1. Problem Statement

International logistics companies require a robust Configuration Management System to manage the onboarding requirements for organizations from different countries. Each country has its own unique set of requirements for business registration and compliance. For instance, onboarding a business in India requires details like Business Name, PAN, and GSTIN numbers, whereas in the USA, it might require different registration numbers and details. We need an API that can handle these varying requirements through CRUD (Create, Read, Update, Delete) operations efficiently.

2. Overview

To address the problem, we developed a scalable FastAPI application that provides endpoints to manage the configuration requirements for onboarding businesses in different countries. The application uses PostgreSQL as the database, SQLAlchemy as the ORM, and Pydantic for data validation. We implemented CRUD operations to create, read, update, and delete configurations, where each configuration stores the specific requirements of a country in JSON format. Additionally, we handled various edge cases and errors, such as database connectivity issues and missing configurations, to ensure the robustness of the application.

3. Database Schema

Table: `configurations`

- **Columns**:
  - `id` (integer, primary key): Unique identifier for each configuration entry.
  - `country_code` (string): The country code for which the configuration is applicable. It ensures that each configuration is linked to a specific country.
  - `business_name` (string): The name of the business. This allows for multiple businesses under the same country code, reflecting real-world scenarios where multiple businesses need to be onboarded within the same country.
  - `requirements` (JSON): Stores the country-specific requirements in JSON format. This allows for flexible and dynamic storage of varying requirements across different countries.

  The schema design accommodates the need to handle different configurations for each country by leveraging the flexibility of JSON to store diverse sets of requirements. Storing requirements in JSON format allows the application to adapt to the varying fields required by different countries without the need for altering the database schema.

- Having a separate `business_name` column ensures that each configuration can be uniquely identified by both the country code and the business name. This is essential because multiple businesses may exist within different countries, and each might have different requirements. The `business_name` column helps in querying, updating, and deleting configurations specific to a business within a country, thus providing more granular control over the configurations.

4. Getting Started

Installation

1. Clone the repository:

```
git clone https://github.com/Anubhav670/configuration_management_system
```

```
cd your_repository
```

2. Install dependencies:

   - Python
   - PostgreSQL installed and running
   - FastAPI, SQLAlchemy, and Pydantic

```
pip install -r requirements.txt
```

Configuration

1. Update `.env` file in the root directory with the following environment variables:

```
DATABASE_URL=postgresql://username:password@localhost/dbname
```

Important- Replace `username`, `password`, and `dbname` with your PostgreSQL credentials.

Running the Application

1. Start the FastAPI server:

```
uvicorn app.main:app --reload
```

2. Open a browser and go to `http://localhost:8000/docs` to view the Swagger UI for

   API documentation and interact with the endpoints.

5. API Endpoints

5.1Create Configuration

- **Endpoint Description**: Create a new configuration for a specific country.
- **URL**: `/create_configuration`
- **HTTP Method**: POST

Parameters

- **Request Body**: JSON object with the following fields:

```
{
        "country_code": "IN",
        "business_name": "Car Business Pvt. Ltd.",
        "requirements": {
        "PAN": "ABCDE1234F",
        "GSTIN": "29ABCDE1234F1Z5"
        }
}
```

`country_code` (string):	Country code of the organization.
`business_name` (string):	Name of the business.
`requirements` (object):	Specific requirements (e.g., PAN, GSTIN) for the country.

Output:

- **Response Body** (on success):

```
{
        "id": 1,
        "country_code": "IN",
        "business_name": "Car Business Pvt. Ltd.",
        "requirements": {
        "PAN": "ABCDE1234F",
```

"GSTIN": "29ABCDE1234F1Z5"

}

}

5.2. Get Configuration by Country Code and Business Name

Input:

- **HTTP Method**: GET
- **URL**: `/get_configuration/{country_code}/{business_name}`
- **Path Parameters**:
  - `country_code` (string): Country code of the organization.
  - `business_name` (string): Name of the business.

Output:

- **Response Body** (on success):

{

"id": 1,

"country_code": "IN",

"business_name": "Car Business Pvt. Ltd.",

"requirements": {

"PAN": "ABCDE1234F",

"GSTIN": "29ABCDE1234F1Z5"

}

}

5.3. Get Configurations by Country Code

Input:

- **HTTP Method**: GET
- **URL**: `/get_configurations/{country_code}`
- **Path Parameters**:
  - `country_code` (string): Country code of the organizations.

Output:

Success Response:

- **Status Code**: 200 OK
- **Example Response**:

```
[
        {
        "id": 1,
        "country_code": "IN",
        "business_name": "Car Business Pvt. Ltd.",
        "requirements": {
        "PAN": "ABCDE1234F",
        "GSTIN": "29ABCDE1234F1Z5"
        }
        },
        {
        "id": 2,
        "country_code": "IN",
        "business_name": "Another Business Pvt. Ltd.",
        "requirements": {
        "PAN": "XYZ7890G",
        "GSTIN": "12XYZABC1234X9"
        }
        }
]
```

Error Responses:

- **Status Code**: 404 Not Found
- **Example Error Message**:
    - **When the Country code not found in database**

```
{
    "detail": "No configurations found for the given country code"
}
```

- **Status Code**: 500 Internal Server Error
- **Example Error Messages**:
  - **When the configuration table does not exist**:

```
{
    "detail":  "Database error: The configuration table does not exist. Please
    ensure the table is created."
}
```

  - **When unable to connect to the database**:

```
{
    "detail":  "Database error: Unable to connect to the database. Please
    ensure the database is running."
}
```

  - **For other database-related errors**:

```
{
    "detail": "Database error: {error_message}"
}
```

  - **For unexpected errors**:

```
{
    "detail": "Database error: {error_message}"
}
```

5.4. Update Configuration by Country Code and Business Name

Input:

- **HTTP Method**: POST
- **URL**: `/update_configuration/{country_code}/{business_name}`
- **Path Parameters**:
  - `country_code` (string): Country code of the organization.
  - `business_name` (string): Name of the business.
- **Request Body**:

```
{
        "country_code": "IN",
        "business_name": "Example Business",
        "requirements": {
        "PAN": "AgtgtDE1234F",
        "GSTIN": "245AA0000A1Z5",
        "type":"product based"
        }
}
```

Output:

- **Response Body** (on success):

```
{
  "country_code": "IN",
  "business_name": "Example Business",
  "requirements": {
        "PAN": "AgtgtDE1234F",
        "GSTIN": "245AA0000A1Z5",
```

"type": "product based"

},

"id": 1

}

5.5. Delete Configuration by Country Code and Business Name

Input:

- **HTTP Method**: DELETE
- **URL**: `/delete_configuration/{country_code}/{business_name}`
- **Path Parameters**:
    - `country_code` (string): Country code of the organization.
    - `business_name` (string): Name of the business.

Output:

- **Response Body** (on success):

```
{
    "id": 1,
    "country_code": "IN",
    "business_name": "Car Business Pvt. Ltd.",
    "requirements": {
    "PAN": "UpdatedPAN123",
    "GSTIN": "UpdatedGSTIN987"
    }
}
```

Error Responses:

- **Status Code**: 404 Not Found
- **Example Error Message**:
    - **When the Country code or Business name not found in database**

{

```
    "detail": "No configurations found for the given country code or Business name"
}
```

- **Status Code**: 500 Internal Server Error
- **Example Error Messages**:
  - **When the configuration table does not exist**:

    ```
    {
        "detail":  "Database error: The configuration table does not exist. Please ensure the table is created."
    }
    ```

  - **When unable to connect to the database**:

    ```
    {
        "detail":  "Database error: Unable to connect to the database. Please ensure the database is running."
    }
    ```

  - **For other database-related errors**:

    ```
    {
        "detail": "Database error: {error_message}"
    }
    ```

  - **For unexpected errors**:

    ```
    {
        "detail": "Database error: {error_message}"
    }
    ```

5.6. Delete Configurations by Country Code

Input:

- **HTTP Method**: DELETE
- **URL**: `/delete_configurations/{country_code}`
- **Path Parameters**:
  - `country_code` (string): Country code of the organizations.

Output:

- **Response Body** (on success):
  - No specific response body on successful deletion. Typically, a success status code (200 OK) is returned.

Error Handling:

- When Country Code is not correctly entered-
  - **Response Body** (on failure):
    - **Status Code**: 404 Not Found
    - **Example Error Message**:

      ```
      {
              "detail": "No configurations found for the given country code"
      }
      ```

    - **Status Code**: 500 Internal Server Error
    - **Example Error Messages**:
      - **When the configuration table does not exist**:

      ```
      {

              "detail":  "Database error: The configuration table does not exist.
              Please   ensure the table is created."
      ```

```
}
```

- **When unable to connect to the database**:

```
{
    "detail":  "Database error: Unable to connect to the database.
    Please ensure the database is running."
}
```

- **For other database-related errors**:

```
{
    "detail": "Database error: {error_message}"
}
```

- **For unexpected errors**:

```
{
    "detail": "Database error: {error_message}"
}
```

6. Tech Stack Used

Backend Framework

- **FastAPI**: A modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.

Database

- **PostgreSQL**: An advanced, enterprise-class, and open-source relational database system.

ORM (Object Relational Mapper)

- **SQLAlchemy**: A SQL toolkit and Object Relational Mapper that provides a flexible and easy-to-use interface to interact with the database using Python code.

Data Validation

- **Pydantic**: Used for data validation and settings management using Python type annotations. Ensures the correctness of request and response data.

Server

- **Uvicorn**: A lightning-fast ASGI server implementation, used to serve the FastAPI application.

Development Tools

- **Git**: Version control system used to track changes and manage project source code.
- **PgAdmin**: PostgreSQL administration and development platform for managing the PostgreSQL database.

7. Conclusion

The developed Configuration Management System provides a robust and scalable solution for managing the onboarding requirements of businesses across different countries. By leveraging FastAPI, PostgreSQL, and SQLAlchemy, the system ensures flexibility, performance, and ease of data manipulation. The use of Pydantic for data validation guarantees data integrity and type safety. Comprehensive error handling and clear API documentation further enhance the usability and reliability of the system. This solution is well-suited for international logistics operations, accommodating diverse business requirements across multiple countries.