# Introduction GUI

This report details the development of the front-end components of the application, including the HTML structure, CSS styling for responsiveness, and integration with Flask for backend functionality.

A screenshot of a computer

Description automatically generated

## Method

#### HTML Structure:

1. **Navigation Bar**: We created a fixed navigation bar at the top of the page, which includes links to different sections of the application (*Home, Converted File, Historical Conversions*). This ensures easy navigation for the user.



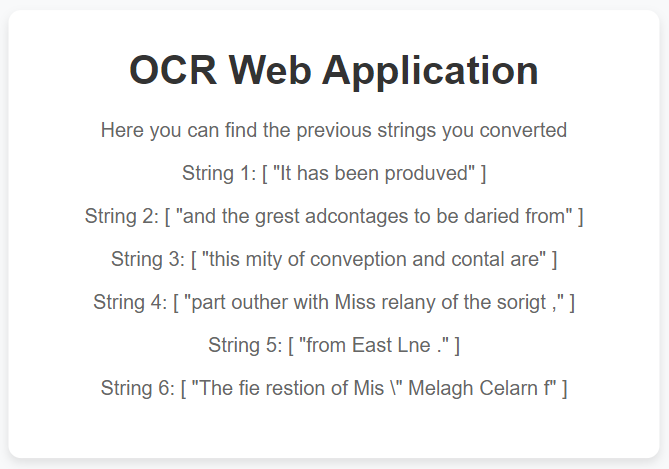
1. A screenshot of a web application

   Description automatically generated**Containers**: The main content is divided into three containers:

* `*home-container*`: This is the default view where users can upload an image or capture a photo using their device’s camera.
* `*converted-file-container*`: This displays the result after the OCR processing.
* `*historical-conversions-container*`: This shows previously converted texts.

#### A close-up of a web application Description automatically generatedCSS Styling:

We employed CSS to enhance the visual appeal and ensure the responsiveness of the web application. Key CSS features include:

* *Responsive Design*: Objects of the form are sized according to screen size of the device used.
* *Button Styling*: Buttons were styled to be visually consistent and included hover effects to improve interactivity.
* *Mirroring Camera Feed*: When capturing an image using the device’s camera, the video feed is mirrored horizontally to make pre-processing feature easier.

JavaScript Functionality:

JavaScript was used to add interactivity to the web page. Key functions include:

* *Camera Access*: Functions to access the user's camera by creating a video element, capturing images, displaying the captured image, and removing the video element from the DOM[1]. Functions involved: openCamera(), capturePhoto(), stopCamera()
* *File Upload*: Handles image uploads, including format conversion and image delivery to the server via a POST request. It displays the uploaded image, waits for the server response, and then switches the view to show the converted string results.

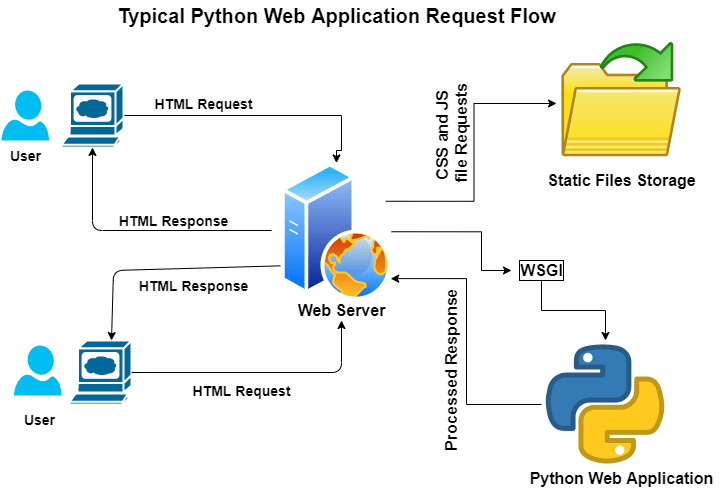
Functions involved: convertImage()

* *Navigation*: Switching between different containers based on user actions.

Functions involved: showHome(), showConvertedFile, showHistoricalConversions

EventListener //Event listener for file input change event

### Integration with Flask[2]

Flask is a WSGI *application*. A WSGI *server* is used to run the application, converting incoming HTTP requests to the standard WSGI environ, and converting outgoing WSGI responses to HTTP responses.

In our project, Flask:

1. Renders the main HTML page: Displays the GUI interface for users.
2. Handles image processing requests: Receives the base64-encoded image data from the client, processes it using the OCR model, and returns the extracted text.
3. Manages different views: Switches between the home view, converted file view, and historical conversions view based on user interactions.

A screen shot of a computer

Description automatically generated

## Results

The application allows users to easily upload images or capture photos directly from their devices. The responsive design ensures that the application is accessible on various devices, providing a consistent user experience.

## Challenges Encountered

1. Camera Access and Mirroring: Implementing the camera functionality and ensuring the video feed was mirrored correctly required careful handling of video streams and CSS transformations.
2. Responsive Design: Ensuring the layout was fully responsive involved extensive testing and adjustments, particularly for mobile devices with different screen sizes.
3. Integration with Flask: Managing the communication between the front end and the Flask backend required a clear understanding of both client-side and server-side programming.

[1] Document Object Model

[2] https://flask.palletsprojects.com/en/2.3.x/deploying/