DSSMV – Project React Report

Tiago Caetano – 1241945 and Ruben Remelhe – 1241847

Instituto Superior de Engenharia do Porto - ISEP
Licenciatura em Engenharia de Telecomunicações e Informática - LETI
DSSMV – Desenvolvimento de Software para Dispositivos Móveis

Abstract. PingMe2 is a mobile application designed to support personal task management through reminders, lists, and categories. The application emphasizes state consistency, modular architecture, and a clear separation between presentation and business logic. This report describes the system objectives, architectural decisions, and the implementation strategies adopted during development.

Key words: Mobile Application, Task Management, React Native, Redux

1 Context

This project consists of the development of an Android mobile application named PingMe2, whose main goal is to support users in managing daily tasks in an organized and efficient manner. The application provides a digital task and reminder management system that enables users to create, organize, and monitor their activities over time.

PingMe2 follows a client–server architecture and interacts with a RESTful API responsible for all data management operations. Through this interaction, the application supports the full set of CRUD operations, allowing users to create, retrieve, update, and delete reminders, categories, and lists. This architectural approach ensures a clear separation of concerns between the mobile client and the backend services.

The primary objective of PingMe2 is to help users structure their daily routines by offering intuitive task organization mechanisms, including categorization, list grouping, completion tracking, and filtering. By combining a mobile user interface with a RESTful backend, the project demonstrates a modern software development approach commonly adopted in contemporary mobile applications.

2 Analysis

2.1 Domain Model

The domain model presents the main entities of the PingMe2 system and their relationships, providing a conceptual view of the application's data structure. It defines the core elements involved in task management, such as users, reminders,

categories, and lists, as well as the associations between them. This model serves as the foundation for the system design and implementation, ensuring consistency between requirements, business logic, and data representation throughout the application.

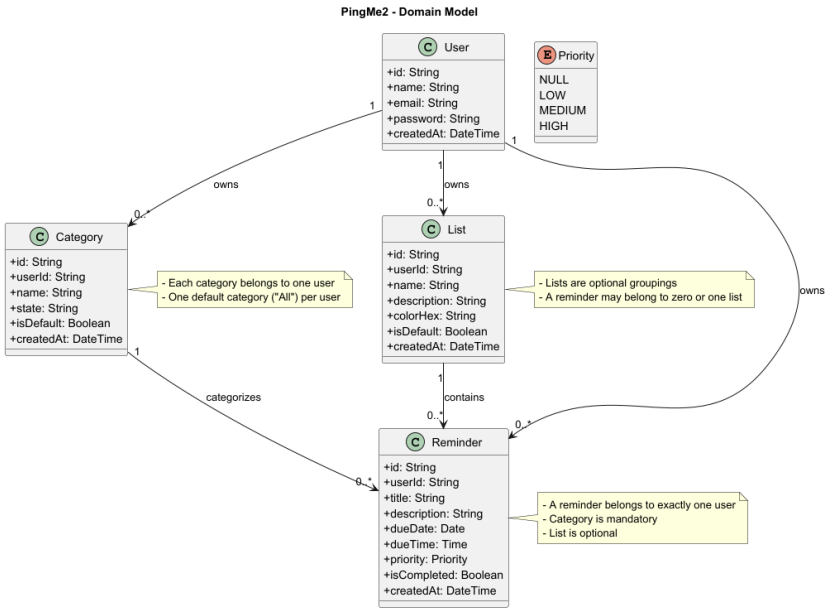

Fig. 1. Domain Model

2.2 Non-functional Requirements

This section describes the non-functional requirements of the PingMe2 application. These requirements define quality attributes and constraints related to performance, usability, security, reliability, and maintainability of the system.

Usability

- **NFR-01**: The system shall provide an intuitive and user-friendly mobile interface suitable for daily use.
- **NFR-02**: The system shall ensure that common operations can be performed with a minimal number of user interactions.
- **NFR-03**: The system shall provide clear visual feedback for user actions, including loading, success, and error states.

Performance

- **NFR-04**: The system shall respond to user interactions within an acceptable time frame under normal operating conditions.
- **NFR-05**: The system shall handle multiple user requests without significant degradation of performance.

Reliability and Availability

- **NFR-06**: The system shall maintain consistent behavior during normal operation and recover gracefully from errors.
- **NFR-07**: The system shall provide appropriate error handling and recovery mechanisms in case of network or service failures.

Security

- **NFR-08**: The system shall restrict access to data based on user authentication and authorization.
- **NFR-09**: The system shall ensure that user data is securely transmitted between the client and the backend services.

Maintainability and Extensibility

- **NFR-10**: The system shall follow a modular architecture to facilitate maintenance and future extensions.
- **NFR-11**: The system shall allow new features to be added with minimal impact on existing functionality.

Portability

- **NFR-12**: The system shall be deployable on Android devices supporting modern versions of the operating system.

2.3 Functional Requirements

This section describes the functional requirements of the PingMe2 application. Each requirement specifies a behavior or capability that the system must provide in order to support task, list, and category management for authenticated users.

User Authentication

- **FR-01**: The system shall allow users to register using an email and password.
- **FR-02**: The system shall allow users to log in using valid credentials.
- **FR-03**: The system shall allow users to log out, clearing all session-related data.

User Profile Management

- **FR-05**: The system shall associate all data (categories, lists, reminders) with a single authenticated user.
- **FR-06**: The system shall prevent users from accessing data belonging to other users.

Category Management

- **FR-07**: The system shall allow users to create custom categories.
- **FR-08**: The system shall allow users to edit existing categories.
- **FR-09**: The system shall allow users to delete categories.
- **FR-10**: The system shall always provide a default category (e.g., "All") that cannot be deleted.
- **FR-11**: The system shall allow one active category to be selected at a time.
- **FR-12**: The system shall filter reminders based on the currently active category.

List Management

- **FR-13**: The system shall allow users to create lists.
- **FR-14**: The system shall allow users to edit list details, including name, description, and color.
- **FR-15**: The system shall allow users to delete lists.
- **FR-16**: The system shall display all reminders associated with a selected list.
- **FR-17**: The system shall allow reminders to belong to zero or one list.

Reminder Management

- **FR-18**: The system shall allow users to create reminders.
- **FR-19**: The system shall allow users to edit reminder details, including title, description, date, time, priority, location, and images.
- **FR-20**: The system shall allow users to delete reminders.
- **FR-21**: The system shall allow users to mark reminders as completed or not completed.
- **FR-22**: The system shall allow reminders to be assigned to exactly one category.
- **FR-23**: The system shall allow reminders to be optionally assigned to one list.

Reminder Filtering and Organization

- **FR-24**: The system shall display reminders due on the current day.
- **FR-25**: The system shall display completed reminders separately.
- **FR-26**: The system shall allow reminders to be filtered by category.
- **FR-27**: The system shall allow reminders to be grouped by list.

Priority Management

- **FR-28**: The system shall support multiple reminder priority levels (None, Low, Medium, High).
- **FR-29**: The system shall visually distinguish reminders based on their priority level.

Data Synchronization and Persistence

- **FR-30**: The system shall persist user data remotely using a backend service.
- **FR-31**: The system shall synchronize reminders, categories, and lists across devices.
- **FR-32**: The system shall handle loading and error states during data synchronization.

Navigation and Usability

- **FR-33**: The system shall provide navigation between Home, Lists, Categories, and Reminder creation screens.
- **FR-34**: The system shall provide feedback messages for successful and failed operations.
- **FR-35**: The system shall prevent invalid operations, such as creating reminders without required fields.

2.4 System Sequence Diagrams (SSDs)

This section presents the System Sequence Diagrams (SSDs) for the main use cases of the PingMe2 application. Each use case is documented by its specification table followed by the corresponding SSD.

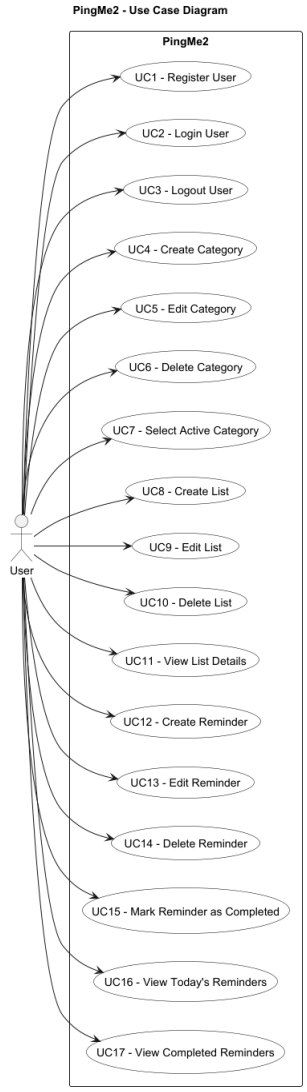

Fig. 2. Use Case Diagram for PingMe2 Application

Table 1. Use Case UC1 – Register User

Field	**Description**
Description	Allows a new user to create an account in the system (Create operation).
Pre-Conditions	The user is not authenticated and has an active internet connection.
Post-Conditions	A new user account is created and stored in the database.
Basic Flow	1. The user enters registration data (name, email, password); 2. The user confirms the registration; 3. The system validates the provided data; 4. The system sends the data to the backend API; 5. The system confirms success and creates the account.
Alternative Flow	Invalid or incomplete data: the system rejects the request and displays an error message.

UC1 – Register User

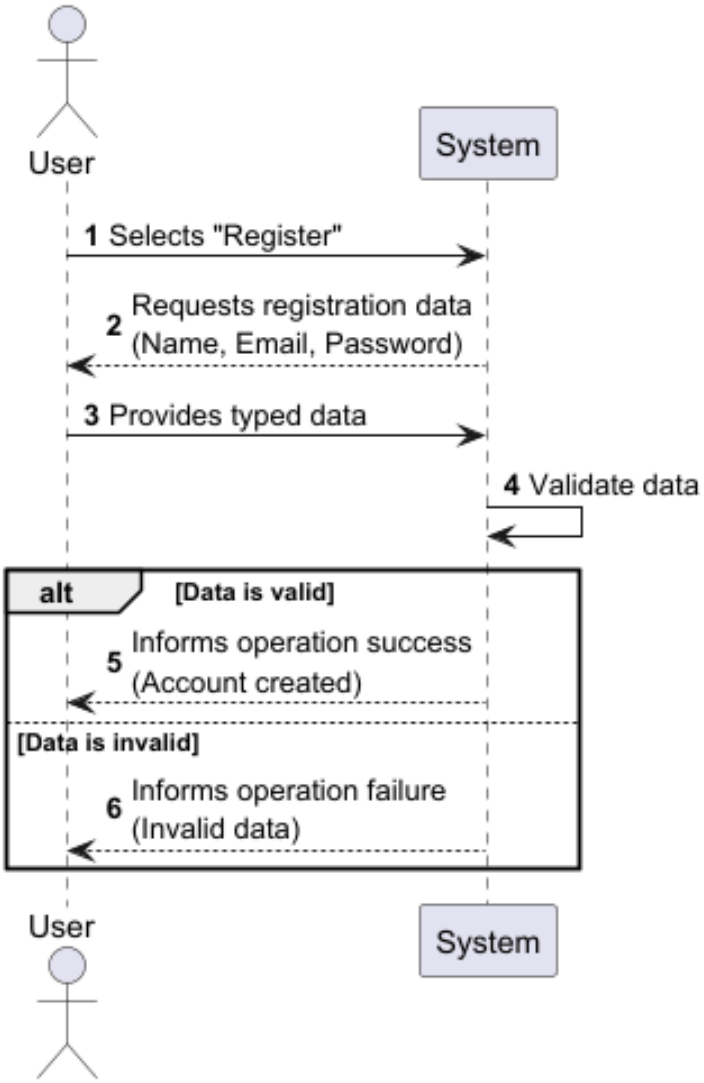

Fig. 3. System Sequence Diagram for UC1 – Register User

Table 2. Use Case UC2 – Login User

Field	**Description**
Description	Authenticates a user in the system (Read operation).
Pre-Conditions	The user has a registered account and an active internet connection.
Post-Conditions	The user is authenticated and gains access to the application.
Basic Flow	1. The user enters email and password; 2. The user submits the login form; 3. The system validates the credentials; 4. The system authenticates the user via the backend API; 5. The system grants access to the application.
Alternative Flow	Invalid credentials: the system denies access and displays an error message.

UC2 – Login User

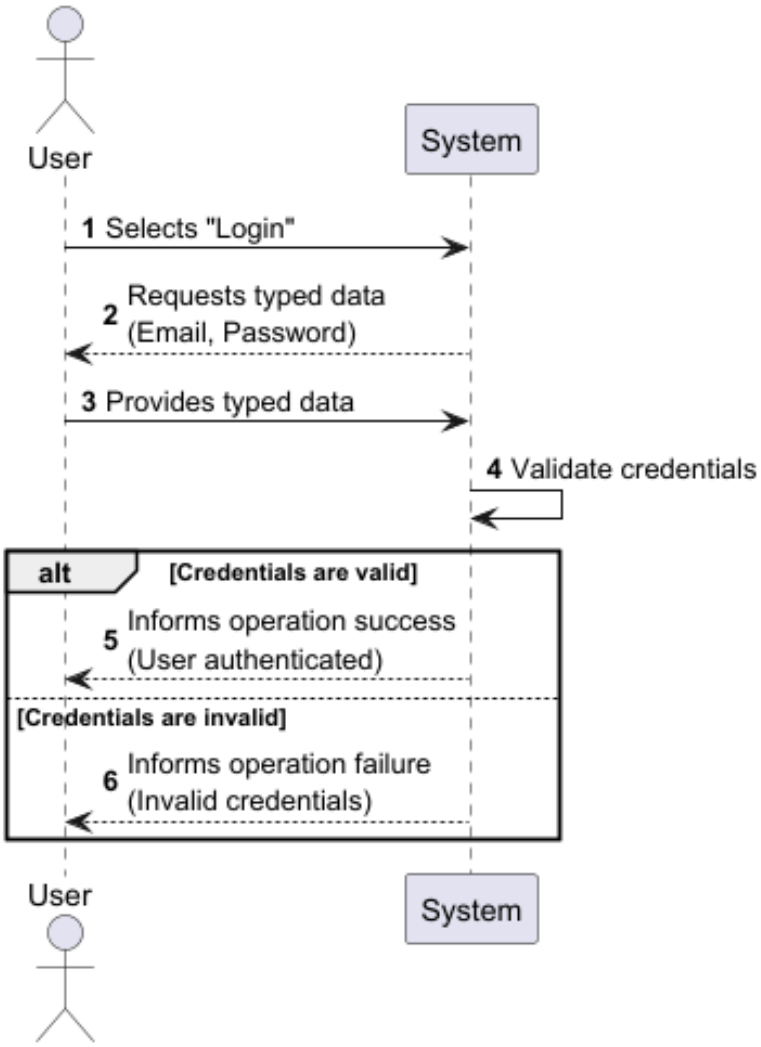

Fig. 4. System Sequence Diagram for UC2 – Login User

Table 3. Use Case UC3 – Logout User

Field	**Description**
Description	Ends the user session (Update operation).
Pre-Conditions	The user is authenticated.
Post-Conditions	The user session is terminated.
Basic Flow	1. The user selects the logout option; 2. The system clears session-related data; 3. The system redirects the user to the login screen.
Alternative Flow	Not applicable.

UC3 – Logout User

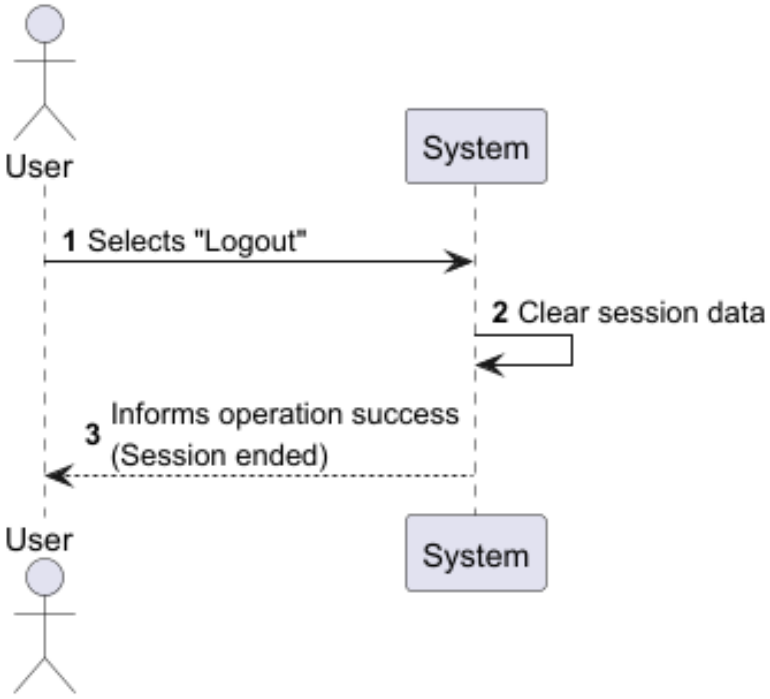

Fig. 5. System Sequence Diagram for UC3 – Logout User

Table 4. Use Case UC5 – Edit Category

Field	**Description**
Description	Updates an existing category (Update operation).
Pre-Conditions	The user is authenticated and the category exists.
Post-Conditions	The category information is updated.
Basic Flow	1. The user selects a category; 2. The user edits the category details; 3. The system validates the data; 4. The system updates the category via the backend API; 5. The system confirms success.
Alternative Flow	Invalid data: the system rejects the update.

UC5 – Edit Category

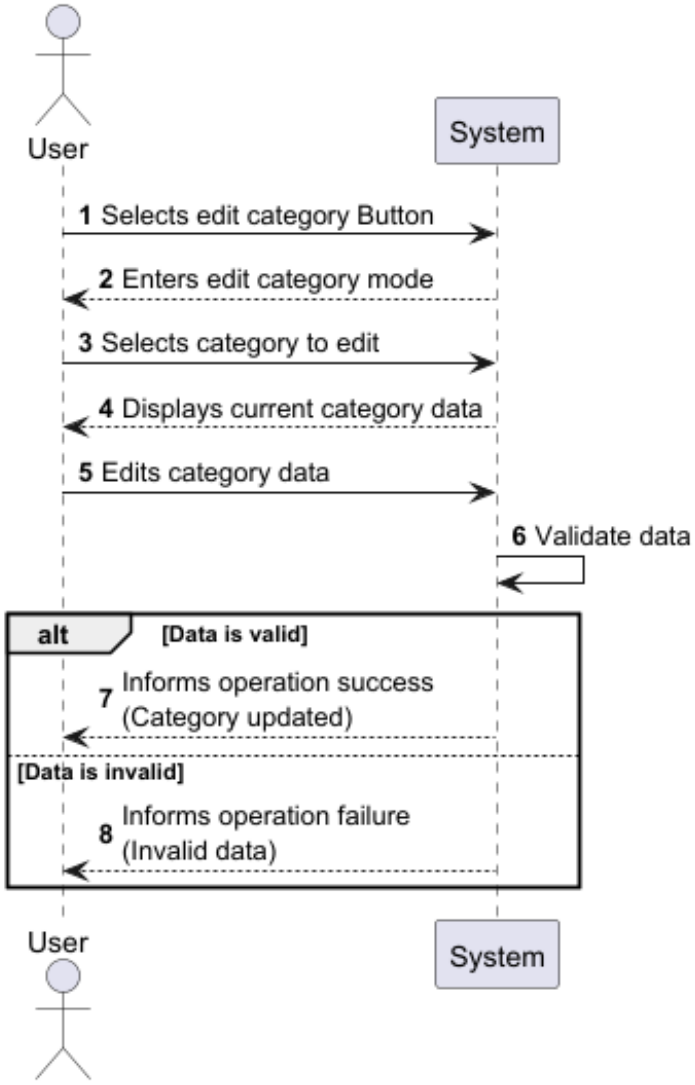

Fig. 6. System Sequence Diagram for UC5 – Edit Category

Table 5. Use Case UC7 – Select Active Category

Field	Description
Description	Sets a category as the active filter (Update operation).
Pre-Conditions	The user is authenticated and categories exist.
Post-Conditions	The selected category becomes the active filter.
Basic Flow	1. The user selects a category; 2. The system marks the category as active; 3. The system updates the displayed reminders.
Alternative Flow	Category not found: the system keeps the previous selection.

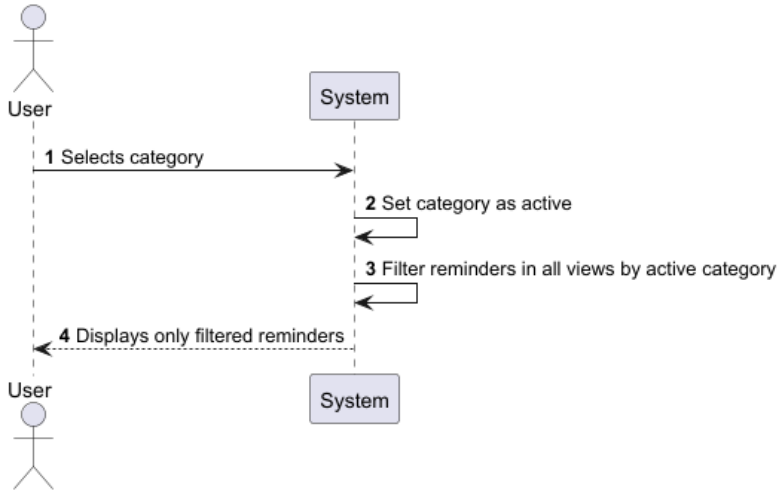

Fig. 7. System Sequence Diagram for UC7 – Select Active Category

UC7 – Select Active Category

Table 6. Use Case UC8 – Create List

Field	**Description**
Description	Creates a new list to group reminders (Create operation).
Pre-Conditions	The user is authenticated and has an active internet connection.
Post-Conditions	A new list is added to the system.
Basic Flow	1. The user enters list details; 2. The user confirms creation; 3. The system validates the data; 4. The system stores the list via the backend API; 5. The system confirms success.
Alternative Flow	Invalid data: the system rejects the creation.

UC8 – Create List

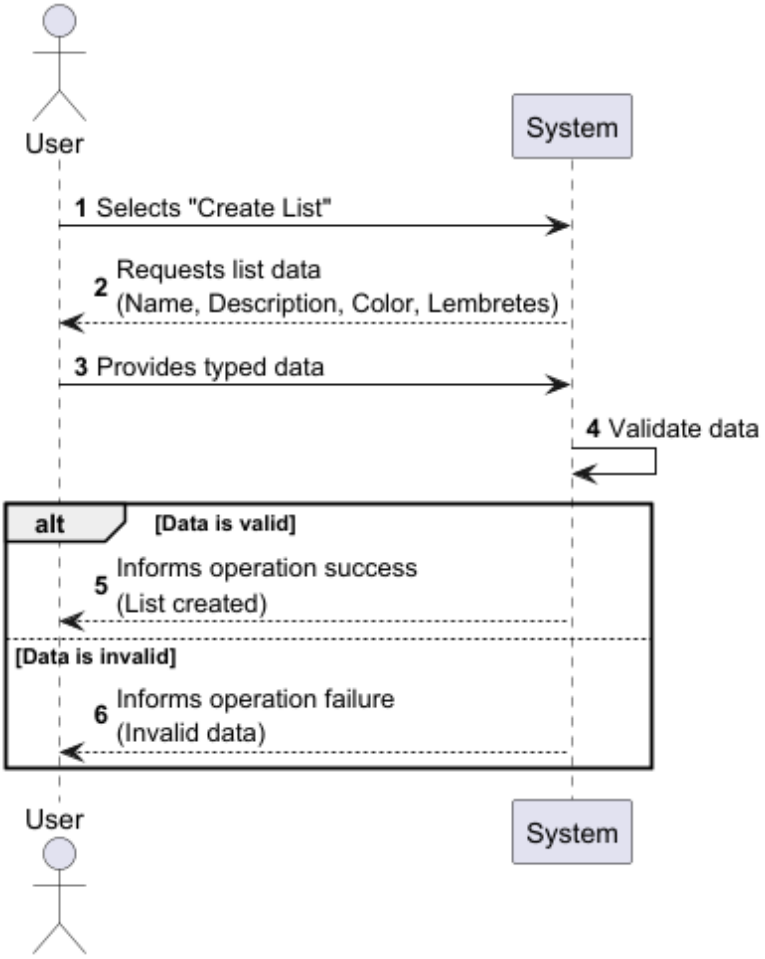

Fig. 8. System Sequence Diagram for UC8 – Create List

Table 7. Use Case UC10 – Delete List

Field	**Description**
Description	Removes an existing list from the system (Delete operation).
Pre-Conditions	The user is authenticated and the list exists.
Post-Conditions	The list is removed from the system.
Basic Flow	1. The user selects a list; 2. The user confirms the deletion; 3. The system deletes the list via the backend API; 4. The system updates the user interface.
Alternative Flow	Deletion cancelled: no changes are made.

UC10 – Delete List

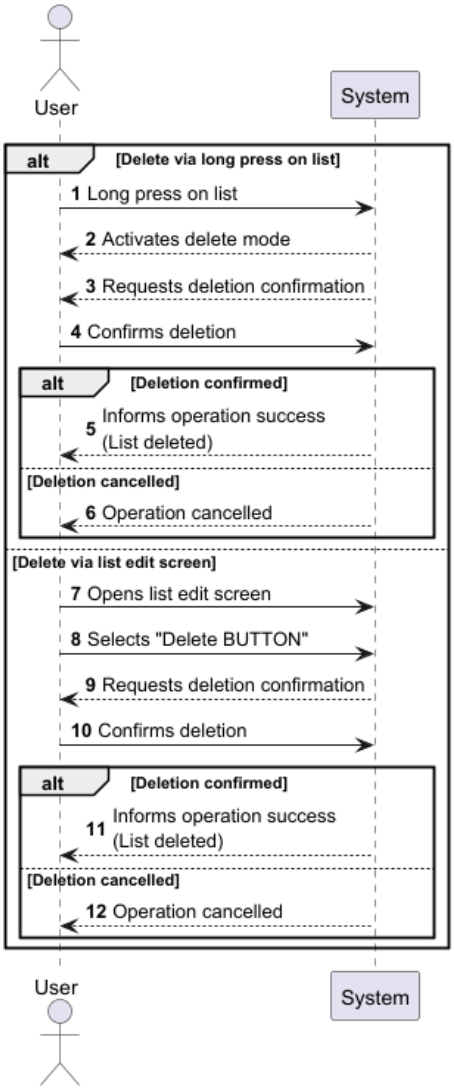

Fig. 9. System Sequence Diagram for UC10 – Delete List

Table 8. Use Case UC12 – Create Reminder

Field	**Description**
Description	Creates a new reminder in the system (Create operation).
Pre-Conditions	The user is authenticated and has an active internet connection.
Post-Conditions	A new reminder is added to the system.
Basic Flow	1. The user enters reminder details; 2. The user submits the form; 3. The system validates the data; 4. The system stores the reminder via the backend API; 5. The system confirms success.
Alternative Flow	Invalid data: the system displays an error message.

UC12 – Create Reminder

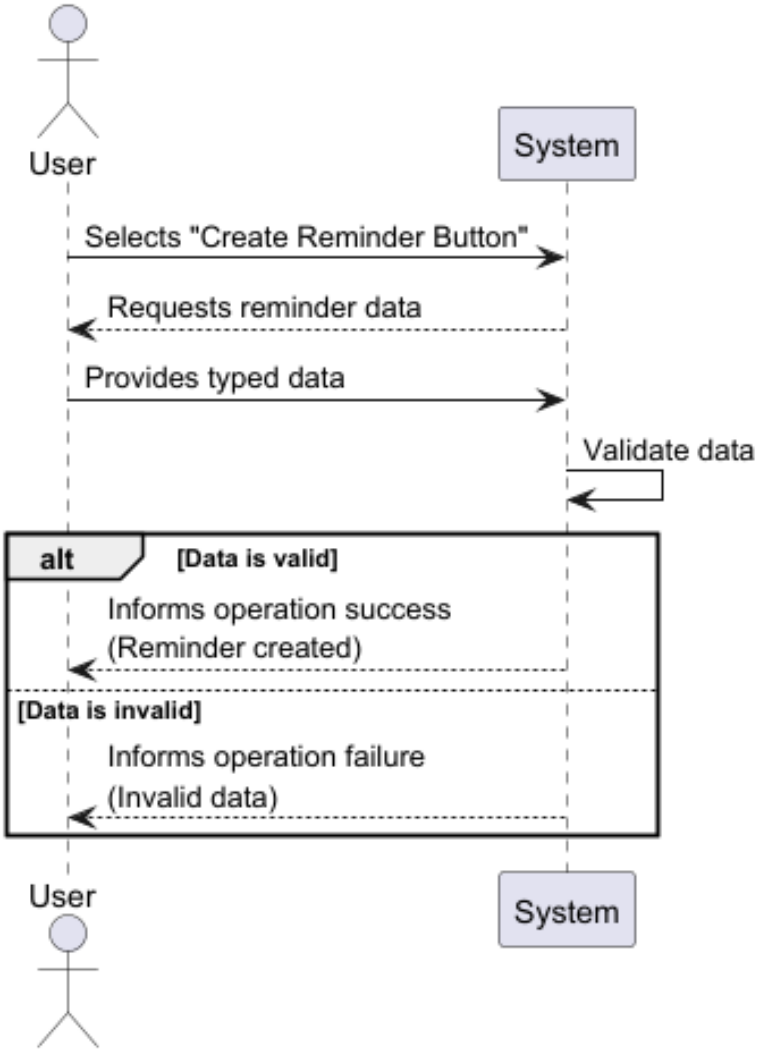

Fig. 10. System Sequence Diagram for UC12 – Create Reminder

Table 9. Use Case UC13 – Edit Reminder

Field	**Description**
Description	Updates an existing reminder (Update operation).
Pre-Conditions	The user is authenticated and the reminder exists.
Post-Conditions	The reminder is updated.
Basic Flow	1. The user selects a reminder; 2. The user edits its details; 3. The system validates the data; 4. The system updates the reminder via the backend API; 5. The system confirms success.
Alternative Flow	Invalid data: the update is rejected.

UC13 – Edit Reminder

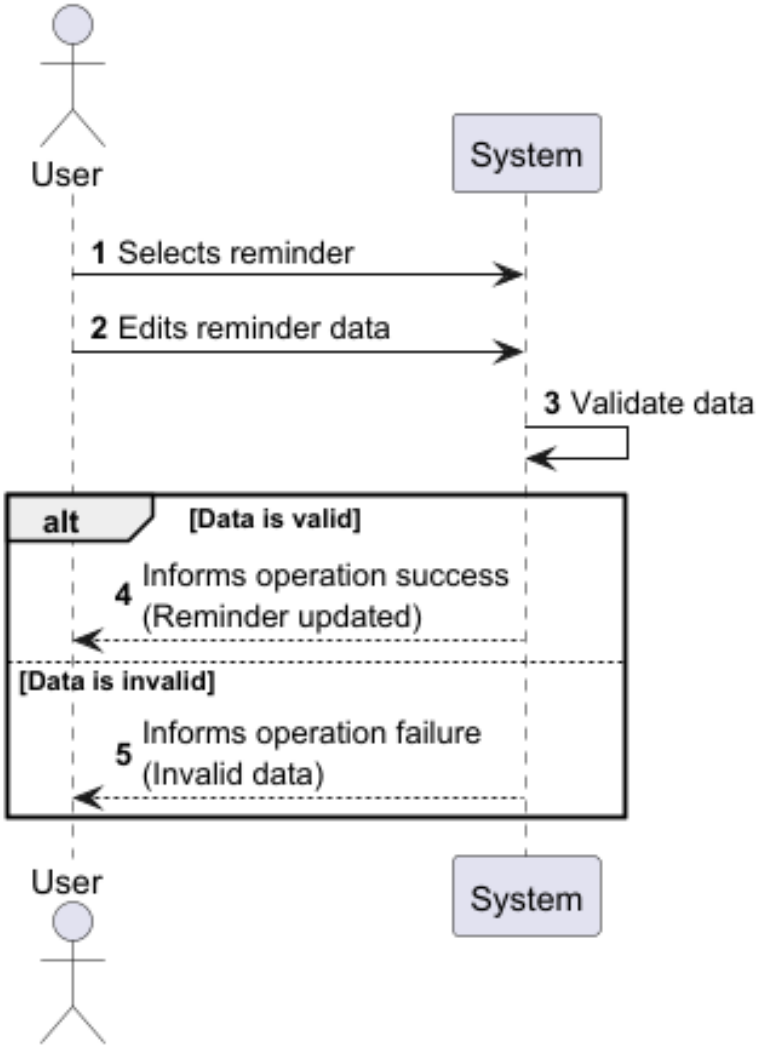

Fig. 11. System Sequence Diagram for UC13 – Edit Reminder

Table 10. Use Case UC14 – Delete Reminder

Field	**Description**
Description	Deletes a reminder from the system (Delete operation).
Pre-Conditions	The user is authenticated and the reminder exists.
Post-Conditions	The reminder is removed from the system.
Basic Flow	1. The user selects a reminder; 2. The user confirms the deletion; 3. The system deletes the reminder via the backend API; 4. The system updates the list of reminders.
Alternative Flow	Deletion cancelled: no changes are made.

UC14 – Delete Reminder

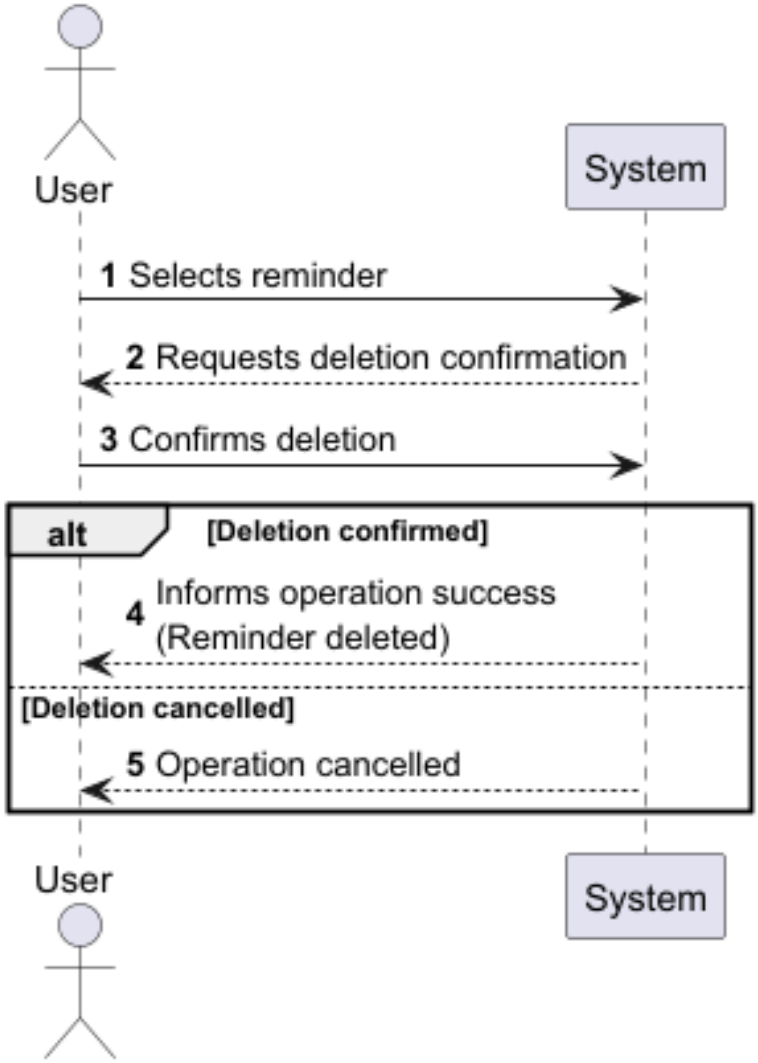

Fig. 12. System Sequence Diagram for UC14 – Delete Reminder

Table 11. Use Case UC15 – Mark Reminder as Completed

Field	Description
Description	Marks a reminder as completed (Update operation).
Pre-Conditions	The user is authenticated and the reminder exists.
Post-Conditions	The reminder status is updated to completed.
Basic Flow	1. The user selects a reminder; 2. The user marks it as completed; 3. The system updates the status via the backend API; 4. The system refreshes the user interface.
Alternative Flow	Update fails: the system displays an error message.

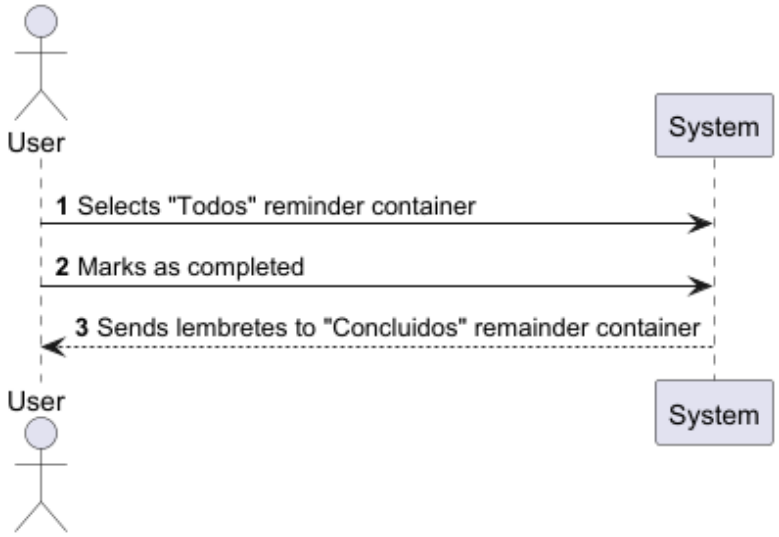

Fig. 13. System Sequence Diagram for UC15 – Mark Reminder as Completed

UC15 – Mark Reminder as Completed

Table 12. Use Case UC17 – View Completed Reminders

Field	Description
Description	Displays all completed reminders (Read operation).
Pre-Conditions	The user is authenticated.
Post-Conditions	Completed reminders are displayed.
Basic Flow	1. The user navigates to the completed reminders view; 2. The system retrieves completed reminders; 3. The system displays the results.
Alternative Flow	No completed reminders: the system displays an empty state.

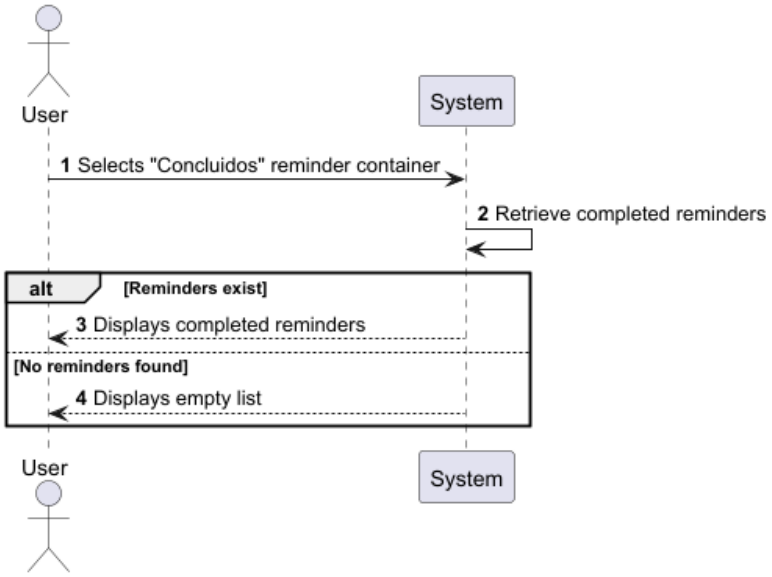

Fig. 14. System Sequence Diagram for UC17 – View Completed Reminders

UC17 – View Completed Reminders

It was decided not to document in detail the following Use Cases: UC4, UC6, UC9, UC11, and UC16. This decision was made because these Use Cases present interaction flows and functional behavior very similar to other Use Cases that were already documented, differing mainly in the entity being manipulated or in minor contextual aspects.

In order to avoid redundancy and keep the documentation clear and concise, representative Use Cases were selected and documented. These are considered sufficient to describe the system's overall behavior. The table below highlights the correspondence between the non-documented Use Cases and the similar Use Cases that were documented.

Table 13. Mapping between non-documented Use Cases and similar documented Use Cases

Non-documented Use Case	**Similar documented Use Case**
UC4 – Create Category	UC8 – Create List; UC12 – Create Reminder
UC6 – Delete Category	UC10 – Delete List; UC14 – Delete Reminder
UC9 – Edit List	UC5 – Edit Category; UC13 – Edit Reminder
UC11 – View List Details	UC17 – View Completed Reminders
UC16 – View Today's Reminders	UC17 – View Completed Reminders

3 Design

3.1 Physical Architecture

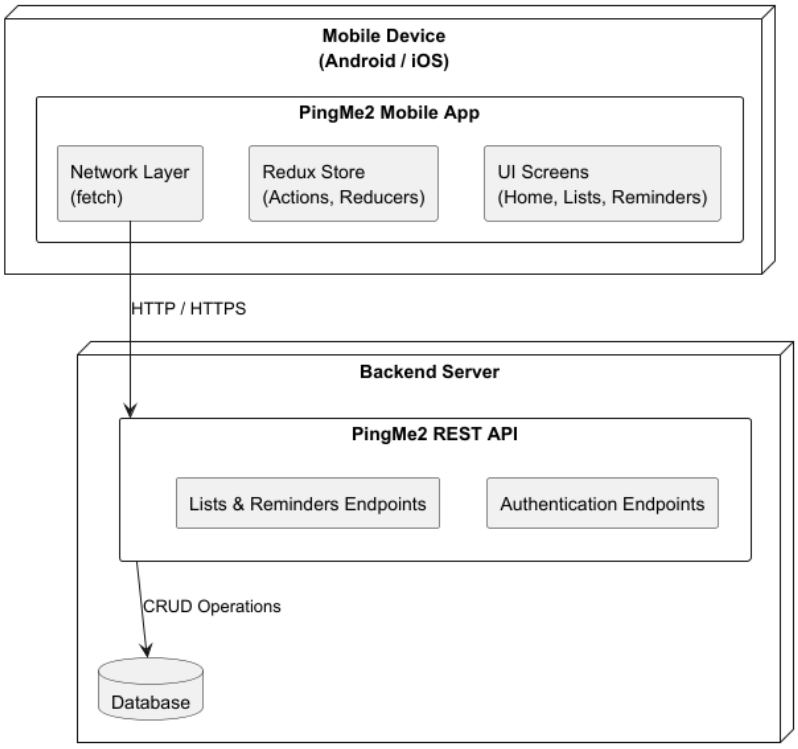

Fig. 15. Physical Architecture of the PingMe2 Application

3.2 Logical Architecture

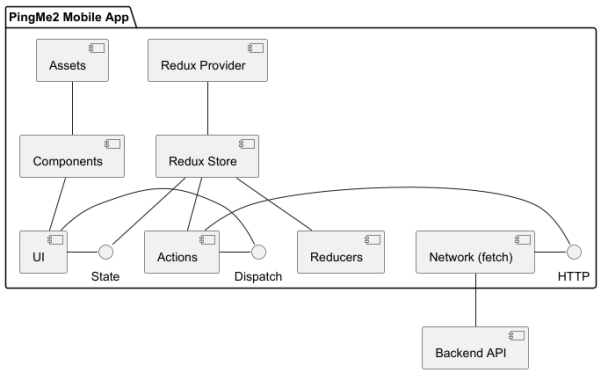

Fig. 16. Logical Architecture of the PingMe2 Application

3.3 Code Organization

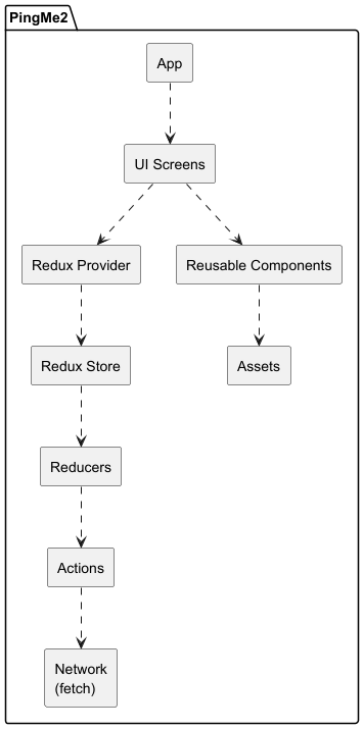

Fig. 17. Code Organization and Project Structure of PingMe2

3.4 Design Sequence Diagrams (SDs)

This section presents the Design Sequence Diagrams (SDs) for the main use cases of the PingMe2 application. These diagrams provide a more detailed view of the internal interactions between the system components, reflecting the design-level responsibilities and message exchanges.

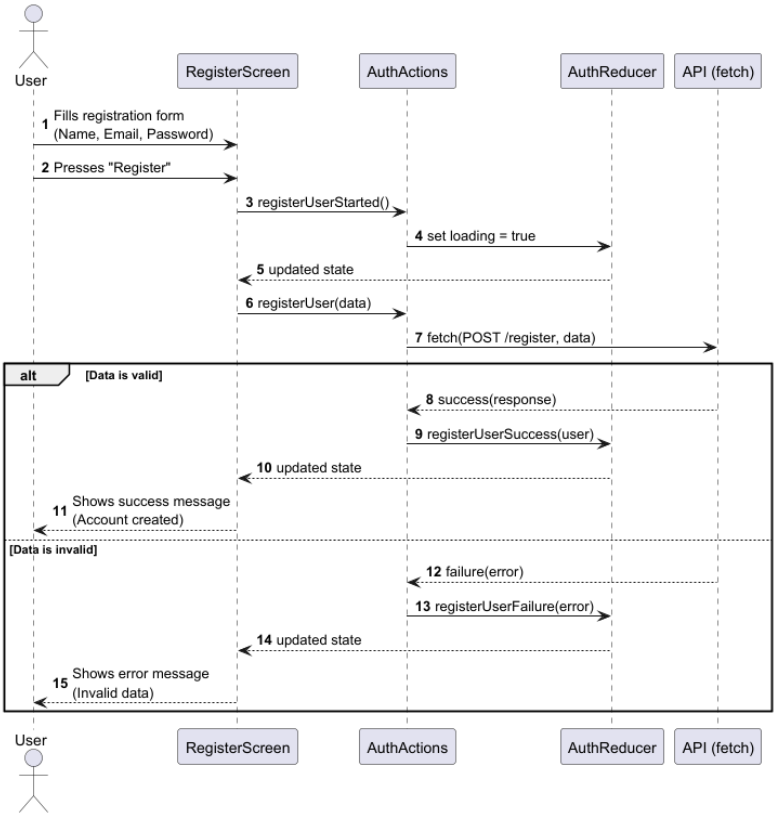

Fig. 18. Design Sequence Diagram for UC1 – Register User

UC1 – Register User

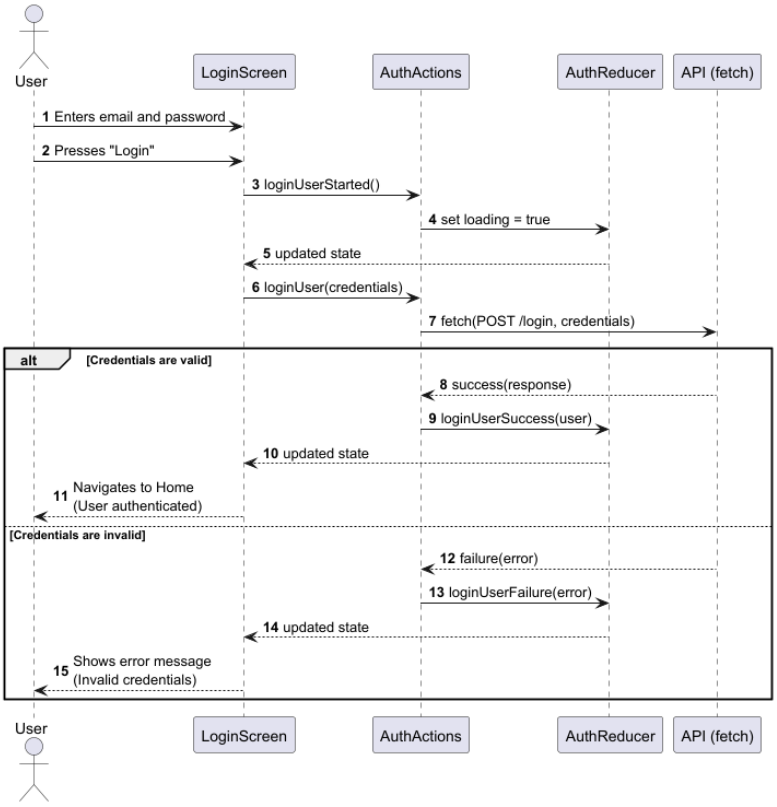

Fig. 19. Design Sequence Diagram for UC2 – Login User

UC2 – Login User

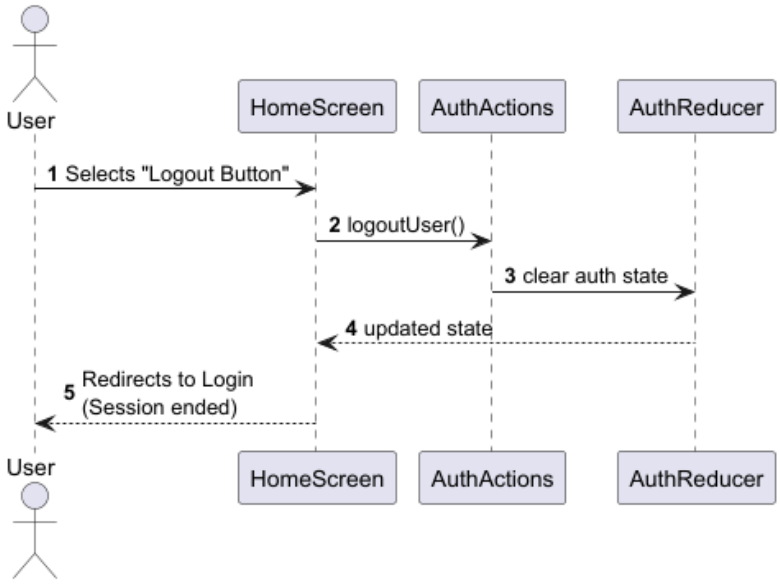

Fig. 20. Design Sequence Diagram for UC3 – Logout User

UC3 – Logout User

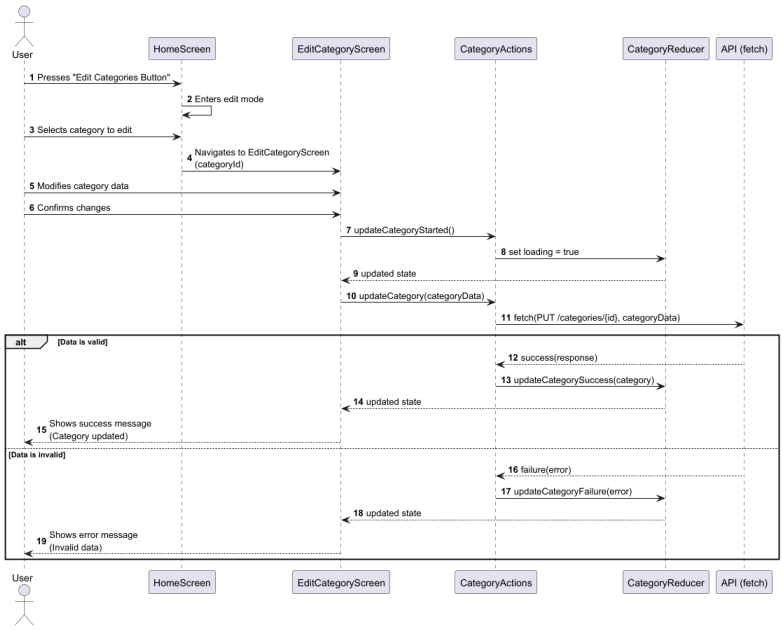

Fig. 21. Design Sequence Diagram for UC5 – Edit Category

UC5 – Edit Category

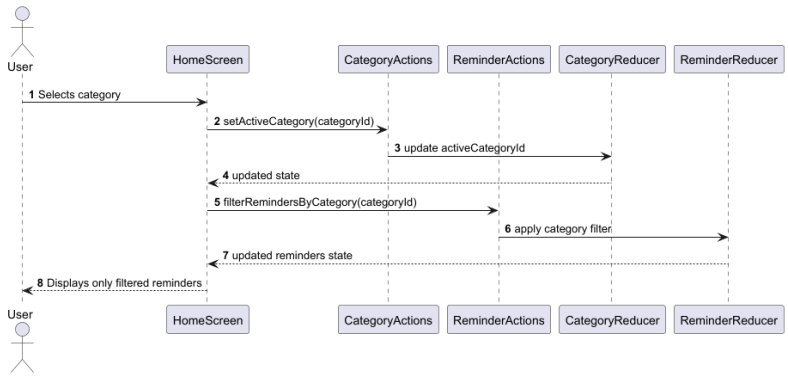

Fig. 22. Design Sequence Diagram for UC7 – Select Active Category

UC7 – Select Active Category

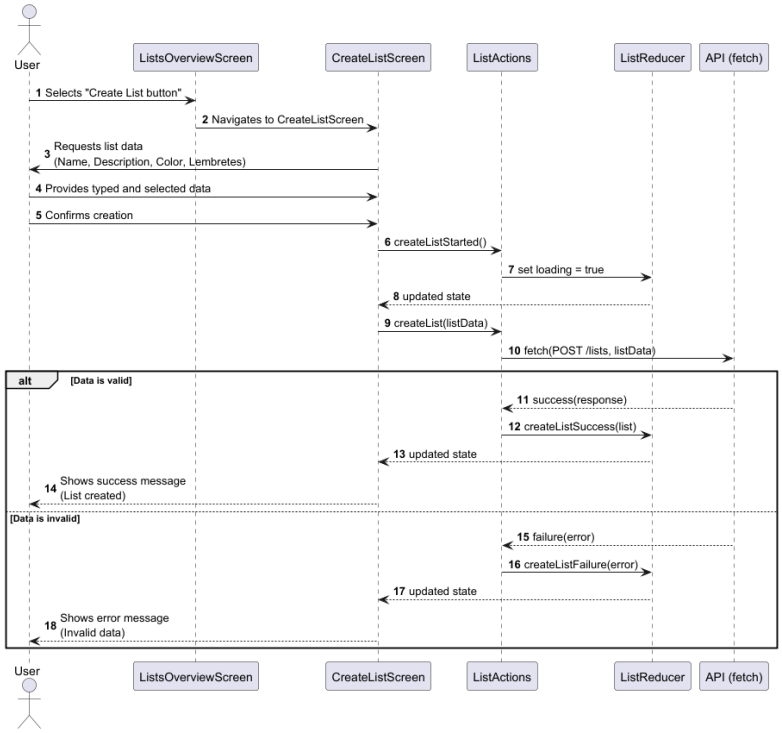

Fig. 23. Design Sequence Diagram for UC8 – Create List

UC8 – Create List

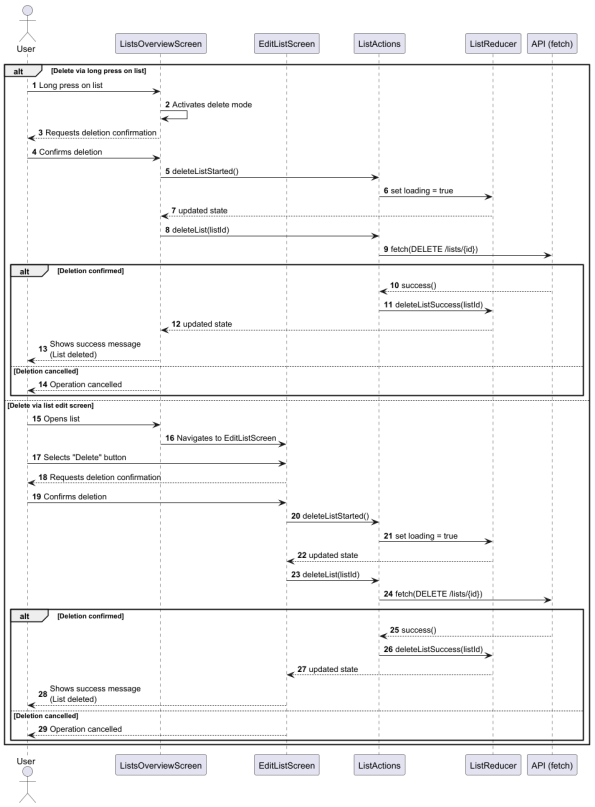

Fig. 24. Design Sequence Diagram for UC10 – Delete List

UC10 – Delete List

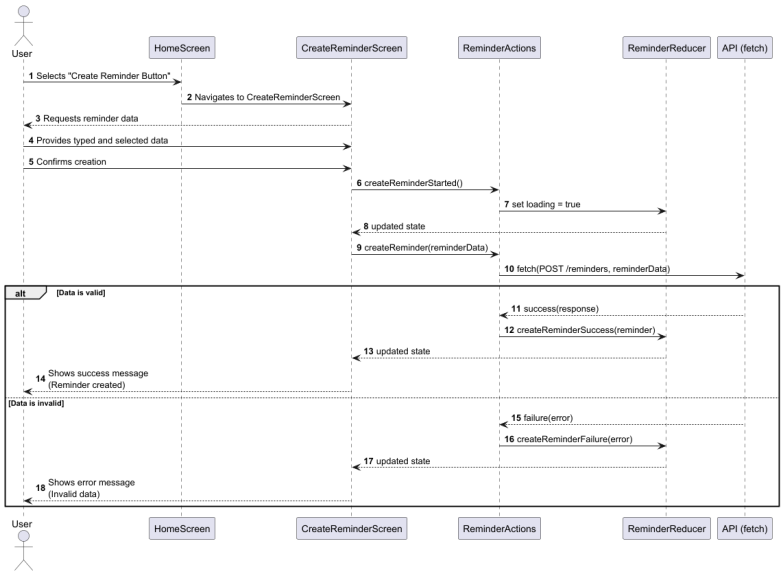

Fig. 25. Design Sequence Diagram for UC12 – Create Reminder

UC12 – Create Reminder

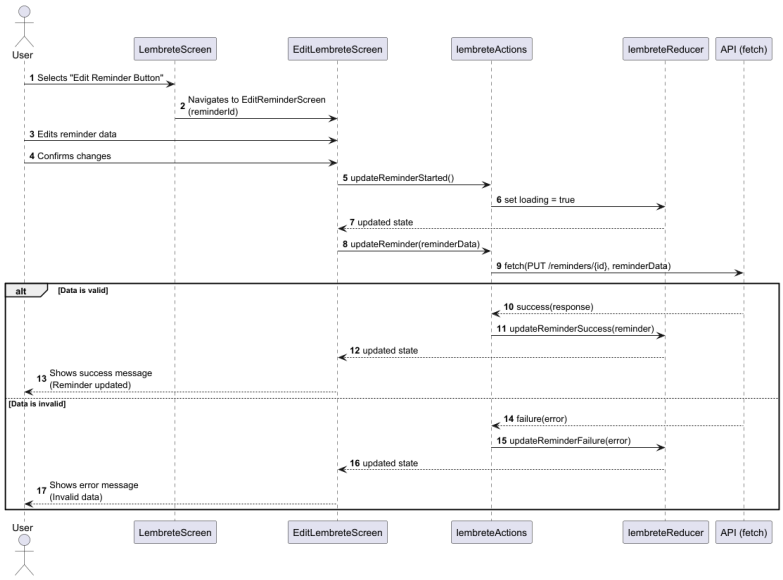

Fig. 26. Design Sequence Diagram for UC13 – Edit Reminder

UC13 – Edit Reminder

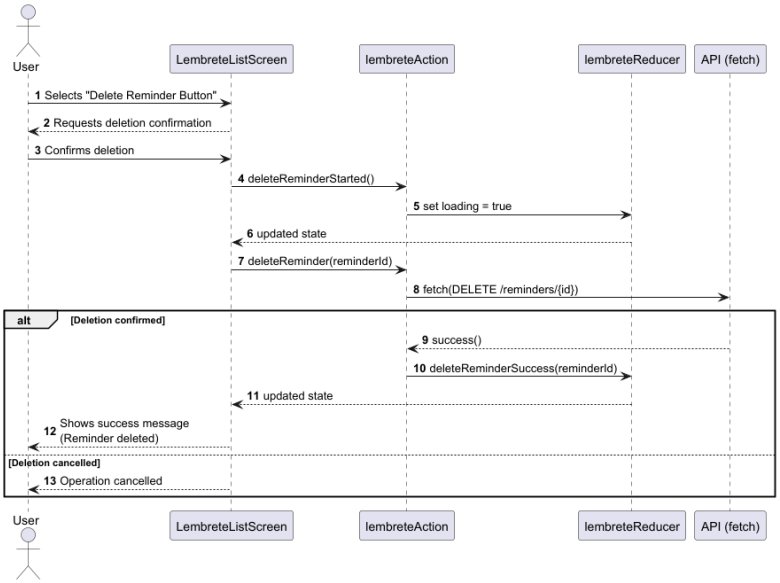

Fig. 27. Design Sequence Diagram for UC14 – Delete Reminder

UC14 – Delete Reminder

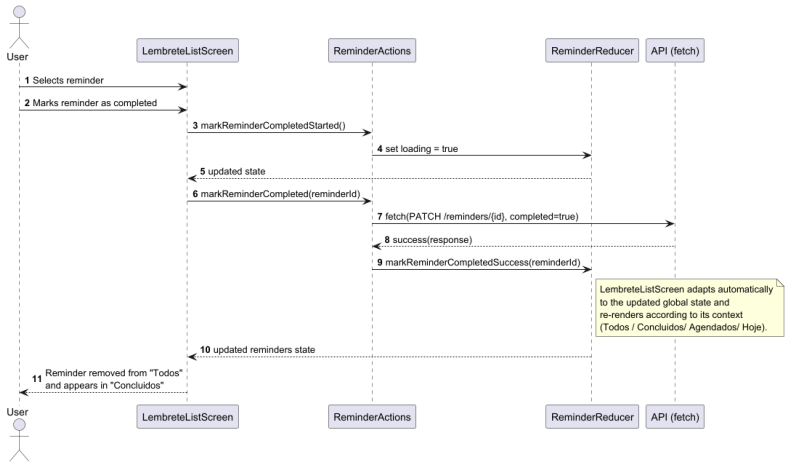

Fig. 28. Design Sequence Diagram for UC15 – Mark Reminder as Completed

UC15 – Mark Reminder as Completed

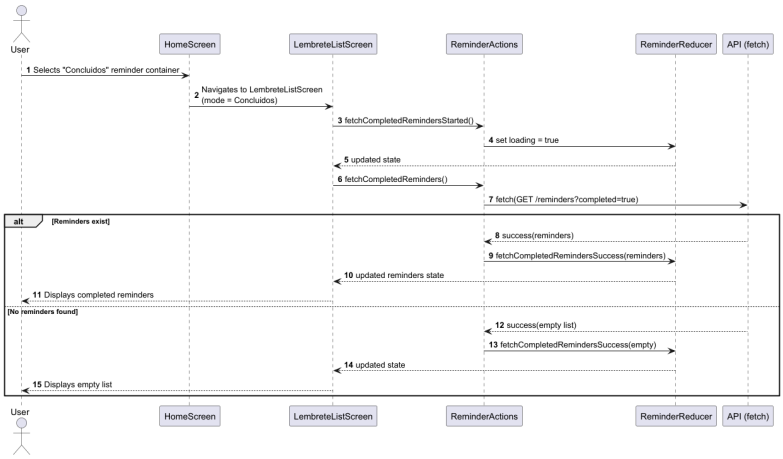

Fig. 29. Design Sequence Diagram for UC17 – View Completed Reminders

UC17 – View Completed Reminders

3.5 User Interface

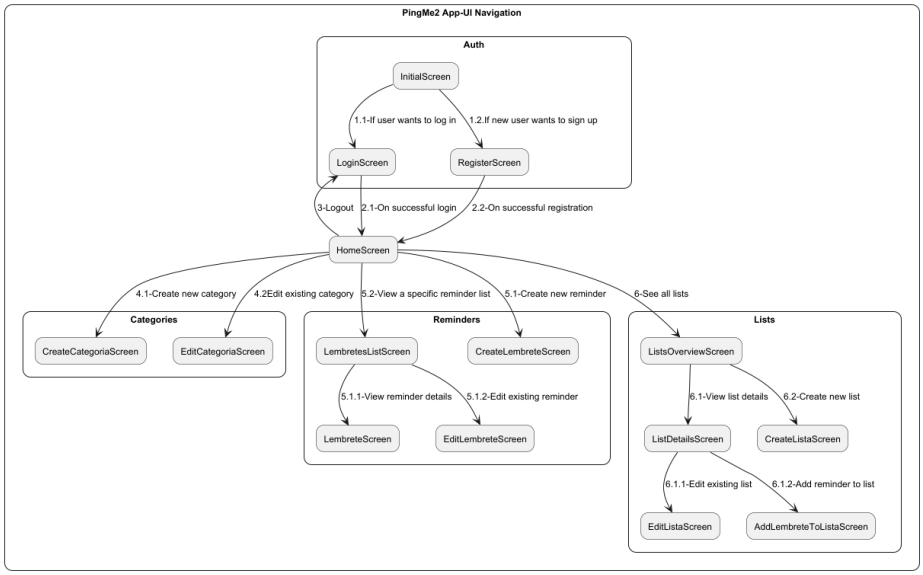

Fig. 30. User Interface Overview of the PingMe2 Mobile Application

4 Implementation

4.1 Introduction

This section presents representative excerpts of the application implementation, selected exclusively for documentation and technical explanation purposes. The presented excerpts correspond directly to the real implementation of the application and illustrate both core functionalities and secondary mechanisms that employ different and technically relevant interaction patterns.

4.2 Active Category Management

Listing 1.1. Redux Action Constant

```
export const SET_CATEGORIA_ATIVA = 'SET_CATEGORIA_ATIVA';
```

Listing 1.2. Persisting Active Category Selection

```
export const setCategoriaAtiva = (id: string) => async (
    dispatch: Dispatch) => {
  await AsyncStorage.setItem(STORAGE_KEY, id);
  dispatch({ type: SET_CATEGORIA_ATIVA, payload: id });
};
```

Listing 1.3. Loading Persisted Active Category

```
export const loadCategoriaAtiva = () => async (dispatch:
    Dispatch) => {
  const stored = await AsyncStorage.getItem(STORAGE_KEY);
  if (stored) {
    dispatch({ type: SET_CATEGORIA_ATIVA, payload: stored
        });
  }
};
```

Listing 1.4. Category Reducer Initial State

```
const initialState: CategoriaState = {
  items: [TODOS_CATEGORIA],
  categoriaAtivaId: ID_TODOS,
  loading: false,
  error: null,
};
```

Listing 1.5. Handling Active Category Update in Reducer

```
case SET_CATEGORIA_ATIVA:
  return {
    ...state,
    categoriaAtivaId: action.payload,
  };
```

Listing 1.6. Selecting Active Category from Global State

```
const categoriaAtivaId = useSelector(
  (state: RootState) => state.categorias.categoriaAtivaId
);
```

Listing 1.7. Filtering Reminders by Active Category

```
const lembretesFiltrados = useMemo(() => {
  if (!categoriaAtivaId || String(categoriaAtivaId) ===
      ID_TODOS) {
    return lembretes;
  }
  return lembretes.filter(
    (l: any) => String(l.categoria_id) === String(
        categoriaAtivaId)
  );
}, [lembretes, categoriaAtivaId]);
```

4.3 Data Creation and Persistence – Reminders

Listing 1.8. Reminder Creation Action Types

```
export const CREATE_LEMBRETE_REQUEST = '
    CREATE_LEMBRETE_REQUEST';
export const CREATE_LEMBRETE_SUCCESS = '
    CREATE_LEMBRETE_SUCCESS';
export const CREATE_LEMBRETE_FAILURE = '
    CREATE_LEMBRETE_FAILURE';
```

Listing 1.9. Asynchronous Reminder Creation Action

```
export const createLembrete = (data: any) => async (
    dispatch: Dispatch) => {
```

```
  dispatch({ type: CREATE_LEMBRETE_REQUEST });

  const response = await fetch(`${API_URL}/lembretes`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(data),
  });

  const lembrete = await response.json();
  dispatch({ type: CREATE_LEMBRETE_SUCCESS, payload:
      lembrete });
};
```

Listing 1.10. Reminder Reducer Initial State

```
const initialState: LembreteState = {
  items: [],
  loading: false,
  error: null,
};
```

Listing 1.11. Handling Reminder Creation in Reducer

```
case CREATE_LEMBRETE_REQUEST:
  return {
    ...state,
    loading: true,
  };

case CREATE_LEMBRETE_SUCCESS:
  return {
    ...state,
    items: [...state.items, action.payload],
    loading: false,
  };
```

Listing 1.12. Dispatching Reminder Creation from UI

```
dispatch(createLembrete({
  title,
  description,
  categoria_id,
  lista_id,
}));
```

4.4 Navigation Between Screens

Listing 1.13. Stack Navigator Configuration

```
const Stack = createNativeStackNavigator();

<Stack.Navigator initialRouteName="InitialScreen">
  <Stack.Screen name="InitialScreen" component={
      InitialScreen} />
  <Stack.Screen name="HomeScreen" component={HomeScreen}
      />
  <Stack.Screen name="CreateLembreteScreen" component={
      CreateLembreteScreen} />
</Stack.Navigator>
```

Listing 1.14. Navigation Trigger from UI Interaction

```
const navigation = useNavigation();

const handleCreateReminder = () => {
  navigation.navigate('CreateLembreteScreen');
};
```

Listing 1.15. Navigation Button Component

```
<Button
  title="Create Reminder"
  onPress={handleCreateReminder}
/>
```

Listing 1.16. Returning to Previous Screen After Success

```
const handleSuccess = () => {
  navigation.goBack();
};
```

4.5 Long Press Interaction on Lists

Listing 1.17. UI State for Delete Mode Control

```
const [deleteMode, setDeleteMode] = useState(false);
```

```
const [selectedListId, setSelectedListId] = useState<
    string | null>(null);
```

Listing 1.18. Short Press vs Long Press Gesture Handling

```
<TouchableOpacity
  onPress={() => {
    if (deleteMode) {
      setSelectedListId(lista.id);
    } else {
      navigation.navigate('ListDetailsScreen', { listId:
          lista.id });
    }
  }}
  onLongPress={() => {
    setDeleteMode(true);
    setSelectedListId(lista.id);
  }}
>
  <Text>{lista.nome}</Text>
</TouchableOpacity>
```

Listing 1.19. Deletion Confirmation and State Reset

```
const handleConfirmDelete = () => {
  if (selectedListId) {
    dispatch(deleteLista(selectedListId));
    setDeleteMode(false);
    setSelectedListId(null);
  }
};
```

Listing 1.20. Deletion Confirmation Modal

```
<ConfirmationModal
  visible={deleteMode}
  onConfirm={handleConfirmDelete}
  onCancel={() => setDeleteMode(false)}
/>
```

5 Tests

6 Conclusion

The objectives defined at the beginning of the PingMe2 project were successfully achieved. The developed application implements all the planned functionalities, providing users with an effective solution for managing reminders, categories, and lists through an intuitive and structured mobile interface.

Throughout the development process, no major technical obstacles were encountered that compromised the implementation. This allowed the project to progress in a clear and well-organized manner. The use of React Native contributed to a modular and maintainable codebase, while Redux ensured consistent and centralized state management across the application.

Furthermore, the interaction with a RESTful API enabled the practical application of a complete set of CRUD operations, extending beyond simple data retrieval to include the creation, update, and deletion of resources. This reinforced the importance of a clear separation between frontend and backend responsibilities within a client–server architecture.

Overall, the PingMe2 project provided valuable practical experience in modern mobile application development, consolidating both theoretical concepts and technical skills acquired throughout the course.