Route Job Fair Flutter Task: Photo Gallery Application

Objective

Develop a simple Photo Gallery application that displays a list of photos fetched from an API, The application should adhere to Clean Architecture principles, use Cubit for state management, and incorporate the specified libraries. It should also support offline access via caching and display network status.

Core Requirements

Photo List Screen

- Display a list of photos fetched from a public API. You can use the pexels Api for this task.
  https://jsonplaceholder.typicode.com/photos

  curl -H "Authorization: YOUR_API_KEY" \

  GET https://api.pexels.com/v1/curated?page=2&per_page=40
- Implement a loading indicator while data is being fetched and an error message if the API call fails.
- The list should be scrollable.
- Attached Screenshots at the end of the Document

Architecture and State Management

- **Clean Architecture:** Structure the project into distinct layers (e.g., data, domain, presentation or similar, following Clean Architecture principles).
- **Cubit:** Use Cubit for state management on the Photo List Screen
- **Dependency Injection:** Implement dependency injection using injectable.
- **Repository Pattern:** Implement the Repository Pattern to abstract data sources, especially for handling both API and cached data.

API Integration and Data Handling

- **Retrofit:** Use retrofit for making API calls.
- **json_serializable:** Utilize json_serializable for efficient and safe JSON serialization/deserialization for your data models (e.g., Photo model).

Offline Support and Caching

- **Data Caching (Hive):** Implement data caching for the list of photos. When the app is offline, it should display the cached list of photos. Use the Hive library for local data storage.
- **Image Caching (cached_network_image):** Use cached_network_image to cache images displayed in both the list and details screens to improve performance and support offline viewing of previously loaded images.

Network Connectivity Indicator

- Implement a visual indicator (e.g., a simple banner, an icon, or text) that informs the user whether the application is currently online or offline. This indicator should update dynamically.

Code Quality

- **Clean Code Principles:** Write clean, readable, maintainable, and well-structured code. Adhere to common Flutter and Dart best practices.
- Proper error handling and meaningful error messages.

Theme Management (Dark/Light Mode)

- Implement support for both light and dark themes.
- The application should allow the user to switch between light and dark modes (e.g., via a settings option or a toggle button).
- The selected theme should persist across app launches.

Bonus (Optional but Highly Recommended)

- **Unit Testing:** Implement unit tests for at least one of the following:
    - Your Cubits (e.g., testing state transitions and error handling).
    - Your use cases/interactors (e.g., testing business logic).
    - Your data repositories (e.g., testing data fetching and error scenarios, including caching logic).
- **Pagination/Infinite Scrolling:** Implement basic pagination or infinite scrolling for the photo list.

Deliverables

- A functional Flutter project demonstrating the above requirements.
- A comprehensive README.md file that includes:
    - **Screenshots (Conceptual Descriptions):**
        - **Photo List Screen (Online - Light Mode):** A screenshot showing the list of photos when connected to the internet in light mode.
        - **Photo List Screen (Online - Dark Mode):** A screenshot showing the list of photos when connected to the internet in dark mode.

        - **Photo List Screen (Offline/Cached - Light Mode):** A screenshot showing the list of photos when offline, demonstrating data loaded from cache in light mode.
        - **Photo List Screen (Offline/Cached - Dark Mode):** A screenshot showing the list of photos when offline, demonstrating data loaded from cache in dark mode.
        - **Loading State:** A screenshot showing the loading indicator while data is being fetched.
        - **Error State:** A screenshot showing an error message if an API call fails.
        - **Network Status Indicator:** Screenshots demonstrating the online and offline states of the network indicator.
    - A detailed explanation of the App Architecture, illustrating how Clean Architecture principles are applied (e.g., a diagram or clear description of layers and their responsibilities, including the Repository Pattern and data flow).
    - Documentation on how to set up and run the project locally, including any necessary flutter pub get or flutter pub run build_runner commands, and any other important setup steps.

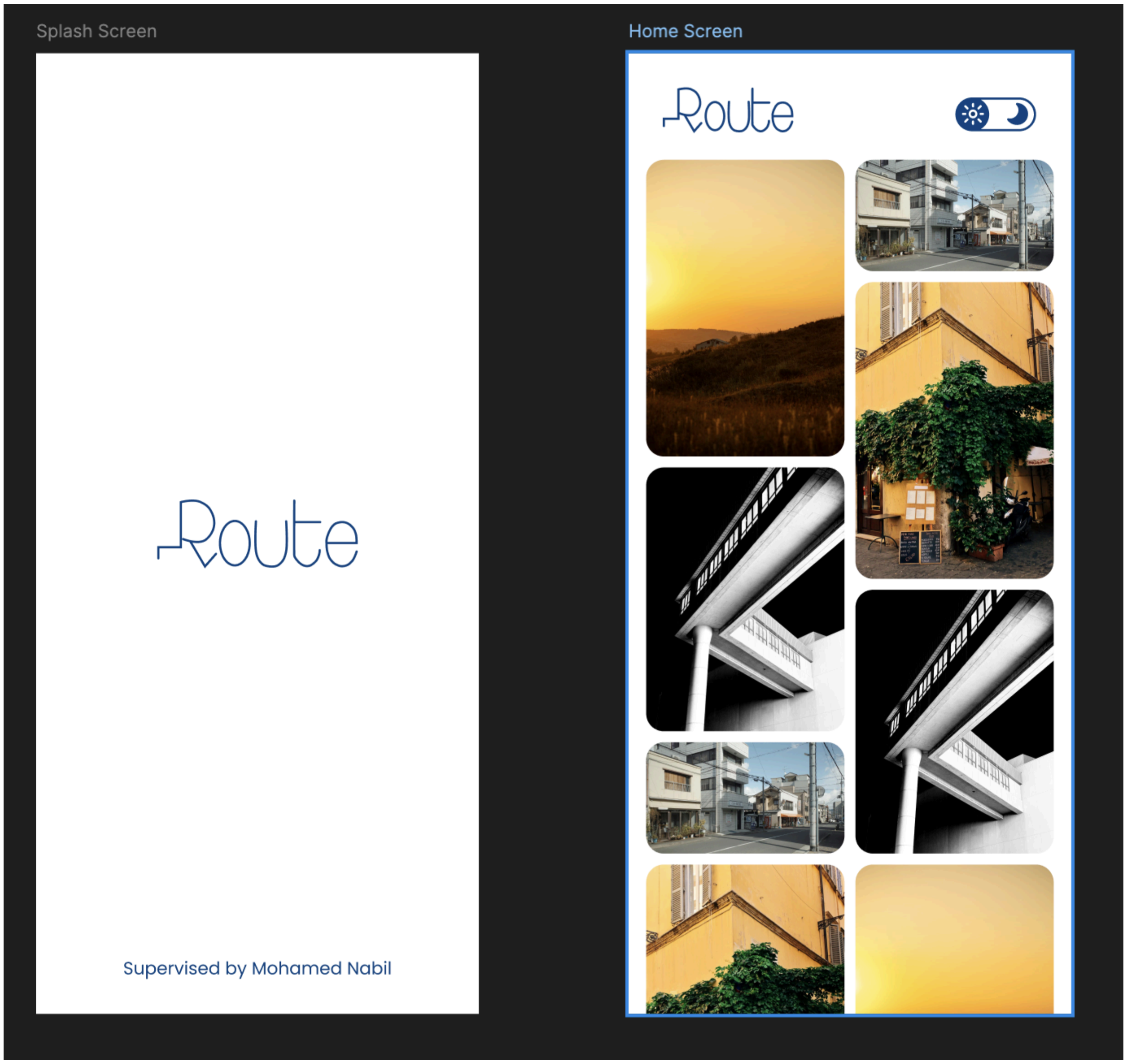
Evaluation Criteria

- Correctness and completeness of the implementation.
- Adherence to Clean Architecture principles and Repository Pattern.
- Effective use of Cubit for state management.
- Proper integration and usage of injectable, retrofit, json_serializable, Hive, and cached_network_image.
- Robust offline support and data/image caching.
- Accurate and responsive network connectivity indicator.
- **Proper implementation of dark and light theme support, including persistence.**
- Code quality, readability, and maintainability (clean code principles).
- Error handling.
- Quality and completeness of the README.md documentation, including clear screenshots and a thorough architecture explanation.
- (Bonus) Quality and coverage of unit tests.

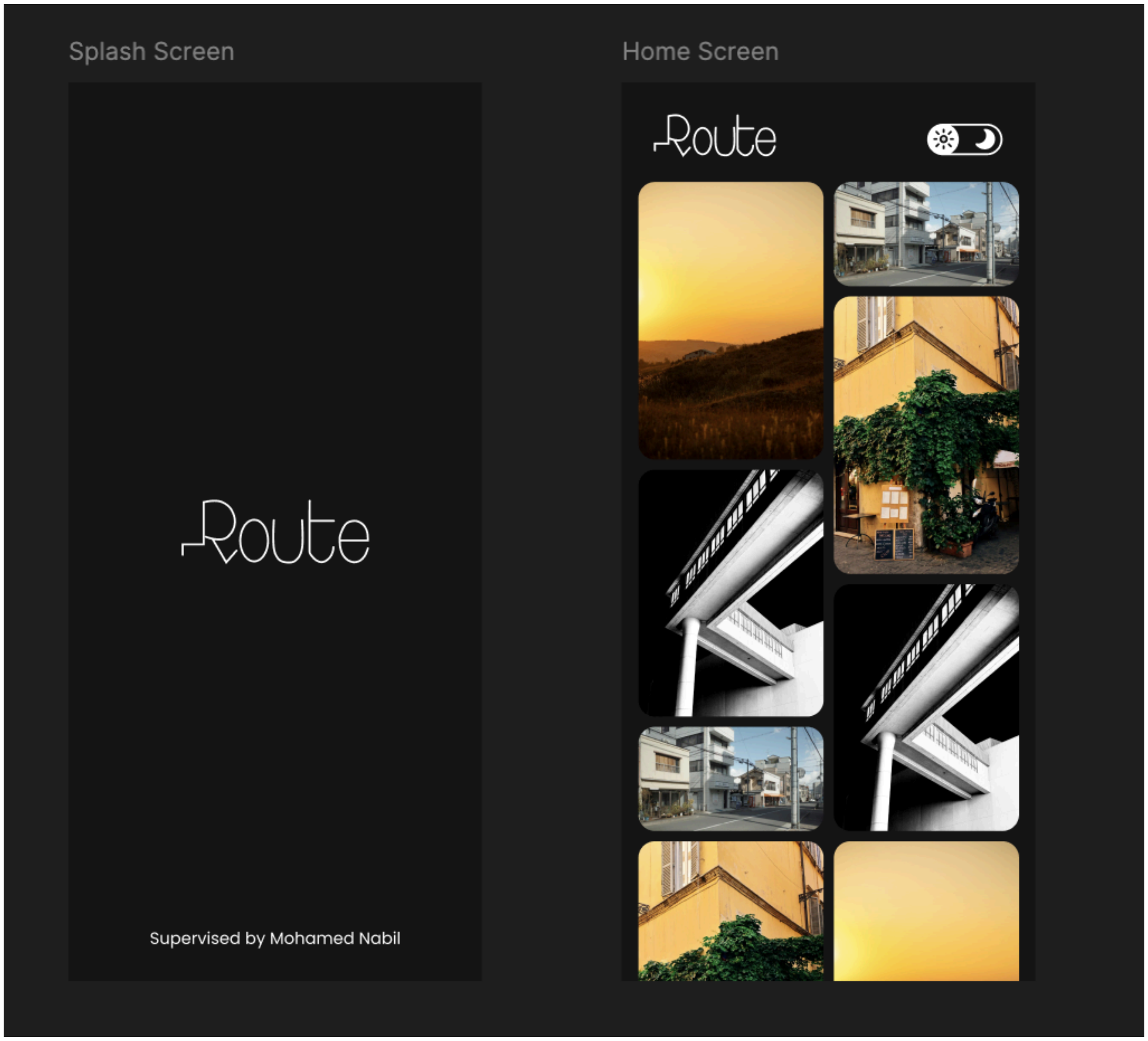
Good luck!

Screenshots

Light Mode



Dark Mode


Splash Screen
Home Screen
Route
Route
Supervised by Mohamed Nabil