CareerSwami · Full Stack Web Development

FREE TWO-WEEK INTERNSHIP

Project 1 — Build StudySwap, where a session cannot be overbooked

Three study sessions, capacities of three, two and four. Anyone can list them, book one, or cancel. The whole project turns on one sentence: the fourth person must never get into a session with three places.

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 cases your code has to survive — 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. StudySwap is fictional. The constraint at its centre — do not let two requests consume the same last place — is the oldest real problem in web development.

Students want to book peer study sessions. There are three to start with: Algebra on Monday at sixteen hundred with three places, Resume review on Tuesday at seventeen hundred with two, and Python on Wednesday at sixteen hundred with four.

The app has to do four things: list the sessions, show one in detail, book a place, and cancel a booking. People are identified by a fictional alias — there are no accounts, no passwords and no payments in this project.

The part that is easy to get wrong is capacity. A session with three places must never end up with four bookings, however the requests arrive. And when the server restarts, every booking must still be there — a list held in memory passes a demo and fails the first real test.

WHAT YOU ARE BEING ASKED TO DO

Build a working StudySwap that lists, shows, books and cancels — and that refuses to overbook, refuses invalid bookings, and still has its data after a restart.

2 The four cases your code has to survive

Nobody is scoring your styling. These four cases are the project, and your request and response examples should show each of them being handled.

The fourth booking on a three-place session

Must be rejected, clearly, with a reason.

Where naive code breaks — The naive version counts bookings, finds three, and inserts a fourth because the check and the insert are two separate steps. Enforce it where the data lives — a constraint or a single atomic operation — not with an if-statement that hopes.

A booking for a session that does not exist

Must be rejected, not stored.

Where naive code breaks — An unchecked session id gets written anyway, and you now have a booking pointing at nothing. Every list query that joins the two tables either crashes or silently drops rows.

The same person booking twice

Decide the rule, then enforce it.

Where naive code breaks — Nothing in the brief tells you whether this is allowed. Both answers are acceptable; what is not acceptable is having no rule, because then the behaviour is whatever the code happens to do.

The server restarts

Every booking is still there.

Where naive code breaks — An in-memory array loses everything and the demo still looks fine, because you never restart during a demo. Use a file or a database from the first commit — retrofitting persistence at the end is a rewrite.

Cancelling has to return the place to the pool, or capacity quietly leaks until nothing can be booked. Test the full cycle — book, cancel, book again — not just the happy path.

3 What you are given

The data and the four flows are given. The stack, the schema and the way you enforce capacity are yours — and the enforcement is what is being marked.

The seed data and the shape of it

`S1 Algebra · Mon 16:00 · capacity 3` `S2 Resume review · Tue 17:00 · capacity 2`

`S3 Python · Wed 16:00 · capacity 4` `session: id, title, start, capacity`

`booking: id, sessionId, alias, status`

The case that decides the grade, as a worked example

`Given`	S1 Algebra, capacity 3, with 3 confirmed bookings
`Request`	POST /bookings { sessionId: "S1", alias: "owl42" }
`Expected status`	409 Conflict
`Expected body`	{ "error": "session_full", "message": "This session has no places left." }
`Must remain true`	SELECT count(*) on S1 bookings still returns 3
`Then`	Cancel one booking; the same request now succeeds
`Prove`	Capture all three responses, not just the last one

Four flows that must work

`List` `Details` `Book` `Cancel`

Use fictional aliases only — no real names, emails or student numbers. No login, no payments and no public hosting are required or wanted. Keep secrets out of the source. Where the brief leaves a rule open, such as double booking, choose one, write it in the README, and enforce it consistently.

4 What finished looks like

At the end of two weeks someone should be able to clone your repository, follow your README, and have the thing running in under ten minutes.

- **A setup README, an entity diagram and a working health check** — the exact commands to install, seed and run, including versions; a diagram showing sessions and bookings and the relationship between them; and a screenshot or terminal capture proving the app actually started on a clean machine.
- **An API contract, a schema and seed, and worked request and response examples** — every endpoint with its method, path, inputs, success response and each error it can return; the schema that enforces

capacity rather than merely recording it; and real captured examples including the rejected fourth booking.

A README that says "install dependencies and run" is not a setup guide. Assume the reader has your language installed and nothing else, and that they will not ask you a question.

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

Model the data and make it start on a clean machine

Design the two tables and the relationship between them, decide where capacity is enforced, and get a minimal app running that someone else could start from your README alone.

Produce

- An entity diagram: sessions, bookings, and the relationship
- A setup README with exact commands and version numbers
- A running app with a health endpoint, and evidence it started

> Your diagram must say where capacity is enforced and how. "In the booking handler" is a valid answer only if you also say what stops two simultaneous requests both passing the check.

PART 2 · WEEK 2

Write the contract, then make the code honour it

Define every endpoint before you implement it — including what each one returns when it refuses. Then build it, seed the three sessions, and capture real examples of both success and refusal.

Produce

- An API contract: method, path, inputs, success and every error case
- A schema and a seed script that creates the three sessions
- Captured request and response examples, including the rejected fourth booking

> Your examples must include at least one refusal with a readable reason. An error response of 500 with an empty body is a defect, not an error case.

6 Before you start

- Persist from the first commit. SQLite, Postgres or a JSON file written to disk — anything but an in-memory list.

- Enforce capacity as close to the data as you can. A unique constraint, a transaction, or a conditional insert beats a check followed by an insert.
- Cancelling must free the place. Decide whether you delete the row or set a status, and be consistent everywhere you count.
- Every error gets a status code and a readable message. Returning 200 with an error inside the body is a defect.
- Your README is a graded deliverable. Test it by following it yourself from a fresh folder before you submit.

Your tools

Any stack you are comfortable with — Node with Express, Python with Flask or FastAPI, PHP, Go. SQLite is ideal for this and needs no server. No paid tool, no cloud account and no public hosting is required, and no ID document is ever required.

CS_full-stack-web-development_Starter_Kit.zip — sessions.csv, an API contract template, an entity diagram template, a README skeleton, an evidence log, and a file index.
https://careerswami.com/wp-content/uploads/careerswami-projects/CS_full-stack-web-development_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 captured request and response, or a terminal screenshot → conclusion. Then say what the evidence does not establish.

Why this one is worth doing. Every backend interview eventually arrives at this question: two people click book at the same moment on the last place — what happens? Most candidates answer with an if-statement. The ones who get offers talk about where the constraint lives, and can point at a repository where it is enforced in the schema. That is exactly what this project puts in your portfolio.

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_full-stack-web-development_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 waitlist that promotes automatically on cancellation, or write a script that fires concurrent bookings and proves the constraint holds. Label it clearly as a stretch.
