Technical Requirements – Winter Travel Gallery

Challenge is accessible on Memberspot: https://pl-coding.mymemberspot.io/library/jx3b7Qik9ip5qpNI8IF2/BwAxp75QNXGaDGF2ZJEG/YO1jGpw5wX8vk5EZ8HS1/details

🎭 Scenario

This mini app represents a simple winter travel browsing experience. The user explores a list of winter destinations and opens a photo gallery for each selected location. The gallery demonstrates image loading, error handling, caching behavior, and basic navigation between screens.

🎨 Figma Mockups

> https://www.figma.com/design/XKuvJCxCybCUpKZP0LelAs/New-Year-Fresh-Start?node-id=0-1

🔤 Font - Plus Jakarta Sans

🎯 Feature Goal

Implement a multi-screen gallery app using Jetpack Compose that demonstrates image loading, basic navigation, and handling of common UI states such as loading, error, caching.

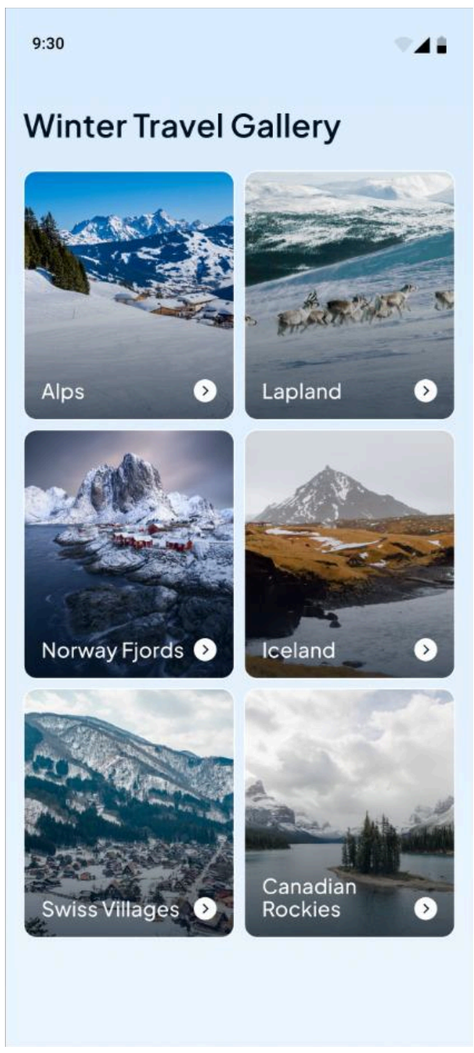
📌 Requirements

Main Screen (Destinations List)

- Serves as the entry screen of the application.
- Displays a list of winter travel destinations.

Header

- Positioned at the top of the screen.
- Displays the title *Winter Travel Gallery*.



Destinations Grid

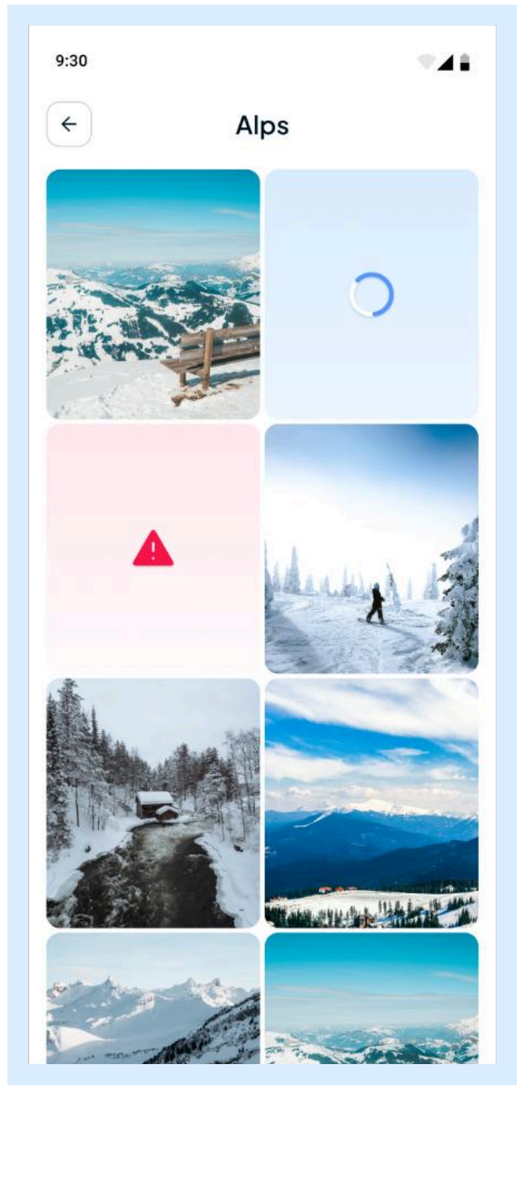
- Positioned below the Top Bar.
- Implemented as a two-column grid.
- The grid scrolls vertically if the content does not fit the screen height.

Destination Card

- Each grid item is a **clickable** destination card.
- The card **includes**:
  - a background image representing the destination;
  - the destination name displayed on top of the image;
  - a navigation indicator in the form of an arrow icon located in the bottom-right corner.
- The entire card area is **interactive**.
- The arrow icon is not a separate button and is part of the card’s clickable area.
- Tapping the card opens the **Gallery Screen** for the selected destination.
- Destinations: Alps, Lapland, Norway Fjords, Iceland, Swiss Villages, Canadian Rockies.

Gallery Screen (Destination Gallery)

- Displays a photo gallery for the selected travel destination.
- Opened after selecting a destination from the main screen.



Top Bar

- Contains a **Back** button.
- Displays the selected destination name (for example, Alps).
- Back button **behavior**:
  - navigates the user back to the main screen;
  - removes the Gallery Screen from the back stack;

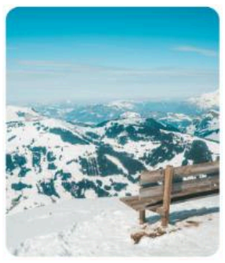
Photos Grid

- Positioned below the Top Bar.
- Implemented as a **two-column grid**.
- Displays a fixed number of photos for the selected destination.
- **Recommended number** of images: 12.
- The grid **scrolls vertically**.

Photo Item States

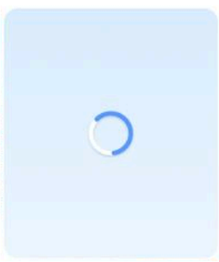
Each grid item can be in one of the following states.

Loaded Image State



- The image is successfully loaded and displayed in the grid.
- Occupies the full area of the grid item.
- Static element.

Loading State



- Displayed while the image is being loaded.
- Shows a visual loading indicator.
- Temporary state that is replaced once loading finishes.

Error State

- Displayed if the image fails to load.
- Shows an error placeholder instead of the image.
- The item remains in the grid and does not break the gallery layout.

Image Loading & Caching

> For image loading, it is recommended to use the **Coil** library. Using Coil is not a mandatory requirement and is provided as a recommended approach.

Using Coil (or an alternative solution), the implementation should demonstrate:

- asynchronous image loading;
- handling of loading and error states;
- image caching;
- faster image display when reopening the same gallery due to cache usage.

💡Image Source

- Images are provided via a predefined list of URLs.
- A link to a GitHub Gist is provided, where you can find a Kotlin file containing all prepared image URLs.
- The file includes image lists for each destination.
- Each destination contains 12 image URLs.
- Some URLs are intentionally invalid and should be used to demonstrate image loading error states.
- No network requests, pagination, or API integration are required.
- The focus is on image loading behavior, caching, and handling loading and error states.

🤔 What's Allowed?

- Standard Android/Jetpack libraries
- Any image loading solution.
- Using Coil as a recommended (but not mandatory) approach for image loading and caching.

⚠️ What's not important

- Responsiveness across every device size or orientation is not mandatory.
- Light / Dark mode.
- State persistence after app restart.
- Animations or advanced visual effects.

🔗 Useful Links for This Challenge

- Loading images
- Coil Compose documentation
- Image Caching with Coil Compose
- Navigation 3 Basics
- Navigation 3 documentation
- Stateful vs. Stateless Composables
- State Hoisting in Compose
- Managing State in Jetpack Compose (Codelab)

🏆 Submission & Rewards

- Successfully submitting this challenge via the **/submit-challenge** command on Discord grants you **100 XP**.
- Your submission must include:
  a. A **Gist link** with your implementation.
  b. A **screen recording** (max 20 seconds) showing:
  - Launching the app and displaying the main destinations screen.
  - Opening a gallery for any destination.
  - Scrolling through the gallery content.
  - Image loading behavior on first open.
  - Reopening the same gallery to demonstrate cached image loading.
  - Disabling the internet connection and opening a gallery to demonstrate the no-internet state.