Index.html

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>Web Reconstruction Tool</title>

    <link rel="stylesheet" href="/static/styles.css" />

    <link

      href="https://fonts.googleapis.com/css2?family=DM+Serif+Display&display=swap"

      rel="stylesheet"

    />

  </head>

  <body>

    <div class="container">

      <img src="/static/logoweb.png" alt="Logo" class="logo" />

      <h1 class="main-title">WEB RECONSTRUCTION TOOL</h1>

      <p class="subtitle">YOUR STYLE, YOUR RULES</p>

      <h2 class="tagline">STYLE YOUR WEBSITE THE WAY YOU WANT TO!</h2>

      <form id="styleForm" class="form">

        <label for="url" class="form-label">Website URL</label>

        <input

          type="url"

          id="url"

          name="url"

          placeholder="https://example.com"

          required

          class="url-input"

        />

        <button type="submit" class="submit-btn">STYLE IT NOW</button>

      </form>

      <div class="dual-panel">

        <div class="panel left-panel">

          <div class="panel-header">

            <span class="panel-title">Body Content</span>

            <span class="panel-subtitle">Source content from scraped website</span>

          </div>

          <div id="result" class="panel-content">

            Enter a URL above and click "Style It Now" to view content.

          </div>

        </div>

        <div class="panel right-panel">

          <div class="panel-header">

            <span class="panel-title">Edited Output</span>

            <span class="panel-subtitle">AI-enhanced content modifications</span>

          </div>

          <div id="edited-result" class="panel-content">

            Select text from the left panel to edit

          </div>

        </div>

      </div>

    </div>

    <script src="/static/scripts.js"></script>

  </body>

</html>

Scripts.js

document.addEventListener('DOMContentLoaded', () => {

  console.log("📦 DOM fully loaded");

  const form = document.getElementById("styleForm");

  const resultDiv = document.getElementById("result");

  if (!form) console.error("❌ Form not found!");

  if (!resultDiv) console.error("❌ Result box not found!");

  form.addEventListener("submit", async function (e) {

    e.preventDefault();

    console.log("📨 Form submitted");

    const url = document.getElementById("url").value.trim();

    console.log("🌐 URL entered:", url);

    if (!url) {

      alert("Please enter a valid URL");

      return;

    }

    // Show loading state

    resultDiv.innerHTML = "<div class='loading'>Scraping website...<div class='spinner'></div></div>";

    try {

      console.log("⏳ Sending request to backend...");

      const response = await fetch('/scrape', {

        method: 'POST',

        headers: {

          'Content-Type': 'application/json',

          'Accept': 'application/json'

        },

        body: JSON.stringify({ url }),

      });

      if (!response.ok) {

        throw new Error(`HTTP error! status: ${response.status}`);

      }

      const data = await response.json();

      console.log("✅ Backend response:", data);

      if (data.error) {

        resultDiv.innerHTML = `<div class="error">Error: ${data.error}</div>`;

        return;

      }

      // Display structured content

      displayStructuredContent(data, url);

      // Initialize text selection functionality

      initializeTextSelection();

    } catch (error) {

      console.error("❌ Error:", error);

      resultDiv.innerHTML = `<div class="error">Error: ${error.message}</div>`;

    }

  });

});

function displayStructuredContent(data, url) {

  const resultDiv = document.getElementById("result");

  let html = `

    <div class="url-header">

      <strong>Source:</strong>

      <a href="${url}" target="\_blank" rel="noopener noreferrer" class="source-url">${url}</a>

    </div>

    <div class="page-info">

      <h2 class="page-title">${data.title}</h2>

      <p class="meta-description">${data.meta\_description}</p>

    </div>

  `;

  // Display images if any

  if (data.images && data.images.length > 0) {

    html += `

      <div class="images-section">

        <h3>Images Found (${data.images.length})</h3>

        <div class="images-grid">

    `;

    data.images.forEach((img, index) => {

      html += `

        <div class="image-block">

          <img src="${img.url}" alt="${img.alt}" class="scraped-image" onerror="this.style.display='none'">

          <p class="image-caption">${img.alt}</p>

        </div>

      `;

    });

    html += `</div></div>`;

  }

  // Display content blocks

  if (data.content\_blocks && data.content\_blocks.length > 0) {

    html += `<div class="content-blocks">`;

    data.content\_blocks.forEach((block, index) => {

      html += `

        <div class="content-block" data-block-id="${block.id}">

          <div class="block-header">

            <span class="block-number">Block ${index + 1}</span>

            <span class="block-type">${block.type.toUpperCase()}</span>

          </div>

          <div class="block-content selectable-content">

            ${block.text}

          </div>

      `;

      // Add images within this block if any

      if (block.images && block.images.length > 0) {

        html += `<div class="block-images">`;

        block.images.forEach(img => {

          html += `

            <div class="block-image">

              <img src="${img.url}" alt="${img.alt}" class="block-img" onerror="this.style.display='none'">

              <p class="block-img-caption">${img.alt}</p>

            </div>

          `;

        });

        html += `</div>`;

      }

      html += `</div>`;

    });

    html += `</div>`;

  } else {

    html += `<div class="no-content">No content blocks found</div>`;

  }

  resultDiv.innerHTML = html;

}

function initializeTextSelection() {

  console.log("🔧 Initializing text selection");

  const resultDiv = document.getElementById("result");

  if (!resultDiv) {

    console.error("❌ Result div not found");

    return;

  }

  // Add event listeners to all selectable content

  const selectableElements = resultDiv.querySelectorAll('.selectable-content');

  selectableElements.forEach(element => {

    element.addEventListener('mouseup', function() {

      const selection = window.getSelection();

      if (!selection.toString().trim()) return;

      console.log("📝 Text selected:", selection.toString().trim());

      showFloatingMenu(selection);

    });

  });

  console.log(`✅ Added selection listeners to ${selectableElements.length} elements`);

}

function showFloatingMenu(selection) {

  console.log("🎯 Creating floating menu");

  // Remove any existing menu

  const existingMenu = document.querySelector('.floating-menu');

  if (existingMenu) {

    existingMenu.remove();

  }

  // Create new menu

  const menu = document.createElement('div');

  menu.className = 'floating-menu';

  // Define edit actions

  const editActions = [

    {name: 'Rephrase', action: 'rephrase'},

    {name: 'Simplify', action: 'simplify'},

    {name: 'Shorten', action: 'shorten'},

    {name: 'Expand', action: 'lengthen'},

    {name: 'Formal', action: 'change\_tone'}

  ];

  // Create buttons for each action

  editActions.forEach(item => {

    const button = document.createElement('button');

    button.textContent = item.name;

    button.dataset.action = item.action;

    button.addEventListener('click', () => handleEdit(selection, item.action));

    menu.appendChild(button);

  });

  // Position menu near selection

  const range = selection.getRangeAt(0);

  const rect = range.getBoundingClientRect();

  menu.style.position = 'absolute';

  menu.style.top = `${rect.bottom + window.scrollY}px`;

  menu.style.left = `${rect.left + window.scrollX}px`;

  document.body.appendChild(menu);

  console.log("✅ Menu added to DOM");

}

async function handleEdit(selection, action) {

  console.log("✏️ Handling edit:", action);

  const selectedText = selection.toString().trim();

  if (!selectedText) {

    console.log("❌ No text selected");

    return;

  }

  const editedResult = document.getElementById("edited-result");

  if (!editedResult) {

    console.error("❌ edited-result element not found");

    return;

  }

  editedResult.textContent = "Editing...";

  try {

    console.log("📤 Sending edit request");

    const response = await fetch('/edit', {

      method: 'POST',

      headers: { 'Content-Type': 'application/json' },

      body: JSON.stringify({

        text: selectedText,

        action: action

      }),

    });

    if (!response.ok) {

      throw new Error(`HTTP error! status: ${response.status}`);

    }

    const data = await response.json();

    console.log("📥 Edit response:", data);

    if (data.error) {

      editedResult.textContent = `Error: ${data.error}`;

    } else {

      editedResult.textContent = data.result || "No changes made";

    }

  } catch (error) {

    console.error("❌ Edit error:", error);

    editedResult.textContent = `Error: ${error.message}`;

  }

  // Remove floating menu

  const menu = document.querySelector('.floating-menu');

  if (menu) {

    menu.remove();

  }

}

// Close menu when clicking elsewhere

document.addEventListener('mousedown', function(e) {

  if (!e.target.closest('.floating-menu')) {

    const menu = document.querySelector('.floating-menu');

    if (menu) {

      menu.remove();

    }

  }

});

Styles.css

/\* Base Styles \*/

body {

  font-family: "DM Serif Display", serif;

  background-color: #f8f9fa;

  margin: 0;

  padding: 0;

  color: #333;

}

.container {

  max-width: 1200px;

  margin: 0 auto;

  padding: 20px;

}

/\* Header Styles \*/

.logo {

  display: block;

  margin: 0 auto 20px;

  max-width: 150px;

}

.main-title {

  text-align: center;

  color: #4b0055;

  font-size: 2.5rem;

  margin-bottom: 0.5rem;

}

.subtitle {

  text-align: center;

  color: #6a0080;

  font-size: 1.2rem;

  margin-bottom: 1.5rem;

}

.tagline {

  text-align: center;

  color: #555;

  font-size: 1.4rem;

  margin-bottom: 2rem;

}

/\* Form Styles \*/

.form {

  background: white;

  padding: 25px;

  border-radius: 10px;

  box-shadow: 0 2px 15px rgba(0, 0, 0, 0.1);

  margin-bottom: 30px;

  max-width: 800px;

  margin-left: auto;

  margin-right: auto;

}

.form-label {

  display: block;

  margin-bottom: 8px;

  font-weight: bold;

  color: #4b0055;

}

.url-input {

  width: 100%;

  padding: 12px;

  border: 2px solid #ddd;

  border-radius: 6px;

  font-size: 1rem;

  margin-bottom: 20px;

  transition: border-color 0.3s;

}

.url-input:focus {

  outline: none;

  border-color: #4b0055;

}

.submit-btn {

  background: #4b0055;

  color: white;

  padding: 12px 30px;

  border: none;

  border-radius: 6px;

  font-size: 1.1rem;

  cursor: pointer;

  transition: background 0.3s;

  display: block;

  margin: 0 auto;

}

.submit-btn:hover {

  background: #6a0080;

}

/\* Dual Panel Layout \*/

.dual-panel {

  display: flex;

  gap: 20px;

  margin-top: 20px;

  width: 100%;

}

.panel {

  flex: 1;

  background: white;

  border-radius: 8px;

  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

  overflow: hidden;

  display: flex;

  flex-direction: column;

  height: 600px;

}

.panel-header {

  background: #4b0055;

  color: white;

  padding: 15px 20px;

  display: flex;

  flex-direction: column;

  gap: 5px;

}

.panel-title {

  font-size: 1.2rem;

  font-weight: bold;

}

.panel-subtitle {

  font-size: 0.9rem;

  opacity: 0.8;

}

.panel-content {

  flex: 1;

  padding: 20px;

  overflow-y: auto;

  line-height: 1.6;

  border: 1px solid #eee;

  border-top: none;

  border-radius: 0 0 8px 8px;

}

/\* URL Header Styles \*/

.url-header {

  background: #f8f9fa;

  border: 1px solid #e9ecef;

  border-radius: 6px;

  padding: 12px 16px;

  margin-bottom: 20px;

  font-size: 0.95rem;

  color: #495057;

  display: flex;

  align-items: center;

  gap: 10px;

}

.source-url {

  color: #4b0055;

  text-decoration: none;

  font-weight: 500;

  transition: all 0.2s ease;

  word-break: break-all;

  flex: 1;

}

.source-url:hover {

  color: #6a0080;

  text-decoration: underline;

}

/\* Page Info Styles \*/

.page-info {

  background: #fff;

  border: 1px solid #e9ecef;

  border-radius: 8px;

  padding: 20px;

  margin-bottom: 20px;

}

.page-title {

  color: #4b0055;

  font-size: 1.5rem;

  margin-bottom: 10px;

}

.meta-description {

  color: #666;

  font-style: italic;

  margin: 0;

}

/\* Images Section \*/

.images-section {

  margin-bottom: 30px;

}

.images-section h3 {

  color: #4b0055;

  font-size: 1.3rem;

  margin-bottom: 15px;

}

.images-grid {

  display: grid;

  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));

  gap: 15px;

  margin-bottom: 20px;

}

.image-block {

  background: white;

  border: 1px solid #e9ecef;

  border-radius: 8px;

  padding: 10px;

  text-align: center;

}

.scraped-image {

  max-width: 100%;

  max-height: 150px;

  object-fit: cover;

  border-radius: 4px;

  margin-bottom: 8px;

}

.image-caption {

  font-size: 0.9rem;

  color: #666;

  margin: 0;

}

/\* Content Blocks \*/

.content-blocks {

  display: flex;

  flex-direction: column;

  gap: 20px;

}

.content-block {

  background: white;

  border: 2px solid #e9ecef;

  border-radius: 10px;

  overflow: hidden;

  transition: all 0.3s ease;

}

.content-block:hover {

  border-color: #4b0055;

  box-shadow: 0 4px 15px rgba(75, 0, 85, 0.1);

}

.block-header {

  background: linear-gradient(135deg, #4b0055, #6a0080);

  color: white;

  padding: 15px 20px;

  display: flex;

  justify-content: space-between;

  align-items: center;

}

.block-number {

  font-size: 1.1rem;

  font-weight: bold;

}

.block-type {

  font-size: 0.9rem;

  background: rgba(255, 255, 255, 0.2);

  padding: 4px 8px;

  border-radius: 4px;

}

.block-content {

  padding: 20px;

  font-size: 1rem;

  line-height: 1.6;

  cursor: text;

}

.selectable-content {

  user-select: text;

  -webkit-user-select: text;

  -moz-user-select: text;

  -ms-user-select: text;

}

.selectable-content:hover {

  background: #f8f9fa;

}

/\* Block Images \*/

.block-images {

  display: flex;

  gap: 10px;

  padding: 0 20px 20px;

  flex-wrap: wrap;

}

.block-image {

  background: #f8f9fa;

  border: 1px solid #e9ecef;

  border-radius: 6px;

  padding: 8px;

  text-align: center;

  flex: 1;

  min-width: 150px;

}

.block-img {

  max-width: 100%;

  max-height: 100px;

  object-fit: cover;

  border-radius: 4px;

  margin-bottom: 5px;

}

.block-img-caption {

  font-size: 0.8rem;

  color: #666;

  margin: 0;

}

/\* Floating Menu \*/

.floating-menu {

  position: absolute;

  background: white;

  border: 1px solid #ddd;

  border-radius: 8px;

  padding: 8px;

  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);

  z-index: 1000;

  display: flex;

  gap: 8px;

  animation: fadeIn 0.2s ease-out;

}

.floating-menu button {

  background: #4b0055;

  color: white;

  border: none;

  padding: 8px 16px;

  border-radius: 4px;

  cursor: pointer;

  font-size: 0.9rem;

  transition: all 0.2s;

}

.floating-menu button:hover {

  background: #6a0080;

}

@keyframes fadeIn {

  from {

    opacity: 0;

    transform: translateY(5px);

  }

  to {

    opacity: 1;

    transform: translateY(0);

  }

}

/\* Loading and Error States \*/

.loading {

  text-align: center;

  padding: 40px;

  color: #666;

  font-style: italic;

}

.spinner {

  width: 40px;

  height: 40px;

  border: 4px solid #f3f3f3;

  border-top: 4px solid #4b0055;

  border-radius: 50%;

  animation: spin 1s linear infinite;

  margin: 20px auto;

}

@keyframes spin {

  0% {

    transform: rotate(0deg);

  }

  100% {

    transform: rotate(360deg);

  }

}

.error {

  background: #f8d7da;

  color: #721c24;

  padding: 15px;

  border-radius: 6px;

  border: 1px solid #f5c6cb;

  margin: 10px 0;

}

.no-content {

  text-align: center;

  color: #666;

  font-style: italic;

  padding: 40px;

}

/\* Responsive Design \*/

@media (max-width: 768px) {

  .dual-panel {

    flex-direction: column;

  }

  .panel {

    height: auto;

    max-height: 400px;

  }

  .main-title {

    font-size: 2rem;

  }

  .tagline {

    font-size: 1.2rem;

  }

  .url-header {

    flex-direction: column;

    align-items: flex-start;

    gap: 5px;

  }

  .images-grid {

    grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));

  }

  .block-images {

    flex-direction: column;

  }

  .floating-menu {

    flex-wrap: wrap;

  }

}

Main.py

from fastapi import FastAPI

from fastapi.staticfiles import StaticFiles

from fastapi.responses import FileResponse

from pydantic import BaseModel

from typing import Literal

import requests

from scraper import scrape\_text\_from\_url

from fastapi.middleware.cors import CORSMiddleware

app = FastAPI()

# Add CORS middleware

app.add\_middleware(

    CORSMiddleware,

    allow\_origins=["\*"],

    allow\_credentials=True,

    allow\_methods=["\*"],

    allow\_headers=["\*"],

)

# Mount static files

app.mount("/static", StaticFiles(directory="static"), name="static")

@app.get("/")

def read\_index():

    return FileResponse("static/index.html")

class URLRequest(BaseModel):

    url: str

@app.post("/scrape")

def scrape\_url(request: URLRequest):

    print(f"📥 Scraping: {request.url}")

    scraped\_data = scrape\_text\_from\_url(request.url)

    return scraped\_data

class EditRequest(BaseModel):

    text: str

    action: Literal['rephrase', 'simplify', 'shorten', 'lengthen', 'change\_tone']

@app.post("/edit")

async def edit\_text(request: EditRequest):

    print(f"Received edit request for action: {request.action}")

    prompt\_map = {

        "rephrase": f"Rephrase this text while keeping the same meaning:\n{request.text}",

        "simplify": f"Simplify this text for better understanding:\n{request.text}",

        "shorten": f"Make this text more concise:\n{request.text}",

        "lengthen": f"Expand on this text with more details:\n{request.text}",

        "change\_tone": f"Rewrite this text in a formal tone:\n{request.text}",

    }

    prompt = prompt\_map.get(request.action, request.text)

    try:

        response = requests.post(

            "http://localhost:11434/api/generate",

            json={

                "model": "llama3",

                "prompt": prompt,

                "stream": False

            },

            timeout=30

        )

        response.raise\_for\_status()

        result = response.json().get("response", "").strip()

        return {"result": result}

    except Exception as e:

        return {"error": str(e)}

@app.get("/test-scrape")

async def test\_scrape():

    test\_url = "https://example.com"

    scraped = scrape\_text\_from\_url(test\_url)

    return scraped

scraper.py

import requests

from bs4 import BeautifulSoup

from urllib.parse import urljoin, urlparse

import json

def scrape\_text\_from\_url(url):

    try:

        headers = {

            "User-Agent": (

                "Mozilla/5.0 (Windows NT 10.0; Win64; x64) "

                "AppleWebKit/537.36 (KHTML, like Gecko) "

                "Chrome/114.0.0.0 Safari/537.36"

            ),

            "Accept": "text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8",

            "Accept-Encoding": "gzip, deflate",

            "Accept-Language": "en-US,en;q=0.9"

        }

        response = requests.get(url, headers=headers, timeout=10)

        response.raise\_for\_status()

        soup = BeautifulSoup(response.text, 'html.parser')

        # === Title ===

        title\_tag = soup.title.string.strip() if soup.title else "No title found"

        # === Meta Description ===

        meta\_desc = soup.find("meta", attrs={"name": "description"})

        meta\_description = meta\_desc['content'].strip() if meta\_desc and 'content' in meta\_desc.attrs else "No meta description found"

        # === Images ===

        images = []

        img\_tags = soup.find\_all('img')

        for img in img\_tags:

            src = img.get('src')

            if src:

                # Convert relative URLs to absolute URLs

                full\_url = urljoin(url, src)

                alt\_text = img.get('alt', 'No alt text')

                images.append({

                    'url': full\_url,

                    'alt': alt\_text

                })

        # === Clean Body Content ===

        for tag in soup(['script', 'style', 'nav', 'footer', 'header', 'aside']):

            tag.decompose()

        # === Extract Content Blocks ===

        content\_blocks = []

        block\_counter = 1

        if soup.body:

            for element in soup.body.find\_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6', 'p', 'div', 'article', 'section']):

                text = element.get\_text(strip=True)

                if text and len(text) > 20:  # Only include substantial text

                    # Check if this element contains an image

                    block\_images = []

                    for img in element.find\_all('img'):

                        src = img.get('src')

                        if src:

                            full\_url = urljoin(url, src)

                            alt\_text = img.get('alt', 'No alt text')

                            block\_images.append({

                                'url': full\_url,

                                'alt': alt\_text

                            })

                    content\_blocks.append({

                        'id': f'block\_{block\_counter}',

                        'type': element.name,

                        'text': text,

                        'images': block\_images

                    })

                    block\_counter += 1

        # === Return Structured Data ===

        return {

            'title': title\_tag,

            'meta\_description': meta\_description,

            'images': images[:10],  # Limit to first 10 images

            'content\_blocks': content\_blocks[:20],  # Limit to first 20 blocks

            'url': url

        }

    except Exception as e:

        return {

            'error': f"Error scraping: {str(e)}",

            'title': 'Error',

            'meta\_description': 'Failed to scrape',

            'images': [],

            'content\_blocks': [],

            'url': url

        }