

Envelope

A thin email client

Nerurkar, Tanmay Ashay

Table of Contents

1. Introduction.....	3
1.1 Purpose	3
1.2 Key Features	3
1.3 Getting Started	3
2. Photo Guide.....	4
2.1 Login.....	4
2.2 Outbox.....	6
2.3 Compose	7
2.4 Delete Multiple	9
2.5 Email Details	12
2.6 Print.....	13
2.7 Delete	14
2.8 Logout	15
3. Technical Details	17
3.1 Overview	17
3.2 Front-End Technologies.....	17
3.3 Back-End Technologies	17
3.4 Communication	17
3.5 Flow of the application	18
4. Components	19
4.1 Server	19
4.2 Client:.....	24
5. Illustrations and explanations of the code.....	31
5.1 Syntax.....	31
5.2 Control Structures.....	31
5.3 Data Structures – Storage Folders	31
5.4 I/O	31
5.5 Error Handling Procedures.....	32
5.6 JavaScript	32
5.7 CSS.....	32
5.8 Event Handlers.....	32
6. Appendix	33
6.1 Structure.....	33
6.2 package.json.....	33

6.3 myserver.json.....	33
6.4 Login.html.....	38
6.5 Outbox.html.....	40
6.6 Compose.html	46

1. Introduction

Welcome to Envelope, a user-friendly email client application designed to streamline your email communication experience. This application boasts a simple and secure user interface, prioritizing both ease of use and the protection of your sensitive information.

1.1 Purpose

Envelope serves as a reliable platform for sending, managing, and organizing your emails. With a focus on user convenience, the application offers a straightforward login process to get you started quickly.

1.2 Key Features

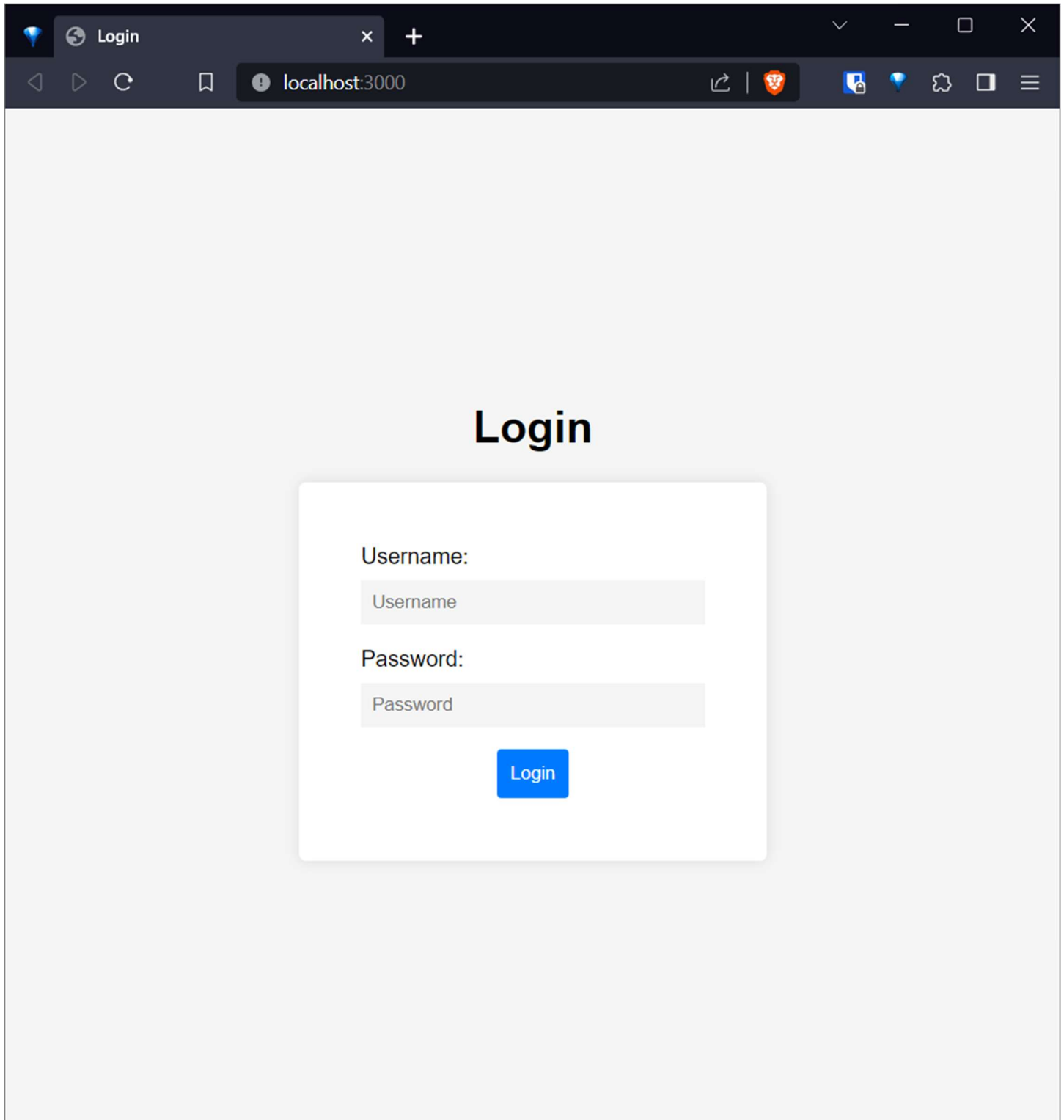
- **Secure User Interface:** Our application prioritizes the security of your data, ensuring a safe environment for all your email-related activities.
- **Sent Email History:** Upon successful login, the application provides a comprehensive display of previously sent emails, allowing you to quickly revisit and reference past correspondence.
- **Detailed Email View:** Explore the details of any email effortlessly by simply clicking on it. This feature enables a closer examination of the content and any attachments associated with each sent email.
- **Compose New Emails:** Seamlessly create and send new emails within the application, streamlining the communication process.
- **Print Functionality:** Our application allows you to print any sent email, providing a tangible record when necessary.
- **Efficient Deletion:** Manage your sent emails efficiently by deleting them from the database either individually or in batch, ensuring a clutter-free and organized mailbox.

1.3 Getting Started

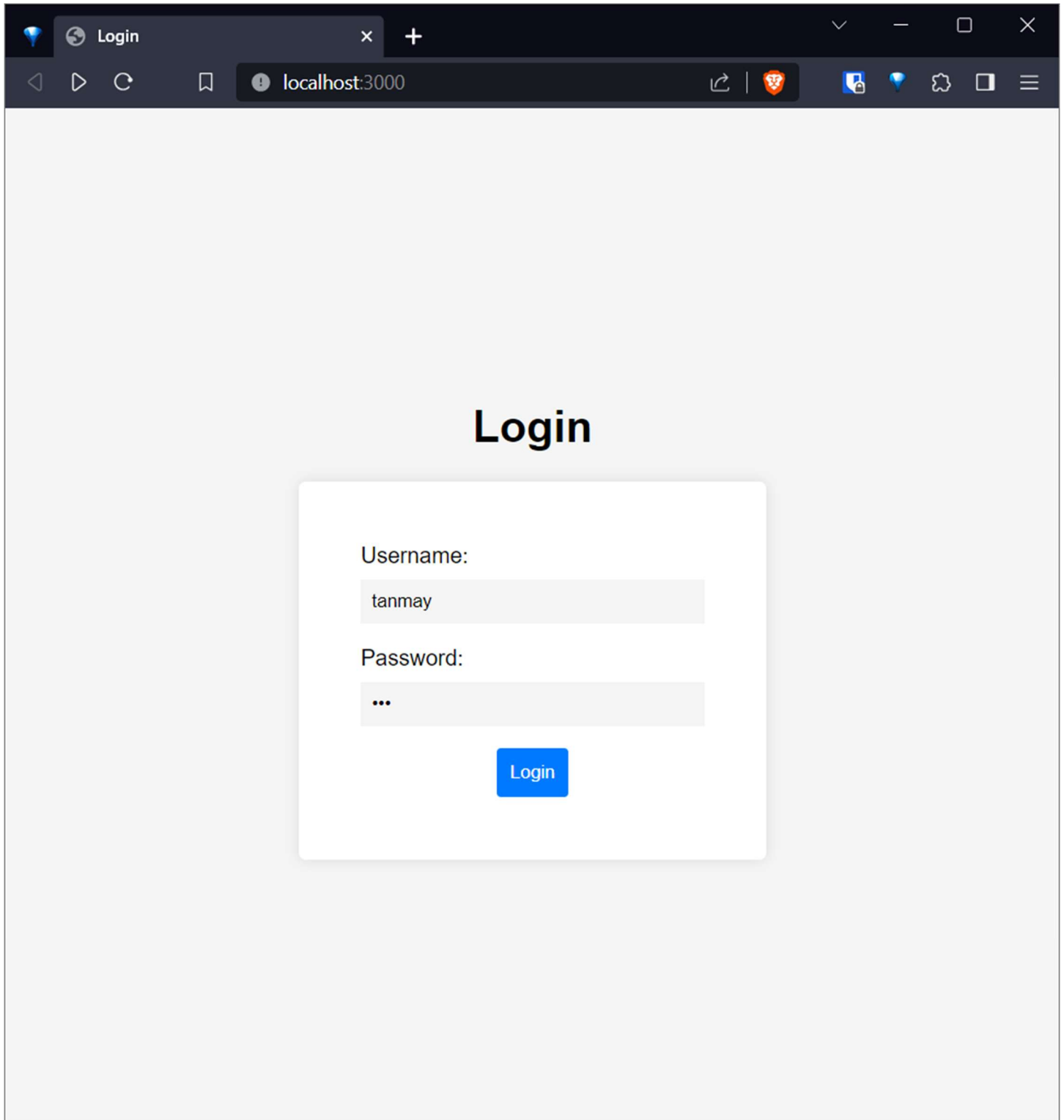
To embark on your journey with Envelope, all you need to do is log in using your credentials. Once logged in, you'll have immediate access to your previously sent emails, and from there, you can navigate the application's intuitive features to compose new emails, view detailed email content, print, and manage your sent emails effortlessly.

2. Photo Guide

2.1 Login



The root folder prominently displays the login form.



The image shows a web browser window with a dark theme. The address bar displays 'localhost:3000'. The page has a light gray background with the word 'Login' centered in a large, bold, black font. Below the title is a white login form with a subtle shadow. The form contains two input fields: 'Username:' with the text 'tanmay' and 'Password:' with three dots indicating a masked password. A blue 'Login' button is positioned below the password field.

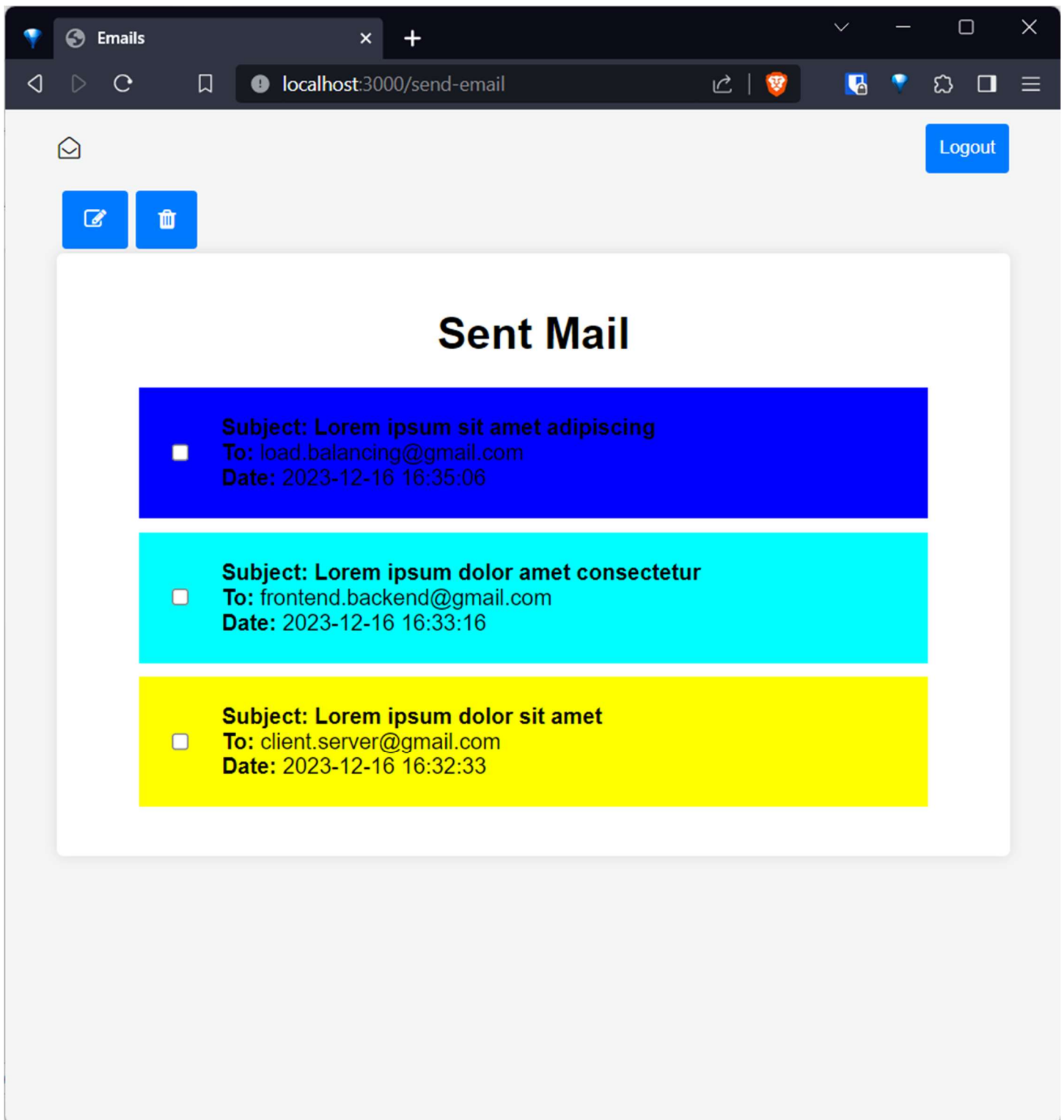
Login

Username:
tanmay

Password:
...

Login

2.2 Outbox



Once successfully logged in, sent emails are shown in reverse chronological order.

This page has the following buttons:

1. Compose: Compose an email
2. Delete: Deletes the emails that are selected. Requires at least one email to be selected.
3. Logout: Logs out of the application. Displays login page.

2.3 Compose

Compose

To:

data.transmission@gmail.com

Subject:

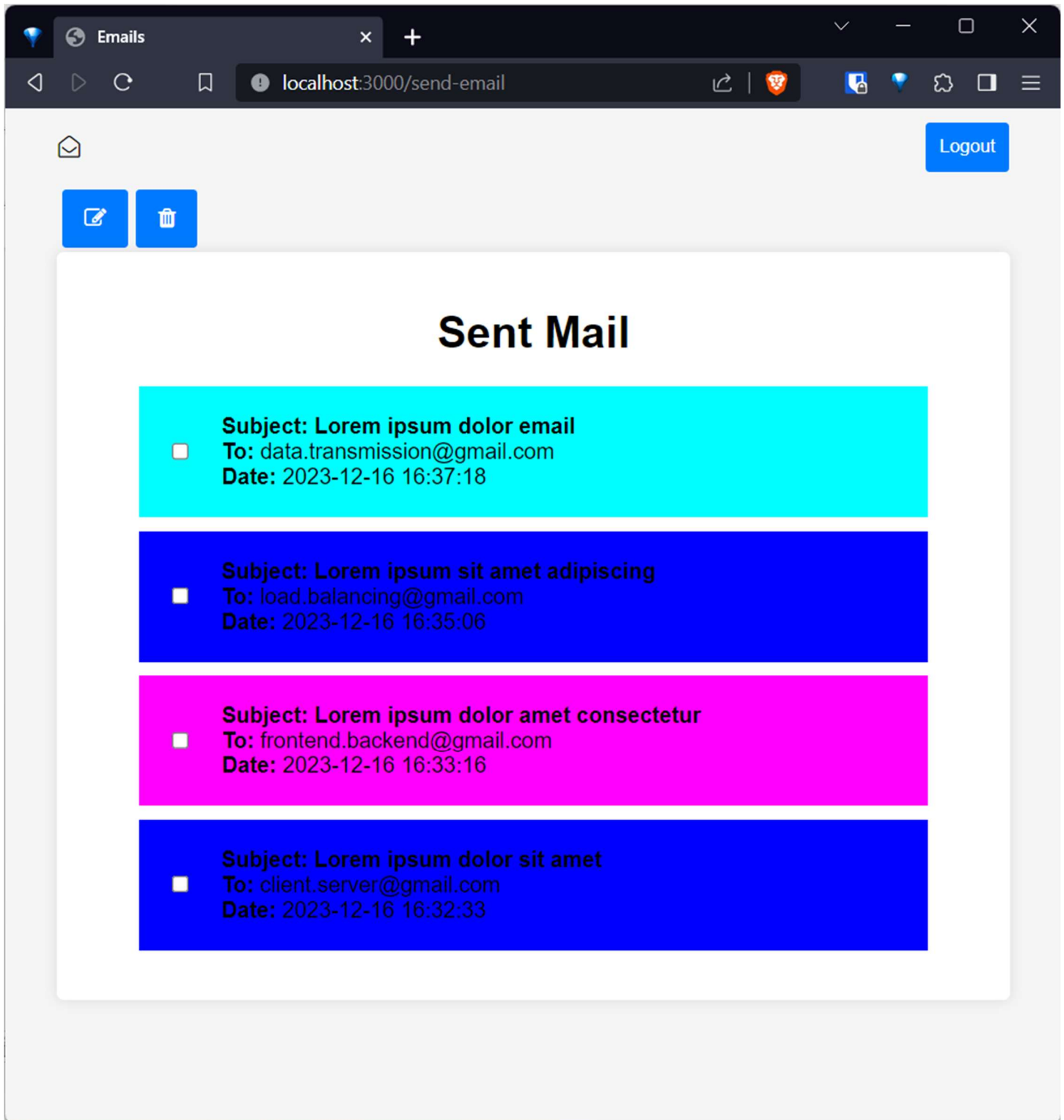
Lorem ipsum dolor email

Message:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed facilisis mauris id mauris sodales, vel posuere elit tristique. Proin ac ex a mi tincidunt aliquet. Quisque sed velit non felis fermentum hendrerit vel vel elit. Vestibulum euismod neque ac orci efficitur, non pellentesque lectus consectetur. Integer in fermentum nunc. Nullam sed justo in urna fermentum mattis eu id tellus. Vivamus nec nisi id tortor aliquam ullamcorper. Aenean efficitur, velit sit amet feugiat euismod, nunc justo ultricies ligula, non consectetur sapien urna ut purus. Maecenas consequat neque nec nulla dapibus, eu bibendum est sodales.

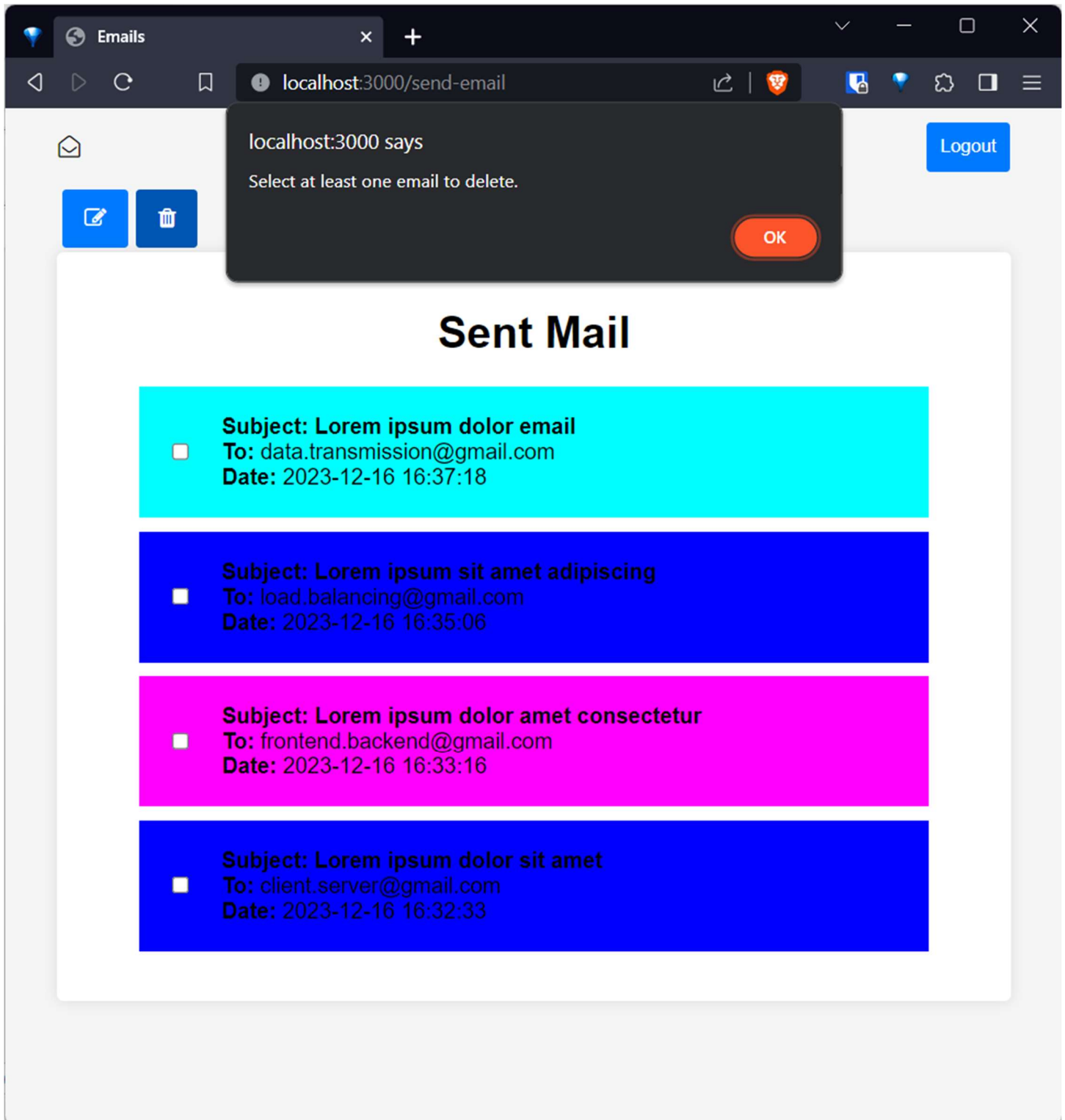
Send Email

- Access the "Compose" page after login.
- Fill out required fields: recipients, subject, and content.
- Click "Send" to dispatch your email.

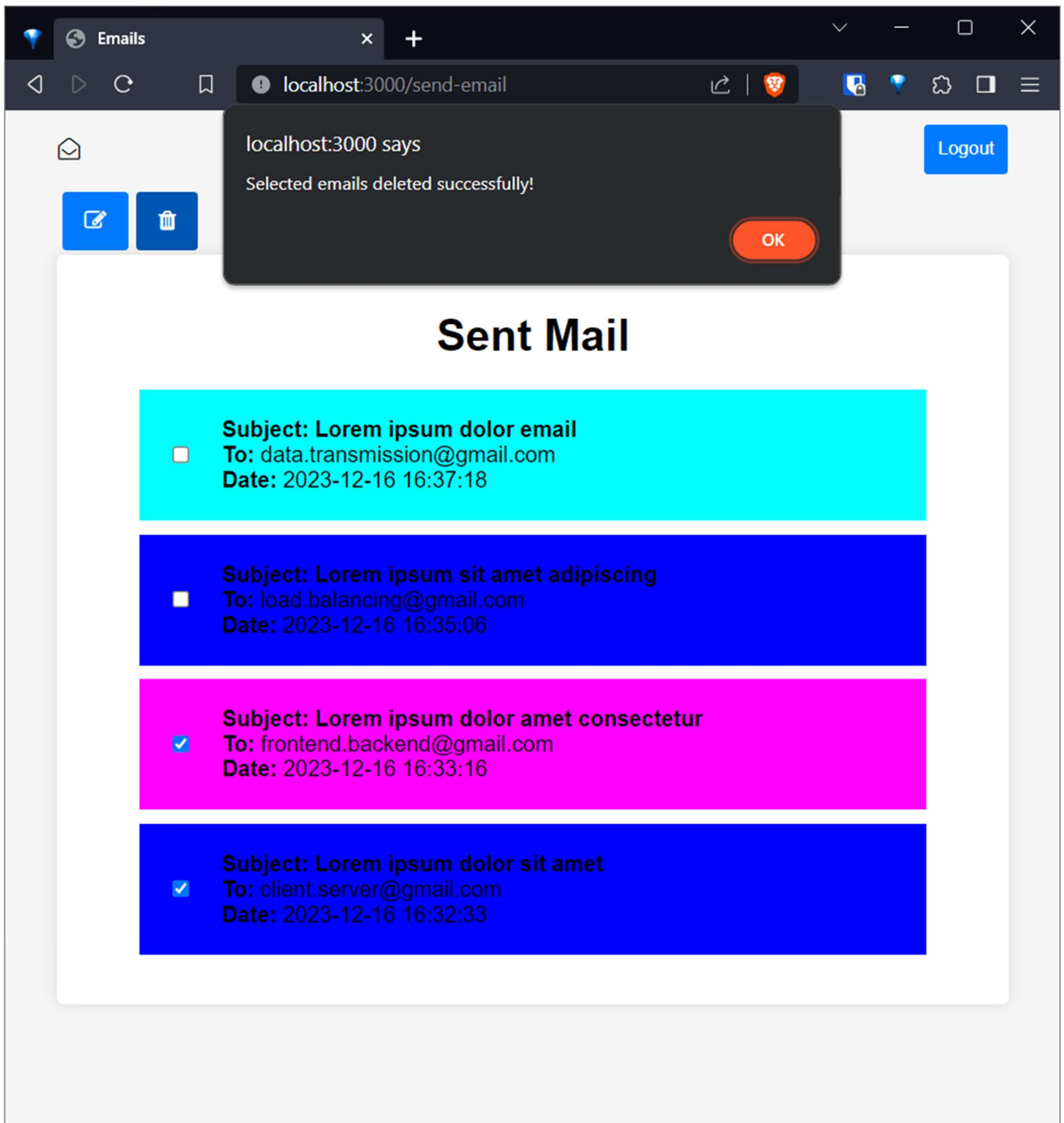


Once the email is sent, redirects to the outbox page. The sent email is displayed at the top.

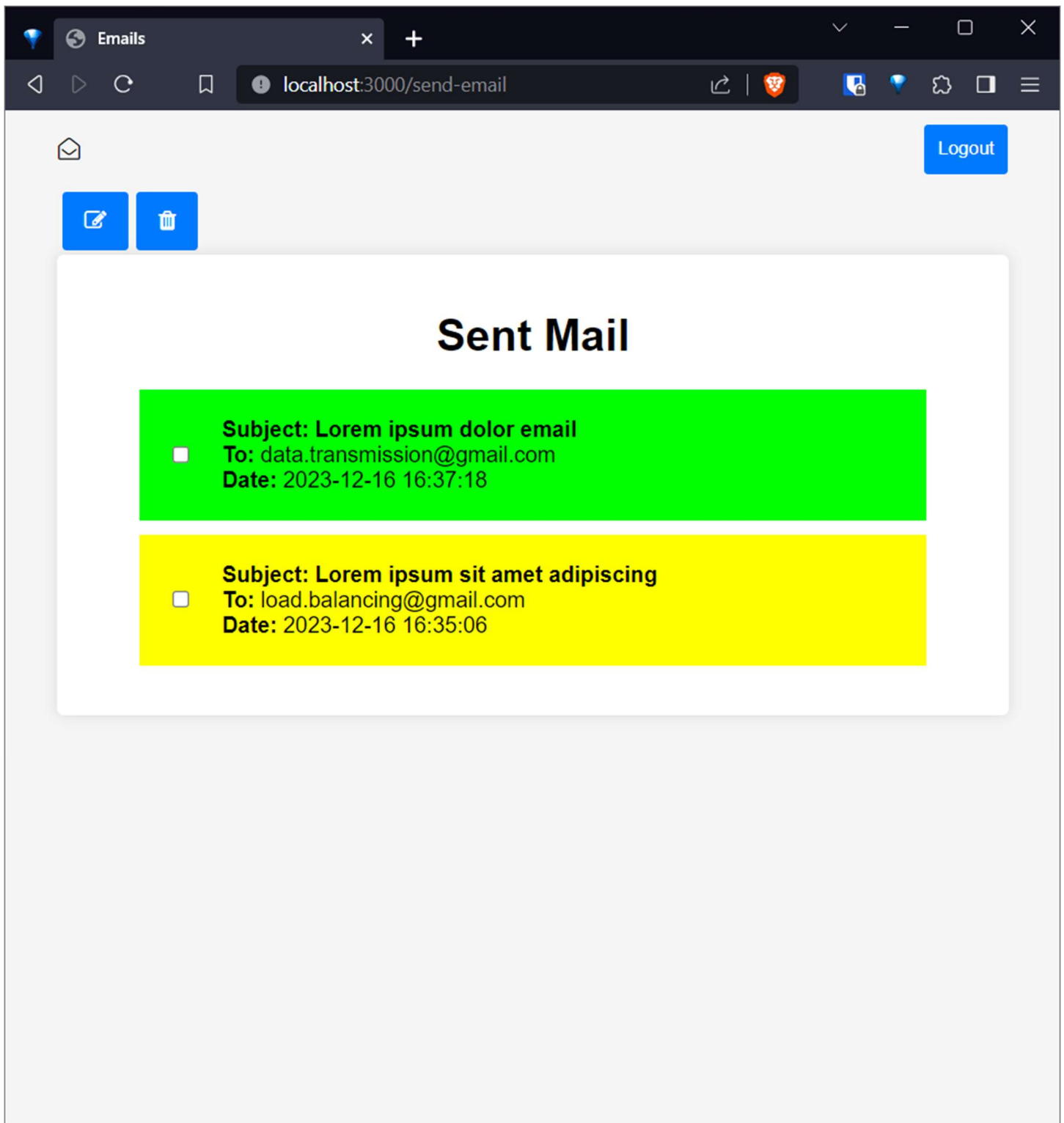
2.4 Delete Multiple



Ensure an email is selected. Receive an alert if no email is chosen.

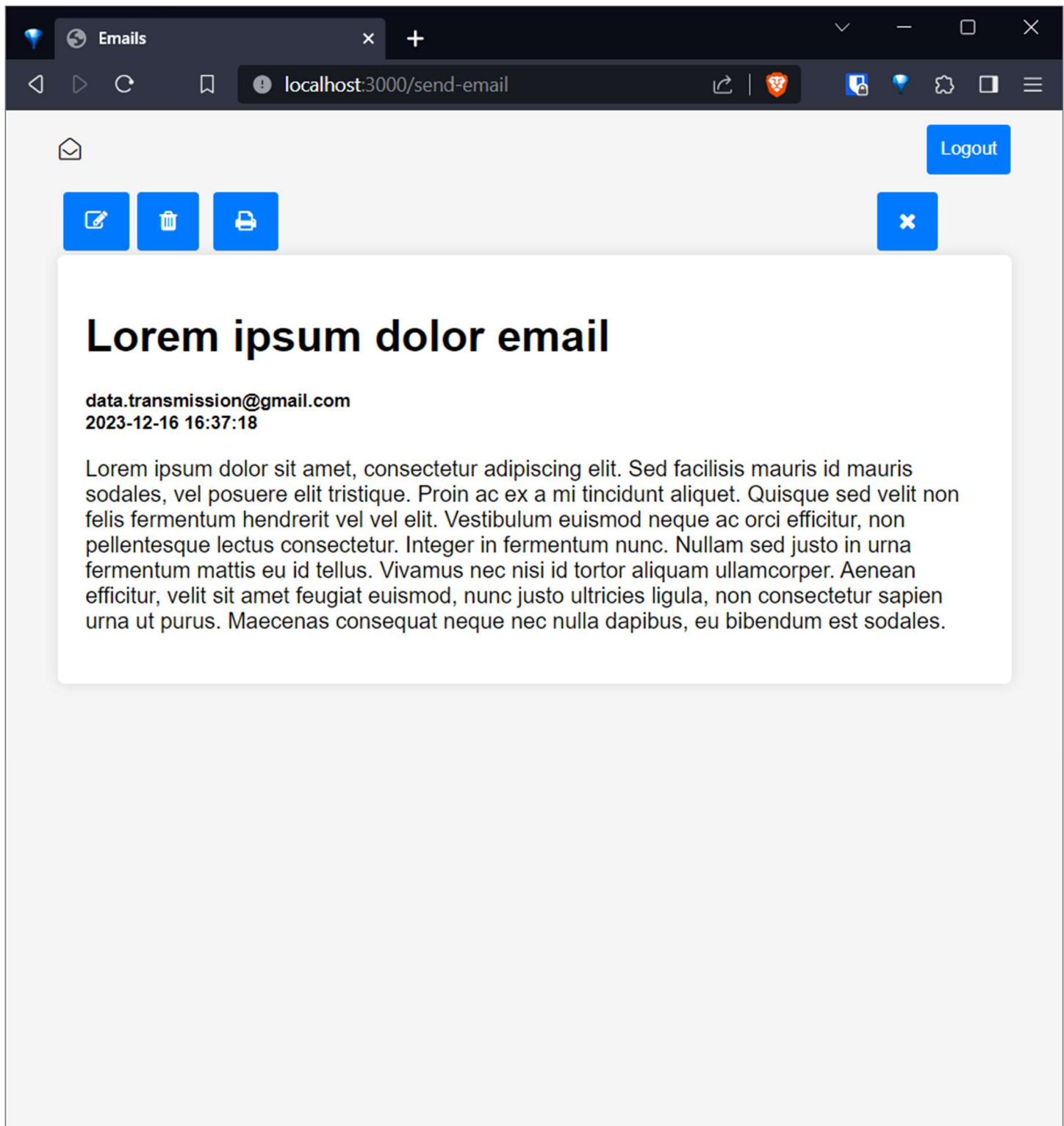


Selected emails are deleted on click.



The outbox is refreshed.

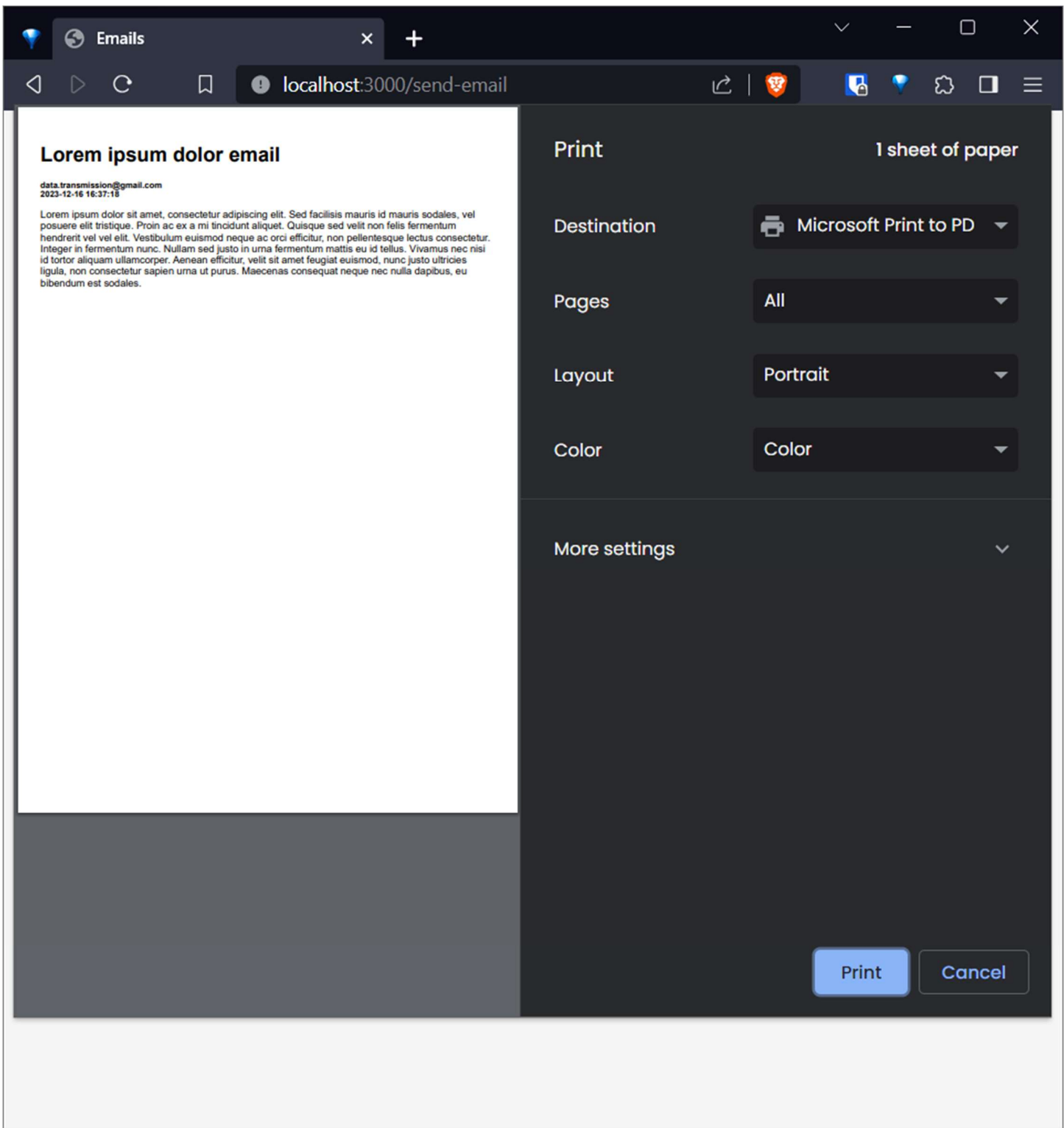
2.5 Email Details



When you click on an email, its contents are displayed. This page has the following buttons:

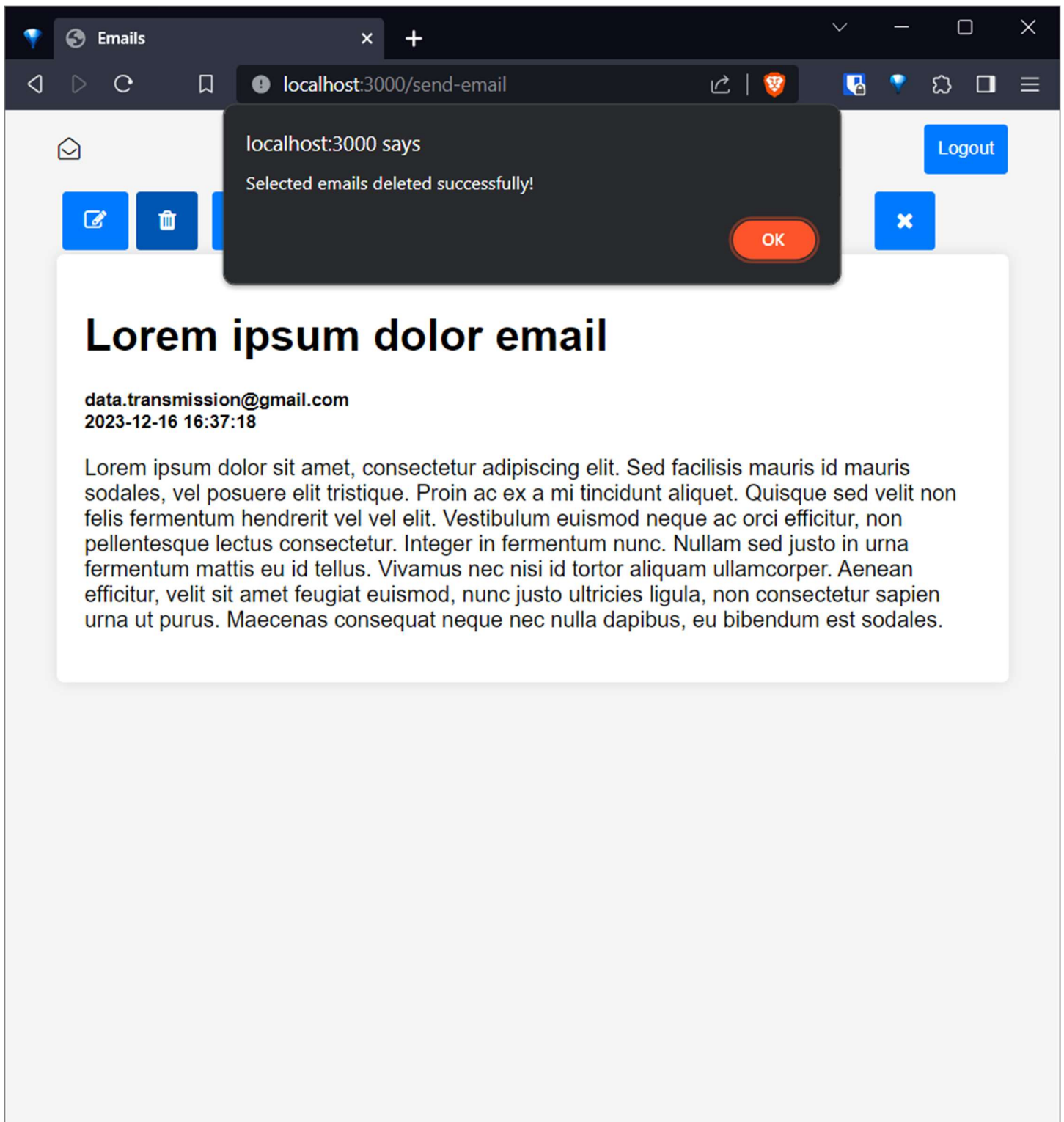
1. Compose: Compose an email.
2. Delete: Deletes the emails that are selected. Requires at least one email to be selected.
3. Print: Prints the email content.
4. Close: Returns to outbox.
5. Logout: Logs out of the application. Displays login page.

2.6 Print



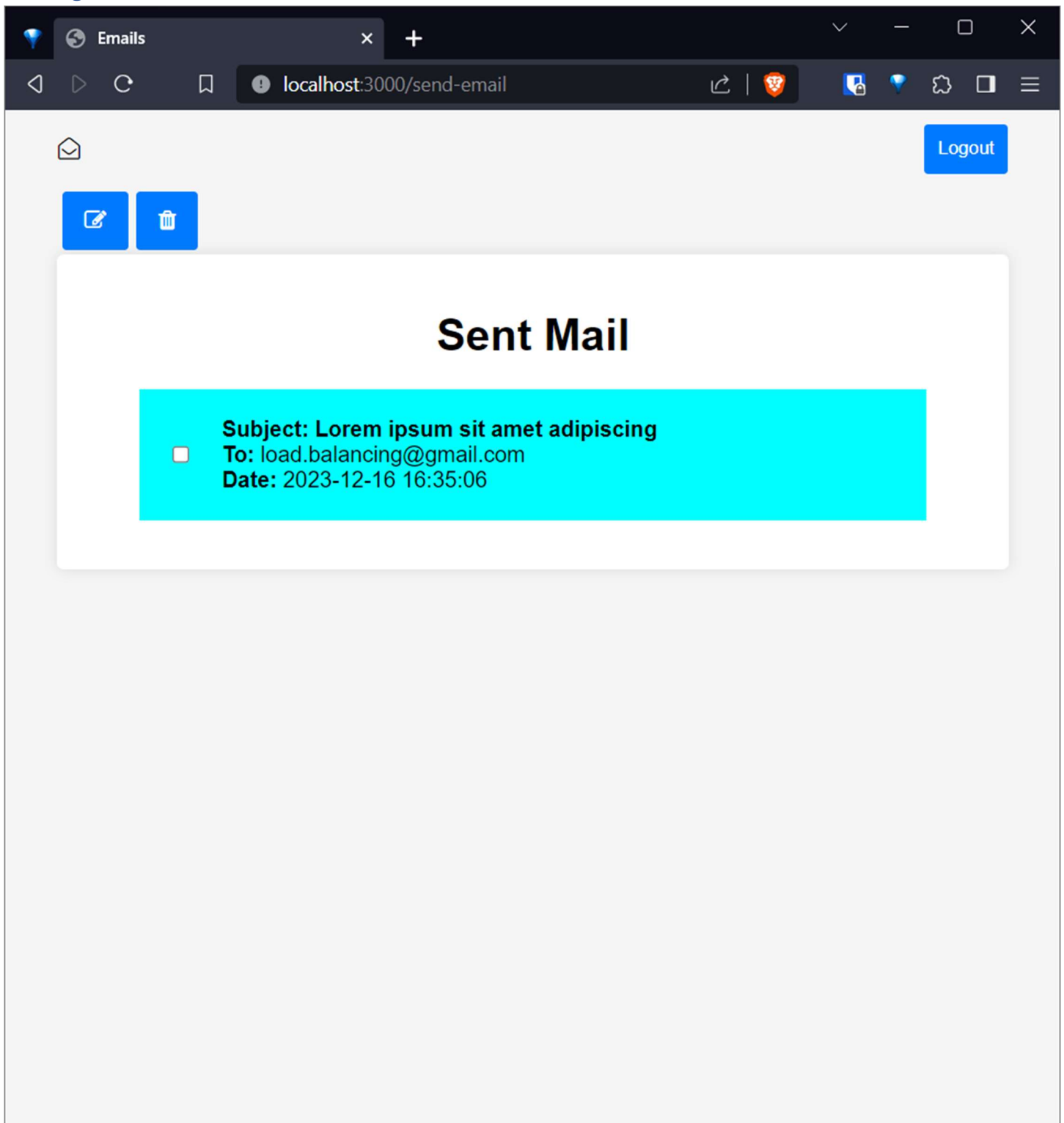
1. Click the "Print" button.
2. The print dialog box appears, printing the selected email.

2.7 Delete

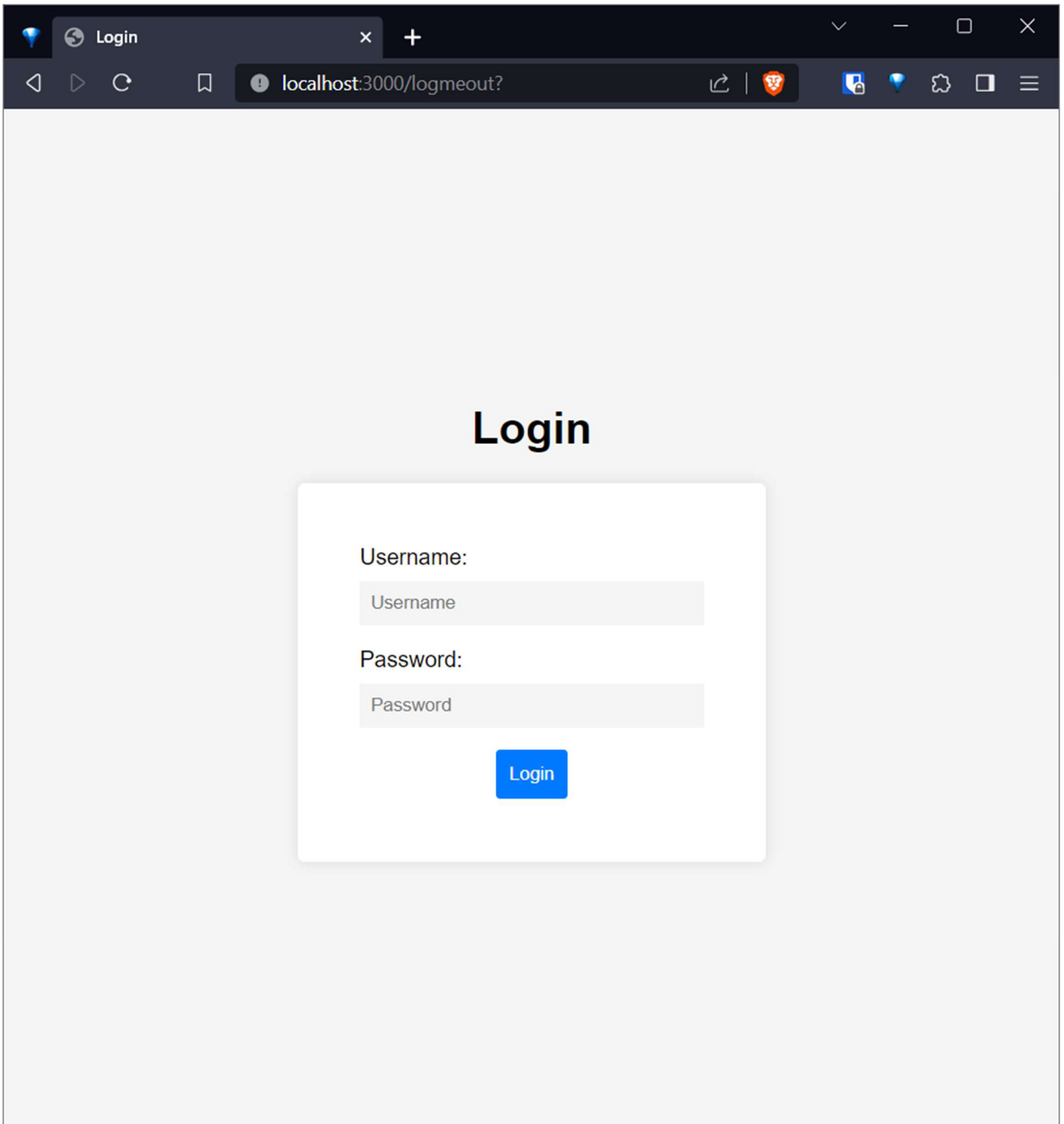


The opened email is deleted when the delete button is pressed.

2.8 Logout



The outbox is refreshed.



Login

localhost:3000/logmeout?

Login

Username:

Password:

Login

Login page is displayed when logout button is clicked.

3. Technical Details

3.1 Overview

Envelope is a web-based email client application designed to streamline email communication. The application's technical foundation leverages HTML for the front end, providing users with an intuitive interface accessible through standard web browsers. Communication between the client and the server is facilitated by HTTP methods, including GET, POST, and DELETE requests.

3.2 Front-End Technologies

- **HTML:** Envelope's front-end is built using HTML, ensuring a structured and user-friendly interface for efficient user interaction.
- **CSS:** Cascading Style Sheets are used to provide a beautiful user interface that is intuitive to use.
- **Client-Side JavaScript:** JavaScript is utilized on the client side to handle validation, error detection, and facilitate communication with the server. This dynamic scripting enhances user experience by enabling responsive interactions within the web browser.

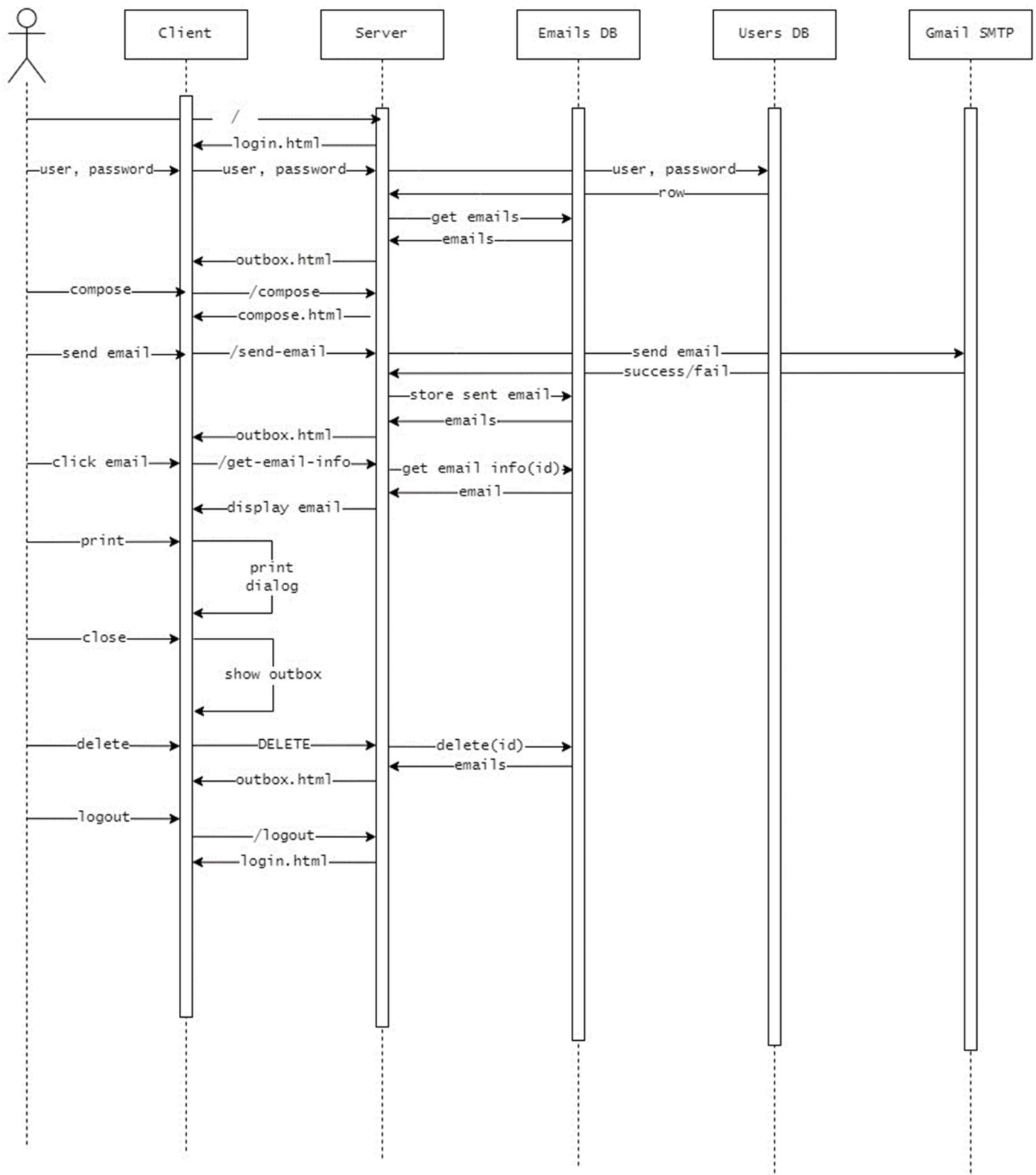
3.3 Back-End Technologies

- **JavaScript (Node.js Express):** Envelope's server-side functionality is powered by JavaScript using Node.js with the Express framework. This combination ensures effective handling of client requests and a smooth communication process.
- **Nodemailer:** Envelope integrates Nodemailer to interface with the Gmail SMTP server, enabling the secure and reliable sending of emails.
- **SQLite3 Database:** The server manages data using the SQLite3 database, employing two distinct databases:
 1. **emails.db:** Stores sent emails, allowing for efficient management of email records.
 2. **users.db:** Manages authorized users, enhancing security and access control.
- **Dotenv:** Envelope employs Dotenv to safeguard sensitive information such as usernames and passwords, ensuring confidentiality and security.

3.4 Communication

Envelope facilitates communication between the client and server through HTTP requests. This standardized approach ensures the seamless flow of data and commands, contributing to the application's efficiency.

3.5 Flow of the application



This flowchart shows the various ways the user can interact with the application, and how the interactions are processed. It shows the various components of the application and how they interact with each other.

4. Components

4.1 Server

The server, built with JavaScript and the Express framework, employs robust security measures in its operations. Prior to responding to any request, it verifies the user's login status, enhancing the overall security of the application. The server relies on a database to store crucial information, implementing comprehensive error-handling procedures to safeguard against unexpected client requests or database responses. These measures ensure the stability, integrity, and security of the application's server-side processes.

```
const express = require('express');
const nodemailer = require('nodemailer');
const bodyParser = require('body-parser');
const sqlite3 = require('sqlite3').verbose();
const dotenv = require('dotenv').config();

const app = express();
const port = 3000;
let loggedIn = 0;

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
```

Include required files.

```
// Connect to SQLite3 database
const db = new sqlite3.Database('emails.db');
const udb = new sqlite3.Database('users.db');

// Create Email table
db.run(`
  CREATE TABLE IF NOT EXISTS emails (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    to_email TEXT,
    subject TEXT,
    message TEXT,
    timestamp TEXT
  )
`);

// Create Users table
udb.run(`
  CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT,
    password TEXT
  )
`);

// udb.run("INSERT INTO users (username, password) VALUES ('omi','meatballs')")
```

Create the database and required tables.

```

// Serve the login page
app.get('/', (req, res) => {
  if(loggedIn == 0){
    res.sendFile(__dirname + '/login.html');
  }
  else {
    res.sendFile(__dirname + '/outbox.html');
  }
});

// Serve the login page
app.get('/outbox', (req, res) => {
  if(loggedIn == 0){
    res.sendFile(__dirname + '/login.html');
  }
  else {
    res.sendFile(__dirname + '/outbox.html');
  }
});

// Handle login
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  udb.get('SELECT * FROM users WHERE username = ? and password = ?', [username, password], (err, row) => {
    if (err) {
      console.log(err)
    }

    if (!row) {
      console.log("Login failed");
      res.status(401).send('Invalid username or password');
      // res.json({ message: 'Login failed!' });
    }

    else {
      res.sendFile(__dirname + '/outbox.html');
      // res.json({ success: true, message: 'Login successful' });
      console.log("Login Successful");
      loggedIn = 1;
      // res.json(row);
    }
  });
});

// Logout function
app.get('/logmeout', (req, res) => {
  loggedIn = 0;
  console.log(loggedIn);
});

```

```

console.log("Please log in")
res.sendFile(__dirname + '/login.html');
});

```

Handle login and logout functionality, using POST request and SQL to interface with server.

```

// Serve the compose page
app.get('/compose', (req, res) => {
  res.sendFile(__dirname + '/compose.html');
});

// Handle POST request to send emails
app.post('/send-email', async (req, res) => {
  const { to, subject, text } = req.body;
  console.log(loggedIn);

  const transporter = nodemailer.createTransport({
    service: 'gmail',
    auth: {
      user: process.env.MY_EMAIL,
      pass: process.env.MY_PASSWORD,
    },
  });

  const mailOptions = {
    from: 'Tanmay <tanmaynedu@gmail.com>',
    to,
    subject,
    text,
  };

  // Send email
  if(loggedIn == 1){
    try {
      await transporter.sendMail(mailOptions);
      console.log('Email sent successfully!');
      res.sendFile(__dirname + '/outbox.html');
    } catch (error) {
      console.error(error);
      console.log('Error sending email');
    }
    storeEmail(to, subject, text)
  }
  else {
    console.log("Please log in")
    res.sendFile(__dirname + '/login.html');
  }
});

```

Send emails using Nodemailer and the gmail SMTP service.

- The server receives data via a POST request. This request contains the to, subject, and body of the email to be sent.
- This email is then sent using Nodemailer and stored to the database using the following function.

```
function storeEmail(to, subject, text){
  console.log('Saving email to Database');

  // Store email in the SQLite3 database
  db.run(
    "INSERT INTO emails (to_email, subject, message, timestamp) VALUES (?, ?, ?, datetime('now', 'localtime'))",
    [to, subject, text],
    (err) => {
      if (err) {
        // return res.status(500).send(err.toString());
        console.log("error saving to db")
      }

      // res.status(200).send('Email sent and stored successfully!');
    }
  );
}
```

Store sent emails to the database using SQLite3.

```
// Route to get all sent emails
app.get('/get-sent-emails', (req, res) => {
  if (loggedIn == 0) {
    console.log("Please log in")
    res.sendFile(__dirname + '/login.html');
  } else {
    db.all('SELECT * FROM emails', (err, rows) => {
      if (err) {
        return res.status(500).send(err.toString());
      }
      else {
        res.json(rows);
      }
    });
  }
});

// Route to get a single email by ID
app.get('/email/:id', (req, res) => {
  const emailId = req.params.id;

  if (loggedIn == 0) {
    console.log("Please log in")
    res.sendFile(__dirname + '/login.html');
  } else {
```

```

db.get('SELECT * FROM emails WHERE id = ?', [emailId], (err, row) => {
  if (err) {
    return res.status(500).send(err.toString());
  }

  if (!row) {
    return res.status(404).json({ message: 'Email not found' });
  }

  else {
    res.json(row);
  }
});
}
});

```

Functions for the client to request emails from the server. Server fetches the emails from the database and sends them to the client.

- If the user isn't logged in, they are redirected to the login page.
- If the user is logged in, a request is made to the database to fetch the emails, and the emails are sent to the client as json data.

```

// Route to delete selected emails
app.delete('/delete-emails', (req, res) => {
  const emailIds = req.body.emailIds;

  if (!emailIds || !Array.isArray(emailIds) || emailIds.length === 0) {
    return res.status(400).json({ message: 'Invalid request. Please provide valid email IDs.' });
  }

  const placeholders = emailIds.map(() => '?').join(',');
  const query = `DELETE FROM emails WHERE id IN (${placeholders})`;

  db.run(query, emailIds, (err) => {
    if (err) {
      return res.status(500).send(err.toString());
    }

    res.json({ message: 'Selected emails deleted successfully!' });
  });
});

```

Function to delete emails from the database.

- If no emails are selected, an error message is returned.
- If there is at least one emailId to be deleted, a query is made to the database to delete the emails, and a success message is returned once emails are deleted.

```

// Debugging
app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
  console.log(process.env.MY_EMAIL);
});

```



```
});
```

Start debugging on the server terminal.

4.2 Client:

The client-side of the application is constructed using JavaScript and HTML, providing a dynamic and responsive user experience. It interacts with the server, sending requests for various actions such as login, viewing emails, composing new messages, and managing sent emails. The client also incorporates error-detection mechanisms to handle unexpected user actions. Through its integration with the server and interaction with the user, the client plays a crucial role in ensuring a smooth and secure communication channel between the Envelope application and its users.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
  <title>Emails</title>
  <style>
    @media print {
      body * {
        visibility: hidden;
      }

      #print-section, #print-section * {
        visibility: visible;
      }

      #print-section {
        position: absolute;
        left: 0;
        top: 0;
      }
    }
  </style>
</head>
```

CSS to handle print.

```
body {
  font-family: Arial, sans-serif;
  background-color: #f5f5f5;
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: start;
  height: 100vh;
}

#headerbar {
  min-width: 90%;
```

```
display: flex;
flex-direction: row;
align-items: center;
justify-content: space-between;
padding: 10px;
}

#navbar {
  min-width: 90%;
  display: flex;
  flex-direction: row;
  align-items: start;
}

#email-controls {
  min-width: 85%;
  display: flex;
  flex-direction: row;
  align-items: center;
  justify-content: space-between;
}

#outbox {
  background-color: #fff;
  padding: 20px;
  /* margin-top: 1rem; */
  border-radius: 5px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  text-align: center;
  width: 85%;
}

#email-list {
  list-style-type: none;
  align-items: start;
  justify-content: start;
  margin-right: 40px;
}

.email-box {
  /* color: #f5f5f5; */
  display: flex;
  flex-direction: row;
  margin: 10px;
  padding: 10px;
}

.email{
```

```
/* color: #f5f5f5; */
display: block;
/* flex-direction: column; */
/* margin: 10px; */
padding: 10px;
text-align: left;
margin-left: 10px;
width: 100%;
}

#email {
  background-color: #fff;
  padding: 20px;
  /* margin-top: 1rem; */
  border-radius: 5px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  text-align: left;
  width: 85%;
}

#print-section {
  width: 100%;
  display: flex;
  flex-direction: column;
  text-align: left;
  align-items: left;
}

#print-section h3, h5 {
  margin: 0;
}

#logout_but {
  background-color: #007BFF;
  color: #fff;
  padding: 10px;
  border: none;
  border-radius: 3px;
  cursor: pointer;
}

#logout_but:hover {
  background-color: #0056b3;
}

.btn {
  background-color: #007BFF;
  border: none;
  color: white;
  padding: 12px 16px;
```

```

border-radius: 3px;
margin: 3px;
font-size: 16px;
cursor: pointer;
}

/* Darker background on mouse-over */
.btn:hover{
  background-color: #0056b3;
}

</style>
</head>

```

CSS to define how the webpage will look.

```

<body>
  <div id="headerbar">
    <i class="fa fa-Envelope -open-o" aria-hidden="true"></i>
    <form action="logout" method="get">
      <button id="logout_but" type="submit">Logout</button>
    </form>
  </div>

  <div id="navbar">
    <form action="compose" method="get">
      <button type="submit" class="btn"><i class="fa fa-pencil-square-o"></i></button>
    </form>
    <button id="del_but" onclick="deleteSelectedEmails()" class="btn"><i class="fa fa-trash"></i></button>

    <div id="email-controls" style="display: none;">
      <div>
        <button id="delete_button" onclick="deleteSelectedEmail()" class="btn"><i class="fa fa-trash"></i></button>
        <button onclick="printContent()" class="btn"><i class="fa fa-print"></i></button>
      </div>
      <div>
        <button id="close_me" onclick="showOutbox()" class="btn"><i class="fa fa-close"></i></button>
      </div>
    </div>

  </div>

  <div id="outbox">
    <h1>Sent Mail</h1>
    <ul id="email-list"></ul>
  </div>

  <div id="email" style="display: none;">
    <main id="print-section">

```

```

<div class = "print-line">
  <h1 id = "subject"></h1>
</div>
<div class = "print-line">
  <h5 id = "to"></h5>
</div>
<div class = "print-line">
  <h5 id = "timestamp"></h5>
</div>
<div class = "print-line">
  <p id = "message"></p>
</div>
</main>

</div>

```

HTML to define the structure of the webpage.

- Various event handlers, like onclick are used to make the webpage interactive and dynamic, assuring a smooth user experience.

```

<script>

let cur_email = 0;

// Function to fetch and display emails
async function fetchEmails() {
  const response = await fetch('/get-sent-emails');
  const emails = await response.json();

  const emailList = document.getElementById('email-list');
  emailList.innerHTML = '';

  emails.forEach(email => {
    const listItem = document.createElement('li');
    listItem.innerHTML = `
      <div class = "email-box" >
        <input type="checkbox" class="email-checkbox" data-id="${email.id}">
        <div class = "email" onclick="showEmailDetail(${email.id})">
          <strong>Subject: ${email.subject} </strong><br>
          <strong>To:</strong> ${email.to_email}<br>
          <strong>Date:</strong> ${email.timestamp}<br>

        <div>
      </div>

    `;
    assignRandomColor(listItem);
    emailList.insertBefore(listItem, emailList.firstChild);
  });
}

```

```
fetchEmails();
```

Function to fetch and display emails.

- A request of /get-sent-emails is made to the server, and the client waits for the server to respond.
- The response is stored in the variable emails, and this variable is the parsed and its content is appended to the emailList division defined in HTML.
- A random color is assigned to the email using the following function.

```
function assignRandomColor(element) {
  // List of possible colors
  const colors = ["#ff0000", "#00ff00", "#0000ff", "#ffff00", "#ff00ff", "#00ffff"];

  // Get a random index from the colors array
  const randomIndex = Math.floor(Math.random() * colors.length);

  // Get the color from the array
  const randomColor = colors[randomIndex];

  // Set the background color of the element
  element.style.backgroundColor = randomColor;
}
```

Function to give element a random color.

```
// Function to fetch and display a particular email
async function showEmailDetail(id){
  cur_email = id
  const response = await fetch('/email/' + id);
  const email = await response.json();
  console.log(email);
  document.getElementById('subject').innerHTML = `${email.subject}`
  document.getElementById('to').innerHTML = `${email.to_email}`
  document.getElementById('timestamp').innerHTML = `${email.timestamp}`
  document.getElementById('message').innerHTML = `${email.message}`
  showEmailPage()
  console.log("i ran", id);
}
```

Function to show details of a particular email.

- The details of a particular email are fetched from the server using the /email/id route.
- The contents of the response are parsed and displayed on the webpage.

```
// Function to print contents of email
function printContent() {
  window.print();
}
```

Print the email.

```
// Function to delete selected emails
async function deleteSelectedEmails() {
  const checkboxes = document.querySelectorAll('.email-checkbox:checked');
```

```

const emailIds = Array.from(checkboxes).map(checkbox => checkbox.dataset.id);
console.log(emailIds)

if (emailIds.length === 0) {
  alert('Select at least one email to delete.');
```

return;

}

```

const response = await fetch('/delete-emails', {
  method: 'DELETE',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({ emailIds }),
});

const result = await response.json();
alert(result.message);

// Refresh the email list after deletion
fetchEmails();
}

// del sing email
async function deleteSelectedEmail() {
const emailIds = [cur_email];

console.log("hererere")
const response = await fetch('/delete-emails', {
  method: 'DELETE',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({ emailIds }),
});

const result = await response.json();
alert(result.message);

// Refresh the email list after deletion
fetchEmails();
showOutbox();
}
```

Function to delete emails.

- The ids of the selected emails are put into a variable emailIds. This variable is sent to the server along with a delete request.
- Once the server sends an ok response, the outbox is refreshed.

```
function showOutbox(){
```

```

document.getElementById('email').style.display = 'none';
document.getElementById('email-controls').style.display = 'none';
document.getElementById('outbox').style.display = 'block';
document.getElementById('del_but').style.display = 'block';
}

function showEmailPage(){
  document.getElementById('email').style.display = 'flex';
  document.getElementById('email-controls').style.display = 'flex';
  document.getElementById('outbox').style.display = 'none';
  document.getElementById('del_but').style.display = 'none';
}

</script>
</body>
</html>

```

Functions to display appropriate components of the webpage.

5. Illustrations and explanations of the code

5.1 Syntax

Express Framework Initialization:

```

const express = require('express');
const app = express();

```

5.2 Control Structures

```

app.get('/', (req, res) => {
  if(loggedIn == 0){
    res.sendFile(__dirname + '/login.html');
  }
  else{
    res.sendFile(__dirname + '/outbox.html');
  }
});

```

5.3 Data Structures – Storage Folders

```

// Connect to SQLite3 database
const db = new sqlite3.Database('emails.db');
const udb = new sqlite3.Database('users.db');

```

5.4 I/O

```

<div id="compose">
  <form action="send-email" method="post">
    <label for="to">To:</label>
    <input title="email" type="email" name="to" required>

```



```

<br>
<label for="subject">Subject:</label>
<input title="subject" type="text" name="subject" required>
<br>
<label for="text">Message:</label>
<textarea title="text" name="text" rows="10" required></textarea>
<br>
<button id = "send" type="submit">Send Email</button>
</form>

```

5.5 Error Handling Procedures

```

udb.get('SELECT * FROM users WHERE username = ? and password = ?', [username, password], (err, row) => {
  if (err) {
    console.log(err)
  }

  if (!row) {
    console.log("Login failed");
    res.status(401).send('Invalid username or password');
    // res.json({ message: 'Login failed!' });
  }
}

```

5.6 JavaScript

```

function printContent() {
  window.print();
}

```

5.7 CSS

```

body{
  font-family: Arial, sans-serif;
  background-color: #f5f5f5;
  margin: 0;
  padding: 0;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: start;
  height: 100vh;
}

```

5.8 Event Handlers

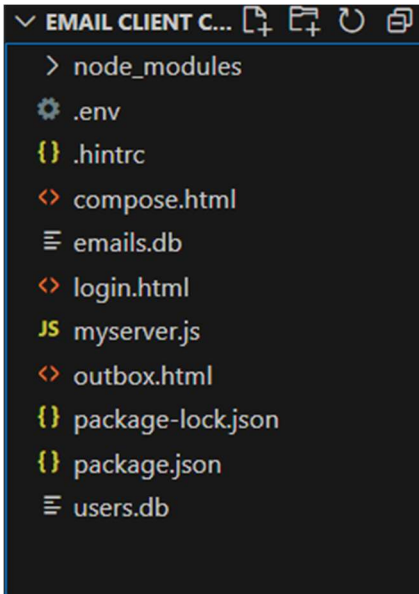
```

<div>
  <button id="delete_button" onclick="deleteSelectedEmail()" class="btn"><i class="fa fa-trash"></i></button>
  <button onclick="printContent()" class="btn"><i class="fa fa-print"></i></button>
</div>

```

6. Appendix

6.1 Structure



6.2 package.json

```
{
  "name": "email-client-clean",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "nodemon myserver.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.2",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "nodemailer": "^6.9.7",
    "nodemon": "^3.0.2",
    "sqlite3": "^5.1.6"
  }
}
```

6.3 myserver.json

```
const express = require('express');
const nodemailer = require('nodemailer');
const bodyParser = require('body-parser');
const sqlite3 = require('sqlite3').verbose();
const dotenv = require('dotenv').config();
```

```

const app = express();
const port = 3000;
let loggedIn = 0;

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

// Connect to SQLite3 database
const db = new sqlite3.Database('emails.db');
const udb = new sqlite3.Database('users.db');

// Create Email table
db.run(`
  CREATE TABLE IF NOT EXISTS emails (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    to_email TEXT,
    subject TEXT,
    message TEXT,
    timestamp TEXT
  )
`);

// Create Users table
udb.run(`
  CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT,
    password TEXT
  )
`);

// udb.run("INSERT INTO users (username, password) VALUES ('omi','meatballs')")

// Serve the login page
app.get('/', (req, res) => {
  if(loggedIn == 0){
    res.sendFile(__dirname + '/login.html');
  }
  else {
    res.sendFile(__dirname + '/outbox.html');
  }
});

// Serve the login page
app.get('/outbox', (req, res) => {
  if(loggedIn == 0){
    res.sendFile(__dirname + '/login.html');
  }
  else {

```

```

    res.sendFile(__dirname + '/outbox.html');
  }
});

// Handle login
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  udb.get('SELECT * FROM users WHERE username = ? and password = ?', [username, password], (err, row) => {
    if (err) {
      console.log(err)
    }

    if (!row) {
      console.log("Login failed");
      res.status(401).send('Invalid username or password');
      // res.json({ message: 'Login failed!' });
    }

    else {
      res.sendFile(__dirname + '/outbox.html');
      // res.json({ success: true, message: 'Login successful' });
      console.log("Login Successful");
      loggedIn = 1;
      // res.json(row);
    }
  });
});

// Logout function
app.get('/logmeout', (req, res) => {
  loggedIn = 0;
  console.log(loggedIn);
  console.log("Please log in")
  res.sendFile(__dirname + '/login.html');
});

// Serve the compose page
app.get('/compose', (req, res) => {
  res.sendFile(__dirname + '/compose.html');
});

// Handle POST request to send emails
app.post('/send-email', async (req, res) => {
  const { to, subject, text } = req.body;
  console.log(loggedIn);

```

```

const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: process.env.MY_EMAIL,
    pass: process.env.MY_PASSWORD,
  },
});

const mailOptions = {
  from: 'Tanmay <tanmaynedu@gmail.com>',
  to,
  subject,
  text,
};

// Send email
if(loggedIn == 1){
  try {
    await transporter.sendMail(mailOptions);
    console.log('Email sent successfully!');
    res.sendFile(__dirname + '/outbox.html');
  } catch (error) {
    console.error(error);
    console.log('Error sending email');
  }
  storeEmail(to, subject, text)
}
else {
  console.log("Please log in")
  res.sendFile(__dirname + '/login.html');
}
});

function storeEmail(to, subject, text){
  console.log('Saving email to Database');

  // Store email in the SQLite3 database
  db.run(
    "INSERT INTO emails (to_email, subject, message, timestamp) VALUES (?, ?, ?, datetime('now', 'localtime'))",
    [to, subject, text],
    (err) => {
      if (err) {
        // return res.status(500).send(err.toString());
        console.log("error saving to db")
      }

      // res.status(200).send('Email sent and stored successfully!');
    }
  );
}

```

```

}

// Route to get all sent emails
app.get('/get-sent-emails', (req, res) => {
  if (loggedIn == 0) {
    console.log("Please log in")
    res.sendFile(__dirname + '/login.html');
  } else {
    db.all('SELECT * FROM emails', (err, rows) => {
      if (err) {
        return res.status(500).send(err.toString());
      }
      else {
        res.json(rows);
      }
    });
  }
});

// Route to get a single email by ID
app.get('/email/:id', (req, res) => {
  const emailId = req.params.id;

  if (loggedIn == 0) {
    console.log("Please log in")
    res.sendFile(__dirname + '/login.html');
  } else {
    db.get('SELECT * FROM emails WHERE id = ?', [emailId], (err, row) => {
      if (err) {
        return res.status(500).send(err.toString());
      }

      if (!row) {
        return res.status(404).json({ message: 'Email not found' });
      }

      else {
        res.json(row);
      }
    });
  }
});

// Route to delete selected emails
app.delete('/delete-emails', (req, res) => {
  const emailIds = req.body.emailIds;

  if (!emailIds || !Array.isArray(emailIds) || emailIds.length === 0) {

```

```

    return res.status(400).json({ message: 'Invalid request. Please provide valid email IDs.' });
  }

  const placeholders = emailIds.map(() => '?').join(',');
  const query = `DELETE FROM emails WHERE id IN (${placeholders})`;

  db.run(query, emailIds, (err) => {
    if (err) {
      return res.status(500).send(err.toString());
    }

    res.json({ message: 'Selected emails deleted successfully!' });
  });
});

// Debugging
app.listen(port, () => {
  console.log(`Server is running at http://localhost:${port}`);
  console.log(process.env.MY_EMAIL);
});

```

6.4 Login.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Login</title>

  <style>
    body{
      font-family: Arial, sans-serif;
      background-color: #f5f5f5;
      margin: 0;
      padding: 0;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      height: 100vh;
    }

    #login-container {
      background-color: #fff;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      text-align: center;
      width: 300px;
    }
  </style>

```

```

}

form {
  min-width: 50%;
  margin: 0 auto;
  padding: 25px;
}

label {
  display: block;
  margin-bottom: 8px;
  text-align: left;
}

input,
textarea {
  width: 100%;
  padding: 8px;
  margin-bottom: 16px;
  box-sizing: border-box;
  border: none;
  background-color: #f5f5f5;
}

#login_but {
  background-color: #007BFF;
  color: #fff;
  padding: 10px;
  border: none;
  border-radius: 3px;
  cursor: pointer;
}

#login_but:hover {
  background-color: #0056b3;
}
</style>
</head>

<body>
  <h1>Login</h1>
  <div id="login-container">

    <form id="login-form" action="login" method="post">
      <label>Username:</label>
      <input type="text" name="username" placeholder="Username" required>
      <br>
      <label>Password:</label>
      <input type="password" name="password" placeholder="Password" required>

```



```

    <br>
    <button id="login_but" type="submit">Login</button>
  </form>
</div>
</body>

</html>

```

6.5 Outbox.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
  <title>Emails</title>
  <style>
    @media print {
      body * {
        visibility: hidden;
      }

      #print-section, #print-section * {
        visibility: visible;
      }

      #print-section {
        position: absolute;
        left: 0;
        top: 0;
      }
    }

    body {
      font-family: Arial, sans-serif;
      background-color: #f5f5f5;
      margin: 0;
      padding: 0;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: start;
      height: 100vh;
    }

    #headerbar {
      min-width: 90%;
      display: flex;
      flex-direction: row;

```

```
    align-items: center;
    justify-content: space-between;
    padding: 10px;
}

#navbar {
    min-width: 90%;
    display: flex;
    flex-direction: row;
    align-items: start;
}

#email-controls{
    min-width: 85%;
    display: flex;
    flex-direction: row;
    align-items: center;
    justify-content: space-between;
}

#outbox {
    background-color: #fff;
    padding: 20px;
    /* margin-top: 1rem; */
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    text-align: center;
    width: 85%;
}

#email-list {
    list-style-type: none;
    align-items: start;
    justify-content: start;
    margin-right: 40px;
}

.email-box {
    /* color: #f5f5f5; */
    display: flex;
    flex-direction: row;
    margin: 10px;
    padding: 10px;
}

.email{
    /* color: #f5f5f5; */
    display: block;
```

```
/* flex-direction: column; */
/* margin: 10px; */
padding: 10px;
text-align: left;
margin-left: 10px;
width: 100%;
}

#email{
  background-color: #fff;
  padding: 20px;
  /* margin-top: 1rem; */
  border-radius: 5px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  text-align: left;
  width: 85%;
}

#print-section{
  width: 100%;
  display: flex;
  flex-direction: column;
  text-align: left;
  align-items: left;
}

#print-section h3,h5{
  margin: 0;
}

#logout_but{
  background-color: #007BFF;
  color: #fff;
  padding: 10px;
  border: none;
  border-radius: 3px;
  cursor: pointer;
}

#logout_but:hover{
  background-color: #0056b3;
}

.btn{
  background-color: #007BFF;
  border: none;
  color: white;
  padding: 12px 16px;
  border-radius: 3px;
  margin: 3px;
```

```

    font-size: 16px;
    cursor: pointer;
}

/* Darker background on mouse-over */
.btn:hover{
    background-color: #0056b3;
}

</style>
</head>

<body>
  <div id="headerbar">
    <i class="fa fa-Envelope -open-o" aria-hidden="true"></i>
    <form action="logout" method="get">
      <button id = "logout_but" type="submit">Logout</button>
    </form>
  </div>

  <div id = "navbar">
    <form action="compose" method="get">
      <button type="submit" class="btn"><i class="fa fa-pencil-square-o"></i></button>
    </form>
    <button id = "del_but" onclick="deleteSelectedEmails()" class="btn"><i class="fa fa-trash"></i></button>

    <div id="email-controls" style="display: none;">
      <div>
        <button id="delete_button" onclick="deleteSelectedEmail()" class="btn"><i class="fa fa-trash"></i></button>
        <button onclick="printContent()" class="btn"><i class="fa fa-print"></i></button>
      </div>
      <div>
        <button id="close_me" onclick="showOutbox()" class="btn"><i class="fa fa-close"></i></button>
      </div>
    </div>

  </div>

  <div id="outbox">
    <h1>Sent Mail</h1>
    <ul id="email-list"></ul>
  </div>

  <div id="email" style="display: none;">
    <main id="print-section">
      <div class = "print-line">
        <h1 id = "subject"></h1>
      </div>

```

```

<div class = "print-line">
  <h5 id = "to"></h5>
</div>
<div class = "print-line">
  <h5 id = "timestamp"></h5>
</div>
<div class = "print-line">
  <p id = "message"></p>
</div>
</main>

</div>

<script>

let cur_email = 0;

// Function to fetch and display emails
async function fetchEmails() {
  const response = await fetch('/get-sent-emails');
  const emails = await response.json();

  const emailList = document.getElementById('email-list');
  emailList.innerHTML = '';

  emails.forEach(email => {
    const listItem = document.createElement('li');
    listItem.innerHTML = `
      <div class = "email-box" >
        <input type="checkbox" class="email-checkbox" data-id="${email.id}">
        <div class = "email" onclick="showEmailDetail(${email.id})">
          <strong>Subject: ${email.subject} </strong><br>
          <strong>To:</strong> ${email.to_email}<br>
          <strong>Date:</strong> ${email.timestamp}<br>

        <div>
      </div>

    `;
    assignRandomColor(listItem);
    emailList.insertBefore(listItem, emailList.firstChild);
  });
}

fetchEmails();

function assignRandomColor(element) {
  // List of possible colors
  const colors = ["#ff0000", "#00ff00", "#0000ff", "#ffff00", "#ff00ff", "#00ffff"];

  // Get a random index from the colors array

```

```

const randomIndex = Math.floor(Math.random() * colors.length);

// Get the color from the array
const randomColor = colors[randomIndex];

// Set the background color of the element
element.style.backgroundColor = randomColor;
}

// Function to fetch and display a particular email
async function showEmailDetail(id){
  cur_email = id
  const response = await fetch('/email/' + id);
  const email = await response.json();
  console.log(email);
  document.getElementById('subject').innerHTML = `${email.subject}`
  document.getElementById('to').innerHTML = `${email.to_email}`
  document.getElementById('timestamp').innerHTML = `${email.timestamp}`
  document.getElementById('message').innerHTML = `${email.message}`
  showEmailPage()
  console.log("i ran", id);
}

// Function to print contents of email
function printContent() {
  window.print();
}

// Function to delete selected emails
async function deleteSelectedEmails() {
  const checkboxes = document.querySelectorAll('.email-checkbox:checked');
  const emailIds = Array.from(checkboxes).map(checkbox => checkbox.dataset.id);
  console.log(emailIds)

  if (emailIds.length === 0) {
    alert('Select at least one email to delete.');
```

```

    return;
  }

  const response = await fetch('/delete-emails', {
    method: 'DELETE',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ emailIds }),
  });

  const result = await response.json();

```

```

alert(result.message);

// Refresh the email list after deletion
fetchEmails();
}

// del sing email
async function deleteSelectedEmail() {
  const emailIds = [cur_email];

  console.log("hererere")
  const response = await fetch('/delete-emails', {
    method: 'DELETE',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify({ emailIds }),
  });

  const result = await response.json();
  alert(result.message);

  // Refresh the email list after deletion
  fetchEmails();
  showOutbox();
}

function showOutbox(){
  document.getElementById('email').style.display = 'none';
  document.getElementById('email-controls').style.display = 'none';
  document.getElementById('outbox').style.display = 'block';
  document.getElementById('del_but').style.display = 'block';
}

function showEmailPage(){
  document.getElementById('email').style.display = 'flex';
  document.getElementById('email-controls').style.display = 'flex';
  document.getElementById('outbox').style.display = 'none';
  document.getElementById('del_but').style.display = 'none';
}

</script>
</body>
</html>

```

6.6 Compose.html

```

<!DOCTYPE html>
<html lang="en">

```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Compose</title>

  <style>
    body{
      font-family: Arial, sans-serif;
      background-color: #f5f5f5;
      margin: 0;
      padding: 0;
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: start;
      height: 100vh;
    }

    #compose{
      background-color: #fff;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      text-align: center;
      width: 80%;
    }

    form{
      min-width: 50%;
      margin: 0 auto;
      padding: 25px
    }

    label{
      display: block;
      margin-bottom: 8px;
      text-align: left;
    }

    input,
    textarea{
      width: 100%;
      padding: 8px;
      margin-bottom: 16px;
      box-sizing: border-box;
      border: none;
      background-color: #f5f5f5;
    }

    #send{
```



```
background-color: #007BFF;
color: #fff;
padding: 10px;
border: none;
border-radius: 3px;
cursor: pointer;
}

#send:hover{
background-color: #0056b3;
}
</style>
</head>
<body>
<h1>Compose</h1>
<div id="compose">
<form action="send-email" method="post">
<label for="to">To:</label>
<input title="email" type="email" name="to" required>
<br>
<label for="subject">Subject:</label>
<input title="subject" type="text" name="subject" required>
<br>
<label for="text">Message:</label>
<textarea title="text" name="text" rows="10" required></textarea>
<br>
<button id = "send" type="submit">Send Email</button>
</form>

</div>
</body>
</html>
```