CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING
CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Subject : Mobile Application Development **Semester:** 5

Subject Code: AIML308 **Academic Year :**2025-26(ODD)

NAME : CHOKSI OM CHIRAGBHAI **ID: 23AIML010**

Practical 1

Problem Definition

You are building a mobile application where users navigate through multiple screens like login, dashboard, and profile. Create a basic multi-screen Flutter app with navigation, passing data between pages.

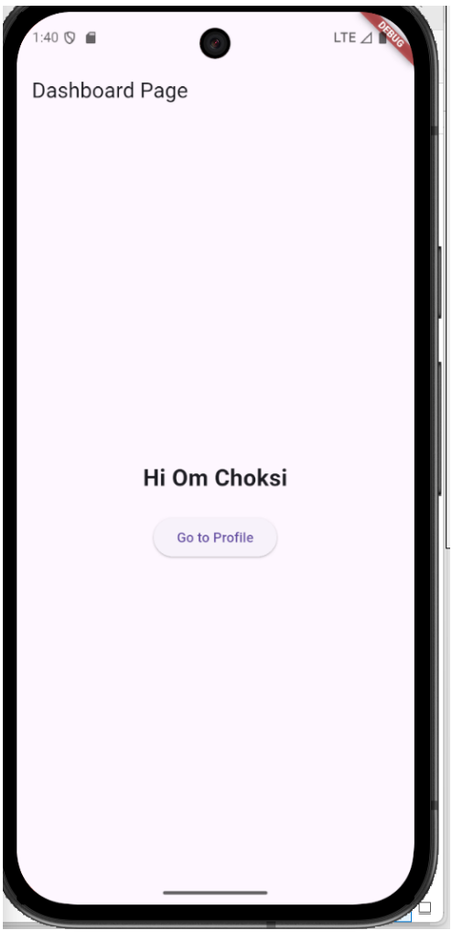
Technical Approach:

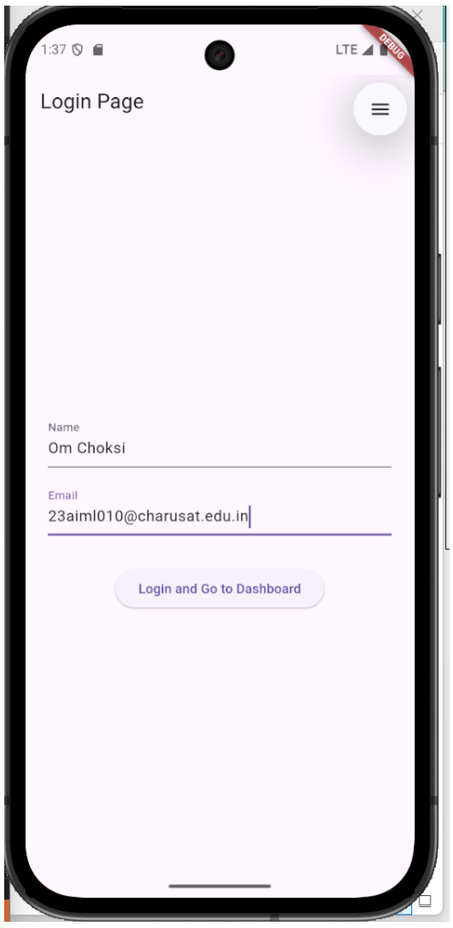
- State Management: Uses StatefulWidget with setState() for UI updates in login and dashboard screens.
- Navigation: Named routes in MaterialApp for screen transitions (pushReplacementNamed for login→dashboard, pushNamed for dashboard→profile).
- Data Passing: Route arguments via ModalRoute.of(context) for passing user data between screens.
- Core Widgets: Scaffold, TextField, ElevatedButton, Text; no external packages used.

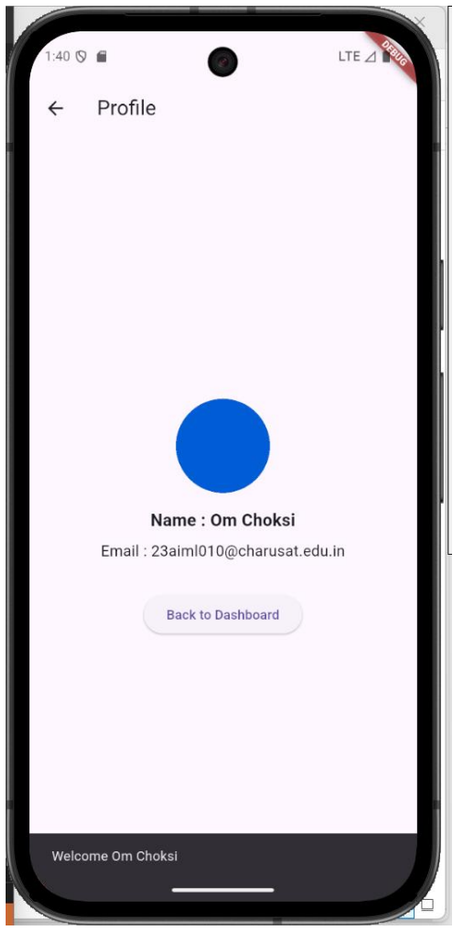
File Structure:

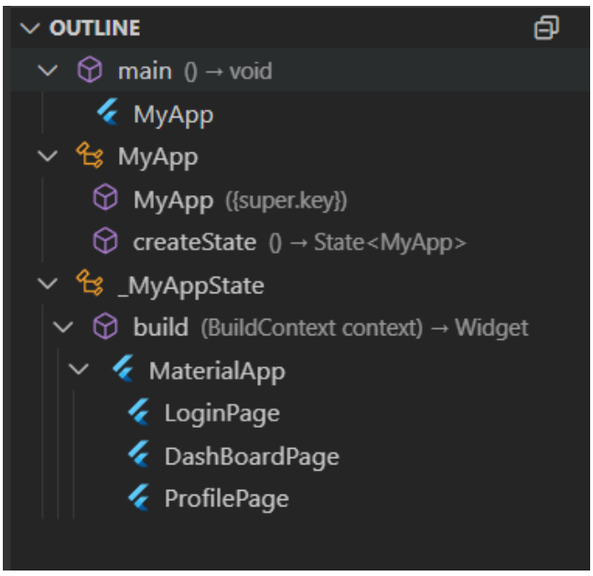
```
lib/
├─── main.dart        # Application entry point with MaterialApp and route configuration
├─── login.dart       # Login screen with form inputs and navigation to dashboard
├─── dashboard.dart   # Dashboard screen displaying user data and navigation to profile
└─── profile.dart     # Profile screen showing user details with back navigation
```

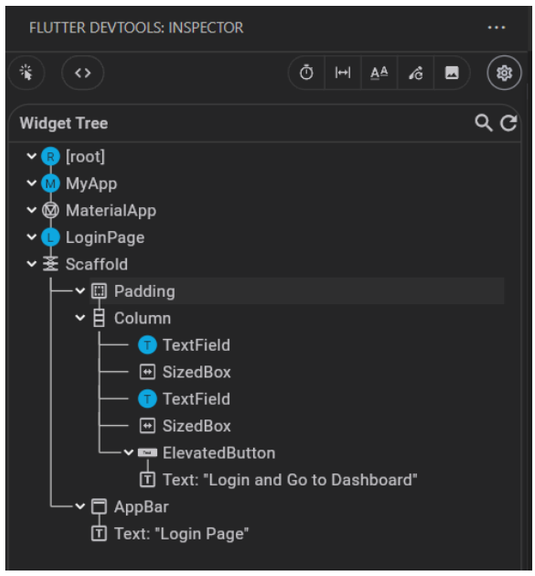
Screenshots:











Key Questions:

1. How does navigation work in Flutter?

Ans: Navigation managed by Navigator widget using a stack of Route objects.In this project: Named routes defined in MaterialApp (main.dart). pushReplacementNamed (login → dashboard): Replaces current screen, passes arguments. pushNamed (dashboard → profile): Adds to stack, allows return via pop().

2. How can data be passed between screens?

Ans: - Data passed via route arguments in RouteSettings.
Sender: Include Map in `arguments` parameter of navigation methods.
Receiver: Access via ModalRoute.of(context).settings.arguments.
Example: Login passes 'name' and 'email' to dashboard and profile.

3. What is the difference between push and pushReplacement?

Ans: push(): Adds new screen to navigation stack; allows back navigation.
pushReplacement(): Replaces current screen; removes from history.
Usage: pushReplacementNamed for login (no return), pushNamed for profile (allow back).

Key Skills to be Understand:

Widget Tree, Navigation routing

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING
CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Subject : Mobile Application Development **Semester:** 5

Subject Code: AIML308 **Academic Year :**2025-26(ODD)

NAME : CHOKSI OM CHIRAGBHAI **ID: 23AIML010**

Practical 2

Problem Definition

You are building a mobile application where users navigate through multiple screens like login, dashboard, and profile. Develop a temperature converter app using Dart functions and input widgets.

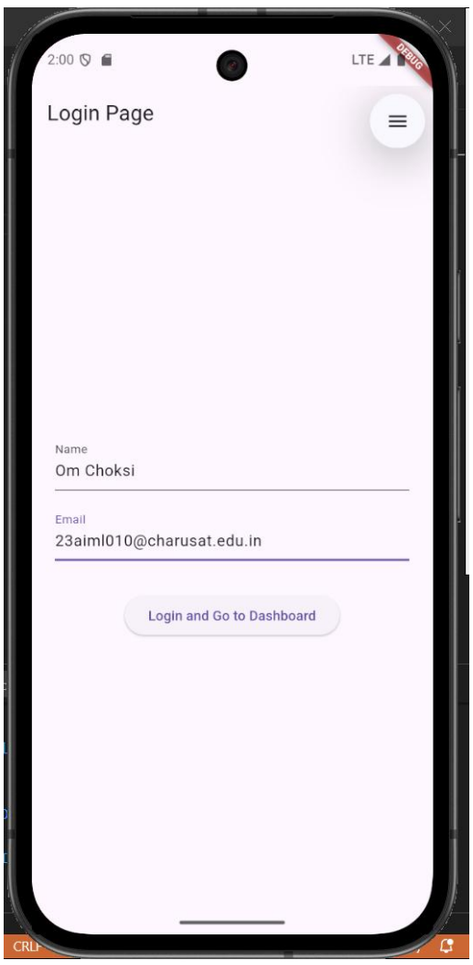
Technical Approach:

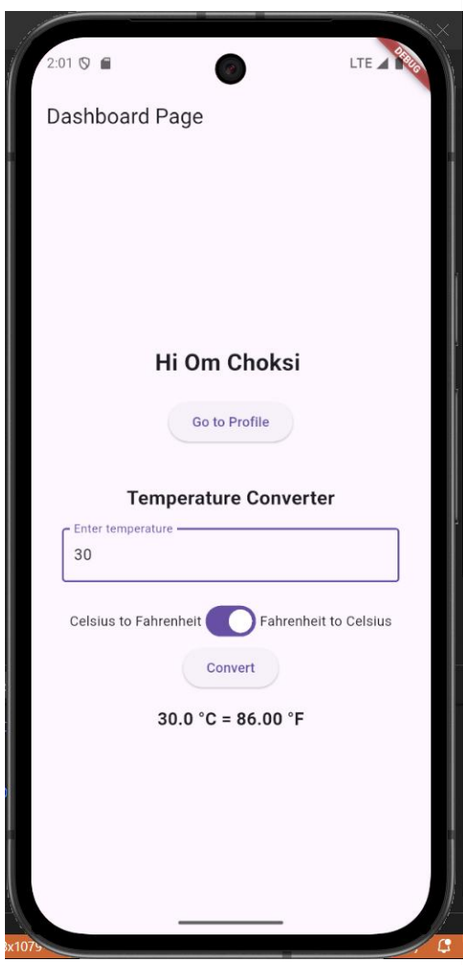
This Flutter app is a multiscreen application with login, dashboard (temperature converter), and profile screens. It uses setState for UI updates, TextField for user input with double.tryParse validation, and Navigator for routing. Core widgets include StatefulWidget, TextField, ElevatedButton, Switch, and Scaffold.

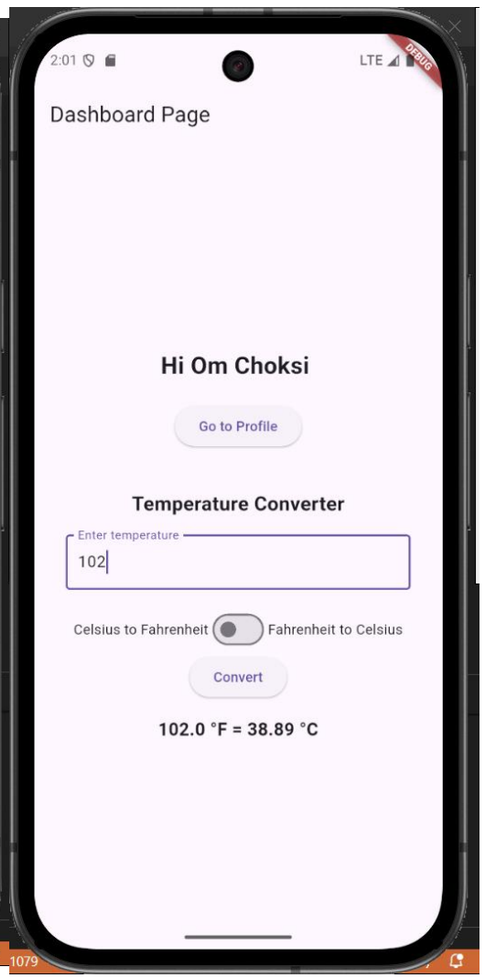
File Structure:

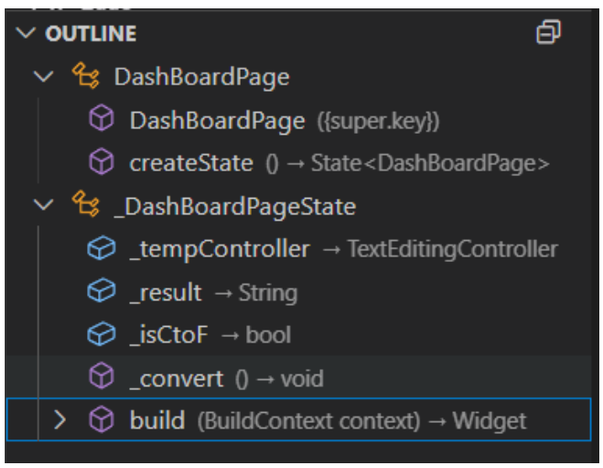
```
lib/
├── main.dart          # Main entry point of the Flutter app, sets up routes and app structure
├── login.dart         # Login screen with TextField inputs for name and email, handles navigation to dashboard
├── dashboard.dart     # Dashboard screen temperature converter with Switch for mode toggle and conversion logic
└── profile.dart       # Profile screen that displays username and email retrieved from navigation arguments
```

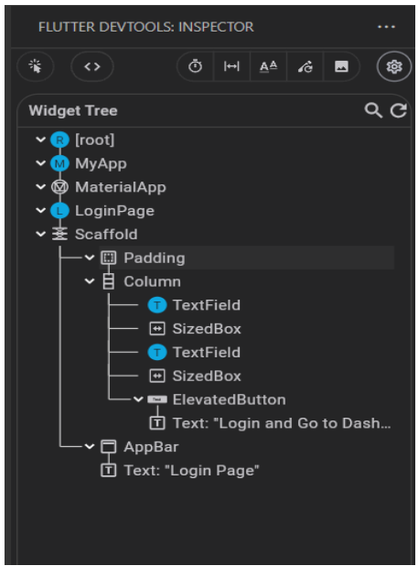
Screenshots:











Key Questions:

1. How to take and validate user input?

Ans: User input is captured using TextField widgets with controllers (e.g., _tempController in dashboard.dart). Validation is performed by parsing inputs with double.tryParse in the _convert function, displaying error messages via setState if invalid, ensuring only numeric values are processed for temperature conversion.

2. What is the role of Dart functions?

Ans: Dart functions encapsulate reusable logic, such as the _convert function in dashboard.dart, which handles input validation, calculation, and UI updates. They enable modular code by separating concerns,

like navigation in the login button's onPressed callback, allowing for clean, maintainable event handling and state management.

3. How to update UI on user action?
Ans: UI updates are triggered using setState in stateful widgets, as seen in the Switch's onChanged callback and the _convert function in dashboard.dart, which rebuilds the widget tree to reflect changes like toggled modes or computed results, providing immediate visual feedback.

Key Skills to be Understand:

Dart Basics, Stateful Widget, Event Handling

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING
CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Subject : Mobile Application Development **Semester:** 5

Subject Code: AIML308 **Academic Year :**2025-26(ODD)

NAME : CHOKSI OM CHIRAGBHAI **ID: 23AIML010**

Practical 3

Problem Definition

You are building a mobile application where users navigate through multiple screens like login, dashboard, and profile. Create a dynamic TODO app using State Management (setState) and ListView.builder.

Supplementary Problems -	Add categories, filter completed tasks

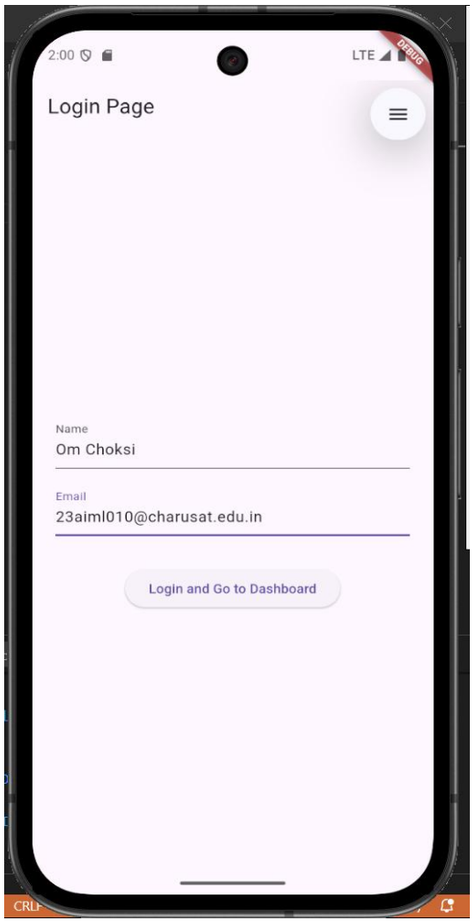
Technical Approach:

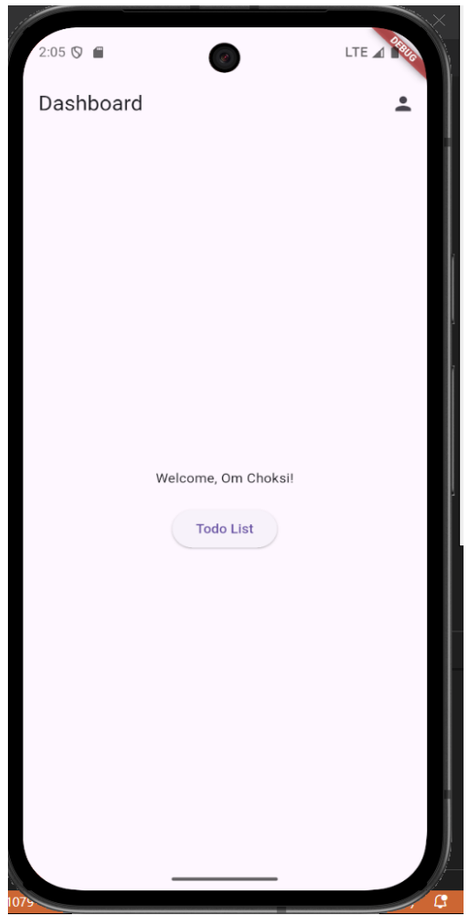
This Flutter app implements a dynamic TODO list using setState for state management within StatefulWidget to handle UI updates efficiently. Core logic relies on Dart functions for task operations like adding, deleting, and filtering, with ListView.builder for rendering dynamic lists. Key widgets include TextField for input, DropdownButton for categories, Checkbox for completion, and Switch for filtering, ensuring responsive and interactive UI without external packages beyond Flutter's material library.

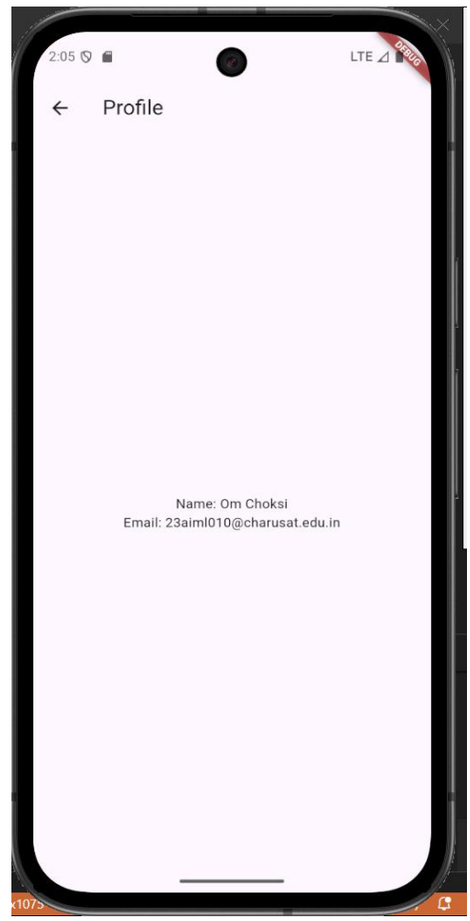
File Structure:

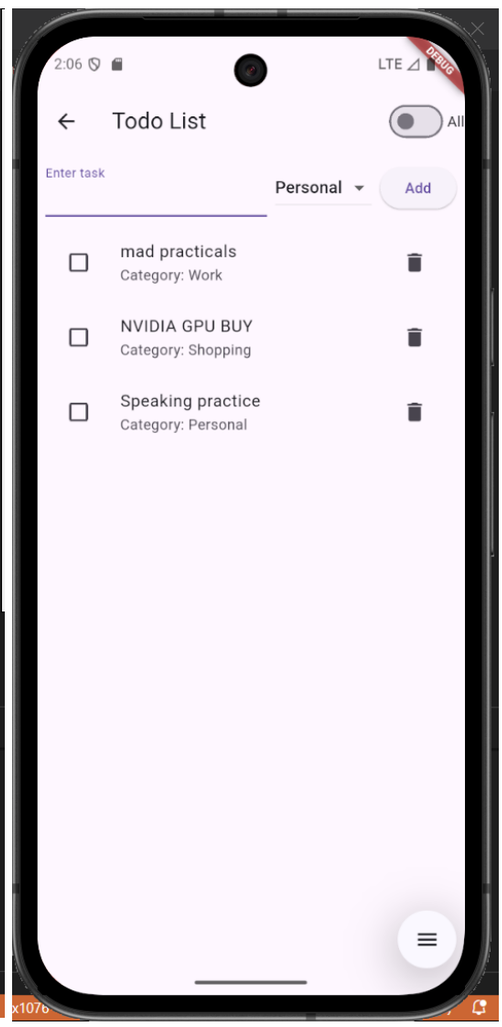
```
lib/
├── main.dart        # App entry point with route configuration
├── login.dart       # Login screen with name and email input
├── dashboard.dart   # Dashboard with user greeting and navigation
├── profile.dart     # Profile screen displaying user details
└── todolist.dart    # TODO list screen with dynamic task management
```

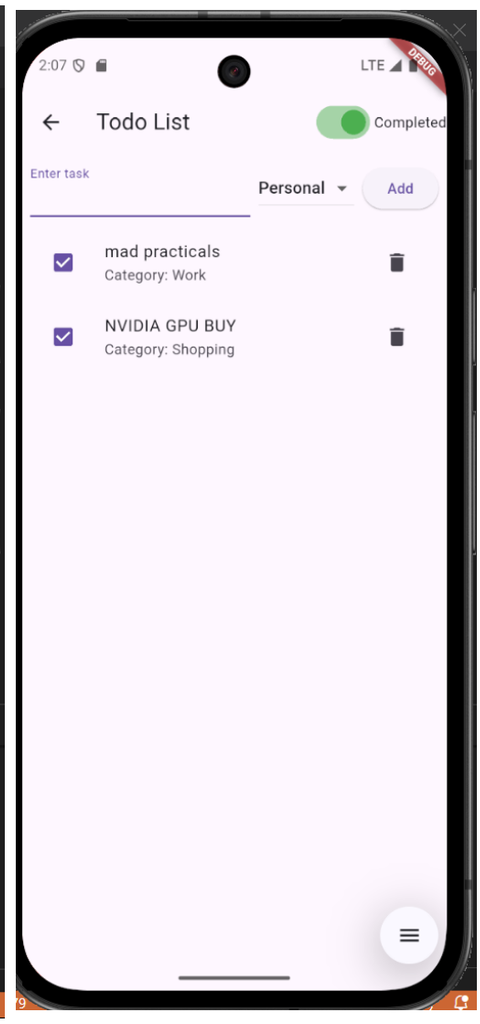
Screenshots:

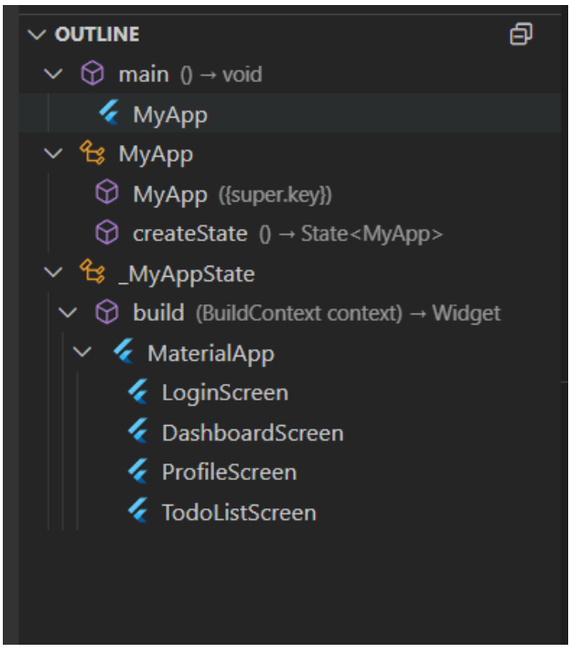


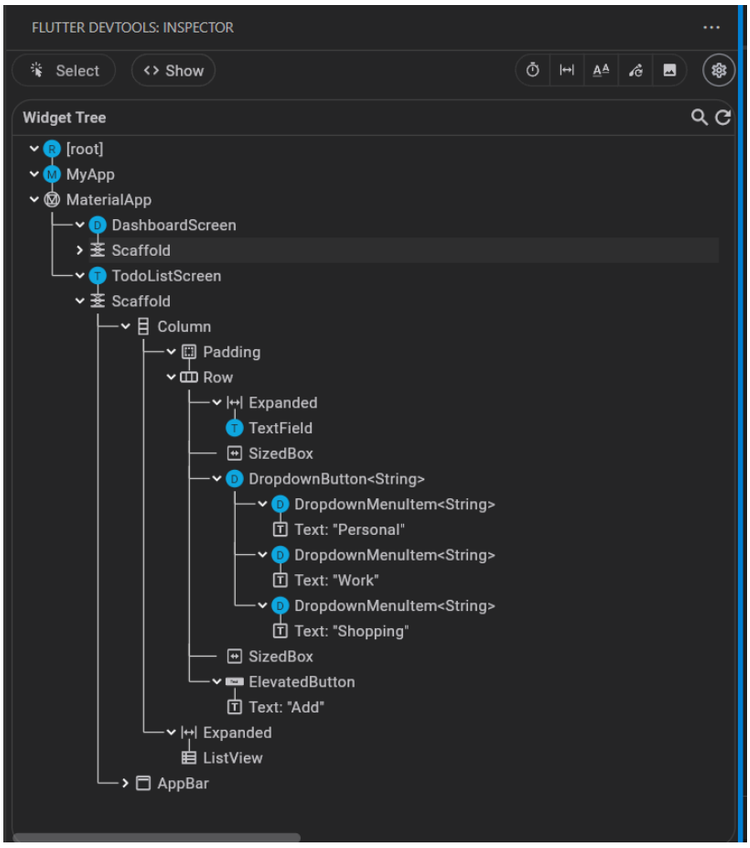












Key Questions:

1. How to take and validate user input?
Ans: Items are added using tasks.add() in _addTask function with setState to update UI; deleted using tasks.removeAt(index) in _deleteTask, triggering rebuild.

2. What is the role of Dart functions?
Ans: setState marks the widget as dirty, notifying Flutter to rebuild the widget tree with updated state, ensuring UI reflects changes like task additions or completions.

3. How to update UI on user action?
Ans: Use ListView.builder for efficient rendering of variable-length lists; update the underlying List<Task> with setState to reflect additions, deletions, or filters dynamically.

Key Skills to be Understand:

setState, ListView, Dynamic UI

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING
CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Subject : Mobile Application Development **Semester:** 5

Subject Code: AIML308 **Academic Year :**2025-26(ODD)

NAME : CHOKSI OM CHIRAGBHAI **ID: 23AIML010**

Practical 4

Problem Definition

You are building a mobile application where users navigate through multiple screens like login, dashboard, and profile. Design a Form-based Registration App with validation using TextFormField.

Supplementary Problems -	Feedback form with rating

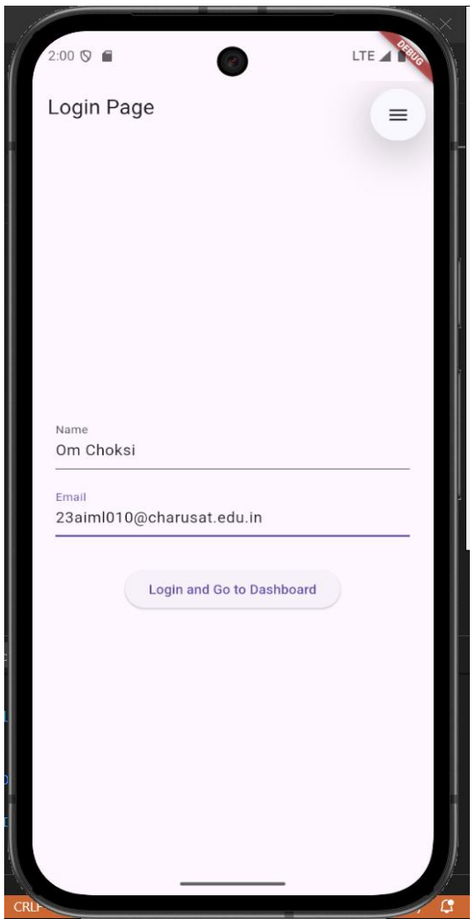
Technical Approach:

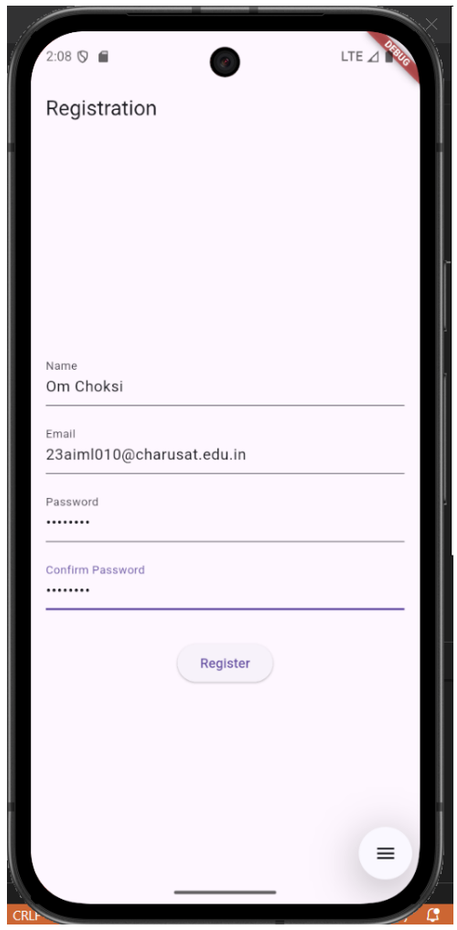
This Flutter application implements a form-based registration system with validation, utilizing TextFormField for input handling and GlobalKey<FormState> for form state management. The core logic employs TextEditingController for accessing input values and custom validator functions to enforce data integrity. Key widgets include Form, TextFormField, Radio, and ElevatedButton, with navigation managed through named routes in MaterialApp.

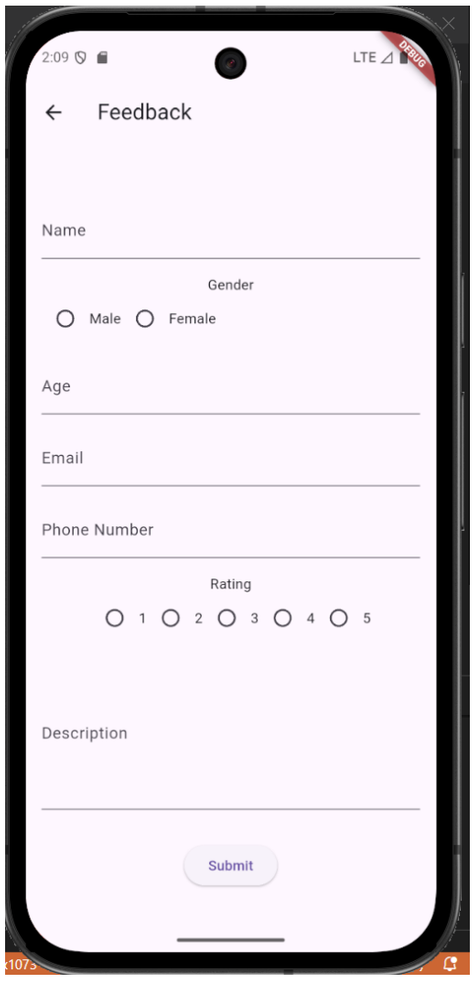
File Structure:

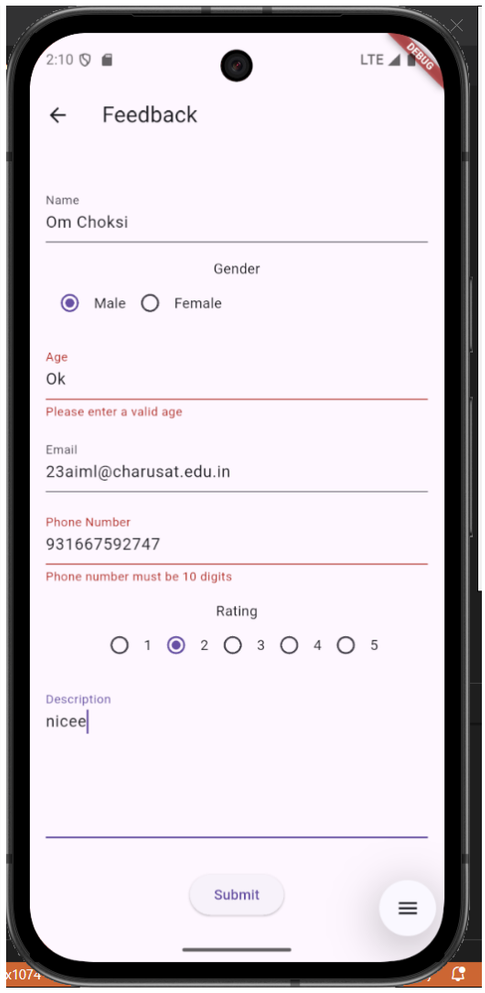
```
lib/
├─── main.dart   #he entry point of the application.
├─── login.dart #Handles user authentication and login logic.
├─── dashboard.dart # Displays the main dashboard after login.
├─── profile.dart #Manages user profile information and settings.
├─── todolist.dart # Implements a to-do list feature for task management.
└─── feedback.dart #Allows users to provide feedback or submit reviews.
```

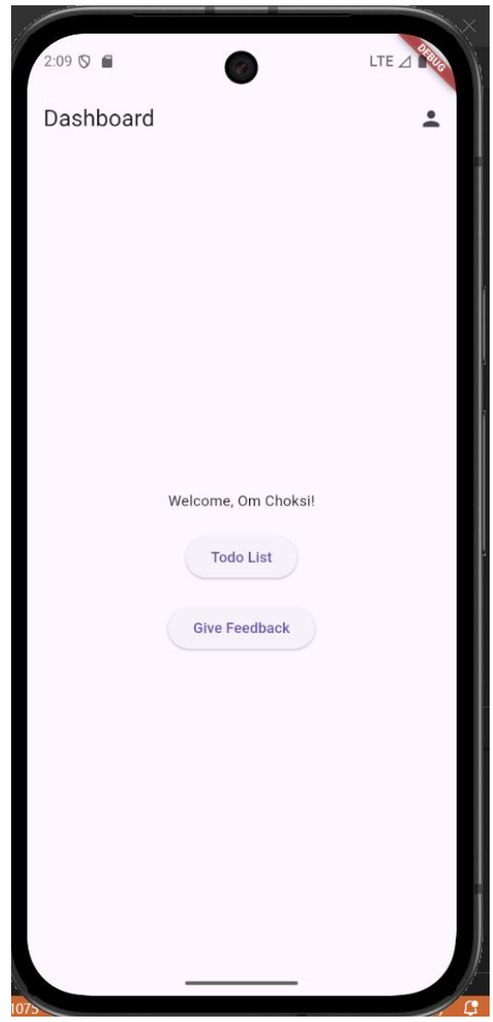
Screenshots:

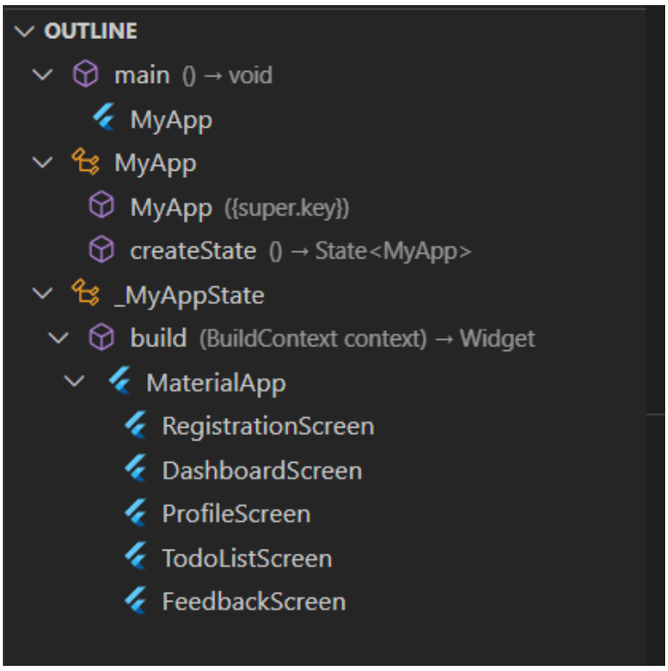


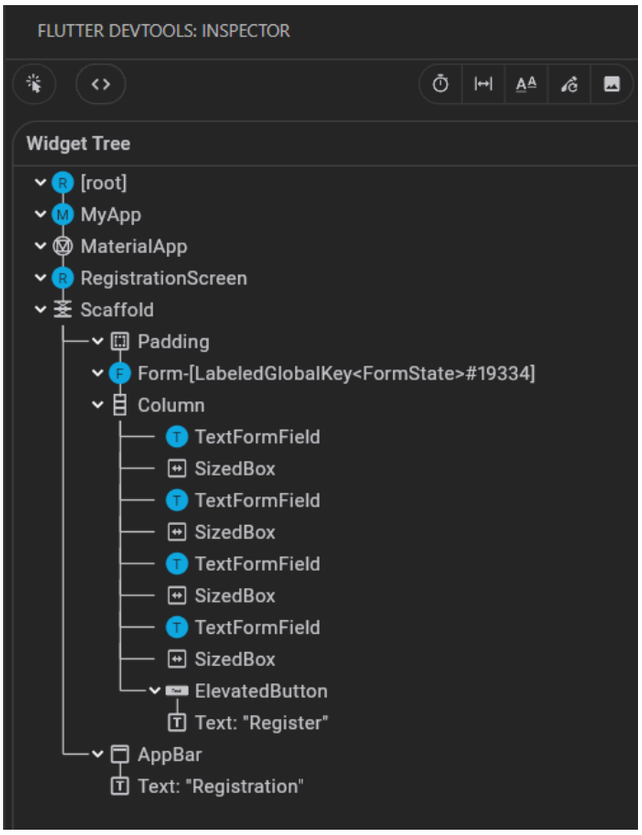












Key Questions:

1. How to take and validate user input?
Ans: Used Form widget with GlobalKey<FormState>, wrap TextFormField in Form, define validator functions that return error strings or null, call _formKey.currentState!.validate() on submit to trigger validation and display errors.

2. What is the role of Dart functions?
Ans: Create TextEditingController instances for each TextFormField to access input values, use GlobalKey<FormState> assigned to Form widget to manage form state and trigger validation programmatically.

3. How to update UI on user action?
Ans: Required field validation (check if empty), email format validation using regex, numeric validation with int.tryParse(), password strength (minimum length), field matching (confirm password), and custom length validation (e.g., phone number digits).

Key Skills to be Understand:

V Form Handling, Input Validation

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING
CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Subject : Mobile Application Development **Semester:** 5

Subject Code: AIML308 **Academic Year :**2025-26(ODD)

NAME : CHOKSI OM CHIRAGBHAI **ID: 23AIML010**

Practical 5

Problem Definition

You are building a mobile application where users navigate through multiple screens like login, dashboard, and profile. Build a Student Records App with CRUD operations using SQLite.

Supplementary Problems -	**Book management** or Expense tracker app

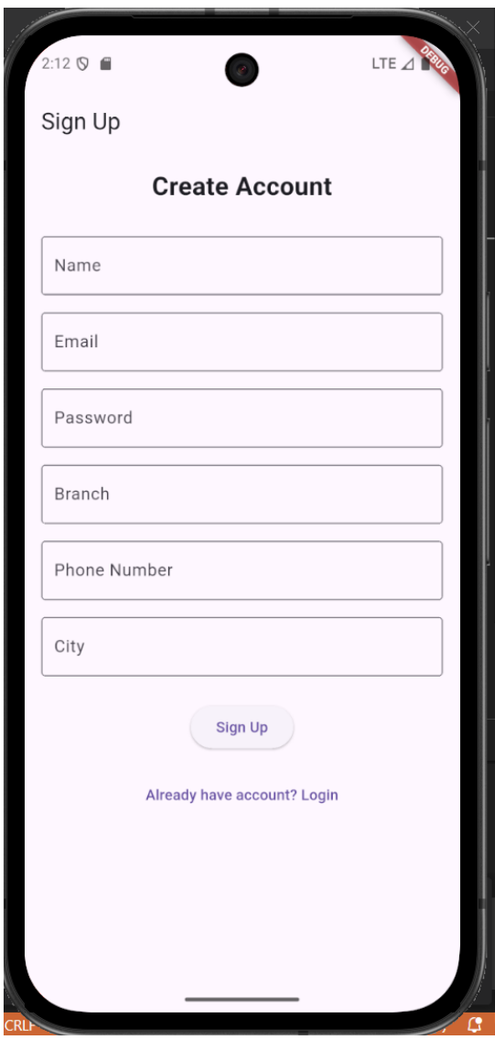
Technical Approach:

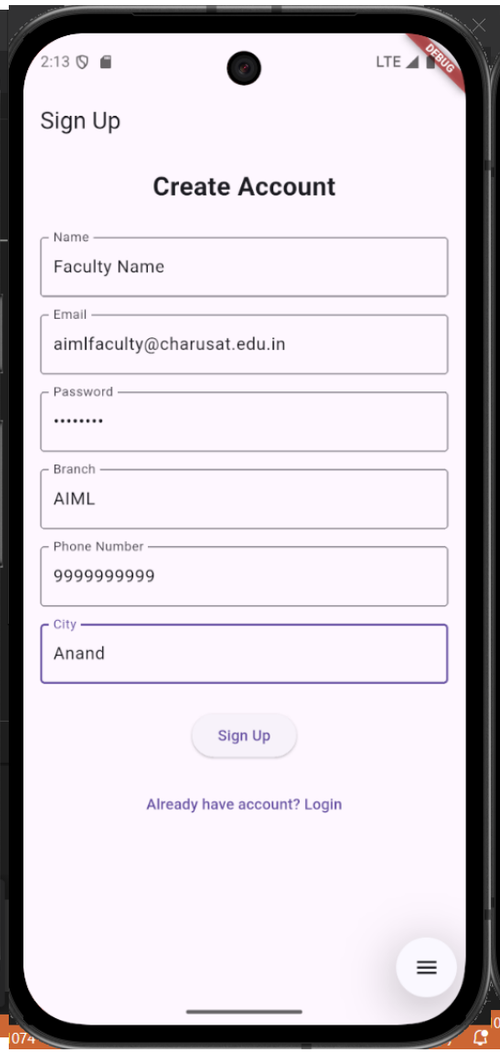
This Student Records App is built using Flutter and demonstrates local data persistence, CRUD operations, and basic authentication. The core logic uses the sqflite package for SQLite database management, shared_preferences for simple user data persistence, and state management via setState. Main widgets include StatefulWidget for dynamic screens, FutureBuilder for async database queries, ListView.builder for displaying student lists, and Form/TextFormField for input validation. The app's navigation is managed with named routes, and all CRUD/database logic is encapsulated in a singleton helper class.

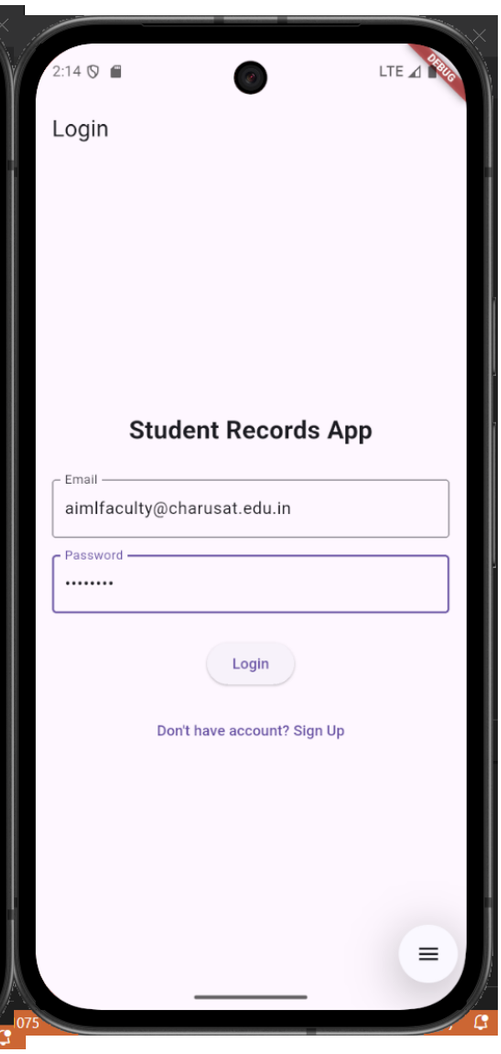
File Structure:

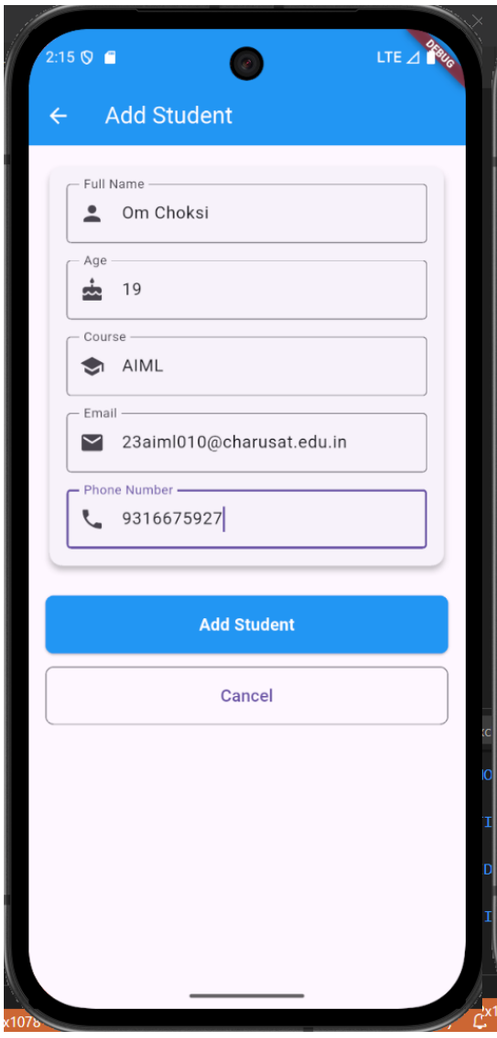
```
lib/
├── main.dart - Entry point for the Flutter app, initializes the widget tree.
├── database/
│   └── database_helper.dart - Singleton class for SQLite CRUD operations on students and users.
├── models/
│   ├── student.dart - Student data model with properties and serialization.
│   └── user_data.dart - User data model for authentication and profile info.
└── screens/
    ├── add_edit_student_screen.dart - Form screen for adding/editing students.
    ├── login_screen.dart - Login screen with authentication.
    ├── profile_screen.dart - User profile display/edit screen.
    ├── signup_screen.dart - User registration screen.
    └── student_list_screen.dart - List screen for viewing/editing/deleting students.
```

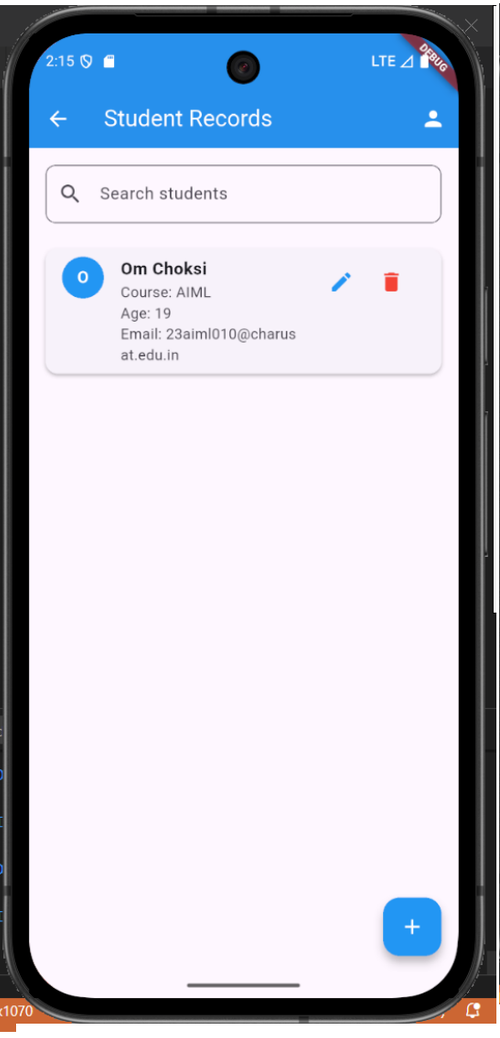
Screenshots:

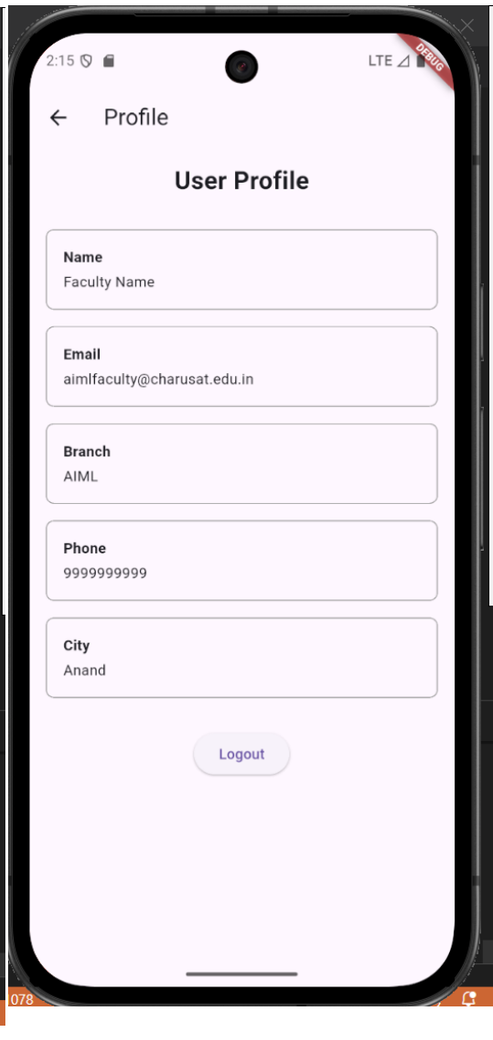


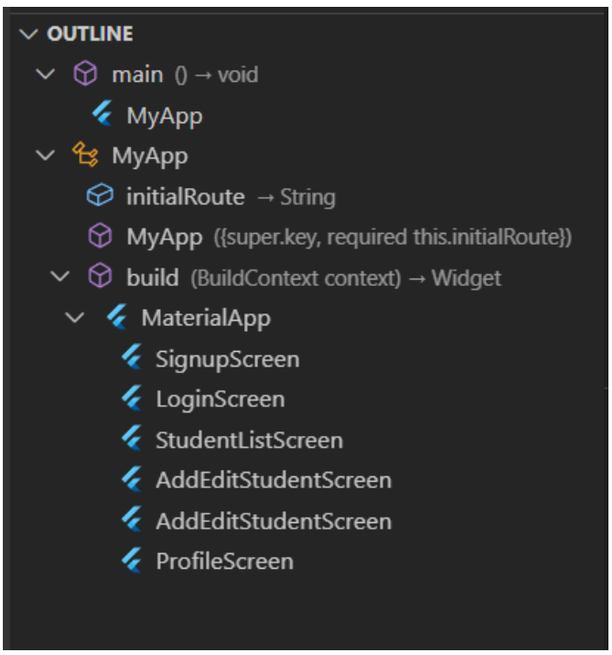


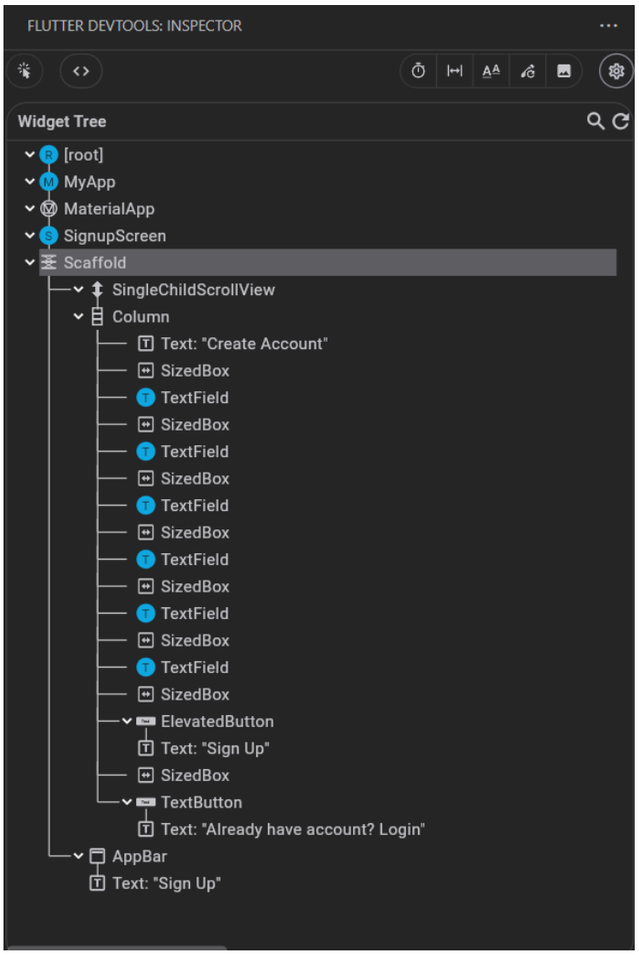












Key Questions:

1. How to store, update, retrieve data locally?
Ans: Data is stored, updated, and retrieved using the sqflite package, with all CRUD operations implemented in DatabaseHelper. Student records are inserted, updated, deleted, and queried from a local SQLite database table (students).

2. What is the role of FutureBuilder?
Ans: FutureBuilder is used to asynchronously fetch and display student records from the database, ensuring the UI updates automatically when data is loaded or changed.

3. How to connect to SQLite?
Ans: The app connects to SQLite using the sqflite package. The database is initialized in DatabaseHelper with openDatabase, and all queries are performed using async methods (insert, query, update, delete).

Key Skills to be Understand:

Database operations, CRUD logic

Implement local DB solutions

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING
CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Subject : Mobile Application Development **Semester:** 5

Subject Code: AIML308 **Academic Year :**2025-26(ODD)

NAME : CHOKSI OM CHIRAGBHAI **ID: 23AIML010**

Practical 6

Problem Definition

You are building a mobile application where users navigate through multiple screens like login, dashboard, and profile. Create a Notes App with persistent storage using Shared Preferences.

Supplementary Problems -	Dark mode toggle, Remember me feature

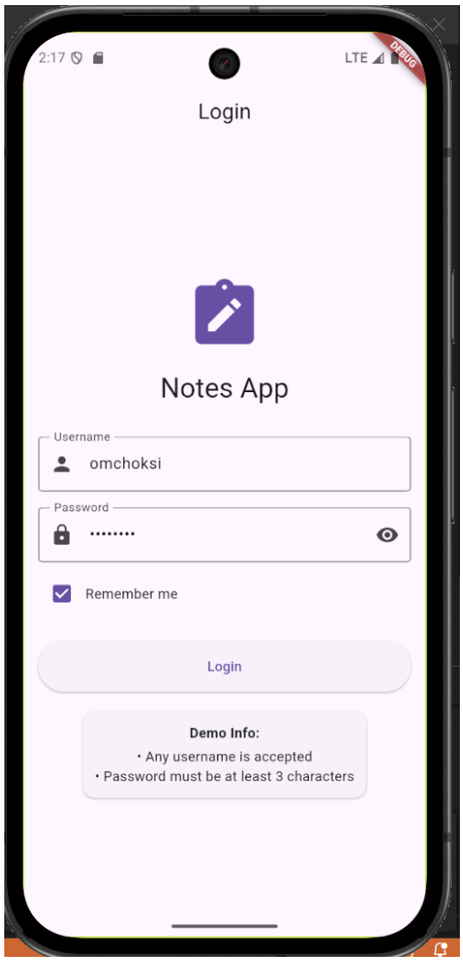
Technical Approach:

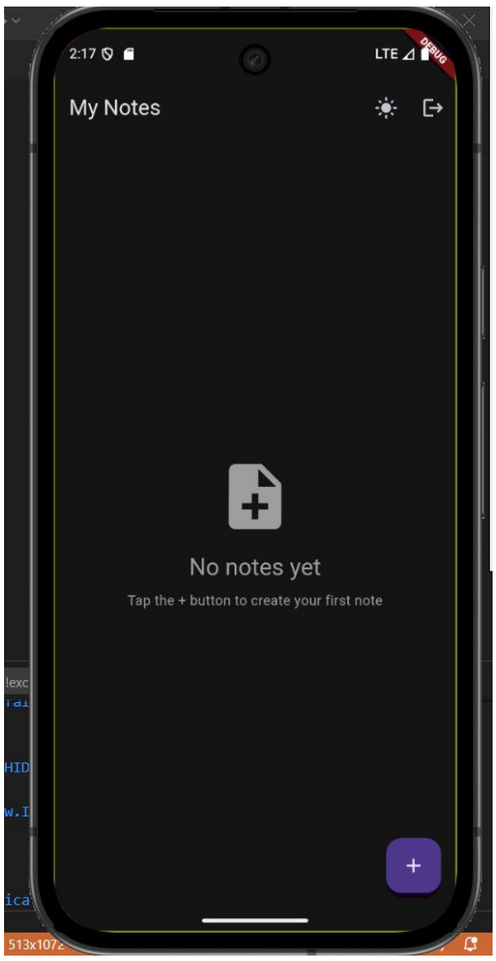
This Flutter application implements a notes management system using SharedPreferences for local data persistence across app sessions. The core logic revolves around StatefulWidget for managing UI state changes with setState, SharedPreferences for storing user preferences and notes data, and JSON encoding/decoding for complex data serialization. Key widgets include ListView.builder for displaying the notes list, Form with TextFormField for input validation, and Material Design components for consistent theming and navigation between screens.

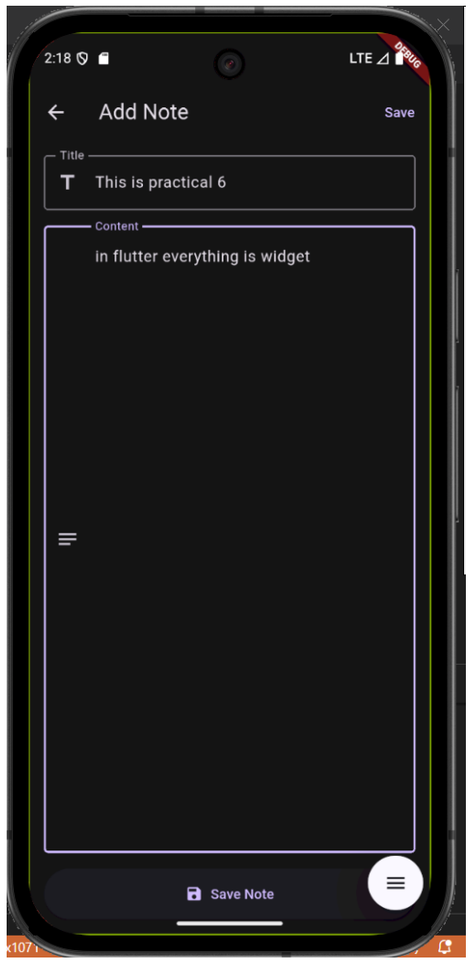
File Structure:

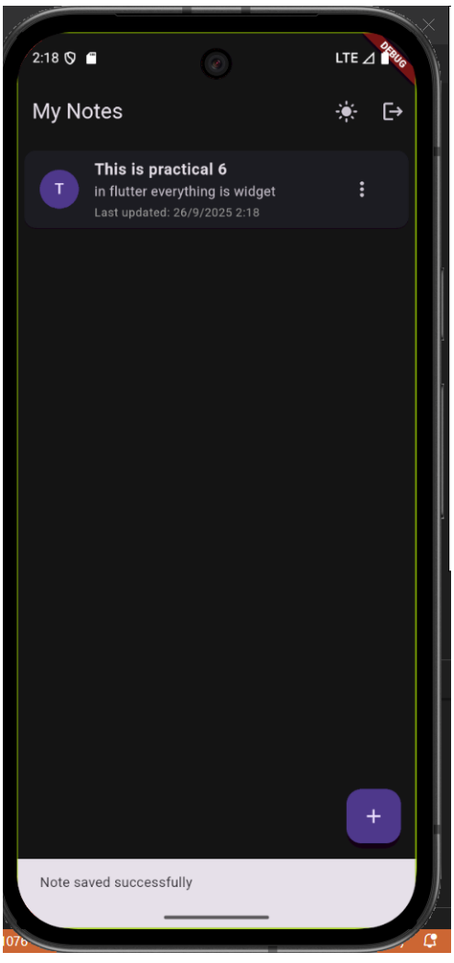
```
lib/
├── main.dart                # Application entry point with theme management
├── models/
│   └── note.dart            # Note data model with JSON serialization
├── screens/
│   ├── login_screen.dart       # Login interface with remember me functionality
│   ├── dashboard_screen.dart   # Notes list with dark mode toggle
│   └── add_edit_note_screen.dart # Create/edit notes interface
└── services/
    └── preferences_service.dart  # SharedPreferences service layer
```

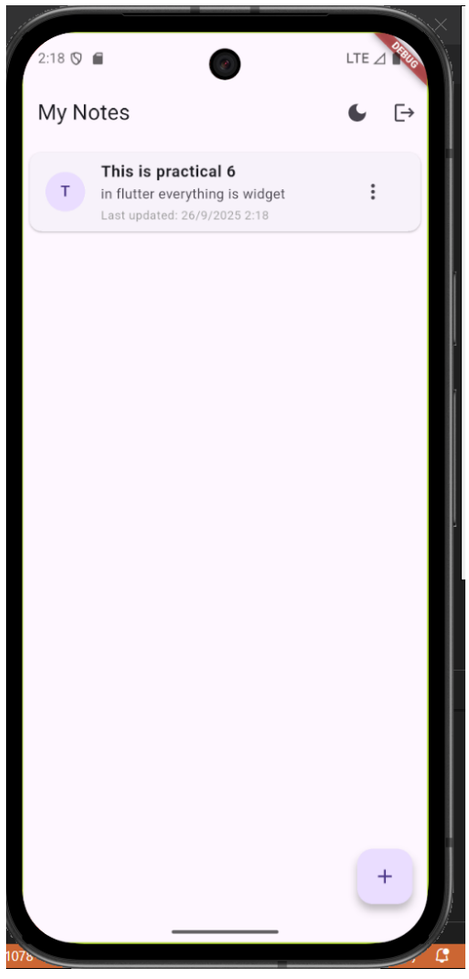
Screenshots:

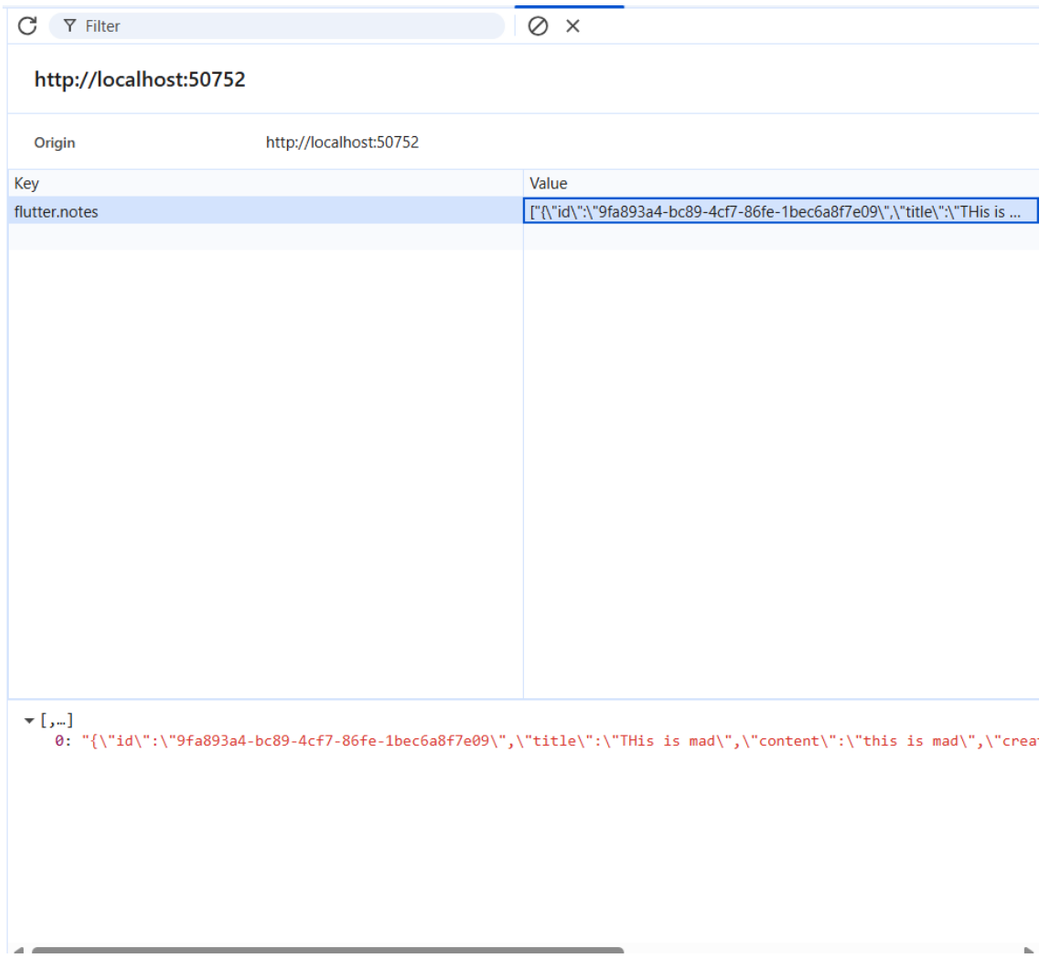


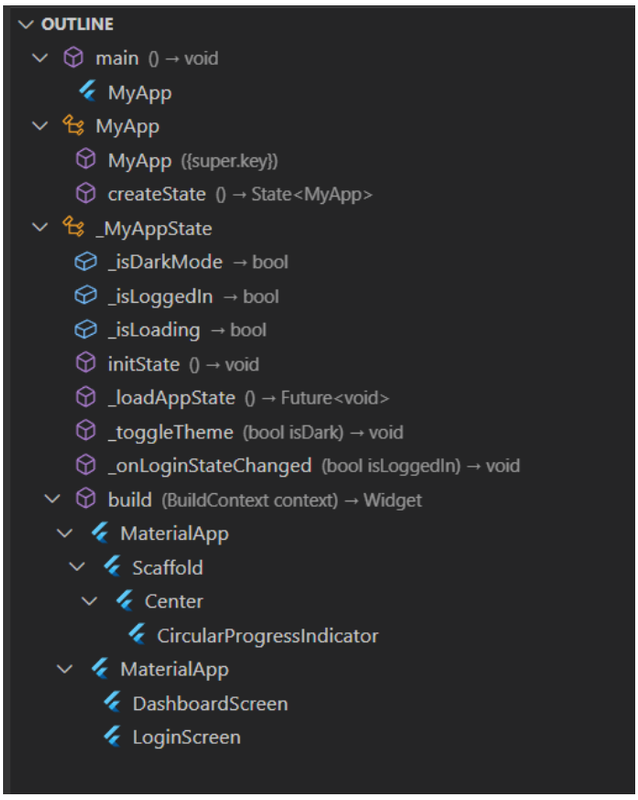


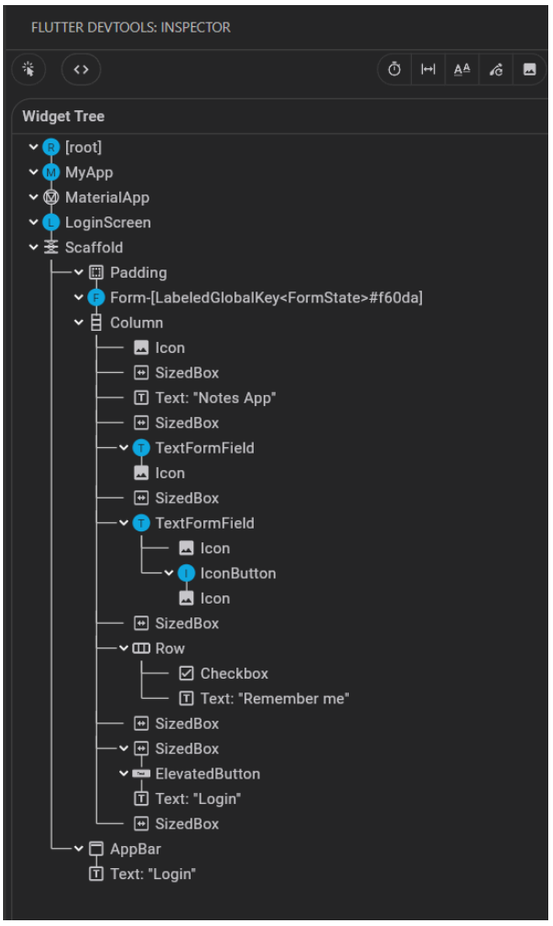







Filter
http://localhost:50752
Origin
http://localhost:50752
Key
Value
flutter.notes
["{\"id\":\"9fa893a4-bc89-4cf7-86fe-1bec6a8f7e09\",\"title\":\"THis is ...
[,…]
0: "{\"id\":\"9fa893a4-bc89-4cf7-86fe-1bec6a8f7e09\",\"title\":\"THis is mad\",\"content\":\"this is mad\",\"crea


OUTLINE
main () → void
MyApp
MyApp
MyApp ({super.key})
createState () → State<MyApp>
_MyAppState
_isDarkMode → bool
_isLoggedIn → bool
_isLoading → bool
initState () → void
_loadAppState () → Future<void>
_toggleTheme (bool isDark) → void
_onLoginStateChanged (bool isLoggedIn) → void
build (BuildContext context) → Widget
MaterialApp
Scaffold
Center
CircularProgressIndicator
MaterialApp
DashboardScreen
LoginScreen


FLUTTER DEVTOOLS: INSPECTOR
Widget Tree
[root]
MyApp
MaterialApp
LoginScreen
Scaffold
Padding
Form-[LabeledGlobalKey<FormState>#f60da]
Column
Icon
SizedBox
Text: "Notes App"
SizedBox
TextFormField
Icon
SizedBox
TextFormField
Icon
IconButton
Icon
SizedBox
Row
Checkbox
Text: "Remember me"
SizedBox
SizedBox
ElevatedButton
Text: "Login"
SizedBox
AppBar
Text: "Login"

Key Questions:

1. When to use shared preferences?.
Ans: Use SharedPreferences for storing lightweight data that needs to persist across app sessions, such as user preferences (theme, language), session tokens, simple configuration flags, and small amounts of structured data under 1MB, as implemented in this notes app for theme settings, login state, and user credentials

2. What data types can be stored?
Ans: SharedPreferences supports primitive data types including booleans, strings, integers, doubles, and string lists, as demonstrated in the app where boolean values store theme preferences and login state, strings store usernames/passwords, and string lists store JSON-encoded notes data.

3. How to persist small data?
Ans: Small data is persisted by obtaining a SharedPreferences instance, using appropriate setter methods (setBool, setString, setStringList) for different data types, and retrieving values with corresponding getter methods, ensuring data survives app restarts as shown in the PreferencesService class for theme, login, and notes storage.

Key Skills to be Understand:

Local storage, Preference handling

Implement local DB solutions

Store app settings or session

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING
CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Subject : Mobile Application Development **Semester:** 5

Subject Code: AIML308 **Academic Year :**2025-26(ODD)

NAME : CHOKSI OM CHIRAGBHAI **ID: 23AIML010**

Practical 7

Problem Definition

You are building a mobile application where users navigate through multiple screens like login, dashboard, and profile. Design a Product Catalog App using GridView and custom cards with images.

Supplementary Problems -	Gallery app or Recipe app

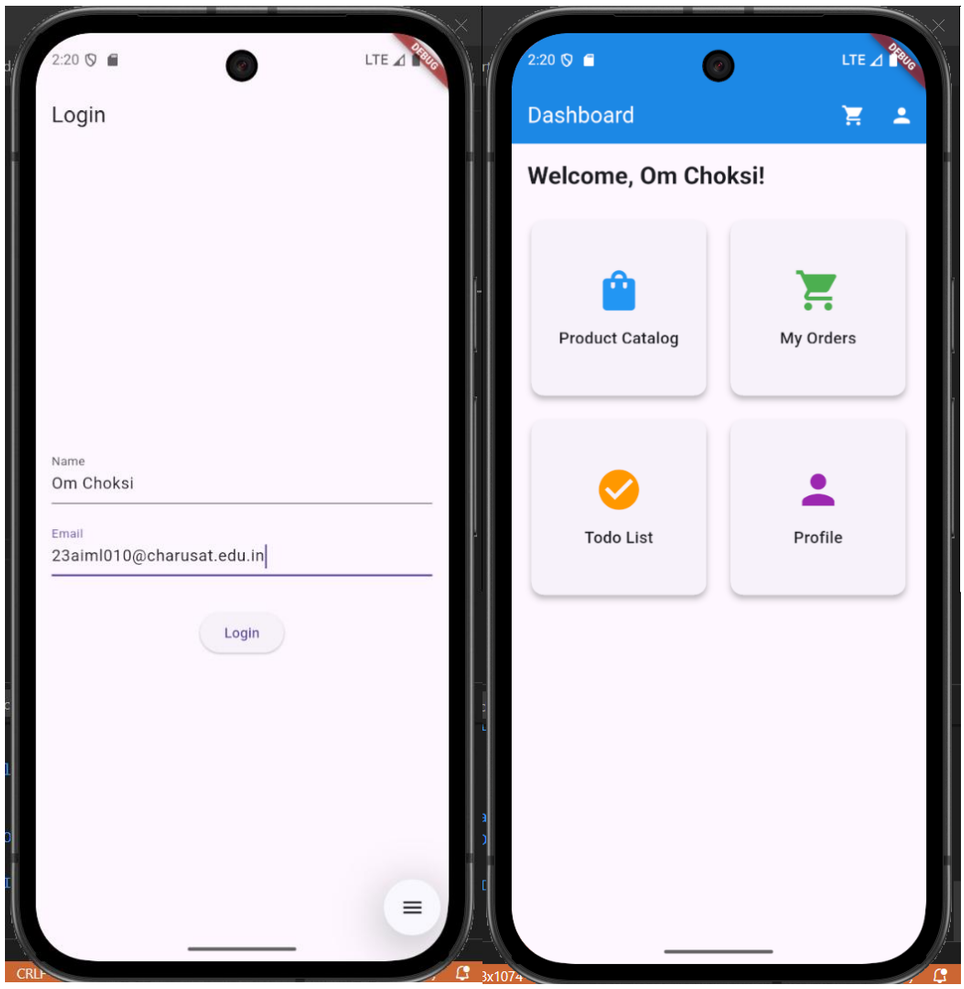
Technical Approach:

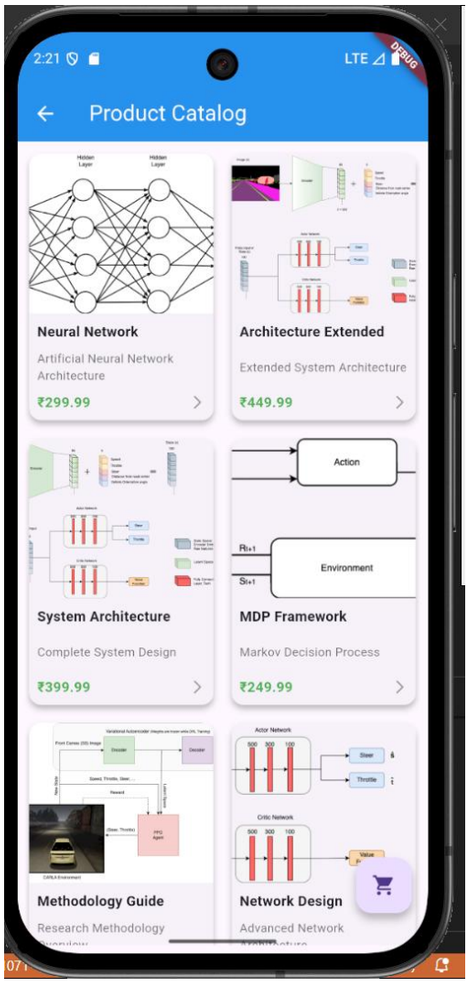
The application implements a product catalog using Flutter's GridView for responsive grid layouts, with custom reusable ProductCard widgets ensuring consistent UI design. State management is achieved through the Provider package, utilizing a CartManager class that extends ChangeNotifier for dynamic cart functionality across screens. Core widgets include StatefulWidget for interactive product details and cart management, GridView for product display, and Image.asset for loading local assets, with setState handling local UI updates in quantity selectors and form validations.

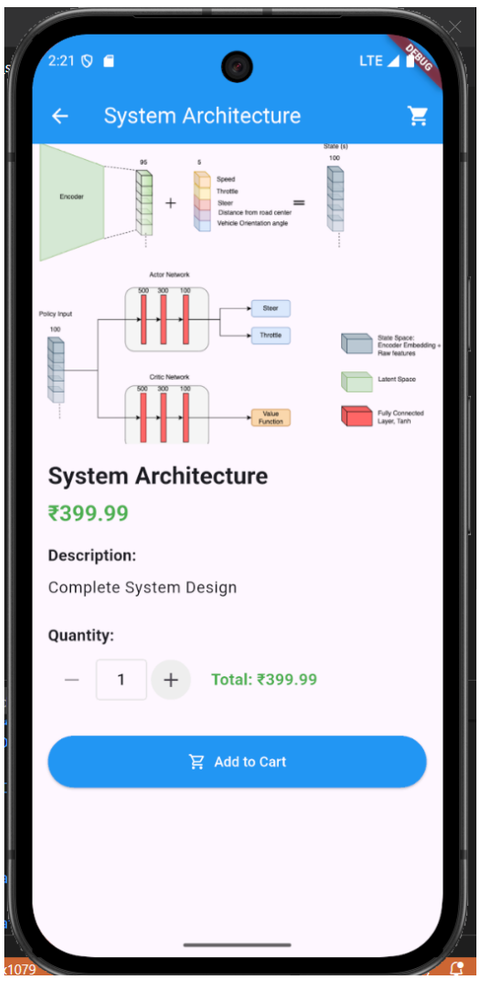
File Structure:

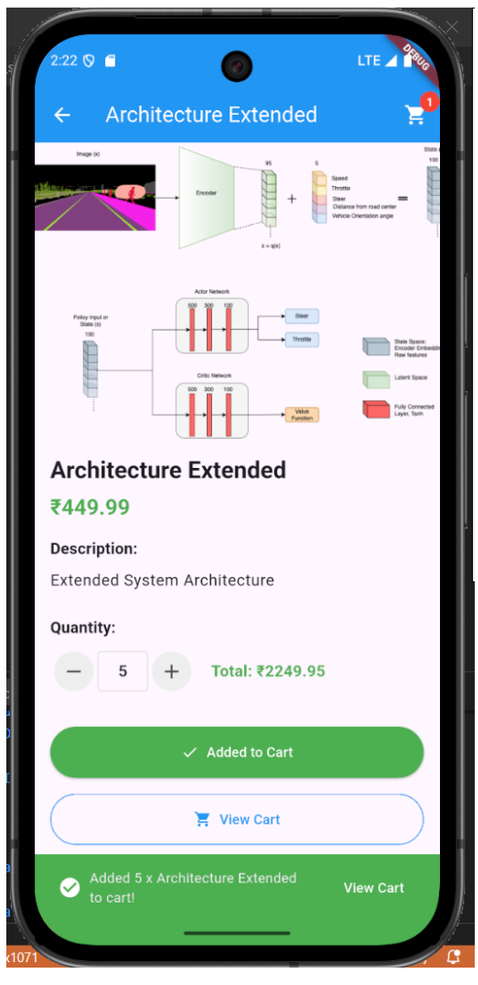
```
lib/
├── cart_manager.dart     // Cart management: add/remove items, update quantities.
├── dashboard.dart        // Main dashboard: app overview and navigation.
├── login.dart            // User auth: login and registration.
├── main.dart             // App entry point: initializes root widget.
├── orders.dart           // Order management: history, details, placement.
├── product_catalog.dart  // Product catalog: browse and search items.
├── product_detail.dart   // Product details: info, images, options.
├── profile.dart          // User profile: details, settings, account.
├── todolist.dart         // To-do list: create, edit, track tasks.
└── widgets/
    └── product_card.dart // Reusable product card widget for lists/catalogs.
```

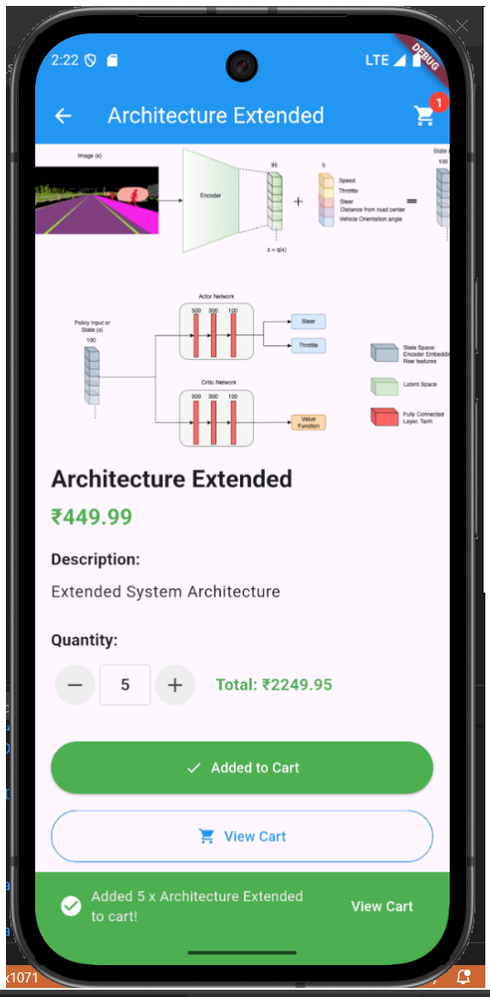
Screenshots:

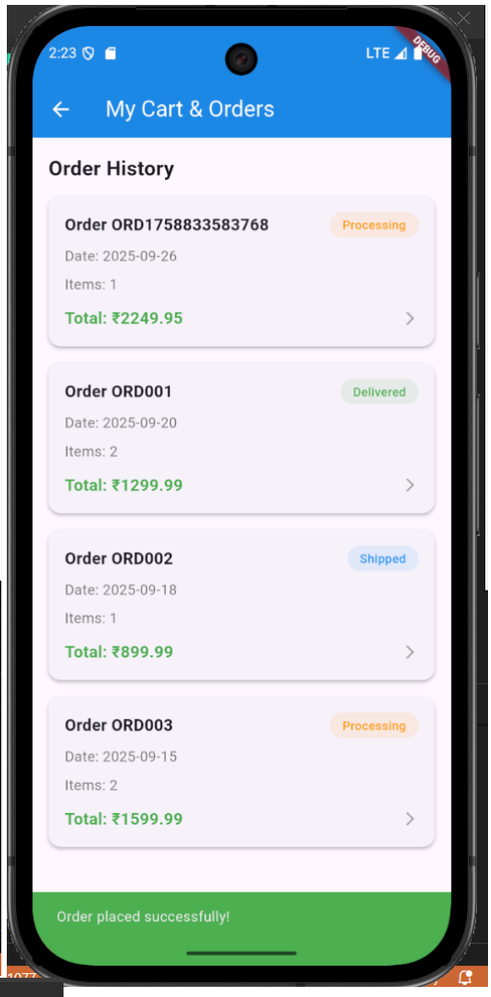


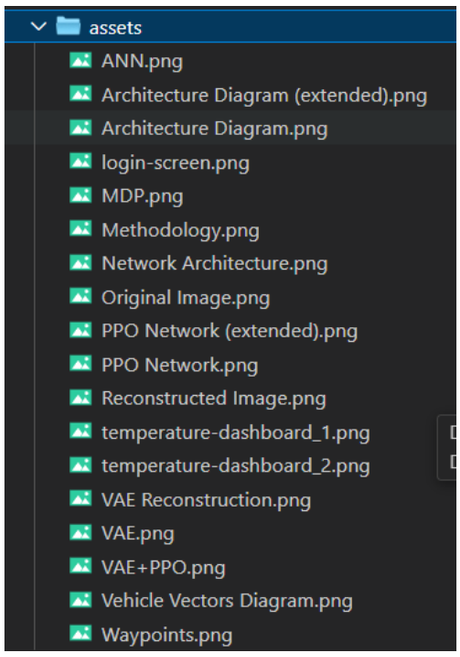


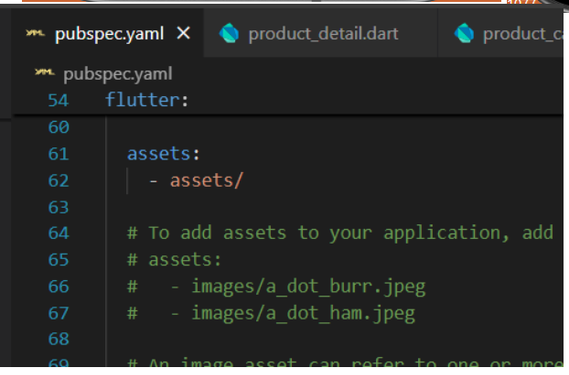


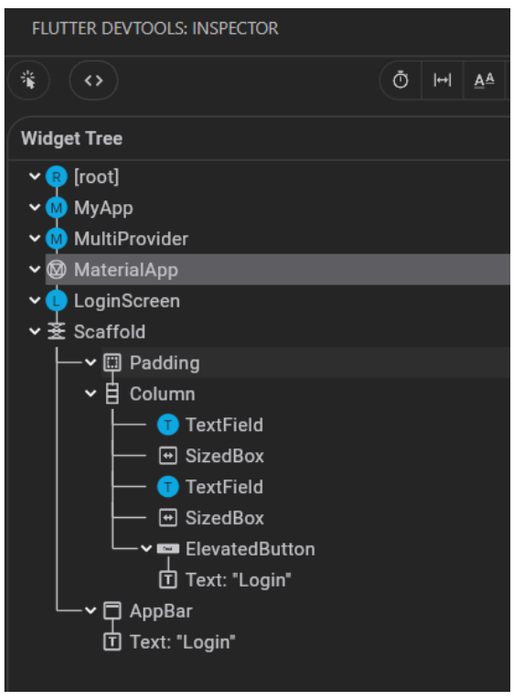


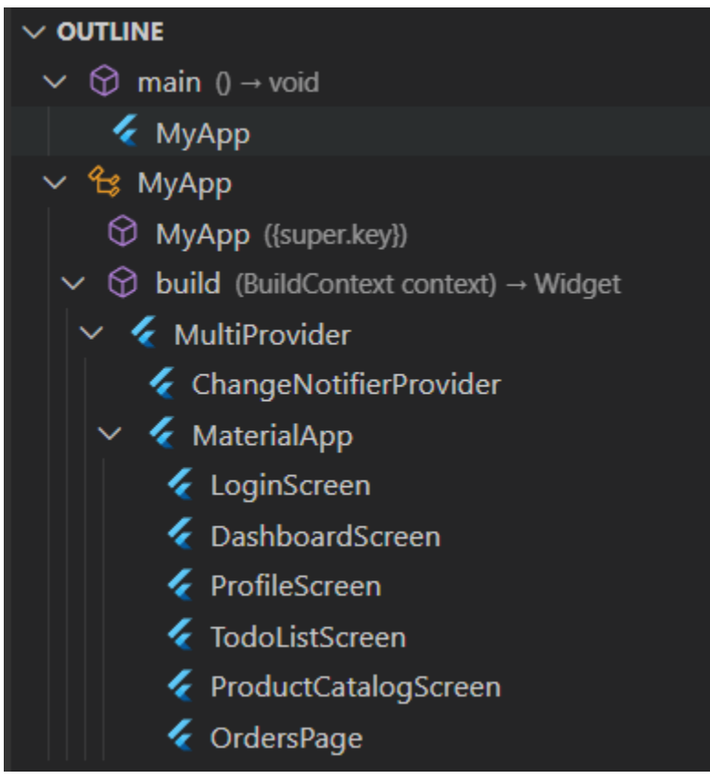

2:22
LTE
Architecture Extended
Architecture Extended
₹449.99
Description:
Extended System Architecture
Quantity:
5
Total: ₹2249.95
Added to Cart
View Cart
Added 5 x Architecture Extended to cart!
View Cart


2:23
LTE
My Cart & Orders
Order History
Order ORD1758833583768
Processing
Date: 2025-09-26
Items: 1
Total: ₹2249.95
Order ORD001
Delivered
Date: 2025-09-20
Items: 2
Total: ₹1299.99
Order ORD002
Shipped
Date: 2025-09-18
Items: 1
Total: ₹899.99
Order ORD003
Processing
Date: 2025-09-15
Items: 2
Total: ₹1599.99
Order placed successfully!


assets
ANN.png
Architecture Diagram (extended).png
Architecture Diagram.png
login-screen.png
MDP.png
Methodology.png
Network Architecture.png
Original Image.png
PPO Network (extended).png
PPO Network.png
Reconstructed Image.png
temperature-dashboard_1.png
temperature-dashboard_2.png
VAE Reconstruction.png
VAE.png
VAE+PPO.png
Vehicle Vectors Diagram.png
Waypoints.png


pubspec.yaml
product_detail.dart
pubspec.yaml
54 flutter:
60
61 assets:
62 - assets/
63
64 # To add assets to your application, add
65 # assets:
66 # - images/a_dot_burr.jpeg
67 # - images/a_dot_ham.jpeg
68


FLUTTER DEVTOOLS: INSPECTOR
Widget Tree
[root]
MyApp
MultiProvider
MaterialApp
LoginScreen
Scaffold
Padding
Column
TextField
SizedBox
TextField
SizedBox
ElevatedButton
Text: "Login"
AppBar
Text: "Login"


OUTLINE
main () → void
MyApp
MyApp
MyApp ({super.key})
build (BuildContext context) → Widget
MultiProvider
ChangeNotifierProvider
MaterialApp
LoginScreen
DashboardScreen
ProfileScreen
TodoListScreen
ProductCatalogScreen
OrdersPage

Key Questions:

1. How to create reusable custom widgets?
Ans: Create a separate StatelessWidget or StatefulWidget class in a dedicated file (e.g., lib/widgets/product_card.dart), define required parameters in the constructor, and use it across multiple screens by importing and instantiating the widget with appropriate props, as demonstrated by the ProductCard widget used in the GridView.

2. How to use GridView for layout?
Ans: Use GridView.builder or GridView.count with SliverGridDelegateWithFixedCrossAxisCount to specify cross-axis count, spacing, and aspect ratio, then provide an itemBuilder function that returns widgets for each grid item, as implemented in the ProductCatalogScreen for displaying product cards in a 2-column grid.

3. How to load local assets (images)?
Ans: Add image files to the assets/ directory, configure the path in pubspec.yaml under the assets section, then use Image.asset() widget with the asset path string, including error handling with errorBuilder to display fallback content if the image fails to load, as shown in the ProductCard widget.

Key Skills to be Understand:

UI design, Reusability, Grid layout

Design reusable and responsive UI

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING
CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Subject : Mobile Application Development **Semester:** 5

Subject Code: AIML308 **Academic Year :**2025-26(ODD)

NAME : CHOKSI OM CHIRAGBHAI **ID: 23AIML010**

Practical 8

Problem Definition

You are building a mobile application where users navigate through multiple screens like login, dashboard, and profile. Design a Product Catalog App using GridView and custom cards with images.

Supplementary Problems -	News app

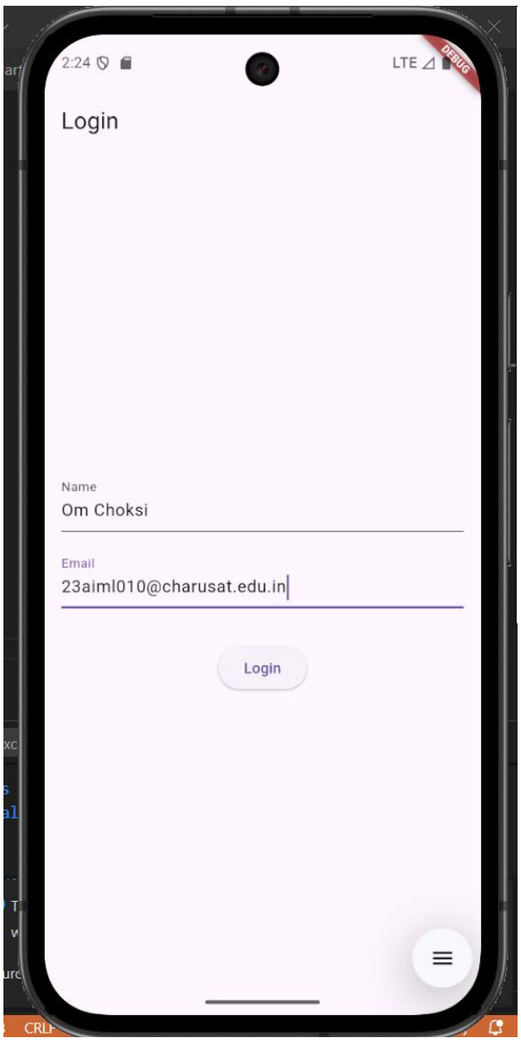
Technical Approach:

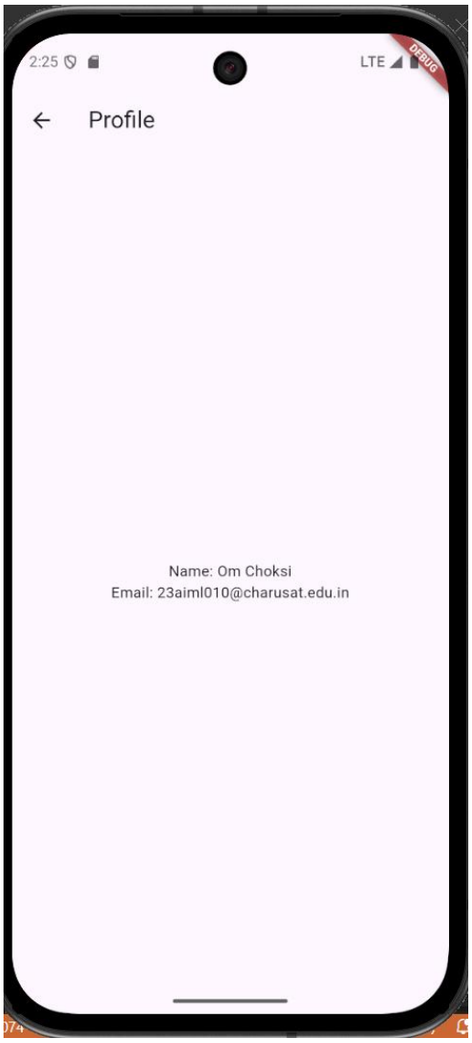
This Flutter application implements a multi-screen news app with REST API integration, focusing on asynchronous data handling and JSON parsing. The core implementation uses the http package for API calls, flutter_dotenv for secure environment variable management, and FutureBuilder for reactive UI updates based on async operations. Key widgets include StatefulWidget for state management, FutureBuilder for handling loading/error/success states, ListView.builder for dynamic article rendering, and Scaffold for consistent app structure across screens.

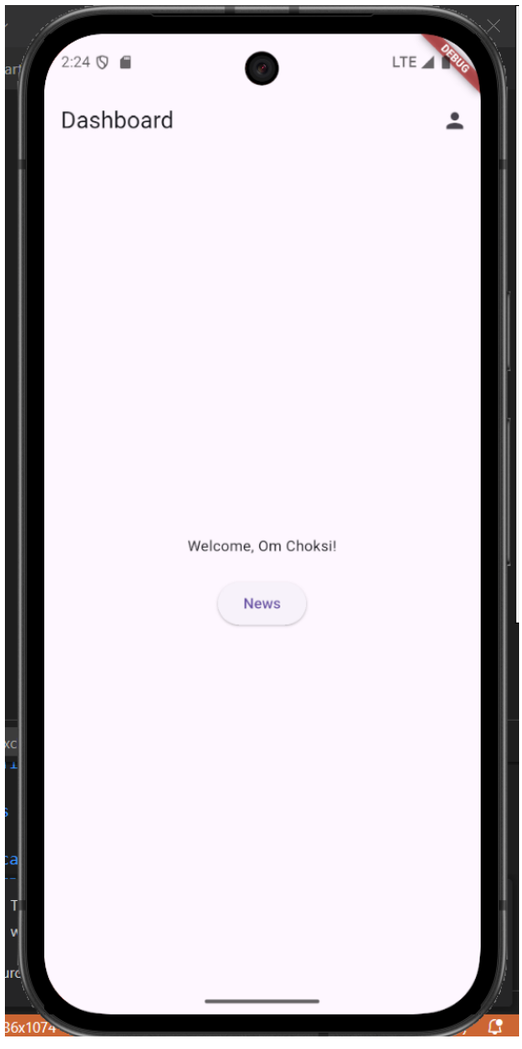
File Structure:

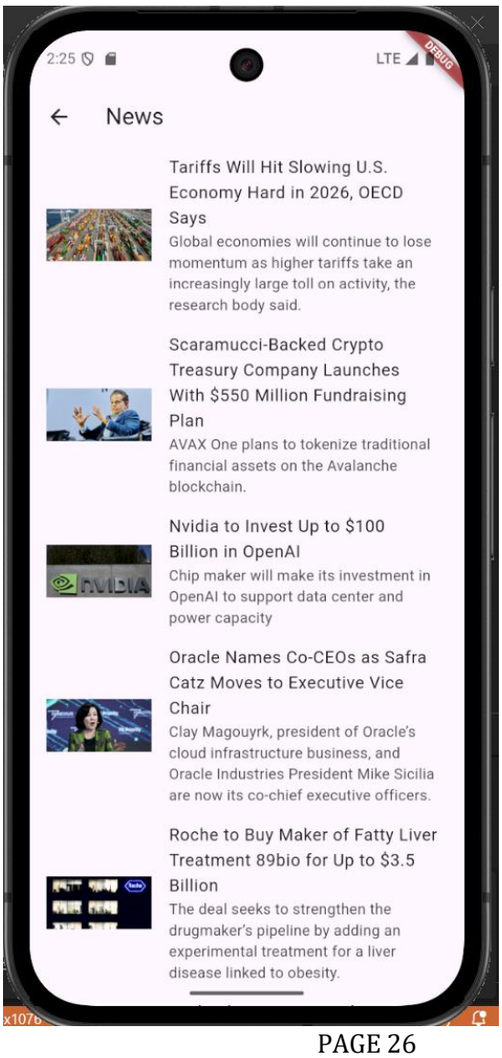
```
lib/
├── main.dart  # Main entry point of the Flutter application
├── login.dart  # Handles user authentication and login screen
├── dashboard.dart  # Displays the main dashboard after login
├── profile.dart  # Manages user profile information
├── news.dart  # Fetches and displays news articles
└── models/
    └── article.dart  # Defines the Article model class
```

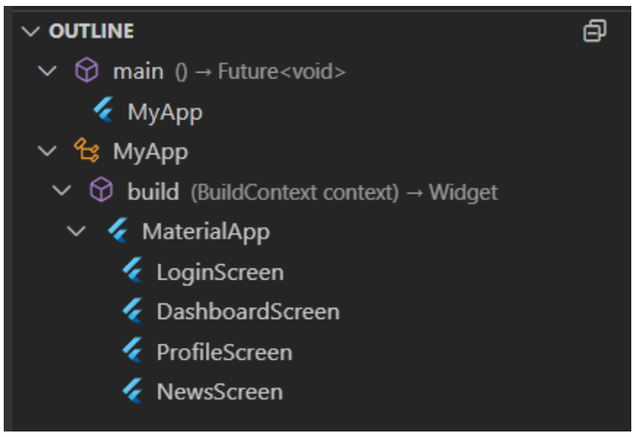
Screenshots:

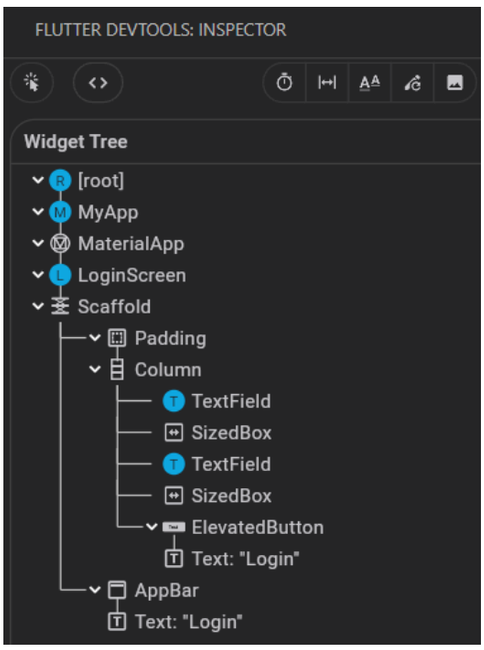












Key Questions:

1. What is JSON parsing?
Ans: JSON parsing converts the JSON response string from the News API into structured Dart objects using factory constructors like Article.fromJson(), which maps JSON fields to object properties for easy data manipulation in the app.

2. How to handle async APIs?
Ans: Async APIs are handled using async/await syntax in functions like fetchNews(), combined with FutureBuilder widget that reacts to different connection states, ensuring UI updates when data arrives without blocking the main thread.

3. How to deal with loading/error states?
Ans: Loading states are managed with FutureBuilder's ConnectionState.waiting showing a CircularProgressIndicator, error states use snapshot.hasError to display error messages, and success states render data using snapshot.hasData with ListView.builder for article display.

Key Skills to be Understand:

API consumption, Async/Await

Handle API data and render UI

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING
CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Subject : Mobile Application Development **Semester:** 5

Subject Code: AIML308 **Academic Year :**2025-26(ODD)

NAME : CHOKSI OM CHIRAGBHAI **ID: 23AIML010**

Practical 9

Problem Definition

You are building a mobile application where users navigate through multiple screens like login, dashboard, and profile. Develop a Login Authentication App using API-based credential check and session handling.

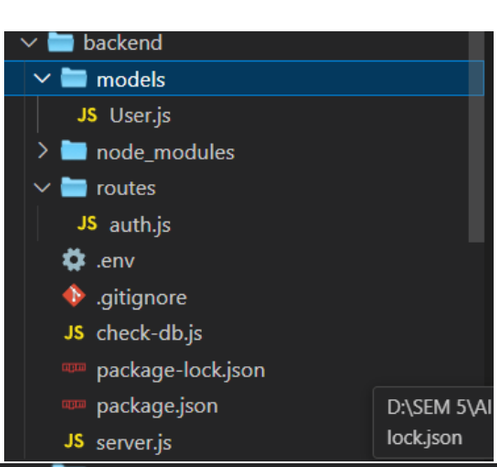
Supplementary Problems -	Two-factor login or role-based access

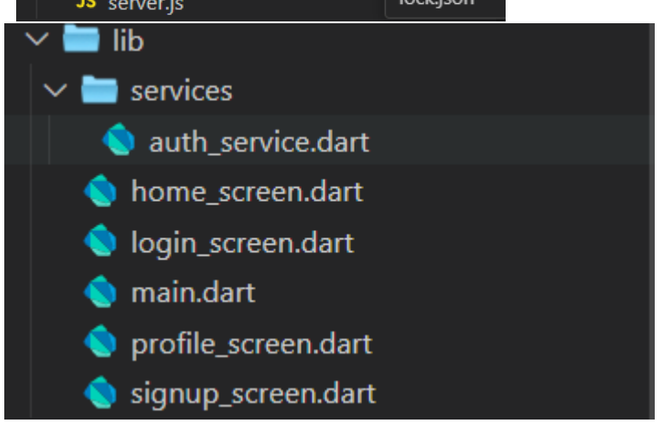
Technical Approach:

This Flutter application implements a multi-screen news app with REST API integration, focusing on asynchronous data handling and JSON parsing. The core implementation uses the http package for API calls, flutter_dotenv for secure environment variable management, and FutureBuilder for reactive UI updates based on async operations. Key widgets include StatefulWidget for state management, FutureBuilder for handling loading/error/success states, ListView.builder for dynamic article rendering, and Scaffold for consistent app structure across screens.

File Structure:

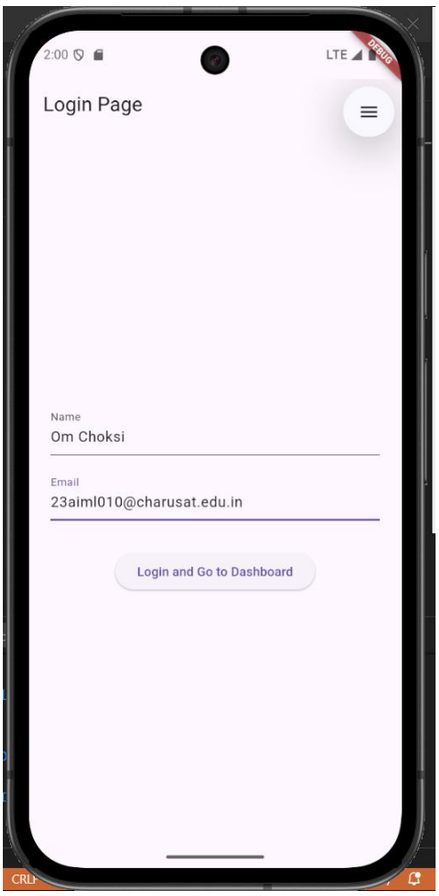
```
lib/
├── main.dart
├── home_screen.dart
├── login_screen.dart
├── profile_screen.dart
├── services/
│   └── auth_service.dart
└── signup_screen.dart

backend/

├── .env
├── .gitignore
├── check-db.js
├── package.json
├── package-lock.json
├── server.js
├── models/
│   └── User.js
```

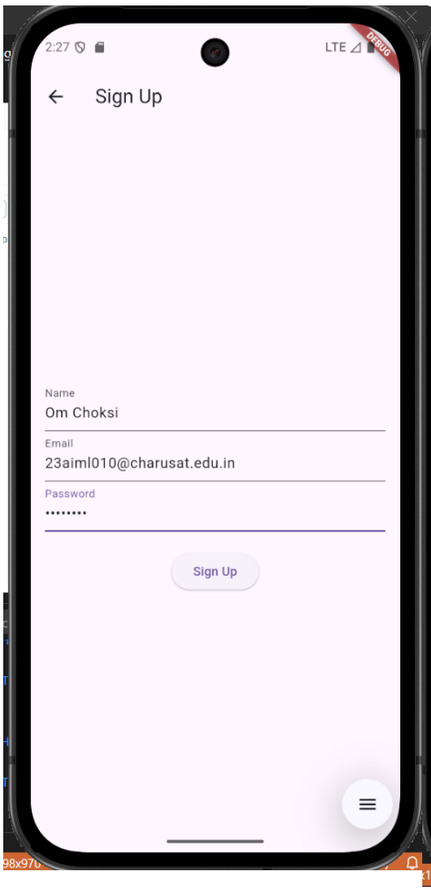


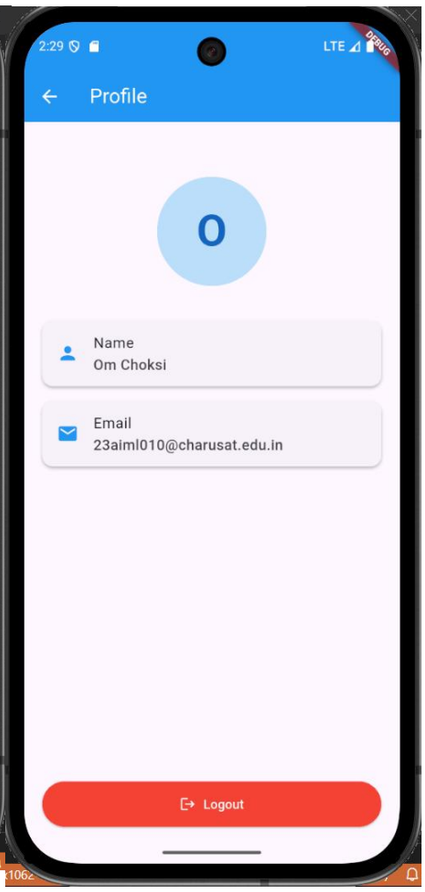


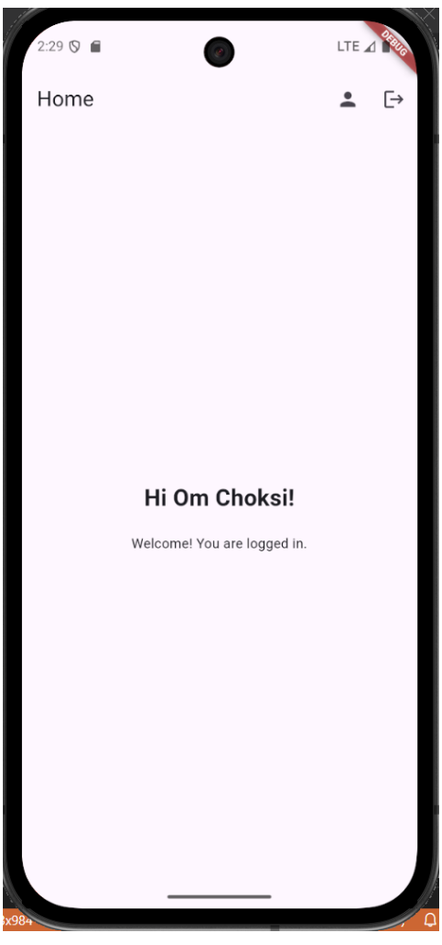
```
├── routes/
│   └── auth.js
```

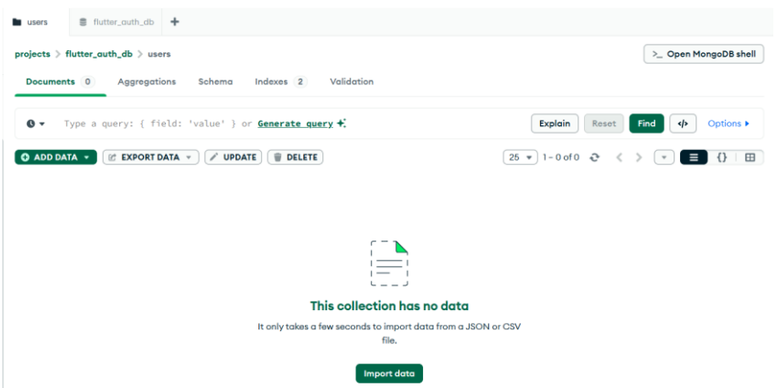
Screenshots:

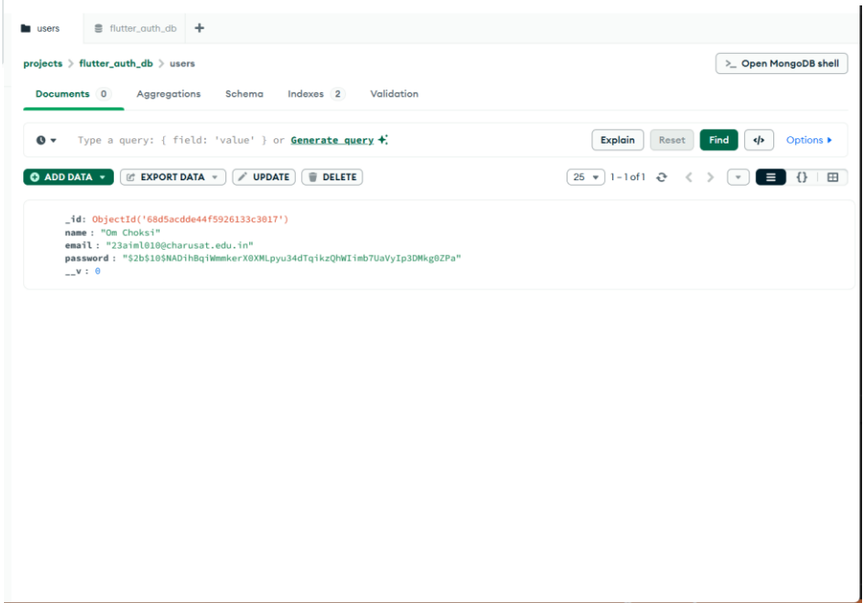


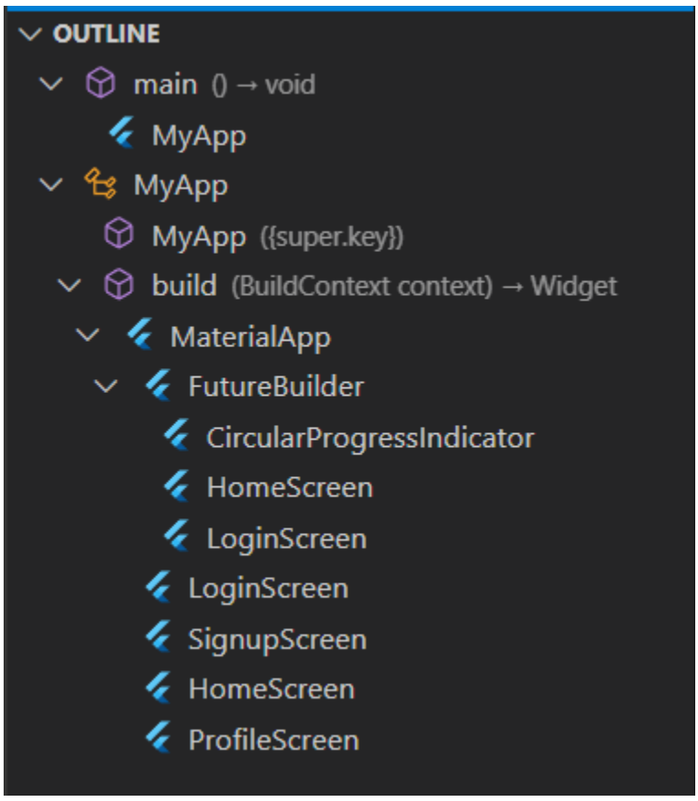


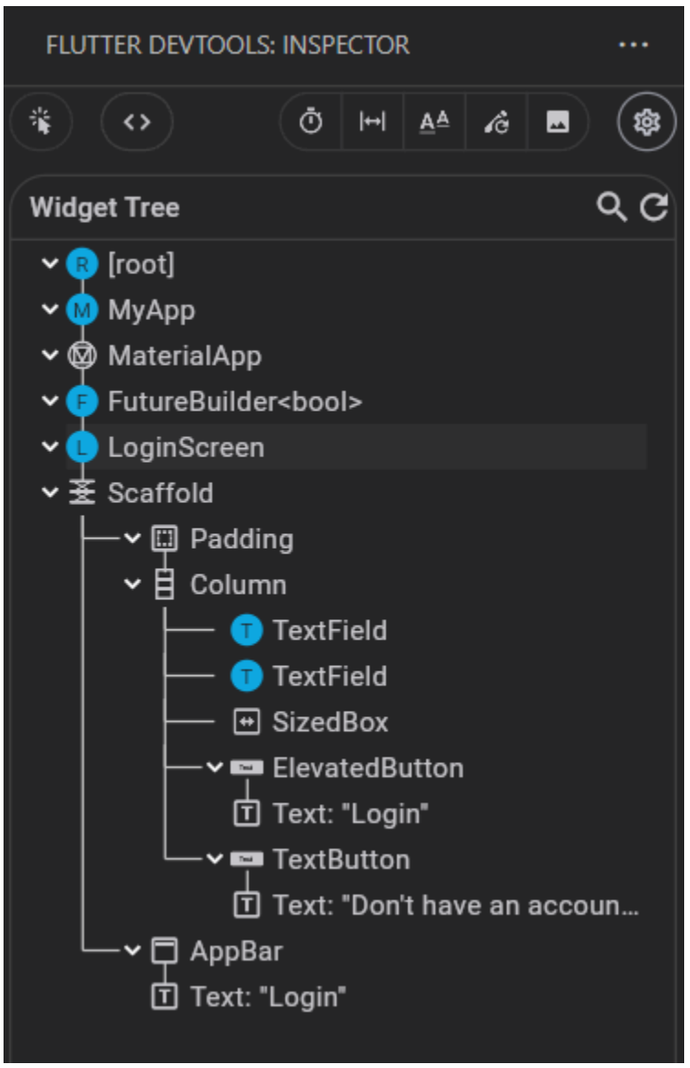












Key Questions:

1. How to send login credentials securely?**
Ans: Credentials are sent via HTTPS POST requests with JSON payload containing email and password, using proper headers and secure token-based authentication with JWT for subsequent requests.

2. How to store token/session?
Ans: JWT tokens and user data are stored locally using SharedPreferences, with automatic session restoration on app restart and complete cleanup on logout.

3. How to handle invalid login?
Ans: Invalid login attempts return appropriate HTTP status codes (400) with error messages, which are parsed from JSON response and displayed to users via SnackBar notifications with user-friendly error text.

Key Skills to be Understand:

Auth Logic, API calls, Sessions

Implement login flow using APIs

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING
CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Subject : Mobile Application Development **Semester:** 5

Subject Code: AIML308 **Academic Year :**2025-26(ODD)

NAME : CHOKSI OM CHIRAGBHAI **ID: 23AIML010**

Practical 10

Problem Definition

You are building a mobile application where users navigate through multiple screens like login, dashboard, and profile. Create and generate a Signed APK for deployment. Document steps to publish the app.

Supplementary Problems -	Include iOS build steps (document only)

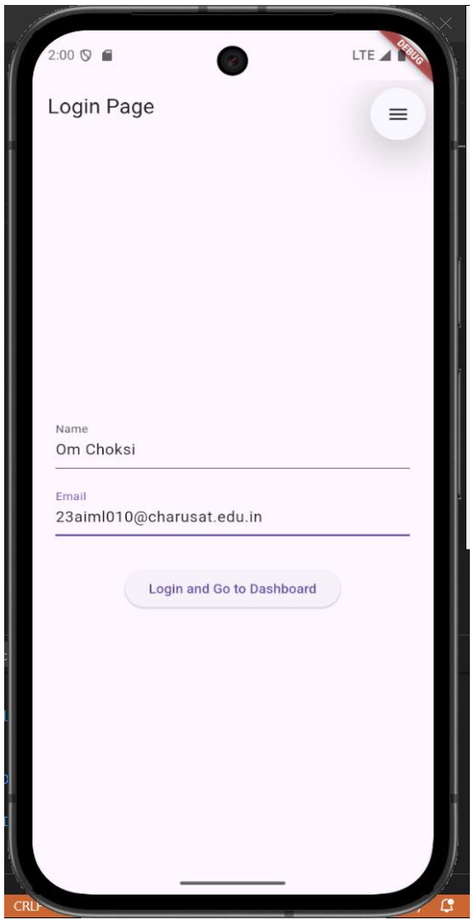
Technical Approach:

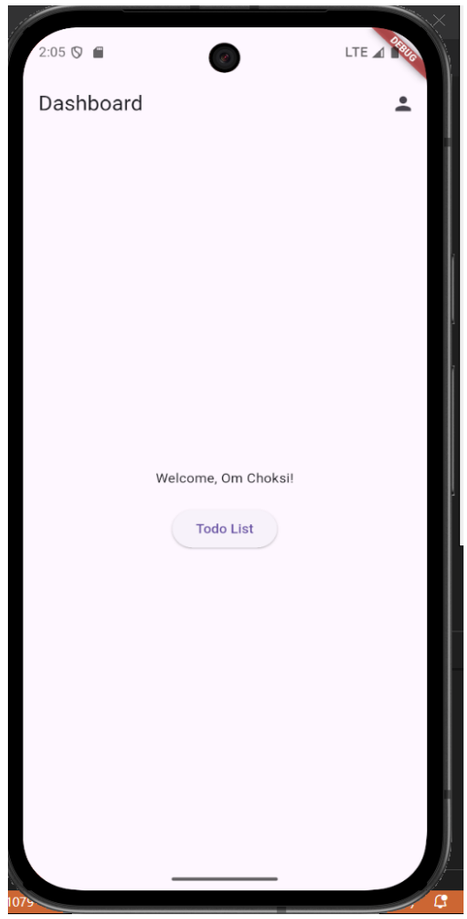
- Framework & Tools: Flutter (Dart) with VS Code IDE, Gradle for Android builds, and Java keytool for keystore generation.
- App Architecture: Cross-platform mobile app with custom UI widgets for login, dashboard, profile, and to-do list features.
- Signing & Security: Generated RSA keystore for app signing, configured build.gradle for release builds with secure password handling.
- Build Process: Used Flutter CLI for cleaning and building signed release APKs, integrated with Android Gradle for deployment.
- Deployment & Future: Supports APK/AAB for Android Play Store; future enhancements include CI/CD, advanced state management, and backend integration. & Documentation

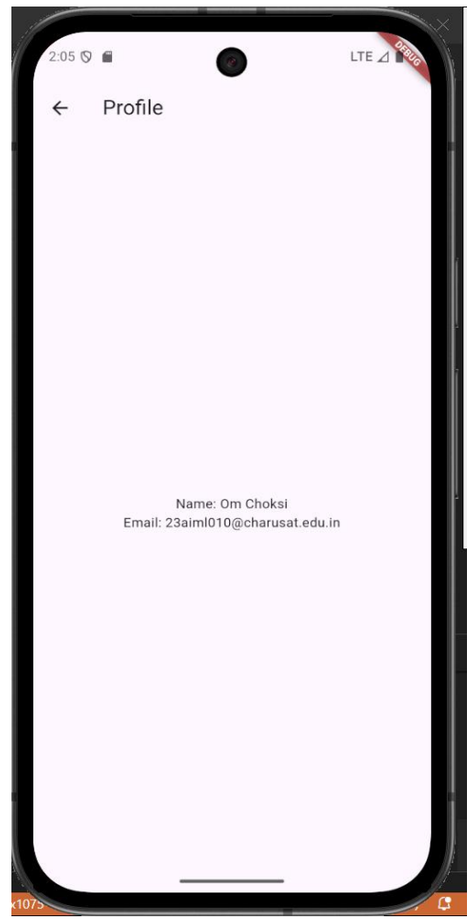
File Structure:

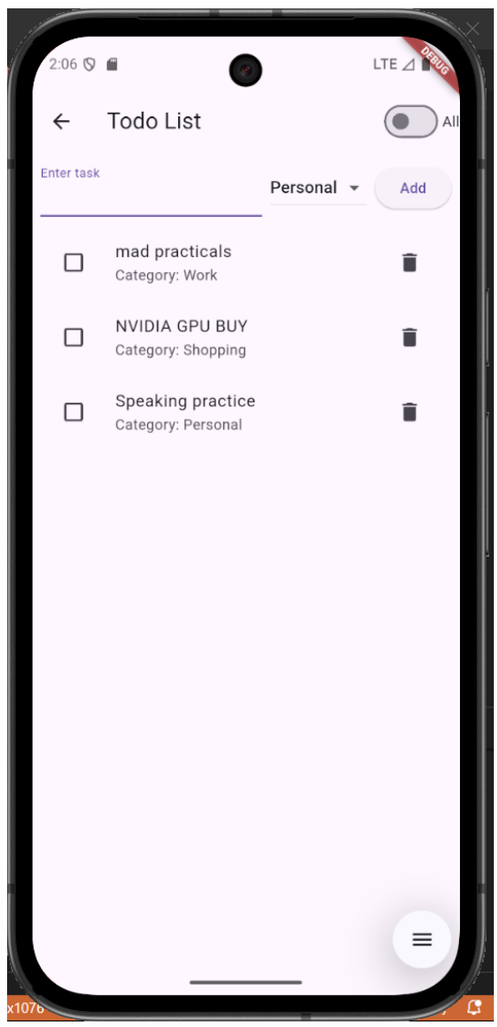
```
lib/
├── main.dart        # App entry point with route configuration
├── login.dart       # Login screen with name and email input
├── dashboard.dart   # Dashboard with user greeting and navigation
├── profile.dart     # Profile screen displaying user details
└── todolist.dart    # TODO list screen with dynamic task management
```

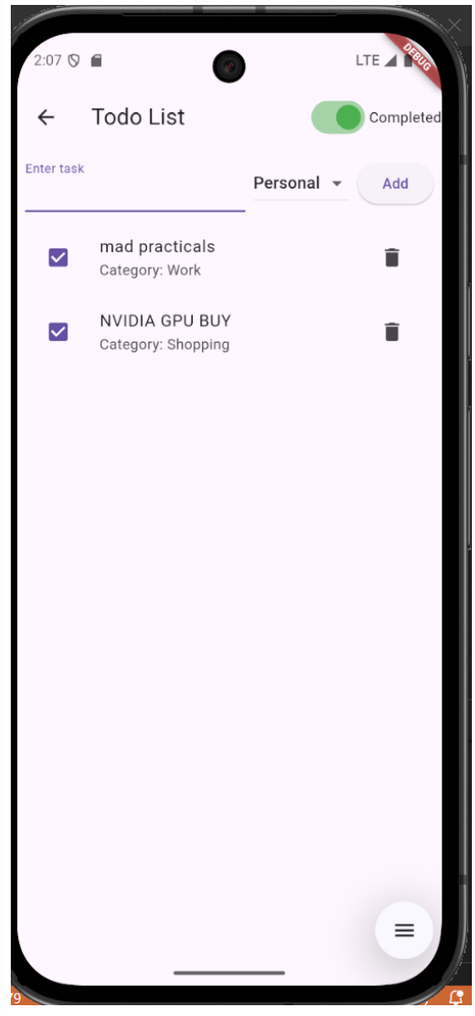
Screenshots:

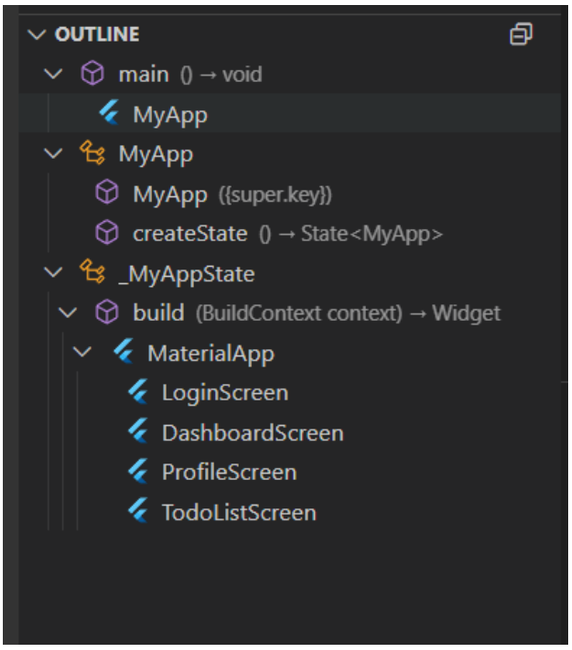


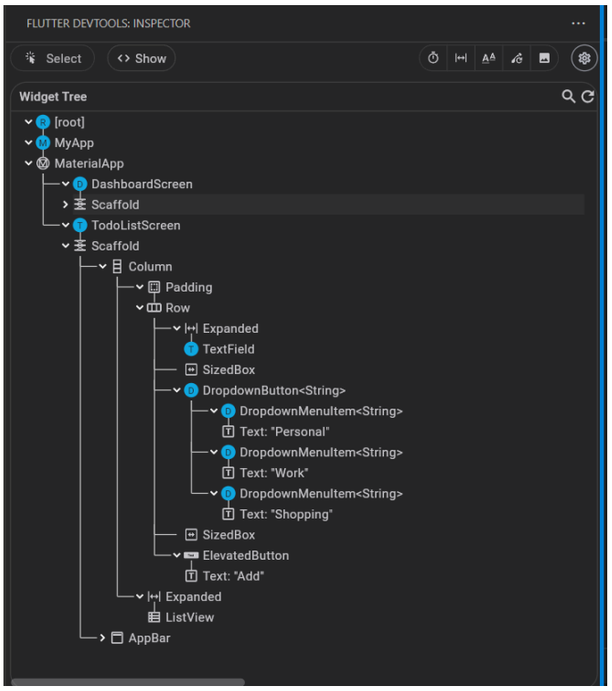












KeySt ore commands

- cd .\android\app
  - Navigates to the Android app directory where the keystore and build files are located.

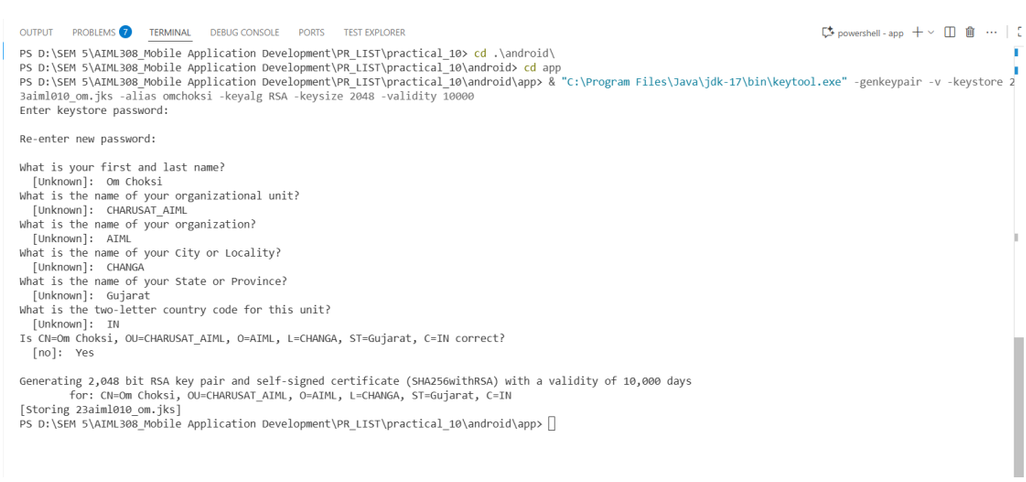
- & "C:\Program Files\Java\jdk-17\bin\keytool.exe" -genkeypair -v -keystore 23aiml010_om.jks -alias omchoksi -keyalg RSA -keysize 2048 -validity 10000

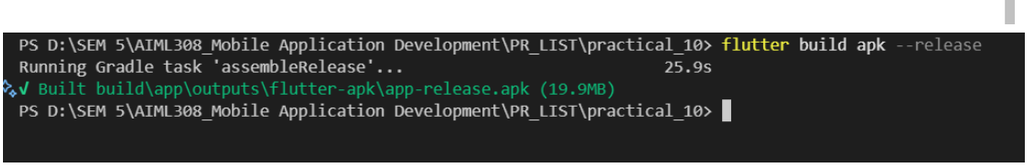
  - Generates a new keystore file (23aiml010_om.jks) with an RSA key pair for signing the Android app. The key is valid for 10,000 days.

- flutter clean
  - Cleans the Flutter project by removing build artifacts, caches, and temporary files to ensure a fresh build.

- cd android; ./gradlew clean
  - Navigates to the android directory and runs Gradle clean to remove Android build outputs and caches.

- flutter build apk --release
  - Builds a signed release APK for the Flutter app using the configured keystore for production deployment.





Key Questions:

1. What is a keystore?

Ans: A keystore is a secure file that stores private keys and certificates used for signing Android applications. It ensures that app updates are verified as coming from the same author, preventing unauthorized modifications and maintaining app integrity.

2. What are deployment options in Flutter?

- Ans: - Debug APK: Built for development with debugging enabled, not optimized for performance.
  - Profile APK: Includes profiling tools for performance analysis, used for testing app performance.
  - Release APK: Optimized, signed APK for production deployment on the Google Play Store.
  - App Bundle (AAB): A newer format that includes all app code and resources, optimized for distribution via Google Play.
  - iOS Build: For iOS deployment, involves building an IPA file via Xcode for App Store submission.

3. How to troubleshoot APK build issues?

- Ans: - PATH Issues: Ensure Java JDK is installed and keytool is in the system PATH. Use full paths if needed.
  - Keystore Not Found: Verify the keystore file path in key.properties and build.gradle is correct. The file should be in android/app/.
  - Gradle Errors: Check for syntax errors in build.gradle. Run ./gradlew clean and rebuild.
  - Mismatched Passwords: Ensure storePassword and keyPassword in key.properties match the keystore passwords exactly.
  - Common Fixes: Update Gradle wrapper, check Flutter doctor for issues, ensure all dependencies are resolved. Use --stacktrace for detailed error logs.

Key Skills to be Understand:

Deployment, Publishing, Versioning

Understand deployment pipeline