**AI-Powered PowerPoint Generator: Backend Project Folder Setup**

**Project Overview**

This backend project will allow users to generate PowerPoint presentations automatically using AI-driven content generation tools like OpenAI's GPT-3/4. The backend is a REST API built with **Flask** or **Django** (depending on your preference) and communicates with several services such as Celery for background tasks, OpenAI for generating content, and AWS for file storage.

**Technology Stack**

* **Backend Framework**: Flask or Django (Flask will be used in this guide)
* **Database**: PostgreSQL (SQLAlchemy ORM)
* **Task Queue**: Celery + Redis
* **File Storage**: AWS S3 (for PPT file storage)
* **Authentication**: JWT Tokens
* **AI Content Generation**: OpenAI API (GPT-3/4)
* **Containerization**: Docker
* **Testing**: Pytest (Unit and Integration Tests)
* **Environment Management**: Virtualenv

**Folder Structure Overview**

bash

Copy

/backend

├── /app *# Main application logic*

│ ├── /api *# API routes and controllers*

│ ├── /models *# Database models*

│ ├── /services *# Business logic and helper services*

│ ├── /tasks *# Celery task queue services*

│ ├── /schemas *# Input validation and serialization schemas*

│ ├── /utils *# Utility functions and constants*

│ ├── config.py *# Configuration file for environment variables and settings*

│ ├── \_\_init\_\_.py *# Application-level initialization*

│

├── /migrations *# Database migrations*

│

├── /tests *# Unit and integration tests*

│ ├── /unit\_tests *# Unit tests for individual modules*

│ ├── /integration\_tests *# Integration tests*

│ ├── /fixtures *# Mock data for testing*

│ └── pytest.ini *# Pytest configuration file*

│

├── /docker *# Docker-related files*

│ ├── Dockerfile *# Dockerfile for backend container*

│ └── docker-compose.yml *# Docker Compose file*

│

├── /scripts *# Automation scripts*

│ ├── deploy.sh *# Deployment script*

│

├── requirements.txt *# Python dependencies*

├── .gitignore *# Git ignore rules*

├── README.md *# Project documentation*

├── .env *# Environment variables*

**Folder and File Breakdown**

**1.**/app**: Core Application Logic**

This directory contains the core functionality of the backend application, such as API routes, database models, services, and tasks.

* /api: Controllers that handle API requests.
  + ppt\_controller.py: Manages PPT generation requests and user inputs.
  + auth\_controller.py: Handles user login, registration, and JWT token management.
  + status\_controller.py: Manages task status checks.
  + user\_controller.py: Handles user-specific operations.
* /models: Defines the database models.
  + user.py: User model for authentication and user-related data.
  + presentation.py: Presentation metadata model, storing information about generated PPTs.
  + task.py: Stores information related to background task status and results.
* /services: Contains business logic and services.
  + ai\_service.py: Interacts with the OpenAI API to generate AI-driven content for the presentations.
  + ppt\_service.py: Logic to generate PowerPoint presentations using the python-pptx library.
  + file\_service.py: Handles file storage and retrieval (e.g., AWS S3).
  + auth\_service.py: Handles user authentication using JWT tokens.
* /tasks: Contains background tasks managed by Celery.
  + generate\_ppt.py: A Celery task that generates the PPT in the background.
* /schemas: Defines input validation and serialization logic.
  + ppt\_schema.py: Defines the schema for the PPT generation request.
  + user\_schema.py: Defines the schema for user-related requests.
* /utils: Utility functions to support the backend.
  + jwt\_utils.py: Provides functions for generating and validating JWT tokens.
  + error\_utils.py: Custom error handling and logging utilities.
  + s3\_utils.py: Helper functions for interacting with S3.
* config.py: Configuration file for environment variables like database URLs, API keys, etc.

**2.**/migrations**: Database Migrations**

* This folder will store the database migration scripts (generated using **Flask-Migrate** or **Django migrations**). Migration scripts track changes to the database schema.

**3.**/tests**: Testing Directory**

This folder contains unit tests, integration tests, and fixtures to mock data used during tests.

* /unit\_tests: Unit tests for individual modules, e.g., testing the logic in ppt\_service.py or auth\_service.py.
* /integration\_tests: Integration tests to ensure the components of the application work together as expected. For example, testing the API endpoints or Celery tasks.
* /fixtures: Contains mock data used for testing, such as example user data or presentation metadata.
* pytest.ini: Pytest configuration file to set up test environments.

**4.**/docker**: Docker Configuration**

This folder contains the necessary Docker files to containerize the application.

* Dockerfile: The Dockerfile defines the environment to run the Flask/Django app inside a container.
* docker-compose.yml: This file defines multi-container services (e.g., backend app, Redis for Celery, PostgreSQL).

**5.**/scripts**: Automation Scripts**

* deploy.sh: A script to automate deployment tasks, such as pushing the app to a cloud service like AWS, Docker deployments, or server setup.

**6.**requirements.txt**: Python Dependencies**

List of all dependencies required to run the backend application, such as Flask, Celery, OpenAI, boto3, psycopg2, etc.

Example:

ini

Copy

Flask==2.0.2

Celery==5.2.0

redis==3.5.3

python-pptx==0.6.21

openai==0.11.0

boto3==1.18.56

psycopg2==2.9.1

SQLAlchemy==1.4.25

**7.**.gitignore**: Git Ignore Rules**

This file lists the files and directories that should be ignored by Git. Common items to ignore include:

markdown

Copy

**\_\_pycache\_\_**/

*\*.pyc*

*\**.pyo

.vscode/

venv/

.env

**8.**README.md**: Project Documentation**

A markdown file with detailed information about the project, setup instructions, how to run the app, etc.

Example:

markdown

Copy

# AI-Powered PowerPoint Generator ## Setup 1. Clone the repository 2. Create a virtual environment and activate it:

python -m venv venv .\venv\Scripts\activate

markdown

Copy

3. Install dependencies:

pip install -r requirements.txt

javascript

Copy

4. Set up your `.env` file with appropriate environment variables (e.g., OpenAI API key, AWS credentials, etc.).

5. Run the app:

python app.py

markdown

Copy

## Usage

- Endpoint to generate PPT: `POST /generate`

- Endpoint to check status: `GET /status/{task\_id}`

**9.**.env**: Environment Variables**

This file contains environment-specific variables, such as database URLs, API keys, and secret keys. It should never be committed to Git.

Example:

ini

Copy

FLASK\_APP=app.py

FLASK\_ENV=development

SECRET\_KEY=mysecretkey

JWT\_SECRET\_KEY=myjwtsecretkey

AWS\_ACCESS\_KEY\_ID=your-aws-access-key

AWS\_SECRET\_ACCESS\_KEY=your-aws-secret-key

OPENAI\_API\_KEY=your-openai-api-key

DATABASE\_URL=postgresql://username:password@localhost/dbname

**Setting Up the Project:**

1. **Create the Folder Structure**: Follow the CMD or terminal commands provided earlier to create the directories and files.
2. **Install Dependencies**:

bash

Copy

pip install -r requirements.txt

1. **Set Up the Database**: Initialize the database and apply migrations.
   * Use Flask-Migrate or Django migrations to apply your database schema.
2. **Run Celery**: Ensure Redis is running and start the Celery worker to handle background tasks.

bash

Copy

celery -A app.tasks.celery worker --loglevel=info

1. **Run the App**:

bash

Copy

python app.py

**Conclusion**

This **industry-level folder structure** ensures a clean, maintainable, and scalable backend for the **AI-Powered PowerPoint Generator**. It separates concerns by using controllers, services, and tasks, while also allowing easy testing, deployment, and integration of new features.

Feel free to adjust the architecture based on your specific needs or environment!

Let me know if you need any additional help or more details about any specific part of the project.