Technical Requirements – First Spring Moment

Challenge is accessible on Memberspot: https://pl-coding.mymemberspot.io/library/jx3b7Qik9ip5qpNI8IF2/2BiP9k9ZAdvPhLG5wf7P/BHlOlNLGcsOapJNznkMS/details

🎭 Scenario

The app lets the user capture their very first spring moment with a photo. Sounds simple - until permissions get involved. The user may allow camera access, deny it, deny it permanently, leave the app, come back later, or try again. This challenge is about making sure the app reacts correctly to every step: nothing happens without user intent, permissions are respected, and previously captured moments are not forgotten.

👉 GitHub Repository

The implementation for this mini-challenge is provided in a GitHub repository: https://github.com/PL-Coding-GmbH/Campus-SpringValidation/tree/challenge/first-spring-moment

The repository contains multiple branches. Each branch corresponds to a **separate mini-challenge** in this series.

Clone the repository, switch to the branch for the current challenge, and work with the existing code. Your task is to focus on writing tests - the implementation itself should not be modified.

🎯 Feature Goal

The goal of this challenge is to test a permission-driven camera flow. You must verify correct permission handling, proper emission of camera-related events, and consistent restoration of captured photo state across app restarts. The focus is on behavior and state management, not camera implementation or visual design.

⚙️ App Behavior Overview

The app allows the user to capture and keep their first spring moment using the device camera. The flow is driven entirely by user actions and runtime permission state.

Initial State

- This is the default state when the app is launched and no photo has been captured yet.
- The screen displays:
  - the text *“Capture your first spring moment”*;

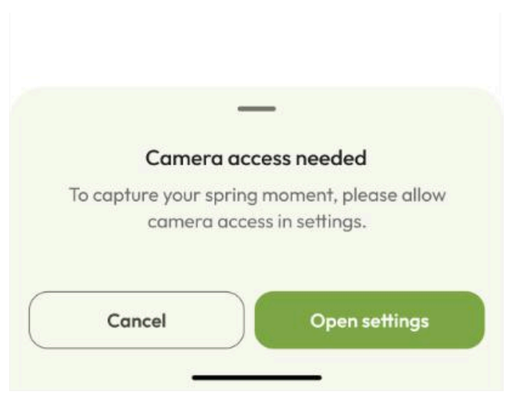
    - a primary action button *“Take spring photo”*.
- No photo preview is shown.
- The camera is not launched automatically.

Taking a Spring Photo

- Tapping “Take spring photo” initiates the camera permission flow.
- Camera permission is requested **only** after this user action.
- The app reacts based on the current permission state:
    - if permission is granted, the camera is launched;
    - if permission is denied, the camera is not launched.

Permanent Permission Denial

- If camera permission is permanently denied:
    - the system permission dialog is not shown;
    - an explanation dialog is displayed instead.
- The explanation dialog provides:
    - an **Open settings** action;
    - a **Cancel** action.
- The camera is never launched automatically in this state.



Success State

- After a photo is successfully captured:
    - the app transitions to the **Success State**;
    - a photo preview is displayed;
    - the title *“First Spring Moment”* is shown;
    - the subtitle *“The beginning of a new season”* is shown;
    - the action *“Update moment”* is available.

Updating the Moment

- Tapping *“Update moment”* triggers the permission flow again.
- Camera permission is re-checked before launching the camera.
- The camera is launched only if permission is currently granted.

State Persistence

- The captured photo is stored locally.

- On app relaunch:
  - if a photo exists, the app opens directly in the **Success State**;
  - the camera is not launched automatically.

This challenge focuses on testing permission-driven behavior, side effects, and state restoration, rather than camera implementation or visual design.

🧪 Validation Tests Requirements

This challenge includes two types of tests:

- **Instrumentation UI tests** — to verify screen rendering and dialog behavior.
- **ViewModel unit tests** — to verify permission handling, event emission, and state restoration.

Each test must use the exact test name specified below and clearly verify the expected behavior.

1️⃣ Initial State (UI Test)

Test name:

- initialState_rendersPromptAndButton

Verify:

- when the app is opened for the first time and no photo has been captured:
  - the text *“Capture your first spring moment”* is **displayed**;
  - the primary action button *“Take spring photo”* is **visible**;
  - success state elements (title, subtitle, update action) are **not displayed**;
  - no permission explanation dialog is shown.

2️⃣ Captured State (UI Test)

Test name:

- capturedState_rendersSuccessUI

Verify:

- when a photo has already been captured:
  - the title *“First Spring Moment”* is **displayed**;
  - the subtitle *“The beginning of a new season”* is **displayed**;
  - the primary action button *“Update moment”* is **visible**;
  - initial state elements (*“Capture your first spring moment”*, *“Take spring photo”*) are **not displayed**.

3️⃣ Permission Explanation Dialog (UI Test)

Test name:

- permissionDialog_rendersWhenShown

Verify:

- when camera permission is permanently denied and the explanation dialog state is active:
  - the permission explanation dialog is **displayed**;
  - the dialog title *“Camera access needed”* is **shown**;
  - the *“Open settings”* action is **visible**;
  - the *“Cancel”* action is **visible**.

4️⃣ Initial State Restoration - No Saved Photo (Unit Test)

Test name:

- initialState_whenNoSavedPhoto

Setup:

- Ensure no photo URI exists in storage.

Action:

- Create a new ViewModel instance (cold start simulation).
- Wait until initialization coroutines complete.

Verify:

- screenState is **Initial**;
- permissionState is **Unknown**;
- showPermissionExplanationDialog is **false**.

5️⃣ State Restoration - Saved Photo Exists (Unit Test)

Test name:

- stateRestoration_loadsSavedPhoto

Setup:

- Save a test photo URI to storage.

Action:

- Create a new ViewModel instance (app restart simulation).
- Wait until state restoration completes.

Verify:

- screenState is **Captured**;
- the captured state contains the **correct saved photo URI**.

6️⃣ Primary Action - Granted Permission Opens Camera (Unit Test)

Test name:

- primaryButton_withGrantedPermission_emitsCameraEvent

Setup:

- Create ViewModel.
- Sync permission state to Granted.

Action:

- Trigger OnPrimaryButtonClicked.
- Collect emitted events.

Verify:

- Exactly one **OpenCamera** event is **emitted**.

7️⃣ Primary Action - Unknown Permission Requests Permission (Unit Test)

Test name:

- primaryButton_withUnknownPermission_requestsPermission

Setup:

- Create ViewModel with default permission state (Unknown).

Action:

- Trigger OnPrimaryButtonClicked.
- Collect emitted events.

Verify:

- Exactly one **RequestCameraPermission** event is **emitted**.

8️⃣ Primary Action - Permanently Denied Permission Shows Explanation (Unit Test)

Test name:

- primaryButton_withPermanentDenial_showsDialog

Setup:

- Create ViewModel.
- Sync permission state to PermanentlyDenied.

Action:

- Trigger OnPrimaryButtonClicked.

Verify:

- showPermissionExplanationDialog becomes **true**.

9️⃣ Permission Result - Grant Opens Camera Automatically (Unit Test)

Test name:

- permissionGrantedResult_opensCamera

Setup:

- Create ViewModel.

Action:

- Trigger OnPermissionResult with Granted.
- Collect emitted events.

Verify:

- Exactly one OpenCamera event is **emitted**.

🔟 Photo Capture — Persistence and State Update (Unit Test)

Test name:

- photoCaptured_savesToStorageAndUpdatesState

Setup:

- Create ViewModel.

Action:

- Trigger OnPhotoCaptured with a test URI.
- Wait until save operation completes.

Verify:

- screenState becomes **Captured**;
- the captured state contains the **correct photo URI**;
- the URI is **persisted** in storage.

ℹ️ Notes

- This challenge focuses on **testing behavior**, not camera implementation details.
- Test method names must be used **exactly as specified**.
- The real camera does not need to be launched - camera-related side effects should be verified via emitted ViewModel events.
- ViewModel recreation (simulating app relaunch) is used to verify state restoration.
- Use appropriate mocking or faking strategies for DataStore and for observing camera- and permission-related side effects.
- Test only the scenarios described in the requirements; additional edge cases or alternative flows are not required.

🤔 What's Allowed?

- Standard Android / Jetpack libraries.
- Instrumentation tests for permission-driven flows.
- Using fakes or stubs for camera-related side effects.
- Any testing approach that verifies permission handling and state restoration logic.

⚠️ What's not important

- Testing the real camera or photo capture behavior.
- Verifying the visual appearance of system permission dialogs.
- Covering additional edge cases beyond those described in the requirements.

🔗 Useful Links for This Challenge

- Testing Kotlin coroutines on Android
- Use test doubles in Android
- Espresso-Intents
- Test your Compose layout
- Compose testing common patterns
- Testing in Jetpack Compose Codelab
- Testing APIs
- The Ultimate Guide to Android Testing

🏆 Submission & Rewards

- Successfully submitting this challenge via the **/submit-challenge** command on Discord 💡grants you **300 XP**.
- Your submission must include:
  a. A **Gist link** with your implementation.
  b. A **screenshot** of the test run results showing:
  - which tests passed,
  - which tests failed,
  - and the names of the executed tests.

> **Note:**
>
> Some tests in this challenge are **expected to fail** due to intentionally incorrect behavior in the app implementation.
>
> Your goal is to write correct tests, not to make all tests pass.

📩 How to Submit a Mini-Challenge

- In any Discord channel, type **/submit-challenge**.
- Attach your screen recording demonstrating the implementation according to the challenge requirements.
- Supported formats: MP4, MOV, AVI, MKV, WEBM, PNG, JPEG, JPG, GIF.
- The total file size must **not exceed** 50 MB.
- If additional materials are required (e.g. screenshots), attach up to 4 additional image files in the command pop-up before submitting.
- Press **Enter** to send the files.
- In the bot flow, select **Mini-Challenge**.
- Choose the month this mini-challenge belongs to (each month includes five mini-challenges).
- Select the exact **challenge name** you are submitting.
- Submit challenge.