# Assignment

## User Story

You are on a team building a small personal finance tool.

**As a user, I can record and review my personal expenses so I can understand where my money is going.**

This tool is assumed to be used in real-world conditions (unreliable networks, browser refreshes, retries).

Acceptance criteria (for this exercise):

1. User can create a new expense entry with amount, category, description, and date.
2. User can view a list of expenses.
3. User can filter expenses by category.
4. User can sort expenses by date (newest first).
5. User can see a simple total of expenses for the current list (e.g., "Total: ₹X").

Aim for **production-like** quality while keeping the feature set small.

---

## The Assignment

You are responsible for a minimal **full-stack Expense Tracker**: a backend API and a simple frontend UI.

Assume this is something you might extend and maintain over time, not a throwaway prototype.

---

## Backend (Required)

Implement a small API that supports:

`POST /expenses`

- Create a new expense.
- Request body should include: amount, category, description, date.

- The API should behave correctly even if the client retries the same request due to network issues or page reloads.

### `GET /expenses`

- Return a list of expenses.
- Support optional query parameters:
  - `category` (filter by category)
  - `sort=date_desc` (sort by date, newest first)

**Data model (minimum):**

- `id`
- `amount` (use an appropriate type for real money)
- `category`
- `description`
- `date`
- `created_at`

You can choose any reasonable persistence mechanism (e.g., in-memory store, JSON file, SQLite, relational DB, no-SQL DB).

Explain your choice briefly in the README.

---

## Frontend (Required)

Implement a simple web UI that talks to your API:

- A form to add a new expense (amount, category, description, date).
- A list/table of existing expenses.
- Controls to:
  - Filter by category.
  - Sort by date (newest first).
- Display the **total amount** of the currently visible expenses (after filters/sorting).

Assume users may:

- Click submit multiple times
- Refresh the page after submitting
- Experience slow or failed API responses

Keep styling simple; focus on correctness and clarity.

---

## Nice to Have

Only attempt these if you have time left after finishing the core:

- Basic validation (e.g., prevent negative amounts, require a date).
- A summary view (e.g., total per category).
- A couple of small automated tests (unit or integration).
- Basic error and loading states in the UI.

Prioritize what you think adds the most real value.

---

## Constraints & Expectations

- You may use any frameworks, libraries, or tools you would normally use on the job.
- You are welcome to use AI-assisted tools (e.g., Copilot, LLMs).

Include a short note in your README explaining:

- Key design decisions
- Trade-offs you made because of the timebox
- Anything you intentionally did **not** do

---

## What Will Be Evaluated

We are not looking for feature completeness.

We will evaluate:

- Whether your system behaves correctly under realistic conditions.
- How you think about data correctness, money handling, and edge cases.
- Code clarity and structure.

- Your judgment in choosing what *matters* vs what doesn't.

---

## How to Submit

- Share a link to a repository containing your code and README.
- Share a link to access the live and deployed application
- Repo structure is up to you (monorepo or separate folders).