# Take-Home Assignment: Reverse Engineering API Requests

## Objective

Your task is to build a small web application that allows users to upload a `.har` (HTTP Archive) file, describe the API they want to reverse-engineer, and receive a `curl` command to replicate the relevant API request. The backend will leverage an LLM to identify the most relevant request within the `.har` file while keeping token usage efficient. You can assume that the API call that should be selected is not returning HTML.

## Requirements

### Core Features

1. **File Upload, Inspector & Input**
    - Provide a UI where users can upload a `.har` file.
    - Users should be able to inspect the individual requests of the uploaded files.
    - Users can enter a description of the API they want to reverse-engineer.
2. **Backend Processing**
    - Parse the `.har` file to extract relevant API requests.
    - Use an LLM to identify the best-matching request.
    - Return a corresponding `curl` command to the frontend.
3. **Frontend Display**
    - Show the extracted `curl` command in a user-friendly format.
    - Allow users to execute the API request and see the response directly in the UI.

### Tech Stack

- **Frontend:** Typescript, Next.js, Tailwind CSS, shadcn/ui
- **Backend:** Typescript, NestJs
- **LLM Models:** Any OpenAi model

# Example Use Cases

## 1. Weather API

- **Website:** SFGate Weather
- **User Input:** "Return the API that fetches the weather of San Francisco."
- **Input file:** www.sfgate.har
- **Expected Output:** sfgate_output.txt

## 2. Recipe API

- **Website:** RecipeScal
- **User Input:** "Can you reverse engineer the API that gives me recipes for a given portion and calorie count?"
- **Input file:** recipescal.com.har
- **Expected Output:** recipescal_output.txt

## 3. Jokes API

- **Website:** JokeApi
- **User Input:** "Can you give me a `curl` command to get 5 jokes via API?"
- **Input file:**
  - **Easy:** jokes.har
  - **Difficult:** jokes.large.har
- **Expected Output:** jokes.txt

# Evaluation Criteria

Your solution will be assessed based on:

1. **Functionality:** Does the app correctly extract and return the relevant API request?
2. **Code Quality:** Is the code clean, well-structured, and maintainable?
3. **Efficiency:** Does the implementation have efficient token usage when querying the LLM? What trade offs are being accepted for token efficiency?
4. **User Experience:** Is the UI intuitive and easy to use?
5. **Security:** Does the implementation adhere to security best practices?
6. **Bonus Features:** Any additional enhancements will be considered a plus.

## Submission

- Share a GitHub repository with a README explaining how to run the project.
- Provide a short demo video (optional but appreciated).

Good luck!