

CODGEN

(Hand-Drawn HTML Element Code Generator)

Table of Contents

| | |
|--|-----------|
| ACKNOWLEDGEMENT... | 7 |
| DEDICATION... | 8 |
| ABSTRACT... | 9 |
| LIST OF FIGURES | 10 |
| LIST OF TABLES | 10 |
| LIST OF ABBREVIATIONS... | 11 |
| LIST OF USEFUL WEBSITES... | 11 |
| CHAPTER 1... | 12 |
| INTRODUCTION | 12 |
| 1.1.1. Brief | 12 |
| 1.1.2. Relevance to Course Modules | 12 |
| 1.1.3. Project Background | 13 |
| 1.1.4. Literature Review | 13 |
| 1.1.5. Methodology and Software Lifecycle | 14 |
| Requirement Gathering | 15 |
| Data Collection and Preprocessing | 15 |
| Model Training | 15 |
| Integration | 16 |
| Front-end Development | 16 |
| Testing and validation | 16 |
| CHAPTER 2... | 19 |
| PROBLEM DEFINITION | 19 |
| 2.1.1. Problem Statement | 19 |
| 2.1.2. Deliverables and Development Requirements | 19 |
| 2.1.3. Current Systems | 20 |
| CHAPTER 3... | 21 |

| | |
|--|-----------|
| REQUIREMENT ANALYSIS | 21 |
| 3.1.1. Use Case Diagrams | 21 |
| 3.1.2. Use case name and identifier (Use Case Narrative) | 23 |
| Main Menu... | 23 |
| 3.1.3. Functional Requirements | 24 |
| 3.1.4. Non-Functional Requirements | 25 |
| 3.1.5 Software Quality Attributes | 26 |
| CHAPTER 4 | 28 |
| DESIGN AND ARCHITECTURE | 28 |
| 4.1.1. System Architecture | 28 |
| User interface | 28 |
| Backend Processing | 28 |
| Optical Character Recognition | 29 |
| Element Recognition | 29 |
| Code Generation Module | 29 |
| 4.1.2. Data Flow | 29 |
| User interaction | 29 |
| Request Handling | 29 |
| API Processing | 30 |
| Backend Processing | 30 |
| Response to User | 30 |
| 4.1.3. Database and Data Storage | 30 |
| Database Schema Overview | 30 |
| Image Data | 30 |
| Generated HTML Code | 31 |
| Element Recognition Result | 31 |
| System Logs | 31 |
| Data Security | 31 |
| Encryption | 31 |
| Access Control | 31 |
| 4.1.4. External APIs | 31 |
| Integration Points | 32 |
| 4.1.5. Process Flow Representation | 33 |
| CHAPTER 5 | 35 |
| IMPLEMENTATION | 35 |

| | |
|----------------------------------|-----------|
| 5.1.1. Algorithms | 35 |
| 5.1.2. External API's | 36 |
| 5.1.3. User Interface | 37 |
| CHAPTER 6 | |
| 44 | |
| TESTING AND EVALUATION | 44 |
| 6.1.1. Testing Methodology..... | 44 |
| CHAPTER 7 | |
| 46 | |
| CONCLUSION AND FUTURE WORK | |
| 46 7.1.1. Conclusion | |
| 46 | |
| 7.1.2. Future Work | 47 |
| REFERENCES | 48 |

ACKNOWLEDGEMENT

We thank and praise Allah for the successful completion of this project. We are also grateful to our parents for their continuous support during this journey, and we pray to Allah to bless them with good health and prosperity both in this life and the hereafter.

We are sincerely grateful to our supervisor, Miss. Laraib Rana, for her continuous support and guidance, as well as her way of dealing with our mistakes and encouragement throughout this project. Her invaluable insights and expertise have significantly contributed to the success of our project. May Allah protect and bless her and her household from any harm.

We also appreciate our family members and siblings for their emotional support. When we felt down, they motivated us and cared for our physical and mental health day and night during our project and degree. May Allah shower His countless blessings upon them. Their care, support, and advice have been invaluable in getting us to this point.

We also acknowledge the contributions of the open-source community and the developers of the various tools and libraries that were instrumental in our project. Their dedication to creating and maintaining these resources has been invaluable.

DEDICATION

We dedicate our project to our supervisor, Miss. Laraib Rana, who provided us with valuable knowledge and guidance, leading us on the best path to achieve our project goals. This work is also dedicated to our parents and siblings, who have supported us through tough situations and challenges. Their continuous support and encouragement have been important to our success, and we are grateful for their trust in us. We hope to make them proud with the results of our hard work and dedication.

ABSTRACT

This project aims to develop a "Hand-Drawn HTML Element Code Generator," a system that combines machine learning with modern web development to enable the automatic conversion of hand-drawn HTML element sketches into code. The front-end interface, built using ReactJS, allows users to upload hand-drawn sketches, which are then processed by the Node.js server backend. The core functionality relies on Python-based machine learning models that detect and classify the hand-drawn elements, translating them into corresponding HTML code. The project utilizes pre-trained models, such as those in PyTorch and Keras, to perform object detection and classification. These models are trained on a custom dataset of hand-drawn HTML elements and are integrated into the system via an API created using Flask. The detected elements are mapped to predefined HTML templates, allowing for seamless code generation. The system is designed to streamline the web prototyping process, enabling faster iterations for designers and developers. By leveraging machine learning and web technologies, the tool offers an intuitive way to convert hand-drawn designs into functional HTML code, significantly reducing the time and effort required for early-stage web development. The project incorporates advanced features such as real-time communication between the front-end and back-end, user feedback mechanisms, and integration with cloud services for storing model data and managing user inputs. This innovative approach provides a practical and efficient solution for rapid prototyping, bridging the gap between design and development.

LIST OF FIGURES

| | |
|------------------------------|----|
| Figure 1 Lifecycle | 14 |
| Figure 2 Website Use Case | 21 |
| Figure 3 System Architecture | 28 |
| Figure 4 Process flow chart | 33 |
| Figure 5 Home Page | 38 |
| Figure 6 Feature | 38 |
| Figure 7 Testimonials | 39 |
| Figure 8 FAQ's | 39 |
| Figure 9 About Us | 39 |
| Figure 10 Contact Us | 40 |
| Figure 11 login | 40 |
| Figure 12 Signup | 41 |
| Figure 13 Password forget | 41 |
| Figure 14 New Chat | 42 |
| Figure 15 Image upload | 42 |
| Figure 16 Code generate | 43 |
| Figure 17 Code output | 43 |

LIST OF TABLES

| | |
|--|----|
| Table 1.1 Concept Relevance to Course Module | 12 |
| Table 2 Lifecycle | 17 |
| Table 3 Derivables | 19 |
| Table 4 Code generation | 23 |

LIST OF ABBREVIATIONS

AI: Artificial Intelligence

HTML: Hypertext Markup Language

REST: Representational State Transfer

API: Application Programming Interface

Flask: Web Development Framework

LIST OF USEFUL WEBSITES

<https://machinelearningmastery.com/the-transformer->

[model/ https://public.roboflow.com/](https://public.roboflow.com/)

<https://docs.ultralytics.com/models/yolov8/>

<https://github.com/>

CHAPTER 1

INTRODUCTION

Brief

The Hand-Drawn HTML Element Code Generator is an innovative tool designed to streamline the early stages of web development by converting hand-drawn sketches of HTML elements into actual code. The project leverages machine learning techniques, specifically object detection, to recognize various HTML components in hand-drawn form and automatically generate corresponding HTML code. This tool targets web designers and developers, offering a faster, more efficient prototyping process, reducing manual coding, and promoting more creative, free-form design exploration.

Relevance to Course Modules

In our project, several core concepts directly align with key course modules, demonstrating its academic relevance. The use of a database to store user inputs and processed data is closely related to Database Management Systems, which teaches how to design, implement, and manage databases efficiently.

For object detection, the concept of Machine Learning was used, a major aspect of this project, ties into the Artificial Intelligence course, where the principles of training models, recognizing patterns, and making predictions are applied.

Additionally, the development of the web application interface falls under the scope of Web Engineering, where the focus is on designing, developing, and deploying user-friendly web systems.

Through the integration of these core concepts, the project not only builds on theoretical knowledge but also emphasizes practical application in real-world scenarios.

| Concept Used | Relevance to Course |
|------------------------|----------------------------|
| Database | Database Management System |
| Machine Learning | Artificial Intelligence |
| Web Application Design | Web Engineering |

Table 1.1 Concept Relevance to Course Module

Project Background

The demand for efficient web prototyping tools is ever-increasing as web design and development processes become more integrated. Traditional methods of sketching out designs and then manually converting them into code can be time-consuming, requiring skilled developers to translate artistic vision into functional websites. This project stems from the idea of merging hand-drawn creativity with automation to eliminate redundant tasks and simplify the design-to-development workflow.

The project draws inspiration from various domains, including the rise of AI-driven design tools and existing object recognition models. The goal is to create a seamless process where a designer can sketch a wireframe by hand, upload it, and receive fully functional HTML code that serves as the foundation for further development. This solution is designed to accelerate prototyping, allowing for rapid iteration, which is crucial in fast-paced development environments.

Literature Review

Numerous advancements in machine learning and object detection have contributed to the foundation of this project. Studies on convolutional neural networks (CNNs) and their ability to recognize patterns in images have laid the groundwork for recognizing hand-drawn HTML elements. Research by scholars on transfer learning and model fine-tuning also supports the approach used in this project to optimize pre-trained models for the specific task of detecting HTML elements.

Additionally, existing AI-powered design tools, such as sketch-to-code platforms, have demonstrated the feasibility of automating design workflows. However, these tools often focus on more structured input, such as digital designs, whereas this project pushes the boundary by incorporating free-hand sketches, an area less explored in literature. Further, most existing tools generate only static representations, whereas this project generates actual HTML code, bringing the design to life.

Methodology and Software Lifecycle

This project follows the Agile software development lifecycle to ensure continuous integration and feedback during the process. consisting of the following stage.

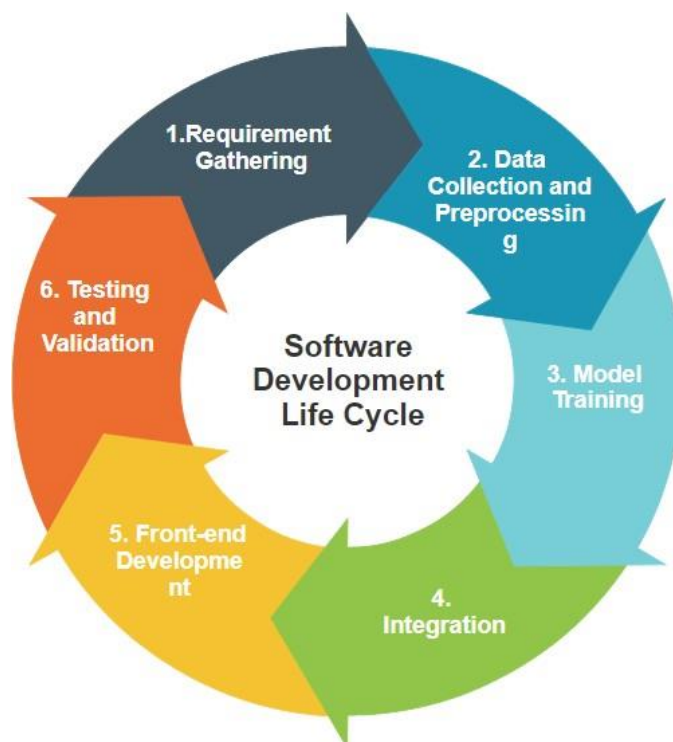


Figure 1 Lifecycle

Requirement Gathering

The first phase involves comprehensive discussions with potential users, including web designers and developers, to identify their needs and pain points in the current prototyping process. This stage entails conducting surveys and interviews to gather insights on desired features, usability requirements, and performance expectations. This information helps define the core functionalities of the tool, such as hand-drawn HTML element recognition, code generation capabilities, and user interface design.

Data Collection and Preprocessing

Data is critical for training the machine learning models. The project involves curating a dataset of hand-drawn HTML elements, sourced from open repositories and user-generated content. This dataset undergoes preprocessing, which includes several steps:

- **Image Resizing:** Standardizing the size of the images to ensure uniformity during model training.
- **Normalization:** Adjusting pixel values to a common scale to improve model training efficiency.
- **Annotation:** Labeling the images to indicate the specific HTML elements they represent. This step is essential for supervised learning, as it guides the model in identifying and classifying elements accurately.

The preprocessing stage ensures the dataset is clean, relevant, and suitable for training the object detection model.

Model Training

With the prepared dataset, the next step involves training the machine learning model. The project utilizes the **YOLOv8** architecture, a state-of-the-art object detection model known for its speed and accuracy. Key aspects of this stage include:

- **Training and Validation:** The dataset is split into training and validation sets, allowing the model to learn from one portion while evaluating its performance on another. This helps to prevent overfitting and ensures the model generalizes well to new data.
- **Hyperparameter Tuning:** Adjusting parameters such as learning rate, batch size, and the number of training epochs to optimize model performance.
- **Evaluation Metrics:** The model's effectiveness is measured using metrics such as precision, recall, and F1-score, which indicate how well the model detects and classifies hand-drawn elements.

This iterative training process continues until the model achieves satisfactory accuracy and reliability in detecting the target elements.

Integration

Once the model is trained and validated, it is integrated into the application using a Flask-based API. This API serves as a bridge between the front-end interface, built with ReactJS, and the backend model. Key activities during integration include:

- **API Development:** Designing endpoints that allow the front end to send hand-drawn images to the model for processing and receive the generated HTML code in response.
- **Server Management:** Ensuring the server can handle multiple requests efficiently, maintaining performance and responsiveness for users.

The integration phase ensures seamless communication between the user interface and the underlying machine learning model.

Testing and Validation

Testing is a crucial phase that ensures the application functions correctly and meets user expectations. This phase consists of:

- **Automated Testing:** Developing unit tests and integration tests to verify that individual components work as intended and that the entire system operates cohesively.
- **User Acceptance Testing (UAT):** Involving real users to evaluate the application's usability, functionality, and performance. Feedback collected during UAT is used to identify areas for improvement.

The goal of this phase is to ensure the tool is reliable, accurate, and easy to use, addressing any issues before deployment.

Deployment

Upon successful testing, the application is deployed in a cloud environment to ensure accessibility for users. The deployment process includes:

- **Continuous Deployment:** Implementing practices that allow for regular updates and improvements based on user feedback and performance metrics.
- **Monitoring and Maintenance:** After deployment, the system is continuously monitored for performance issues and user activity. Regular maintenance ensures the application remains up-to-date and functional.

This phase guarantees that the application is accessible to users and performs well in a real-world environment.

| Life Cycle | Description |
|-----------------------------------|---|
| Requirement Gathering | Understanding the needs of designers and developers, and defining the key features of the tool (hand-drawn HTML recognition, code generation, and UI/UX). |
| Data Collection and Preprocessing | The dataset for this project includes hand-drawn HTML sketches, sourced from open repositories and custom data collection. Preprocessing involved resizing, normalizing, and annotating the images to create a labeled dataset for model training. |
| Model Training | Machine learning models are trained using frameworks like TensorFlow and PyTorch, focusing on object detection algorithms (such as Faster R-CNN). The models are fine-tuned using transfer learning techniques to specialize in detecting hand-drawn HTML elements. |
| Integration | A Flask-based API serves as the communication layer between the front-end (ReactJS) and the machine learning model. The Node.js backend handles routing, server management, and user requests, ensuring that inputs are processed efficiently. |

| | |
|------------------------|---|
| Front-end Development | The ReactJS interface allows users to upload their sketches, receive generated HTML code, and interact with the results. Feedback mechanisms are implemented to allow users to refine their designs. |
| Testing and Validation | The system is continuously tested using automated and manual approaches. The accuracy of the machine learning model is evaluated based on its ability to correctly identify and generate code for hand-drawn elements. User testing is also conducted to ensure the UI is intuitive and the code generated is useful for further development. |
| Deployment | The system will be deployed in a cloud environment, making it accessible via a web application. Continuous deployment practices are followed to integrate updates based on user feedback and performance evaluations. |

Table 2 Lifecycle

By following this comprehensive methodology and software lifecycle, the Hand-Drawn HTML Element Code Generator project effectively combines machine learning and web development, resulting in a valuable tool for designers and developers that streamlines the prototyping process.

CHAPTER 2

PROBLEM DEFINITION

Problem Statement

The rapid evolution of web design and development has led to an increased demand for efficient prototyping tools that can bridge the gap between creative concepts and functional code. However, designers often struggle with the cumbersome process of converting hand-drawn sketches of HTML elements into actual code, resulting in wasted time and resources. Traditional methods are not only labor-intensive but also prone to errors, hindering rapid iteration and the creative process. The Hand-Drawn HTML Element Code Generator aims to resolve these challenges by employing advanced object detection and machine learning techniques to automate the identification of hand-drawn HTML elements and generate corresponding HTML code accurately. This solution streamlines the workflow for designers and developers, enabling faster prototyping and more efficient collaboration, ultimately enhancing productivity in web development projects.

Deliverables and Development Requirements

| Deliverable | Development Requirements |
|--|--|
| Trained model for recognizing hand-drawn HTML elements | Machine learning framework (e.g., TensorFlow, PyTorch) |
| Web application interface for user interaction | Front-end development using ReactJS |
| API for backend communication | Flask for API development |
| Database for storing sketches and generated code | MongoDB for data storage |

Table 3 Deliverables

Current System

The existing system for the **Hand-Drawn HTML Element Code Generator** relies heavily on manual processes, resulting in inefficiencies in the design-to-code workflow. Designers typically create hand-drawn sketches of HTML elements using paper or digital tablets, which capture their design ideas without adhering to any standardized format. This method is not only labor-intensive but also prone to inaccuracies when transitioning from sketches to code.

Currently, there is no automation in place to detect and interpret these sketches into functional HTML code. As a result, designers must invest significant time and effort in manually rewriting their designs in code, often leading to inconsistencies and errors. This lack of automated tools creates a bottleneck in the workflow, hindering rapid prototyping and iteration.

The fragmented nature of the current workflow further complicates collaboration between designers and developers. The separation of tools for sketching and coding means that the overall process becomes disjointed, slowing down project timelines and making it challenging to iterate quickly based on feedback. Consequently, designers find themselves in a cycle of inefficiency, unable to bring their ideas to life as swiftly as desired.

Additionally, the current system does not facilitate quick iterations, as each design must be manually coded. This delay in the feedback loop can significantly slow down the overall development process, making it difficult for teams to adapt to changing requirements or to incorporate new ideas. Moreover, designers struggle to access tools or platforms that streamline the conversion of hand-drawn sketches into code, further impacting their productivity and limiting their ability to effectively implement their creative visions.

In summary, the current system for converting hand-drawn HTML elements into code is characterized by a lack of automation, inefficiency, and integration. The **Hand-Drawn HTML Element Code Generator** seeks to address these challenges by leveraging machine learning and computer vision technologies to automate the detection and coding process, ultimately enhancing productivity and streamlining workflows.

CHAPTER 3

REQUIREMENT ANALYSIS

We explain the workflow for converting hand-drawn images to HTML code. The process starts with the *User/Customer* logging in or signing up, followed by creating a chat session. Once logged in, the user can upload a hand-drawn image containing UI elements. The system then uses machine learning and OCR to recognize these elements and identify their HTML equivalents. After recognition, the system generates the HTML code based on the identified elements. This code is then exported to an internal compiler for processing and displayed in a preview mode, allowing the user to review it. Finally, the generated code is sent to the server, completing the end-to-end process from sketch to HTML.

Use Case Diagrams

Website:

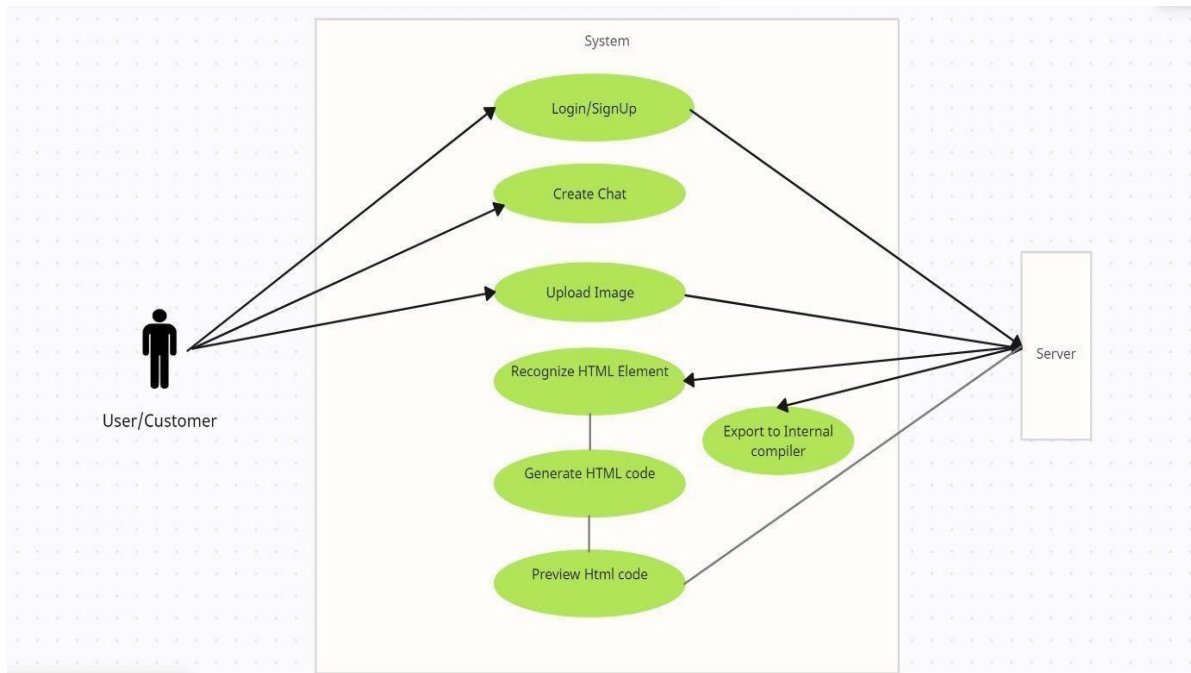


Figure 2 Website Use Case

Description:

Actors:

User: This is the primary actor who interacts with the system to log in, create chats, upload images, and perform various operations like recognizing HTML elements, generating HTML code, and previewing the generated code.

Server: This actor represents the back-end system that processes requests sent by the user.

Use Case and Flow:

1. **Login/Signup:**
 - The user begins by either logging into or signing up for the system. This is the first point of contact between the user and the server, establishing a session for further actions.
2. **Create Chat:**
 - Once logged in, the user can create a chat session within the system. This action connects to the server to create and maintain the chat session.
3. **Upload Image:**

- The user uploads an image (likely containing HTML or other elements) to the system, which is then forwarded to the server for further processing.
- 4. **Recognize HTML Element:**
 - The system processes the uploaded image to detect and recognize any HTML elements within the image. This could involve image recognition or OCR (Optical Character Recognition) to identify text or code elements.
 - The recognized elements are then sent to the server for verification or further processing.
- 5. **Generate HTML Code:** ○ Based on the recognized HTML elements, the system generates actual HTML code.
This code is either generated locally or sent to the server for code generation.
- 6. **Preview HTML Code:**
 - The user can preview the generated HTML code before exporting it for further use or saving it. This step provides feedback to the user to check if the recognized HTML elements and generated code meet their expectations.
- 7. **Export to Internal Compiler:**
 - The system allows the user to export the generated HTML code to an internal compiler for further usage, potentially for deploying the HTML in a live environment or for additional processing on the server.

Use case name and identifier (Use

Case Narrative)Web application:

Main Menu:

| | | |
|----|--|-----------------|
| 1. | Use Case Name: Main Menu | |
| 2 | Implementation Priority: 1 | |
| 3 | Actors: Users | |
| 4 | Summary: Users can upload images or code snippets, and the system extracts relevant information, generates HTML or other code formats, and then offers further processing such as recognizing elements or previewing the generated code. | |
| 5 | Pre-condition: The user must be logged into the web application. | |
| 6 | Post-condition: The system displays the generated code (HTML or other formats), and allows the user to preview or export the generated code for further use. | |
| 7 | Extend: - | |
| 8 | Include: - | |
| 9 | Normal Course of Events: | |
| | System Actor Action | System Response |

| | | |
|----|--|---|
| | 1. User uploads an image or code snippet. 3. System generates corresponding code. 5. User selects further processing (e.g., preview or export). | 2. The system processes the input and extracts relevant elements (e.g., HTML elements from an image). 4. The generated code is displayed. 6. The system performs the selected action and displays the result (e.g., preview or code export). |
| 10 | Alternative Path: If the uploaded image or code snippet is invalid, the system will display an error message. | |
| 11 | Exception: If the internet is down or the server times out, the system will fail to process the request, and the user will be notified. | |

Table 4 Code generation

Description:

The "**Main Menu**" use case allows users to interact with the web application by uploading images or code snippets. Once an image or code snippet is provided, the system processes the input by identifying and extracting relevant elements, such as HTML structures or other code components. The extracted elements are then used to generate functional HTML code, which is displayed to the user.

Users can also choose additional processing options, such as **previewing** the generated code or **exporting** it for further use. The system ensures that any generated code is well-structured and ready for immediate application or modification.

In case an invalid image or code snippet is uploaded, the system provides an error message. Furthermore, the system accounts for potential exceptions like internet connectivity problems or server timeouts, offering feedback to users and ensuring reliable system performance.

Functional Requirements

1. **Sketch Uploading** ○ The system shall allow users (designers) to upload hand-drawn sketches of HTML elements in various image formats (e.g., PNG, JPG). ○ The system shall validate the uploaded files to ensure they meet size and format requirements.
2. **Element Detection** ○ The system shall utilize an object detection model (e.g., YOLOv8) to identify and classify hand-drawn HTML elements from the uploaded sketches. ○ The system shall provide feedback on the detection accuracy, indicating which elements were successfully detected and any that were not.

3. **Code Generation** ○ The system shall generate corresponding HTML code based on the detected elements using predefined templates. ○ The generated HTML code shall be formatted correctly and be compatible with standard web development practices.
4. **User Interface** ○ The system shall provide an intuitive user interface for designers to interact with, allowing easy navigation between uploading sketches, viewing detected elements, and accessing generated code. ○ The interface shall display the original sketch alongside the detected elements and the generated HTML code.
5. **Code Download** ○ The system shall enable users (developers) to download the generated HTML code for integration into their projects. ○ The downloaded code shall be packaged in a user-friendly format (e.g., a HTML file) containing all relevant files.
6. **Performance Requirements** ○ The system shall process uploaded sketches and generate code within a reasonable time frame (e.g., less than 5 seconds for most sketches). ○ The object detection model shall achieve a minimum accuracy threshold (e.g., 85%) on a validation dataset to ensure reliability.

These functional requirements highlight the essential features and functionalities that the Hand-Drawn HTML Element Code Generator must implement to achieve its objectives effectively.

Non-Functional Requirements

1. Performance

- The system shall respond to user inputs and process uploaded sketches within an acceptable time frame (e.g., less than 5 seconds for most sketches).
- The system shall efficiently handle multiple concurrent users without significant performance degradation.

2. Scalability

- The system shall be designed to scale horizontally, allowing for additional resources to be added as user demand increases.
- The architecture should support a growing number of users and increased data volume without compromising performance.

3. Usability

- The user interface shall be intuitive and user-friendly, allowing users with varying technical backgrounds to easily navigate the application. ○ The system shall provide clear instructions and feedback to guide users through the process of uploading sketches and retrieving generated code.

4. Security

- The system shall implement security measures to protect user data and uploaded files, including secure file handling and storage practices.
- User authentication shall be required to access sensitive features, ensuring that only authorized users can upload sketches and download generated code.

5. Compatibility

- The system shall be compatible with modern web browsers (e.g., Chrome, Firefox, Safari) to ensure accessibility for all users.
- The generated HTML code shall be compliant with HTML5 standards and compatible with popular web development tools and frameworks.

6. Maintainability

- The system shall be designed for ease of maintenance and updates, with modular components that allow for isolated testing and deployment.
- Documentation shall be provided to support future developers in understanding the system architecture and codebase.

7. Reliability

- The system shall ensure high availability and minimize downtime, implementing redundancy and backup mechanisms to prevent data loss.
- The object detection model should be robust enough to handle variations in hand-drawn sketches and provide consistent performance.

These non-functional requirements outline the quality attributes and constraints that the Hand-Drawn HTML Element Code Generator must meet to provide a reliable, efficient, and user-friendly experience.

Software Quality Attributes:

Maintainability:

Maintainability refers to the ease with which the system can be updated, modified, and repaired. The Hand-Drawn HTML Element Code Generator is designed with modular architecture, allowing developers to work on individual components without impacting the entire system. This modularity facilitates testing and deployment, making it easier to fix bugs and implement new features. Furthermore, comprehensive documentation will be provided to guide future developers, detailing system architecture, coding standards, and best practices. The use of version control systems will also aid in tracking changes and managing updates efficiently.

Usability:

Usability is critical for ensuring that users can effectively interact with the system. The application will feature an intuitive and user-friendly interface, designed to accommodate users of varying technical expertise. Clear instructions and prompts will guide users through uploading hand-drawn sketches and retrieving the corresponding HTML code. To enhance the user experience, feedback mechanisms will be implemented, such as progress indicators during processing and error messages that clearly describe issues. Regular user testing and feedback sessions will be conducted to identify areas for improvement, ensuring that the system meets the needs and expectations of its users.

Correctness:

Correctness is crucial in ensuring that the Hand-Drawn HTML Element Code Generator performs as expected and delivers accurate results. To achieve this, the system will rely on advanced object detection algorithms, with YOLOv8 being chosen for its accuracy and efficiency in detecting hand-drawn HTML elements. YOLOv8's robust capability to recognize different shapes and structures of HTML elements, such as buttons, text fields, checkboxes, and containers, will allow it to accurately map these hand-drawn components to their corresponding HTML tags.

In addition to object detection, the correctness of the system will be maintained through a comprehensive testing framework. Unit testing will focus on validating each individual component of the system, including both the object detection and HTML generation modules, ensuring they perform their tasks without errors. Integration testing will evaluate the system as a whole, confirming that different components—like the object detection algorithms and code generation processes—interact smoothly and produce the correct output when combined.

System testing will simulate real-world scenarios by providing a variety of hand-drawn sketches to verify the system's overall functionality, ensuring it consistently translates drawings into accurate HTML code. The HTML generated will undergo validation checks to ensure it is free from syntax errors, adhering to the structure and requirements of HTML5 standards. This validation step is essential to guarantee that the output is production-ready and can be seamlessly integrated into web development workflows.

The system will not only focus on initial accuracy but will also implement continuous monitoring and testing to ensure long-term correctness. As the system evolves, updates or changes in its components, such as improvements in object detection models or HTML generation techniques, will be rigorously tested to prevent regressions or issues. Monitoring tools will track the performance and accuracy of the system, allowing for prompt detection and correction of any emerging issues. This approach ensures that the system remains reliable and maintains a high level of correctness even as it adapts to new challenges or expands its capabilities.

To further strengthen correctness, the system may also integrate automated feedback loops, where any errors or incorrect code generation outcomes are analyzed and used to fine-tune the object detection and code generation algorithms. This continual learning process ensures that the system improves over time, becoming more accurate and reliable with each iteration.

In summary, the Hand-Drawn HTML Element Code Generator's focus on correctness is achieved through the combination of YOLOv8's powerful object detection, extensive testing across all system layers, adherence to HTML5 standards, and ongoing monitoring to maintain accuracy throughout the system's lifecycles.

CHAPTER 4

DESIGN AND ARCHITECTURE

4.1.1. System Architecture

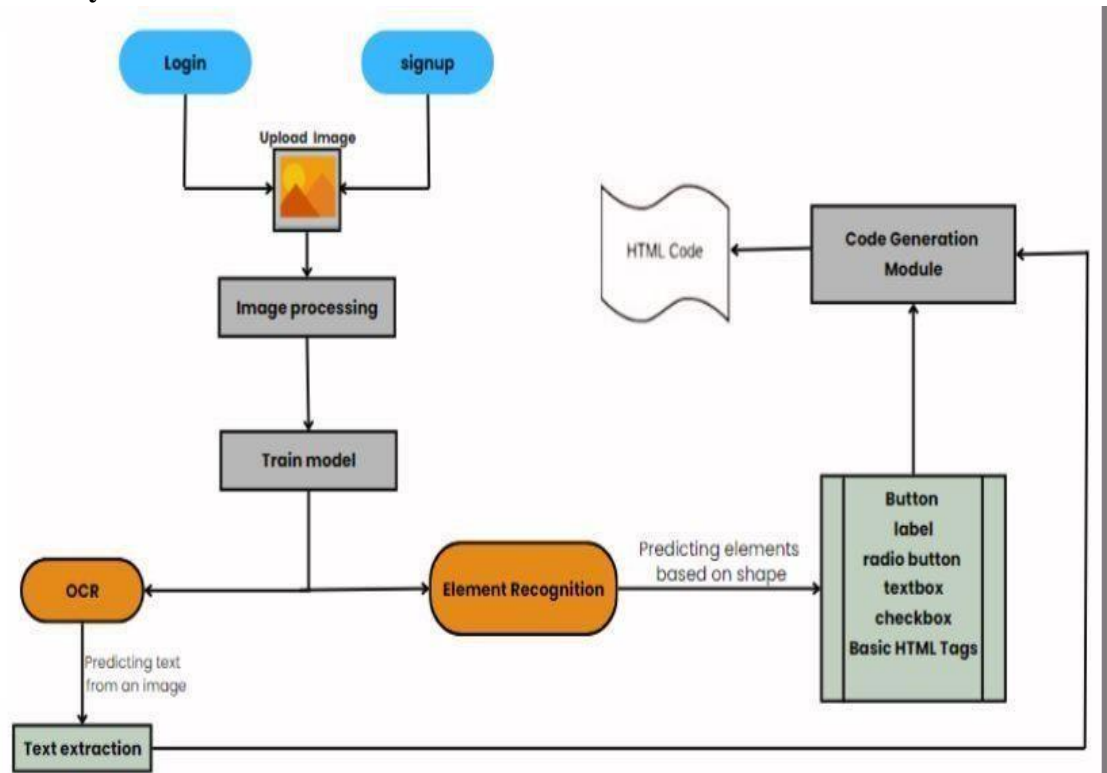


Figure 3 System Architecture

The **Code Generation System's** architecture is designed to efficiently process user inputs, such as uploaded images, and generate corresponding HTML code. This system integrates various modules, ensuring smooth interaction between the user interface, backend processing, and machine learning models for element recognition and code generation.

1. User Interface (UI):

The **user interface (UI)** acts as the front-end portal where users can log in or sign up, providing functionality to upload images that contain HTML-like elements, such as buttons, forms, or text fields. The UI is built using standard web technologies like HTML, CSS, and JavaScript to ensure compatibility with web browsers. Once the user uploads an image, it is sent to the backend for processing.

2. Backend Processing:

In the **backend processing** stage, the system first processes the image. This may involve resizing, filtering, and analyzing the image to identify relevant regions where

HTML elements might be located. The core of the system relies on a trained machine learning model that identifies and classifies different UI elements, such as buttons, checkboxes, and input fields. This model is trained using a dataset of various HTML components and their visual representations, predicting what the elements in the image correspond to in terms of HTML tags...

3. Optical Character Recognition:

Alongside identifying UI elements, the system extracts text from the image using **Optical Character Recognition (OCR)**. OCR converts visually displayed text (such as button labels or form field placeholders) into machine-readable text. This ensures that labels and placeholder values are preserved in the generated HTML. For example, if the image contains a button labeled "Submit," the OCR module extracts "Submit" as text, which will later be embedded within the HTML `<button>` tag.API Layer:

4. Element Recognition:

The system uses a combination of image analysis techniques and deep learning in **Element recognition** to recognize HTML elements based on their shapes and layouts.

Circular shapes might represent radio buttons, while rectangular shapes with labels could indicate text input fields. Based on the identified shapes, the system predicts

Which HTML elements the image corresponds to, such as `<button>`, `<input>`, or `<form>`

5. Code Generation Module:

Once the system has recognized the UI elements and extracted any text, the **Code Generation Module** takes over. This module generates clean, structured HTML code corresponding to the identified elements. For instance, if the system recognizes a form with a button and a text field, it generates a simple form structure with the relevant tags. The code includes basic HTML tags such as `<button>`, `<input>`, `<form>`, and `<label>`, depending on the identified elements.

4.1.2. Data Flow

The data flow within the AI-Powered Code Generation System is designed for efficient processing and minimal latency. Below is the step-by-step flow of data through the system, aligning with the architecture shown in the diagram?

1. User Interaction:

The user begins interaction with the system through the **User Interface (UI)**, which is built using HTML, CSS, and JavaScript. The UI allows users to upload images (such as screenshots of web components or UI designs) that need to be converted into HTML code. The interface provides options for users to log in or sign up and ensures an intuitive experience for seamless navigation and interaction.

2. Request Handling:

Once the user uploads an image, the **Request Handling Component** within the UI forwards the image data to the backend system. This component ensures that the user's request is accurately captured and transmitted to the backend,

managing the connection between the frontend and backend to enable smooth data flow.

3. API Processing:

Upon receiving the request, the **API Layer** serves as the primary interface between the UI and the backend processing modules. The API Layer routes the request (in this case, the uploaded image) to the appropriate backend module, such as the **Image Processing** or **Code Generation Module**, ensuring the request reaches the correct destination for further processing.

4. Backend Processing:

Once the image is received, the **Backend Processing** begins. First, the image undergoes processing, such as resizing and filtering, to identify key regions containing UI components. The **Machine Learning Model** analyzes the processed image, detecting UI elements like buttons, input fields, and forms. Simultaneously, **OCR (Optical Character Recognition)** extracts any visible text from the image. The **Element Recognition Module** then classifies and predicts the corresponding HTML elements based on the image's visual content. Once all elements and text are identified, the **Code Generation Module** constructs the appropriate HTML code. The backend system is optimized for high-performance, allowing it to quickly process the image and generate accurate HTML.

5. Response to User:

After the code is generated, the **Backend** sends the data back to the API Layer, which communicates with the frontend. The **UI** then presents the generated HTML code to the user in an organized format, offering a **Preview** option for users to validate the code against their design. If satisfied, users can export the HTML code to their environment. This concludes the data flow, ensuring that the user's request is handled efficiently from input to the final HTML output.

4.1.4 Database and Data Storage

The Database and Data Storage component is critical for managing and storing the large amount of data processed by the Code Generation System. This includes user-uploaded images, generated HTML code, element recognition results, and system logs. By leveraging a robust database management system, the platform can ensure data security, integrity, and quick retrieval, thereby maintaining smooth system performance and user experience..

Database Schema Overview:

Image Data:

The **Image Data** section stores all uploaded images along with their associated metadata. It includes:

- Image file path or URL
- File type (e.g., PNG, JPG)
- Dimensions (e.g., resolution)
- Timestamps (upload time and processing time)

Efficient storage of this metadata allows the system to quickly retrieve images for processing or reprocessing. It is essential for linking images to the generated HTML code and recognition results, enabling effective tracking of user uploads.

Generated HTML Code:

The Generated HTML Code section stores the system's output. For every uploaded image, the corresponding HTML code generated by the system is stored here. This includes generated HTML snippets, extracted text from OCR, recognized HTML elements, and associations between images and generated code. This structured data allows users to access and reuse previously generated HTML, make updates, and track changes. It also supports version control, enabling easy rollback or comparison of different code versions..

Element Recognition Results:

This part of the schema logs the data related to the element recognition process, such as detected HTML elements, prediction confidence scores from the machine learning model, and shape and layout data from the image. This data is useful for evaluating the performance of the element recognition models, tracking which elements were detected, and improving the accuracy of future predictions through retraining.

System Logs:

The System Logs section records all system activities and user interactions, including user actions, system responses, processing times, and error reports. These logs are critical for troubleshooting and performance analysis. They allow system administrators to monitor performance, track any issues, and optimize system components. Additionally, they provide useful data for trend analysis and user behavior insights.

Data Security:

Encryption:

To protect sensitive data, all stored and transmitted data is encrypted. This ensures that any user-uploaded images, generated HTML code, and logs are secure from unauthorized access. Encryption plays a vital role in maintaining data confidentiality, preserving the integrity of user data, and complying with security standards..

Access Control:

Strict access controls are enforced to ensure that only authorized users can view, modify, or delete sensitive data. This limits unauthorized access to the database and ensures that user-generated content and other critical system data are protected. Role- based access controls ensure that only developers, admins, or authorized personnel can access key areas of the system.

4.1.5. External APIs

The **Code Generation System** utilizes external APIs to enhance its ability to convert images of UI designs into HTML code efficiently. These external APIs help streamline

processes that are essential for image recognition, element extraction, and code generation.

Integration Points:**Optical Character Recognition (OCR) API:**

The system uses an external OCR API to extract text from uploaded images. This ensures that any visible text in UI elements, such as labels on buttons or placeholders in input fields, is accurately captured and integrated into the generated HTML code. Using a reliable OCR API allows the system to handle various fonts and text arrangements, providing high accuracy in text extraction.

Image Recognition API:

To detect and classify UI elements within an image, the system integrates with advanced image recognition APIs. These APIs analyze the uploaded image to identify common web elements (e.g., buttons, input fields, and checkboxes) based on shape, size, and position. This external API allows the system to support a wide range of UI layouts and design patterns.

API Integration and Management:

The system efficiently manages API calls during the backend processing of images. It ensures smooth communication between the frontend, backend, and external APIs, ensuring accurate results with minimal latency. This design also includes error handling for scenarios like failed API requests or timeout issues, providing a seamless user experience.

4.1.6. Process Flow Representation

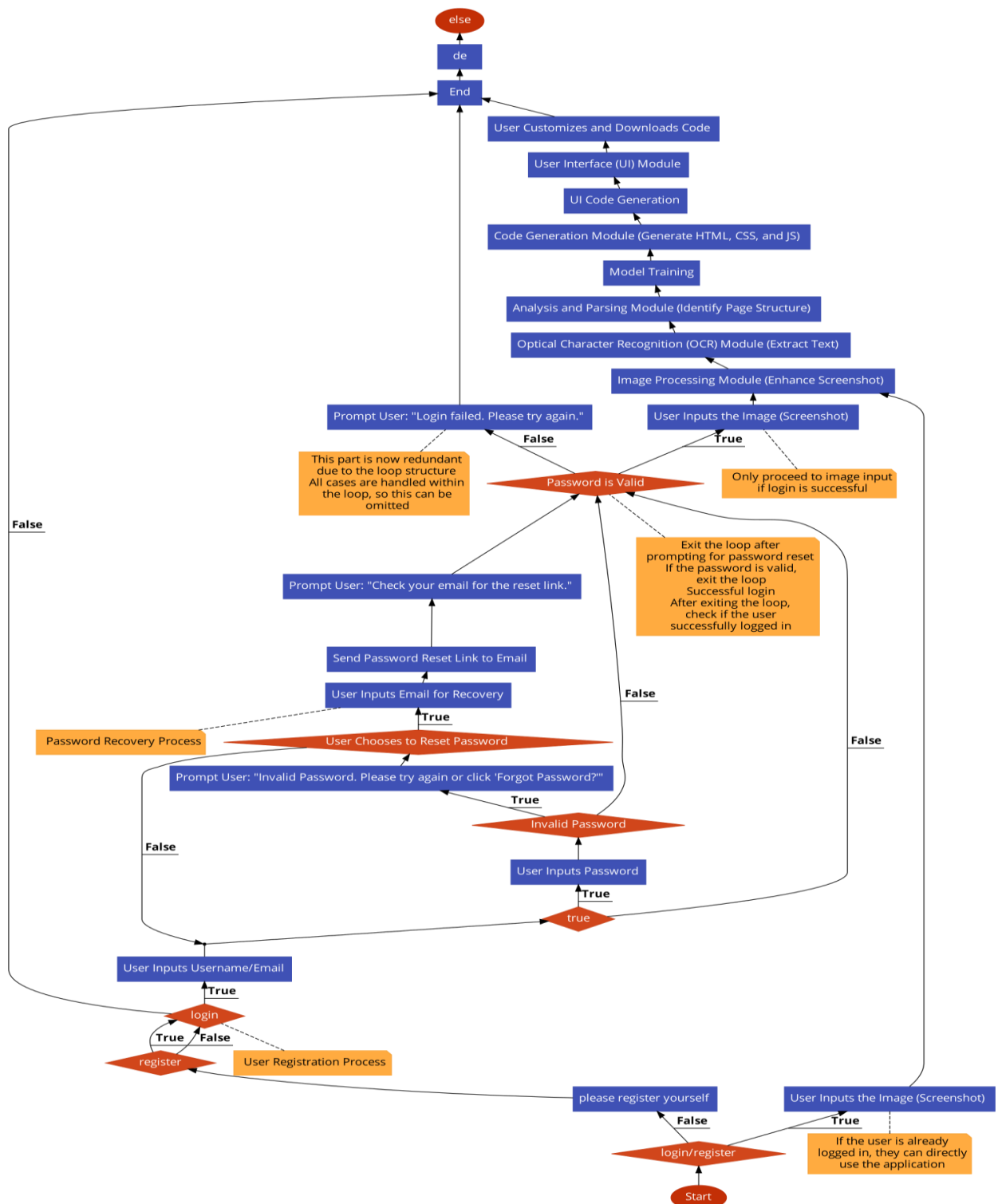


figure 6 Process flow chart

Description:

1. **Start:** The user starts by either logging in or registering for an account.
2. **Login/Registration:**

If the user is already registered, they can log in using their username and password.

If the user is new, they can register by providing their details.

3. User Input: Once logged in, the user inputs a hand-drawn HTML element by providing an image of it.

4. Image Processing:

The application analyzes the image and extracts the relevant data like text and shape information.

5. Code Generation: The application generates HTML code based on the extracted information from the image.

6. UI Code Generation: The application provides a user interface where the user can preview and customize the generated code.

7. Download: The user can download the generated code.

8. Password Recovery:

If the user forgets their password, they can initiate the password recovery process.

The user provides their registered email address.

The system sends a reset link to their email.

The user clicks the link and sets a new password.

9. Feedback:

If there are any errors or issues during the process, the user is notified and provided with options to retry or seek help.

10. End: The user can exit the application.

This flowchart outlines the core functionality of the application, providing a clear understanding of the user journey and the steps involved in converting hand-drawn HTML elements into actual code.

CHAPTER 5

5.1. IMPLEMENTATION

5.1.1. Algorithms

In the development of the **Hand-Drawn HTML Element Code Generator**, several algorithms play a crucial role in the system's functionality. These algorithms are designed to effectively handle the detection of hand-drawn HTML elements and the subsequent generation of HTML code. The following subsections detail the key algorithms employed in the project.

Object Detection Algorithm (YOLOv8)

The primary algorithm used for recognizing hand-drawn HTML elements is the YOLOv8 (You Only Look Once version 8) object detection model. YOLOv8 is renowned for its speed and accuracy in detecting objects within images, making it an ideal choice for this project.

Key Features:

- **Real-Time Detection:** YOLOv8 processes images in real-time, allowing users to receive immediate feedback on their uploaded sketches.
- **Bounding Box Prediction:** The algorithm generates bounding boxes around detected elements, providing clear indicators of where each HTML component is located in the sketch.
- **Class Prediction:** It classifies the detected elements into predefined categories (e.g., buttons, input fields, divs), facilitating the next steps in code generation.

Training Process: To train the YOLOv8 model, we utilized a dataset consisting of annotated images of various hand-drawn HTML elements. The training process involved:

1. **Data Preparation:** Cleaning and preprocessing the dataset, ensuring consistency in annotations and image sizes.
2. **Model Configuration:** Setting hyperparameters and configuring the architecture of the YOLOv8 model for optimal performance.
3. **Training:** Running the training process using a GPU-accelerated environment to expedite the learning process, allowing the model to learn the characteristics of hand-drawn elements effectively.

Code Generation Algorithm

Once the hand-drawn elements are detected, the next step is to translate these elements into HTML code. The code generation algorithm utilizes predefined templates for each detected HTML element, ensuring a consistent and accurate output.

Key Features:

- **Template-Based Approach:** Each recognized element corresponds to a specific HTML template (e.g., `<button>`, `<input>`, `<div>`), which is filled with attributes based on user specifications or default settings.
- **Dynamic Attribute Assignment:** The algorithm can dynamically assign attributes such as id, class, and style based on the context of the drawing and user preferences.

- **Code Formatting:** The generated code is formatted for readability, ensuring that it adheres to standard coding practices, which helps developers integrate the output seamlessly into their projects.

Process Flow:

1. **Element Detection:** After the YOLOv8 model identifies the elements, the system captures their class labels and bounding box coordinates.
2. **Template Selection:** The code generation algorithm selects appropriate templates based on the detected classes.
3. **Attribute Assignment:** The system assigns default or user-specified attributes to each element.
4. **Code Output:** Finally, the generated HTML code is compiled into a string format, which is returned to the frontend for user review.

Data Management Algorithm

Data management algorithms are also integral to the system, particularly for handling user-uploaded images and storing processed results.

Key Features:

- **Image Storage:** Uploaded images are stored in a MongoDB database, allowing for easy retrieval and management.
- **Result Logging:** The results of each processed image, including detected elements and generated code, are logged for future reference and analytics.
- **User Interaction Tracking:** The algorithm tracks user interactions, including uploads and modifications, to enhance the user experience through personalized features and improved feedback.

By integrating these algorithms, the Hand-Drawn HTML Element Code Generator effectively transforms user sketches into usable HTML code, enhancing the workflow of designers and developers in creating web interfaces. The combination of YOLOv8 for detection, a robust code generation mechanism, and effective data management ensures a seamless and efficient implementation.

5.1.2. External API's

In our project, we leverage various external APIs to enhance functionality and streamline data processing. These APIs provide essential services that support different aspects of our system, from accessing and managing video content to facilitating communication between components. By integrating these APIs, we ensure that our application can efficiently handle tasks such as retrieving video information, managing data exchanges, and maintaining seamless integration across various system components. Below is an overview of the external APIs used in our project:

EasyOCR

EasyOCR is an open-source Optical Character Recognition (OCR) library that provides a simple and efficient way to extract text from images. This API enhances the application by enabling users to convert handwritten or printed text in their sketches into digital text, streamlining the process of generating HTML elements from hand-drawn designs.

Key Features:

- **Multi-Language Support:** EasyOCR supports a wide range of languages, making it versatile for users around the globe who may use different writing systems.
- **High Accuracy:** The library leverages deep learning techniques to achieve high accuracy in text recognition, even in challenging conditions like varying handwriting styles or poor image quality.
- **Ease of Integration:** EasyOCR is straightforward to integrate into the existing application, requiring minimal setup and allowing for quick implementation.

By utilizing EasyOCR, the Hand-Drawn HTML Element Code Generator significantly enhances its functionality, allowing users to extract and convert text from their sketches effortlessly. This integration contributes to the system's overall usability and effectiveness, enabling users to create HTML elements from their drawings with greater ease and accuracy.

Rest API:

The Rest API (Representational State Transfer Application Programming Interface) plays a pivotal role in facilitating seamless communication between different components of our system. It allows for the efficient exchange of data between the frontend and backend systems, as well as between various microservices within the project. By using the Rest API, we ensure that data is transmitted reliably and in real-time, which is essential for maintaining the overall performance and responsiveness of the system. The API also supports scalability, enabling us to expand and integrate additional features or services as needed. This integration is key to providing a cohesive user experience and ensuring that all components of the project work together harmoniously.

5.1.3. User Interface

We created the user interface (UI) using the Django Framework, a high-level Python web framework known for its robustness and ease of use. The Django Framework was chosen for its powerful features, including its ability to handle complex data models, built-in admin interface, and extensive libraries for web development. The UI is designed to be user-friendly and intuitive, allowing users to easily upload YouTube video links, select processing modules, and view results. The interface includes components for video link input, module selection (such as Translator, Summarizer, QA Generator, and Quiz Generator), and displays for results such as translated text, summaries, Q&A pairs, and quizzes. We have incorporated responsive design principles to ensure that the UI is accessible and functional across various devices and screen sizes. The clean and organized layout helps users navigate the application with ease, improving their overall experience and interaction with the system. Below are screen captures of the application, showcasing the user interface and highlighting its key features:

Home Page:

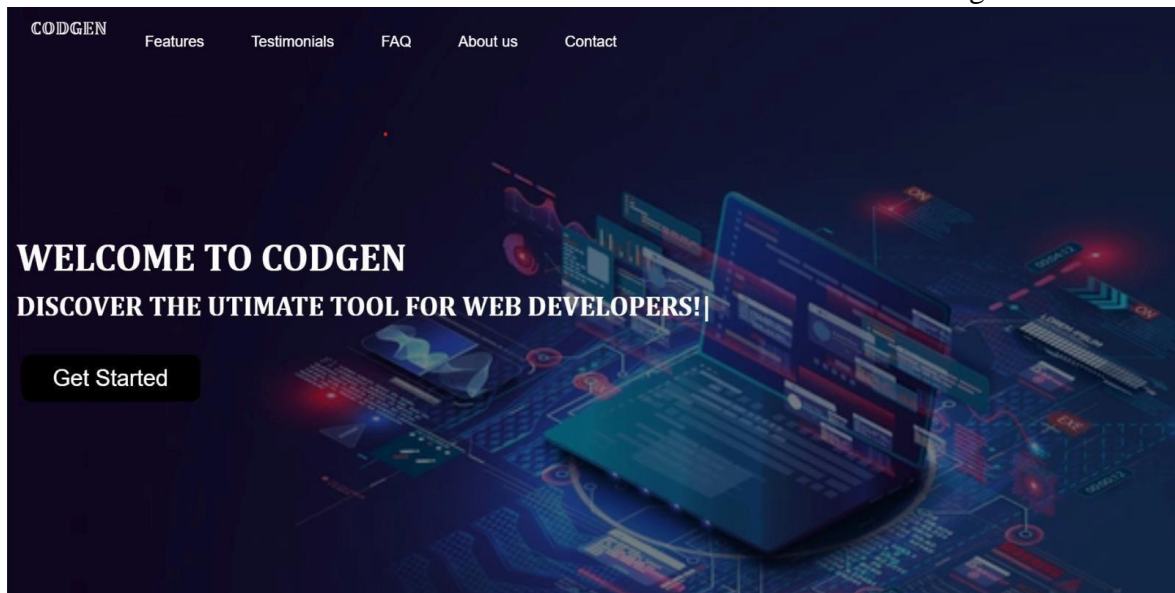


Figure 5 Home Page

Feature:

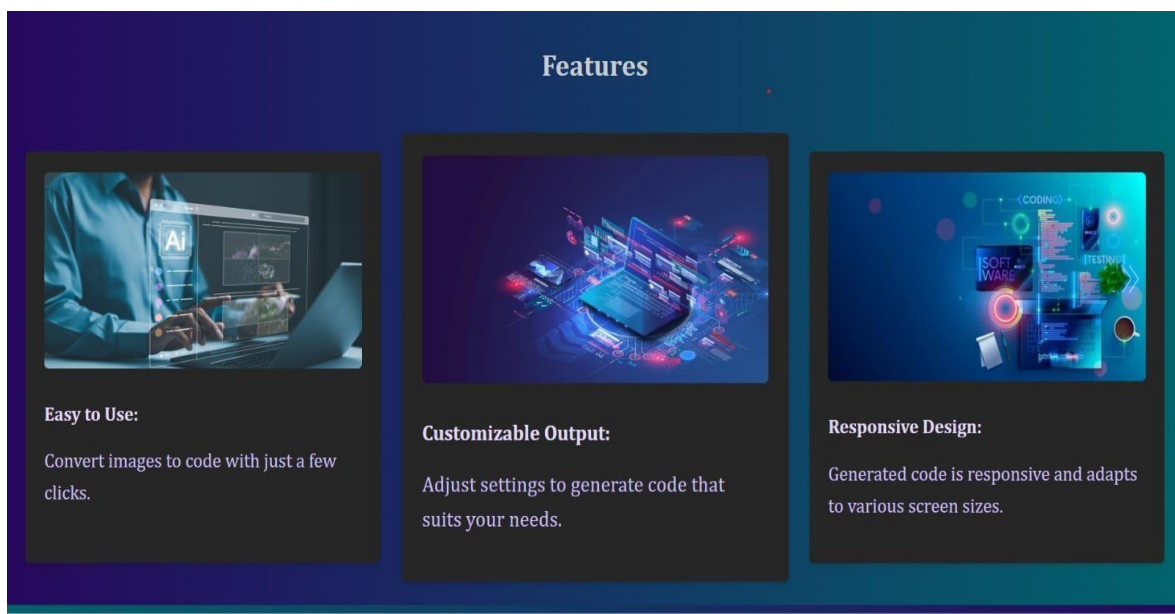


Figure 6 Feature

Testimonials:

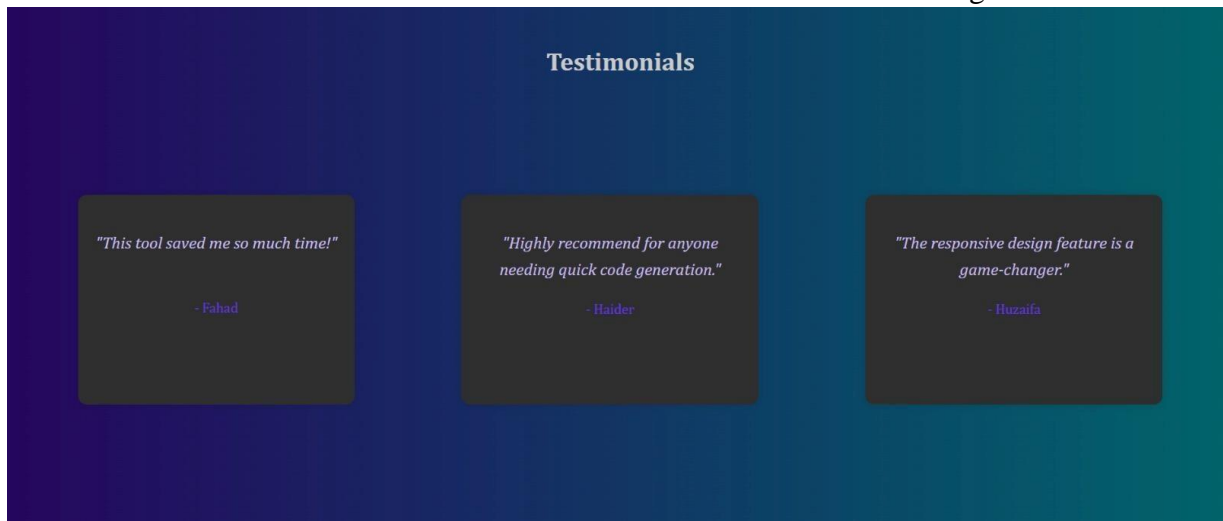


Figure 7 Testimonials

FAQ's:

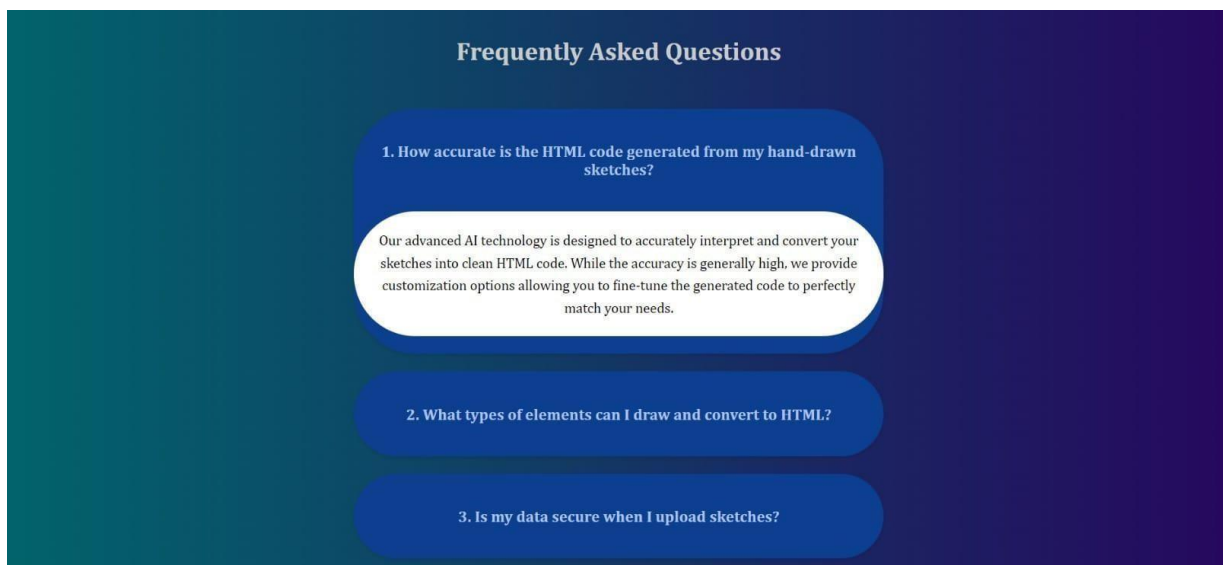


Figure 8 FAQ's

About Us:

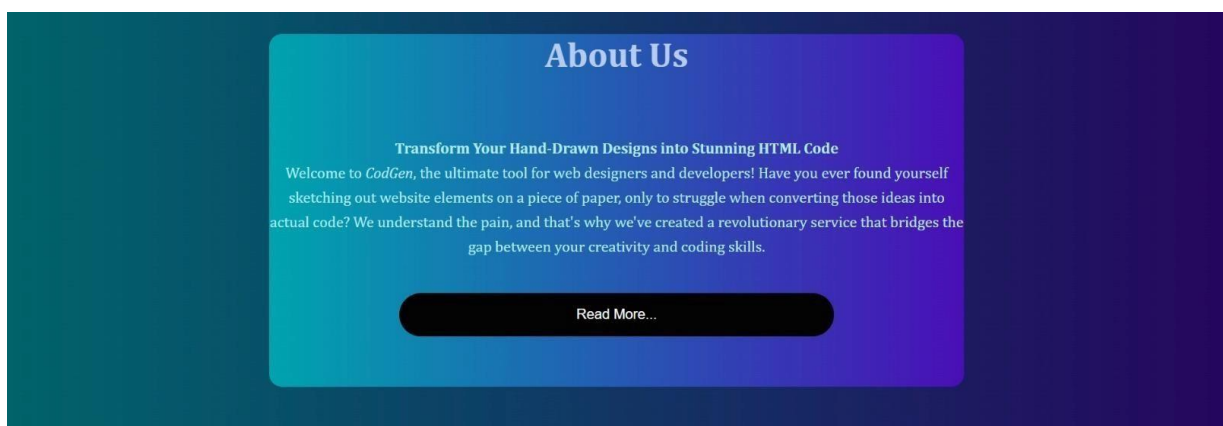
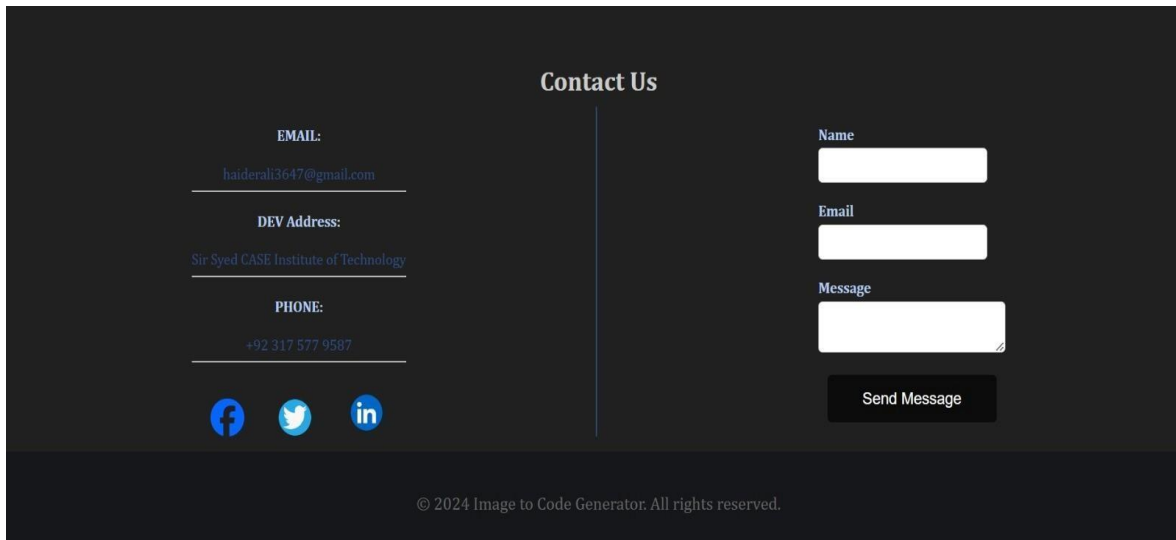
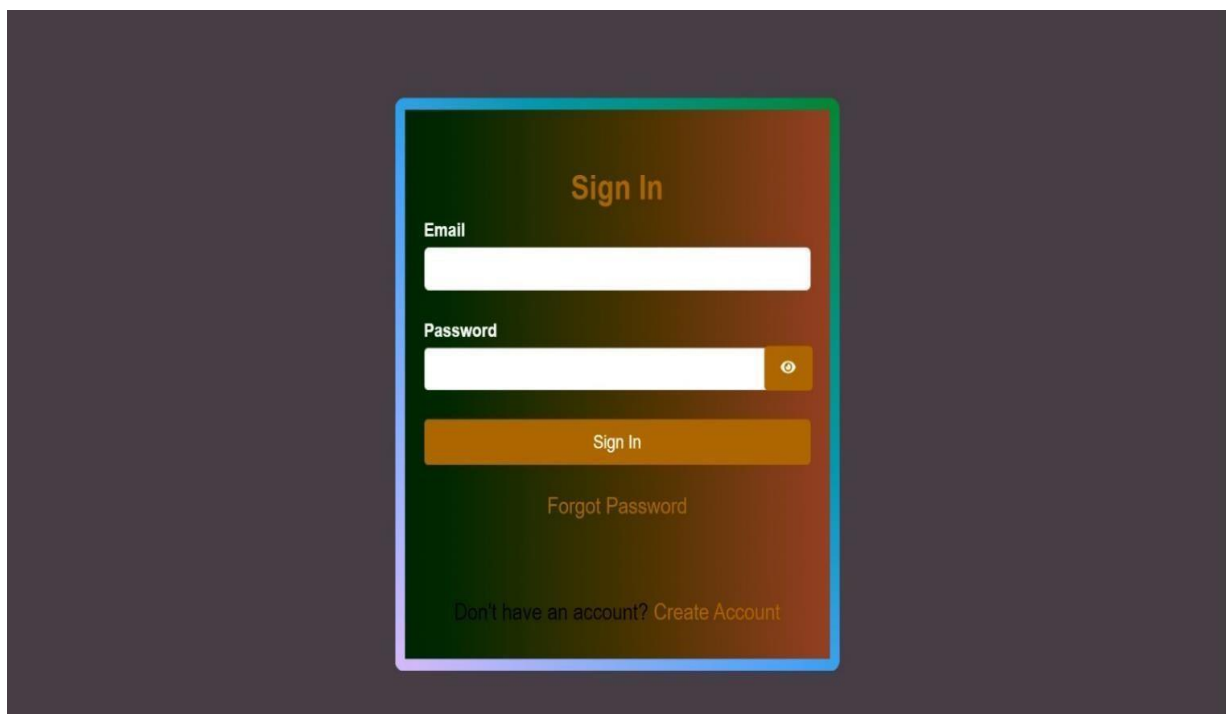


Figure 9 About Us

Contact Us:

The image shows a 'Contact Us' form on a dark background. The form is divided into two main sections by a vertical line. On the left, contact information is listed: EMAIL: haiderali3647@gmail.com, DEV Address: Sir Syed CASE Institute of Technology, and PHONE: +92 317 577 9587. Below this is a row of social media icons for Facebook, Twitter, and LinkedIn. On the right, there is a form with three input fields: Name, Email, and Message. Below these fields is a 'Send Message' button. At the bottom of the form, there is a copyright notice: © 2024 Image to Code Generator. All rights reserved.

*Figure 10 contacts us***Login:**

The image shows a 'Sign In' form on a dark background. The form is a light blue rectangle with a gradient. It has a title 'Sign In' at the top. Below the title are two input fields: 'Email' and 'Password'. The 'Password' field has a toggle icon (an eye) to its right. Below the input fields is a 'Sign In' button. Below the button is a link 'Forgot Password'. At the bottom of the form is a link 'Don't have an account? Create Account'.

*Figure 11 Login***SignUp:**

Signup

Name

Email

Password

Confirm Password

Sign Up

Already have an account? [Sign in](#)

Figure 12 Signup

Password Forget:

Forgot Password

Enter your email

Send Code

Verify Code

Figure 13 password forget

New Chat:

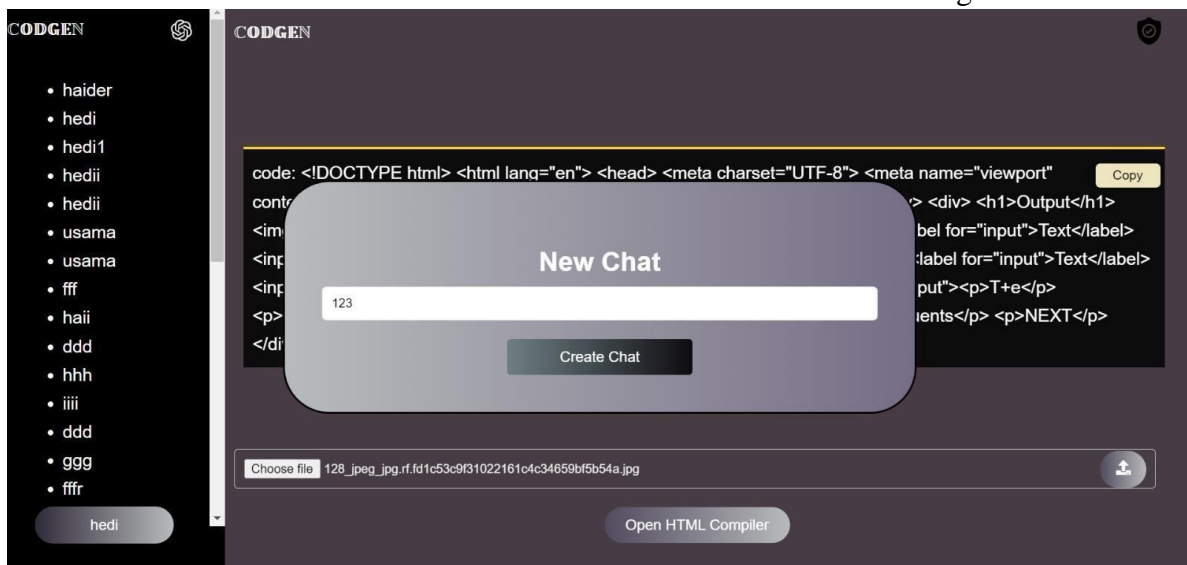


Figure 14 New Chat

Image upload:

Figure 15 image upload

Code Generate:

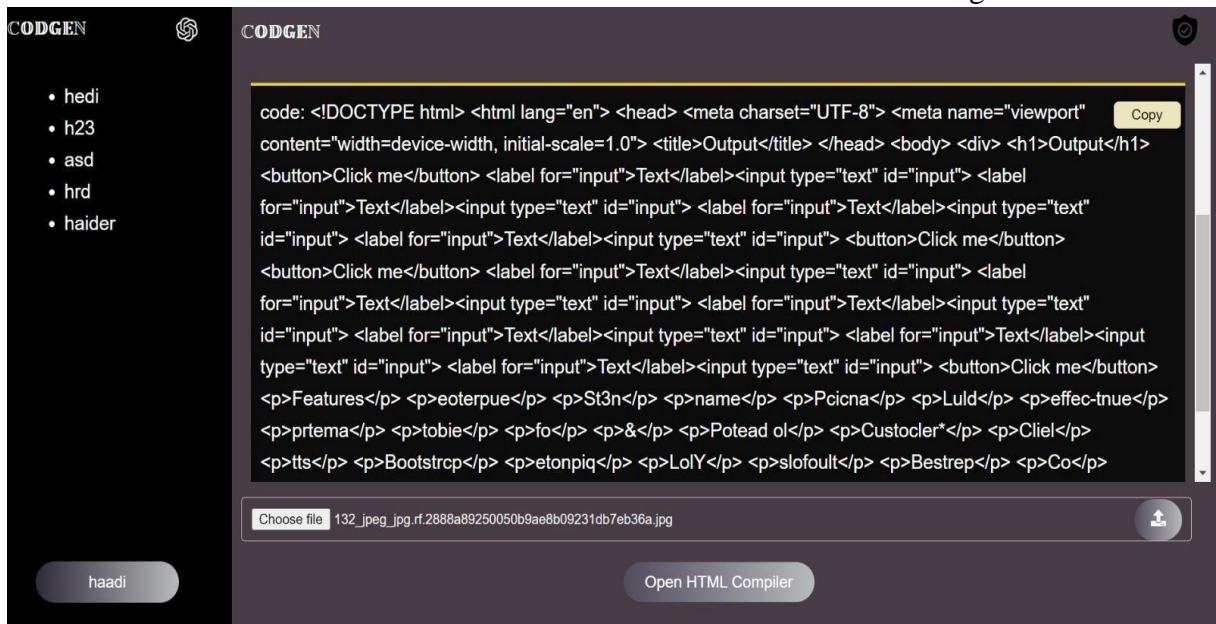


Figure 16 code generator

Code Output:

Figure 17 code output

CHAPTER 6**TESTING AND EVALUATION****Testing Methodology:**

Testing and evaluation were essential to ensure that the Hand-Drawn HTML Element Code Generator functioned correctly and met its intended objectives. Our comprehensive testing

approach encompassed various phases to address potential issues at multiple stages of development.

Unit Testing:

Objective: To verify that individual components of the system operated correctly in isolation.

Approach: We conducted unit tests on critical modules, such as the drawing input interface, HTML code generation logic, and text extraction functionality. Each unit was assessed for expected performance, accuracy, and reliability. For example, we tested the drawing input to ensure that hand-drawn sketches were accurately captured and that the HTML code generation produced the correct output. Edge cases, such as invalid inputs and extreme drawing styles, were evaluated to confirm the robustness and efficiency of each component under various conditions. Test cases were designed to encompass both typical and atypical scenarios, thoroughly validating the reliability of the system.

Integration Testing:

Objective: To evaluate whether different components of the system worked seamlessly together.

Approach: Integration tests focused on the interaction between interconnected modules, particularly how data flowed from the drawing input to the text extraction and HTML generation stages. We examined scenarios where users input hand-drawn sketches and verified that the system accurately extracted text and generated corresponding HTML elements. This phase helped identify issues related to data integrity, communication between modules, and overall system cohesion. User interaction scenarios were tested to ensure that the end-to-end process was smooth and met user expectations. We also assessed the performance of the system under high loads to ensure responsiveness and stability.

User Acceptance Testing (UAT)

Objective: To gather feedback from end-users to determine whether the system meets their needs and expectations.

Approach: A group of potential users was invited to test the application in real-world scenarios. Users were asked to hand-draw sketches and observe the system's performance in extracting text and generating HTML. Feedback was collected through surveys and interviews, focusing on usability, accuracy, and overall satisfaction. Any identified issues were prioritized and addressed in subsequent iterations of the development process.

Conclusion:

Through this comprehensive testing methodology, we ensured that the Hand-Drawn HTML Element Code Generator was robust, reliable, and capable of delivering a high-quality user experience while meeting its design objectives. The thorough testing process allowed us to identify and resolve potential issues early, resulting in a well-integrated and effective system. Continuous monitoring and user feedback were incorporated into the development cycle, enabling iterative improvements to adapt to evolving requirements and maintain system reliability, ensuring it performs well and meets user needs over time.

CHAPTER 7

CONCLUSION AND FUTURE WORK

Conclusion

In summary, the Hand-Drawn HTML Element Code Generator successfully merges intuitive design with robust functionality, allowing users to translate hand-drawn sketches into functional HTML elements seamlessly. The project addressed the challenges faced by developers and designers in visualizing ideas quickly and efficiently, particularly for those who may not be well-versed in coding. Through a combination of machine learning techniques, text extraction, and HTML generation logic, the system demonstrates the potential to enhance productivity and creativity in web development.

The comprehensive testing methodology employed throughout the development process ensured that the application is reliable, user-friendly, and meets the needs of its target audience. By gathering feedback from users during the testing phases, we were able to fine-tune functionalities, enhancing the overall user experience. The positive outcomes of this project highlight the feasibility of using AI and image processing technologies to simplify complex tasks and promote accessibility in web design.

Future Work

While the Hand-Drawn HTML Element Code Generator has achieved its primary objectives, there remains significant scope for future enhancements. Several potential areas for development include:

1. **Expanded Element Recognition:** Future iterations could incorporate more complex HTML elements and CSS styles, allowing for a broader range of design possibilities directly from sketches. Enhancing the system's ability to recognize various elements like buttons, forms, and responsive layouts would make it even more powerful for web developers.
2. **User Interface Improvements:** Based on user feedback, the interface could be further refined to make it more intuitive and visually appealing. This may include adding tooltips, help sections, or tutorials to assist users in navigating the application and maximizing its potential.
3. **Integration with Web Development Tools:** Future versions could explore integrating the generator with popular web development platforms, such as Visual Studio Code or WordPress, enabling users to directly implement generated HTML elements into their projects without manual copying.

4. **Enhanced Machine Learning Models:** Ongoing research and advancements in machine learning techniques could be leveraged to improve the accuracy of text extraction and element recognition. Incorporating more sophisticated models could enhance the system's ability to process a variety of hand-drawn styles and fonts.
5. **Cross-Platform Compatibility:** Developing a mobile version of the application could expand its usability, allowing users to generate HTML elements directly from their smartphones or tablets. This flexibility would cater to a wider audience and facilitate creative brainstorming on the go.

By focusing on these areas for future development, the Hand-Drawn HTML Element Code Generator can evolve into an even more powerful tool, fostering creativity and efficiency in web design for a diverse range of users.

.

REFERENCES

- [1] https://www.irjmets.com/uploadedfiles/paper/issue_1_january_2022/18158/final/fin_irjmets1641387167.pdf
- [2] https://ijirt.org/master/publishedpaper/IJIRT152018_PAPER.pdf
- [3] <https://www.upwork.com/resources/sketch-to-code>
- [4] <https://github.com/YogeshUpdhyay/SketchToCode>
- [5] <https://github.com/ashnkumar/sketch-code>
- [6] <https://www.ijert.org/automatic-generation-of-html-code-from-hand-drawn-images-using-machinelearning-techniques>
- [7] <https://ieeexplore.ieee.org/document/9824521>
- [8] <https://www.scribd.com/document/718238727/Automated-HTML-Code-Generation-from-HandDrawn-Images-using-Machine-Learning-Methods-1>
- [9] <https://fantastech.co/sketch-to-html/>
- [10] https://www.ijprems.com/uploadedfiles/paper//issue_5_may_2023/31221/final/fin_ijprems1684079233.pdf