UI-Generator

CSCI 2340 Spring ‘24

Brian Sutioso, Shuyi Qi, Yajie Wan, Qingyuan Lin, Kaan Aygen, Liming Chen, Alejandro Jackson

Table of Contents

Requirements

- Our users are frontend engineers who develop web-based user interfaces and would like to expedite their development process.
- The users want a tool which generates a web page based on text description. Our current goal is to build an API, which the user can call and get back a webpage in HTML/CSS.
- Down the road, we also want to provide the user with more customized web page generation as well as template pages to start from.

Specifications

Summary

The goal of this project is to develop a prototype system that can take descriptions of user interfaces as input and generate code that implements the corresponding interface. The system should be able to handle an array of descriptions for different websites, such as e-commerce sites, social media front pages, and personal blogs, and generate code for HTML, CSS, and JavaScript.

The system should allow users to edit their sketches and see the corresponding code update in real-time, and vice versa. The implementation could involve analyzing and interpreting the sketch, using machine learning to find appropriate infrastructure, using search to find relevant code, or using a language model to generate a starting point and then modifying the result. The generated interface should be responsive and handle different screen sizes. The overall goal is to create a practical tool that can be incorporated into popular IDEs to simplify the process of UI design and development.

User Persona

Who are they?
Sarah is a skilled frontend engineer with several years of experience in developing web-based user interfaces. She is proficient in HTML, CSS, and JavaScript and often works on projects that require rapid development of web pages.

Why would they want to use this application?
Sarah wants to expedite her development process by automating the initial stages of creating web pages.

What are they looking for?
She is looking for a tool that can generate web pages based on text descriptions, allowing her to quickly prototype and iterate on different UI designs. Sarah also desires more customized web page generation capabilities, as well as access to template pages to provide a starting point for her projects.

How does the application provide what they need? How does the application match the experience these user(s) are looking for?
The application should offer a straightforward API that seamlessly integrates into Sarah's development workflow, enabling her to quickly obtain web page code based on her descriptions. It should provide customizable options and templates to cater to her diverse project requirements, ultimately streamlining the UI design and development process.

Functional Requirements

User Interface

- The application will provide a text-based form which asks for details.
  - The form elements include:
    - Page title
    - Color scheme preferences
    - Layout structure & sections
    - Content
    - Usage
    - Additional information
- Landing Page
  - The landing page should contain a clear and concise description of the application's functionality.
  - It should also provide a prominent button or link to start creating the webpage.
- Text-Based Form
  - Upon clicking the "Create Webpage" button, the user should be presented with a text-based form.
  - The form should include input fields for the following details:
    - Page title (text input)
    - Color scheme preferences (color picker or dropdown selection)
    - Layout structure & sections (text input or dropdown selection)
    - Content (textarea for free-form text input)
    - Specific elements (checkboxes or multi-select dropdown)
      - Forms
      - Tables
      - Images
      - Links
      - Et Cetera
    - Usage (dropdown selection)
      - e-commerce
      - Personal web page
      - Social media front page
      - Personal blog

            - Data analytics
                - Dashboard
            - Et Cetera
        - Additional information (textarea for free-form text input)
- Preview Section
    - Below the form, there should be a preview section that displays a live preview of the webpage as the user fills out the form.
    - The preview should update in real-time based on the input provided in the form.
- Source Code Output:
    - Upon completing the form, the user should be able to click a "Generate Code" button.
    - Clicking this button should display the source code of the webpage (HTML, CSS, and JavaScript) in a separate section or modal.
- Final Webpage View
    - Below the source code output, there should be an iframe that displays the final view of the webpage based on the input provided in the form.
    - The iframe should update in real-time as the user makes changes in the form or when the source code is generated.
- Navigation and Feedback
    - The user should be provided with clear navigation options to go back to the landing page or start a new webpage creation process.
    - There should also be a way for the user to provide feedback or report issues with the application.

Web Page Generation

- Generation
    - Pre-process user input
        - This step involves preparing the user input, such as text descriptions or sketches, to be used as input for the GPT API. It may include tasks like cleaning the input, formatting it appropriately, or extracting relevant information.
    - Prompt GPT API
        - In this step, the pre-processed user input is sent to the GPT API. The GPT model will process the input and generate a response based on the provided prompt. The GPT model utilizes its language processing capabilities to generate human-like text.
    - Send user input and receive response
        - The pre-processed user input is sent to the GPT API, and in return, you receive a response generated by the GPT model. This response can be in the form of text that represents the generated web page or any other desired output.
    - Post-process GPT output
        - After receiving the response from the GPT API, you may need to post-process the output to refine or modify it as per your requirements.

This step could involve tasks like formatting the generated code, adding additional functionality, or optimizing the output.

- Display to the user the final website along with source code
    - Finally, the processed output, which represents the generated web page, is displayed to the user. This can include rendering the web page in a browser or presenting it in any other suitable format. Additionally, the source code generated by the GPT model can also be shown to the user if desired.
- Example
    - Prompt

      the webpage has a background color of ${user input}. For headings, use the font ${user input} …. Split html and css with “split_here$”
    - Expected Results

      <html_code>
      split_here$
      <css_code>
      Create a folder
      Html_code → index.html
      Css_code → styles.css

Program Output

- The program will output the source code of a webpage.
    - This includes:
        - HTML
        - CSS
        - JavaScript
- The final view of the webpage will be displayed in iframe.

Technical Stack

Programming Language(s) *Proposed*

- Frontend:
    - JavaScript/TypeScript: For building interactive user interfaces and handling client-side logic.
    - React or Vue.js: Frontend frameworks for creating dynamic and responsive user interfaces.
    - HTML/CSS: For defining the structure and styling of the generated web pages.
- Backend:
    - Node.js: For server-side scripting and handling asynchronous tasks.
    - Express.js: A web application framework for Node.js, providing a robust set of features for building APIs and handling HTTP requests.

    - Python/Django: Alternatively, Python with Django could be used for backend development, offering a powerful web framework for rapid development.
- API Development:
    - RESTful API: For communication between the frontend and backend, allowing the frontend to send text descriptions and receive the corresponding web page code.
- Data Storage:
    - MongoDB or PostgreSQL: For storing user preferences, templates, and other relevant data if the system requires persistent storage.
- Code Generation:
    - Template Engine (e.g., Handlebars, EJS): For generating HTML/CSS code based on the user's input descriptions.
- Real-Time Updates:
    - WebSockets: To enable real-time communication between the frontend and backend, facilitating immediate updates as the user edits their input descriptions.
- Version Control:
    - Git: For managing the codebase, tracking changes, and facilitating collaboration among team members if applicable.
- Deployment and Hosting:
    - Docker: For containerization and deployment of the application, ensuring consistency across different environments.
    - AWS/Azure/GCP: For cloud-based hosting, providing scalability, reliability, and infrastructure services.
- Testing and Quality Assurance:
    - Jest/Mocha/Chai: Testing frameworks for writing and running unit tests and ensuring code quality.
    - Continuous Integration/Continuous Deployment (CI/CD) tools: Automating the build, test, and deployment processes.
- Security:
    - HTTPS: Ensuring secure communication between the client and server.
    - Authentication and Authorization: Implementing user authentication and access control measures to protect sensitive functionality and data.

GitHub

- The program will be held in a public github repo.
- The program will be open-source.

Non-functional Requirements

Performance

- The program should generate the web page code within a reasonable timeframe, allowing users to quickly obtain the desired output. This includes efficient parsing and interpretation of the input text descriptions to minimize processing time.

Compatibility

- Responsive Design
    - The user interface should be designed to be responsive and accessible on various devices, including desktops, tablets, and mobile phones.

Scalability

- The program should be able to handle a growing number of user requests and increasing complexity of input descriptions without significant degradation in performance. This scalability ensures that the program can accommodate a larger user base and more extensive projects.

Quality Assurance

Testing Requirements

- The program will be tested through integration testing.
    - This integration testing will test the appropriate changes to the program’s state and behavior upon various control inputs.
- The program will be tested through unit testing.
    - This unit testing will test appropriate outputs from the program’s methods.

Error Handling

- The program will handle invalid user inputs gracefully—such inputs will not crash the program.
    - The program can try to prevent errors through robust input validation.
    - The program will return clear and informative messages with poor inputs or when an error occurs.
        - This can include details about the nature of the error, the potential causes, and suggestions for resolution.

Documentation

Code Comments

- The program code will include detailed documentation.
- The program code will include comprehensive comments throughout its methods.

User Manual

- The program will include a README that explains navigation, inputs, and how the program works.

Challenges

- Web Page Embedding

    - Integrating the generated web page into an existing website or embedding it within a specific framework can be a challenge. Ensuring seamless integration, maintaining consistent styling, and addressing potential conflicts with existing code are important considerations.
- GPT Agent Prompting
    - Prompting the GPT agent effectively to generate accurate and relevant web page content may require careful consideration. Crafting clear and specific prompts that effectively communicate the user's requirements while aligning with the GPT model's capabilities is crucial.
- Understanding User Intent
    - Interpreting and precisely understanding the user's input to capture their exact requirements and preferences can be challenging. Addressing potential ambiguity in user input, accommodating diverse design expectations, and accurately translating user descriptions into actionable web page components are important aspects to consider.

Conclusion

The goal of this project is to develop a prototype system that can take descriptions of user interfaces as input and generate code that implements the corresponding interface. The system should be able to handle an array of descriptions for different websites, such as e-commerce sites, personal web pages, et Cetera, and generate code for HTML, CSS, and JavaScript.

The system should allow users to input a text description and receiving a sketch with attached HTML and CSS code. The implementation would involve analyzing and interpreting the text, using machine learning to find appropriate infrastructure, using search to find relevant code, or using a language model to generate a starting point and then modifying the result. The generated interface should be responsive and handle different screen sizes. The overall goal is to create a practical tool that can be incorporated into popular IDEs to simplify the process of UI design and development.

Sources

Specification guide put together will help from How to Build a Software Specification Document by Mayven Studios and Opera GX's in-browser AI, Aria.