CareerSwami · Python & Backend Engineering

FREE TWO-WEEK INTERNSHIP

Project 1 — Build a lending API that knows what is overdue

Three books, six copies between them, a seven-day loan. Borrow, return, and report what is late. The hard part is not the endpoints — it is that "available" and "overdue" are calculated truths, and you must never store them.

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 rules your service has to get right — 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 library is fictional. The modelling decision at its centre — derive state rather than store it — is the one that separates an API that stays correct from one that slowly drifts.

A small lending library has three titles: Python Basics with two copies, Design Notes with one, and Data Stories with three. Six physical copies in total.

Borrowers are fictional aliases. A loan records which book, which alias, the day it went out, the day it is due, and whether it has come back. The loan period is seven project days — this service counts in day numbers, not calendar dates, which keeps the arithmetic and the tests clean.

Four things are needed: list and search the books, borrow one, return one, and produce an overdue report. Overdue has an exact definition you are given, and the temptation to store a flag called *is_overdue* is the trap this brief is built around.

WHAT YOU ARE BEING ASKED TO DO

Build a local lending API with list and search, borrow, return and an overdue report — where availability and overdue status are always derived from the loan records, never stored alongside them.

2 The four rules your service has to get right

These four are the whole project. Your worked request and response examples should show each of them, including the refusals.

Borrowing the second copy of a one-copy book

Must be refused while the first is out.

Where it goes wrong — Availability is copies minus active loans — a calculation, not a column. Store a boolean called available and it will eventually disagree with the loans table, and you will have no way to tell which one is lying.

"Overdue" as a stored flag

current_day > due_day and not returned.

Where it goes wrong — A stored flag is correct only at the instant it is written. Tomorrow it is stale. Compute it on read, every time, from the day you are asked about — then your overdue report is right at any point in time, including in a test.

Returning a loan twice

The second return must be refused, not silently ignored.

Where it goes wrong — An unguarded return sets returned to true again, looks successful, and your availability count is now permanently wrong by one. Check the current state before you change it and say no.

Borrowing a book that does not exist

Refused with a clear 404, nothing written.

Where it goes wrong — An unvalidated book id gets stored anyway and you have an orphan loan. Every join after that either errors or quietly loses rows, and the bug surfaces weeks later in the report.

Because overdue depends on the current day, your API needs a way to ask "as of day N" — otherwise the overdue report is untestable, and an untestable rule is one you cannot prove you got right.

3 What you are given

The catalogue and the rules are given. The schema, the queries and the error design are yours, and that is where the marks are.

The catalogue and the loan record

`B1 Python Basics · 2 copies` `B2 Design Notes · 1 copy` `B3 Data Stories · 3 copies`

`loan: id, book_id, borrower_alias, checkout_day, due_day, returned`

`loan period: 7 project days`

The overdue rule, worked, so there is no ambiguity

`Loan`	L7 · B2 Design Notes · alias "finch"
`checkout_day`	3
`due_day`	3 + 7 = 10
`returned`	false
`Asked on day 10`	10 > 10 is false → not overdue
`Asked on day 11`	11 > 10 is true, and not returned → overdue
`Never store this`	Compute it from current_day on every read

Four operations that must work

`List / search` `Borrow` `Return` `Overdue report`

Use fictional borrower aliases only — no real names, emails or student numbers. No public hosting is required or wanted. Keep secrets out of the source and out of version control. Where the brief leaves a rule open, such as whether one alias may hold two copies of the same title, decide, document it in the README, and enforce it.

4 What finished looks like

At the end of two weeks someone should be able to clone your repository, follow the README, and borrow a book within ten minutes.

- **A domain diagram, a route plan and a setup README** — books, copies and loans with the relationships between them and a note on which values are derived; every endpoint planned before it is written; and a setup guide with exact commands and versions that works on a clean machine.
- **A schema and seed, the data access code, and worked request and response examples** — the three books and six copies created by a script rather than by hand, the queries that derive availability and

overdue status, and captured examples including every refusal case.

A README that assumes your virtual environment and your Python version is not a setup guide. Pin the version, list the commands, and test it from an empty folder yourself.

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 domain and plan every route

Decide what is stored and what is computed, draw the relationships, and write down every endpoint — including its failure responses — before implementing any of them.

Produce

- A domain diagram: books, copies, loans, and what is derived
- A route plan: method, path, inputs, success and each error case
- A setup README with exact commands and pinned versions

> Your diagram must mark availability and overdue explicitly as derived. If either appears as a stored column, the design is wrong and the rest of the project will fight you.

PART 2 · WEEK 2

Build it, seed it, and capture the refusals

Write the schema and a seed script, implement the data access, and capture real request and response pairs — the successes and, more importantly, every way the service says no.

Produce

- A schema and a seed script creating three books and six copies
- Data access code deriving availability and overdue status
- Captured examples for borrow, return, search and overdue, with refusals

> The overdue endpoint must accept the day to evaluate against, so it can be tested. Every refusal needs a correct status code and a readable message — 500 with an empty body is a defect, not an error case.

6 Before you start

- Count in project days, not calendar dates. It keeps the arithmetic obvious and makes the overdue rule testable.
- Overdue means current_day > due_day and not returned. Due day itself is not overdue.

- Availability is copies minus active loans, computed on read. No stored availability column.
- The overdue endpoint must take the evaluation day as a parameter, or you cannot test it without waiting a week.
- Validate input at the edge. A borrow request with a missing alias or an unknown book id must be refused before anything is written.

Your tools

FastAPI or Flask, with SQLite. Both are free, neither needs a server, and FastAPI gives you request validation and documentation without extra work. No paid tool, no cloud account and no public hosting is required, and no ID document is ever required.

CS_python-backend-engineering_Starter_Kit.zip — books.csv, a route plan template, a domain diagram template, a README skeleton, an evidence log, and a file index.
https://careerswami.com/wp-content/uploads/careerswami-projects/CS_python-backend-engineering_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 query output → conclusion. Then say what the evidence does not establish.

Why this one is worth doing. Backend interviews are full of people who can write an endpoint. The question that sorts them is this one: where does "overdue" live? The candidate who stores a flag has given away that they have not maintained a system long enough to watch stored state go stale. The candidate who derives it, and can point at the query, has.

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_python-backend-engineering_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 reservation queue for a title with no copies free, or a loan-history endpoint per alias. Label it clearly as a stretch.