Backend Exercise: Building an Integration System for AI Messaging

Objective:

We are excited to invite you to participate in this exercise as part of the Senior Software Engineer role application. This task is designed to simulate a real-world scenario relevant to our work. You will be responsible for developing a core component of our system that processes job application events and initiates conversations with candidates, which in turn trigger key downstream processes that help automate our recruitment efforts.

Contextual Overview:

This exercise is part of a larger system designed to streamline our recruitment process. When a job application event is received, your system will create a conversation record in the database with the status `CREATED`. This action will trigger a downstream process that listens for new conversations, generates an initial message to the candidate, and updates the conversation status to `ONGOING`. Once the conversation is complete, the status should be updated to `COMPLETED`.

Although you do not need to implement the downstream processes, understanding them will help you design a robust and compatible component.

For additional guidance, consider reviewing these resources:

- RESTful API Design: Best Practices
- Introduction to GraphQL
- Testing with Jest in TypeScript

Your Task:

1. **Webhook Handler Implementation:**
    - **Objective:** Implement a webhook handler that listens for incoming job application events. Upon receiving an event, your system should create a new conversation record in the database with an initial status of `CREATED`.
    - **Key Business Logic:**

- **Multiple Statuses:** Conversations can have statuses such as `CREATED`, `ONGOING`, and `COMPLETED`.
- **Active Conversation Check:** Ensure that a candidate does not have any active conversations (`CREATED` or `ONGOING`) before creating a new one.
- **Duplicate Application Check:** Prevent the creation of a new conversation if the candidate has already applied for the same job, even if the conversation has been marked as `COMPLETED`.
- **Phone Number Validation:** Validate the phone number format as this is crucial for contacting candidates via SMS or WhatsApp. Ensure it has a valid country code, appropriate length, and no invalid characters.

- **Webhook Payload Example:**

```json
{
  "id": "application-id",
  "job_id": "associated-job-id",
  "candidate_id": "candidate-id",
  "candidate": {
    "phone_number": "+1234567890",
    "first_name": "Jane",
    "last_name": "Doe",
    "email_address": "jane.doe@example.com"
  }
}
```

- **Guidance:** Use a RESTful POST endpoint for the webhook handler and design a database schema that enforces these rules through constraints and validations.

2. **API Development:**
   - **Objective:** Develop an API that allows other internal systems to interact with conversation data.
   - **API Requirements:**
     - Fetch all conversations.
     - Retrieve a single conversation by its ID.
     - Filter conversations based on their status (`CREATED`, `ONGOING`, `COMPLETED`).
   - **Implementation Option:** You may choose to implement the API using REST or GraphQL, depending on your preference and familiarity. REST may be simpler to implement if you're unsure.
   - **Document Model Example:**

```json
```

```
{
  "id": "conversation-id",
  "candidate_id": "candidate-id",
  "job_id": "associated-job-id",
  "status": "CREATED",
  "created_at": "2024-08-28T12:00:00Z",
  "updated_at": "2024-08-28T12:00:00Z"
}
```

3. **Security Measures (Optional but Encouraged):**
    - **Objective:** Implement security measures to protect access to the API and webhook.
    - **Suggestions:** This could include token-based authentication or other methods you find suitable. Aim for a solution that balances security with ease of implementation and testing.

4. **Testing (Optional but Encouraged):**
    - **Objective:** Write unit tests for key functionalities using a testing framework like Jest or Mocha with TypeScript support.
    - **Focus Areas:** Test the webhook handler, API endpoints, and key business logic, particularly around data validation and error handling.

Submission Guidelines:

1. **GitHub Repository:**
    - Submit your solution in a GitHub repository and grant access to the reviewers (ilyesBen and abshirahmed).
    - Consider committing your work incrementally and pushing updates regularly. This allows us to provide early feedback if necessary.

2. **Documentation:**
    - Include clear instructions on how to run the application and initialise the database.
    - Document any optional features, such as security measures or tests, with instructions on how to run them.

3. **Communication:**
    - If you have any questions or need clarification, feel free to reach out. We want to ensure

you have the information you need to succeed.

Completion Checklist:

- [ ] Webhook handler implemented and tested.
- [ ] API endpoints for conversations implemented and tested.
- [ ] (Optional) Security measures implemented.
- [ ] (Optional) Unit tests written and passing.
- [ ] Documentation updated with setup instructions.
- [ ] Final code pushed to GitHub repository.