PromptFolio: Build Your CV with Prompts

Project Overview

PromptFolio is a modern web application designed to simplify and automate the creation of a professional, aesthetically pleasing résumé. The core problem it solves is the difficulty many individuals face in crafting well-structured and well-written CVs. The application replaces this challenge with an intuitive, prompt-based system.

Users interact with a clean web interface to input their personal and professional details. By writing simple, natural language prompts for each section of their CV (like "Education" or "Work Experience"), the application leverages the power of the Google Gemini Large Language Model (LLM) to generate correctly formatted LaTeX code. This code is then compiled into a final, polished CV document, ready for the user.

Technology Stack

- **Frontend:** React.js (with Vite for a fast development environment)
- **Backend API:** Python with the FastAPI framework
- **AI / Language Model:** Google Gemini (via the google-generativeai library)
- **Data Handling:** Pandas library (for managing CV section data)
- **Styling:** Plain CSS with a component-based organization

High-Level Architecture

The project follows a classic and robust client-server architecture:

Client (Frontend): A React single-page application that runs in the user's browser. It is responsible for the entire user interface, capturing user input, and communicating with the backend.

Server (Backend): A Python application powered by FastAPI. It serves as the brain of the operation, handling API requests from the frontend, processing data, interacting with the Gemini LLM, and managing the file system.

Detailed Folder Structure and Purpose

Here is a breakdown of each folder and its role within the project:

📁 API/

Purpose: This folder contains the web server component of the backend. It's responsible for exposing API endpoints that the frontend can communicate with. It acts as the bridge between the user interface and the core CV generation logic.

Detailed File Breakdown:
main.py: The heart of the API. It initializes the FastAPI application, defines all the API endpoints (e.g., /generate-template, /generate-section), handles incoming requests, and calls the appropriate functions from the Backend directory.

env/: A Python virtual environment to isolate the project's dependencies, ensuring it runs consistently across different machines.

📁 Backend/

Purpose: This is the core logic center of the entire application. It contains all the Python modules responsible for the actual CV generation, from interacting with the AI to creating and manipulating the LaTeX files. It does not handle any web requests directly; it only provides functions for the API layer to call.

Detailed File Breakdown:
llm_module.py: This module is the direct interface to the Google Gemini AI. Its primary function, generate_section_latex, takes user input for a specific CV section, combines it with a carefully crafted prompt, sends it to the Gemini API, and returns the formatted LaTeX code.

prompts.py: This is a critical file containing the "secret sauce" of the application. It stores a dictionary of detailed prompt templates. Each prompt is engineered to instruct the LLM on the exact LaTeX format to return for a given section, ensuring consistency and quality.

section_generator.py: This module orchestrates the generation of CV content. When the user submits a prompt, this module calls the llm_module to get the AI-generated

LaTeX. It then saves this generated content into an Excel file (Output/df.xlsx), which acts as a simple database or persistent storage for the CV sections.

header_generator.py: A specialized module that handles the CV's header. It takes the user's basic information (name, email, etc.) and directly replaces placeholders in the LaTeX template.

template_maker.py: This module is responsible for creating a custom LaTeX template for each user. It starts with a MasterTemplate.tex and, based on the sections the user selected in the UI, it removes the unselected sections, producing a clean, tailored template.tex.

render_cv.py: This is the final step in the backend process. It reads all the generated LaTeX sections from the df.xlsx file and injects them into the custom template.tex, producing the final, complete final_cv.tex file.

📁 Frontend/

Purpose: This folder contains the entire user-facing application. It's a complete React project that was set up using Vite.

Key Files & Folders:
Promptfolio/src/: The main source code for the React app.

Promptfolio/src/components/: Contains reusable React components that make up the UI, such as CVSectionSelector.jsx, PromptSection.jsx, and LatexSnippet.jsx.

Promptfolio/src/css/: Contains the CSS stylesheets, organized per-component to keep the styling modular and maintainable.

Promptfolio/src/App.jsx: The main component that assembles all other components to create the application layout.

Promptfolio/package.json: Defines the project's frontend dependencies (like react) and scripts for running (dev) and building (build) the application.

📁 Output/

Purpose: This folder acts as a storage location for generated files.

Key Files:
df.xlsx: An Excel file used as a simple data store for the generated LaTeX content of each CV section. This is a clever way to manage the state of the CV being built.

final_cv.tex: The final, complete LaTeX file that is generated after all sections are processed and merged.

📁 Templates/

Purpose: This folder stores the base LaTeX templates that provide the CV's design and structure.

Key Files:
MasterTemplate.tex: The main template file containing all possible sections and placeholders.

template.tex: A temporary, customized template that is generated by template_maker.py based on the user's choices.

How to Run This Project: A Step-by-Step Guide

Before you begin, ensure you have the following software installed on your system:

* **Git:** For cloning the repository.
* **Python** (version 3.8 or higher) and *pip*.
* **Node.js** (which includes *npm*).

Step 1: Clone the Repository

Step 2: Configure the Backend

Create and Activate a Virtual Environment:

From the project's root's API directory (*/Promptfolio/API/*), create a Python virtual environment. This will keep the project's dependencies isolated.

```
# Create the virtual environment
python -m venv env


# Activate the environment
.\env\Scripts\Activate
```

Install Python Dependencies:

Once the virtual environment is active, install all the required Python packages using the requirements.txt file.

```
pip install -r requirements.txt
```

Set Up Your API Key:

The application uses the Google Gemini API. You need to provide your API key.

- Navigate to the Backend/ folder.
- Create a new file named .env

Inside this .env file, add the following line, replacing your_key_here with your actual Gemini API key:

```
GEMINI_API_KEY=your_key_here
```

Step 3: Configure the Frontend

The frontend is a React application. You'll need to install its Node.js dependencies.

Navigate to the Frontend Directory:

From the project root, move into the frontend's directory.

```
cd Frontend/Promptfolio
```

Install Node.js Dependencies:

Use npm to install all the packages defined in the package.json file.

```
npm install
```

Step 4: Run the Application

To use the application, both the backend server and the frontend server must be running at the same time. It's best to use two separate terminals for this.

In your FIRST terminal (for the Backend):

- Make sure you are in the project's root directory (Promptfolio).
- Ensure your Python virtual environment is active.
- Start the FastAPI server:

```
uvicorn API.main:app --reload
```

Your backend should now be running at http://127.0.0.1:8000/docs

In your SECOND terminal (for the Frontend):

- Navigate to the frontend directory (Promptfolio/Frontend/Promptfolio).
- Start the React development server:

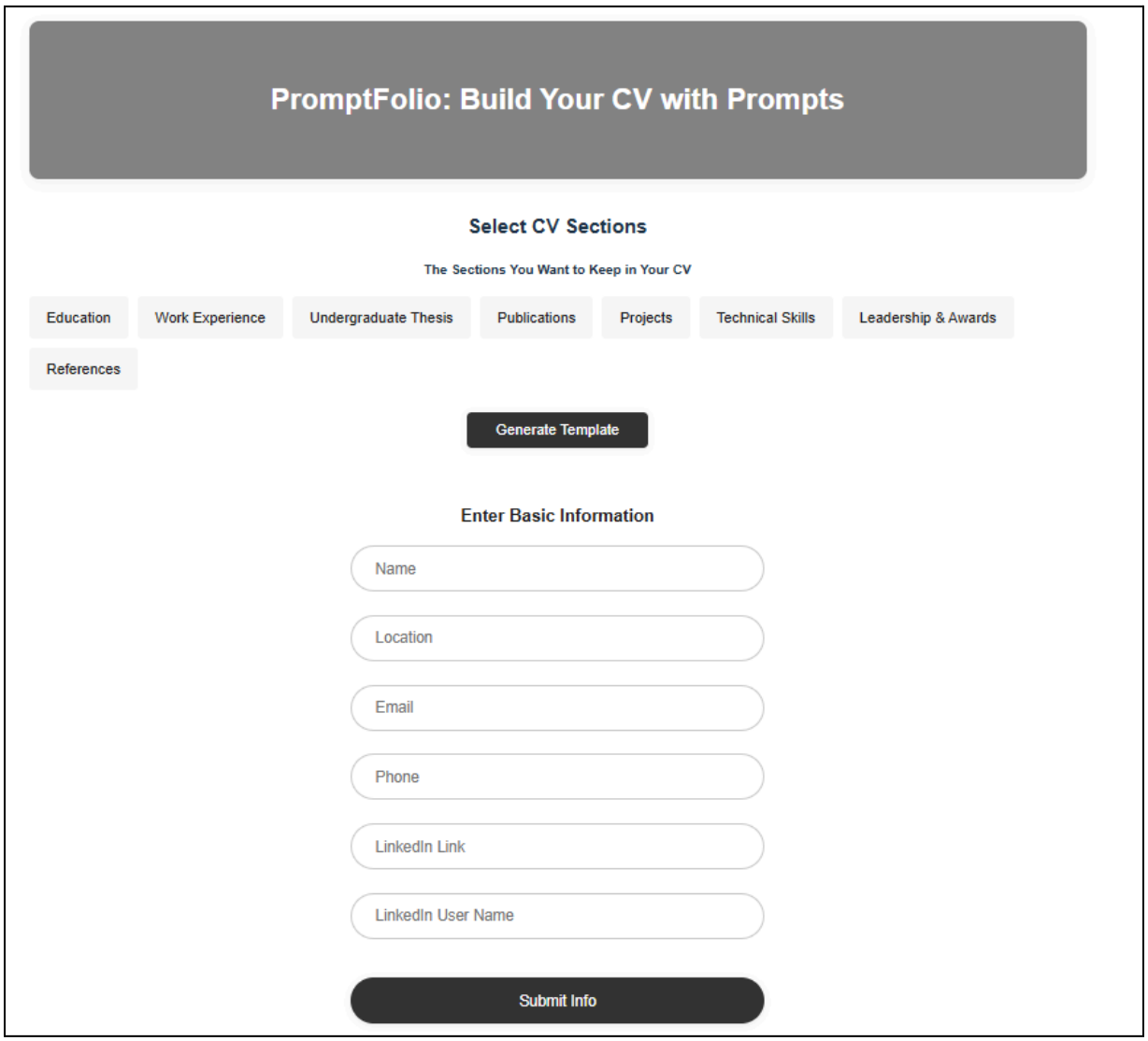
```
npm run dev
```

Your frontend should now be running at http://localhost:5173
You should now see the "PromptFolio" application running and ready to use.

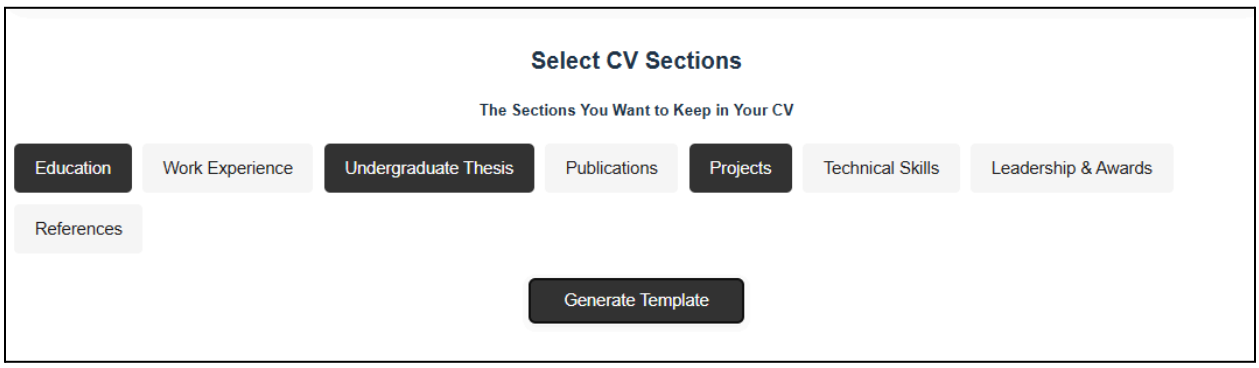
Make Sure You set the .env file and set your Gemini API. Also, Check Your API and Frontend is running in the given urls

Screenshots

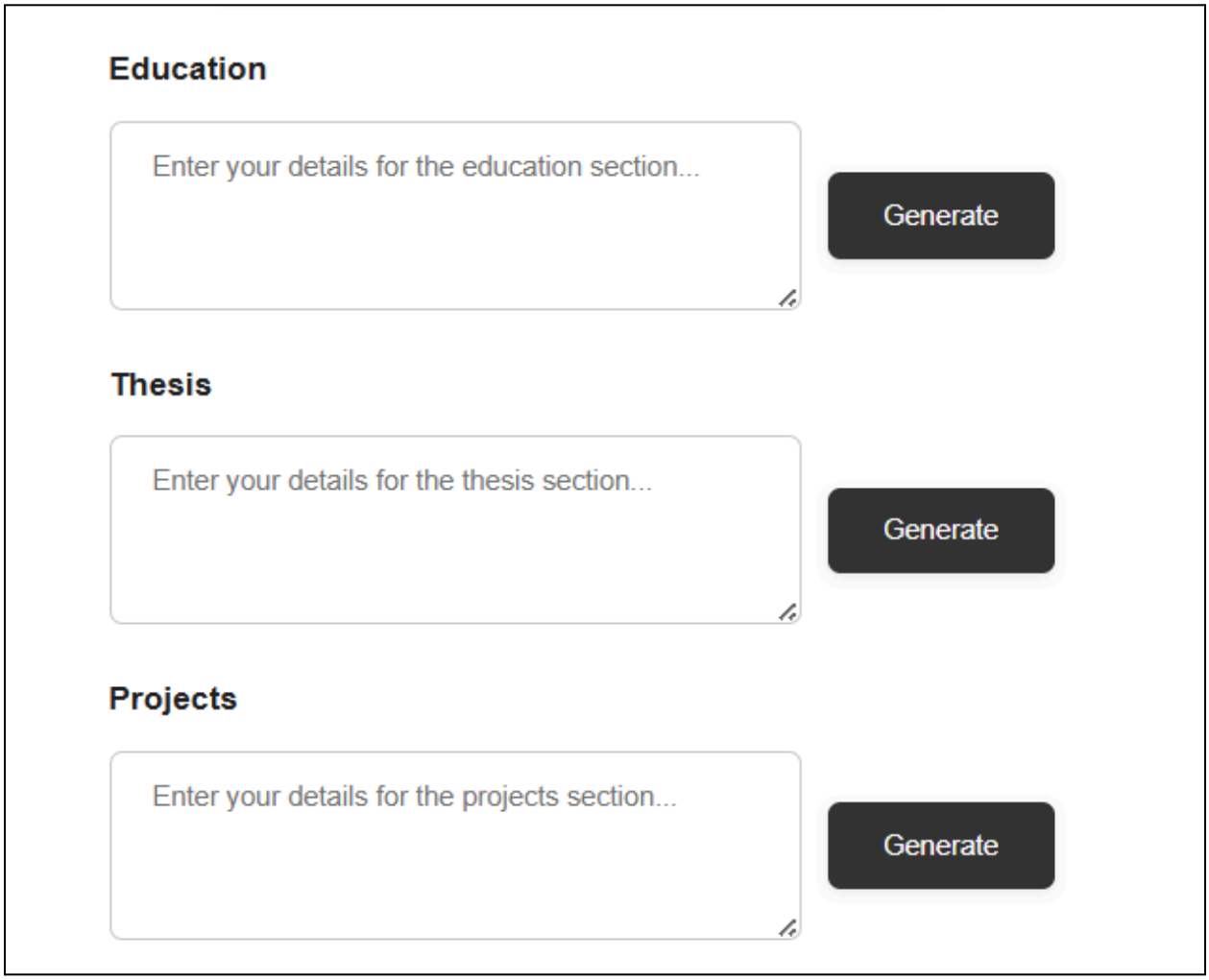
Website Interface



Select Your Preferred Section and generate!



Give Your Section Prompt and Generate each Sections



Output CV

Tushar Das

211, Narsingdi, Dhaka, Bangladesh | tushardas9085@gmail.com | 01731351544 | tushar9085

Education

Rajshahi University of Engineering & Technology — *Rajshahi, Bangladesh*
Bsc in CSE — *2018 – 2024*

- **GPA/CGPA:**

Notre Dame College — *Dhaka, Bangladesh*
HSC in Science Group — *2016 – 2018*

- **GPA/CGPA:**

Brahmondi K.K.M. Govt. High School — *Brahmondi, Bangladesh*
SSC in Science Group — *2014 – 2016*

- **GPA/CGPA:**

Undergraduate Thesis

Exploring EEG Signals for Predicting Human Emotions Using Machine Learning and Deep Learning Methods
EEG, Machine Learning, Deep Learning, Feature Extraction

- This thesis investigated the feasibility of predicting human emotions from electroencephalography (EEG) signals using machine learning and deep learning techniques. A novel feature extraction method was developed and implemented, significantly improving the accuracy of emotion classification compared to existing approaches. The results demonstrate the potential of this methodology for applications in affective computing and human-computer interaction.

Projects

PromptFolio
Frontend, Backend, API, LLM

- Developed a full-stack application generating LaTeX CV code using Large Language Models.
- Designed and implemented the frontend, backend, and API infrastructure.

6-bit CPU
Computer Architecture

- Designed and implemented a 6-bit CPU with ADD, MUL, and DIV operations.
- Completed as part of a computer architecture course.

Personal Portfolio Website

- Developed a personal website to showcase projects and skills.

SentimentFlex App
Naive Bayes, NLP

- Created an application predicting movie sentiment (positive or negative reviews) using Naive Bayes algorithm and NLP techniques.

Road Pavement Detection
YOLO Algorithm, Custom Dataset

- Developed a classification model to detect road damage (potholes, cracks) using the YOLO algorithm.
- Created a custom dataset for training and evaluation.