Porch+

Porchplus Backend Developer Role Technical Assessment

Thank you for your interest in the Backend Developer role at Porchplus! This take-home test is designed to assess your skills and problem-solving abilities in a backend development context

Scenario:

Fitness+ offers gym memberships with various billing structures. Some memberships are annual with a single upfront payment, while others have monthly dues. Additionally, members can subscribe to optional add-on services (personal training sessions, towel rentals) with separate monthly charges. Here's the twist:

- For new members, the first invoice combines the annual membership fee with the first month's charges for any add-on services they choose.
- In subsequent months, only the monthly charges for the add-on services are billed.

Task:

Develop a backend system that implements the following functionalities based on your expertise in these frameworks to build a robust and scalable backend API. Consider Nest.js, Postgresql, well-documented RESTful APIs and you can use your Gmail SMTP server to send out the mails.

1. **Data Model:**
    - Design a data model to store information about gym memberships, including:
        - Membership ID (unique identifier)
        - First Name
        - Last Name
        - Membership type (e.g., Annual Basic, Monthly Premium)
        - Start date
        - Due date (for annual memberships) or monthly due date (for add-on services)
        - Total amount (for annual memberships) or monthly amount (for add-on services)
        - Member email address

        - IsFirstMonth (boolean flag indicating if it's the first month of the membership)
        - (Optional) Link to the detailed membership invoice

2. **Cron Job Implementation:**

    - Develop a cron job that will run periodically (e.g., daily) to check for upcoming membership fees.
    - The cron job should:
        - Query the database for memberships with upcoming due dates.
        - Differentiate between annual memberships and add-on services, considering the IsFirstMonth flag.
        - For new members (first month):
            - Calculate the reminder date (e.g., 7 days before the due date) based on the annual membership due date.
            - Send an email reminder with the membership type, total amount for the combined annual fee and first month's add-on service charges, and a link to the full invoice detailing both.
        - For existing members (subsequent months):
            - Check if the current date falls within the month for which the add-on service applies.
            - If yes, send an email reminder with the service name, monthly amount, and a link to the invoice for that specific month's add-on service charge.

3. **Email Functionality:**

    - Implement a mechanism to send email reminders using an email service provider (ESP) or a local mail server (your choice).
    - The email should contain:
        - Subject: Fitness+ Membership Reminder - [Membership Type]
        - Body: A message reminding the member about the upcoming payment, including the membership details (type, due date for annual or month for add-on services), and a link to the relevant invoice. Clearly differentiate between the first month's combined invoice and subsequent month's add-on service invoices in the message.

Porch+

Evaluation Criteria:

Your code will be evaluated based on the following criteria:

- **Correctness:** Does the system correctly identify upcoming due dates and send reminder emails considering the first-month combined invoice and subsequent monthly add-on service charges?
- **Code Quality:** Is the code well-structured, documented, and easy to understand?
- **Error Handling:** Does the system handle potential errors gracefully, such as issues with the database connection or email delivery?
- **Scalability:** Can the system be easily scaled to handle a large number of memberships and members?
- **Testing:** Have you included unit tests to ensure the functionality of your code?

Additional Notes:

- You are free to use any libraries or frameworks that you are comfortable with.
- Feel free to make any assumptions you need about the data format or external integrations (e.g., database schema, email service provider API). Just document your assumptions clearly.
- You can focus on the core functionalities and leave out aspects like user login or invoice management for this assessment.

Submission Guidelines:

- Given the competing demands on your time, we understand that this assessment might require more flexibility. While we initially estimated 4-6 hours for completion, we are happy to accept submissions by **10:00 PM PM on Saturday, June 15th**.
- Please submit your code along with a README file explaining your design choices, implementation details, and any assumptions you made. You can submit your code via a link to a public repository (GitHub).

DATASET: An **incomplete** dataset of 40 members will be shared alongside this assessment.