PROJECT REPORT

SEMESTER 1, ACADEMIC YEAR: 2025-2026

CT312H: MOBILE PROGRAMMING

- **Project/Application name: Abacus (Spending Management)**
- **GitHub links:** https://github.com/25-26Sem1-Courses/ct312hm01-project-AndrewNguyenITVN.git
- **Student ID 1: B2203438**
- **Student Name 1: Huỳnh Tấn Đạt**
- **Student ID 2: B2205896**
- **Student Name 2: Nguyễn Minh Nhựt**
- **Class/Group Number (e.g, CT312HM01): CT312HM01**
- **Video links:** https://youtu.be/wzkd5Z7rRwo

I. Introduction

- Project/application description: Abacus is a mobile spending management application built with Flutter. The system allows users to seamlessly track personal and group expenditures. Key features include user authentication, an interactive dashboard with financial summaries, transaction management, budget planning, and insightful reports with charts.
- A task assignment sheet for each member if working in groups.

No.	Task Description	Member
1	Initialize the project	Nguyen Minh Nhut
2	Design the Login and Signup page	Nguyen Minh Nhut
3	Design the Account page	Huynh Tan Dat Nguyen Minh Nhut
4	Design the Home page	Huynh Tan Dat
5	Design the Transaction page	Huynh Tan Dat
6	Design the Saving Goal page	Nguyen Minh Nhut
7	Design the Categories page	Huynh Tan Dat Nguyen Minh Nhut
8	Setup Pocketbase and SQLite for data storage	Nguyen Minh Nhut
9	Responsive and animation	Huynh Tan Dat

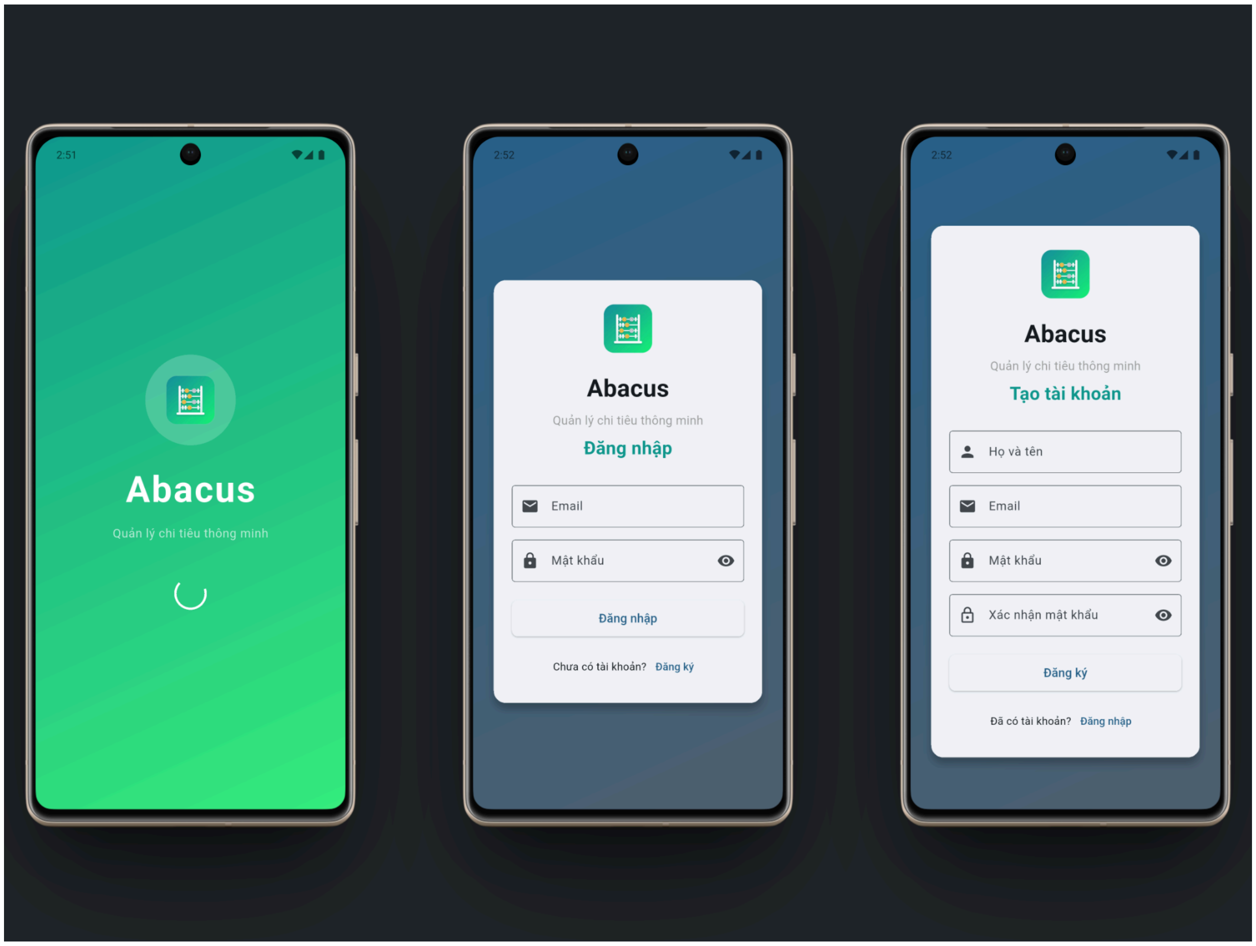
10	Local notification	Nguyen Minh Nhut
11	Design the theme mode, app icon & splash screen	Nguyen Minh Nhut Huynh Tan Dat

II. Details of implemented features

1. Authentication

- Description: This feature allows users to sign up for a new account and log in to the system to access personal financial management features. User information is securely stored, and the login session is maintained for user convenience.

- Screenshots:



Abacus
Quản lý chi tiêu thông minh
Abacus
Quản lý chi tiêu thông minh
Đăng nhập
Email
Mật khẩu
Đăng nhập
Chưa có tài khoản? Đăng ký
Abacus
Quản lý chi tiêu thông minh
Tạo tài khoản
Họ và tên
Email
Mật khẩu
Xác nhận mật khẩu
Đăng ký
Đã có tài khoản? Đăng nhập

Splash, Login and Sign Up screens

- Implementation details:

+ List the widgets used for this feature/page:
    - LoginScreen / SignupScreen:
        - Scaffold: The main structure of the screen.
        - Container: Creates a gradient background (LinearGradient) for visual appeal.
        - SafeArea: Ensures content is not obscured by notches or status bars.
        - SingleChildScrollView: Allows scrolling when the keyboard appears.
        - AuthCard (Custom Widget): Contains the entire login/signup form logic.
    - AuthCard:
        - Card: Creates a container for the form with shadow effects (elevation).
        - Form: Manages state and validation of input fields.
        - TextFormField: Input fields for Email, Password, Name (using controller and validator).
        - Icon: Displays decorative icons (wallet (account_balance_wallet), lock, person).
        - ValueListenableBuilder: Listens to _isSubmitting state to show the loading spinner (CircularProgressIndicator) without rebuilding the entire widget.
        - ElevatedButton: The "Login" or "Signup" submission button.
        - TextButton: Button to switch between Login and Signup modes.

+ Libraries or plugins used:
    - pocketbase: The core library for interacting with the PocketBase Backend, handling user authentication, token storage, and session management.
    - provider: Uses context.read<AuthManager>() to invoke login/signup functions and ChangeNotifierProvider for global authentication state management.
    - flutter_dotenv: Loads environment variables (Backend URL) during service initialization.

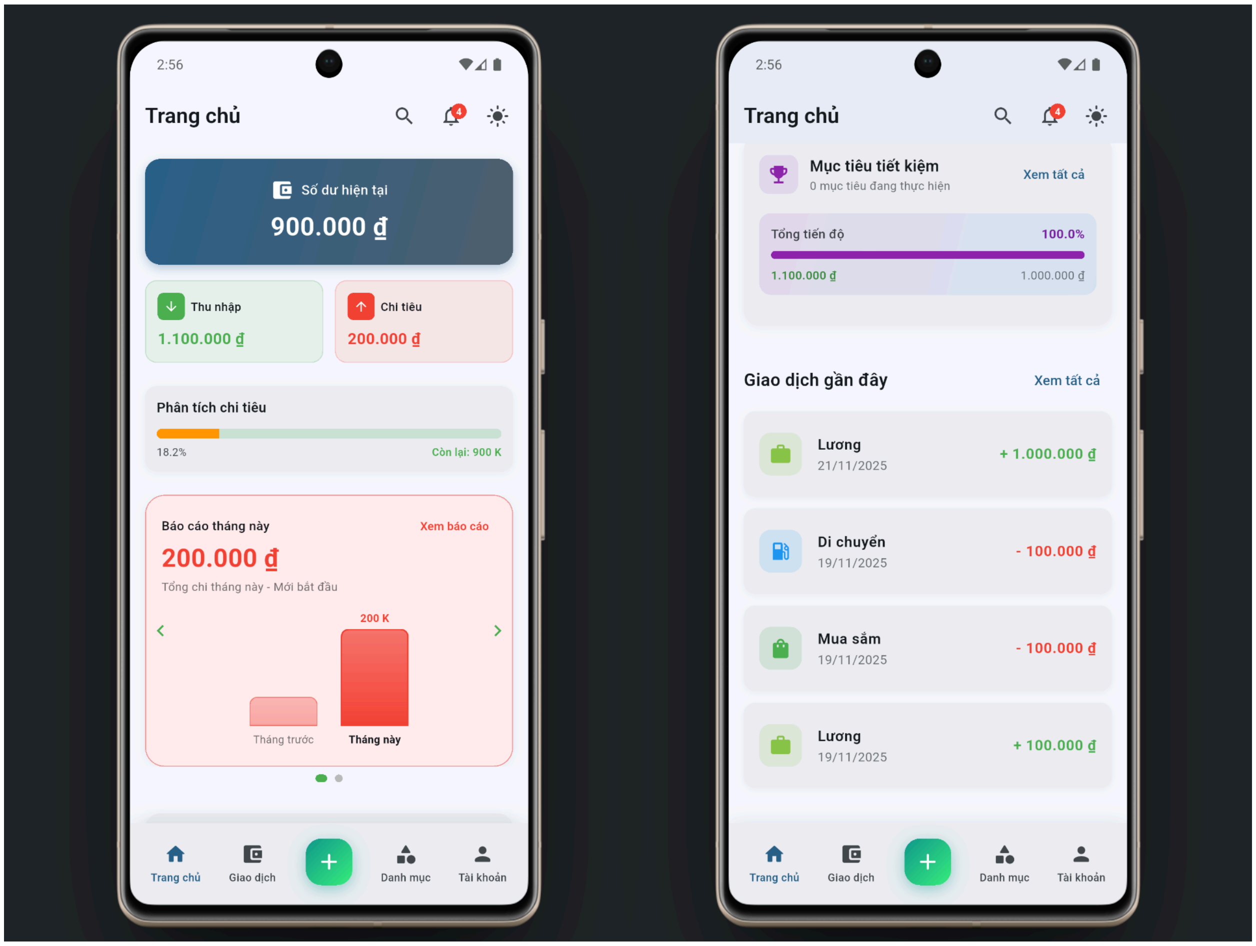
- go_router: Manages navigation and routing, automatically redirecting users to the home screen upon successful login.

+ Shared state management solution:
  - This feature uses Provider for state management.
  - The AuthManager class extends ChangeNotifier. It holds a _loggedInUser variable (type Account?) representing the current user.
  - It implements a tryAutoLogin() method used by the Splash Screen to restore the user session from local storage automatically.
  - Upon successful login or logout via AuthService, AuthManager calls notifyListeners() to notify the entire application to update the UI accordingly (e.g., switching from Login screen to Home screen).
+ Data reading or storage:
  - Remote (PocketBase):
    - Interacts with the users collection in PocketBase.
    - users table structure: id, username, email, name, password, verified.
    - API Calls:
      - pb.collection('users').authWithPassword(email, password): Sends a login request to retrieve a token.
      - pb.collection('users').create(body: {...}): Sends a request to create a new user.
      - pb.collection('users').authRefresh(): Refreshes the token and retrieves the latest user data.
  - Local:
    - PocketBase's AuthStore automatically persists the authentication Token to the device storage (implicitly using SharedPreferences within the plugin) to maintain login state when the app is closed.

2. Home Screen

- Description: This is the main screen of the application, providing a comprehensive financial overview for the user. It displays the current balance, a summary of total income and expenses, a spending ratio analysis chart, sliding monthly comparison report cards, savings goals, and a list of recent transactions.

- Screenshots:



Home screen

- Implementation details:

+ List the widgets used for this feature/page:
    - HomeScreen (Main Screen):
        - Scaffold: The main structure.
        - OrientationBuilder: Supports different layouts for Portrait and Landscape modes.

    - SingleChildScrollView & Column: Vertical scrolling structure containing child components.
    - AppBar: Header bar containing search and notification buttons (with a Stack badge for unread count).
  - HomeSummaryCard (Custom Widget):
    - Uses enum SummaryCardType to reuse a single widget for 3 card types: Balance, Income/Expense, and SpendingAnalysis.
    - Container, BoxDecoration, LinearGradient: Creates styled backgrounds and shadows.
    - LinearProgressIndicator: Progress bar displaying the spending-to-income ratio.
  - MonthlyReportCard (Sliding Reports):
    - PageView: Creates a swiping effect between report cards (Expense & Income).
    - Stack & Positioned: Overlays navigation arrow buttons on top of the PageView.
    - Custom Bar Chart (_buildBarColumn): Uses Container with dynamic height to compare current vs. previous month data.
  - RecentTransactionsList (Transaction List):
    - ListView.builder: Efficiently displays the list of transactions.
    - ListTile: Displays individual transaction items with category icon, date, and amount.
    - CircleAvatar simulated via rounded Container: Displays category icons with corresponding background colors.
  - SavingsGoalsBlock: Widget displaying a summary of savings goals detailed further in the Savings section.

+ Libraries or plugins used:
  - provider: Uses context.watch<TransactionsManager>() and context.watch<CategoriesManager>() to listen to and automatically update the UI when data changes.
  - intl: Formats currency (VND) and dates (dd/MM/yyyy).

+ Shared state management solution:
  - Uses Provider.

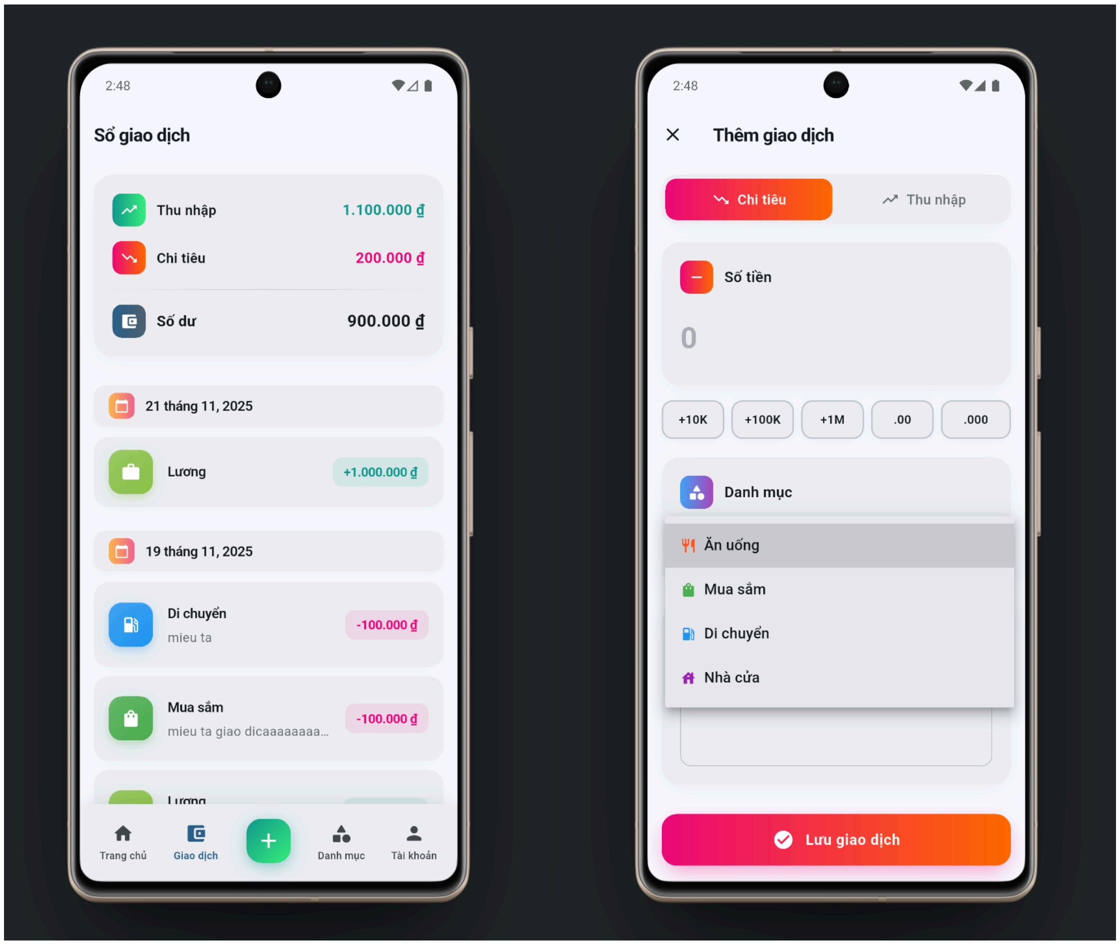
- TransactionsManager: Provides data on balance, total income/expense, previous month comparison data, and recent transactions.
- ThemeManager: Manages the isDarkMode state, allowing the user to toggle the entire app's visual theme directly from the AppBar.
- CategoriesManager: Provides category information (name, icon, color) to display alongside transactions.
- NotificationsManager: Manages the unread notification count shown on the bell icon.

+ Data reading or storage:
    - Data is read from TransactionsManager (which is loaded from the server upon app startup).
    - Statistical calculations (totals, monthly comparisons) are performed directly within the Manager (Client-side logic) based on the loaded transaction list.

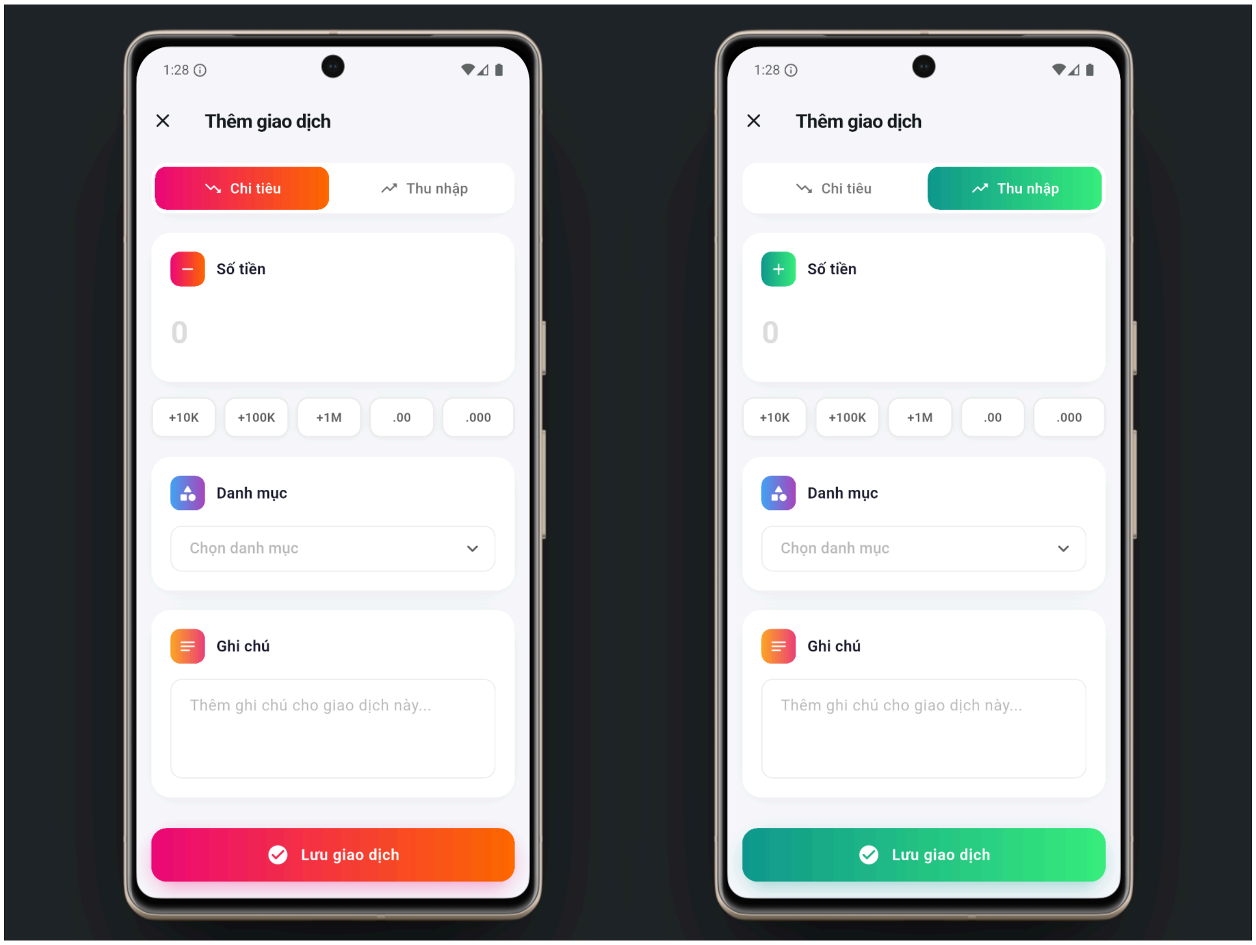
3. Transactions

- Description: This is the core feature of the application, allowing users to manage daily income and expense activities in detail. Users can add new transactions, edit, and delete existing ones. The system supports transaction classification (Income/Expense), category selection, amount input with quick suggestions, and adding notes. Transaction data is also used to monitor and send spending alerts if limits are exceeded.

- Screenshots:



Transaction and select categories



Add expense and income screen

- Implementation details:

+ List the widgets used for this feature/page:
  - TransactionsScreen (Main List):
    - TransactionSummaryCard: A dashboard widget at the top displaying Total Income, Total Expense, and Current Balance using gradient backgrounds for visual distinction.
    - OrientationBuilder: Adapts the layout dynamically:
      - Portrait: Vertical list with summary on top.
      - Landscape: Split view with Summary on the left (35%) and List on the right (65%).
  - TransactionItem (List Item):
    - Container: Styled with shadows and rounded corners to create a card effect.

        - Gradient Icon Background: The category icon uses a dynamic LinearGradient derived from the category's color string.
        - ListTile: Displays the Category Name, Description (optional), and Amount.
    - AddTransactionScreen/EditTransactionScreen:
        - Scaffold, AppBar: Standard screen structure with a title and a delete button (in edit mode).
        - Form: Manages the validation state of the entire form.
        - SingleChildScrollView: Allows the form to scroll when the keyboard appears.
        - TransactionForm: A separate custom widget containing all input logic.
    - TransactionForm (Input Form):
        - _buildTypeSelector: Custom widget using Container and InkWell to create a stylish Income/Expense switcher with gradient effects.
        - _buildAmountInputCard: Amount input card with large font TextFormField, supporting currency formatting.
        - QuickAmountSelector: Widget suggesting quick amounts (e.g., 50k, 100k) for rapid entry.
        - DropdownButtonFormField: Category selection widget with corresponding icons and colors.
        - TextFormField (Description): Multi-line note input field (maxLines: 3).
        - Material & InkWell (Action Button): "Update/Save" confirmation button with shadow and gradient effects.

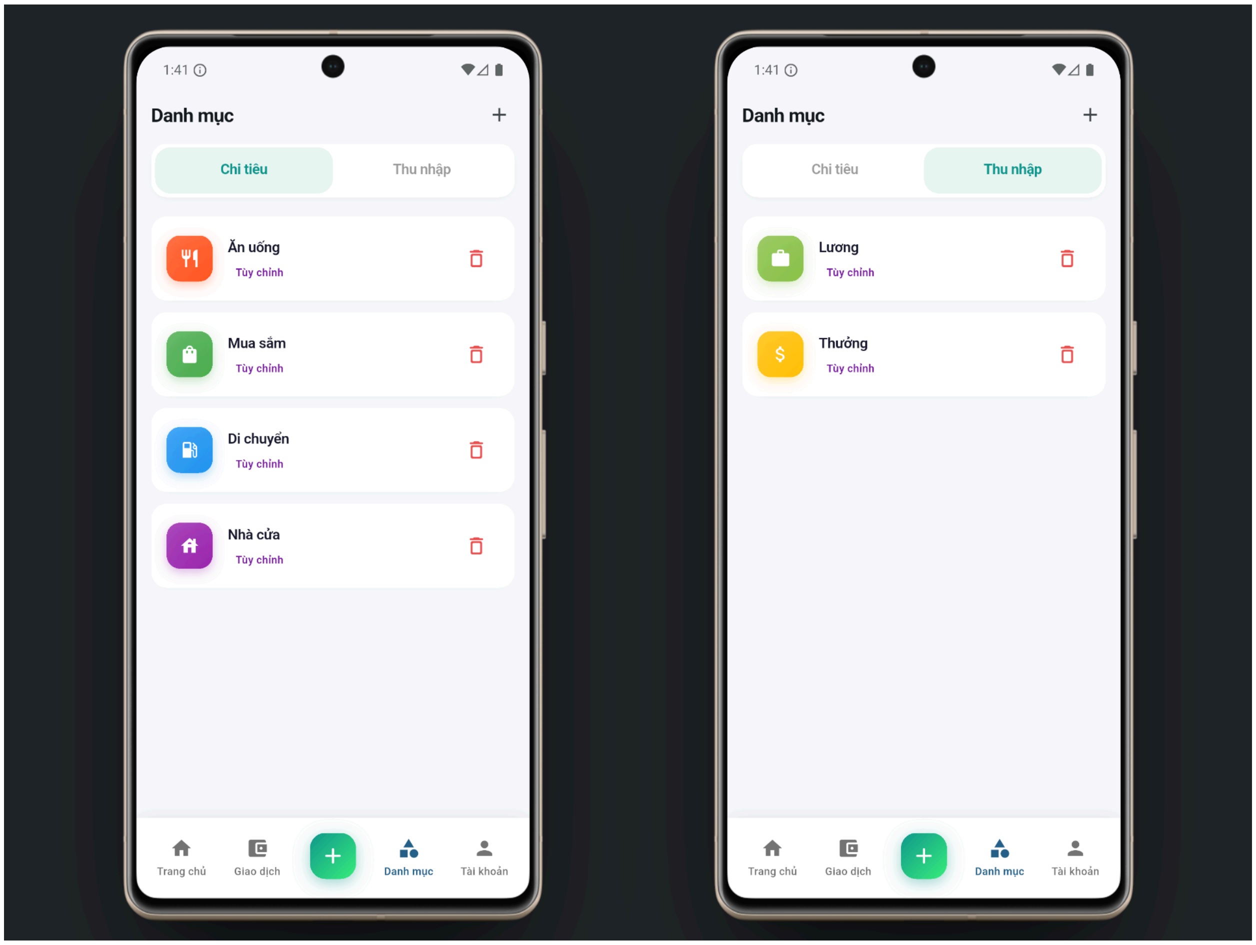
+ Libraries or plugins used:
    - sqflite: Used for storing transaction data locally on the user's device (Local Database).
    - provider: Manages TransactionsManager state to add/edit/delete and update the UI instantly.
    - intl: Formats currency display (e.g., 100.000₫) and dates.
    - flutter_local_notifications (via NotificationService): Sends alerts when spending exceeds defined thresholds after adding/editing transactions.

+ Shared state management solution:
    - Uses Provider with TransactionsManager.
    - TransactionsManager acts as the intermediary between the UI and the Database.
    - Reactive UI Updates: When a user performs an action (Add/Edit/Delete), the Manager calls TransactionService to update the SQLite Database, updates the in-memory _transactions list, and calls notifyListeners() to immediately refresh the UI.
+ Data reading or storage:
    - Local (SQLite):
        - Uses the sqflite library to interact with the SQLite database file.
        - transactions table: Stores columns id, amount, description, date, categoryId, type (income/expense), note.
        - Spending Alert Logic: After every Add/Edit/Delete operation, TransactionService automatically recalculates the monthly spending percentage relative to income. If it exceeds a threshold (e.g., 90%), the system triggers a local notification.

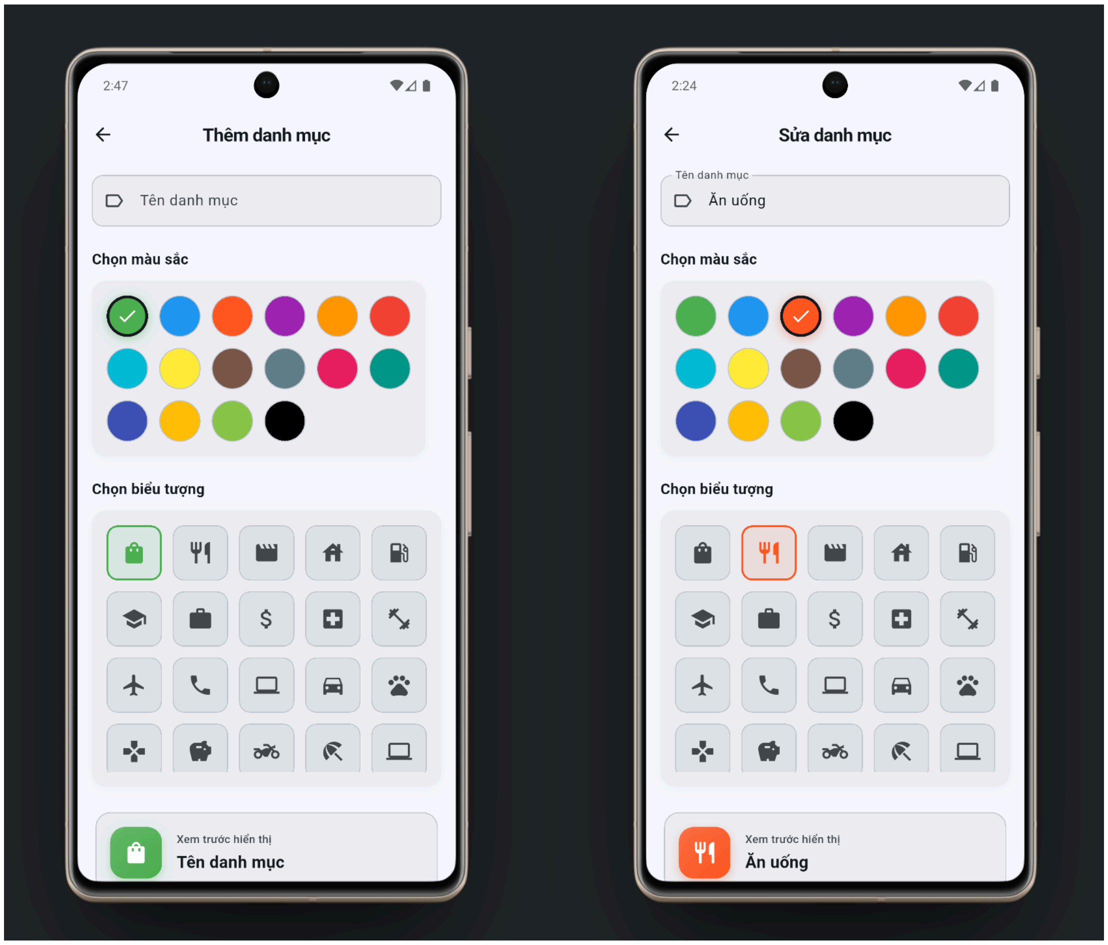
4. Categories

- Description: This feature allows users to create, edit, and delete custom income and expense categories. Categories help classify transactions, making financial management more organized and visual. The system clearly distinguishes between Income and Expense categories and provides a diverse set of icons and color palettes for user personalization.

- Screenshots:



Categories management screen



Add and update categories screen

- Implementation details:

+ List the widgets used for this feature/page:

- CategoriesScreen (Main Screen):
    - TabBar, TabBarView: Creates two separate tabs for "Expense" and "Income," allowing quick switching between lists.
    - ListView.builder: Efficiently renders the list of category items.
    - CategoryItem: Displays individual category items with icon, name, and edit/delete buttons.
    - FloatingActionButton (or AppBar icon): Button to add a new category.
- EditCategoryScreen and CategoryForm (Add/Edit Screen):
    - Form, TextFormField: Input field for category name.

        - Icon Picker: Uses GridView.builder to display a grid of available icons for selection (e.g., dining, shopping, housing...).
        - Color Picker: Displays a list of color circles (circular Container) for users to assign a specific color to the category.

+ Libraries or plugins used:
    - sqflite: Stores the list of categories in the local database.
    - provider: Uses CategoriesManager to synchronize categories across the application (e.g., when a new category is added, it immediately appears in the Transaction screen).

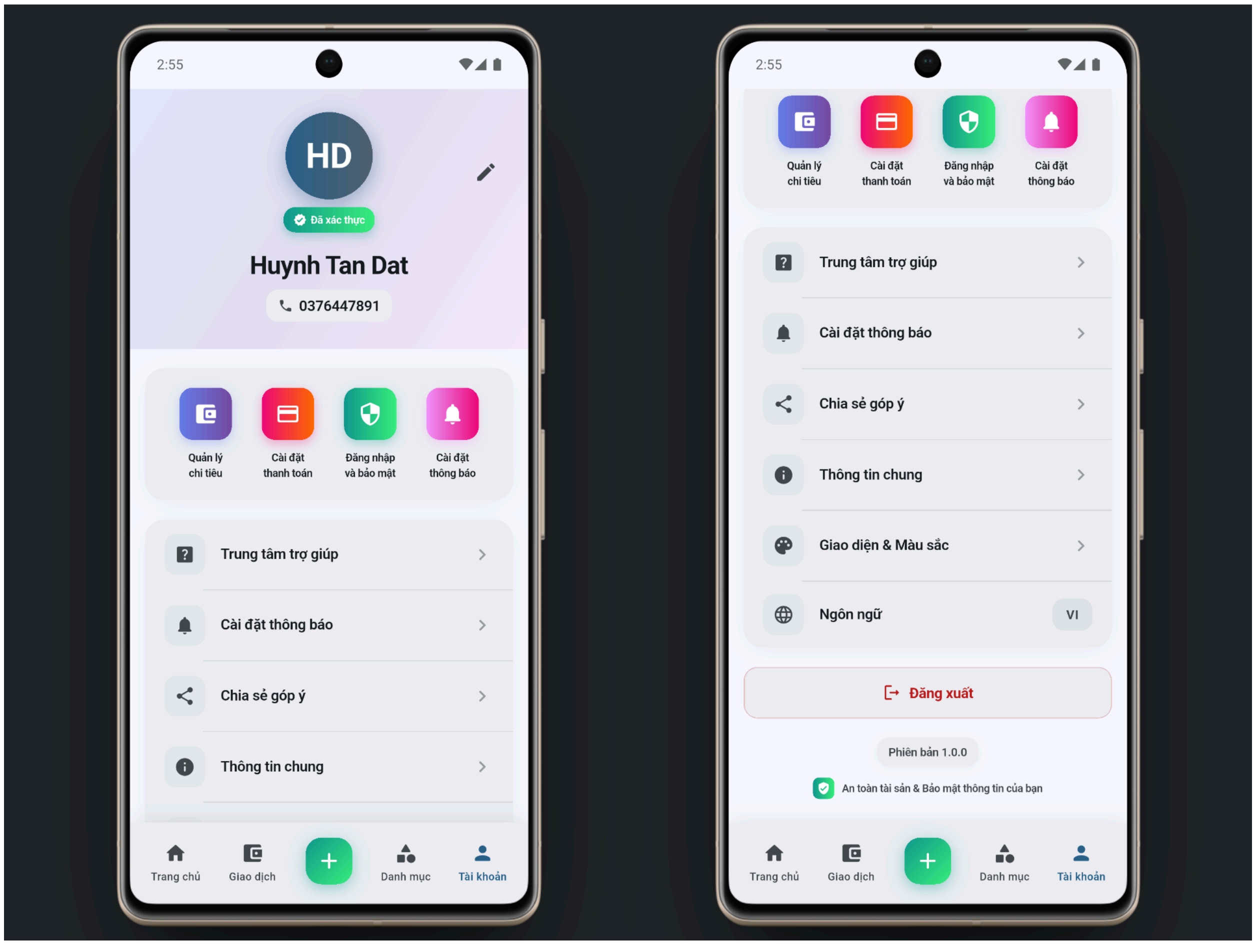
+ Shared state management solution:
    - Uses Provider with CategoriesManager.
    - CategoriesManager maintains two separate lists: _expenseCategories and _incomeCategories.
    - Upon first launch, CategoriesManager checks and automatically initializes a set of default categories (Dining, Transport, Salary...) if the Database is empty.

+ Data reading or storage:
    - Local (SQLite):
        - categories table: Stores id, name, icon (string name, e.g., 'restaurant'), color (hex code, e.g., '#FF5722'), type, is_default.
        - CategoryService handles CRUD operations (Create, Read, Update, Delete) with the Database.

5. Account / User Information

- Description: This feature allows users to view and manage their personal profile information, including name, email, phone number, address, date of birth, and gender. Users can edit their details, configure notification settings, and log out of their account. The interface is designed with a modern look, featuring gradient effects and shadows for a premium user experience.

- Screenshots:



Account screen

- Implementation details:

+ List the widgets used for this feature/page:

- AccountScreen (Main Profile Page):
  - SafeArea, SingleChildScrollView: Basic scrolling structure.
  - Custom Header: Uses Container with a multi-color LinearGradient (Purple, Deep Purple, Pink) for an attractive background.
  - Container (Avatar): Displays the user's initials with a circular gradient background and shadow.
  - IconButton (Edit): Quick edit button.
  - _buildActionButtons: Row containing quick action buttons (Spending Mgmt, Security...) with distinct gradients for each button.

        - Settings List: _buildSettingsSection uses a custom ListTile implementation with:
            - A semi-transparent colored container for the leading icon.
            - A custom trailing arrow icon.
            - Items include: Notification Settings, Theme Settings (ThemeSettingsDialog), and Language.
        - ElevatedButton: Logout button with a distinct style (red border, red text).
    - EditProfileScreen (Edit Page):
        - Stack (Avatar section): Allows placing a camera icon (Positioned) over the avatar to indicate the change photo function.
        - TextFormField: Input fields for Full Name, Email (read-only), Phone, Address.
        - DropdownButtonFormField: Dropdown menu for Gender selection.
        - showDatePicker: Standard Flutter date picker dialog.

+ Libraries or plugins used:
    - provider: Uses AccountManager to retrieve current user info and update the UI after editing.
    - go_router: Handles navigation and logout logic (redirecting to Login).
    - pocketbase: Calls the API to update user info (users collection) on the Backend.

+ Shared state management solution:
    - Uses Provider with AccountManager.
    - AccountManager calls AuthService to retrieve the latest user info from the server or local store (_authService.getUserFromStore()).
    - When a user updates their profile, AccountManager sends a request to the server, then updates the local _account variable and calls notifyListeners() to immediately refresh the AccountScreen.

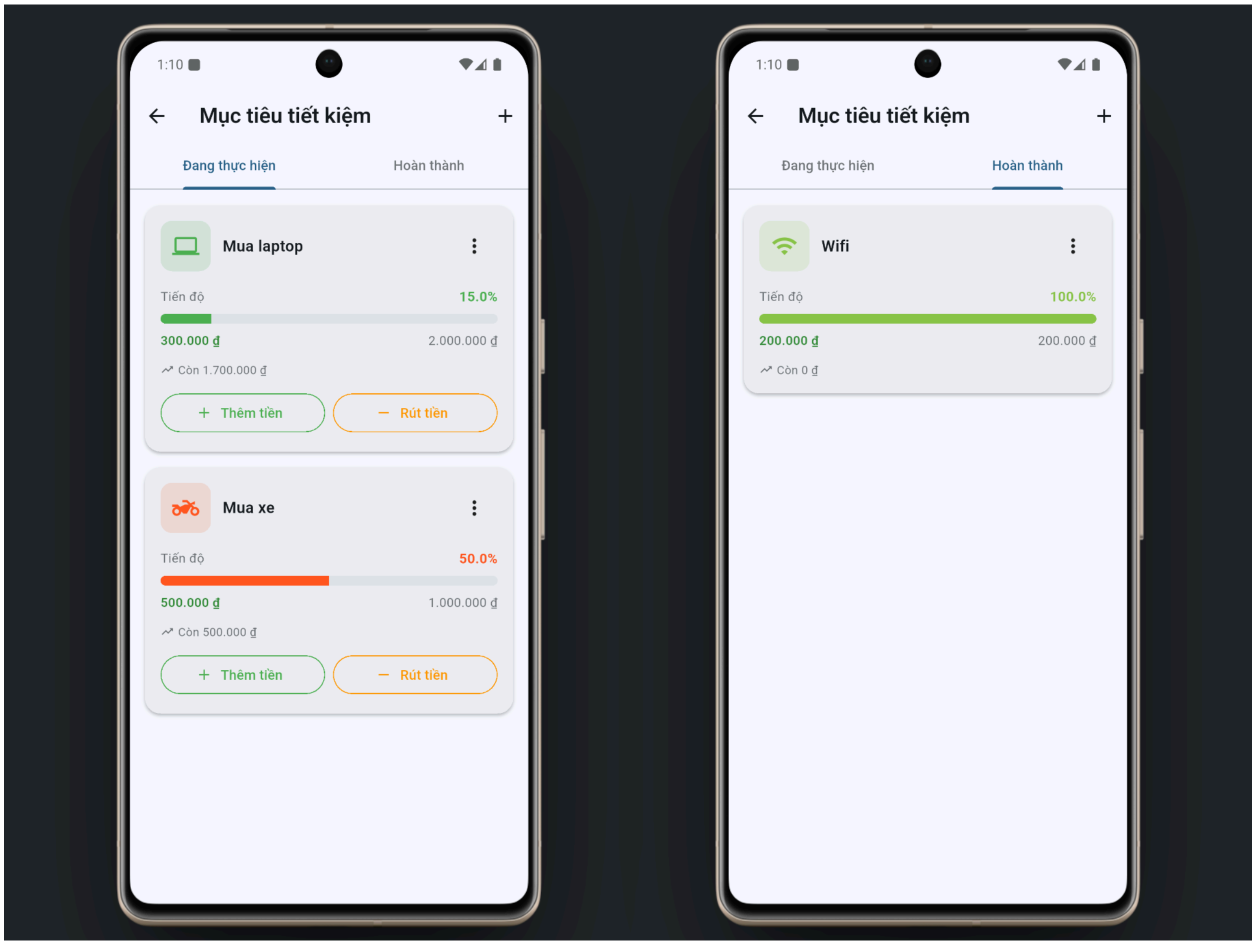
+ Data reading or storage:
    - Remote (PocketBase):

    - Interacts with the users collection.
    - API Update: _authService.updateProfile(id, body) sends a PATCH request to the server with fields: name, email, phone, address, gender, dateOfBirth.
- Local:
    - Initial display data is retrieved from AuthStore (cached upon login). When online, the app automatically refreshes data from the server to ensure it is up-to-date.

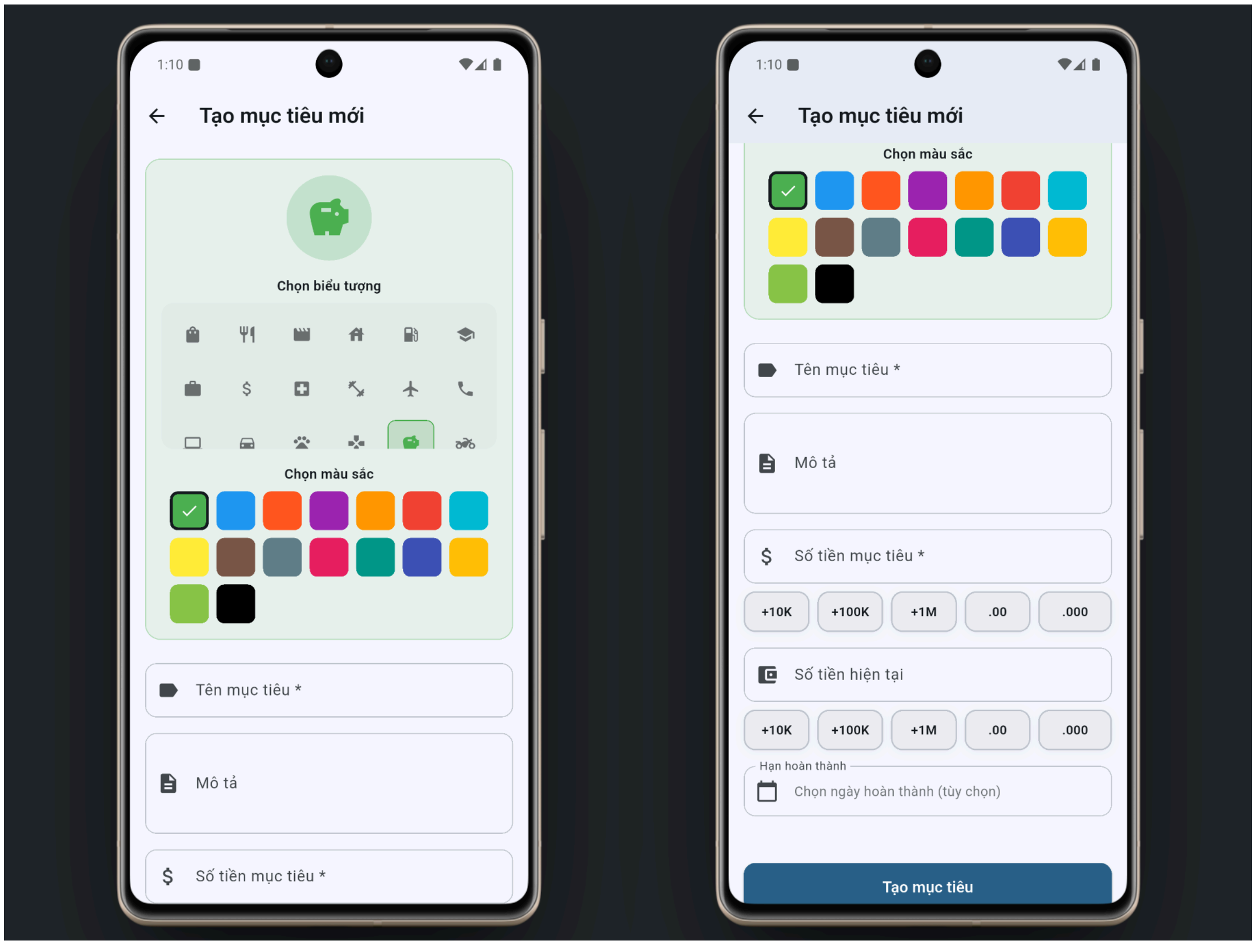
6. Saving Goal

- Description: This feature helps users set and track financial goals ("Buy a motorbike," "Travel"). It provides a visual motivation system with progress bars, deadline tracking, and celebratory notifications when a goal is reached. Users can easily add funds to or withdraw funds from specific goals directly from the interface. A summary block on the Home Screen keeps users constantly aware of their overall progress.

- Screenshots:



Saving Goal screen



Add Saving Goal screen

- Implementation details:

+ List the widgets used for this feature/page:
    - SavingsGoalsBlock (Home Dashboard):
        - Container & Decoration: A styled card on the Home screen displaying aggregate statistics.
        - LinearProgressIndicator: Visualizes the Overall Progress of all active goals combined.
        - Goal Summary: Displays totalSaved vs. totalTarget and a compact list of the top 3 active goals.
    - SavingsGoalsScreen (Main List):
        - TabBar: Separates goals into "Active" (Đang thực hiện) and "Completed" (Hoàn thành) tabs.
        - SavingsGoalCard: A versatile card widget with two modes:
        - Summary Mode: Compact view for the Home screen.

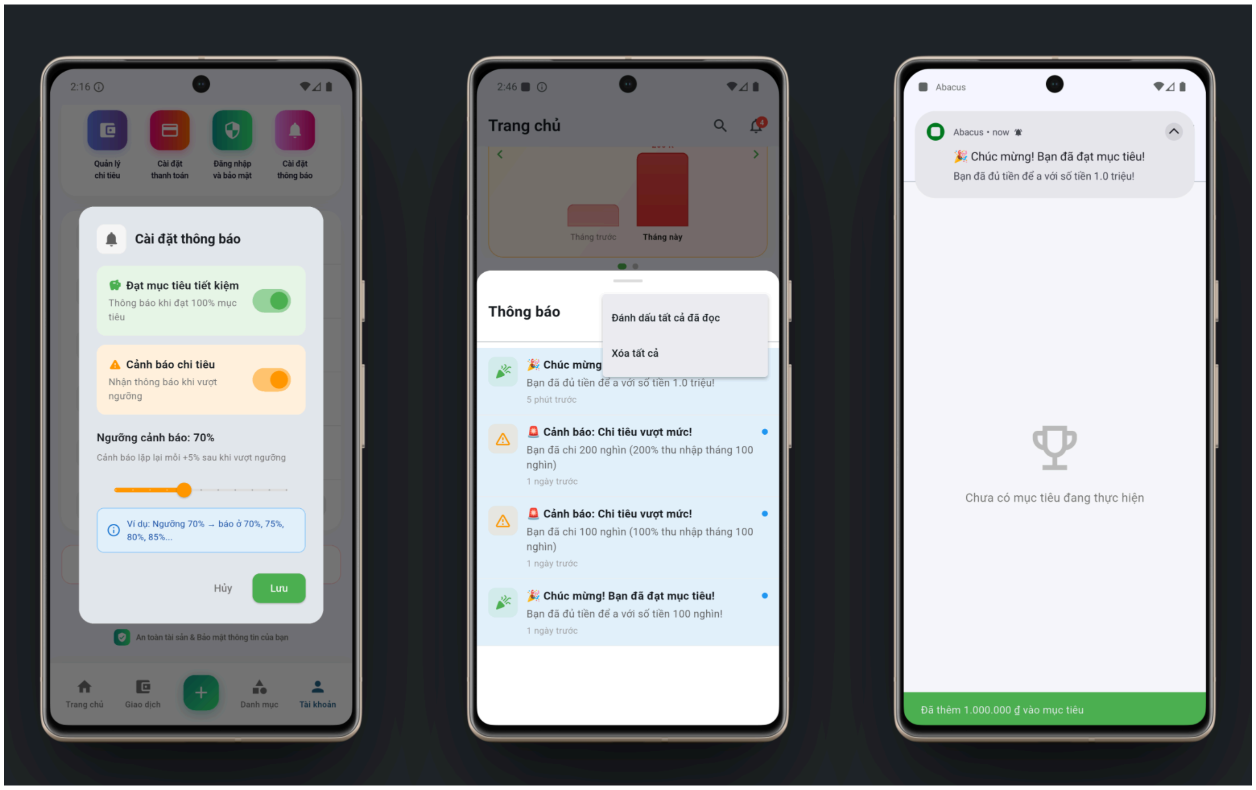
    - Detail Mode: Expanded view with "Add Money" (+) and "Withdraw" (-) buttons, progress percentage, and remaining amount.
  - AddEditGoalScreen (Input Form):
    - Visual Preview: A live-updating card showing how the goal will look with the selected Icon and Color.
    - QuickAmountSelector: A custom helper widget providing shortcut buttons to speed up number entry.
    - Input Fields: Name, Target Amount, Current Amount, and optional Target Date/Description.
  - GoalActionDialogs: Update Progress Dialog: A modal allowing users to quickly add or subtract funds from a specific goal without entering the full edit screen.
- Libraries or plugins used:
  - sqflite: Stores savings goals persistently in the local database.
  - provider: Uses SavingsGoalsManager to calculate totals and update UI instantly.
  - intl: Formats currency and dates.
  - flutter_local_notifications: Sends a celebratory notification when a goal reaches 100% completion.
- Shared state management solution:
  - Uses Provider with SavingsGoalsManager.
  - Computed Logic: The manager efficiently calculates derived state such as overallProgress and activeGoalsCount to drive the dashboard UI.
  - Reactive Updates: Operations like updateProgress immediately update the database and local state, triggering a UI refresh across the Home Screen and Goals Screen simultaneously.
- Data reading or storage:
  - Local (SQLite):
    - Table: savings_goals.
    - Columns: id, name, description, target_amount, current_amount, target_date, icon, color, created_at, updated_at.

- Notification Logic: The SavingsGoalService compares the goal's status before and after an update. If the goal transitions from incomplete to completed, it triggers a system notification to congratulate the user.

7. Notifications & Alerts

- **Description**: This feature provides a comprehensive alert and notification system for users. The application automatically monitors spending against monthly income and sends alerts when safety thresholds are exceeded. Users can customize the alert threshold (e.g., 70% of income) and toggle notifications for specific categories (Spending, Savings Goals). The notification management interface allows users to review history, mark as read, or delete notifications.

- Screenshots:



Cài đặt thông báo

Đạt mục tiêu tiết kiệm
Thông báo khi đạt 100% mục tiêu

Cảnh báo chi tiêu
Nhận thông báo khi vượt ngưỡng

Ngưỡng cảnh báo: 70%
Cảnh báo lặp lại mỗi +5% sau khi vượt ngưỡng

Ví dụ: Ngưỡng 70% → báo ở 70%, 75%, 80%, 85%...

Hủy Lưu

Thông báo

Đánh dấu tất cả đã đọc
Xóa tất cả

🎉 Chúc mừng
Bạn đã đủ tiền để a với số tiền 1.0 triệu!
5 phút trước

🚨 Cảnh báo: Chi tiêu vượt mức!
Bạn đã chi 200 nghìn (200% thu nhập tháng 100 nghìn)
1 ngày trước

🚨 Cảnh báo: Chi tiêu vượt mức!
Bạn đã chi 100 nghìn (100% thu nhập tháng 100 nghìn)
1 ngày trước

🎉 Chúc mừng! Bạn đã đạt mục tiêu!
Bạn đã đủ tiền để a với số tiền 100 nghìn!
1 ngày trước

Abacus • now
🎉 Chúc mừng! Bạn đã đạt mục tiêu!
Bạn đã đủ tiền để a với số tiền 1.0 triệu!

Chưa có mục tiêu đang thực hiện

Đã thêm 1.000.000 ₫ vào mục tiêu

Notifications

- Implementation details:

+ List the widgets used for this feature/page:
  - NotificationSettingsDialog (Settings Dialog):

        - Slider: Allows users to drag and select the alert threshold from 50% to 100%.
        - Switch: Toggles for enabling/disabling Spending Alerts and Savings Goal notifications.
        - AlertDialog: The container for the settings UI.
    - NotificationsBottomSheet (Notification List):
        - showModalBottomSheet: Displays the notification list sliding up from the bottom.
        - ListView.builder & Dismissible: Displays the list and allows swiping to delete individual notifications.
        - PopupMenuButton: Options menu to "Mark all as read" or "Clear all".
        - Empty State Widget: Displays an icon and message when the list is empty.
    - Local Notification UI (System Notification):
        - Uses FlutterLocalNotificationsPlugin to display notifications on the device's status bar (System Tray).
- Libraries or plugins used:
    - flutter_local_notifications: The primary plugin for handling local notifications on Android/iOS devices.
    - shared_preferences: Stores user settings (alert thresholds, on/off state) and tracks notified milestones for the month to prevent spamming.
    - provider: Manages the notification list state (NotificationsManager) to update the unread badge count across the app.
- Shared state management solution:
    - NotificationsManager is responsible for managing the in-memory _notifications list.
    - When NotificationService generates a new notification (e.g., Spending Alert), it triggers the onNotificationCreated callback, which causes NotificationsManager to add the notification to the list and update the UI.
- Data reading or storage:
    - Settings Storage: Alert thresholds (spending_threshold) and toggle states are stored in SharedPreferences.

- Notification History: Notification history is stored locally (via NotificationStorage - JSON file or SharedPreferences) so users can review them after dismissing the system notification.
- Alert Logic:
    - The system automatically checks after every transaction.
    - Example: If the user sets a 70% threshold, the system alerts once when 70% is exceeded. It then continues to alert at subsequent +5% increments (75%, 80%...) to provide continuous reminders if spending continues to rise.