Minal Pawar:

Orchids SWE Internship Take-Home Challenge: Website Cloning

Overview

This project is part of the Orchids Software Engineering Internship take-home challenge. It is a basic version of a website cloning tool. The main idea is to let a user enter any public website link, and the app will show a clean, simple copy of that website.
The app works in three steps:

1. It visits the website and gathers important design details like the title, styles, and page content.
2. It creates a screenshot image of the original website.
3. It shows a rebuilt version of the site using the collected information, styled to look similar.
   Currently, the HTML copy is created using Python code, but the design allows future updates to plug in a real AI model (LLM) to make even better clones.

Tech Stack

- Frontend: Next.js (React, TypeScript)
- Backend: Python, FastAPI
- Web Scraping: BeautifulSoup, Requests
- Screenshot Generation: wkhtmltoimage (via subprocess)

Features

Web App

A simple frontend interface allows the user to enter a public website URL and preview the cloned HTML.

Website Scraping

The backend collects useful design information from the input website to help recreate its look. It extracts:

- **Page Title** – the name shown in the browser tab
- **Meta Tags** – for SEO and page details
- **Stylesheets** – inlined when possible, to preserve basic styling

- **Favicon** – the small icon next to the title in the browser tab
- **Body Content** – cleaned up by removing scripts, iframes, styles, and other unnecessary elements for safety and simplicity

Screenshot Capture

The backend uses a tool called wkhtmltoimage to take a screenshot of the original website. This image is then made available through the FastAPI server.
• **Purpose**: To help users visually compare the original website with the cloned HTML version and ensure the design looks similar.

LLM Cloning (Simulated)

What was required:
The challenge asked for a system that uses an AI model (called an LLM — Large Language Model) to analyze a public website and then generate new HTML that looks and behaves like the original. The goal was to use AI to recreate the structure, layout, and styling of any given site.

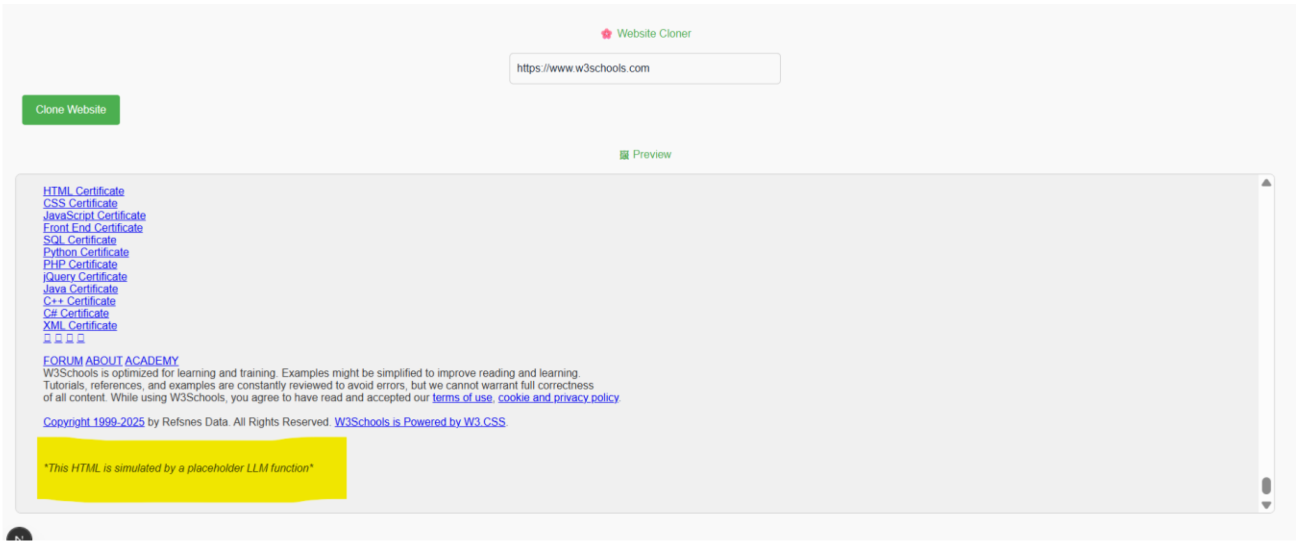
What I did instead:
Since I didn't have access to paid LLM APIs like GPT-4, Claude, or Gemini, I created a placeholder function in the backend called generate_html_with_llm().
This function simulates what an AI would generate — using the scraped title and body content to return a basic, styled HTML page. It's clearly marked as a simulation and helps show how the LLM integration would work in the real version.

Why I chose this method:
Adding this simulated function allowed me to follow the required pipeline without needing actual API access.
Instead of stopping at scraping and screenshotting, my backend now includes a **dedicated section** where the LLM would be plugged in. This shows my understanding of the architecture and makes future upgrades easier.

How it can be replaced with a real LLM:
The system is already modular. When I gain access to a real LLM (like GPT-4 via OpenAI API), I just need to update the generate_html_with_llm() function to:

1. **Send a prompt** to the LLM using the scraped context — title, body content, styles, meta tags, etc.
2. **Receive and render** the AI-generated HTML returned by the model.
3. **Replace** the simulated output with that real HTML before it's sent to the frontend for preview.

Because this placeholder is already wired into the flow, the future update will be seamless.



HTML Certificate
CSS Certificate
JavaScript Certificate
Front End Certificate
SQL Certificate
Python Certificate
PHP Certificate
jQuery Certificate
Java Certificate
C++ Certificate
C# Certificate
XML Certificate

FORUM ABOUT ACADEMY
W3Schools is optimized for learning and training. Examples might be simplified to improve reading and learning.
Tutorials, references, and examples are constantly reviewed to avoid errors, but we cannot warrant full correctness
of all content. While using W3Schools, you agree to have read and accepted our terms of use, cookie and privacy policy.

Copyright 1999-2025 by Refsnes Data. All Rights Reserved. W3Schools is Powered by W3.CSS.

This HTML is simulated by a placeholder LLM function

Setup Instructions

1. Backend Setup

```
cd backend                          # Go to backend folder
python -m venv .venv                     # Create virtual environment
.\.venv\Scripts\activate                # Activate it (on Windows)
pip install -r requirements.txt      # Install Python packages
mkdir screenshots                      # Folder for screenshot storage
uv run fastapi dev                     # Start the FastAPI backend
```

Make sure wkhtmltoimage is installed and available in your system path.

2. Frontend Setup

```
cd ../frontend          # Go to frontend folder
npm install             # Install node packages
npm run dev              # Start the frontend (Next.js)
```

3. Open in Browser

- Frontend: http://localhost:3000
- Backend: http://localhost:8000
- Screenshot endpoint: http://localhost:8000/screenshots/

Limitations

- No real LLM was used due to access limitations.
- Screenshot generation may fail on complex JS-heavy sites.
- Only one stylesheet is inlined for performance reasons.

Demo Video

Link: https://youtu.be/bk8Gcs2YXCs
