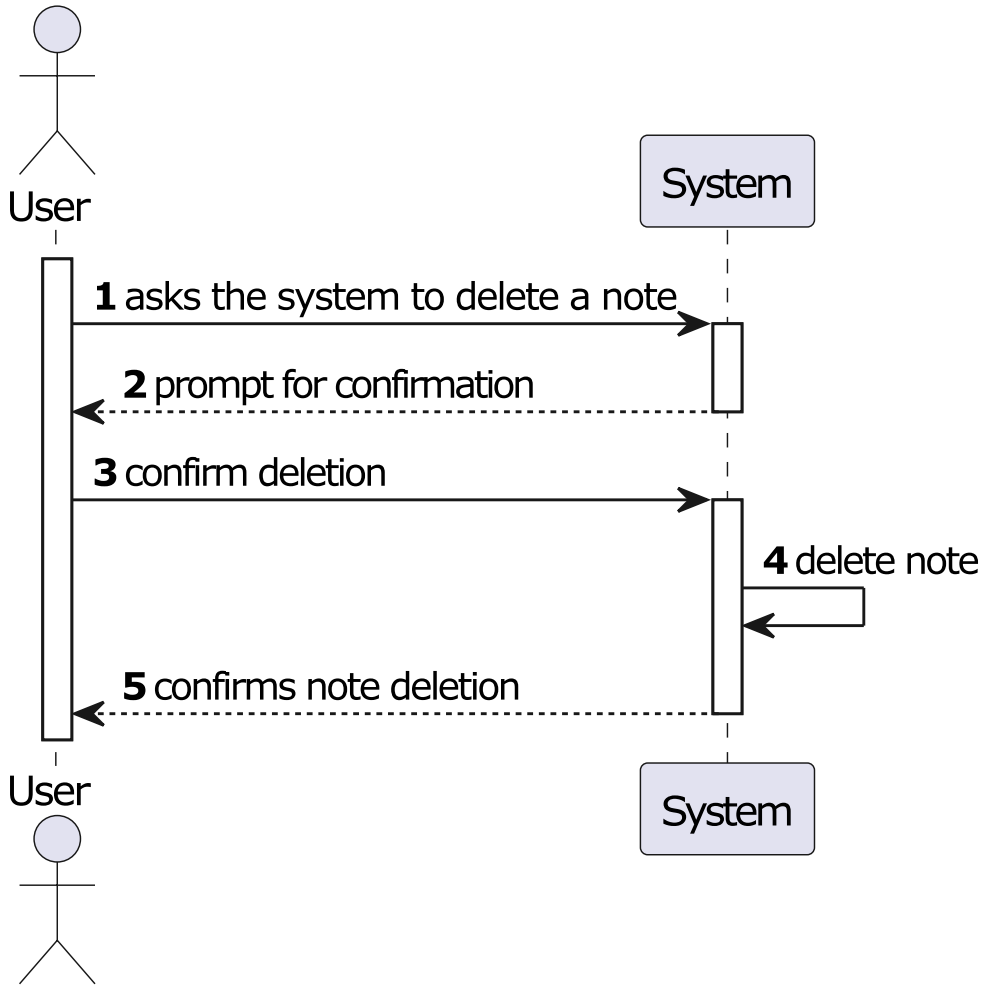

Figure 2.7: System Sequence Diagram (UC05)

UC06 - Log into Todoist

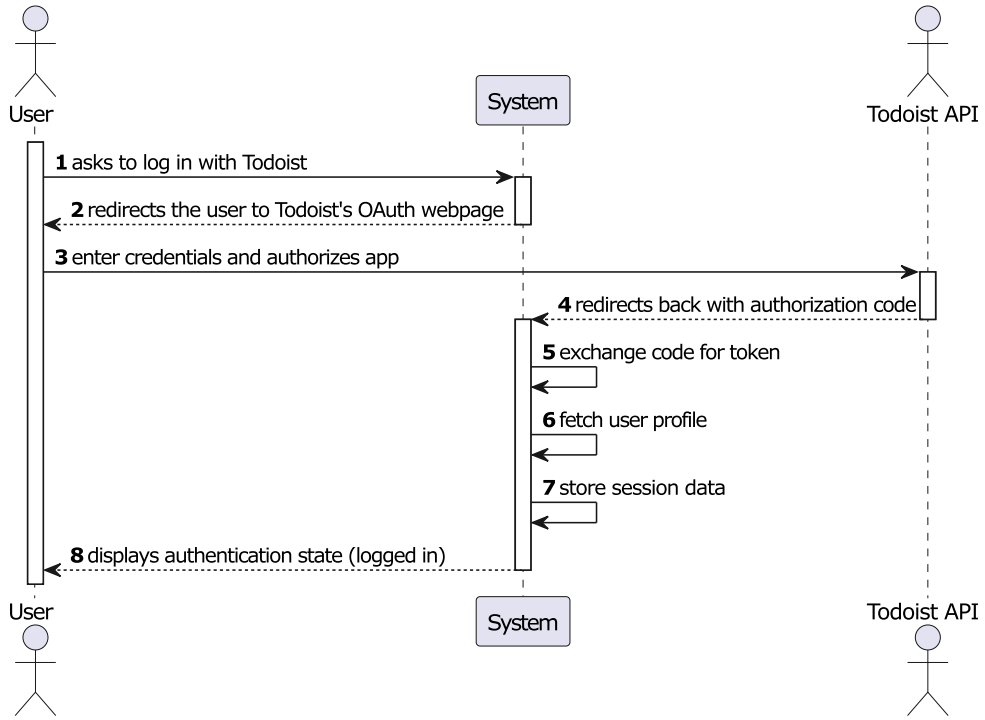

Figure 2.8: System Sequence Diagram (UC06)

UC07 - Log out

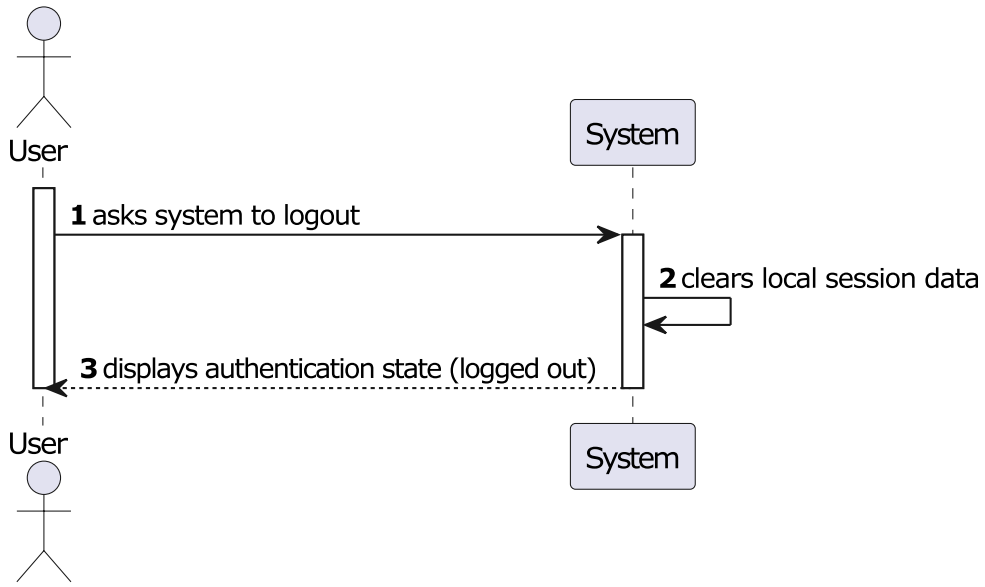

Figure 2.9: System Sequence Diagram (UC07)

UC08 - View Weather

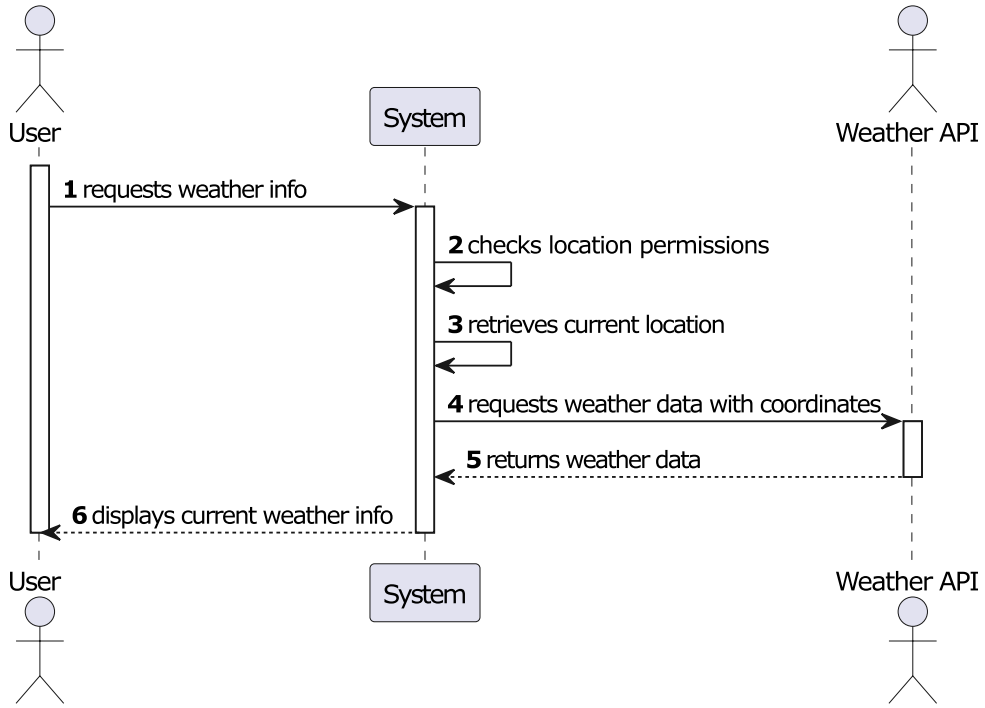

Figure 2.10: System Sequence Diagram (UC08)

UC09 - Sync Data

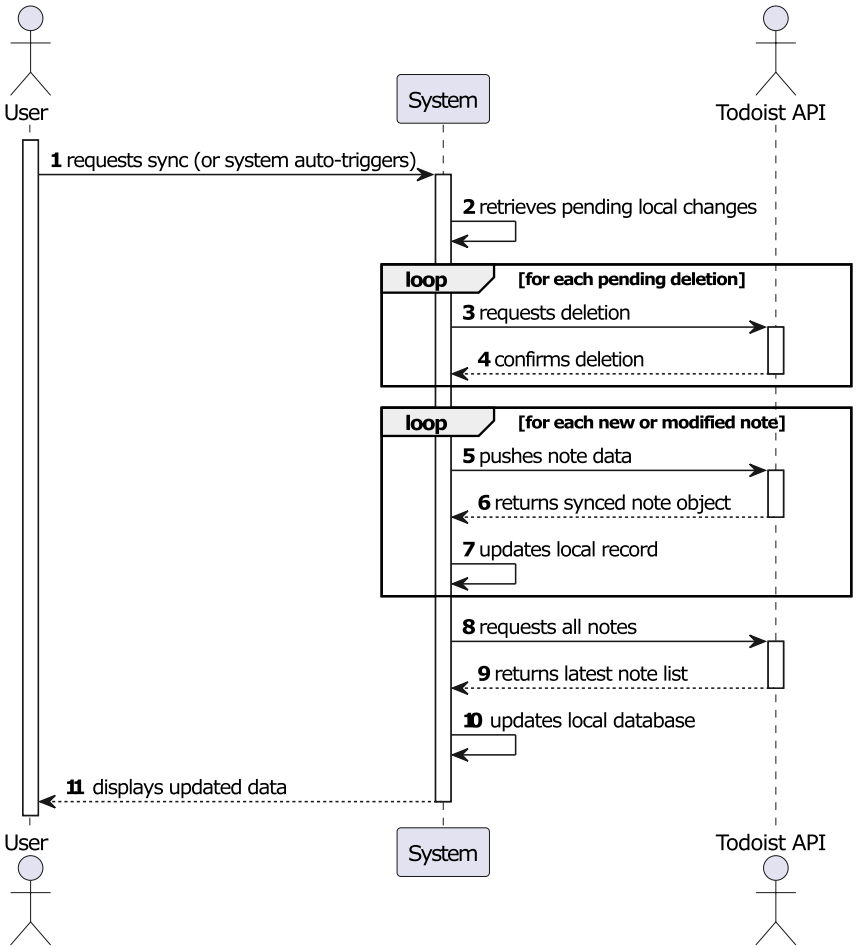

Figure 2.11: System Sequence Diagram (UC09)

Chapter 3

Application Architecture and Design

3.1 Physical Architecture

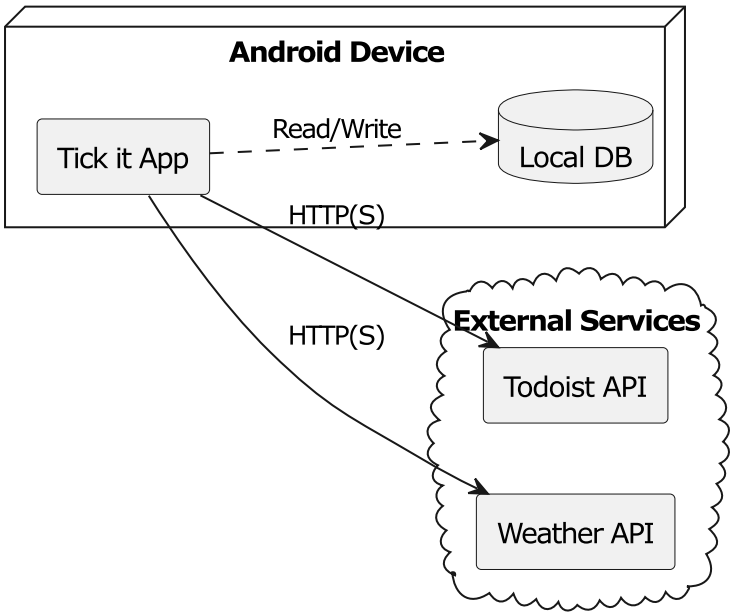

Figure 3.1: Deployment Diagram

3.2 Code Structure

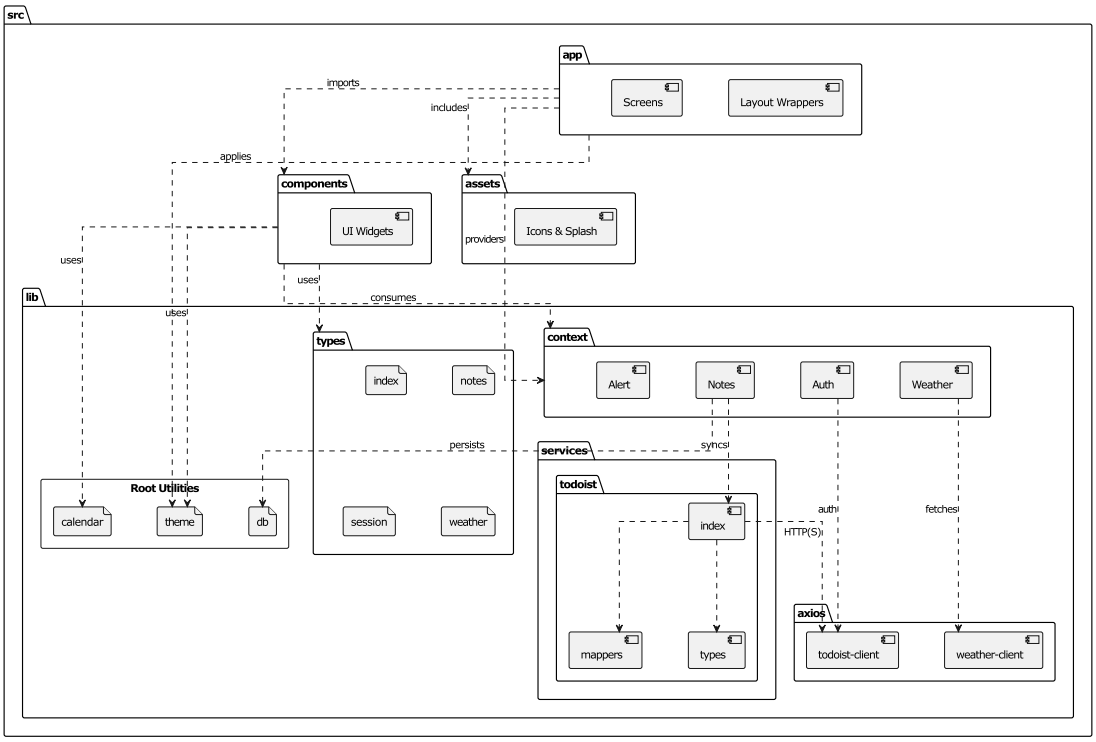

Figure 3.2: Package Diagram

3.3 Sequence Diagrams

3.3.1 UC01 - Unlock Application

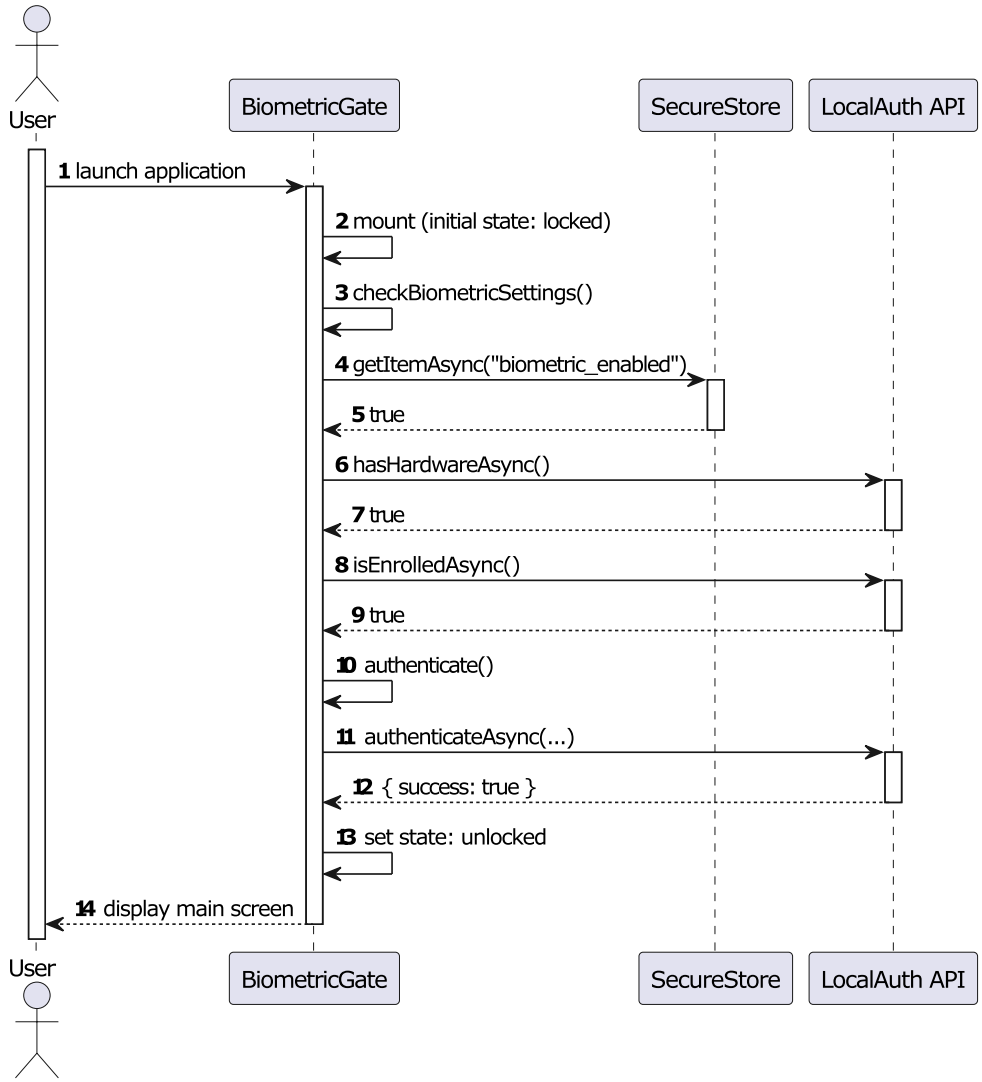

Figure 3.3: Sequence Diagram (UC01)

3.3.2 UC02 - View Notes

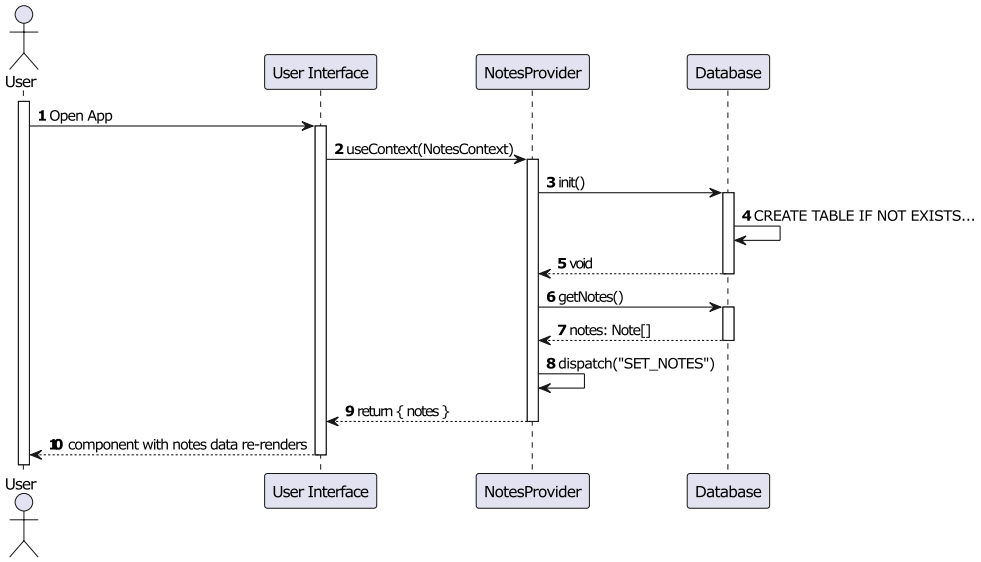

Figure 3.4: Sequence Diagram (UC02)

3.3.3 UC03 - Create Note

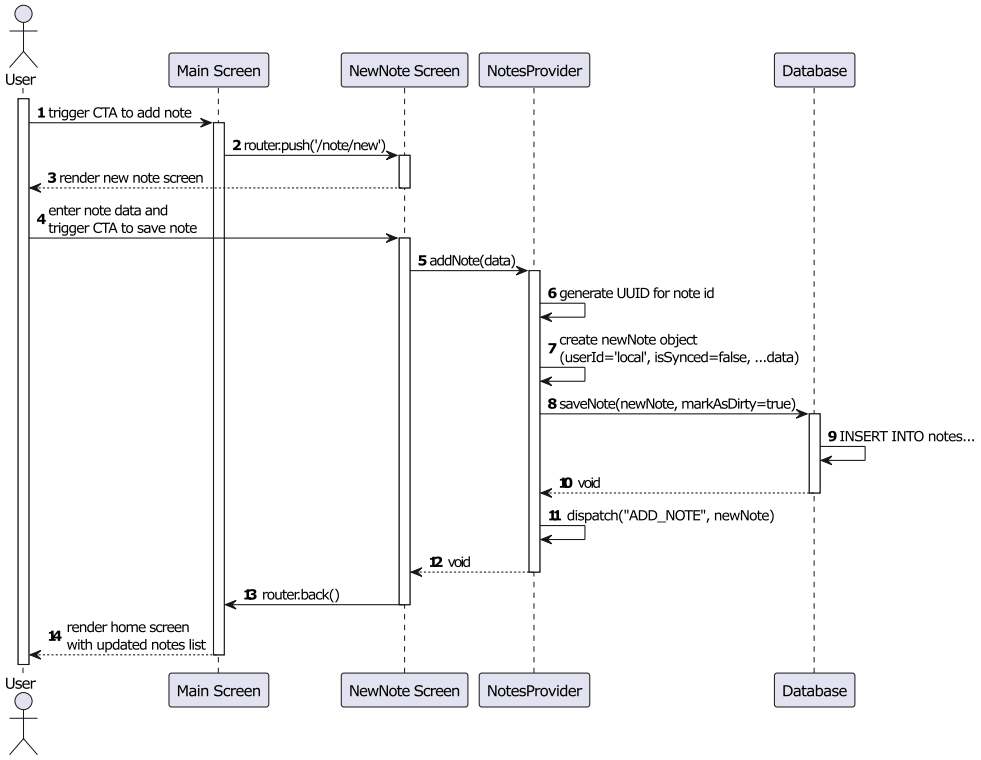

Figure 3.5: Sequence Diagram (UC03)

3.3.4 UC04 - Edit Note

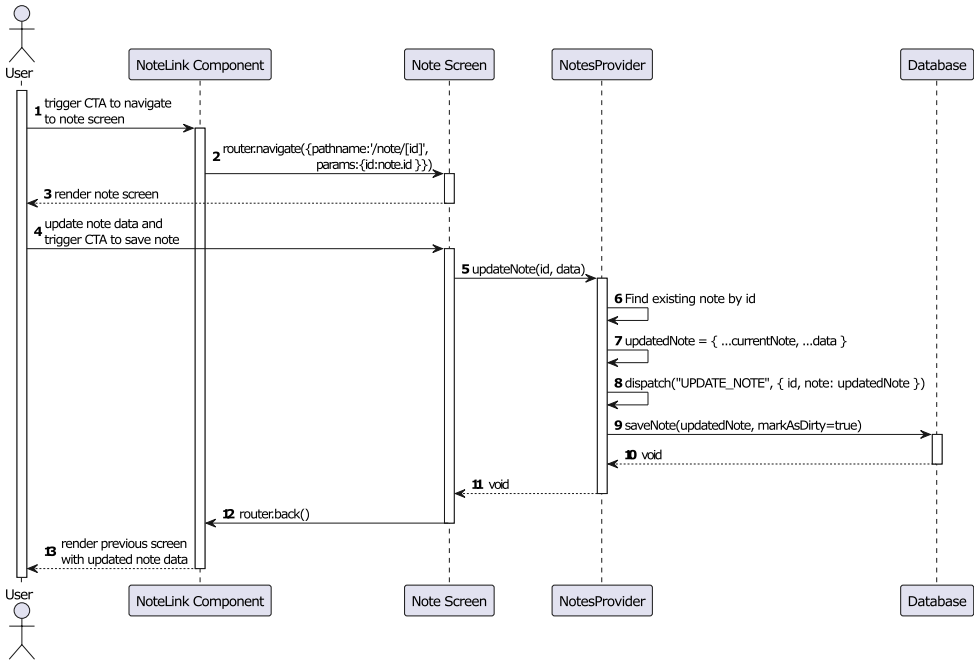

Figure 3.6: Sequence Diagram (UC04)

3.3.5 UC05 - Delete Note

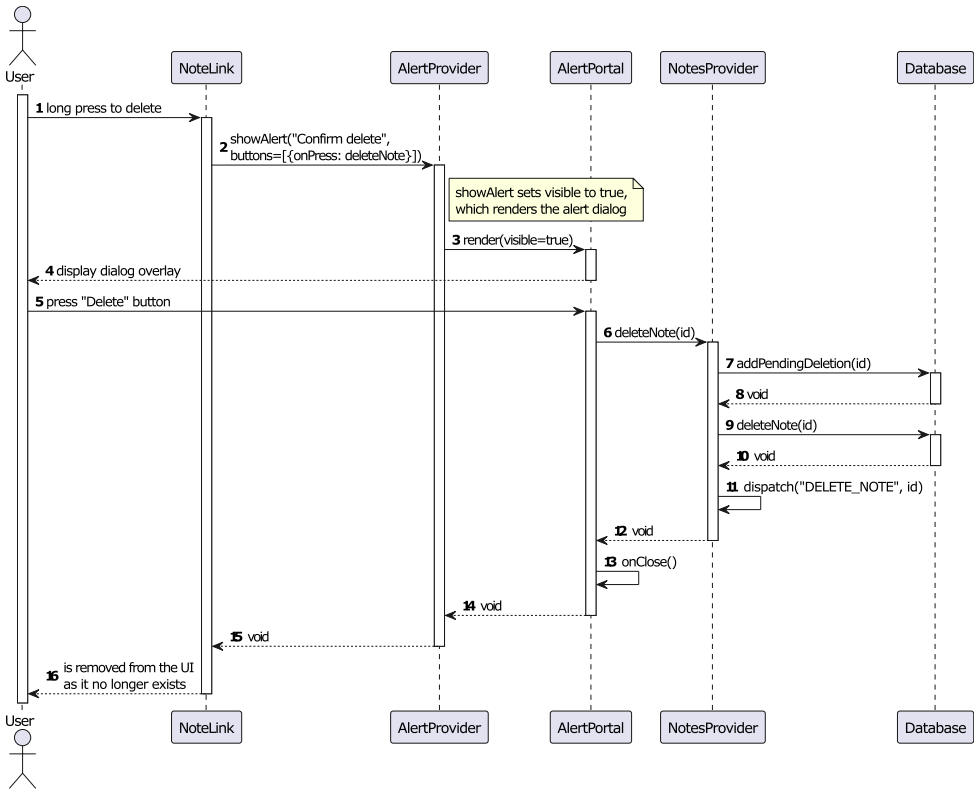

Figure 3.7: Sequence Diagram (UC05)

3.3.6 UC06 - Log into Todoist

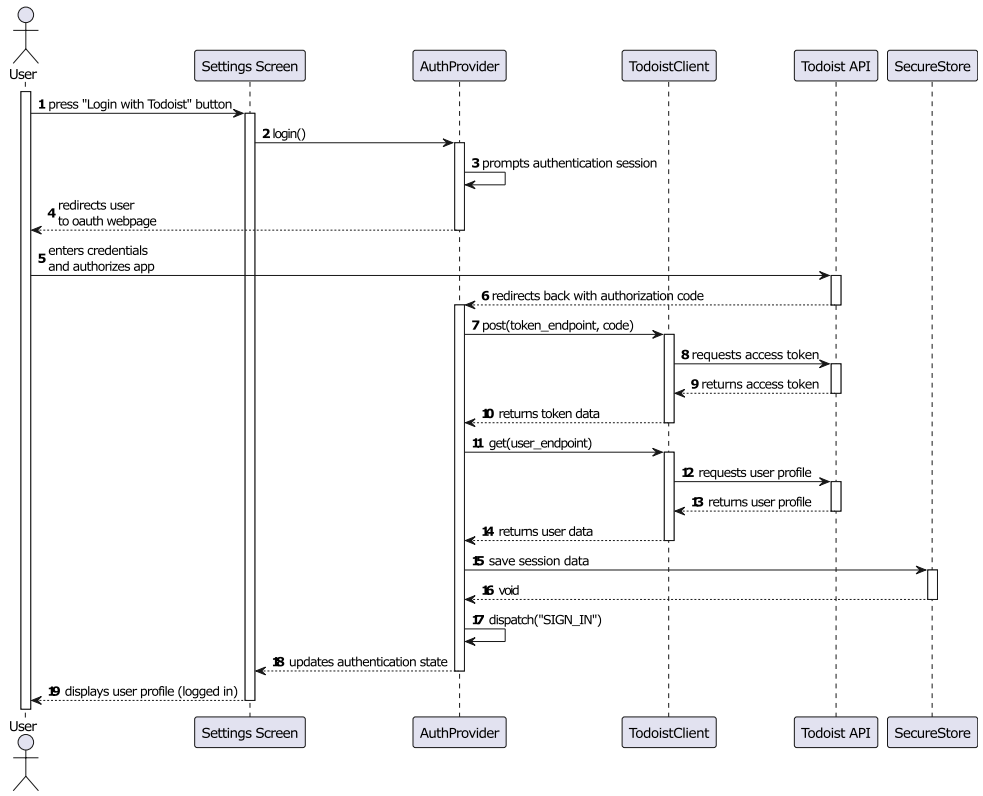

Figure 3.8: Sequence Diagram (UC06)

3.3.7 UC07 - Log out

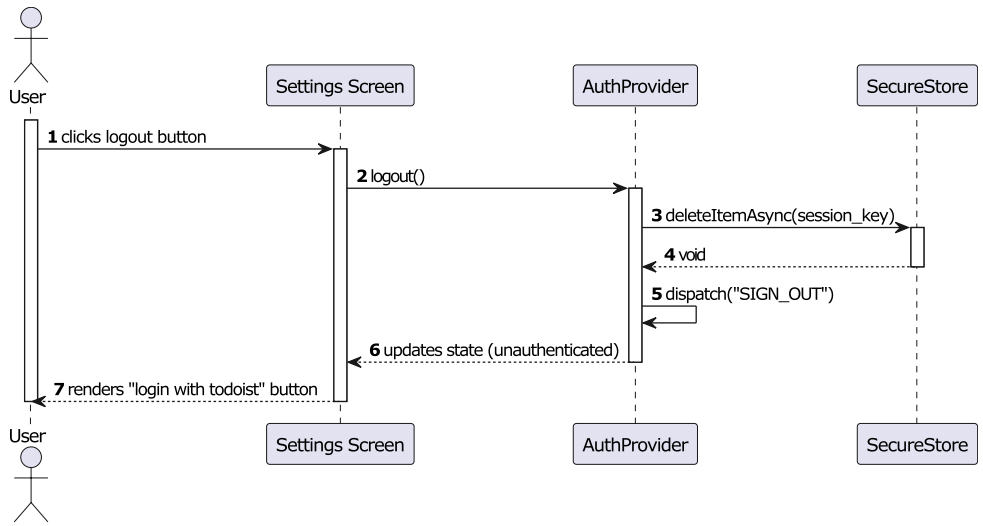

Figure 3.9: Sequence Diagram (UC07)

3.3.8 UC08 - View Weather

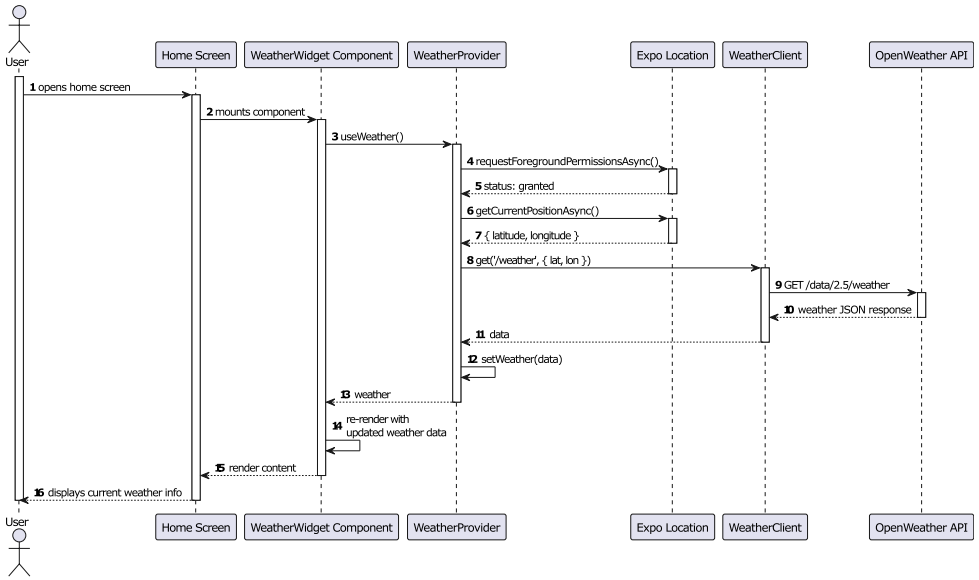

Figure 3.10: Sequence Diagram (UC08)

3.3.9 UC09 - Sync Data

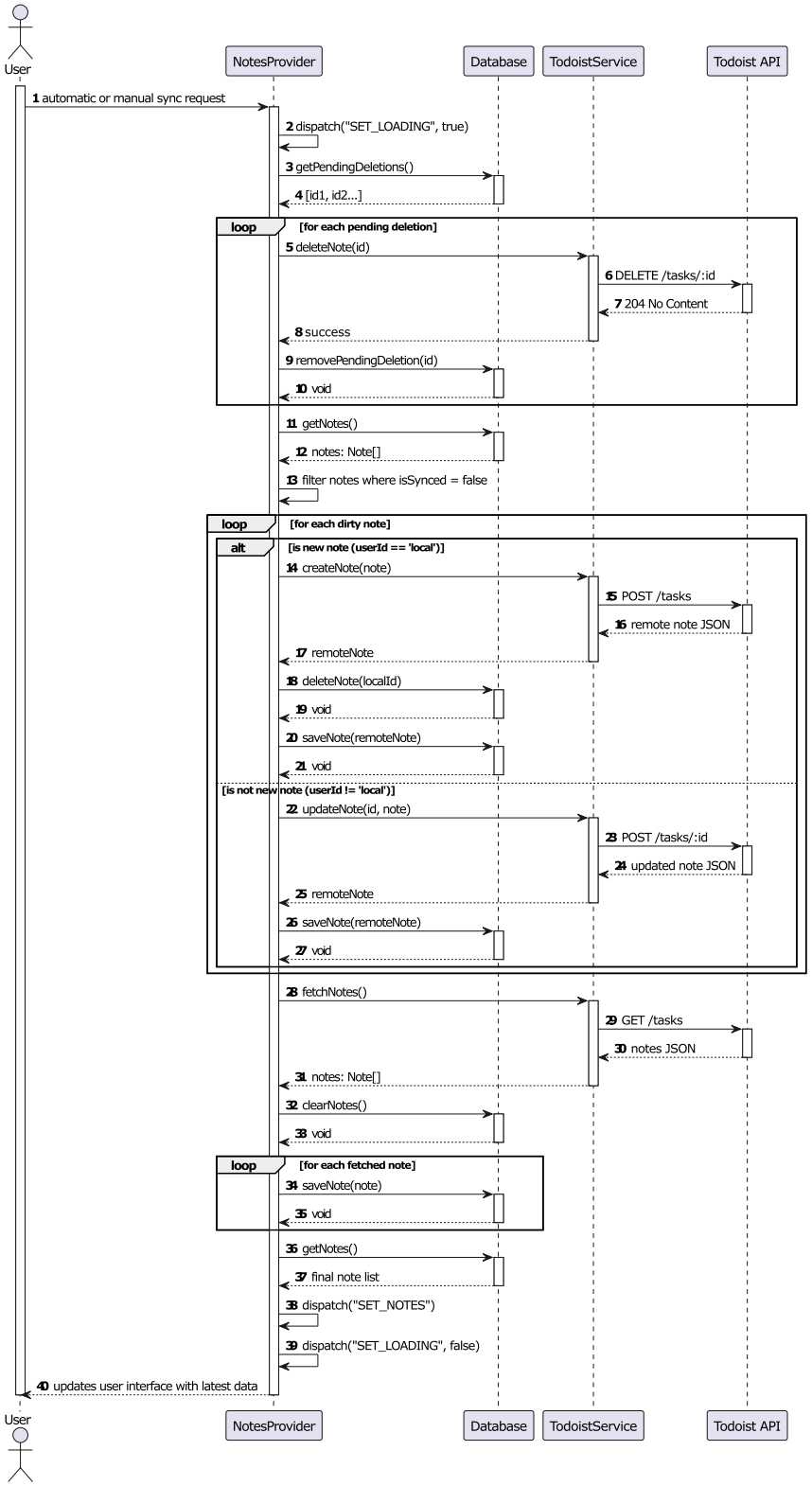

Figure 3.11: Sequence Diagram (UC09)

3.4 User Interface

The user interface of **Tick it (Revamped)** is designed utilizing high-contrast elements to ensure readability while maintaining a modern aesthetic. The navigation is handled via a persistent bottom tab bar, allowing quick access to the four core screens: Calendar, Home, Statistics, and Settings.

3.4.1 Home and Calendar

The application's landing screen is the **Home**, which provides the user with an immediate overview of the current weather, a weekly calendar strip (pure UI, no functionality), and the list of notes.

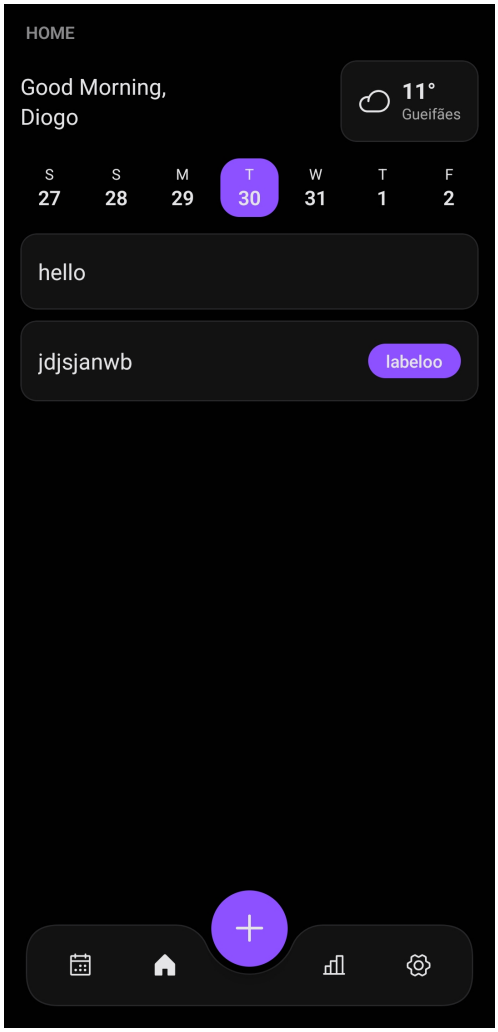

Figure 3.12: Home Screen

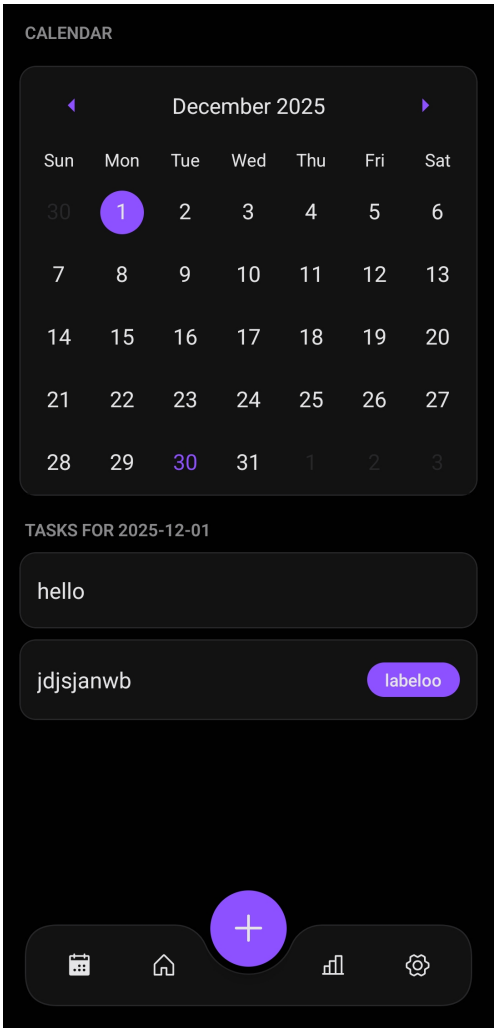

Figure 3.13: Calendar Screen

Figure 3.14: Main Screens

As shown in Figure 3.13, the **Calendar** screen allows users to view notes scheduled for specific dates. The interface provides visual cues to highlight the selected date and the current day.

3.4.2 Productivity Statistics

To encourage user engagement and synchronization with Todoist's services, the application includes a **Statistics** screen (Figure 3.16). This screen visualizes the user's "Karma" points and daily task completion streaks, retrieving data directly from the Todoist API when the user is logged in.

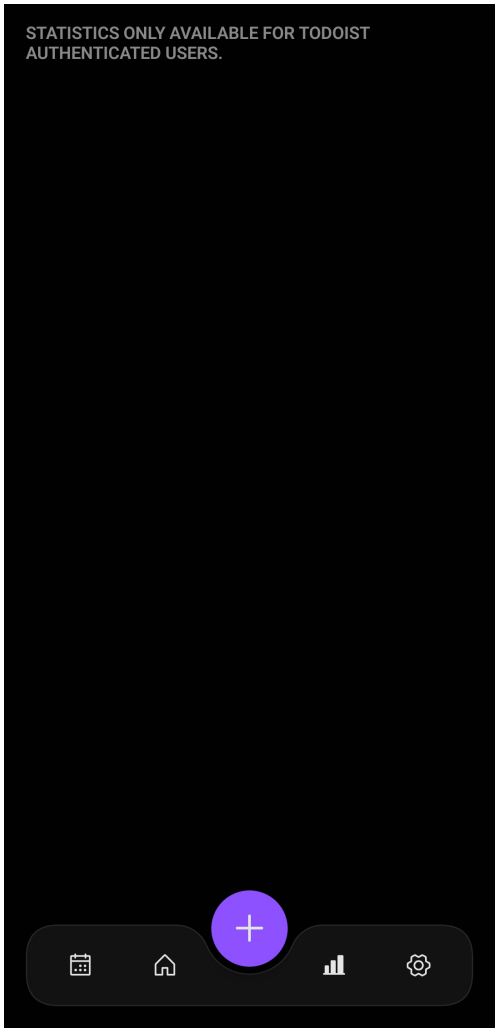

Figure 3.15: Statistics Screen (logged out)

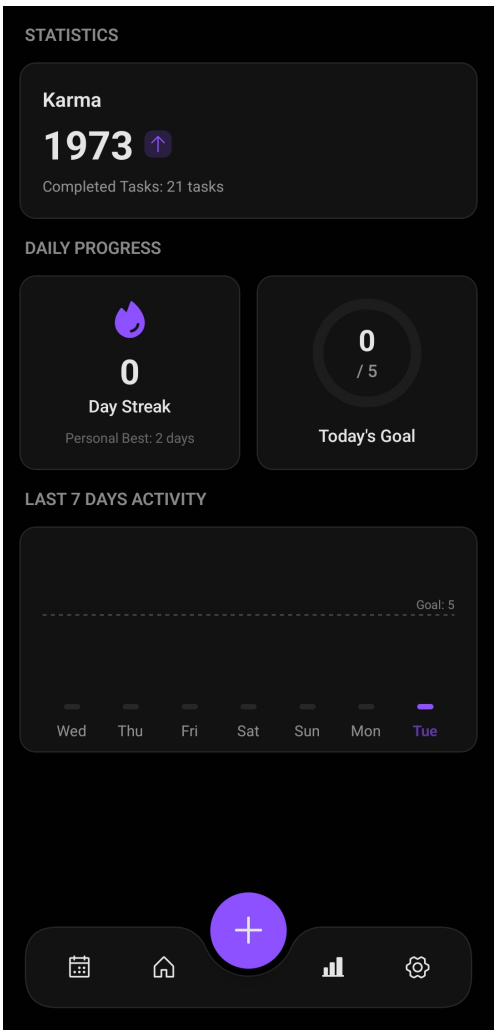

Figure 3.16: Statistics Screen (logged in)

Figure 3.17: Settings Screens

3.4.3 Settings and Security

The settings interface manages the connection to external services and local security preferences. A key feature is the optional **Biometric Lock**, which users can toggle on or off.

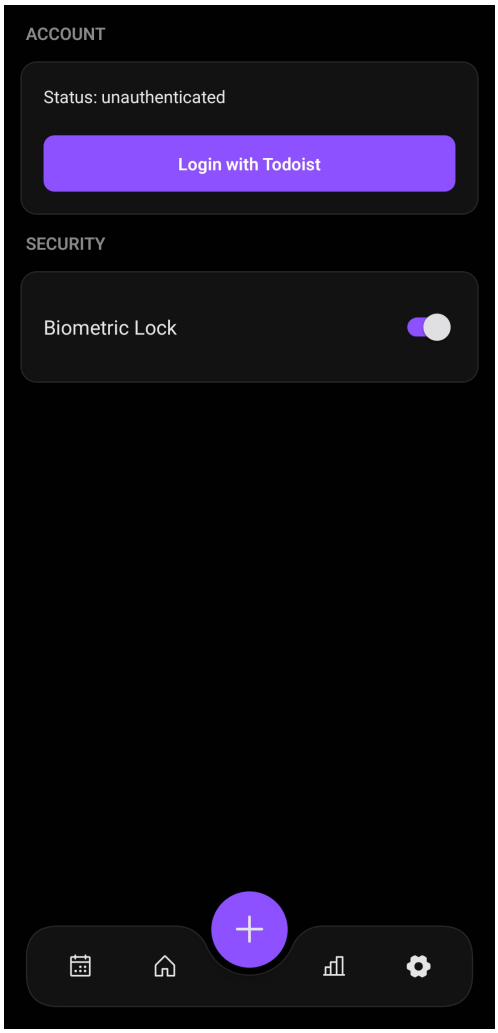

Figure 3.18: Settings Screen (logged out)

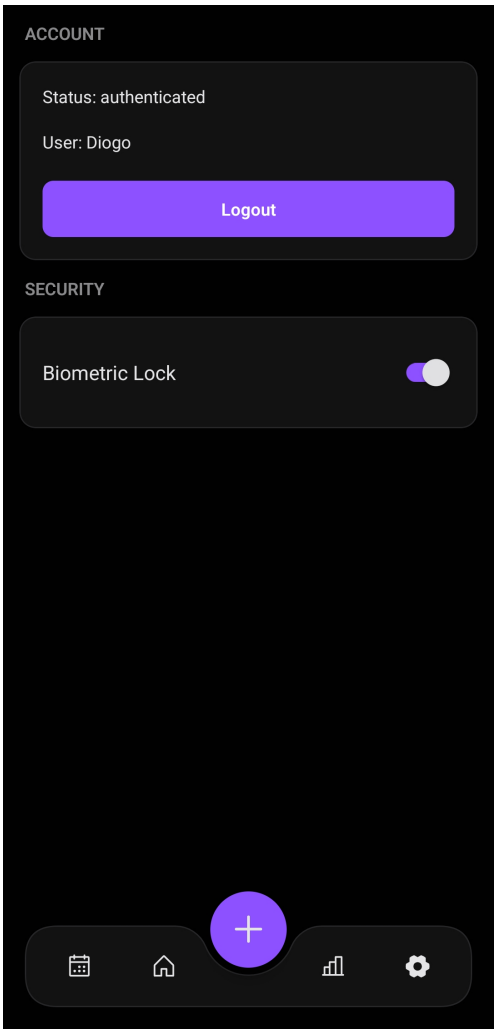

Figure 3.19: Settings Screen (logged in)

Figure 3.20: Settings Screens

Figure 3.18 shows the interface when the user is disconnected from Todoist, presenting a clear call-to-action. Once authenticated (Figure 3.19), the user sees their profile status and the option to log out. The "Biometric Lock" switch utilizes the device's native hardware (FaceID/Fingerprint) to secure the app upon future launches.

3.4.4 Note Management

The core functionality of this project revolves around the creation and manipulation of notes. This flow is handled by two screens: the **New Note** screen and the **Note Editor** screen.

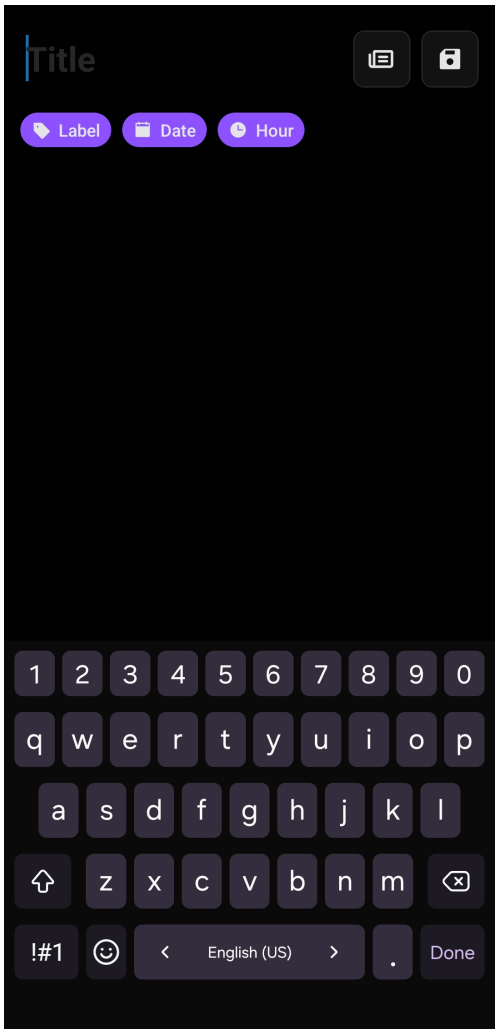

Figure 3.21: New Note Screen

Figure 3.22: Note Edit Screen

Figure 3.23: Note Management Interfaces

The **creation** interface (Figure 3.21) is designed to be distraction-free. It auto-focuses the title field upon opening, allowing users to type immediately.

The **editing** interface (Figure 3.22) corresponds to the route `/note/[id]`. This screen fetches the specific note data from the database based on the ID passed through the navigation parameters.

Chapter 4

Implementation

4.1 Offline-First Synchronization Strategy

To guarantee data availability regardless of network status, the application utilizes `expo-sqlite` as the single source of truth for the UI. The database connection is established asynchronously during the application's launch.

The synchronization mechanism (Listing 4.1) is the function that is called after every database operation that follows a "local-first" pattern:

1. **Local Commit:** User actions are immediately committed to the local database for instant feedback.
2. **Background Sync:** A function triggers when network connectivity is detected (`isOnline`).
3. **Remote Reconciliation:** It pushes pending deletions and "dirty" (unsynced) notes to Todoist, then performs a full fetch to ensure local consistency.

```
const runBackgroundSync = useCallback(async () => {
    if (!isOnline) return

    try {
        dispatch({ type: 'SET_LOADING', payload: true })
        const pendingDeletions = db.getPendingDeletions()
        for (const id of pendingDeletions) {
            await TodoistService.deleteNote(id)
                .catch(() => {})
                .finally(() => db.removePendingDeletion(id))
        }

        const dirtyNotes = db.getNotes().filter(n => !n.isSynced)

        for (const note of dirtyNotes) {
            try {
                dispatch({ type: 'SET_LOADING', payload: true })
                if (note.userId === 'local') {
                    const remoteNote = await TodoistService.createNote(
   note)
                    db.deleteNote(note.id)
                    db.saveNote(remoteNote, false)
                } else {
                    const remoteNote = await TodoistService.updateNote(
   note.id, note)
                    db.saveNote(remoteNote, false)
                }
            } catch (error) {
                console.error('Failed to push note', note.id, error)
            } finally {
                dispatch({ type: 'SET_LOADING', payload: false })
            }
        }

        const remoteNotes = await TodoistService.fetchNotes()

        db.clearNotes()
        remoteNotes.forEach(note => db.saveNote(note, false))

        dispatch({ type: 'SET_NOTES', payload: db.getNotes() })
    } catch (e) {
        console.error('Sync Error', e)
    } finally {
        dispatch({ type: 'SET_LOADING', payload: false })
    }
}, [isOnline])
```

Listing 4.1: Synchronization function

Chapter 5

Tests

5.1 Testing Methodology

Given the project's scope and the heavy emphasis on user interface interaction, the testing strategy focused on **Manual Black-Box Testing**. This approach involves validating the application's functionality without inspecting the internal code structure, simulating the behavior of a real end-user.

The testing phase was divided into two categories:

- **Functional Testing:** Verifying that specific features (e.g., saving a note, syncing) produce the expected results.
- **Usability Testing:** Evaluating the responsiveness, navigation flow, and visual feedback of the interface.

5.2 Functional Validation

The following test scenarios were executed manually on a physical device (Android) to validate the core requirements.

Table 5.1: Manual Test Cases

Feature	**Action Performed**	**Result**
Offline Mode	Created a note while Airplane Mode was active, then restarted the app.	Pass (Note persisted locally)
Sync	Disabled Airplane Mode and pulled down to refresh.	Pass (Note appeared on Todoist)
Biometrics	Enabled "Biometric Lock" in settings and relaunched the app.	Pass (Prompt appeared immediately)
Navigation	Navigated deep into a note detail and pressed the "Back" hardware button.	Pass (Returned to list correctly)
Validation	Attempted to save a note with an empty title.	Pass (Save button disabled)

5.3 Device Compatibility

The application was manually deployed and tested on the following environments to ensure stability:

- **Development Environment:** No emulators were used.
- **Physical Devices:**
  - Samsung Galaxy S25 Ultra (Android 16)
  - Xiaomi 14T Pro (Android 16)

No critical crashes were observed during the final stability tests.

Chapter 6

Conclusions

6.1 Summary of Achievements

The **Tick it (Revamped)** project successfully delivered a functional, offline-first mobile application that integrates modern mobile capabilities. The primary goal of creating a unified space for tasks (Todoist) and personal notes was achieved. The application runs stably, fulfilling the core requirements of the DSSMV course.

6.2 Limitations

Despite the success, there are known limitations:

- **Sync Conflicts:** Currently, if a note is edited on two devices simultaneously, the last update simply overwrites the previous one.
- **Media Support:** The notes currently support text only; multimedia attachments (images/voice) were scoped out due to time constraints.

6.3 Future Work

For future iterations of the product, the following features are proposed:

1. **Push Notifications:** Reminders for notes with due dates set.