A report on internship:

MASS EMAIL DISPATCHER WEBSITE

To

EXPOSYS DATA LABS

From

Sharanya Rao
N.M.A.M Institute of Technology
CSE student
Exposys data labs intern
Web development

Internship date: 21st August 2023 to 25th September 2023

A special thanks to

A special thanks to Exposys Data Labs for giving me this internship opportunity. I express my sincere gratitude for the invaluable support and guidance I received during my one-month remote web developer internship at Exposys Data Labs. Hence I write a report to summarize my experience and to provide a comprehensive overview of my internship experience and the significant impact it has had on my professional development.

Table of contents

Introduction

The Mass-Mail Dispatcher project is a web-based application designed to simplify the process of sending mass emails to many recipients. A mass email dispatcher, email validation, and a user-friendly front-end website play a crucial role in effective email communication for businesses and organizations.

The reason why need such a website is as follows:

1. Efficient Communication: A mass email dispatcher allows organizations to send emails to a large group of recipients simultaneously. This is vital for disseminating important information, promotions, newsletters, or updates efficiently.

2. Time and Resource Savings: Sending emails individually is time-consuming and resource-intensive. A mass email dispatcher automates the process, saving time and reducing the workload for employees.

3. Personalization: Mass email dispatchers often include personalization features, enabling organizations to tailor messages to individual recipients. This enhances engagement and relevance.

4. Scalability: Businesses can scale their email marketing efforts as they grow, reaching a larger audience without significantly increasing their workload.

5. Analytics and Tracking: Many mass email dispatchers provide detailed analytics and tracking tools. These insights help organizations gauge the effectiveness of their email campaigns, allowing for adjustments to improve results.

6. Email Validation: Filtering out valid and invalid emails is essential to maintain a clean and efficient email list. Valid emails are those that are active and reachable, while invalid ones include bounced, fake, or misspelled addresses.

7. Importance of Email Validation:
    a. Reduces bounce rates: Invalid emails result in bounces, damaging the sender's reputation and deliverability.
    b. Enhanced deliverability: ISPs and email services favor senders with low bounce rates, ensuring emails reach the inbox.
    c. Cost savings: Sending emails to invalid addresses wastes resources and may lead to unnecessary charges.

8. Font-End Website: A front-end website for email dispatching and management is essential for several reasons:
    a. User-friendly interface: It simplifies the process of creating and managing email campaigns, making it accessible to non-technical users.
    b. Customization: Businesses can brand the interface to match their identity and provide a consistent user experience.
    c. Data collection: The website can capture important recipient data, such as preferences and engagement metrics, for future targeting.
    d. Accessibility: A web-based interface allows authorized users to access email tools from anywhere with an internet connection.
    e. Security: A well-designed front-end can incorporate security features to protect sensitive data and user information.

Therefore a mass email dispatcher, coupled with email validation and a user-friendly front-end website, streamlines email communication, improves efficiency, ensures email list quality, and provides valuable insights for businesses and organizations engaged in email marketing and communication efforts.

Functional requirements

1. Email Sending Functionality:

   The core functionality of the system is to enable users to send mass emails. Users should be able to:

   a. Specify a sender's email address.
   b. Define the subject of the email.
   c. Compose the email message.
   d. Upload a CSV file containing recipient email addresses.

2. Email Validation:

   The application should validate recipient email addresses to distinguish between valid and invalid addresses. Valid email addresses should be sent emails, while invalid ones should be identified and displayed to the user.

3. CSV File Upload:

   Users should have the ability to upload CSV files. The system should support the acceptance of CSV files and handle them effectively.

4. User Feedback:

   The application must provide clear and timely feedback to users, including

   a. The number of valid email addresses detected.
   b. The number of invalid email addresses detected.
   c. Confirmation when emails have been successfully sent to valid recipients.

Non-Functional requirements

1. Usability:

   The user interface should be user-friendly and intuitive, making it easy for users to upload CSV files and initiate mass email sending. The design should follow best practices for user experience.

2. Performance:

   The system should be capable of handling a large number of email addresses efficiently. It should not experience significant delays during email processing.

3. Security:

   The security of user data, especially email addresses, is of utmost importance. Sensitive information must be protected from unauthorized access and potential breaches.

4. Reliability:

   The application should be reliable and robust. It should not crash or produce errors during email sending, ensuring a smooth user experience.

Methodology

Front-end Development:

The front end of the Mass-Mail Dispatcher website is developed using a combination of HTML, CSS, and JavaScript. HTML is used for defining the structure of the webpage, CSS is applied for styling and visual presentation, and JavaScript is employed for interactivity and handling user actions.

The front-end development of the Mass-Mail Dispatcher website is a critical aspect of the project, as it directly influences user experience and usability. The following steps outline the methodology for front-end development:

User-Centric Design: The development process begins with a thorough understanding of user needs and expectations. A user-centric design approach is adopted, involving understanding the requirements for the development of the project.

HTML Structure: HTML5 is employed to structure the webpage effectively. Semantic HTML elements are used to enhance accessibility and search engine optimization (SEO). Properly structured forms and input fields are created to facilitate user interaction.

CSS Styling: CSS (Cascading Style Sheets) is used to style the webpage, providing a visually appealing and consistent user interface. Careful attention is paid to typography, color schemes, and spacing to create a pleasing aesthetic.

Interactive Elements: JavaScript is utilized to enhance interactivity. Event handlers are implemented to capture user actions, such as form submissions and file uploads. Dynamic feedback mechanisms are established to update users on the progress of their actions.

Back-end Development:

The back-end functionality is facilitated through the integration of the EmailJS service. EmailJS allows for the sending of emails by providing templates and user-defined parameters. JavaScript is used extensively to read and validate CSV files, manage email sending, and handle responses from the EmailJS service.

The back-end development of the Mass-Mail Dispatcher website focuses on integrating external services for email handling and implementing server-side logic. The following steps outline the methodology for back-end development:

Email Service Integration: EmailJS, a third-party email service, is integrated into the application to handle email sending. Email templates are configured, allowing for dynamic content insertion.

Email Validation: JavaScript is used to validate recipient email addresses from the uploaded CSV file. Regular expressions are employed to ensure the correctness of email formats.

Server-Side Logic: Server-side logic is implemented to manage the interaction between the front end and EmailJS service. This includes handling user requests, composing emails with template parameters, and sending emails to valid recipients.

Error Handling: Robust error handling mechanisms are put in place to handle potential issues gracefully. This includes checking for network errors, invalid user inputs, and service failures.

Implementation

The front-end design of the application focuses on creating a visually appealing and user-friendly interface. Key features of the implementation include:

- Input fields for sender email address, subject, and message.
- A file input element for uploading CSV files.
- JavaScript code that validates the CSV file and sends emails to valid recipients using the EmailJS service.
- Displaying valid and invalid email addresses to the user for transparency and feedback.

The back-end functionality is powered by the EmailJS service, which efficiently sends emails using pre-defined templates and user-specified parameters. Email validation is accomplished through the use of JavaScript regular expressions.

The below code is how the website is implemented:

File index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Send Mail</title>
    <script src="https://cdn.emailjs.com/dist/email.min.js"></script>
    <link rel="stylesheet" href="style.css" />
    <link
      href="https://fonts.googleapis.com/css?family=Raleway:200,100,400"
      rel="stylesheet"
      type="text/css"
    />
  </head>

  <body>
    <p id="heading">
      Mass Mail<br />
      Dispatcher
    </p>
    <form method="post">
      <label for="senderEmail">From :</label>
      <input type="email" id="senderEmail" name="senderEmail" required />
```

```
      <br /><br /><br />

      <label
        for="subject"
        style="margin-top: -2em; z-index: 22; position: absolute"
        >Subject:</label
      >
      <textarea id="subject" name="subject" required></textarea>

      <br /><br /><br /><br />

      <label for="csvFile">CSV File:</label>
      <input type="file" id="csvFile" name="csvFile" accept=".csv"
required />

      <br /><br />

      <label for="message">Message:</label>
      <textarea id="message" name="message" required></textarea>

      <br /><br />

      <input
        type="button"
        value="Send Emails"
        style="font-weight: bold"
        onclick="sendEmails()"
      />

      <br />

      <div style="display: flex">
        <div>
          <p style="color: #393e41">
            Valid Emails: <span id="validEmailCount"></span>
          </p>
          <div id="validEmails" style="float: left"></div>
        </div>
        <div style="margin-left: 15px">
```

```
          <p style="color: #393e41">
            Invalid Emails: <span id="invalidEmailCount"></span>
          </p>
          <div id="invalidEmails" style="float: left"></div>
        </div>
      </div>
    </form>

    <script type="text/javascript">
      (function () {
        emailjs.init("nPbQxXJaUVkVPcHQS"); // For implementation from your
system, replace with your emailJS user ID
      })();

      function sendEmails() {
        var senderEmail = document.getElementById("senderEmail").value;
        var message = document.getElementById("message").value;
        var subject = document.getElementById("subject").value;

        var validEmails = [];
        var invalidEmails = [];

        // Read contents of CSV file
        var file = document.getElementById("csvFile").files[0];
        var reader = new FileReader();
        reader.readAsText(file);
        reader.onload = function (event) {
          var csv = event.target.result;
          var lines = csv.split("\n");
          for (var i = 0; i < lines.length; i++) {
            var email = lines[i].trim();
            var emailRegex = /^[\w-\.]+@([\w-]+\.)+[\w-]{2,3}$/;
            if (emailRegex.test(email)) {
              validEmails.push(email);
            } else {
              invalidEmails.push(email);
            }
          }

          // Code to send email to valid email addresses from csv file
```

```
          for (var j = 0; j < validEmails.length; j++) {
            var templateParams = {
              to_name: validEmails[j],
              from_name: senderEmail,
              message_html: message,
              subject_html: subject,
            };

            // Code for implementation: replace your emailJS Service ID
and Template ID.
            emailjs
              .send("service_m3fz4gr", "template_9esk4jg", templateParams)
              .then(
                function (response) {
                  console.log("SUCCESS", response);
                },
                function (error) {
                  console.log("FAILED", error);
                }
              );
          }

          alert("Emails sent to valid email addresses.");
        };
      }
    </script>
    <script type="text/javascript">
      document
        .getElementById("csvFile")
        .addEventListener("change", function () {
          var validEmails = [];
          var invalidEmails = [];

          // Code to read contents of CSV file
          var file = document.getElementById("csvFile").files[0];
          var reader = new FileReader();
          reader.readAsText(file);
          reader.onload = function (event) {
            var csv = event.target.result;
            var lines = csv.split("\n");
```

```
            for (var i = 0; i < lines.length; i++) {
              var email = lines[i].trim();
              var emailRegex = /^[\w-\.]+@([\w-]+\.)+[\w-]{2,3}$/;
              if (emailRegex.test(email)) {
                validEmails.push(email);
              } else {
                invalidEmails.push(email);
              }
            }

            // Displaying the valid and invalid emails to front-end
            document.getElementById("validEmails").innerHTML =
              validEmails.join("<br><br>");
            document.getElementById("invalidEmails").innerHTML =
              invalidEmails.join("<br><br>");
            document.getElementById("validEmailCount").innerText =
              "(" + validEmails.length + ")";
            document.getElementById("invalidEmailCount").innerText =
              "(" + invalidEmails.length + ")";
          };
        });
    </script>
  </body>
</html>
```

File style.css:

```
/* Set default font family, background color, and text color */
body {
  background-color: #d5f4e6;
  font-family: cursive;
  background-size: cover;
  position: relative;
  color: #333;
  overflow-y: hidden;
  z-index: 2;
}

/* Center the main heading and add margin at the top */
#heading {
  color: #618685;
  margin-left: 4em;
  font-family: Lucida Handwriting;
  font-size: 4em;
  font-weight: normal;
  position: fixed;
  margin-top: 4em;
}

/* Style the form container */
form {
  max-width: 600px;
  margin-left: 52em;
  margin-top: 1em;
  /* From https://css.glass */
  background: rgba(255, 255, 255, 0.1);
  border-radius: 16px;
  box-shadow: 0 4px 30px rgba(0, 0, 0, 0.1);
  backdrop-filter: blur(2.7px);
  -webkit-backdrop-filter: blur(2.7px);
  border: 1px solid rgba(255, 255, 255, 0.03);
  padding: 20px;
  /* border-radius: 10px; */
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
  position: relative;
```

```
}

/* Style form labels */
label {
  display: block;
  margin-bottom: 10px;
  font-weight: bold;
  color: #575761;
}

/* Style text inputs and text areas */
input[type="email"],
textarea {
  width: 95%;
  padding: 10px;
  margin-bottom: 10px;
  border: 1px solid #767a7d;
  border-radius: 5px;
  font-size: 16px;
  color: #766f6f;
  background: rgba(187, 179, 179, 0.1);
}

/* Style text inputs and text areas */
#subject {
  width: 89%;
  padding: 10px;
  margin-bottom: -1em;
  border: 1px solid #767a7d;
  border-radius: 5px;
  font-size: 16px;
  color: #766f6f;
  z-index: 22;
  position: absolute;

  background: rgba(187, 179, 179, 0.1);
}

/* Style file input */
input[type="file"] {
```

```
  margin-bottom: 12px;
  color: #393e41;
}

/* Style form submit button */
input[type="button"] {
  display: block;
  width: 100%;
  padding: 10px;
  background-color: #c5c3c6;

  border: none;
  border-radius: 5px;
  color: #393e41;
  font-size: 16px;
  cursor: pointer;
  transition: background-color 0.3s;
}

/* Style container for valid and invalid emails */
#validEmails {
  margin-top: 1px;
}
#invalidEmails {
  margin-top: 1px;
}

/* Style paragraphs within container for valid and invalid emails */
#validEmails p,
#invalidEmails p {
  font-weight: bold;
  color: #393e41;
}

/* Style divs within container for valid and invalid emails */
#validEmails div {
  color: #393e41;
}

#invalidEmails div {
```

```
  color: #393e41;
}

#message {
  color: #393e41;
}
#senderEmail {
  color: #393e41;
}

/* Style text inputs and text areas */
input[type="email"]:focus,
textarea:focus {
  outline: none !important;
  border-color: #393d40;
}
#validEmails,
#invalidEmails {
  height: 150px;
  overflow: auto;
}

.img1 {
  height: 250px;
  margin-top: -15em;
  position: fixed;
}

.img2 {
  height: 200px;
  margin-top: -23em;
  position: absolute;
  margin-left: 13.3em;
}

h1,
h2 {
  font-weight: 200;
  margin: 0.4em 0;
  z-index: 12;
```

```
  position: absolute;
  margin-top: 23em;
  margin-left: 14em;
}
h1 {
  font-size: 18px;
}
h2 {
  color: #888;
  font-size: 18px;
}
```

Testing, and Quality Assurance

Testing is an integral part of the development process to ensure the reliability and functionality of the Mass-Mail Dispatcher website. The testing phase includes:

- Unit testing of JavaScript functions for email validation and sending.
- Integration testing to verify the seamless interaction between front-end and back-end components.

Quality assurance measures are in place to guarantee that the application meets the specified requirements and adheres to best coding practices.

Conclusion

The Mass-Mail Dispatcher website has been successfully designed and developed to meet its functional and non-functional requirements. Users can now enjoy a user-friendly and reliable tool for sending mass emails to a vast number of recipients.

Future Scope

User Authentication: Implement user authentication to ensure that only authorized users can access and use the system.

Email Templates: Allow users to create and use email templates for more customized email content.

Performance Optimization: Optimize the application to handle even larger email lists without compromising performance.

Reporting and Analytics: Provide users with insights and reports on email delivery status and open rates.

Attachment Support: Extend the system to support email attachments for more versatile communication.

Integration: Integrate with popular email marketing services to enhance functionality.

The Mass-Mail Dispatcher website serves as a valuable tool for individuals and organizations looking to efficiently manage mass-email communication.