INTERNSHIP REPORT

Exposys data labs | Bengaluru, Karnataka 560064

MASS-mAIL DISPATCHER

Sudhir Das

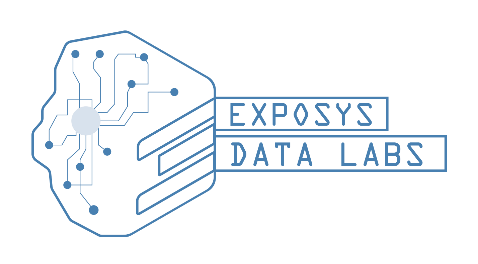
2023

PROJECT REPORT

On

**MASS-MAIL DISPATCHER**

For



**EXPOSYS DATA LABS**

Yelahanka, Bengaluru, Karnataka 560064

*Submitted by*

**SUDHIR DAS**

Of

**VEER SURENDRA SAI UNIVERSITY OF TECHNOLOGY**

Burla, Odisha 758017

Aug 2023

**ABSTRACT**

The "Mass Mail Dispatcher" project introduces a web-based solution for efficient mass email distribution. Through this application, users can upload recipient email addresses via CSV files, leading to automated validation and categorization of valid and invalid emails. Leveraging HTML, CSS, and JavaScript, the user-friendly interface guides users through the process. The application enhances communication by enabling personalized email composition. Noteworthy features include real-time categorization, validation logic, and integration with the EmailJS API for sending emails. Challenges included CSV parsing and email validation implementation. In conclusion, the "Mass Mail Dispatcher" project demonstrates the strategic application of modern web technologies to address communication hurdles, optimizing workflows, and offering insights into the potential of user-centric applications.

**TABLE OF CONTENTS**

A

**Contents Page No.**

[1. Introduction 3](#_Toc143340451)

[1.1. Background 3](#_Toc143340452)

[1.2. Problem Statement 3](#_Toc143340453)

[1.3. Project Objective 3](#_Toc143340454)

[1.4. Scope and Approach 3](#_Toc143340455)

[2. Existing Method 5](#_Toc143340456)

[2.1. Manual Email Distribution 5](#_Toc143340457)

[2.2. Challenges with the Existing Method 5](#_Toc143340458)

[3. Proposed Method With Architecture 7](#_Toc143340459)

[3.1. Process Flow 7](#_Toc143340460)

[3.1.1. User Interface (Front-end) 7](#_Toc143340461)

[3.1.2. CSV File Parsing 7](#_Toc143340462)

[3.1.3. Email Validation and Categorization 7](#_Toc143340463)

[3.1.4. Third-party API Integration (EmailJS) 7](#_Toc143340464)

[3.1.5. Personalized Email Composition 7](#_Toc143340465)

[3.1.6. Email Sending and Feedback 8](#_Toc143340466)

[3.2. Architecture Diagram 8](#_Toc143340467)

[4. Methodology 9](#_Toc143340468)

[4.1. Project Planning and Scope Definition 9](#_Toc143340469)

[4.2. System Design 9](#_Toc143340470)

[4.3. Technology Selection 9](#_Toc143340471)

[4.4. Implementation and Development 9](#_Toc143340472)

[4.5. Testing and Quality Assurance 10](#_Toc143340473)

[4.6. Deployment 10](#_Toc143340474)

[5. Implementation 11](#_Toc143340475)

[5.1. User Interface Development 11](#_Toc143340476)

[5.2. CSV File Parsing and Data Extraction 12](#_Toc143340477)

[5.3. Email Validation and Categorization 13](#_Toc143340478)

[5.4. Third-party API Integration (EmailJS) 15](#_Toc143340479)

[5.4.1. Sign Up and Create an Email Service 15](#_Toc143340480)

[5.4.2. Get Your API Key 15](#_Toc143340481)

[5.4.3. Include EmailJS Library in Your HTML 16](#_Toc143340482)

[5.5. Email Composition and Personalization 16](#_Toc143340483)

[5.6. Handle or Setup the Form Submission 17](#_Toc143340484)

[5.7. Sending emails using *emailjs.send* method 18](#_Toc143340485)

[5.8. Testing and Quality Assurance 19](#_Toc143340486)

[5.9. Deployment and Accessibility 20](#_Toc143340487)

[6. Conclusion 21](#_Toc143340488)

# **1. INTRODUCTION**

## **1.1. Background**

In today's digital era, effective communication is paramount, and email has become a ubiquitous medium for correspondence. With the increasing reliance on email for personal, professional, and promotional purposes, managing mass email distribution has emerged as a significant challenge.

## **1.2. Problem Statement**

The exponential growth in the volume of emails requires an efficient solution to handle mass email distribution. Manually sending individual emails to a large number of recipients is time-consuming, prone to errors, and may lead to email deliverability issues. Therefore, there is a need for a streamlined approach to sending bulk emails.

## **1.3. Project Objective**

The "Mass-Mail Dispatcher" project aims to develop a web-based application that simplifies the process of sending mass emails to a broad audience. The project seeks to create an intuitive platform that allows users to upload recipient email addresses, compose personalized messages, and ensure the successful delivery of emails.

## **1.4. Scope and Approach**

The project's scope includes the development of a user-friendly interface for uploading recipient email addresses, automated email validation, real-time categorization of valid and invalid emails, and integration with third-party email services.

We are using [EmailJS](https://www.emailjs.com/) as a third-party email service for this project which is a JavaScript library that simplifies sending emails directly from the client-side without the need for a server-side setup. It works by connecting to third-party email sending services through APIs, eliminating the need for configuring SMTP servers.

Pros of Email.js:

* No need to set up and maintain a server-side SMTP service.
* Utilizes third-party APIs, which can simplify the email sending process.
* Suitable for applications where sending emails from the client-side is preferred.

Cons of Email.js:

* Limited to the features and limitations of the chosen email sending service.
* Certain usage limits and subscription plans depending on the chosen service.

Why EmailJS over other options?

All email services require some sort of authentication to send the emails on your behalf. That makes it a really bad idea to use them directly from client-side – revealing your password or your secret keys will allow anyone to send emails on your behalf. EmailJS keeps your authentication details on the server-side, and the client-side code just triggers a predefined email template, similarly to how any client-server application is working.

However, if you anticipate sending a large volume of emails or require more control over the email sending process, you might need to explore server-side options like SMTP.js or other server-based email solutions.

# **2. EXISTING METHOD**

The existing method for managing mass email distribution typically involves manual processes and the use of email client software. Here's an overview of the traditional approach.

## **2.1. Manual Email Distribution**

* **Compile Email List**: The sender manually creates a list of recipient email addresses, often stored in a spreadsheet or a text document.
* **Copy and Paste**: The sender composes the email content using an email client or word processing software. The same email is then copied and pasted individually into the recipient field for each email address.
* **Send Emails**: The sender manually sends each email individually, clicking "send" for each recipient. This process is time-consuming and error-prone.
* **Managing Responses**: Any responses or interactions from recipients must be manually tracked and managed by the sender.

## **2.2. Challenges with the Existing Method**

* **Time-Consuming**: The manual process of copying and pasting the same email for each recipient is inefficient and time-consuming, especially for large email lists.
* **Prone to Errors**: Manually entering recipient email addresses can lead to typos and mistakes, resulting in failed deliveries.
* **Limited Personalization**: Mass emails sent through this method lack the personal touch required for effective communication.
* **Tracking and Analytics**: Tracking email responses, bounces, and interactions is difficult without specialized tools.
* **Deliverability Issues**: Bulk emails sent from personal email accounts may trigger spam filters or cause deliverability issues.

The "Mass-Mail Dispatcher" project seeks to address these challenges by automating the email distribution process, enabling automated email validation, personalization, and integration with email services to ensure successful delivery and efficient management of mass emails.

# **3. PROPOSED METHOD WITH ARCHITECTURE**

## **3.1. Process Flow**

### **3.1.1. User Interface (Front-end)**

The project's user interface is designed using HTML, CSS, and JavaScript. Users can upload CSV files containing recipient email addresses, compose email content, and initiate email sending.

### **3.1.2. CSV File Parsing**

The uploaded CSV file is parsed using JavaScript's FileReader API. The parsed data is processed to extract email addresses and other relevant information.

### **3.1.3. Email Validation and Categorization**

The parsed email addresses undergo automated validation to ensure proper email format. Valid and invalid email addresses are categorized in real-time using JavaScript logic.

### **3.1.4. Third-party API Integration (EmailJS)**

The EmailJS library is integrated into the application. The application uses the EmailJS API to send emails on behalf of the user. Integration with EmailJS ensures reliable email delivery and avoids common deliverability issues.

### **3.1.5. Personalized Email Composition**

Users can compose personalized email messages through the application's interface. Placeholders can be used to dynamically insert recipient names and other personalized details.

### **3.1.6. Email Sending and Feedback**

Once email composition is complete, users initiate the email sending process. The application loops through the valid email addresses and sends personalized emails using the EmailJS API. Users receive feedback on the successful sending of emails and any errors encountered.

## **3.2. Architecture Diagram**

Fig. 1 Diagram

# **4. METHODOLOGY**

## **4.1. Project Planning and Scope Definition**

The project's objectives and scope were clearly defined, outlining the primary goal of streamlining mass email communication. During this phase, an understanding of the challenges associated with manual email distribution was established, setting the foundation for the project's design.

## **4.2. System Design**

A comprehensive system design was framed, including the architecture, user interface layout, and data flow. Mock-ups were created to visually depict the application's appearance and user interactions.

## **4.3. Technology Selection**

Appropriate technologies were carefully chosen to implement the project. HTML, CSS, and JavaScript were selected for the front-end development, ensuring a responsive and user-friendly interface. The EmailJS library was identified as the ideal solution for seamless email integration.

## **4.4. Implementation and Development**

The project's implementation phase involved translating the design into a functional application. The user interface was developed, allowing users to upload CSV files, compose personalized emails, and initiate email sending. JavaScript's FileReader API facilitated CSV file parsing, while custom logic validated email addresses and categorized them based on validity.

## **4.5. Testing and Quality Assurance**

Rigorous testing was conducted to verify the accuracy and reliability of each component. This phase included exhaustive checks for email validation, proper email composition, successful integration with the EmailJS API, and the overall user experience.

## **4.6. Deployment**

The application was deployed on a web server to make it accessible to users. Extensive testing was carried out to ensure compatibility across different browsers and devices, guaranteeing a seamless experience.

# **5. IMPLEMENTATION**

## **5.1. User Interface Development**

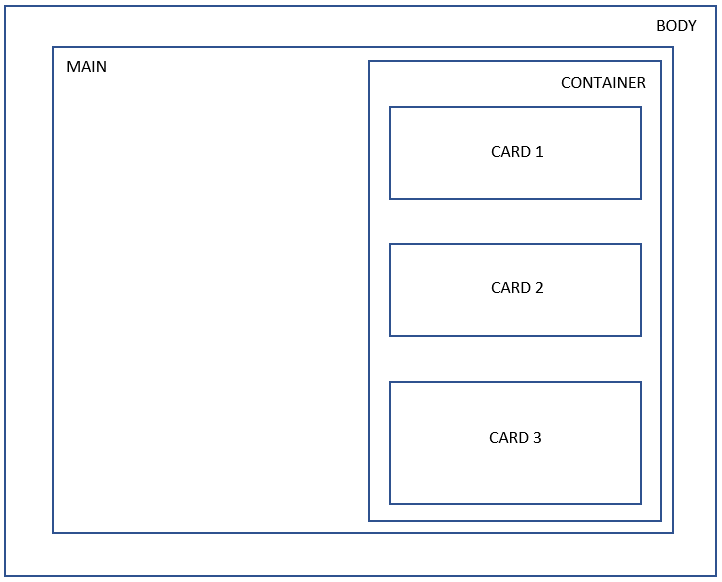
The initial phase of implementation focused on creating an intuitive user interface using HTML, CSS, and JavaScript. The interface allowed users to upload CSV files, compose personalized emails, and initiate email sending.

Fig. 2 Page layout

## **5.2. CSV File Parsing and Data Extraction**

JavaScript's FileReader API was employed to parse uploaded CSV files and extract recipient email addresses. Data formatting and transformation were performed to prepare for subsequent validation.



The code starts by grabbing an HTML element with the ID "upload" and assigns it to the variable upload which corresponds to the input field where users can upload a CSV file. An event listener is added to the upload element which triggers when the user selects a file for upload. The function inside the event listener will be executed when this event occurs. Inside the event listener function, a new FileReader object, denoted by “fr”, is created which allows us to read the contents of a file. The readAsText method of the FileReader is called on the selected file which reads the contents of the selected file as text. The parameter, *upload.files[0]*, refers to the first file selected in the upload input.

Then onload event is assigned a function that will be executed once the file is successfully loaded and read. This is a callback function that handles the file content processing. Inside the onload function, the content of the file is split into lines using a regular expression to match newline characters. Each line is then further split into an array of values using the comma as the delimiter. This is done using the map function.

Now we begin the processing of the CSV data. We initialize variables “valNo” and “invalNo” to keep track of the counts of valid and invalid email addresses, respectively. We also create an empty array “valMail” to store valid email addresses. Then we iterate through each element “data” in the “Arr” array, which represents a line from the CSV file. We convert “data” into a string called “email”. For each element “data” in the line, we use the map function to transform the values into HTML table cells “td” and store them in the array “tableCells”.

We create a new table row element “tr” using *document.createElement*, and assign it to the variable “tableRow” and set the inner HTML of the newly created row element to the array “tableCells”. This means each cell from the CSV file will become a table cell in the row.

## **5.3. Email Validation and Categorization**

Custom validation logic was implemented to validate email addresses in real-time during CSV parsing. Valid and invalid email addresses were categorized, forming the basis for subsequent actions.



Here, the outer if statement checks if the “email” variable is not an empty string.

Inside the outer if statement, we have another if statement that focuses on email validation using regular expressions which is used to validate the format of an email address.

It's checking if the email adheres to the following format:

1. **[A-Za-z\.\_\-0-9]\*** - This matches a sequence of characters that can include uppercase letters (A-Z), lowercase letters (a-z), digits (0-9), underscores (\_), periods (.), and hyphens (-). The \* means that this sequence can occur zero or more times.
2. **[@]** - This matches the "@" symbol, which is used to separate the username from the domain in an email address.
3. **[A-Za-z]\*** - This matches a sequence of uppercase and lowercase letters that make up the domain name (e.g., "gmail", "yahoo").
4. **[\.]**- This matches the period (.) symbol, which separates the domain name from the top-level domain (TLD) in the email address.
5. **[a-z\.]** - This matches a sequence of lowercase letters and periods that make up the TLD.
6. **{2,}** - specifies that the TLD must consist of at least two characters, which covers most common TLDs like ".com", ".org", ".net", etc.
7. **$** - This asserts the end of the string, ensuring that the regular expression matches the entire input.

The match function is used to test if the “email” matches the given regular expression. If the email matches this pattern, it's considered a valid email address and if the email doesn't meet the conditions specified in the inner if statement, the code block within the else is executed, indicating an invalid email.

## 

## **5.4. Third-party API Integration (EmailJS)**

Integration with the EmailJS API enabled automated email sending. API keys and configurations were set up to facilitate communication with the EmailJS service. Here's a step-by-step guide on how to integrate EmailJS into your code:

### **5.4.1. Sign Up and Create an Email Service**

1. Go to the EmailJS website (<https://www.emailjs.com/>) and sign up for an account if you haven't already.
2. After signing up, log in to your EmailJS account.
3. In your EmailJS dashboard, navigate to the "Email Services" section and create a new service. This service will define the email provider you want to use (like Gmail, Outlook, etc.).
4. Follow the instructions to configure the email service you created. This usually involves providing your email provider's credentials and settings.
5. Obtain the Service ID.

### **5.4.2. Get Your API Key**

1. In the EmailJS dashboard, go to the "API Keys" section in Account settings.
2. You will find Public Key.

### **5.4.3. Include EmailJS Library in Your HTML**

1. Add the following script tag to just before your HTML file’s closing tag to include the library:



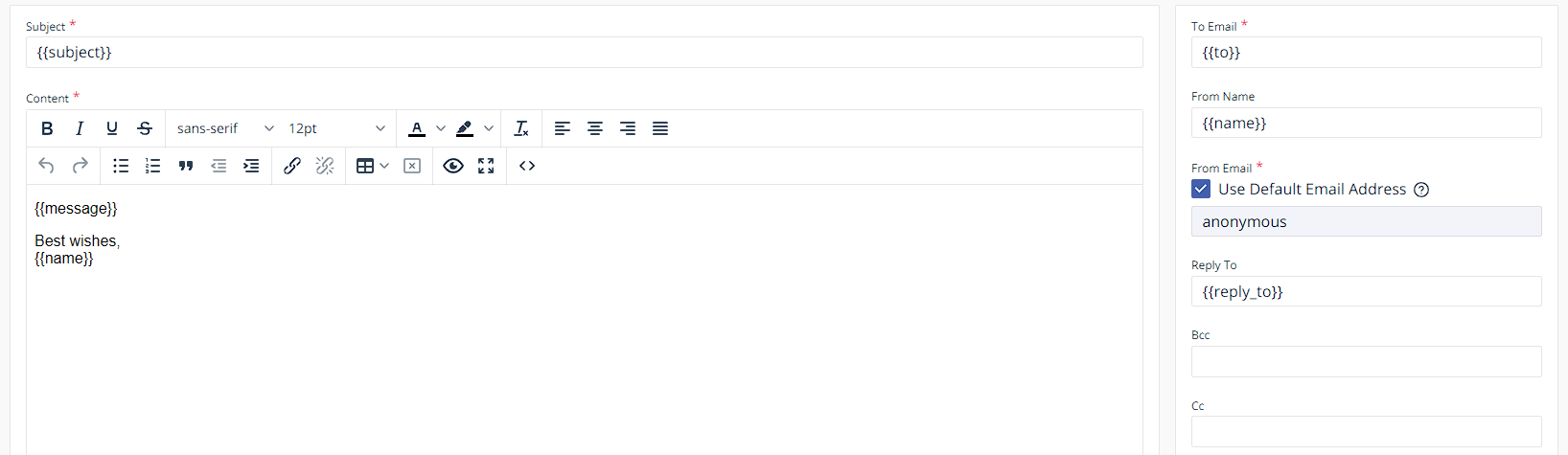
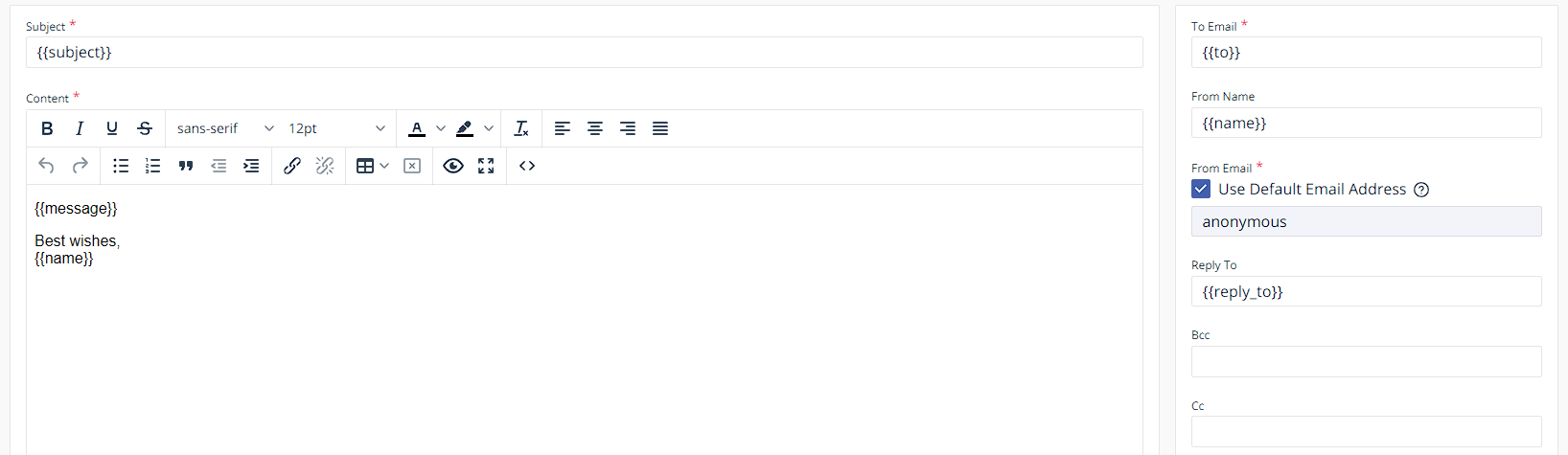
1. Initialize EmailJS using your public key that you created:



## **5.5. Email Composition and Personalization:**

Now we have the email service, we need to create our email template, which will define the name of the sender, subject of our email, what content will it contain, where should it be sent to, etc.

Let's navigate to the Email Templates, where we'll create a new template. Now let's go back to form in the HTML file, where we'll see “to”, “name”, “subject” and “message” input fields. Any field can contain [dynamic variables](https://www.emailjs.com/docs/user-guide/dynamic-variables-templates/), so we can set the value programmatically from our JavaScript code. Now we'll create unique values for each input fields.

We'll do this by adding dynamic variables – {{to}}, {{name}}, {{subject}} and {{message}} in the template for input fields having IDs – “to”, “name”, “subject” and “message” respectively. Now the value of these dynamic variables are depended on whatever the user fills in input fields on website.

## **5.6. Handle or Setup the Form Submission**

The code starts by selecting an HTML element with the ID "submit" and attaching a "click" event listener to it. When the button is clicked, the provided arrow function is executed. The event parameter represents the event object associated with the click event.



These lines retrieve the values of different form fields. It assumes that there are form input elements with the IDs "name," "subject," and "message." The name and subject variables capture the values of the corresponding input fields, while the message variable captures the content of the "message" field. Note that the content is retrieved using textContent for the "message" field, but if that's not available, it falls back to the value property.



This loop iterates through an array named valMail. For each email address in the array, it prepares an object called templateParams containing the recipient's email address (to), the sender's name (name), the email message content (message), and the email subject (subject).



**5.7. Sending emails using *emailjs.send* method**

Syntax -



The code uses the EmailJS library to send emails. The send function takes three parameters: the email service ID, the template ID, and the templateParams object. It returns a promise. If the promise is fulfilled, the provided success callback function is executed, displaying a success message in the console. If the promise is rejected, the error callback function is executed, displaying an error message in the console.



After the loop completes, an alert is displayed indicating the number of successfully sent emails. The variable “valNo” holds the number of emails sent.

## **5.8. Testing and Quality Assurance:**

## 

Every component of the application underwent comprehensive functional testing. The parsing of CSV files, email validation, email composition, and the integration with the EmailJS API were all meticulously scrutinized. Test cases were designed to cover a range of scenarios, including valid and invalid inputs, different email formats, and boundary conditions.

**Validation and Categorization**: The accuracy of email validation and categorization was of paramount importance. A diverse set of test email addresses was used to validate the application's handling of various formats and cases.



**User Interface Testing**: The user interface was thoroughly tested for responsiveness across different devices, browsers, and screen sizes. The application's appearance on various platforms was assessed to guarantee a consistent user experience.

## **5.9. Deployment and Accessibility:**

The application was deployed on a web server, making it accessible to users. Extensive deployment testing ensured cross-platform compatibility and responsiveness.

# **6. CONCLUSION**

The completion of the "Mass Mail Dispatcher" project marks a significant achievement in streamlining mass email communication. Through meticulous planning and agile development the project has succeeded in simplifying the process of sending personalized emails to a wide audience. Notably, the integration of real-time feedback mechanisms has empowered users with immediate insights into the validity of their email lists. This transparency has enabled users to make informed decisions and tailor their communications effectively. Furthermore, the user-friendly interface, coupled with seamless integration with the EmailJS API, has facilitated efficient email distribution. An essential consideration was the project's adherence to the limitations of the free EmailJS plan, which permits the sending of up to 200 emails per month, with around 150 email credits remaining. Make sure, the replacement of correct IDs and keys must be precisely checked before using. For those seeking a comprehensive understanding, a demo and walkthrough video have been prepared. This resource provides a detailed insight into the application's functionality, covering CSV file uploading, email validation, personalized email composition, and the initiation of the email sending process.

In conclusion, the "Mass Mail Dispatcher" project not only successfully achieved its primary objectives but also demonstrated design innovation and user-focused development. The experience gained throughout this project lays the groundwork for future endeavours, as technology continuously evolves to reshape and optimize digital interactions.

**References**

EmailJS Documentation - <https://www.emailjs.com/docs/>

MDN web docs JavaScript - <https://developer.mozilla.org/en-US/docs/Web/JavaScript>

The code for this project can be found in the GitHub repository: