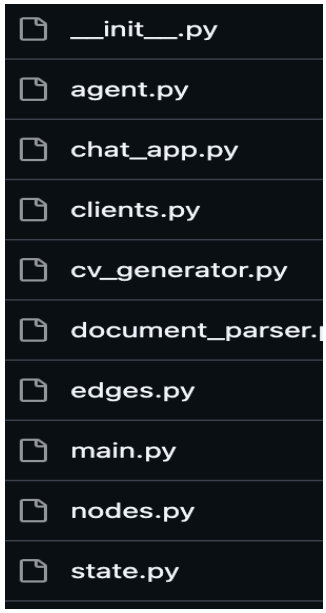
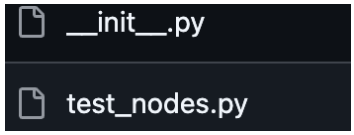
Code Structure :

- **cv_creator_app**
  - **cv_agent**
    - 

  - **tests**
    - 

  - **Dockerfile**
  - **run.sh**
  - **.env**
  - **deploy.sh**
  - **requirements.txt**

Containerized deployment in Google Cloud Run :

1. **Creating the virtual environment and activate that (local) :**
   brew install python@3.11
   Python3.11 -m venv venv
   source vnev/bin/activate
   pip install –upgrade pip
   pip install -r requirements.txt
2. **Check if the LLM API key is properly read from the .env file**
   LLM_API_KEY=<your_key>
   (no spaces and no quotes)

3. **Run the code locally**
   bash run.sh prod (what is prod)
   /run.sh prod
4. **Run with Docker locally :**
   a. Install docker in your system locally (docker desktop)
   b. docker build -t cv_agent
   c. docker run –rm –env-file .env -p 8080:8080 cv_agent
5. **Putting this docker container in Google Cloud Run to run**
   a. **Install google cloud sdk :**
      i. brew install –cask google-cloud-sdk
   b. **Google cloud authorization :**
      i. gcloud auth login
      ii. It will prompt for gmail id that you want to use
   c. **You will have to do two things :**
      i. Create a project
      ii. Create a billing account
      iii. Linking the project id with the billing account
      iv. **gcloud beta billing accounts list** (give your billing account lists)
      v. **gcloud projects list** (list of projects) - this will give project id or you can see from web
      vi. **gcloud config set project <project_id>**
      vii. **gcloud beta billing projects link** <project_id> –billing-account=<billing_account_id>
   d. **Now add google cloud services that will be required :**
      i. gcloud services enable \ run.googleapis.com \ cloudbuild.googleapis.com \ artifactregistry.googleapis.com \ containerregistry.googleapis.com
   e. **Google cloud role / permission access and run**
      i. **Create a service account** : gcloud iam service-accounts create <name> –display-name=”some display name”
      ii. This service account should be given permissions to run as admin , artifactory registry writer , iam service account user

```
PROJECT_ID="your-project-id"
SERVICE_ACCOUNT_EMAIL="your-service-account@your-project-id.iam.gserviceaccount.com"
```

iii. ***# Grant Cloud Run Admin role*** **gcloud projects**
add-iam-policy-binding $PROJECT_ID \
--member="serviceAccount:$SERVICE_ACCOUNT_EMAIL" \
--role="roles/run.admin" *# Grant Artifact Registry Writer role*
gcloud projects add-iam-policy-binding $PROJECT_ID \
--member="serviceAccount:$SERVICE_ACCOUNT_EMAIL" \
--role="roles/artifactregistry.writer" *# Grant Service Account User role* gcloud projects add-iam-policy-binding $PROJECT_ID \
--member="serviceAccount:$SERVICE_ACCOUNT_EMAIL" \
--role="roles/iam.serviceAccountUser"

f. Deployment time :

i. deploy.sh
ii. Generate a public URL

Why there is two different code snippet for the fast api production app ?
1. Reload = False normally done in development so that the server does not start automatically when the code changes. Reload = true means the server will restart automatically when there is some code changes
2. You can adjust the logging level differently in production vs dev environment
3. Development behavior vs Product behavior separation :
   - Debugging , frequent changes , logs (dev env)
   - Dev settings vs prod setting

Depending on the load, Cloud run will run 1 or more instances of my container . When no request coming it can scale to zero.
Auto Scaling limit : for a given service , there could be 1000 instances that can be created
Which means that concurrently it handle 1000 x 80 = 80000 requests simultaneously

Concurrency : default : 80 requests per container , you can configure this as well

Load balancing : we dont configure

Architecture :
main.py —> fastapi + creates the agent object + initiates the UI
chat_app.py—> event driven (click on upload) , click on analyze resume (agent will be called back and agent workflow starts …agent will wait after every step for button to press or human input to be gives