Configuration Management System with FastAPI

Objective

To develop a robust and scalable FastAPI application for managing onboarding configurations for organizations from different countries. The application will support CRUD operations (Create, Read, Update, Delete) to handle country-specific onboarding requirements.

Project Setup

Created a structured project layout to organize the code effectively.

Directory Structure

```
config_management/
├── app/
│   ├── __init__.py
│   ├── main.py
│   ├── models.py
│   ├── schemas.py
│   ├── crud.py
│   ├── database.py
│   ├── api/
│   │   ├── __init__.py
│   │   ├── endpoints/
│   │   │   ├── __init__.py
│   │   │   ├── configurations.py
├── .env
├── requirements.txt
└── alembic/
    ├── env.py
    ├── script.py.mako
    └── versions/
```

Environment Configuration

- **Purpose**: To store sensitive information such as the database connection URL securely.
- List of all required libraries installed.
  - `fastapi`
  - `uvicorn`
  - `sqlalchemy`
  - `databases`
  - `pydantic`
  - `asyncpg`
  - `python-dotenv`

- `alembic`

Database Setup

- **Purpose**: To set up a connection to the PostgreSQL database.
- **Implementation**:
  Configure the database connection in `database.py`

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from databases import Database
import os
from dotenv import load_dotenv

load_dotenv()

DATABASE_URL = os.getenv('DATABASE_URL')

database = Database(DATABASE_URL)
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)

Base = declarative_base()
```

Models

- **Purpose**: To define the structure of the database tables.
- **Implementation**:
  Created SQLAlchemy models in `models.py`

```
    from sqlalchemy import Column, Integer, String, JSON
from .database import Base

class Configuration(Base):
    __tablename__ = "configurations"

    id = Column(Integer, primary_key=True, index=True)
    country_code = Column(String, unique=True, index=True)
    requirements = Column(JSON, nullable=False)
```

Pydantic Schemas

- **Purpose**: To validate request and response data structures.
- **Implementation**:
  Created Pydantic models in `schemas.py`

```
    from pydantic import BaseModel
from typing import Dict, Any

class ConfigurationCreate(BaseModel):
    country_code: str
    requirements: Dict[str, Any]

class ConfigurationUpdate(BaseModel):
    requirements: Dict[str, Any]

class Configuration(BaseModel):
    id: int
    country_code: str
    requirements: Dict[str, Any]

    class Config:
        orm_mode = True
```

CRUD Operations

- **Purpose**: To implement functions for Create, Read, Update, and Delete operations.
- **Implementation**:
  Created CRUD functions in `crud.py`

```
from sqlalchemy.orm import Session
from . import models, schemas

def get_configuration(db: Session, country_code: str):
    return
db.query(models.Configuration).filter(models.Configuration.country
_code == country_code).first()

def create_configuration(db: Session, config:
schemas.ConfigurationCreate):
    db_config =
models.Configuration(country_code=config.country_code,
requirements=config.requirements)
    db.add(db_config)
    db.commit()
    db.refresh(db_config)
    return db_config
```

```
def update_configuration(db: Session, country_code: str, config:
schemas.ConfigurationUpdate):
    db_config = get_configuration(db, country_code)
    if db_config:
        db_config.requirements = config.requirements
        db.commit()
        db.refresh(db_config)
        return db_config
    return None

def delete_configuration(db: Session, country_code: str):
    db_config = get_configuration(db, country_code)
    if db_config:
        db.delete(db_config)
        db.commit()
        return True
    return False
```

API Endpoints

- **Purpose**: To define RESTful API endpoints for managing configurations.
- **Implementation**:
  Created API endpoints in `api/endpoints/configurations.py`

```
from fastapi import APIRouter, HTTPException, Depends
from sqlalchemy.orm import Session
from typing import List
from ... import crud, schemas, models, database

router = APIRouter()

def get_db():
    db = database.SessionLocal()
    try:
        yield db
    finally:
        db.close()

@router.post("/create_configuration",
response_model=schemas.Configuration)
async def create_configuration(config:
schemas.ConfigurationCreate, db: Session = Depends(get_db)):
    db_config = crud.get_configuration(db, config.country_code)
    if db_config:
        raise HTTPException(status_code=400, detail="Configuration
already exists")
    return crud.create_configuration(db=db, config=config)

@router.get("/get_configuration/{country_code}",
response_model=schemas.Configuration)
```

```
async def get_configuration(country_code: str, db: Session =
Depends(get_db)):
    db_config = crud.get_configuration(db, country_code)
    if db_config is None:
        raise HTTPException(status_code=404, detail="Configuration
not found")
    return db_config

@router.post("/update_configuration",
response_model=schemas.Configuration)
async def update_configuration(country_code: str, config:
schemas.ConfigurationUpdate, db: Session = Depends(get_db)):
    db_config = crud.update_configuration(db=db,
country_code=country_code, config=config)
    if db_config is None:
        raise HTTPException(status_code=404, detail="Configuration
not found")
    return db_config

@router.delete("/delete_configuration")
async def delete_configuration(country_code: str, db: Session =
Depends(get_db)):
    success = crud.delete_configuration(db=db,
country_code=country_code)
    if not success:
        raise HTTPException(status_code=404, detail="Configuration
not found")
    return {"detail": "Configuration deleted"}
```

Application Entry Point

- **Purpose**: To initialize and configure the FastAPI application.
- **Implementation**:
  Created the main application file in `main.py`

```
from fastapi import FastAPI
from .api.endpoints import configurations
from .database import database, engine, Base

Base.metadata.create_all(bind=engine)

app = FastAPI()

app.include_router(configurations.router)

@app.on_event("startup")
async def startup():
    await database.connect()
```

```
@app.on_event("shutdown")
async def shutdown():
    await database.disconnect()
```

Database Migrations

- **Purpose**: To manage database schema changes.
- **Implementation**:
- Configure `db_migration.py` to use the same database

```
from __future__ import with_statement
from logging.config import fileConfig
from sqlalchemy import engine_from_config
from sqlalchemy import pool
from alembic import context

import os
from dotenv import load_dotenv
load_dotenv()

config = context.config
fileConfig(config.config_file_name)

DATABASE_URL = os.getenv('DATABASE_URL')

config.set_main_option('sqlalchemy.url', DATABASE_URL)

from app.models import Base
target_metadata = Base.metadata

def run_migrations_offline():
    url = config.get_main_option("sqlalchemy.url")
    context.configure(url=url, target_metadata=target_metadata,
literal_binds=True)

    with context.begin_transaction():
        context.run_migrations()

def run_migrations_online():
    connectable = engine_from_config(
        config.get_section(config.config_ini_section),
        prefix="sqlalchemy.",
        poolclass=pool.NullPool,
    )

    with connectable.connect() as connection:
```

```
        context.configure(connection=connection,
target_metadata=target_metadata)

        with context.begin_transaction():
            context.run_migrations()

if context.is_offline_mode():
    run_migrations_offline()
else:
    run_migrations_online()
```

Comprehensive Error Handling

- **Purpose**: To provide meaningful error messages and handle exceptions gracefully.
- **Implementation**:
  - Use FastAPI's built-in exception handling mechanisms to catch and handle different types of errors (e.g., validation errors, database errors).

Example Usage

Create a Configuration

Endpoint: `POST /create_configuration`
Request Body

```
{
    "country_code": "IN",
    "requirements": {
        "BusinessName": "Example Business",
        "PAN": "ABCDE1234F",
        "GSTIN": "22AAAAA0000A1Z5"
    }
}
```

Get a Configuration

Endpoint: `GET /get_configuration/{country_code}`
Response:

```
{
    "id": 1,
    "country_code": "IN",
    "requirements": {
```

```
        "BusinessName": "Example Business",
        "PAN": "ABCDE1234F",
        "GSTIN": "22AAAAA0000A1Z5"
    }
}
```

Update a Configuration

Endpoint: `POST /update_configuration`
Request Body:

```
{
    "country_code": "IN",
    "requirements": {
        "BusinessName": "Updated Business",
        "PAN": "ABCDE1234F",
        "GSTIN": "22AAAAA0000A1Z5"
    }
}
```

Delete a Configuration

Endpoint: `DELETE /delete_configuration`
Request Body:

```
{
    "country_code": "IN"
}
```

Summary

This documentation provides a comprehensive approach to building a FastAPI application for managing country-specific onboarding configurations. It covers project setup, environment configuration, database design, API development, and example usage, ensuring the system is robust, scalable, and easy to maintain.