**DEM Section 1 and 2**

The following endpoints have been implemented

Graphical user interface, application

Description automatically generated

**Endpoints Section 1**

1. Post: Upload\_user : This endpoint helps you post an excel file provided and runs your ETL job and inserts it into a postgres database. Raw data is saved in table (**users**) and transformed data is saved in table(**user\_transfrom**)

Example can be seen in below two screenshots:

Graphical user interface, application

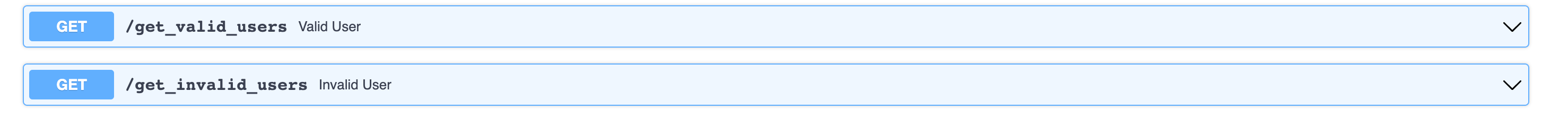
Description automatically generated

Logs of the ETL process: The ETL process validates the email and ipaddress and saves the results in the same postgres database but a different table.

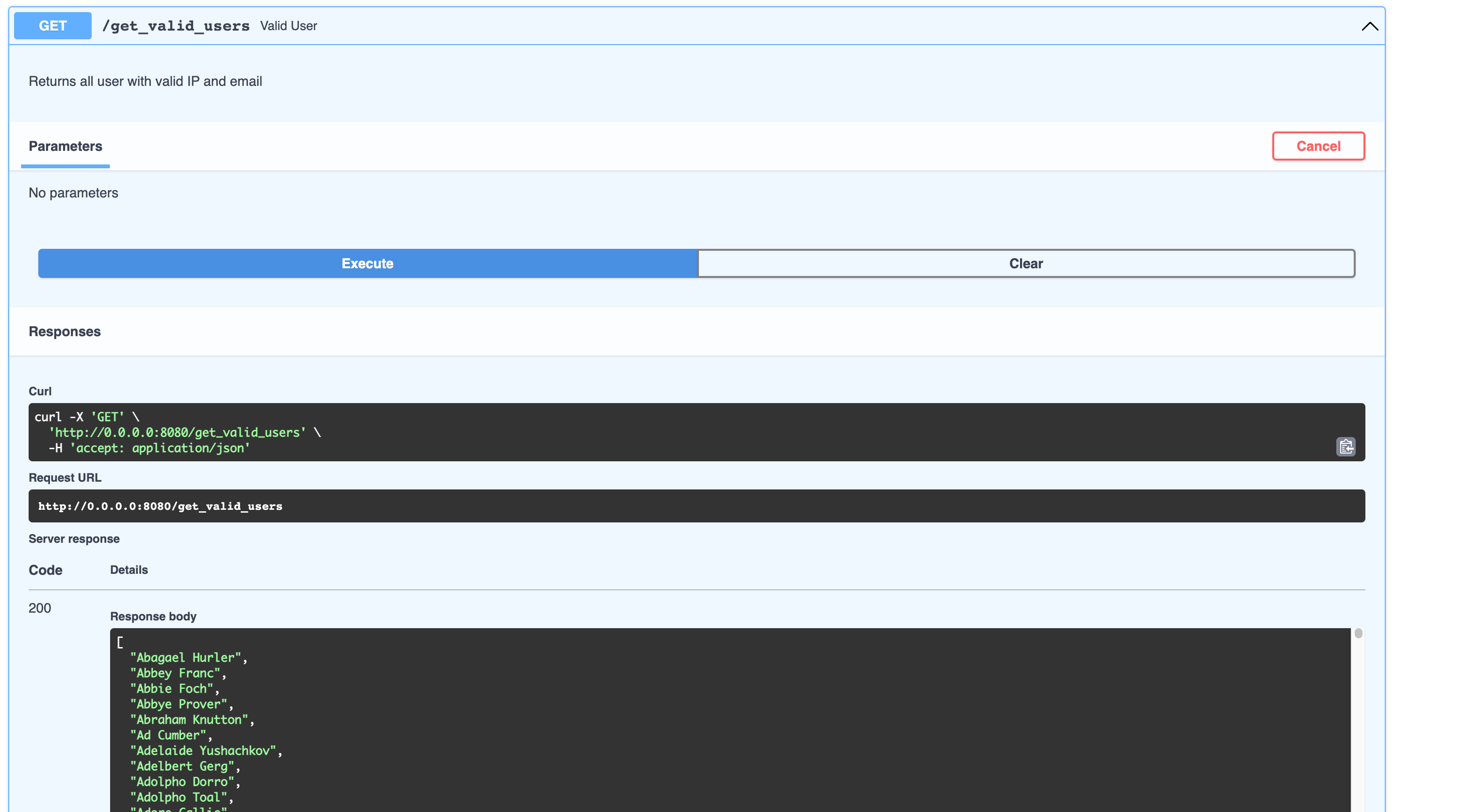
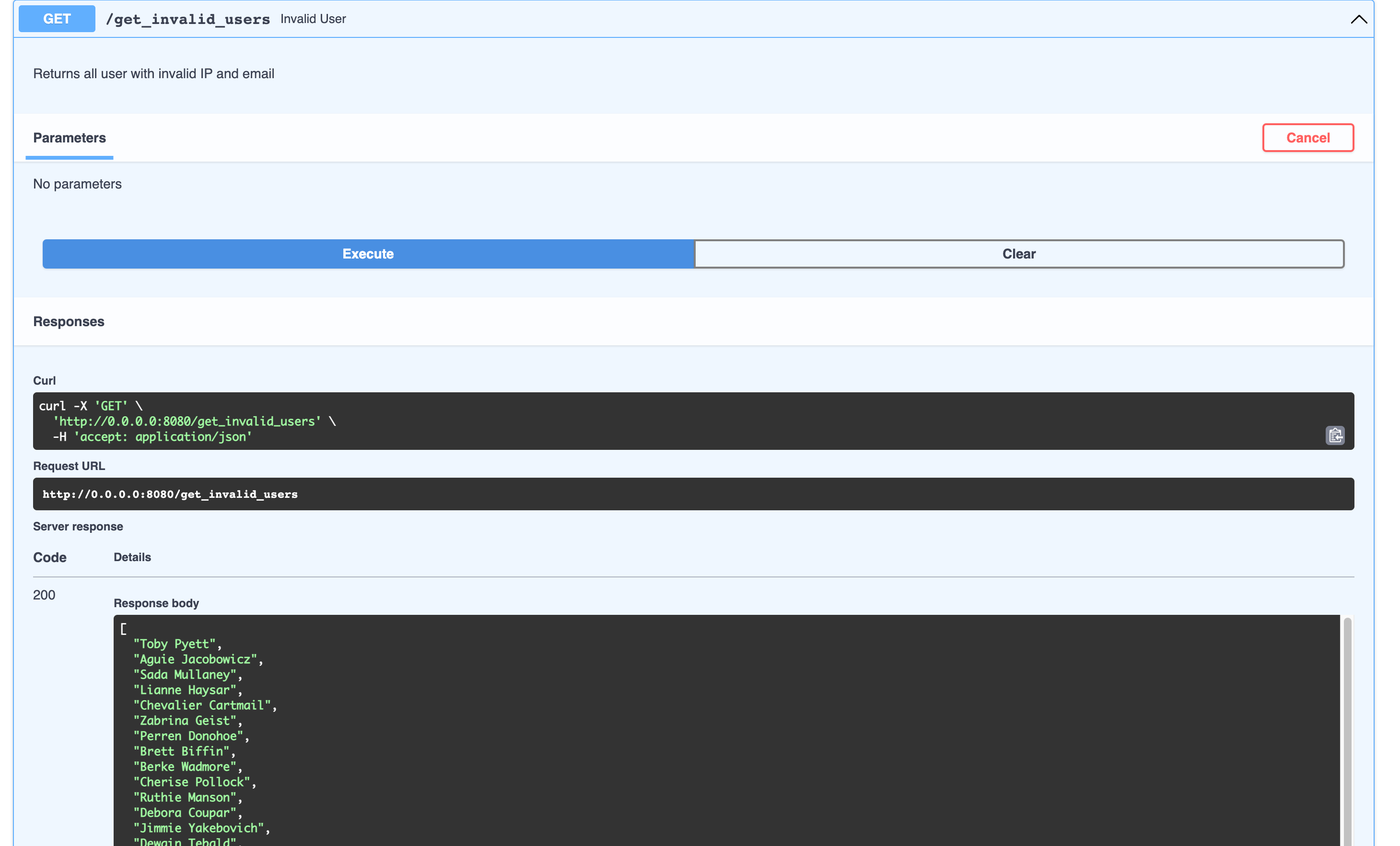
Text

Description automatically generated

1. Get : get\_valid\_users & get\_invalid\_users



1. These endpoints help with retrieving the results from transformed data in table(**user\_transfrom**)

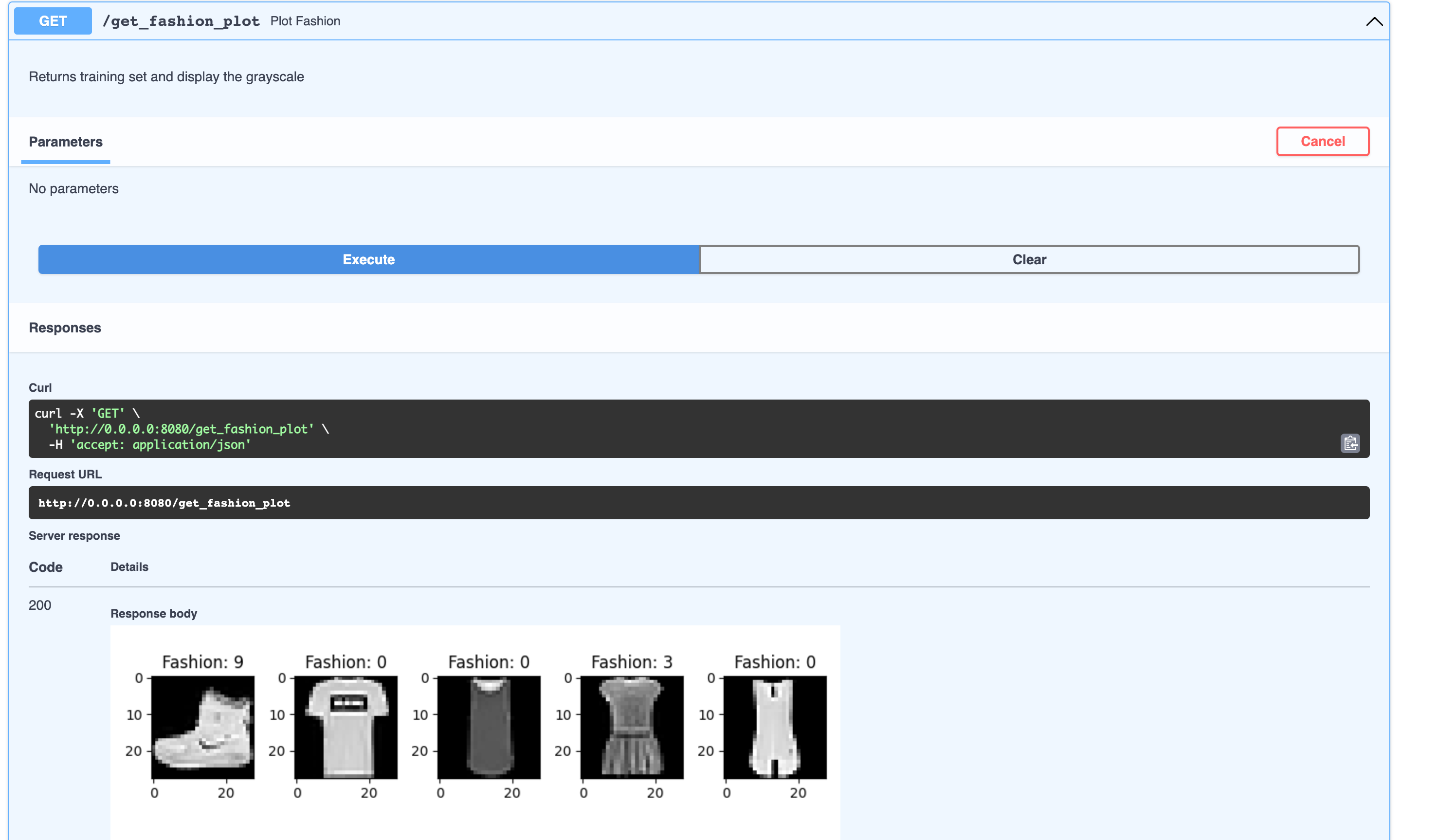
 

The above endpoints demonstrate sqlalchemy, pandas and sqlalchemy sql to extract, load and transform data, also report data through get points.

**Endpoints Section 2**

**DEM ML PLOT**

1. Endpoint (Get): get\_fashion\_plot

This endpoint helps retrieve grayscale plots using Tensorflow and matplotlib. 

**DEM Deployment**

This project contains the source code for DEM challenge.

**Prerequisites**

* Docker
* Python
* Pipenv
* Local Kubernetes environment- (Using minikube instead of Docker Desktop). Use `brew upgrade minikube` if you have an older version of minikube, as some incompatibilities may exist.
* Kubectl. Aliases are optional but highly recommended.

**Setup**

We will setup a kube cluster and deploy the application locally.

Note: If you cannot fulfill all the requirements above or if you have anyform of difficulty in deploying the application you can also run (**docker-compose build and then docker-compose up**) to start the application & skip all the steps below.

This repo supports unix like operating systems like Linux, Mac, & WSL. Native windows is not supported.

It is normal for the initial run of build-minikube.sh to take a long time. Subsequent reruns, even if minikube delete is run in between, should be much faster.

1. Clone the main/master branch.

- Run docker build command to prep the image on local

2. If you have run `minikube start` at any point beforehand run `minikube delete` before continuing to next step.

3. To use minikube run `sh build.sh`

- It is normal for pods to briefly have the \*\*CrashLoopback\*\* status during startup. If this status persists for more than 30 seconds something is likely wrong and needs troubleshooting.

- After the script completes you can run `kubectl get pods` to review if the pods are running. *\*Note\** if you do not see STATUS = Running for all pods listed something went wrong.

Screenshot from lens:

A screenshot of a computer

Description automatically generated with medium confidence

Screenshot from command line:

Text

Description automatically generated

**Note**: If you minikube registry is not configured correctly feel free to build the image separately and push the image to minikube docker registry.

Cmd1:

docker build -f Dockerfile -t web:latest\_52 .

Cmd2:

minikube cache add web:latest\_52

4.Once you have the pods running you want to grab the web svc pod name and port forward to expose the application:

kubectl port-forward web-7f694969c6-g2zt4 8080:8080

kubectl port-forward <pod\_name> 8080:8080



5. You should be able to access the application through: [http://localhost:8080/docs#](http://localhost:8080/docs)

~\*~