CareerSwami · Mobile App Dev (Flutter / Android)

FREE TWO-WEEK INTERNSHIP

Project 1 — Build PocketPlan, a study planner that survives being closed

Add a task, tick it off, filter by subject. Four screens at most. The entire difficulty of this project is that a phone kills your app whenever it wants, and the user expects their list to be exactly where they left it.

2 weeks	1 project	6–8 hrs
START TO SUBMIT	ONE SUBMISSION	PER WEEK

WHAT IS IN THIS BRIEF

1. The situation — the problem you are being handed
2. The four conditions the app has to meet — the cases your work must survive
3. What you are given — your data and starting files
4. What finished looks like — the two things you hand in
5. How the two weeks run — week by week
6. Before you start — rules, tools and your starter kit
7. How the work is judged — the marking scheme
8. How to submit — one link, pasted into your dashboard

Read 1 to 4 before you do anything. They take five minutes and they are what the marking scheme is built on. Sections 5 to 8 are the how — come back to them once you know what you are solving.

1 The situation

Treat this as a real client brief. PocketPlan is fictional, and deliberately small — the marks are in state, persistence and navigation, not in features.

Students want somewhere to plan study tasks. A task has a title, a subject, the project day it is planned for, and whether it is done. Four are seeded to begin with: Algebra for Maths on Day 2, Read brief for Design on Day 3, Python exercises for Code on Day 2, and Revise notes for Maths on Day 4.

The app needs to add, edit and delete tasks, mark them done, filter by subject, and keep everything on the device. There is no login, no cloud, no payments and no notifications — all of which would be the first things a weaker brief asked for, and all of which would hide the actual problem.

The actual problem is that mobile applications do not stay alive. The operating system suspends and destroys them to reclaim memory, and the user neither knows nor cares. If your list lives only in memory, it is gone — and you will not notice during development, because you never leave the app.

WHAT YOU ARE BEING ASKED TO DO

Build PocketPlan with add, edit, delete, done, and subject filtering — where every change is written to device storage immediately, and the app comes back exactly as it was left.

2 The four conditions the app has to meet

These four are what is being marked. Your screenshots and test notes should show each one, including the awkward ones.

The app is closed and reopened

Every task, exactly as it was.

Where mobile builds fail — An in-memory list looks perfect throughout development because hot reload preserves it. Then it ships and the first real user loses everything. Write to storage on every change, not on exit — there may not be an exit you get told about.

A task is deleted by accident

A way back, or at least a confirmation.

Where mobile builds fail — An immediate irreversible delete on a tap is the single most common complaint about small task apps. Confirm, or undo. Either is fine; neither is not.

A filter is active and a task is added

Predictable behaviour the user can understand.

Where mobile builds fail — Filter by Maths, add a Design task, and it vanishes as you create it. The user concludes the add button is broken. Decide what should happen — clear the filter, show a message, or something else — and make it deliberate.

Someone using large text or a screen reader

Readable layout, labelled controls.

Where mobile builds fail — Fixed-height rows clip when the system font scales up. A delete icon with no semantic label announces as nothing useful. Both are quick to fix and almost never are.

There is a quieter trap too: the very first launch, when storage is empty. Code that assumes it will read back a list crashes on the one run every user experiences.

3 What you are given

The data shape and the required features are given. The architecture, the storage choice and the interaction decisions are yours.

The task record and the seed data

`task: id, title, subject, planned_day, done`

`Algebra · Math · Day 2`

`Read brief · Design · Day 3`

`Python exercises · Code · Day 2`

`Revise notes · Math · Day 4`

The case that decides the grade, as a test you can run

`Step 1`	Fresh install, open the app
`Expect`	Four seeded tasks, no crash on empty storage
`Step 2`	Mark "Python exercises" done; add "Past papers · Math · Day 5"
`Step 3`	Force-stop the app from the system settings
`Step 4`	Reopen
`Expect`	Five tasks; Python exercises still ticked
`Evidence`	Screenshots at steps 2 and 4, plus your storage code

Four things that must work

`Add / edit / delete` `Mark done` `Filter by subject` `Persist`

All tasks and subjects are fictional. No login, no cloud sync, no payments, and do not request notification, location, contacts or camera permissions — a planner needs none of them and asking is a real-world red flag. Project-day labels rather than calendar dates are expected and acceptable.

4 What finished looks like

At the end of two weeks someone should be able to clone your repository, build it from your notes, and have the app running on an emulator.

- **Setup notes, a navigation map and a screenshot** — the exact commands and SDK versions needed to build; a map of every screen and how the user moves between them, including how back behaves; and a screenshot proving it runs.
- **A navigable interface with reusable style notes** — every screen built and reachable with real data, plus a short note on the handful of reusable pieces you defined — spacing, text styles, the task row — and where they live, so a second developer is not inventing a sixth shade of grey.

A navigation map takes fifteen minutes and saves the common mobile mistake: a screen you can reach but cannot get back from.

5 How the two weeks run

Work through it in two parts — but you submit **once**, at the end, not once per week.

PART 1 · WEEK 1

Get it building, and map where the user can go

Set up the project so someone else could build it, decide your screens, and draw how the user moves between them — including what the back button does from each one.

Produce

- Setup notes: commands, SDK and framework versions, how to run
- A navigation map of every screen and every route between them
- A screenshot of the app running on an emulator

> Your map must show the back path from every screen. A route a user can enter and not leave is a bug, and it is easier to see on a diagram than in code.

PART 2 · WEEK 2

Build it with persistence from the first change

Implement add, edit, delete, done and filter — writing to storage on every single change. Then define the handful of reusable styles so the app looks like one app.

Produce

- A working, navigable interface over real persisted data
- Style notes: the reusable pieces you defined and where they live
- Evidence of a close-and-reopen cycle preserving state

> Show a full close and relaunch with state intact — a screenshot before and after. Handle the empty first launch explicitly; it is the one run every user has.

6 Before you start

- Write to storage on every change, not on app exit. You are not reliably told about exit.
- Handle first launch with empty storage explicitly. Do not assume a read will return a list.
- Decide what happens when a task is added while a filter excludes it, and make the behaviour visible to the user.

- Deleting needs a confirmation or an undo. Choose one and say which.
- Test with the system font scaled up. Fixed-height rows that clip are a failure, not a cosmetic issue.

Your tools

Flutter, or native Android with Kotlin. Either is fine. An emulator is sufficient; a physical device is not required. No paid tool, no Play Store account and no cloud service is required, and no ID document is ever required.

CS_mobile-app-dev-flutter-android_Starter_Kit.zip — tasks.csv, a navigation map template, a style notes sheet, a lifecycle test checklist, an evidence log, and a file index.
https://careerswami.com/wp-content/uploads/careerswami-projects/CS_mobile-app-dev-flutter-android_Starter_Kit.zip

7 How the work is judged

WHAT IS ASSESSED	WEIGHT
Problem and requirements — did you understand what you were solving	20%
The core artefact — the thing you actually made	40%
Validation and evidence — how you know it works	25%
Clarity and reproducibility — can someone else open and follow it	15%

Strong	Complete, correct, evidence-backed and easy to inspect.
Developing	A useful partial result, with the gaps explained.
Needs revision	Core deliverable missing, claims unverified, or the work cannot be opened.

Credit reflects demonstrated work, not visual polish. A small, honest result beats a large, unsupported one.

What counts as evidence

expected result → actual result → the screenshot, log output or storage dump → conclusion. Then say what the evidence does not establish.

Why this one is worth doing. Mobile interviews converge on lifecycle. The app was killed in the background — what happened to the user's data? Candidates who have only built on an emulator they never closed do not have an answer. A repository where persistence is wired in from the first commit, with a force-stop test documented, answers it before the question is asked.

8 How to submit

One link, pasted into your dashboard

1. Put your work somewhere with a link — Google Drive, Figma, GitHub, Notion, Loom. Whatever fits the work.
2. **Set it so anyone with the link can view.** A locked link is the single most common reason a submission comes back.
3. Include a short README or cover page saying what to open first, and where each deliverable lives.
4. Open your dashboard, paste the link into the submit box, and add a line or two on anything you want the reviewer to know.

Name files `CS_mobile-app-dev-flutter-android_YourName_v1`. Keep your own copy. A screenshot on its own is not enough for anything interactive. A video is optional — annotated screenshots or narrated slides are fine.

Submitted and heard nothing, or the link came back? Email **support@careerswami.com** with the email address you enrolled with.

Work honestly. Use fictional data and your own work. Credit anything you reused, and disclose AI assistance — including what you checked yourself. Never invent interviews, users, tests or results. Nothing here requires you to spend money, contact real organisations, or deploy anything publicly. Never upload identity documents, real customer records or credentials.

Stuck, confused, or something is broken? Write to us.

`support@careerswami.com`

Or WhatsApp **+91 7383138418**. We answer on working days. Use it for anything at all — a part of this brief that does not make sense, a starter file that will not open, a dashboard link that has stopped working, a deadline you need moved, or a submission you are not sure about.

To get a useful answer fast, spend twenty minutes narrowing the problem down first, then tell us four things: what you expected, what actually happened, what you already tried, and your one specific question. Include your track name and the email you enrolled with.

Asking is not penalised. Submitting partial work with a clearly stated blocker always scores better than a finished answer you copied, and better than a silent no-show.

Optional stretch

Optional, and only once the main work is done — cap it at an hour. Add a grouped-by-day view, or an undo snackbar on delete. Label it clearly as a stretch.