# Lilly Technical Challenge - Documentation

## Overview

My main goal for this challenge was to get both the backend and frontend working smoothly together and present the data in a clean, user-friendly way. I focused on understanding how the backend API behaved, then built the frontend around it in a way that handles messy data and allows users to add new medicines easily.

I kept everything simple, practical, and focused on the requirements instead of over-engineering.

## What I built

**1. Fetching and Displaying Medicines**

I created a script that loads all medicines from the backend using `/medicines`.

The frontend displays them in a clean table.

If anything is missing or invalid (like a missing name or price), the UI shows a fallback such as “Unknown name” or “N/A” instead of breaking.

**2. Handling Bad or Missing Data**

The dataset intentionally had gaps.

To avoid errors, I added checks before reading any field so the UI could still render the row even if the data wasn't perfect.

**3. Adding New Medicines (User-Friendly Form)**

The `/create` endpoint expects form data, so the frontend sends `FormData` instead of JSON.

The form has basic validation and updates the table automatically after a successful submission.

**4. UI/UX Improvements**

I redesigned the frontend to look cleaner and more structured:

- card layout
- better spacing
- clearer headings
- loading/error messages
- simple responsive structure

Nothing fancy, just straightforward improvements that make the site easier to use.

## Problems I Faced (and How I Fixed Them)

**1. Backend Not Starting Due to Wrong File Structure**

The provided start script kept failing because it expected `requirements.txt` in the project root, but it was inside the `backend` folder.

**Fix:**

I ignored the script, created a virtual environment manually inside `/backend`, installed dependencies, and ran `main.py` directly.

**2. "Failed to Load Medicines" on the Frontend**

This happened because FastAPI wasn't actually running (due to the issue above).

Once the backend finally ran on port 8000, the frontend immediately loaded the data correctly.

**3. Git Remote Issues**

At first GitHub rejected pushes because the remotes weren't set correctly.

**Fix:**

I renamed the upstream and added my repo as the new origin.

**4. Form Not Working Initially**

Sending JSON didn't work because the backend expects `Form(...)`.

Switching to `FormData` fixed the issue instantly.

The challenge was straightforward once the environment issues were cleared up.

## Final Thoughts

The main takeaway was making sure the frontend gracefully handles whatever the backend sends, even if it's imperfect. I kept the code clean, defensive, and practical. Overall, I met all main objectives and ended up with a simple, functional mini-app that's easy to explain and demo.

***Faizan Naveed***