1. Project Overview

Loop Lab is a Flutter-based mobile application designed to facilitate educational content management, event handling, and user profile management. It targets students, teachers, and administrators, enabling them to interact with courses, announcements, and personal account settings.

The app supports dark and light themes, offers role-based access (admin, teacher, student), and uses Firebase Firestore for backend data storage and retrieval.

---

2. Technologies Used

- **Flutter**: Cross-platform UI toolkit for building natively compiled apps.
- **Dart**: Programming language for Flutter development.
- **Firebase Firestore**: NoSQL cloud database for storing user data, events, announcements.
- **Adaptive Theme**: Flutter package used for dynamic light/dark mode theming.
- **Google Firebase Authentication** (presumed for user management).
- **Android & iOS** native platform support.
- **Kotlin / Gradle**: For Android build system configuration.

---

3. Project Features

- **User Roles**: Admins, teachers, and students with role-specific UI and permissions.
- **Profile Management**: Users can view and edit their profile including name, email, phone, and role.
- **Announcements**: Display announcements with "new" badges, and allow admins to add announcements.
- **Events Management**: Display events and allow admins to add events with details like title, date, location, etc.
- **Dark/Light Theme**: Seamless switching between dark and light UI modes.
- **Firestore Integration**: CRUD operations for user data, announcements, and events.
- **Responsive UI**: Scrollable lists, bottom sheets, floating action buttons with proper theming.

---

4. Architecture and Design

The app follows a **component-based** architecture using Flutter's widget system. State management is handled via StatefulWidgets and `setState` calls for simplicity, with adaptive theme support for UI consistency.

Key screens and features are encapsulated into separate widgets and screens, e.g.:

- `AnnouncementsScreen` for listing and adding announcements.
- `EditProfileScreen` for user profile editing.
- `ProfileScreen` as the main user account hub.
- Bottom sheets for data entry forms.

---

5. Key Components and Screens

Profile Screen

- Displays user avatar and profile options.
- Allows navigation to editing profile, notification settings, privacy, and help screens.
- Provides theme toggle and logout button.
- Fetches user data from Firestore and passes user ID to Edit Profile screen.

Edit Profile Screen

- Editable fields: name, email, phone, role (dropdown).
- User data pre-filled from Firestore.
- Saves updates back to Firestore for persistence.
- The role dropdown ensures single unique selected value.

Announcements Screen

- Lists all announcements fetched from Firestore.
- Highlights new announcements.
- Admin users can add announcements via a floating action button and bottom sheet form.
- Bottom sheet allows input for title, date, content, and “Is New” toggle.

Events (Planned / Implemented)

- Similar to announcements: list and add events with details including images and dates.
- Data stored and fetched from Firestore.

---

6. Firebase Integration

- **User Data**: Stored in `users` collection.
- **Announcements**: Stored in `announcements` collection with fields: title, date, content, isNew.
- **Events**: Stored in `events` collection with fields: title, date, time, location, description, imageUrl, isUpcoming.
- **Firestore Queries**: Used to fetch, add, and update documents with proper error handling.

- Firestore plugin requires minimum Android SDK version 23.

---

7. UI/UX Design Approach

- Clean, modern interface with consistent color themes adapting to light and dark modes.
- Use of Material Design widgets: AppBar, ListView, FloatingActionButton, BottomSheet.
- User input forms with proper validation hints.
- Responsive scrolling lists with padding and spacing for readability.
- Use of badges (e.g., "NEW" tag) to highlight recent announcements.

---

8. State Management and Theming

- Uses Flutter's `StatefulWidget` and `setState` for local UI state changes.
- Theme switching handled via the `adaptive_theme` package allowing dark/light mode toggle.
- Consistent use of custom color palettes (`AppColors`) depending on theme mode.
- Data fetching triggers UI rebuild via `setState`.

---

9. Error Handling and Data Validation

- Firestore data fetching wrapped in try/catch blocks.
- Snackbar notifications show errors or success messages.
- Form inputs trim whitespace and ensure mandatory fields before submission.
- Role dropdown prevents duplicate or invalid selections.

---

10. Known Issues and Limitations

- Minimum Android SDK 23 requirement limits older device compatibility.
- Current state management is basic; may need advanced solutions (e.g., Provider, Riverpod) for larger scale.
- Image uploading and selection are placeholders and require full implementation.
- No offline support or caching is implemented.
- Limited user authentication and security checks (assumed to be implemented separately).

---

11. Future Enhancements

- Add full event management including editing and deleting.
- Integrate Firebase Authentication for real user login.
- Add profile picture upload and storage.
- Implement notification push services.
- Improve state management for scalability.
- Add analytics and user activity tracking.
- Offline data sync and caching.