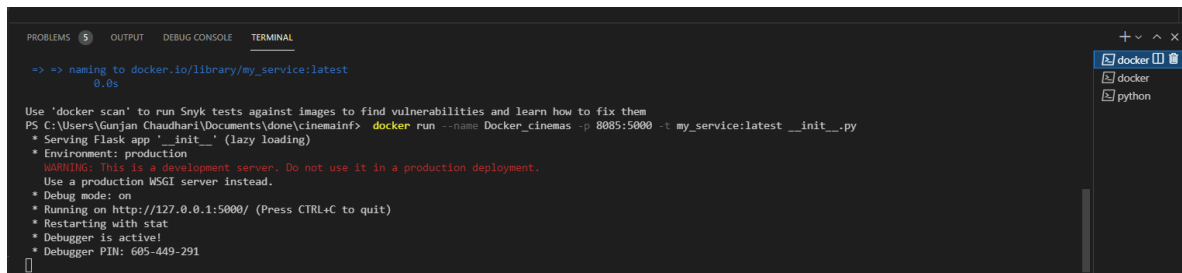


COMP9322 Assignment 2

Zid: Z5302729

Steps to run the code,

1. install: conda install flask-CORS
2. in folder Cinemainf, run command : build -t my_service:latest .
3. run command: docker run --name Docker_cinema -p 8085:5000 -t my_service:latest __init__.py

A screenshot of a terminal window with a dark background. The terminal shows the command 'docker run --name Docker_cinema -p 8085:5000 -t my_service:latest __init__.py' being executed. The output includes a warning about using a development server and a list of debug messages. On the right side of the terminal, there is a sidebar with icons for 'docker' and 'python'.

```
>> naming to docker.io/library/my_service:latest
0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them
PS C:\Users\Gunjan Chaudhari\Documents\done\cinemainf> docker run --name Docker_cinema -p 8085:5000 -t my_service:latest __init__.py
* Serving Flask app '__init__' (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 605-449-291
```

4. in folder timslot, run command : build -t my_service:latest .
5. run command: docker run --name Docker_booker -p 8080:5000 -t my_service:latest __init__.py
6. Go to bot/chatbot/demo and run command : python __init__.py

This will start the environment by running both the dockers, containing booking and cinema information services, which are called in the 'bot' folder connected to the 'wit.ai'

To connect to a chatbot navigate to the folder 'jake' and

Run: python -m http.server

Using a browser, Go to <http://localhost:8000>

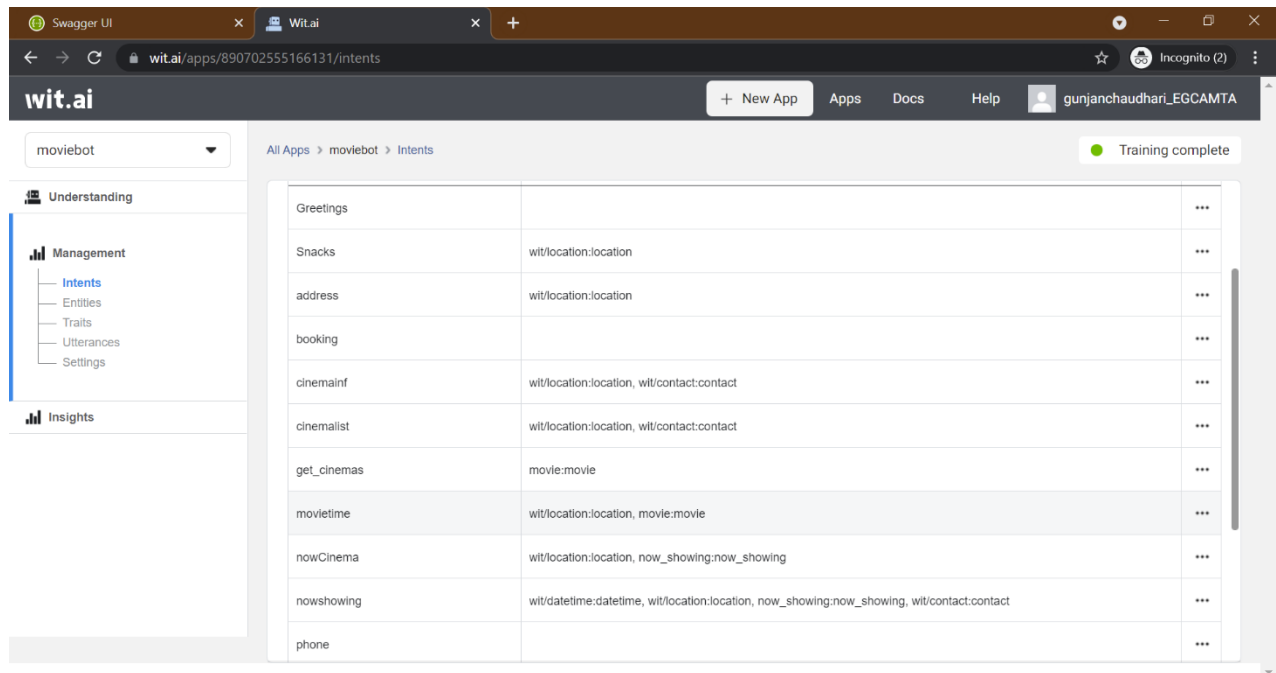
WIT usage:

Facebook Credentials:

Email address: gunjanunsw@gmail.com

Password: Gunjan1234

Intents Page:



The screenshot shows the WIT.ai interface for the 'moviebot' app. The left sidebar contains a navigation menu with 'Intents' selected. The main area displays a table of intents with their entities and training status.

Intent	Entities	Training Status
Greetings		...
Snacks	wit/location:location	...
address	wit/location:location	...
booking		...
cinemainf	wit/location:location, wit/contact:contact	...
cinemalist	wit/location:location, wit/contact:contact	...
get_cinemas	movie:movie	...
movietime	wit/location:location, movie:movie	...
nowCinema	wit/location:location, now_showing:now_showing	...
nowshowing	wit/datetime:datetime, wit/location:location, now_showing:now_showing, wit/contact:contact	...
phone		...

Chatbot Working:

Flow:

The chatbot ideally starts by 'Greetings' after which it asks the user to enter their name. This name is stored in a sample.txt file for remembering the username.

The user can then ask multiple questions related to cinema, movies, and the timeslots. The respective intents are created on WIT. Depending on which intent is called the chatbot searches for the respective 'if' statement.

e.g., questions,

Which can I watch a movie?

What is address of Ritz?

When can I watch No Time To Die at HOYTS Broadway?

I want to book tickets for Halloween Kills

Conversation:

The sample.txt stores conversation dictionary which contains initially:

```
conversations = { "name": 0, "tickets": 0, "timeslot": 0, "ttype": 0, "booking_id": 0, "intent": 0 }
```

As the conversation goes ahead each of the fields is filled when everything is filled the chatbot asks user if they want to confirm the booking. If the user says 'Yes' the booking is registered in the database.

CinemaInf:

The cinemaInf contains code, which returns cinema and movie related queries, to the swagger.

Timeslot:

The timeslot contains code, which returns timeslot and booking related queries and insertion.

Bot:

The bot contains the main conversational logic of the chatbot which imports data from the 2 dockers

Jake:

Jake contains the frontend chatbot files.

Documents:

Contains yaml files, and the installation instruction