NEON APPS CASE STUDY

Introduction

This Case Study introduces an informational application developed in line with Neon Apps' vision. The application provides users with real-time access to The Metropolitan Museum of Art's collection data, offering a rich and engaging cultural experience.

Project Objective and Vision

In harmony with Neon Apps' mission of blending technology with meaningful content, this project aims to foster cultural awareness and provide users with an intuitive way to explore the vast collection of The MET Museum. The app helps users effortlessly discover, learn about, and engage with artworks, artists, and historical periods through a responsive, informative, and visually appealing experience.

Technical Details and Expectations

The application must work integrated with The Metropolitan Museum of Art Collection API:
♣ https://metmuseum.github.io/#search

You are expected to use the following endpoints during the process:
- Search Endpoint for Home: /public/collection/v1/search
- Object Endpoint: /public/collection/v1/objects/{objectID}
- Departments Endpoint: /public/collection/v1/departments

Notification Mechanism:

Although this API does not provide real-time alerts, you are expected to implement a custom notification mechanism that can notify users with: Daily Artwork Suggestions based on categories or user preferences (e.g., "Explore a new piece from the European Paintings collection today!").

Navigation Management - Auto Route:

The app's navigation structure should be managed using the `auto_route` package for a clean and scalable routing approach.

State Management:

Given the application and its conditions, you are expected to use the state management structure that you believe is the most appropriate. Please avoid using `setState`! Try to prioritize application performance and adhere to principles!

Management of API Requests:
It is required that RESTful API calls made over the internet within the app are carried out using the `dio` or `http` package. You are expected to handle error management.

Adaptive User Interface:
The app is expected to be built responsive, adapting to different screen sizes and devices. Use `MediaQuery` or your preferred method to dynamically scale the UI components based on screen size.

Atomic Design and SOLID Principles:
The app's UI design is expected to be constructed using the Atomic Design approach. With this method, small and reusable components are combined to form more complex structures. In addition, SOLID principles are observed in every component to ensure the maintainability and expandability of the application code.

Architectural Structures - Clean Architecture:
Please use a layered structure and a Clean Architecture based on independent modules. This structure establishes a clear boundary between business rules and data providers in the data-domain layer and the user interface in the interface layer.

Architectural Structures – Design Patterns:
Please use a design development model such as `mvvm`, `mvc`, or `mvp` to support your other structures.

User Interface and Animations:
After the opening screen, the app can display an animated transition with a classic or modern art theme. Animation can also be used between page transitions or in imprint entries.

Benefits

- ObjectBox, a fast and efficient database solution, is expected to be used for storing the user's selected search preferences, favorite artworks, and last fetched artwork details for offline use.
- Users should be able to mark artworks as favorites and revisit them later, even offline, thanks to ObjectBox.
- The app may support dynamic theming based on system settings (Dark/Light mode) for enhanced visual comfort.
- The UI should be built with accessibility in mind, supporting screen readers and scalable text.