# Developer Manual for the AI-Powered Gmail Bill Scanner Project

This manual explains every aspect of the project—from setting up each component to deploying it on the cloud. It includes detailed instructions, links to relevant documentation, and explanations of the technologies used. Even if you've never built a web app or done cloud deployment before, this guide will help you get started.

---

## Table of Contents



---

## 1. Project Overview

The **AI-Powered Gmail Bill Scanner** is a web application that:

- Connects to a user's Gmail account using Google OAuth2.
- Scans emails for invoices or bills from various sources:
    - PDF attachments.
    - Image attachments (using OCR).
    - Inline email text.
    - URLs that point to external bill documents (PDFs or HTML pages).

- Uses Azure OpenAI to extract structured data (vendor, date, amount, etc.) from the bill text.
- Displays the bills in a modern, responsive React dashboard with:
    - Tabular views (with Paid/Unpaid tabs).
    - Interactive filters (by month, vendor, and category).
    - Charts (showing monthly trends and category breakdowns).
- Deploys using a modern Python backend (FastAPI) and supports local development via Docker Compose as well as production deployment on Azure Kubernetes Service (AKS).

---

# 2. Prerequisites

Before you begin, you'll need the following:

- **Basic Command Line Skills:** Ability to run commands in a terminal.
- **Git:** For source control (download Git).
- **Docker & Docker Compose:** For local development and building container images (Docker Desktop).
- **An Azure Account:** For provisioning resources such as Azure OpenAI, Blob Storage, and AKS (Azure Free Account).
- **A Google Cloud Account:** To set up Gmail API and OAuth credentials (Google Cloud Platform).
- **Node.js & npm:** For the frontend development (Download Node.js).
- **A Text Editor or IDE:** VSCode, PyCharm, etc.

---

# 3. Project Architecture

## Overview

The project consists of several interconnected components:

- **Backend:**
    - **FastAPI** web server to provide REST APIs.
    - **Celery** for background processing (email scanning and OCR tasks).
    - **SQLAlchemy** for managing the PostgreSQL database.
    - Integration with Gmail API to fetch emails.
    - OCR using Tesseract (via pytesseract) for image attachments.
    - PDF and HTML extraction for invoices.

    - **Azure OpenAI** (via Azure AI Foundry) for extracting structured invoice data.
    - **Azure Blob Storage** for storing attachments.
- **Frontend:**
    - **React** with **TypeScript** for the user interface.
    - **Material UI (MUI)** for a modern look and responsive design.
    - **Recharts** for visualizing data (charts).
    - Interactive components such as filters and tabs for viewing bills.
- **Database:**
    - **PostgreSQL** stores user data and extracted bill information.
- **Message Queue:**
    - **Redis** is used as the broker for Celery tasks.
- **Deployment:**
    - **Docker Compose** for local development.
    - **Kubernetes (AKS)** for production deployment.

---

# 4. Setting Up Each Component

## 4.1 Gmail API & Google OAuth2

**Steps:**

1. **Create a Google Cloud Project:**
    - Go to Google Cloud Console.
    - Create a new project.
2. **Enable the Gmail API:**
    - In the Cloud Console, navigate to **APIs & Services > Library**.
    - Search for "Gmail API" and click **Enable**.
    - Gmail API Documentation
3. **Configure the OAuth Consent Screen:**
    - Navigate to **APIs & Services > OAuth consent screen**.
    - Choose **External** (if not using a G Suite account) and fill in the required details.
    - Add scopes such as https://www.googleapis.com/auth/gmail.readonly, email, and profile.
4. **Create OAuth Client Credentials:**
    - In **APIs & Services > Credentials**, click **Create Credentials > OAuth client ID**.
    - Choose **Web application** as the type.
    - Set **Authorized redirect URIs** (for example, http://localhost:8000/auth/google/callback for local development and your production URL for deployment).

    - Save the **Client ID** and **Client Secret**.
5. **Configure Environment:**
    - Add these credentials to your environment variables or Kubernetes secrets (see section Kubernetes Deployment).
    - Example environment variables:
      `GOOGLE_CLIENT_ID`, `GOOGLE_CLIENT_SECRET`, `GOOGLE_REDIRECT_URI`.

**Helpful Links:**

- Google Cloud Console
- Gmail API Quickstart
- OAuth 2.0 for Web Server Applications

---

## 4.2 Backend Setup (FastAPI, Celery, SQLAlchemy)

**Steps:**

1. **Clone the Repository:**
    - Use Git to clone the project repository to your local machine.
2. **Python Environment:**
    - Create a virtual environment (optional but recommended):

      ```
      bash
      Copy
      python -m venv venv
      source venv/bin/activate  # On Windows: venv\Scripts\activate
      ```

    - Install the required packages:

      ```
      bash
      Copy
      pip install -r backend/requirements.txt
      ```

3. **Database Setup:**
    - For local development, you can use the PostgreSQL Docker container provided in `docker-compose.yml`.
    - Alternatively, install PostgreSQL locally.
    - Ensure your `DATABASE_URL` is correctly set (for example, `postgresql://postgres:postgres@localhost:5432/postgres`).
4. **Celery Setup:**
    - Celery is configured to use Redis as its broker. Ensure Redis is running (via Docker Compose or a local installation).
5. **Configuration:**
    - The file `backend/app/config.py` uses Pydantic to load environment variables.

    - Create a .env file in the project root (or set environment variables) with the necessary keys.
6. **Running the Backend:**
    - To run the FastAPI server locally:

      bash
      Copy
      uvicorn app.main:app --reload --host 0.0.0.0 --port 8000

    - To run the Celery worker:

      bash
      Copy
      celery -A app.celery_app:celery_app worker --loglevel=info

**Helpful Links:**

- FastAPI Documentation
- Celery Documentation
- SQLAlchemy Documentation
- Uvicorn

---

## 4.3 OCR Setup (Tesseract)

**Steps:**

1. **Install Tesseract OCR:**
    - On Ubuntu:

      bash
      Copy
      sudo apt-get update
      sudo apt-get install tesseract-ocr tesseract-ocr-heb

    - On macOS using Homebrew:

      bash
      Copy
      brew install tesseract

    - On Windows, download the installer from Tesseract at UB Mannheim.
2. **Python Wrapper:**
    - The project uses pytesseract. This is included in the requirements.txt.
3. **Verify Installation:**
    - Run tesseract --version in your terminal to verify that it's installed.

4. **Configuration in Docker:**
    - The backend Dockerfile installs Tesseract and the Hebrew language pack.

**Helpful Links:**

- Tesseract OCR GitHub
- pytesseract Documentation

---

## 4.4 Azure OpenAI (Azure AI Foundry)

**Steps:**

1. **Provision Azure OpenAI:**
    - Sign in to your Azure Portal.
    - Search for "Azure OpenAI" and create a new resource.
2. **Deploy a Model:**
    - Follow the documentation to deploy a model (e.g., GPT-35 Turbo or GPT-4).
    - Note the deployment name (used as `AZURE_OPENAI_ENGINE`).
3. **Obtain Credentials:**
    - In your Azure OpenAI resource, copy the API endpoint and API key.
4. **Configuration:**
    - Add these details to your environment variables (`AZURE_OPENAI_ENDPOINT`, `AZURE_OPENAI_KEY`, `AZURE_OPENAI_ENGINE`, `AZURE_OPENAI_API_VERSION`).

**Helpful Links:**

- Azure OpenAI Service Documentation
- Azure OpenAI Quickstart

---

## 4.5 Azure Blob Storage

**Steps:**

1. **Create a Storage Account:**
    - In the Azure Portal, create a new **Storage Account** (General-purpose v2).
2. **Create a Blob Container:**
    - Within the Storage Account, create a new container (e.g., named "attachments").

3. **Get Connection String:**
    - In the Storage Account, navigate to **Access keys** and copy the connection string.
4. **Configuration:**
    - Add the connection string and container name to your environment variables (`AZURE_BLOB_CONNECTION_STRING`, `AZURE_BLOB_CONTAINER`).

**Helpful Links:**

- Azure Storage Documentation
- Azure Blob Storage Quickstart

---

## 4.6 Database Setup (PostgreSQL)

**Steps:**

1. **Local Development:**
    - Use the PostgreSQL container provided in the `docker-compose.yml` file.
    - Alternatively, install PostgreSQL locally (Download PostgreSQL).
2. **Azure Database for PostgreSQL:**
    - In the Azure Portal, create an **Azure Database for PostgreSQL** instance.
    - Configure connection settings (server name, username, password, database name).
3. **Configuration:**
    - Update the `DATABASE_URL` environment variable accordingly.
    - Ensure your database uses UTF-8 encoding to support multilingual text.

**Helpful Links:**

- PostgreSQL Documentation
- Azure Database for PostgreSQL

---

## 4.7 Frontend Setup (React, TypeScript, Material UI, Recharts)

**Steps:**

1. **Install Node.js & npm:**
    - Download and install from Node.js Official Site.
2. **Project Setup:**

    - Navigate to the frontend/ directory.
    - Run npm install to install dependencies.
3. **Material UI:**
    - The project uses Material UI for styling. Documentation is available at Material UI.
4. **Charts with Recharts:**
    - Recharts is used for charting. See Recharts Documentation.
5. **Running the Frontend:**
    - Use npm start to run in development mode.
    - For production, run npm run build to create an optimized build.
6. **Configuration:**
    - The frontend expects an environment variable REACT_APP_API_BASE_URL (set in .env or via Docker Compose) pointing to the backend API.

**Helpful Links:**

- React Documentation
- TypeScript Handbook
- Material UI Documentation
- Recharts Documentation

---

## 4.8 Local Development with Docker Compose

**Steps:**

1. **Install Docker & Docker Compose:**
    - Follow instructions at Docker Desktop.
2. **Create a .env File:**
    - In the project root, create a .env file with all necessary environment variables (refer to the sample values provided in k8s/secret.yaml for ideas).
3. **Run Docker Compose:**
    - In the terminal, execute:

      bash
      Copy
      docker-compose up --build

    - This starts the backend (FastAPI), worker (Celery), frontend (React served via Nginx), PostgreSQL, and Redis.
4. **Access the Application:**
    - Frontend: http://localhost:3000
    - Backend API: http://localhost:8000/docs (FastAPI's interactive docs)

**Helpful Links:**

- Docker Compose Documentation
- Using .env Files in Docker Compose

---

## 4.9 Production Deployment with Kubernetes (AKS)

**Steps:**

1. **Provision an AKS Cluster:**
   - Use the Azure CLI or Azure Portal to create an AKS cluster.
   - AKS Documentation
2. **Push Docker Images:**
   - Build the backend and frontend images and push them to a container registry (e.g., Azure Container Registry).
   - Example commands:

     bash
     Copy
     docker build -t <your-registry>/gmail-bill-scanner-backend:latest ./backend
     docker build -t <your-registry>/gmail-bill-scanner-frontend:latest ./frontend
     docker push <your-registry>/gmail-bill-scanner-backend:latest
     docker push <your-registry>/gmail-bill-scanner-frontend:latest

3. **Configure Kubernetes Secrets:**
   - Create a Kubernetes Secret (see k8s/secret.yaml) with all sensitive values.
   - Use the command:

     bash
     Copy
     kubectl apply -f k8s/secret.yaml -n your-namespace

4. **Deploy Resources:**
   - Apply the provided Kubernetes manifests:

     bash
     Copy
     kubectl apply -f k8s/backend-deployment.yaml -n your-namespace
     kubectl apply -f k8s/backend-service.yaml -n your-namespace
     kubectl apply -f k8s/worker-deployment.yaml -n your-namespace
     kubectl apply -f k8s/frontend-deployment.yaml -n your-namespace
     kubectl apply -f k8s/frontend-service.yaml -n your-namespace
     kubectl apply -f k8s/redis-deployment.yaml -n your-namespace
     kubectl apply -f k8s/redis-service.yaml -n your-namespace
     kubectl apply -f k8s/ingress.yaml -n your-namespace

5. **DNS & TLS:**
    - Configure your domain to point to the Ingress IP.
    - Use cert-manager to obtain TLS certificates from Let's Encrypt.
6. **Monitoring & Logging:**
    - Use tools such as Azure Monitor or Prometheus with Grafana to track application metrics and logs.

**Helpful Links:**

- Azure Kubernetes Service Quickstart
- Helm – Package Manager for Kubernetes
- Cert-Manager Documentation

---

# 5. Testing and Troubleshooting

## Testing

- **Gmail OAuth Flow:** Log in through the frontend and verify that the backend receives a valid token and stores user details.
- **Email Scanning:** Send sample emails (with PDF attachments, images, inline text, and links) to your Gmail account and trigger a sync. Check logs to see that the email content is processed.
- **OpenAI Extraction:** Verify that both English and Hebrew invoices yield correctly formatted JSON output.
- **Frontend Functionality:** Ensure the dashboard displays the bills correctly, the filters work, and the charts reflect the aggregated data.

## Troubleshooting Tips

- Check Docker logs using `docker-compose logs <service>`.
- For Kubernetes, use `kubectl logs <pod-name> -n your-namespace`.
- Use FastAPI's interactive docs at http://localhost:8000/docs to test API endpoints.
- Verify environment variables in your containers with `kubectl exec` or by checking the container logs.
- Refer to the documentation links provided for troubleshooting specific components.

---

# 6. Maintenance and Further Enhancements

- **CI/CD:** Set up automated pipelines (using GitHub Actions, Azure Pipelines, etc.) to build, test, and deploy your application.
- **Scaling:** Monitor resource usage; adjust the number of backend replicas or Celery workers based on load.
- **Security:** Regularly update dependencies, rotate secrets, and perform vulnerability scans.
- **User Feedback:** Incorporate feedback to improve UI/UX and add features (e.g., notifications, more detailed analytics).

---

# 7. Useful Links and Resources

- **Google Cloud & Gmail API:**
  - Google Cloud Console
  - Gmail API Documentation
- **FastAPI:** https://fastapi.tiangolo.com/
- **Celery:** https://docs.celeryq.dev/
- **SQLAlchemy:** https://www.sqlalchemy.org/
- **Tesseract OCR:** https://github.com/tesseract-ocr/tesseract
- **Azure OpenAI Service:** Azure OpenAI Documentation
- **Azure Blob Storage:** Azure Storage Documentation
- **PostgreSQL:** https://www.postgresql.org/
- **React:** https://reactjs.org/
- **Material UI:** https://mui.com/
- **Recharts:** https://recharts.org/en-US/
- **Docker & Kubernetes:**
  - Docker Documentation
  - Kubernetes Documentation

---

# 8. Conclusion

This manual covers every step required to set up, configure, and deploy the AI-Powered Gmail Bill Scanner project—from obtaining necessary credentials and provisioning cloud resources to running and scaling the application. With this guide, even someone new to web development and cloud deployment should be able to successfully build, test, and maintain the system.