Exercices

Basic UI and Layouts

Exercise 1: Create a simple login screen with text fields for email and password, and a login button. Apply basic validation to ensure the fields are not empty.

Exercise 2: Design a profile screen that includes an avatar, user's name, bio, and a list of hobbies or interests. Practice using layout widgets like Row, Column, and Stack.

Exercise 3: Build a responsive grid layout to display a list of products or items, adapting to different screen sizes.

State Management

Exercise 4: Create a simple counter app using Riverpod. Add increment and decrement buttons and use state management to update the counter value.

Exercise 5: Develop a to-do list app where you can add, remove, and mark tasks as completed. Use Riverpod to manage the list state.

Exercise 6: Implement a shopping cart where you can add products, adjust quantities, and see the total price update dynamically.

Networking and API Integration

Exercise 7: Fetch data from a public API (e.g., JSONPlaceholder) and display a list of posts or comments. Include pull-to-refresh functionality.

Exercise 8: Build a weather app that fetches and displays current weather information based on the user's location using an API like OpenWeatherMap.

Exercise 9: Create a movie search app where users can search for movies and see the results fetched from an API like The Movie Database (TMDb).

Animations and Custom UI

Exercise 10: Create a loading animation using AnimatedBuilder and AnimationController. Design an animated progress indicator.

Exercise 11: Implement a page transition animation where one screen fades out while the next screen fades in.

Exercise 12: Design a custom slider widget using CustomPainter that allows users to select a range of values.

Advanced UI and Interaction

Exercise 13: Build a carousel slider that displays a series of images with custom animations and indicators.

Exercise 14: Develop a chat interface with a list of messages and an input field. Include the ability to send and receive messages.

Exercise 15: Create a drag-and-drop interface where items can be dragged from one list to another. Implement visual feedback during the drag operation.

Advanced State Management and Architecture

Exercise 16: Implement a multi-page form wizard using Riverpod, where each page collects a different set of data and the final step shows a summary.

Exercise 17: Create a news app using the Clean Architecture pattern. Include features like article fetching, bookmarking, and category filtering.

Exercise 18: Develop a note-taking app with local data persistence using SQLite. Use Riverpod for state management and observe changes in the database.

Local Storage and Persistence

Exercise 19: Implement a preferences/settings screen where users can toggle dark mode and save their choice using shared_preferences.

Exercise 20: Build a simple offline note app using Hive for local storage. Include the ability to add, edit, and delete notes.

Exercise 21: Create a habit tracker app that stores daily habits and their completion status using SQLite or Hive.

Testing and Debugging

Exercise 22: Write unit tests for a simple business logic function in a Flutter app, such as a function that calculates the total price of items in a cart.

Exercise 23: Implement widget tests for a form to ensure it displays validation errors correctly and submits data when valid.

Exercise 24: Create integration tests for a multi-screen app to verify navigation and data passing between screens.

Platform Integration

Exercise 25: Implement a plugin to access the device's native camera functionality, capturing and displaying a photo in your app.

Exercise 26: Use platform channels to fetch battery level information from native code and display it in a Flutter widget.

Exercise 27: Integrate a native Android and iOS share feature to share text or images from your Flutter app.

Navigation and Routing

Exercise 28: Implement deep linking in your app to navigate to a specific screen based on the URL.

Exercise 29: Create a tab-based app with custom transitions between tabs using Navigator 2.0.

Exercise 30: Implement a wizard-like onboarding flow with multiple pages, where each page provides different setup options for the user.

Advanced Features

Exercise 31: Develop a chat application that uses WebSockets for real-time messaging.

Exercise 32: Implement a map-based app where users can drop pins on the map, save locations, and display routes between points.

Exercise 33: Create a media player app that can play audio and video files from local storage or a URL, with play, pause, and seek controls.

Performance Optimization

Exercise 34: Profile an app with complex animations or heavy network requests and optimize it to reduce frame drops and improve responsiveness.

Exercise 35: Implement lazy loading for a large list of images fetched from an API, optimizing memory usage and network requests.

Exercise 36: Use Isolates to perform background computations without blocking the main thread, such as parsing a large JSON file.

Complex UI Challenges

Exercise 37: Create a custom calendar widget that allows date selection, event creation, and displays events in a grid view.

Exercise 38: Build a dashboard with complex charts and graphs using a package like fl_chart or charts_flutter.

Exercise 39: Develop a kanban board app where users can drag and drop tasks between different columns.

Security Practices

Exercise 40: Implement secure user authentication using Firebase Authentication and store sensitive data using flutter_secure_storage.

Exercise 41: Use SSL pinning to ensure secure network communication with an API.

Exercise 42: Implement OAuth2 authentication in your app to securely connect to third-party services.

Cross-Platform and Web

Exercise 43: Extend one of your Flutter apps to run on the web, optimizing for different screen sizes and adapting the UI/UX.

Exercise 44: Develop a desktop version of a Flutter app, handling platform-specific behaviors like window resizing and system dialogs.

Exercise 45: Implement platform-specific features, such as accessing the file system on desktop or integrating with browser APIs for the web.

Custom Plugin Development

Exercise 46: Create a custom Flutter plugin that interacts with a native Android or iOS API not currently covered by existing Flutter packages.

Exercise 47: Publish your custom plugin to pub.dev with proper documentation and example usage.

Exercise 48: Implement a cross-platform plugin that requires different implementations on iOS and Android (e.g., accessing a device sensor).