FINAL PROJECT REPORT

SEMESTER 1, ACADEMIC YEAR: 2025-2026

CT312H: MOBILE PROGRAMMING

- **Project/Application name:** Anonymous Social Media App
- **GitHub link:** https://github.com/25-26Sem1-Courses/ct312hm01-project-Letrongtri
- **Student ID 1: B2206019**
- **Student Name 1: Lê Trọng Trí**
- **Student ID 2: B2205973**
- **Student Name 2: Huỳnh Tiểu Băng**
- **Class/Group Number: CT312HM01**

I. Introduction

- Project/application description: This is a cross-platform social media application, focusing on anonymity and free speech. The project aims to create a safe space where users can share their thoughts and discuss various topics without revealing their real identity. Users can create posts and interact with the community through features such as liking, commenting, and reporting inappropriate content.

- A task assignment sheet for each member working in groups.

STT	Task	Assign
1	Implement auth features	Lê Trọng Trí
2	Implement feed screen	Lê Trọng Trí
3	Implement search feature	Lê Trọng Trí
4	Implement create, edit, and delete post	Huỳnh Tiểu Băng
5	Implement detail post screen, comment features	Lê Trọng Trí
6	Implement notification features	Lê Trọng Trí
7	Implement profile screen	Huỳnh Tiểu Băng
8	Implement repost feature	Huỳnh Tiểu Băng
9	Implement responsive features	Huỳnh Tiểu Băng

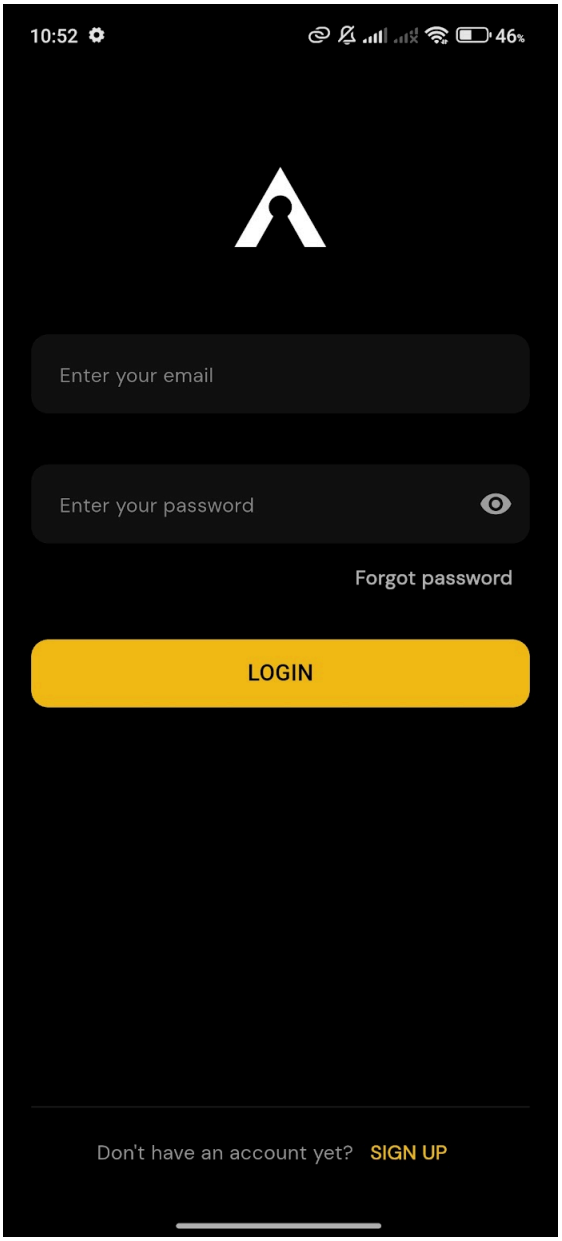
10	Deploy app	Huỳnh Tiểu Băng

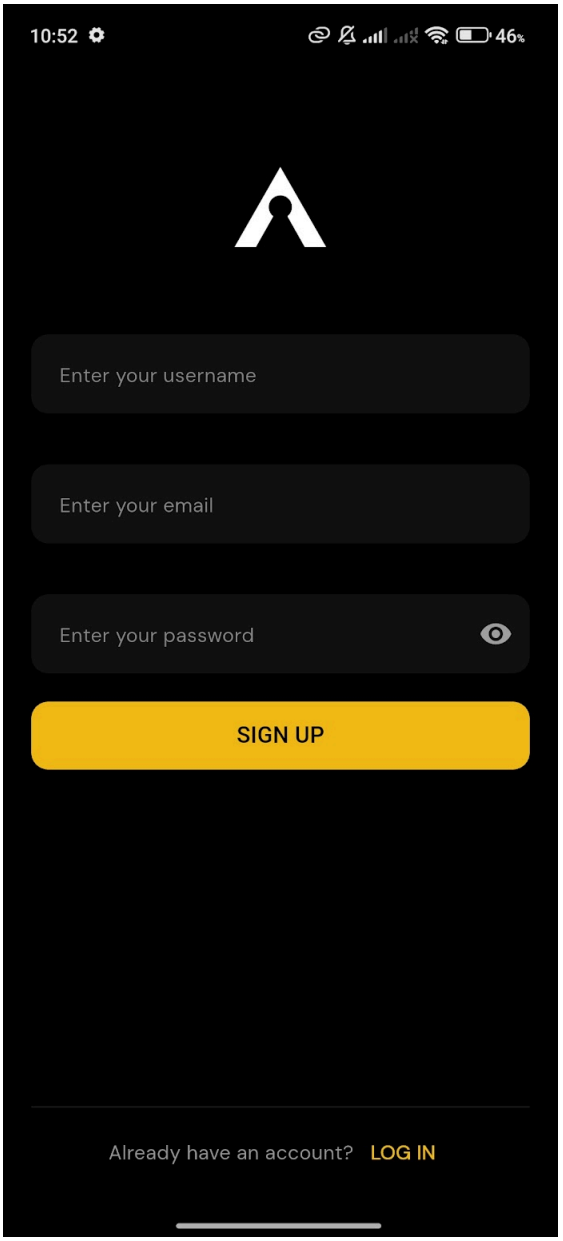
II. Details of implemented features

1. Authentication User:

- **Description:** This feature handles the user authentication process, which includes sign up, sign in.

- **Screenshots:**





- Implementation details:
    - **Standard Widgets:** Scaffold , SafeArea , Padding , Column , Row , SizedBox , Center , Divider , Form , TextFormField , Text , ElevatedButton, TextButton , IconButton , Icon , CircularProgressIndicator , AlertDialog , Image (Image.asset).
    - **Special Widgets:**
        - ValueListenableBuilder: Used to listen to the _isSubmitting state (a ValueNotifier) to toggle the loading indicator. This is a specific performance optimization widget to rebuild only the button part of the UI, which was not introduced in the labs (the labs focused on setState, FutureBuilder, or Consumer).

    - Spacer: Used in the layout to automatically occupy the available space between the form and the bottom "Switch Auth Mode" link, ensuring the layout adjusts dynamically to the screen size.
    - OutlineInputBorder: While part of InputDecoration, this specific class was used to customize the border radius and color of the text fields to match the dark theme design, which differs from the standard Material Design styling used in the labs.
- Does the feature use any libraries or plugins? If so, state them and state the role of those libraries/plugins.
    - pocketbase: This is the core library used to communicate with the backend. It handles the actual API requests for creating users (signup), authenticating users (login), and managing the authentication store (saving/clearing tokens).
    - provider: Used for state management. It injects the AuthManager into the widget tree, allowing the UI (AuthScreen) to access authentication methods and listen for state changes (e.g., loading status, successful login) to update the interface.
    - shared_preferences: Used within the pocketbase_client.dart file to persist the user's authentication data locally. This enables the "Auto Login" feature, allowing the app to remember the user session even after the app is closed and reopened.
    - go_router: Manages the application's navigation and routing. It handles the critical logic of redirecting unauthenticated users to the Login/Signup page and redirecting authenticated users to the Home Feed.
    - flutter_dotenv: Used to load configuration variables (specifically the POCKETBASE_URL) from the .env file, ensuring the authentication service connects to the correct server environment.
    - http: Used within the AuthService to handle specific network-related exceptions (specifically ClientException) that might occur during the login or signup process.
    - google_fonts: Used to load and apply the custom fonts ("Outfit" and "DM Sans") to the text elements within the Login and Signup forms.
    - device_preview: Used in main.dart to wrap the application, allowing the developer to preview the Authentication UI on various device screen sizes and resolutions.

- Does this feature use a shared state management solution? If so, state briefly how your solution works and describes the code architecture.

    This feature implements a shared state management solution using the Provider package, structured around the MVVM (Model-View-ViewModel) architecture. In this architecture, the AuthManager class acts as the ViewModel. It extends ChangeNotifier to manage the global authentication state, such as the current user object and loading status. This manager is initialized and provided to the entire widget tree via the MultiProvider in main.dart. The workflow operates as follows: The AuthScreen (View) interacts with the AuthManager to trigger actions like login or signup. The AuthManager then delegates the actual data processing and API calls to the AuthService (Service). Once the service returns a result, the AuthManager updates its internal state and calls notifyListeners(). This signal notifies the GoRouter (which listens to AuthManager) to automatically redirect the user to the home screen upon success, or updates the UI to show errors or loading indicators

- Does this feature read or store any data? Locally or remotely? If so, state the data table structure. If you are using a REST API, briefly describe that API (how to call, input, output).
    - Data Storage Locations:
        - Remote Storage: User data is stored in a self-hosted PocketBase backend database.

- Local Storage: Authentication session data (token and user model) is stored locally on the device using SharedPreferences under the key 'pb_auth'.

- Data Table Structure (Remote):

  The data is stored in the users collection within PocketBase. Based on the User model and the signup function, the table structure includes the following fields:
  - id (Text): Unique record identifier (system generated).
  - username (Text): The user's display name.
  - email (Email): The user's email address.
  - password (Password): The user's password (hashed and handled securely by PocketBase).
  - avatar (File): Optional profile picture (referenced in the User model).
  - created (DateTime): Timestamp of account creation.
  - updated (DateTime): Timestamp of the last update.

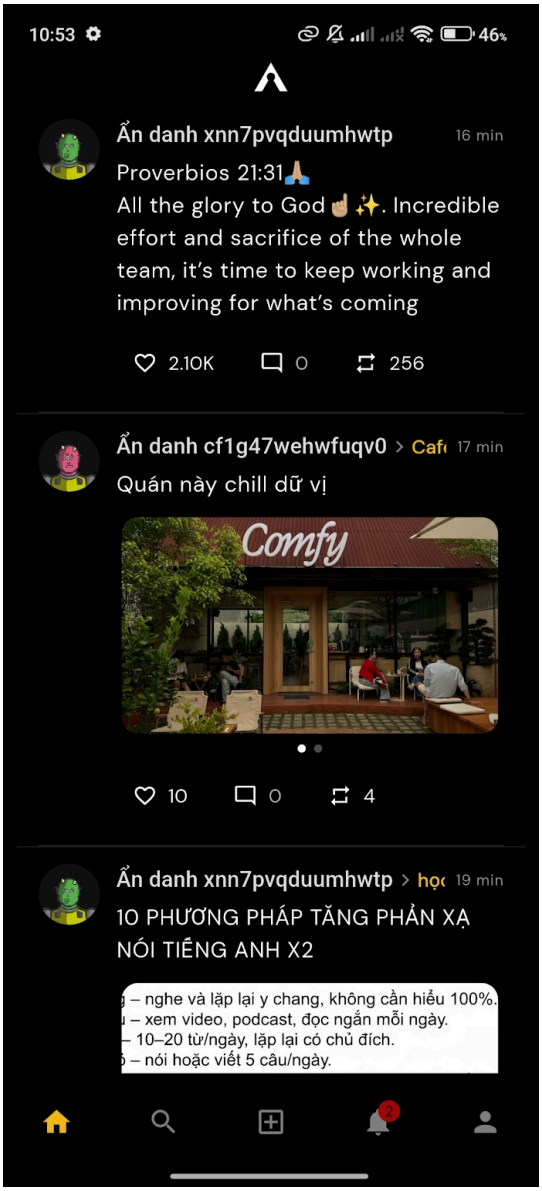
- REST API Description:

  The application uses the PocketBase SDK to interact with the underlying REST API.
  - Sign Up (Create User):
    - How to call: Invoked via pb.collection('users').create(...) in AuthService.
    - Input: A JSON body containing email, username, password, and passwordConfirm.
    - Output: Returns a RecordModel containing the newly created user's data (excluding the password).
  - Log In (Authenticate)
    - How to call: Invoked via pb.collection('users').authWithPassword(...) in AuthService.
    - Input: The user's email and password.
    - Output: Returns an AuthRecord object containing the valid JWT token and the user record data. The SDK automatically caches this token in the AsyncAuthStore for subsequent requests

2. News Feed:

- Description: The “News Feed” feature displays a dynamic list of posts shared by users. It allows users to browse recent posts, view details such as author, category, and engagement metrics (likes, comments, shares), and interact directly within the feed

- Screenshots:



Implementation details:

- **List of widgets used**: Scaffold, AppBar, SafeArea, Padding, ListView.separated, Center, CircularProgressIndicator, Text, Image.asset, RefreshIndicator, InkWell, Column, Row, SizedBox, Expanded, AspectRatio, PageView.builder, ClipRRect, Stack, Positioned, Icon.
- **Special widgets:**
    - SinglePostItem: A custom widget used to display individual posts, including user info, content, images, and interactive buttons.
    - RefreshIndicator: Used to implement the "pull-to-refresh" functionality to reload the feed.
    - PageView.builder: Utilized within the post item to create a scrollable slider for posts with multiple images.

+ Does the feature use any libraries or plugins? If so, state them and state the role of those libraries/plugins.

- provider: It allows the Feed Screen to listen for updates from the PostsManager ViewModel. Whenever the post data changes (e.g., when new posts are fetched or updated), the UI automatically rebuilds to reflect the latest state. This ensures efficient data flow between the UI and business logic, avoiding manual setState() calls.
- pocketbase: The SDK used to connect to the backend to fetch the list of posts and handle pagination.
- go_router: Manages navigation, allowing users to tap on a post to navigate to the detailed view.
- flutter_dotenv: Used to retrieve the environment configuration (like the base URL) to construct image URLs dynamically.
- photo_view: Used in the FullImageViewer widget. This allows users to zoom, pan, and view post images in a full-screen gallery when interacting with the feed.
- flutter_screenutil: Used in main.dart to initialize screen adaptation. It ensures the News Feed layout (padding, font sizes, avatar sizes) remains consistent across different device screen sizes.
- device_preview: Wraps the entire application to allow previewing the News Feed layout on multiple virtual devices and resolutions during development.
- flutter_timezone & timezone: Used in main.dart to configure the local timezone. This ensures that the "time ago" display (e.g., "5 min ago") in the feed is accurate relative to the user's current location.

+ Does this feature use a shared state management solution? If so, state briefly how your solution works and describe the code architecture.

- This feature uses the Provider package for shared state management following the MVVM (Model-View-ViewModel) architecture:
    - Model: The Post class defines the data structure.
    - ViewModel: The PostsManager acts as a ChangeNotifier that stores post data (_posts) and notifies the UI when changes occur (e.g., after fetching data).
    - View: The FeedScreen listens to these updates using context.watch<PostsManager>() and rebuilds automatically when the data updates.

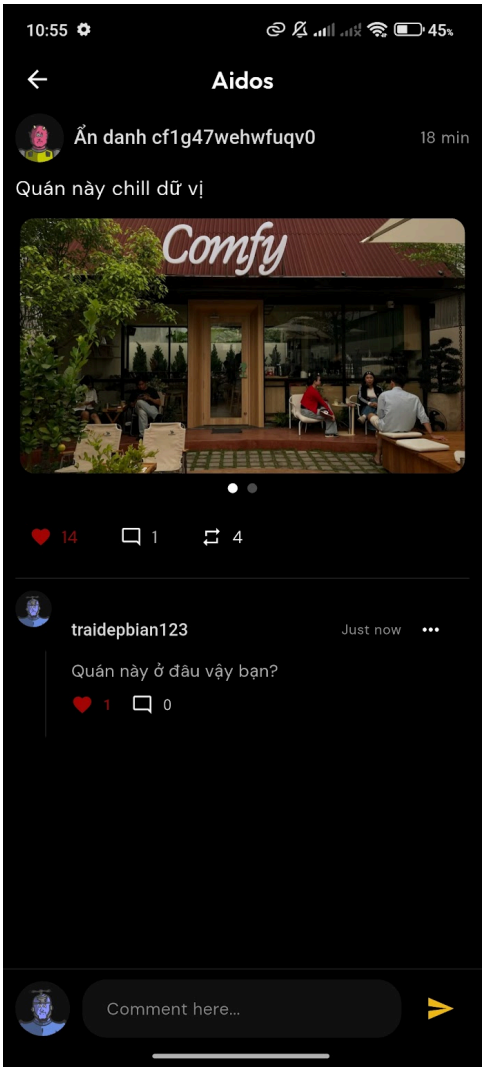
+ Does this feature read or store any data? Locally or remotely? If so, state the data table structure. If you are using a REST API, briefly describe that API (how to call, input, output).

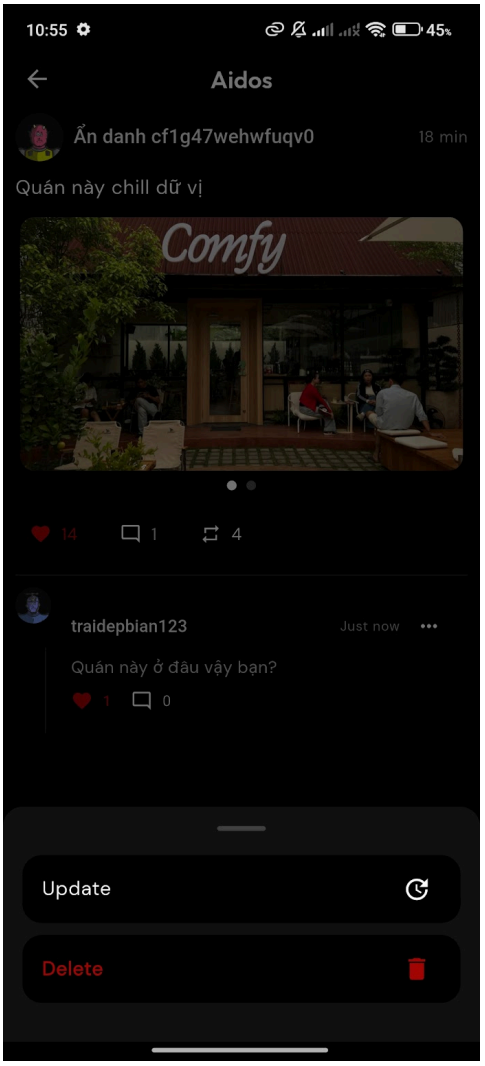
- Data Storage: The feature reads data remotely from the PocketBase backend.
- Data Structure: Data is stored in the posts collection with fields including: id, userId (relation), content (text), topicId (relation), images (file list), and counters for likes, comments, and reposts.
- REST API:
    - How to call: The app calls pb.collection('posts').getList(...) via the PostService.
    - Input: The API accepts parameters for pagination (page, perPage), sorting (sort: '-created'), and expansion of relations (expand: 'userId,topicId').
    - Output: It returns a ResultList containing post records, which are mapped into a list of Post objects.

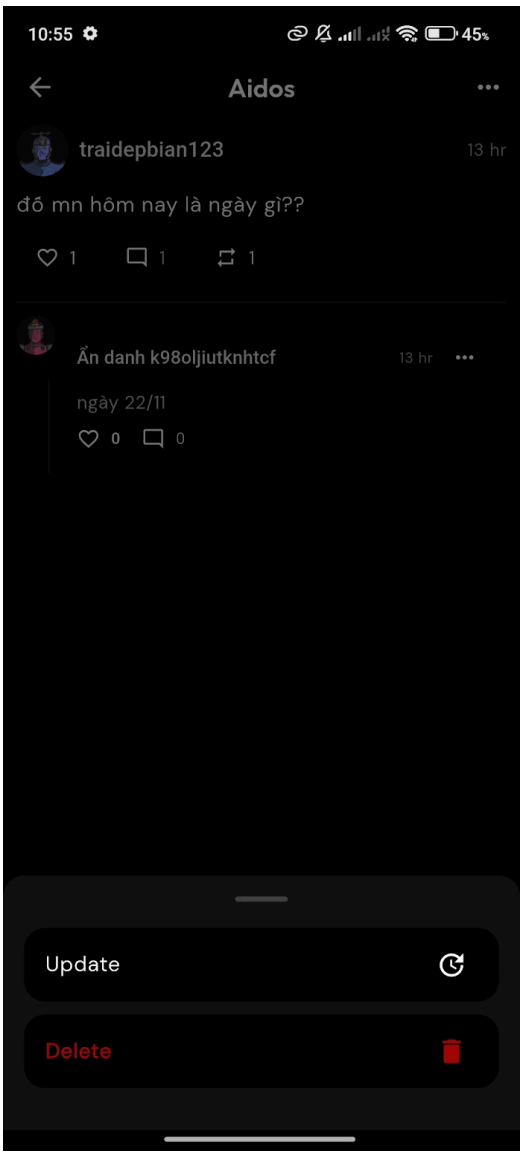
3. Post Detail:

- Description: It is used to display the full content of a selected post, including all comments and replies.Users can view detailed discussions, reply to other users, like comments, and add new comments directly from this page.

- Screenshots:


10:55
45%
Aidos
Ẩn danh cf1g47wehwfuqv0
18 min
Quán này chill dữ vị
Comfy
14
1
4
traidepbian123
Just now
Quán này ở đâu vậy bạn?
1
0
Comment here...


10:55
45%
Aidos
Ẩn danh cf1g47wehwfuqv0
18 min
Quán này chill dữ vị
Comfy
14
1
4
traidepbian123
Just now
Quán này ở đâu vậy bạn?
1
0
Update
Delete



- Implementation details: students should answer the following questions:

+ List the widgets used for this feature/page. Is there any special widget (not introduced in the lesson) used? If so, state them.

- **List of widgets used:** Scaffold, AppBar, SafeArea, Column, Row, Expanded, Padding, Container, SizedBox, Text, Icon, IconButton, Divider, RefreshIndicator, ListView, SingleChildScrollView, TextField, Stack, Positioned, ClipRRect, GestureDetector, Image.network, CircularProgressIndicator, TextButton.
- **Special widgets:**
  - ChangeNotifierProxyProvider: Used to initialize CommentManager which depends on AuthManager to handle user-specific logic like highlighting own comments.
  - AnimatedPadding: Used in DetailPostScreen to smoothly push the AddComment input bar up when the keyboard appears (MediaQuery.of(context).viewInsets.bottom).

    - PageView.builder: Implemented in DetailPostContent to create a swipeable gallery for posts containing multiple images.
    - CommentTreeWidget (External): Used in CommentList to render nested comments (parent-child relationship) visually with connecting lines.
    - PhotoViewGallery (External): Used within FullImageViewer (navigated from this screen) to allow zooming and panning of images.
    - Avatar (Custom): A custom reusable widget to display user profile pictures or generated avatars.

+ Does the feature use any libraries or plugins? If so, state them and state the role of those libraries/plugins.

- comment_tree: Specifically used to display the hierarchical structure of discussions (Root comments vs. Replies).
- photo_view: Provides the zoomable image viewer functionality when a user taps on an image in the post details.
- provider: Connects the DetailPostScreen, CommentList, and AddComment widgets to CommentManager and PostsManager. It handles state updates for fetching comments, liking, and replying without rebuilding the entire screen.
- pocketbase: The backend SDK used to fetch the specific post details, load comments, and handle actions like "Add Comment" or "Like".
- go_router: Manages navigation parameters (passing the postId to the screen) and handles routing logic.
- flutter_dotenv: Retrieves the base URL configuration to correctly constructing image links for the post and avatar.

+ Does this feature use a shared state management solution? If so, state briefly how your solution works and describe the code architecture.

- It uses the Provider package implementing the MVVM pattern.
    - Proxy Provider: The CommentManager is created using ChangeNotifierProxyProvider because it needs access to the AuthManager (to know who is the current user).
    - Optimistic UI: In CommentManager, methods like addComment or onLikeCommentPressed update the local list _comments immediately and call notifyListeners() to update the UI instantly. Then, it calls the API in the background. If the API fails, it rolls back the changes.
    - Lazy Loading: The CommentList widget fetches root comments first. Child replies are only fetched when the user explicitly interacts with a comment thread.

+ Does this feature read or store any data? Locally or remotely? If so, state the data table structure. If you are using a REST API, briefly describe that API (how to call, input, output).

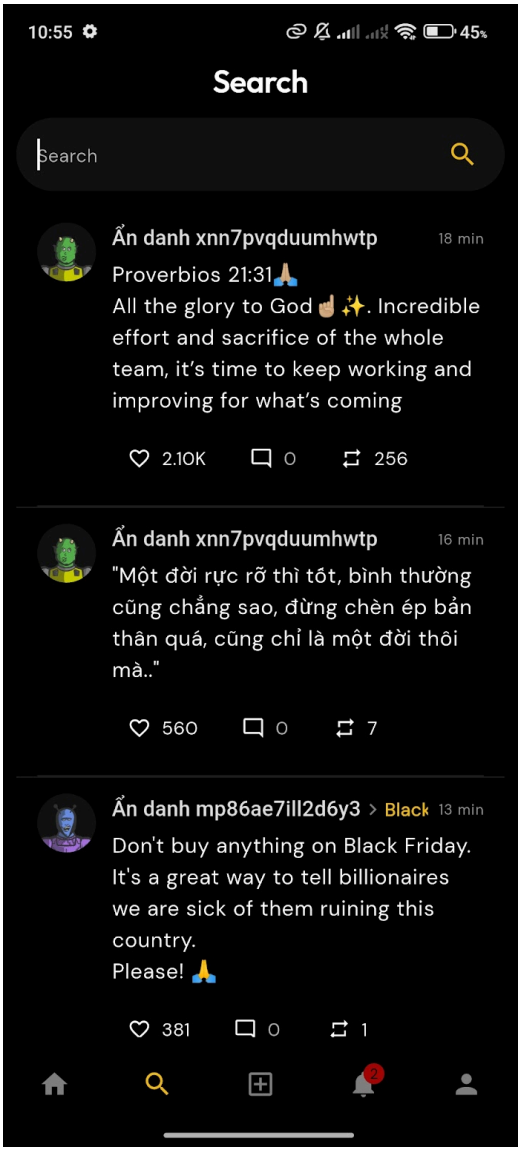
- Remote Data: Yes, this feature reads and stores data remotely in the PocketBase backend. It interacts primarily with the comments collection, and also updates the posts collection (to update comment counts) and likes collection.
- Data Table Structure (Remote): The data is stored in the comments collection. Based on the Comment model and toPocketBase method, the structure includes:
    - id (Text): Unique record identifier.

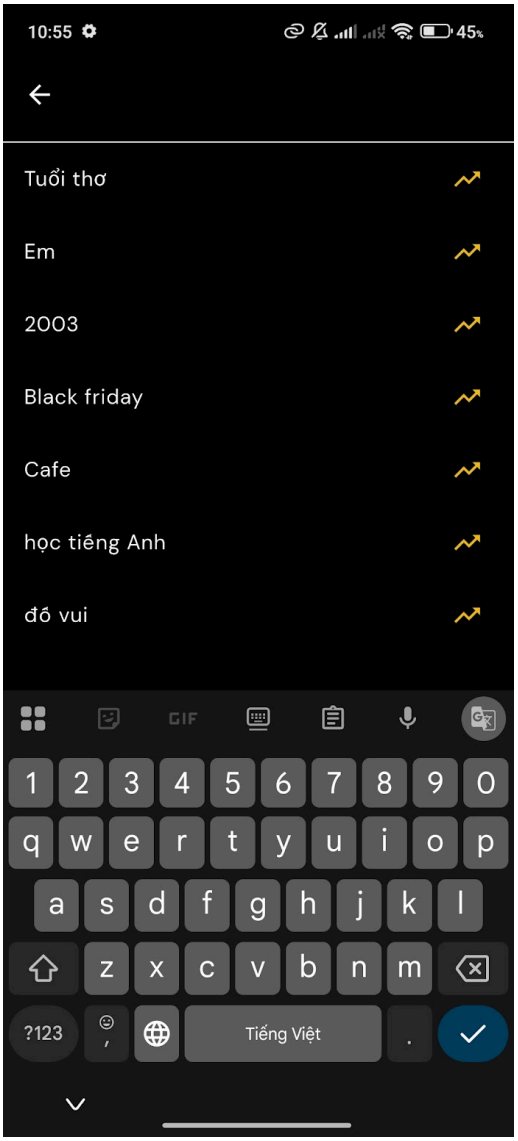
        - postId (Relation): The ID of the post this comment belongs to.
        - userId (Relation): The ID of the user who created the comment.
        - content (Text): The actual text content of the comment.
        - parentId (Relation): The ID of the immediate parent comment (if it's a reply).
        - rootId (Relation): The ID of the top-level ancestor comment (used for fetching entire threads).
        - likesCount (Number): Counter for the number of likes.
        - replyCount (Number): Counter for the number of replies.
        - created / updated (DateTime): Timestamps.
    - REST API Description: The app uses the PocketBase SDK to interact with the API via CommentService:
        - Fetch Root Comments:
            - How to call: pb.collection('comments').getList(...)
            - Input: Filter postId="<id>" && (parentId="" || parentId=null), sort by -replyCount,-likesCount,-created, and expand userId.
            - Output: Returns a list of comment records, which are mapped to Comment objects.
        - Fetch Replies:
            - How to call: pb.collection('comments').getFullList(...)
            - Input: Filter rootId="<id>" to get all replies in a thread.
            - Output: Returns a list of nested comment records.
        - Add Comment:
            - How to call: pb.collection('comments').create(...)
            - Input: A JSON body containing postId, userId, content, parentId, and rootId.
            - Output: Returns the created comment record. Note: The service also makes additional API calls to update the replyCount of the parent comment and the comments count of the post.

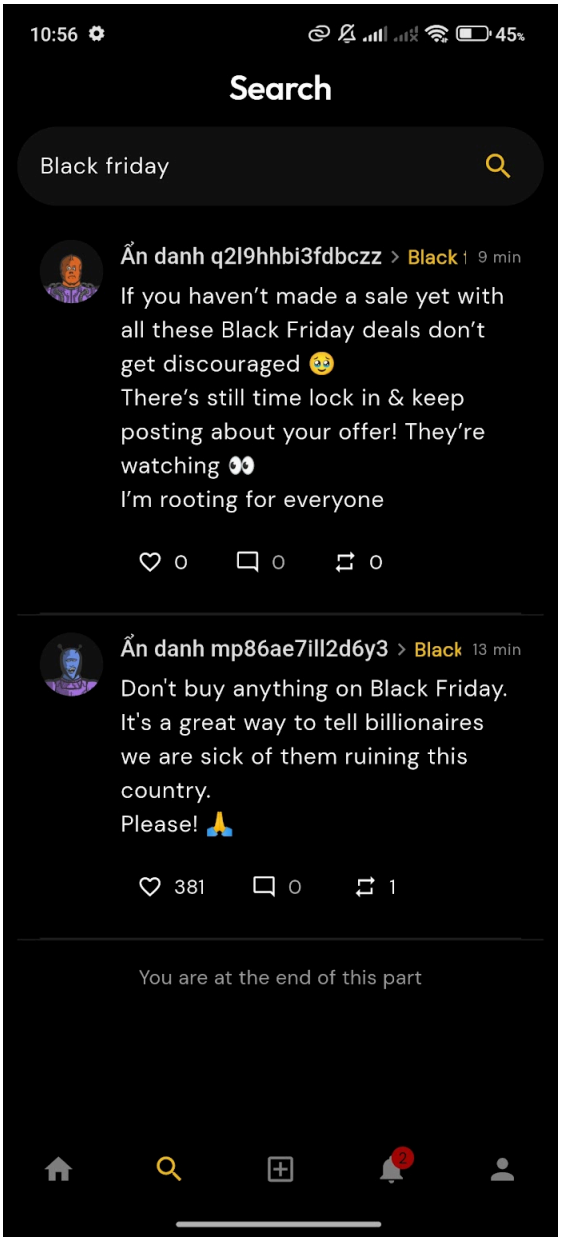
4. Search Page:

- Description: This feature is the Search Screen, which allows users to search for posts or topics within the application. It displays search suggestions and dynamically loads results based on user input. Users can view trending topics and tap on suggestions to explore related posts.

- Screenshots:


10:55
45%
Search
Search
Ẩn danh xnn7pvqduumhwtp
18 min
Proverbios 21:31🙏
All the glory to God☝🏻✨. Incredible effort and sacrifice of the whole team, it's time to keep working and improving for what's coming
2.10K
0
256
Ẩn danh xnn7pvqduumhwtp
16 min
"Một đời rực rỡ thì tốt, bình thường cũng chẳng sao, đừng chèn ép bản thân quá, cũng chỉ là một đời thôi mà.."
560
0
7
Ẩn danh mp86ae7ill2d6y3 › Black
13 min
Don't buy anything on Black Friday. It's a great way to tell billionaires we are sick of them ruining this country.
Please! 🙏
381
0
1


10:55
45%
Tuổi thơ
Em
2003
Black friday
Cafe
học tiếng Anh
đố vui
GIF
1 2 3 4 5 6 7 8 9 0
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123
Tiếng Việt



- Implementation details: students should answer the following questions:

+ List the widgets used for this feature/page. Is there any special widget (not introduced in the lesson) used? If so, state them.

- List of widgets used: Scaffold, SafeArea, Padding, Column, Text, SizedBox, Expanded, Center, CircularProgressIndicator, ListView.separated, Divider, ListTile, Icon, IconButton, Focus.
- Special widgets:
  - SearchAnchor & SearchBar: These are modern Material 3 widgets used to create the search experience. SearchAnchor manages the suggestion view that expands when the user taps the search bar, while SearchBar provides the input field styling and behavior.
  - SinglePostItem: A custom reusable widget used to display the search results (posts), ensuring consistency with the News Feed.

    - Provider methods (context.watch, context.read): Used to connect the UI to the SearchManager for listening to state changes (like loading status or search results) and triggering search actions.

+ Does the feature use any libraries or plugins? If so, state them and state the role of those libraries/plugins.

- provider: This is the core state management library. It injects the SearchManager into the widget tree, allowing the SearchScreen to listen for changes (like when search results are loaded or the loading state changes) and rebuild the UI automatically. It also facilitates the communication between SearchManager and PostsManager to synchronize the "Like" and "Repost" status of posts.
- pocketbase: This is the backend SDK used to communicate with the server. In the search feature, it is used within PostService to query the database for posts matching specific keywords (findPostByKeywords) or to fetch trending posts (fetchTrendingPosts). It is also used in TopicService to fetch topic suggestions (fetchTopics) for the search bar.
- go_router: This library manages the application's routing. In the search page, it handles the navigation when a user taps on a specific post in the results list to view its full details (AppRouteName.detailPost).
- flutter_dotenv: This plugin is used to load environment variables (specifically the POCKETBASE_URL). This is essential for the SinglePostItem widget (used in the search results) to correctly construct the full URL for displaying post images.

+ Does this feature use a shared state management solution? If so, state briefly how your solution works and describe the code architecture.

- ViewModel: The SearchManager holds the state (_localSearchResults, topicSuggestions, _isSearching). It contains the business logic for fetching trending posts (when the query is empty) or filtering posts by keywords.
- Synchronization: A unique aspect is that SearchManager uses the update method (via ChangeNotifierProxyProvider) to link with PostsManager. This allows it to check if a post in the search results already exists in the Feed. If so, it reuses that post object to ensure that "Like" or "Repost" statuses remain synchronized across both screens.

+ Does this feature read or store any data? Locally or remotely? If so, state the data table structure. If you are using a REST API, briefly describe that API (how to call, input, output).

. Data Storage Locations

- Remote Data: The feature reads data from the PocketBase backend database. It queries the posts collection for search results and the topics collection for suggestions.
- Local Data: There is no persistent local storage code (like SQLite) specifically for the Search Page in the provided source. However, it reads in-memory state from PostsManager to synchronize the "Like" and "Repost" status of posts if they are already loaded in the Feed.
- Data Table Structure (Remote)
    - The feature interacts with two main collections in PocketBase:
    - posts Collection: Stores the content users search for.
        - id (Text): Unique ID.

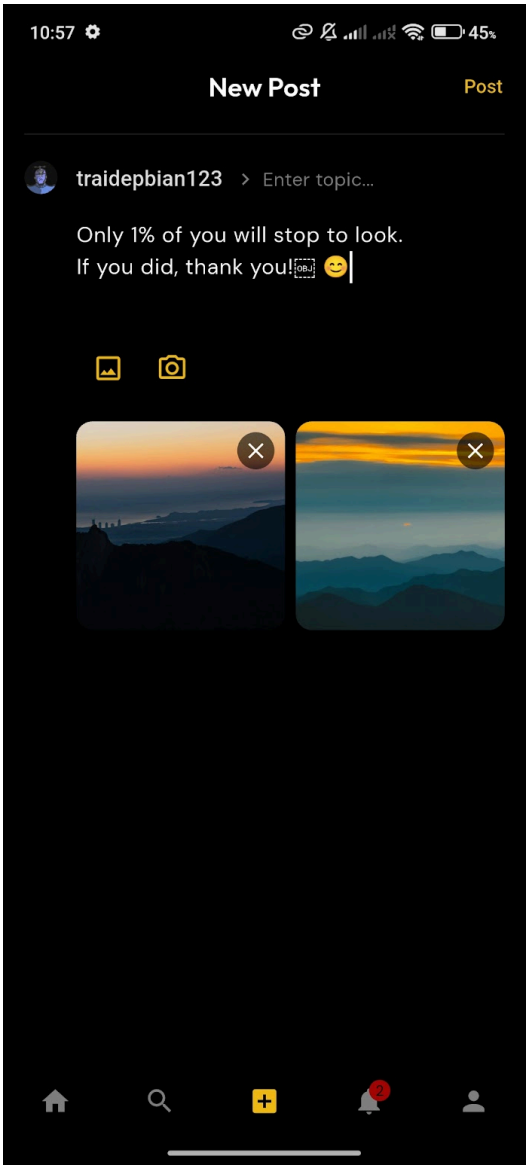
- content (Text): The body text of the post (indexed for search).
- topicId (Relation): Links to the topics table (used to filter by topic name).
- userId (Relation): The author of the post.
- likes, comments, reposts (Number): Engagement metrics used for sorting trending posts.
- created (DateTime): Used for sorting recent results.
- topics Collection: Stores topic names for search suggestions.
- id (Text): Unique ID.
- name (Text): The name of the topic (e.g., "Technology", "Life").

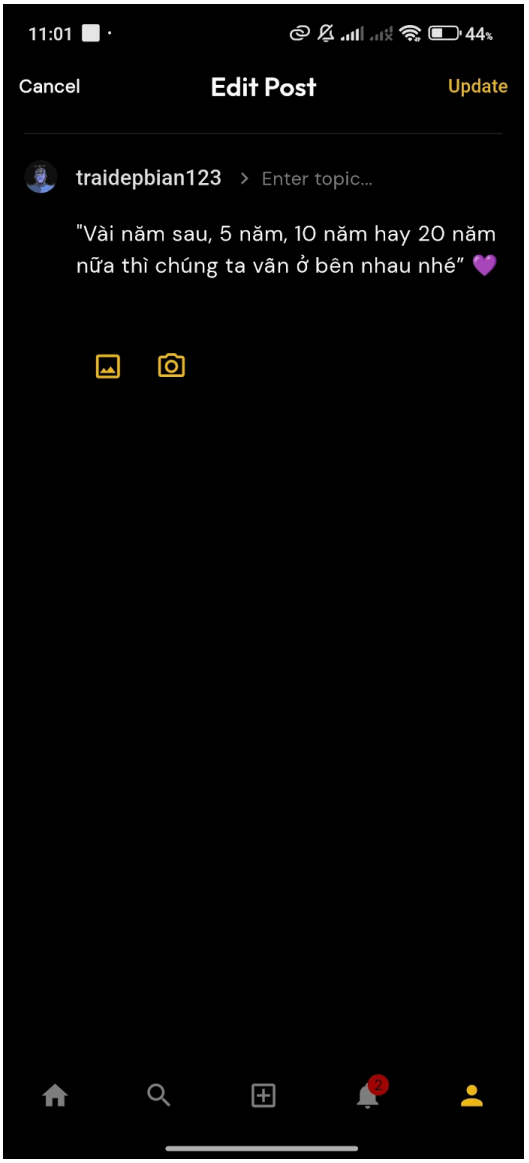
- REST API Description
  - The app uses the PocketBase SDK (which wraps the REST API) via PostService and TopicService.
  - Search Posts by Keyword:
    - How to call: pb.collection('posts').getList(...).
    - Input:
    - filter: A dynamic string constructed from keywords, e.g., content ?~ "keyword" || topicId.name ?~ "keyword". This performs a "contains" query on both post content and topic names.
    - expand: 'userId,topicId,posts_via_topicId' to retrieve related data in one request.
    - Output: Returns a list of Post records matching the query.
  - Fetch Trending Posts (Default Empty Search):
    - How to call: pb.collection('posts').getList(...).
    - Input:
    - sort: '-likes,-comments,-reposts,-created' (Sorts by highest engagement first).
    - Output: Returns a list of the most popular Post records.
  - Fetch Topic Suggestions:
    - How to call: pb.collection('topics').getList(...).
    - Input: sort: '-created' (to get the latest topics).
    - Output: Returns a list of Topic records used for the search bar's auto-complete suggestions.

5. Post Screen:

- Description: This is the Post Creation Page, where users can create a new post. It allows users to type their post content, choose a topic, and then submit it.

- Screenshots:





- Implementation details: students should answer the following questions:

+ List the widgets used for this feature/page. Is there any special widget (not introduced in the lesson) used? If so, state them.

- **List of widgets used**: Scaffold, AppBar, Text, TextButton, SafeArea, GestureDetector, Padding, Column, Divider, SizedBox, Expanded, SingleChildScrollView, Row, Flexible, Icon, TextField, TextFormField, IconButton, GridView.builder, Stack, ClipRRect, Image, Positioned, Container, SnackBar.
- **Special widget:**
  - Avatar: A custom reusable widget to display the current user's profile picture consistently.
  - GridView.builder: Used to display the selected images in a responsive grid layout.
  - Stack & Positioned: Used to overlay the "remove" (X) button on top of the selected image thumbnails.

    - Correction: Unlike the user's draft, FutureBuilder is not used in this screen. Instead, asynchronous operations (posting) are handled using a local state variable _isPosting and setState.

+ Does the feature use any libraries or plugins? If so, state them and state the role of those libraries/plugins.

- image_picker: This plugin is crucial for this screen. It allows the user to pick multiple images from the gallery (pickMultiImage) or take a new photo using the camera (pickImage) to attach to their post.
- provider: Used to access the AuthManager (to get the current user's ID) and the PostsManager (to trigger the create/update post logic).
- go_router: Handles navigation logic, such as popping the screen after a successful post or cancelling the action.
- http: Indirectly used via PostService to handle MultipartFile uploads for the images.

+ Does this feature use a shared state management solution? If so, state briefly how your solution works and describe the code architecture.

- Local State: The PostScreen uses StatefulWidget to manage transient data like the text input (_contentController, _topicController), the list of selected images (_selectedImages), and the loading status (_isPosting).
- Shared State (Provider): When the user submits the post, the screen calls postsManager.createPost() or postsManager.updatePost(). The PostsManager (ViewModel) handles the API communication and then updates its global list of posts, notifying listeners (like the Feed Screen) to refresh automatically.

+ Does this feature read or store any data? Locally or remotely? If so, state the data table structure. If you are using a REST API, briefly describe that API (how to call, input, output).

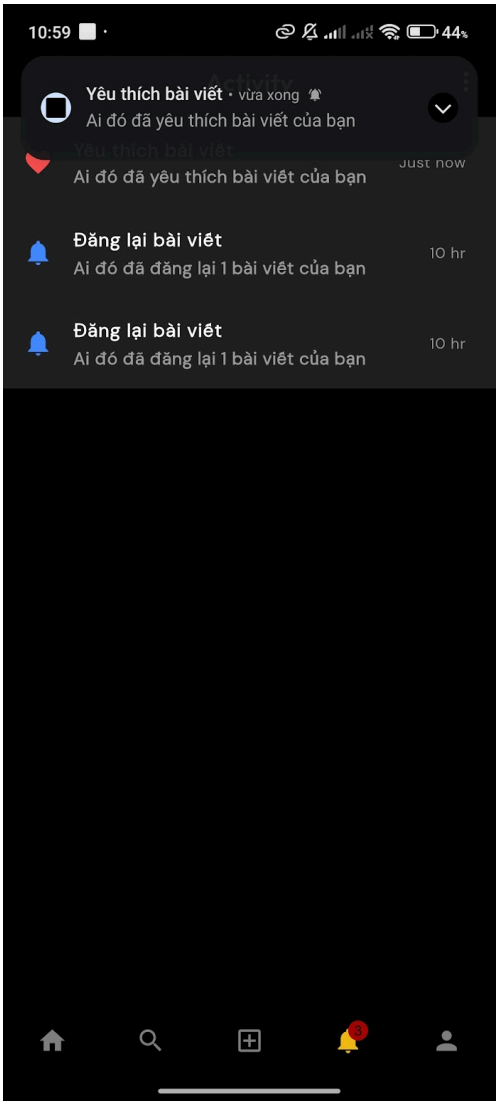
- Data Storage Locations:
    - Remote Storage: The primary storage is the PocketBase backend database. Data is saved into the posts and topics collections.
    - Local Storage: The app does not persist draft posts locally using a database like SQLite. However, it manages temporary local state (text input, selected image files) within the PostScreen widget while the user is editing.
- Data Table Structure (Remote): The feature interacts with two collections in PocketBase:
    - posts Collection:
        - id (Text): Unique identifier.
        - content (Text): The main text content of the post.
        - userId (Relation): Links the post to the user creating it.
        - topicId (Relation): Links the post to a specific topic (optional).
        - images (File): Stores the uploaded image files (supports multiple files).
        - topics Collection:
        - name (Text): The name of the topic (e.g., "Travel", "Tech").
- REST API Description: The app uses the PocketBase SDK (wrapping the REST API) via the PostService class.
    - Create Post:
        - How to call: pb.collection('posts').create(...).

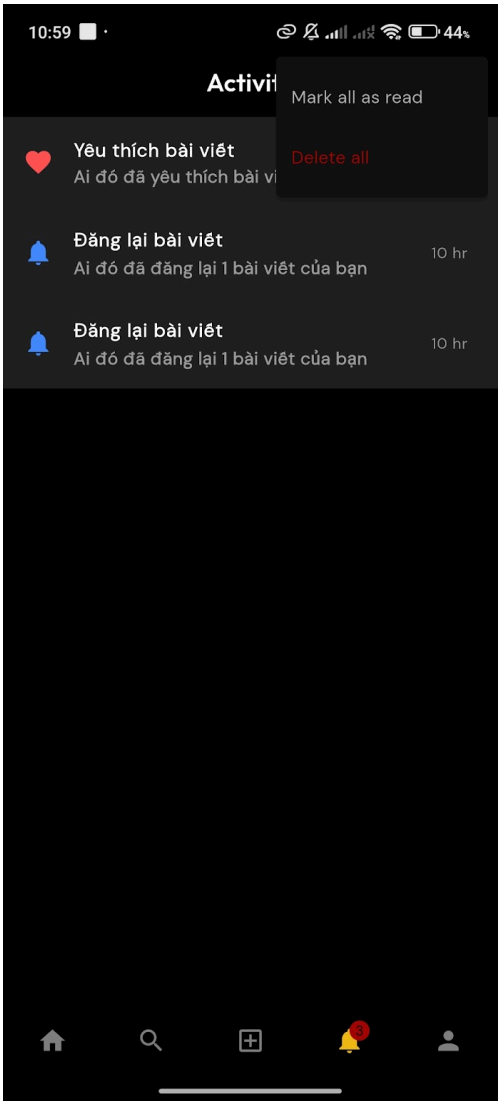
    - Input: A MultipartRequest is constructed containing:Body fields: content, userId, and topicId (if a topic is selected). Files: A list of http.MultipartFile objects for the images field.
    - Output: Returns the created RecordModel, which is then converted into a Post object to update the local feed immediately.
- Find or Create Topic (Helper Logic):
    - How to call: Before creating a post, the app checks if the entered topic exists using pb.collection('topics').getList(filter: 'name="..."').
    - Logic: If the topic exists, its ID is used. If not, a new topic is created via pb.collection('topics').create(...), and the new ID is used for the post.

6. Notification

- Description: This is the Activity Page, where users can view their recent activities such as all interactions, followed users, and replies. It helps users track engagement and updates in the app.

- Screenshots:





- Implementation details: students should answer the following questions:

+ List the widgets used for this feature/page. Is there any special widget (not introduced in the lesson) used? If so, state them.

- **List of widgets used**: Scaffold, AppBar, Text, Column, Expanded, ListView.builder, Center, Icon, Padding, Container, ListTile, AlertDialog, TextButton, PopupMenuButton, PopupMenuItem.
- **Special widgets**:
  - Dismissible: A specific widget used in NotificationItem to implement the "Swipe-to-delete" gesture. It handles the background UI (showing the red delete icon) and the confirmation dialog logic before removing the item.
  - Badge: Used in the HomePageScreen (Shell) to display the unread count on the bottom navigation bar icon, connected to the NotificationManager.

+ Does the feature use any libraries or plugins? If so, state them and state the role of those libraries/plugins.

- sqflite: This is a unique library for this feature. It is used to create a local SQLite database to cache notifications, allowing users to view their history offline and reducing server load.
- flutter_local_notifications: Another unique plugin. It is used to show system-level push notifications (in the phone's status bar) when a new event arrives via the real-time stream, even if the user is not looking at the app.
- provider: Connects the NotificationScreen to the NotificationManager. It allows the UI to update automatically when new notifications arrive via real-time stream or when the unread count changes.
- pocketbase: Handles the remote data synchronization. It fetches the list of notifications and, crucially, manages the Realtime Subscription (subscribe) to listen for new events (likes, comments) instantly.
- path: Used by SqliteService to correctly determine the file system path for storing the local database file (notifications.db).

+ Does this feature use a shared state management solution? If so, state briefly how your solution works and describe the code architecture.

- ViewModel: The NotificationManager coordinates data from three sources: Local DB (SQLite), Remote API (PocketBase), and Realtime Stream.
- Workflow:
    - Init: When the app starts, it loads data from SQLite immediately for instant UI feedback.
    - Sync: It then fetches the latest data from PocketBase to update the local cache and the UI.
    - Realtime: It listens to the PocketBase stream. When a new event (create) occurs, it saves it to SQLite, updates the in-memory list, and triggers a Local Notification.
    - View: The UI listens to notifyListeners() to refresh the list and update the unread badge count.

+ Does this feature read or store any data? Locally or remotely? If so, state the data table structure. If you are using a REST API, briefly describe that API (how to call, input, output).

- Data Storage Locations:
    - Remote Storage: Notifications are stored in the PocketBase backend database in the notifications collection.
    - Local Storage: The app uses SQLite (via the sqflite library) to persist notifications locally. This allows users to view their notification history even without an internet connection.
- Data Table Structure:
+ Remote (PocketBase): The notifications collection contains the following fields:
    - id (Text): Unique record identifier.
    - userId (Relation): The ID of the user receiving the notification.
    - title (Text): The title of the notification (e.g., "New Comment").
    - body (Text): The detail text of the notification.
    - type (Text): The category of notification (e.g., post, comment, like, reply).
    - targetId (Text): The ID of the related object (e.g., the postId to navigate to).
    - isRead (Boolean): Status indicating if the user has read the notification.
    - created / updated (DateTime): Timestamps.

+ Local (SQLite): The local notifications table mirrors the remote structure but adapts data types for SQLite:
    - id (TEXT PRIMARY KEY), userId (TEXT), title (TEXT), body (TEXT), type (TEXT), targetId (TEXT), created (TEXT), updated (TEXT).
    - isRead is stored as an INTEGER (0 for false, 1 for true) since SQLite does not have a native boolean type.
- REST API Description: The app interacts with the PocketBase SDK, which wraps the REST API calls via PocketBaseNotificationService:
+ Subscribe (Realtime):
    - How to call: pb.collection('notifications').subscribe('*', ...)
    - Input: The wildcard '*' subscribes to all events, but the client logic filters to only process events where userId matches the current user.
    - Output: Returns a RecordSubscriptionEvent whenever a new notification is created on the server.
+ Fetch Notifications:
    - How to call: pb.collection('notifications').getList(...)
    - Input: A query filter {'userId': userId} to retrieve only the current user's notifications.
    - Output: Returns a list of notification records.
+ Mark as Read:
    - How to call: pb.collection('notifications').update(...)
    - Input: The notificationId and a body {'isRead': true}.
    - Output: Returns the updated record.
+ Create Notification (Triggered by Actions):
    - How to call: pb.collection('notifications').create(...) (called from CommentManager or PostsManager).
    - Input: A JSON body containing the notification details (title, body, targetId, etc.).
    - Output: Returns the newly created notification record.

7. Profile Page:

- Description: This is the Profile Page, where users can view and manage their personal information such as name, username, bio, and follower count. Users can also edit or share their profile, and view different tabs for posts, replies, and reposts.

- Screenshots:

10:59 44%

traidepbian123
lttri@gmail.com

Posts | Replied | Reposts

traidepbian123 17 min

"Vài năm sau, 5 năm, 10 năm hay 20 năm nữa thì chúng ta vẫn ở bên nhau nhé" 💜

4 0 2

Ẩn danh vijw7687gaxmz6p 13 hr

đố mn hôm nay là ngày gì??

1 1 1

10:59 44%

traidepbian123
lttri@gmail.com

Posts | Replied | Reposts

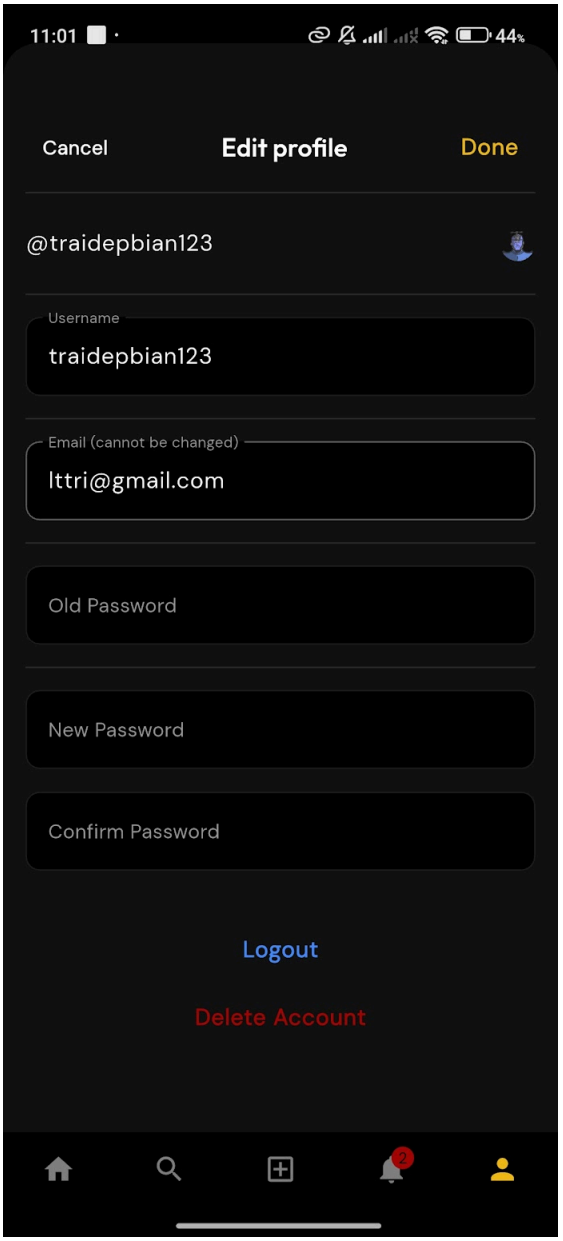
Ẩn danh cf1g47wehwfuqv0 23 min

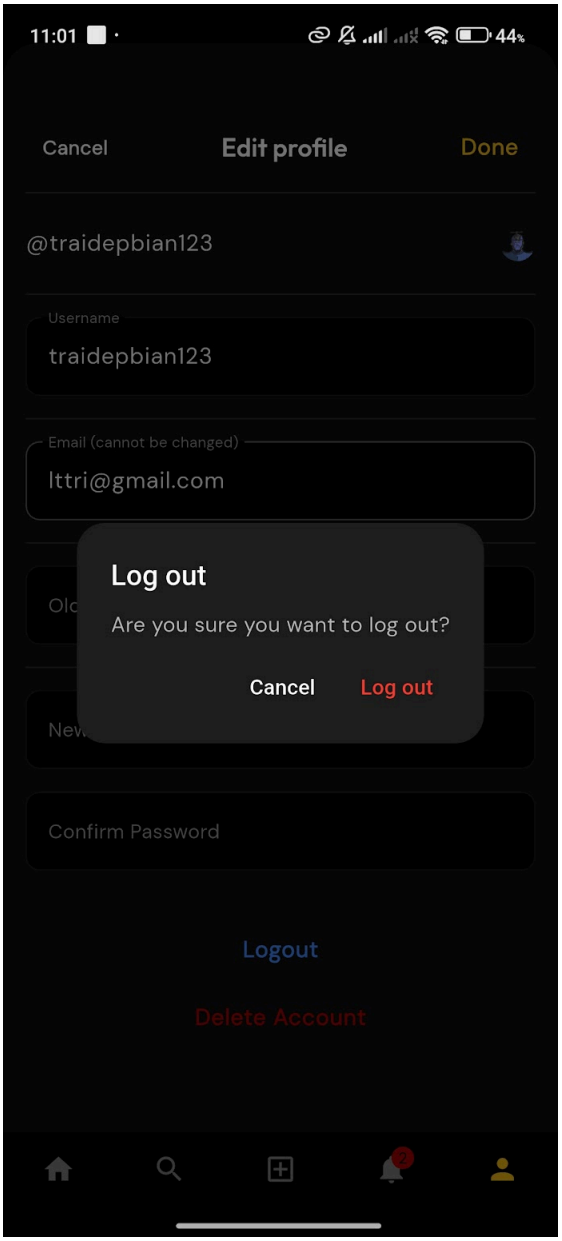
Quán này chill dữ vị

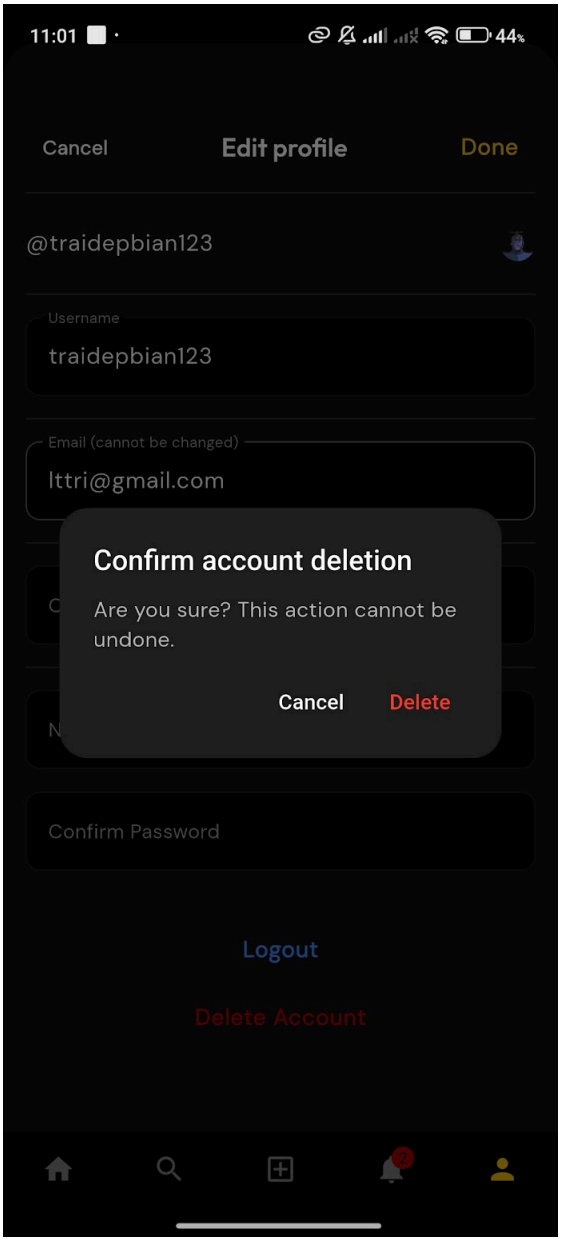
14 1 4

Quán này ở đâu vậy bạn?


11:00
44%
traidepbian123
lttri@gmail.com
Posts
Replied
Reposts
Ẩn danh k98oljiutknhtcf
13 min
Các bạn 2003 ơi, mng có ổn không? Mình thì không ạ, mình tốt nghiệp hồi tháng 8 đến bây giờ vẫn chưa tìm được việc. Trong tg thất nghiệp mình cũng đi thực tập được 3 tháng rồi 😔. Mình cứ chọn đại thế thôi mình cảm thấy bản thân không có mục tiêu định hướng rõ ràng, không biết bản thân thích gì nữa. Mình tính nếu vẫn không tìm được việc chắc sẽ đi làm 1 công việc part time nào đó. Gđ có khuyên mình về quê nhưng mà mình không muốn về lắm mình cảm thấy ch tới lúc 😭
312
0
35
Ẩn danh ah35srbncqa1dac
12 min
In English, we say: “I wish you the best.”
But in poetry, we say: “May your sky


11:01
44%
Cancel
Edit profile
Done
@traidepbian123
Username
traidepbian123
Email (cannot be changed)
lttri@gmail.com
Old Password
New Password
Confirm Password
Logout
Delete Account





- Implementation details: students should answer the following questions:

+ List the widgets used for this feature/page. Is there any special widget (not introduced in the lesson) used? If so, state them.

- List of widgets used: Scaffold, AppBar, SafeArea, Column, Row, Expanded, Padding, Container, SizedBox, Text, Icon, IconButton, Divider, Center, CircularProgressIndicator, TabBar, TabBarView, Tab, ListView.separated, RefreshIndicator, TextButton, TextFormField.
- Special widgets:
  - SlidingUpPanel: Used to contain the entire EditProfileScreen. This creates a modern, non-modal sliding drawer effect for managing account settings.
  - DefaultTabController: Manages the three distinct tabs within the profile (Posts, Replied, Reposts).

    - ProfileHeader / EditProfileScreen / ProfilePostList: These are custom reusable widgets designed specifically for the Profile layout.
    - Avatar: A custom widget used to display the user's profile picture or a generated RoboHash image.

+ Does the feature use any libraries or plugins? If so, state them and state the role of those libraries/plugins.

    - provider: The state management solution. It connects the UI to the ProfileManager and PostsManager to listen for user data changes, loading states, and dynamically update post/repost lists.
    - sliding_up_panel: Provides the modern sliding panel widget used to display the EditProfileScreen non-modally over the main profile view.
    - pocketbase: The Backend SDK used by UserService to fetch the latest user data, handle profile updates (username, avatar), process password changes, and manage account deletion.
    - go_router: Handles navigation logic after critical actions, such as redirecting to the Authentication screen upon logging out or deleting the account.
    - image_picker: Used to access the device's camera or gallery if the user chooses to update their profile picture/avatar.

+ Does this feature use a shared state management solution? If so, state briefly how your solution works and describe the code architecture.

- ViewModel (ProfileManager): Serves as the central logic hub. It holds the profile state (user, isLoading) and manages the text controllers for the edit form. It delegates API calls related to user data (load, update, delete) to UserService.
- Data Synchronization: ProfileScreen loads data by calling vm.loadUser() in initState. It also subscribes to PostsManager to fetch the user's unique lists of posts, replies, and reposts. ProfileManager uses the provided user object for read-write operations (Edit Profile) and calls the necessary UserService methods for persistence.

+ Does this feature read or store any data? Locally or remotely? If so, state the data table structure. If you are using a REST API, briefly describe that API (how to call, input, output).

- Data Storage: Data is stored remotely in the PocketBase backend.
- Collections Used:
    - users Collection: Read and Write operations are mandatory for profile updates (username, email, password, avatar).
    - posts, comments, reposts Collections: Read-only access is used to compile the user's timeline activity.
- REST API Description (Via SDK):
    - Update Profile: Invoked via _userService.updateUser(...).
        - How to call: pb.collection('users').update(userId, body: {...}).
        - Input: User ID and a JSON/Multipart body containing fields to update (e.g., username) and potentially the avatarFile.
        - Output: Returns the updated user record.
    - Change Password: Invoked via _userService.changePassword(...).

    - How to call: pb.collection('users').update(userId, body: {oldPassword, password, passwordConfirm}).
    - Output: Returns the updated record upon successful password change.
- Delete Account: Invoked via _userService.deleteUser().
    - How to call: pb.collection('users').delete(userId).
    - Output: Successful API call returns null or a success status code.