CareerSwami · Frontend Development (React)

FREE TWO-WEEK INTERNSHIP

Project 1 — Build an opportunity dashboard that works on a phone and a screen reader

Six opportunities, a search box, a filter, and a save button that has to remember what you saved after you close the tab. Small enough to finish in two weeks. Large enough to expose every habit you have.

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 build 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. The dataset is small on purpose — the marks are in how you structure it, not how much of it there is.

A student opportunities board has six listings: a community poster, a library cleanup, a club website, event operations, an accessibility audit and a newsletter. Each has a category, a mode of remote or on-campus, and a number of days it runs.

Students need to find things fast. That means searching by text, filtering by category, opening one to see detail, and saving the ones they want to come back to. The saved list has to survive closing the browser.

Six items fits on one screen, so none of this is about performance. It is about whether your components have clear responsibilities, whether the thing works at three hundred and twenty pixels wide, and whether someone using a keyboard and a screen reader can do everything a mouse user can.

WHAT YOU ARE BEING ASKED TO DO

Build a responsive opportunity dashboard with search, category filtering, a detail view, and saved items that persist — and structure it so a second developer could add a seventh field without touching five files.

2 The four conditions the build has to meet

A dashboard with six rows is easy to make work for you, on your laptop, with a mouse. These four are the conditions that separate a demo from a build.

A 320px phone screen

Everything reachable, nothing cut off, no sideways scrolling.

Where builds like this fail — Fixed widths, a filter bar laid out as a single row, and a card grid with a hard minimum column width. It looks fine until you narrow the window, which is usually the day before the deadline.

Someone using only a keyboard

Tab to every control, activate it, and see where they are.

Where builds like this fail — A clickable div is invisible to the tab key. Removing the focus outline because it looked untidy leaves the user with no idea where they are. Both are easy to avoid and almost universal in first builds.

Someone using a screen reader

Controls that announce what they are and what they do.

Where builds like this fail — A save button that is just a heart icon announces as "button". The search box with only a placeholder has no label. Filter results that update silently never get announced at all — the user has no idea anything happened.

The next developer

To add a field or a filter without reading the whole app.

Where builds like this fail — Filtering logic spread across three components, state lifted to the wrong place, and every card knowing how the save system works. The app still runs; it has just become expensive to change, which is the real cost nobody sees in a demo.

Persistence has one trap worth naming: reading from storage on first render before anything has been written gives you null, and a component that does not handle that renders nothing and reports no error.

3 What you are given

Six rows of data and four required features. The structure, the state decisions and the accessibility work are yours, and they are where the marks are.

The six opportunities — id, title, category, mode, days

`1 Community poster · Design · Remote · 7`
`2 Library cleanup · Data · Campus · 14`
`3 Club website · Tech · Remote · 21`
`4 Event operations · Business · Campus · 7`
`5 Accessibility audit · Tech · Remote · 10`
`6 Newsletter · Design · Remote · 7`

One component note, so the level of detail is clear

`Component`	OpportunityCard
`Receives`	one opportunity object, isSaved, onToggleSave
`Owns`	nothing — it holds no state of its own
`Does not know`	how saving is stored, or what else is in the list
`Renders`	an <article> with a heading, meta, and a real <button>
`Accessible name of save`	"Save Library cleanup" — not "Save" alone
`Why it matters`	A screen reader user hearing six identical "Save" buttons has no idea which is which

Four features that must work

`Search` `Filter` `Details` `Save`

All listings are fictional. No login, no backend and no public hosting are required. Keep secrets out of the source. Where the brief leaves something open — whether filters combine with search, what an empty result shows — decide, write it in the README, and be consistent.

4 What finished looks like

At the end of two weeks someone should be able to clone your repository, run it from your README, and use the whole thing without a mouse.

- **A setup README, a component tree and wireframes** — exact commands and versions to get it running; a diagram naming every component, what it owns and what it receives; and simple wireframes for the list view and the detail view at both phone and desktop width.
- **A working list and detail interface with component notes** — search, category filtering, detail view and saved items with local persistence, plus a short note per component saying what it is responsible for

and what it deliberately does not know about.

The component tree is not paperwork. Draw it before you build and you will discover where state belongs; draw it afterwards and you will be documenting a mistake.

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

Decide the structure before you write components

Sketch both views at both widths, draw the component tree, and decide where each piece of state lives. Then get the project running with a README someone else could follow.

Produce

- Wireframes for list and detail, at phone and desktop width
- A component tree showing ownership and data flow
- A setup README with exact commands and versions

> Your tree must say where the search term, the active filter and the saved list each live, and why. If all three are in the top component by default, say whether that was a decision or an accident.

PART 2 · WEEK 2

Build it, then use it without a mouse

Implement search, filter, detail and save with persistence. Then put the mouse down and tab through the entire app. Fix what you find.

Produce

- A working interface: search, filter, detail, saved items that persist
- Component notes: what each owns, what it does not know about
- A record of your keyboard pass and what it turned up

> Every interactive element must be a real button or link, reachable by keyboard, with a visible focus state and an accessible name. An icon-only control needs a label even though nothing appears on screen.

6 Before you start

- Search and filter must combine sensibly. Decide whether they are AND or OR and state it; both are defensible, silence is not.
- Handle the empty state. Six items means a search for "xyz" returns nothing, and a blank screen is a bug.

- Reading from local storage on first render can return null. Handle it, or the app renders nothing and tells you nothing.
- Do not remove focus outlines. Restyle them if you want, but a keyboard user must always be able to see where they are.
- A div with an onClick is not a button. Use real buttons and links — you get keyboard support and announcement for free.

Your tools

React with Vite is the straightforward choice. Create React App, Next.js or plain JavaScript modules are all acceptable if you prefer. No UI library is required and none is penalised. No paid tool is required, and no ID document is ever required.

CS_frontend-development-react_Starter_Kit.zip — opportunities.csv, a component tree template, wireframe frames, a keyboard-pass checklist, an evidence log, and a file index.
https://careerswami.com/wp-content/uploads/careerswami-projects/CS_frontend-development-react_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, keyboard-pass note or console output → conclusion. Then say what the evidence does not establish.

Why this one is worth doing. React is not the scarce skill; plenty of people can wire up a filter. What is scarce is a developer who draws the component tree first, puts state where it belongs, and has actually tabbed through their own app. Those three habits show up in a code review in about ninety seconds, and they are the ones that get you past it.

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_frontend-development-react_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 URL-synced filters so a search can be shared as a link, or an announced live region for result counts. Label it clearly as a stretch.