Project Documentation: Development of Email Notification Feature POC

Objective:

The objective of this task is to create a proof of concept (POC) for an email notification feature within an application, enabling it to send automated emails to users under specified conditions such as registration confirmation and password reset.

Requirements:

- Research and select an email delivery service or SMTP server.
- Implement a mechanism to trigger email notifications under predefined conditions.
- Ensure dynamic content support in emails, including personalized user information.
- Design email templates for at least two scenarios (e.g., welcome email, password reset).
- Implement proper error handling and logging for email delivery status.

Configuration Process for Google's SMTP Server:

The configuration process for integrating Google's SMTP server into our backend application for sending emails. By following these steps, you can ensure seamless communication with your users via email.

Step 1: Choose SMTP Server

Before proceeding, it's essential to confirm that Google's SMTP server aligns with our project's requirements and constraints. It is free, stable and scalable.

Step 2: SMTP Authentication Setup

Enable Less Secure Apps:

Google's SMTP server requires App passwords to be enabled in our Google account. This allows applications to access our account using our username and password.

Steps to enable it:

1. Go to our Google Account settings: “Manage your Google Account”.
2. Navigate to the "Security" section.
3. Navigate to the "2-Step Verification" section.
4. If "2-Step Verification" is disabled we have to enable it.
5. Navigate to the " App passwords" section at the bottom of the page.
6. Give an app name and it will provide a key.

Gmail Email Address:

Use our Gmail email address associated with the account to send emails from.

Step 3: Configuration in Application

Update the backend code to integrate Google's SMTP server for sending emails.

The steps:

Update Backend Code:

Open the backend code where the email sending functionality will be implemented (e.g., EmailController).

Create variables FromEmail and Key within the EmailController.

Set its value to :

```
        public string FromEmail = "priyanshukoley0@gmail.com";
        public string Key = "oinq humj srsb kqry";
```

Here `"priyanshukoley0@gmail.com"` is my personal email ID and

`"oinq humj srsb kqry"` is my key. Here we have to use the email Id and its key from which we want to send mail.

Step 4: Test Configuration

Run Application Locally:

Run our backend application locally or deploy it to a test environment.

Send Test Emails:

Use a tool like Swagger UI to send test requests to the RegistrationSuccessful and PasswordReset endpoints, providing sample data for the email models.

Verify that the emails are being sent successfully and received in the designated recipient email addresses.

Check Logs and Delivery Status:

Monitor our application logs for any errors related to email sending.

Check the recipient email accounts for the test emails and ensure they are delivered without any issues.

By following these steps, we can successfully configure our backend application to utilize Google's SMTP server for sending emails.

Configuring the API's endpoints:

The provided backend code consists of an EmailController responsible for sending registration successful and password reset confirmation emails. Below is the documentation detailing its usage and functionality.

EmailController:

Namespace: API.Controllers

Base Route: /api/Email

Models:

RegistrationSuccessfulEmailModel:

Properties:

Name: Represents the recipient's name.

ToEmail: Represents the recipient's email address.

PasswordResetEmailModel:

Properties:

Name: Represents the recipient's name.

ToEmail: Represents the recipient's email address.

NewPassword: Represents the newly reset password.

Endpoints:

POST /api/Email/RegistrationSuccessful:

Description: Sends a registration successful email to the specified recipient.

Request Body: RegistrationSuccessfulEmailModel

Name: Recipient's name.

ToEmail: Recipient's email address.

Response: 200 OK with a message `"Registration Confirmation Email Sent"`

POST /api/Email/PasswordReset:

Description: Sends a password reset confirmation email to the specified recipient.

Request Body: PasswordResetEmailModel

Name: Recipient's name.

ToEmail: Recipient's email address.

NewPassword: Newly reset password.

Response: 200 OK with a message `"Password Reset Confirmation Email Sent"`

Usage:

Send a POST request to /api/Email/RegistrationSuccessful endpoint with a JSON body containing Name and ToEmail fields to send a registration successful email.

Send a POST request to /api/Email/PasswordReset endpoint with a JSON body containing Name, ToEmail, and NewPassword fields to send a password reset confirmation email.

Conclusion:

The EmailController provides endpoints for sending registration successful and password reset confirmation emails. Frontend developers can utilize these endpoints by sending appropriate HTTP requests with the required data in the request body.

Instructions on how to create and modify email templates:

This is a comprehensive instruction on how to create and modify email templates for our application's.

Step 1: Template Creation

HTML Structure:

> Design the HTML structure of our email template using standard HTML elements.
>
> Use inline CSS for styling to ensure compatibility across various email clients.

Content Layout:

> Plan the layout of our email template, including headers, body content, and footer.
>
> Consider the placement of dynamic content placeholders for personalized information.

Responsive Design:

Ensure our email template is responsive and displays correctly on different devices and email clients.

Utilize media queries and fluid layouts to adapt to varying screen sizes.

Visual Elements:

Incorporate visual elements such as images, logos, and icons to enhance the appearance of our email template.

Optimize images for web and email delivery to reduce loading times.

Step 2: Dynamic Content Integration

Placeholder Tags:

Identify sections of our email template that require dynamic content insertion.

Use placeholder tags or variables (e.g., {username}, {resetLink}) to represent dynamic content.

Backend Integration:

Implement functionality in our backend code to populate email templates with dynamic content.

Retrieve user-specific data from the database or request parameters and replace placeholder tags with actual content.

Step 3: Storage and Organization (If we have many templates)

Directory Structure:

Create a designated directory within our application's codebase to store email templates.

Organize templates by category or purpose to facilitate easy retrieval and management.

File Naming Convention:

Adopt a consistent file naming convention for email templates to maintain clarity and organization.

Include relevant identifiers or keywords in file names for easy identification.

Step 4: Modification and Customization

Template Editing:

Document the process for modifying existing email templates or creating new ones.

Provide guidelines on HTML and CSS customization to align with branding and design requirements.

Content Updates:

Regularly review and update email templates to reflect changes in content or messaging.

Keep track of version history and modifications for transparency and accountability.

Conclusion

By following these detailed instructions, we can effectively manage the creation and modification of email templates within our application. Ensure adherence to best practices for design, responsiveness, and dynamic content integration to enhance user engagement and communication. Regularly review and update email templates to maintain relevance and effectiveness in conveying information to users.

Integrating New Triggers for Email Notifications

The process for integrating new triggers for email notifications within our application. Integrating new triggers allows our application to send emails in response to specific events or actions, enhancing user engagement and communication.

Step 1: Identify Trigger Events

Event Analysis:

Identify key events or actions within our application that warrant email notifications.

Examples include user registration, password reset requests, order confirmations, and account updates.

Create the API's endpoint according to the requirements and event.

User Journey Mapping:

Map out the user journey and identify touchpoints where email notifications can enhance the user experience.

Consider both user-initiated actions and system-generated events.

Step 2: Define Email Templates

Template Creation:

Create email templates tailored to each trigger event identified in Step 1.

Design templates with personalized content and clear calls to action relevant to the trigger event.

Dynamic Content Integration:

Implement dynamic content placeholders within the email templates to populate user-specific information.

Ensure placeholders can be replaced with actual data at runtime.

Step 3: Implement Trigger Mechanisms

Backend Logic Development:

Develop backend logic to detect trigger events and initiate email notification workflows.

Utilize event-driven programming or hooks to capture trigger events effectively.

Integration with Email Service:

Integrate the backend logic with Google SMTP server.

Use APIs or libraries provided by the email service to send emails programmatically.

Step 4: Error Handling and Logging

Error Handling:

Implement robust error handling mechanisms to manage failures during email notification delivery.

Handle exceptions gracefully and provide informative error messages to aid troubleshooting.

Logging:

Log email notification events, including successful deliveries and failures, for audit and debugging purposes.

Utilize logging frameworks or services to capture relevant information such as timestamps, recipient details, and delivery status.

Step 5: Testing and Validation

Unit Testing:

Write unit tests to validate the functionality of trigger mechanisms and email notification workflows.

Test different scenarios, including successful trigger events, invalid input, and error conditions.

Integration Testing:

Perform integration tests to ensure seamless interaction between the application, trigger events, and email service.

Validate end-to-end functionality across different environments and configurations.

Step 6: Documentation and Training

Documentation:

Document the newly integrated triggers, including their purpose, implementation details, and associated email templates.

Provide clear instructions for developers on how to maintain and extend the trigger mechanisms.

Conclusion

By following the outlined process, we can effectively integrate new triggers for email notifications within our application, enhancing user engagement and communication. Ensure thorough testing, error handling, and documentation to maintain reliability and scalability of the email notification system. Regularly review and optimize trigger mechanisms to align with evolving user needs and application requirements.

Usage of the feature:

Register Form:

Usage:

Access the Register form from the application's navigation menu or designated landing page.

Enter the required information:

Name

Email

Password

Confirm Password

Ensure the following validations are met:

Password and Confirm Password fields match.

Email follows a valid format.

Name and Password fields are not empty.

On Submission:

- Click on the "Register" button to submit the form.
- Upon submission, the form will display a message indicating registration and email sending processes are in progress.
- **If registration and email sending are successful:**
    - A toast notification will appear with a success message.
    - A confirmation email will be sent to the provided email address.

Forgot Password Form:

Usage:

Access the Forgot Password form from the application's navigation menu or designated landing page.

Enter the required information:

Email

Ensure the following validation is met:

Email follows a valid format.

On Submission:

- Click on the "Submit" button to submit the form.
- Upon submission, the form will display a message indicating the email sending process is in progress.
- **If email sending is successful:**
    - A toast notification will appear with a success message.
    - A confirmation email will be sent to the provided email address.

Login Form:

Usage:

Access the Login form from the application's navigation menu or designated landing page.

Enter the required information:

Email

Password

Click on the "Login" button to submit the form.

Upon submission:

A toast notification will appear with a message indicating that the Login feature has not been implemented.

Use the buttons provided to navigate to the Forgot Password form or the Register form if needed.

Backend Integration:

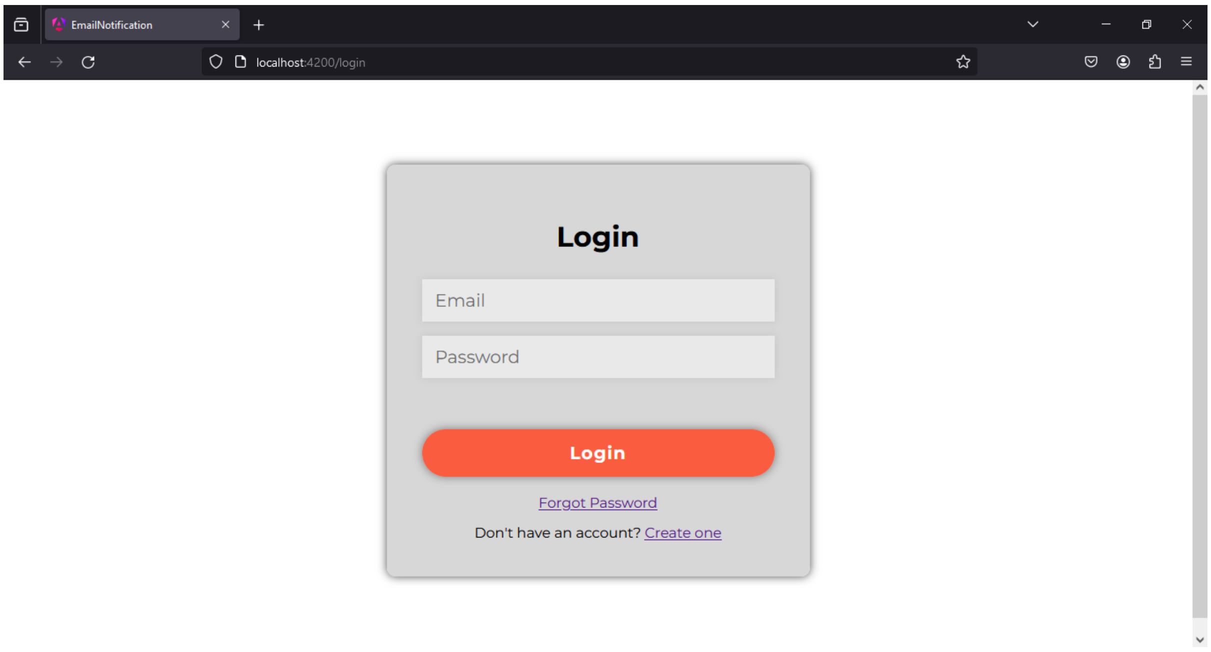
Ensure that the frontend forms are integrated with the provided API endpoints for user registration, and password reset.

Utilize the API endpoints provided above to interact with the backend functionalities seamlessly.

By following these usage instructions, users can effectively register, login, and reset their passwords using the integrated frontend forms, with all necessary interactions handled by the backend API.

Screenshots:

Login Form:



Login

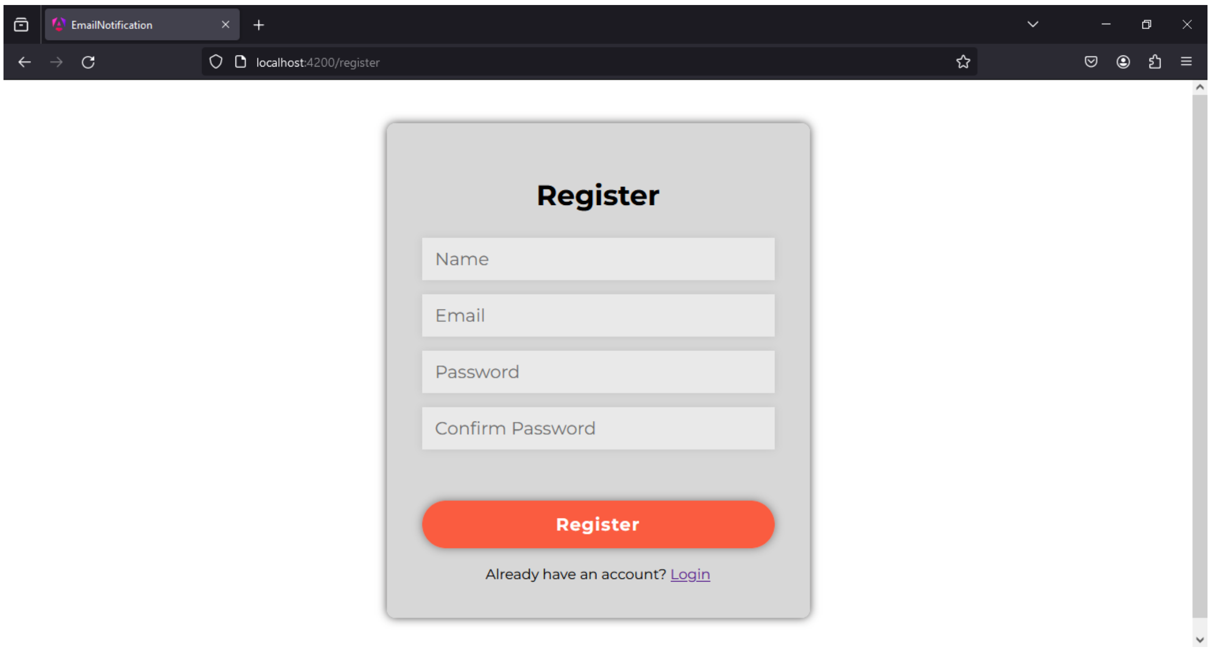
Email

Password

Forgot Password

Don't have an account? Create one

Register Form:



Register

Name

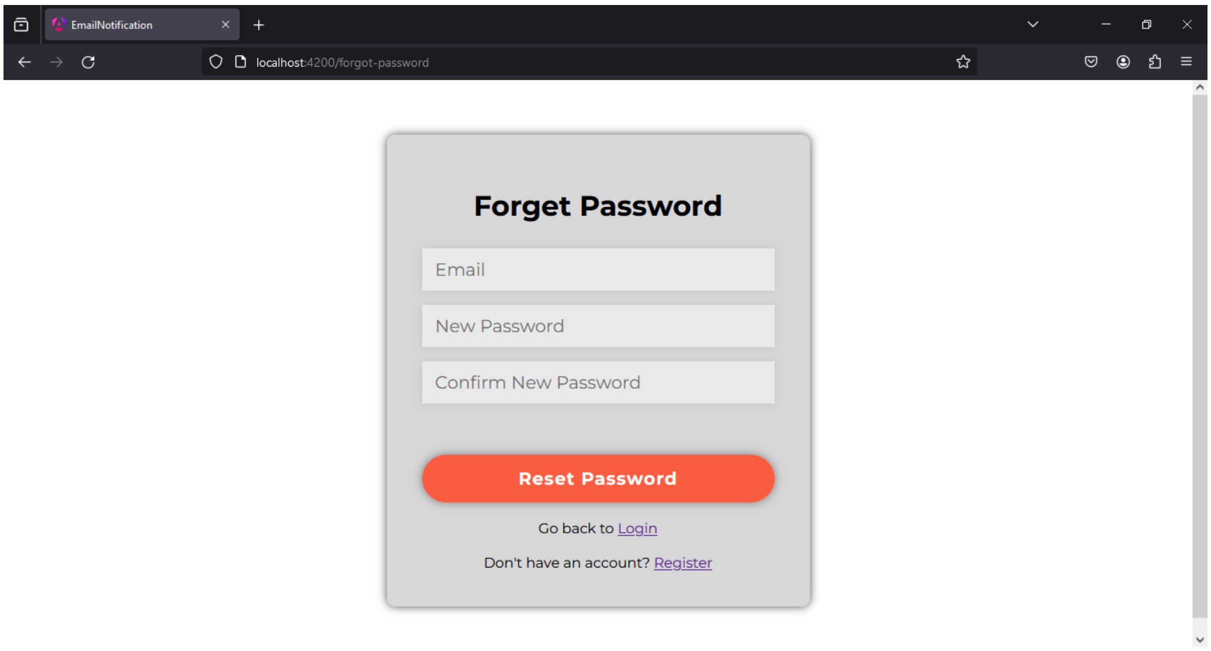
Email

Password

Confirm Password

Already have an account? Login

Forgot Password:



Conclusion

In summary, the documentation provides a detailed guide on setting up the Email Notification Feature POC. It includes configuring Google's SMTP server, defining API endpoints, managing email templates, and integrating new triggers for notifications. Additionally, it offers clear usage instructions for the frontend forms, facilitating user registration, login, and password reset processes. Overall, this comprehensive documentation ensures effective communication and user engagement within the application.