Take Home Exercise - Transactional Email Service

Overview:

Rupa Health uses internal and third party API's extensively throughout our platform and applications to process and store data, as well as manage communication to end users. The goal of this exercise is to build and consume an internal API and display the data according to the specifications outlined below. This exercise is meant to assess full stack coding ability, code quality, use of object oriented principles and your personal coding standards.

Prompt:

In order to prevent downtime during an email service provider outage, you're tasked with creating a service that provides an abstraction between at least two different email service providers. This way, if one of the services goes down, you can quickly fail over to a different provider without affecting your customers.

Please allocate 2 - 4 hours for this exercise.

Specifications:

Please create an HTTP service that accepts POST requests with json to an `/email` endpoint. The endpoint should accept the following parameters.

- `to` - the email address to send to
- `to_name` the name to accompany the email
- `from` the email address in the from and reply fields
- `from_name` the name to accompany the from/reply emails
- `subject` The subject line of the email
- `body` the HTML body of the email

Example Request Payload:

```
{
  "to": "fake@example.com",
  "to_name": "Mr. Fake",
  "from": "no-reply@fake.com",
  "from_name":"Ms. Fake",
  "subject": "A message from The Fake Family",
  "body": "<h1>Your Bill</h1><p>$10</p>"
}
```

Your service should then do a bit of data processing on the request:

- Do the appropriate validations on the input fields. (Note: All fields are required)
- Convert the `body` HTML to a plain text version to send along to the email provider. You can simply remove the HTML tags. Or if you'd like, you can do something smarter.

Once the data has been processed and meets the validation requirements, it should send the email by making an HTTP request (don't use SMTP) to one of the two services:

- Mailgun
  - Main Website: www.mailgun.com
  - Simple Send Documentation: https://documentation.mailgun.com/en/latest/quickstart-sending.html#how-to-start-sending-email
- Sendgrid
  - Main website: www.sendgrid.com
  - Getting Started Documentation: https://sendgrid.com/docs/for-developers/sending-email/api-getting-started/

Both services are free to try and are painless to sign up for, so please register your own test accounts for each.

Your service should send emails using one of the two options by default, but a simple configuration change and/or redeploy of the service should switch it to the other provider.

Implementation Requirements

Please DO NOT use the client libraries that Sendgrid or Mailgun provide. Your service will be making simple POST requests, so please build these requests with lower level package or your language's built in commands.

This is a simple exercise, but please organize, design, document and test your code as if it were going into production.

Please include a READMe file in your repository with the following information.

- How to install your app
- Which language, framework, and libraries you chose and why
- Tradeoffs you may have made, anything you left out, or what you might do differently if you were to spend additional time on the project.
- How much time you spent on the exercise.
- Anything else you wish to include.

Provide us the link to your final product as a cloneable repo on github, gitlab or bitbucket.

Bonus Points

You'll receive bonus points for well written documentation and tests as well as anything else you do that goes above and beyond the implementation requirements.

How We Review

- Functionality - Does the app do what we asked?
- Code Quality - Did you stick to OO principles? Is the code easy to understand and maintainable? Is it well tested?
- Technical choices: Do your choices of libraries, architecture, etc seem appropriate for the chosen app?