

Name: Sonal Bedi

Employee Id: 22704

Program: Software Developer (Framework Engineer) Python Training

Python Training

Project-6

Title: Create Movie Booking System using FastApi Concept.

In this project, we will discuss how to develop a **movie booking system** and how to store data in database. As in this we are using Mysql database and also all the operations i.e. to add

,update,display, and delete (CRUD) data from database Mysql using restapi.

Most of the functionalities we will be developing in this project :

Table 1: Users

user_id(pk), name, email

Table 2: Movie

Movie_id(pk), movie_name

, movie_description, rating, type

Table 3: Shows

Show_id(pk), movie_id(fk), timeslot_id, the

atre_id, seats_booked, seat_available

Table 4: Theatre

theatre_id(pk), theatre_

name

Table 5: Seats

Seat_id(pk), seat_type, s

eat_price

Table 6: Ticket_Booked

ticket_id(pk), show_id(fk), seat_id(fk), theatre_id(fk)

user_id(fk),timeslot_id(fk),movie_id

- Get:Display all records
- Get/id: Display details of record by id
- Post/:Add new record
- Put/id: Update details of record given id
- Delete/id: Delete given record from database with given id

PURPOSE :

Some important goals of movie ticket booking are as follows:

- Online movie ticket booking system project is aimed to provide facility to book movie tickets online anytime and from anywhere without going to Theatres.
- It involves less number of staffs at the ticket-boxwindow.
- Promote new movies over the internet and gain maximum profit.
- Provide a 24x7 service to the customer.
- Faster reliable sytem.
- The main objective of the Movie Ticket Booking System is to manage the details of Seats, Booking, Customer, Payment, Shows. It manages all the information about Seats, Movie, Shows, Seats.

TOOLS AND TECHNOLOGIES USED:

This project is completely developed using FastAPI for making the APIs for performing all the CRUD operations on the database ,the database used for this project is MySQL and used the SQLALCHEMY ORM and PYDANTIC libraries. For the API testing purposes, used the SWAGGER UI.

THE PROBLEM STATEMENT :

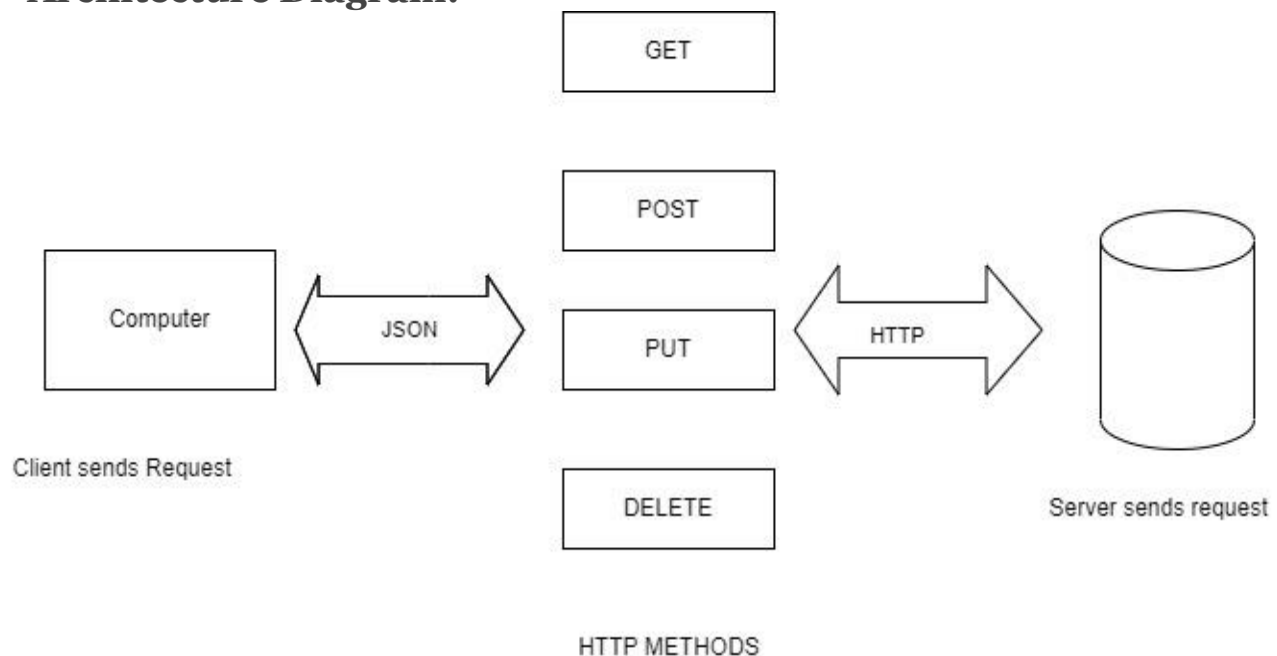
This project requires to make APIs for the steps involved in booking a movie ticket where one has to take care of every situation that may arise during booking a movie ticket. This project also requires one to handle all the corner cases and errors that can occur during the flow of this project.

FLOW OF THE PROJECT:

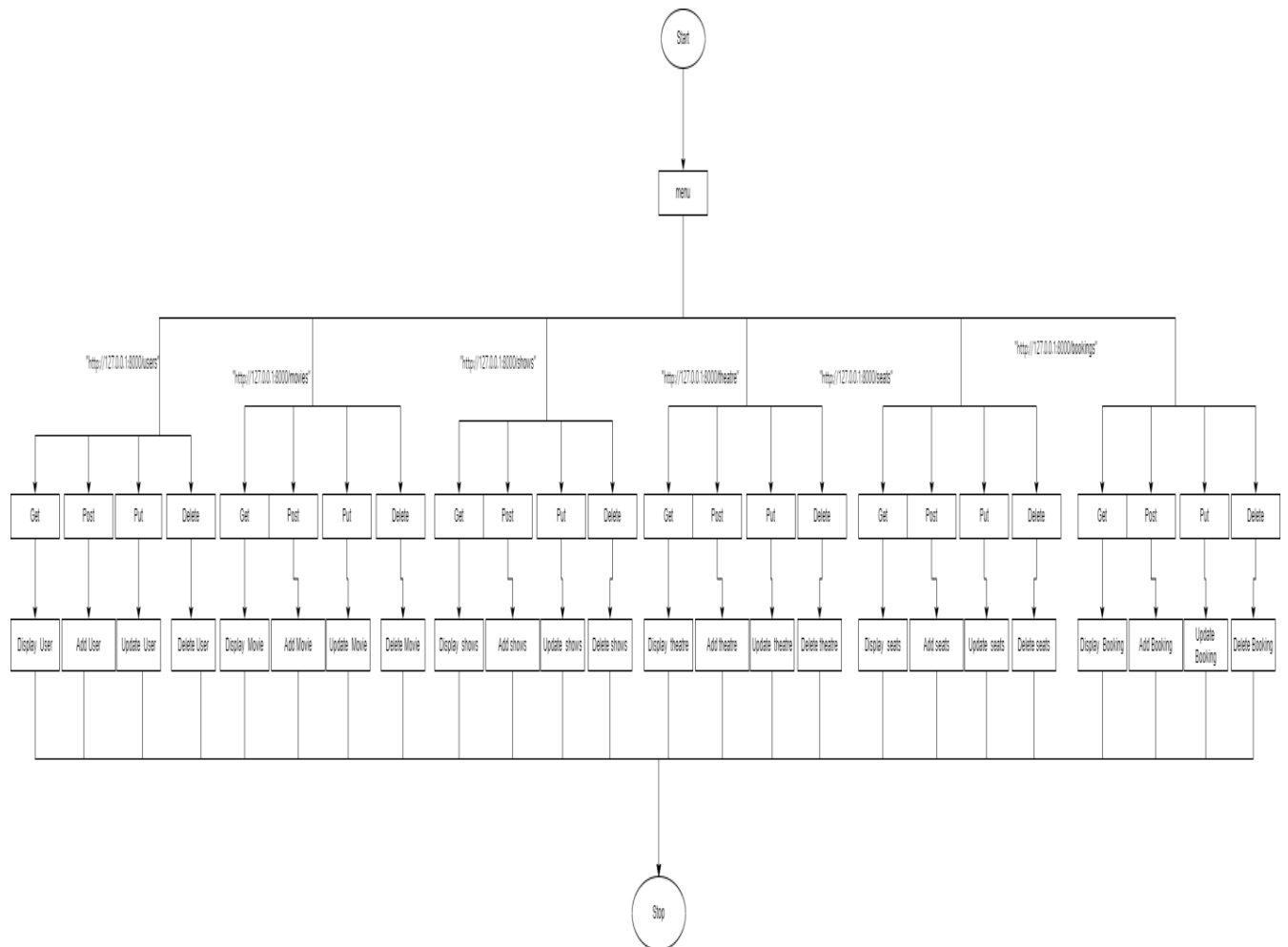
This project has APIs made specifically for multiple steps involved during booking a movie ticket. Specifically this project has six dedicated APIs for :

- Adding a new movie
- Booking a movie ticket
- Fetching the booking details of a user
- Listing all the available movies
- Fetching the total number of available seats for a particular movie
- The First three APIs mentioned in the above list are for generating a POST request and the rest three APIs for generating the GET request.

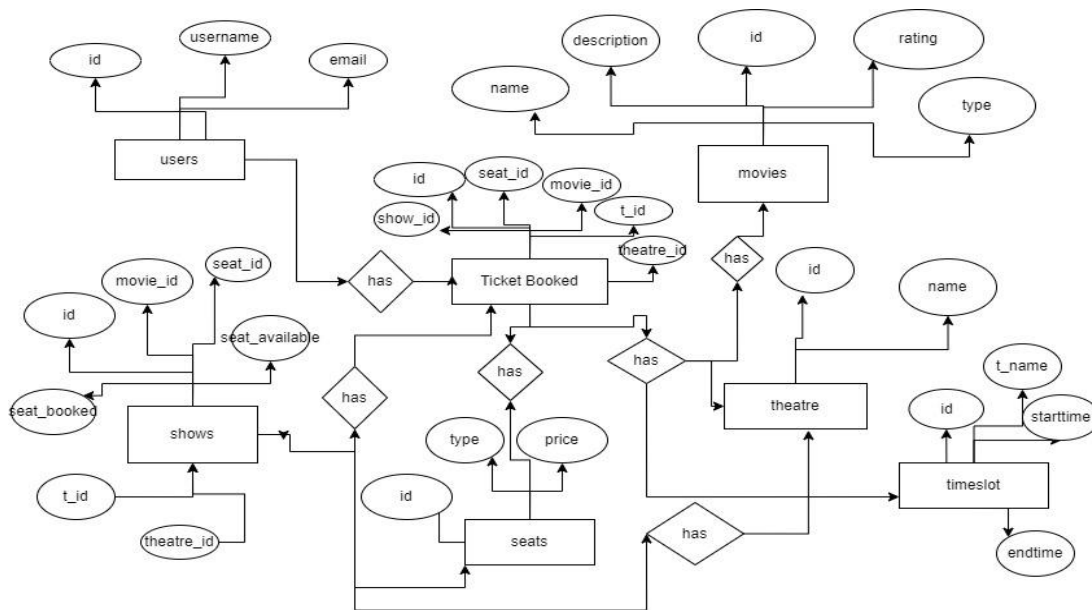
Architecture Diagram:



Flow Diagram:



ER-Diagram:



1. Install FastAPI

The first step is to install FastAPI

```
pip install fastapi
```

2. Install uvicorn to work as the server:

```
pip install uvicorn
```

3. Install database requirements

```
pip install fastapi, uvicorn, pip install sqlalchemy
```

4. To Run:

```
Uvicorn main:app --reload
```

Github Link:-

<https://github.com/Technical-06/movie.git>

5.db.py

1.schema:

```
schemas.py > ShowsSchema
1  from typing import Optional
2  from pydantic import BaseModel
3
4  class UserSchema(BaseModel):
5      id:int
6      username:str
7      email:str
8
9      class Config:
10         orm_mode =True
11
12
13  class MovieSchema(BaseModel):
14      movie_id:int
15      movie_name:str
16      movie_description:str
17      movie_duration:str
18      rating:int
19      type:str
20
21
22      class Config:
23         orm_mode =True
24
25
26  class TheatreSchema(BaseModel):
27      theatre_id:int
28      theatre_name:str
29
30      class Config:
31         orm_mode =True
32
33  class SeatsSchema(BaseModel):
34      seat_id:int
35      seat_type:str
36      seat_price:int
37
38      class Config:
39         orm_mode =True
40
41  class ShowsSchema(BaseModel):
42      show_id:int
43      theatre_id:int
44      movie_id:int
45      timeslot_id:int
46      seats_booked:int
47      seat_available:int
```

File Edit Selection View Go Run Terminal Help schemas.py - movie - Visual Studio Code

EXPLORER

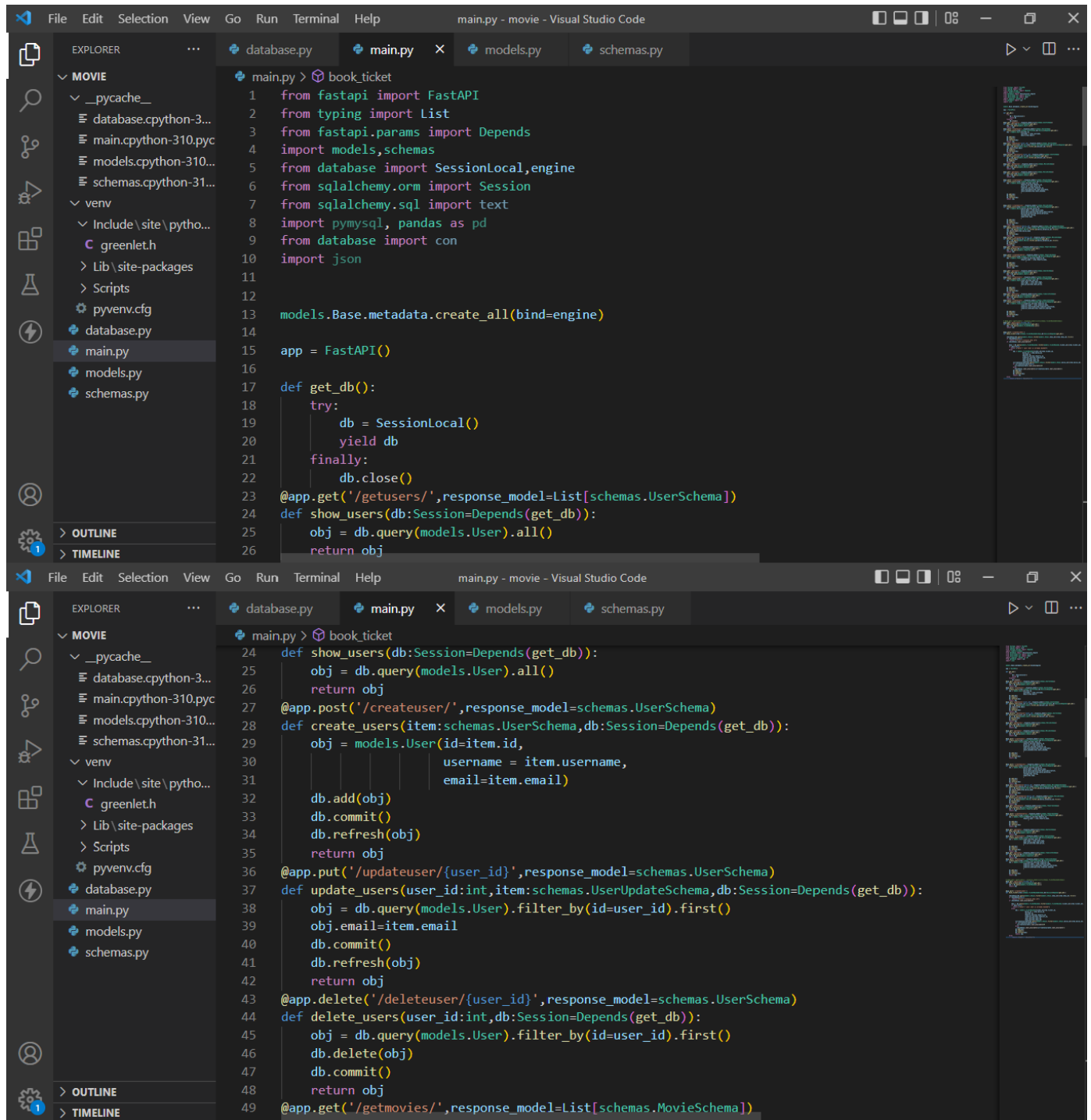
- MOVIE
 - __pycache__
 - database.cpython-310.pyc
 - main.cpython-310.pyc
 - models.cpython-310.pyc
 - schemas.cpython-310.pyc
 - venv
 - Include\site\pytho...
 - greenlet.h
 - Lib\site-packages
 - Scripts
 - pyvenv.cfg
 - database.py
 - main.py
 - models.py
 - schemas.py
- OUTLINE
- TIMELINE

database.py main.py models.py schemas.py X

schemas.py > ShowsSchema

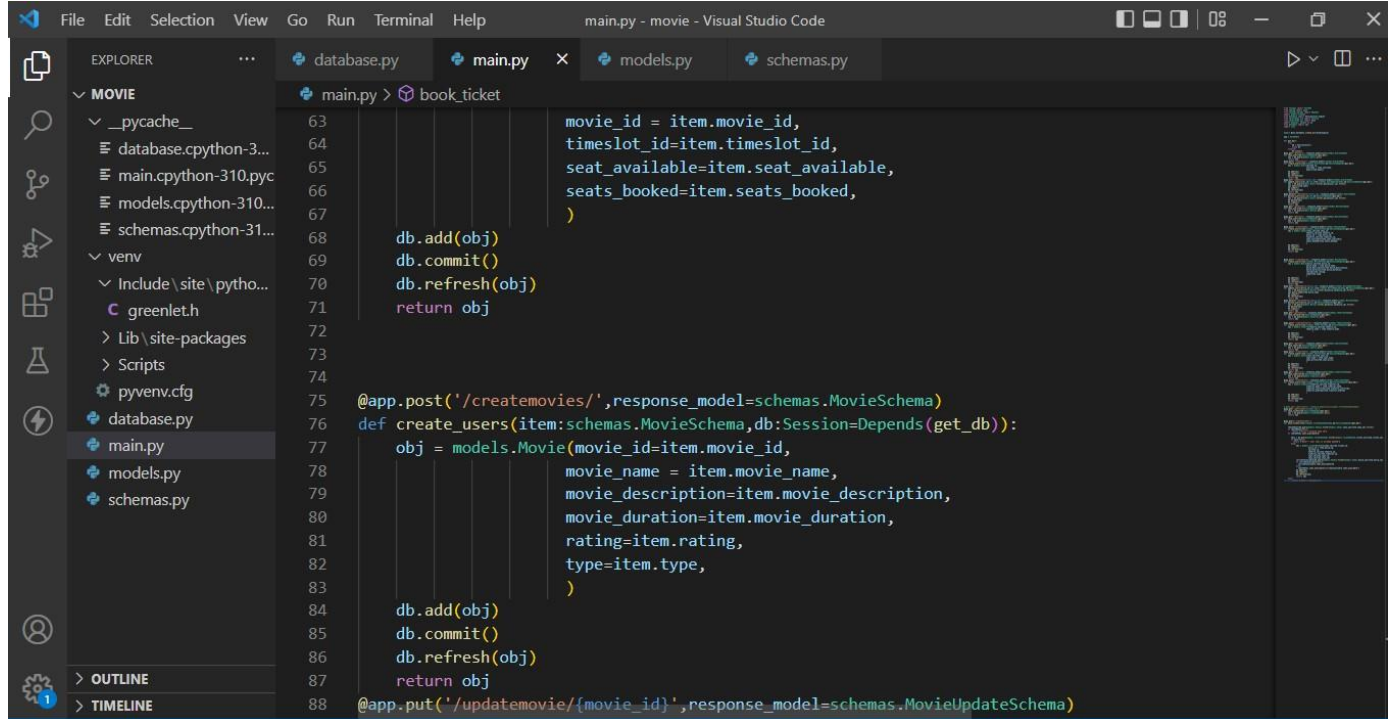
```
51 class TicketBookedSchema(BaseModel):
52     ticket_id:int
53     movie_id:int
54     id:int
55     theatre_id:int
56     timeslot_id:int
57     seat_id:int
58     show_id:int
59     class Config:
60         orm_mode =True
61 class TimeslotSchema(BaseModel):
62     timeslot_id:int
63     timeslot_name:str
64     timeslot_starttime:str
65     timeslot_endtime:int
66
67     class Config:
68         orm_mode =True
69
70
71
72 class UserUpdateSchema(BaseModel):
73     email:str
74
75     class Config:
76         orm_mode =True
```

2. Routes for users:



```
1 from fastapi import FastAPI
2 from typing import List
3 from fastapi.params import Depends
4 import models, schemas
5 from database import SessionLocal, engine
6 from sqlalchemy.orm import Session
7 from sqlalchemy.sql import text
8 import pymysql, pandas as pd
9 from database import con
10 import json
11
12 models.Base.metadata.create_all(bind=engine)
13
14 app = FastAPI()
15
16 def get_db():
17     try:
18         db = SessionLocal()
19         yield db
20     finally:
21         db.close()
22
23 @app.get('/getusers/', response_model=List[schemas.UserSchema])
24 def show_users(db: Session=Depends(get_db)):
25     obj = db.query(models.User).all()
26     return obj
27
28 @app.post('/createuser/', response_model=schemas.UserSchema)
29 def create_users(item: schemas.UserSchema, db: Session=Depends(get_db)):
30     obj = models.User(id=item.id,
31                       username=item.username,
32                       email=item.email)
33     db.add(obj)
34     db.commit()
35     db.refresh(obj)
36     return obj
37
38 @app.put('/updateuser/{user_id}', response_model=schemas.UserSchema)
39 def update_users(user_id: int, item: schemas.UserUpdateSchema, db: Session=Depends(get_db)):
40     obj = db.query(models.User).filter_by(id=user_id).first()
41     obj.email=item.email
42     db.commit()
43     db.refresh(obj)
44     return obj
45
46 @app.delete('/deleteuser/{user_id}', response_model=schemas.UserSchema)
47 def delete_users(user_id: int, db: Session=Depends(get_db)):
48     obj = db.query(models.User).filter_by(id=user_id).first()
49     db.delete(obj)
50     db.commit()
51     return obj
52
53 @app.get('/getmovies/', response_model=List[schemas.MovieSchema])
54 def get_movies():
55     return []
```


3.Routes for movies:



The screenshot shows the Visual Studio Code interface with the 'main.py' file open. The Explorer sidebar on the left shows a project structure with a 'MOVIE' folder containing files like 'database.py', 'main.py', 'models.py', and 'schemas.py'. The main editor displays the following Python code:

```
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
```

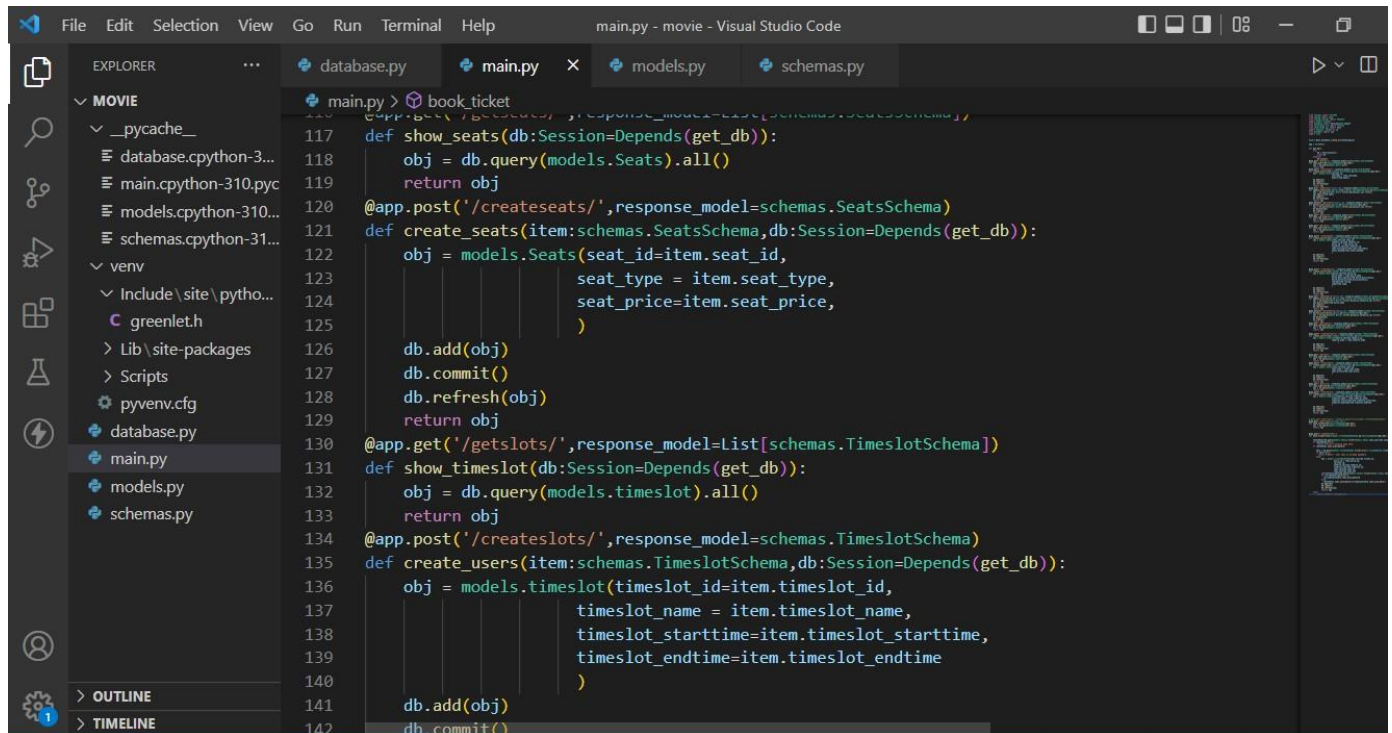
```
movie_id = item.movie_id,
timeslot_id=item.timeslot_id,
seat_available=item.seat_available,
seats_booked=item.seats_booked,
)

db.add(obj)
db.commit()
db.refresh(obj)
return obj

@app.post('/createmovies/',response_model=schemas.MovieSchema)
def create_users(item:schemas.MovieSchema,db:Session=Depends(get_db)):
    obj = models.Movie(movie_id=item.movie_id,
                        movie_name = item.movie_name,
                        movie_description=item.movie_description,
                        movie_duration=item.movie_duration,
                        rating=item.rating,
                        type=item.type,
                        )
    db.add(obj)
    db.commit()
    db.refresh(obj)
    return obj

@app.put('/updatemovie/{movie_id}',response_model=schemas.MovieUpdateSchema)
```

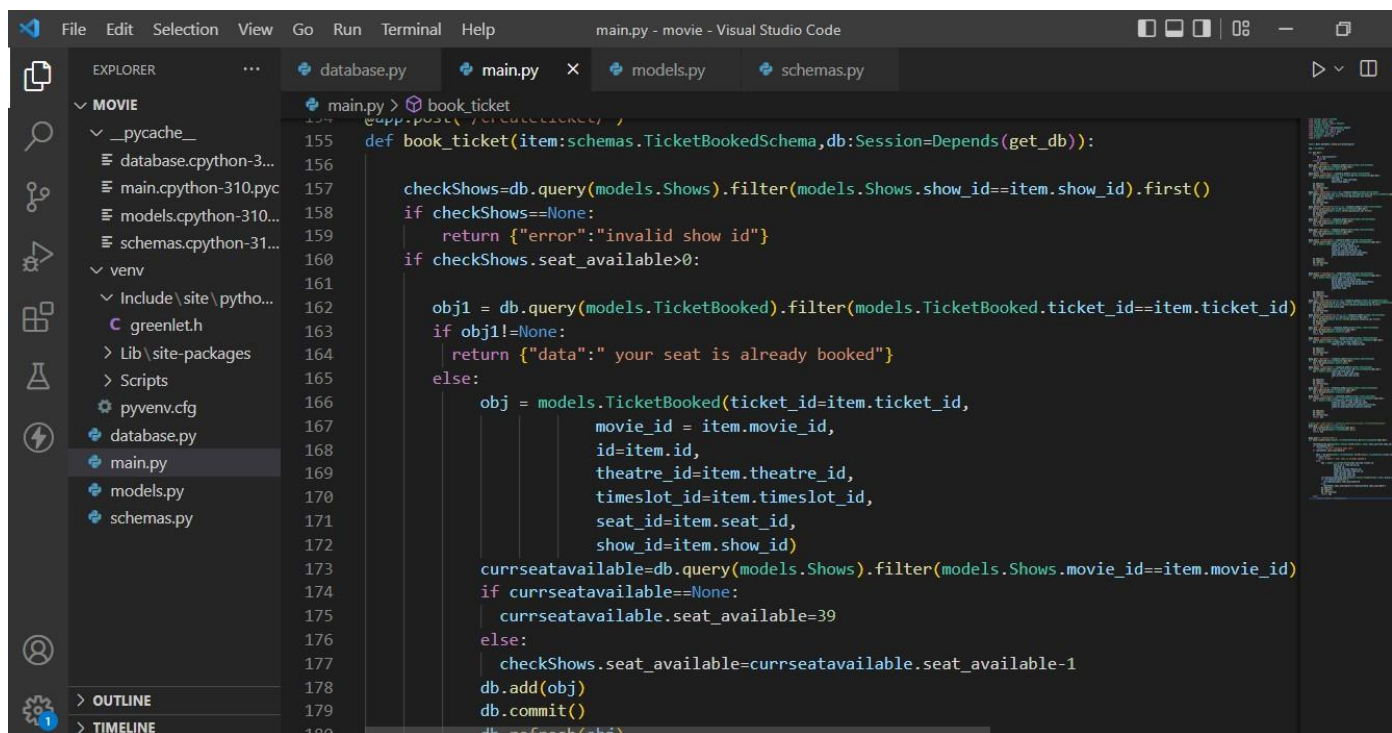
4. Routes for slots:



The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying a project structure for a movie application. The main editor window shows the `main.py` file with the following code:

```
117 def show_seats(db:Session=Depends(get_db)):
118     obj = db.query(models.Seats).all()
119     return obj
120 @app.post('/createseats/',response_model=schemas.SeatsSchema)
121 def create_seats(item:schemas.SeatsSchema,db:Session=Depends(get_db)):
122     obj = models.Seats(seat_id=item.seat_id,
123                       seat_type = item.seat_type,
124                       seat_price=item.seat_price,
125                       )
126     db.add(obj)
127     db.commit()
128     db.refresh(obj)
129     return obj
130 @app.get('/getslots/',response_model>List[schemas.TimeslotSchema])
131 def show_timeslot(db:Session=Depends(get_db)):
132     obj = db.query(models.timeslot).all()
133     return obj
134 @app.post('/createslots/',response_model=schemas.TimeslotSchema)
135 def create_users(item:schemas.TimeslotSchema,db:Session=Depends(get_db)):
136     obj = models.timeslot(timeslot_id=item.timeslot_id,
137                           timeslot_name = item.timeslot_name,
138                           timeslot_starttime=item.timeslot_starttime,
139                           timeslot_endtime=item.timeslot_endtime
140                           )
141     db.add(obj)
142     db.commit()
```

5. ROUTES FOR BOOKING:



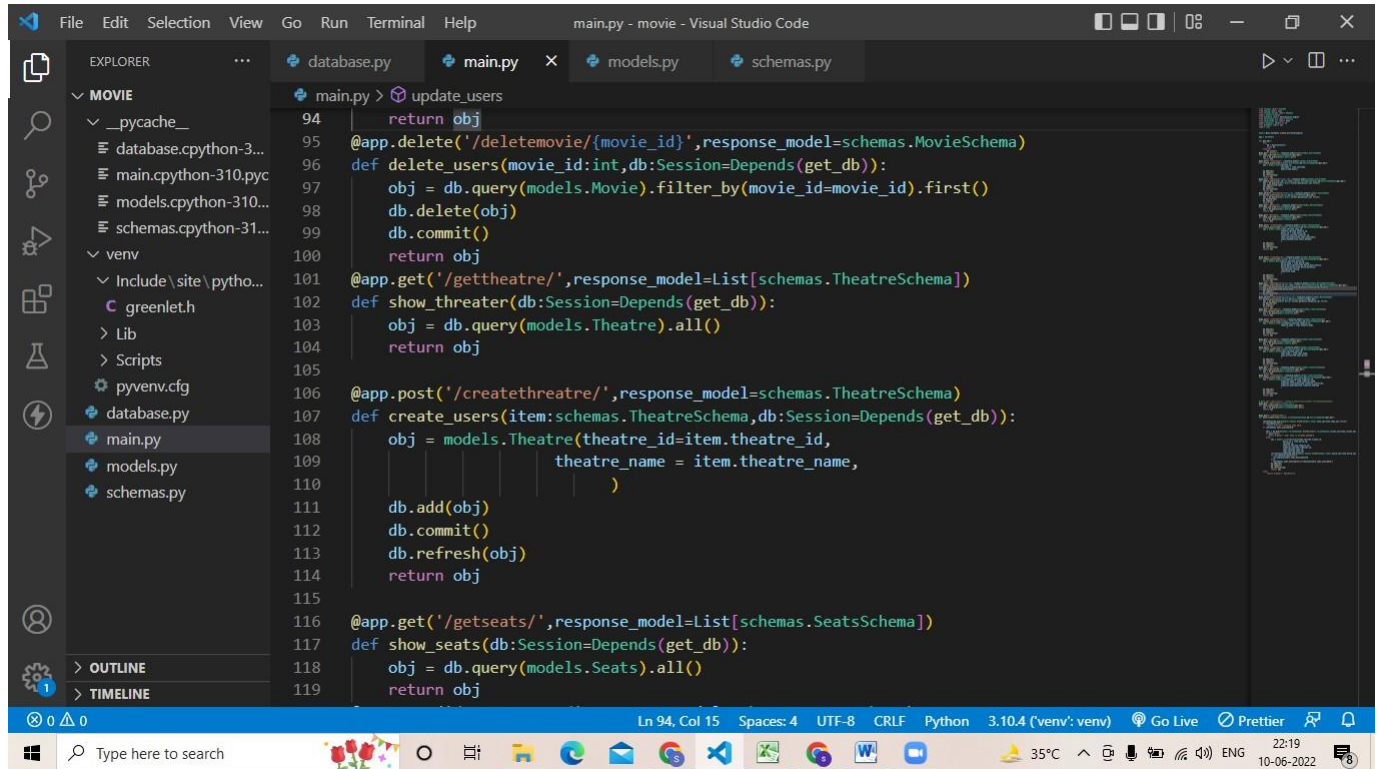
The screenshot shows the Visual Studio Code editor with the file explorer on the left displaying a project structure for a movie application. The main editor window shows the `main.py` file with the following code:

```
155 def book_ticket(item:schemas.TicketBookedSchema,db:Session=Depends(get_db)):
156
157     checkShows=db.query(models.Shows).filter(models.Shows.show_id==item.show_id).first()
158     if checkShows==None:
159         return {"error":"invalid show id"}
160     if checkShows.seat_available>0:
161
162         obj1 = db.query(models.TicketBooked).filter(models.TicketBooked.ticket_id==item.ticket_id)
163         if obj1!=None:
164             return {"data":" your seat is already booked"}
165         else:
166             obj = models.TicketBooked(ticket_id=item.ticket_id,
167                                       movie_id = item.movie_id,
168                                       id=item.id,
169                                       theatre_id=item.theatre_id,
170                                       timeslot_id=item.timeslot_id,
171                                       seat_id=item.seat_id,
172                                       show_id=item.show_id)
173             currseataavailable=db.query(models.Shows).filter(models.Shows.movie_id==item.movie_id)
174             if currseataavailable==None:
175                 currseataavailable.seat_available=39
176             else:
177                 checkShows.seat_available=currseataavailable.seat_available-1
178             db.add(obj)
179             db.commit()
180             db.refresh(obj)
```

6. Routes for Seats :

```
106 @app.post('/createtheatre/', response_model=schemas.TheatreSchema)
107 def create_theatre(item: schemas.TheatreSchema, db: Session = Depends(get_db)):
108     obj = models.Theatre(theatre_id=item.theatre_id,
109                          theatre_name=item.theatre_name,
110                          )
111     db.add(obj)
112     db.commit()
113     db.refresh(obj)
114     return obj
115
116 @app.get('/getseats/', response_model=List[schemas.SeatsSchema])
117 def show_seats(db: Session = Depends(get_db)):
118     obj = db.query(models.Seats).all()
119     return obj
120
121 @app.post('/createseats/', response_model=schemas.SeatsSchema)
122 def create_seats(item: schemas.SeatsSchema, db: Session = Depends(get_db)):
123     obj = models.Seats(seat_id=item.seat_id,
124                       seat_type=item.seat_type,
125                       seat_price=item.seat_price,
126                       )
127     db.add(obj)
128     db.commit()
129     db.refresh(obj)
130     return obj
131
132 @app.get('/getslots/', response_model=List[schemas.TimeslotSchema])
133 def show_timeslot(db: Session = Depends(get_db)):
```

7. Routes for Theatre:



The screenshot shows a Visual Studio Code editor window titled 'main.py - movie - Visual Studio Code'. The Explorer sidebar on the left shows a project structure with folders like MOVIE, database.py, main.py, models.py, and schemas.py. The main editor displays the code in main.py, which includes routes for updating users, deleting users, showing theatres, creating users, and showing seats. The code is as follows:

```
94     return obj
95
96 @app.delete('/deletemovie/{movie_id}', response_model=schemas.MovieSchema)
97 def delete_users(movie_id:int,db:Session=Depends(get_db)):
98     obj = db.query(models.Movie).filter_by(movie_id=movie_id).first()
99     db.delete(obj)
100    db.commit()
101    return obj
102
103 @app.get('/gettheatre/', response_model=List[schemas.TheatreSchema])
104 def show_threater(db:Session=Depends(get_db)):
105     obj = db.query(models.Theatre).all()
106     return obj
107
108 @app.post('/createtheatre/', response_model=schemas.TheatreSchema)
109 def create_users(item:schemas.TheatreSchema,db:Session=Depends(get_db)):
110     obj = models.Theatre(theatre_id=item.theatre_id,
111                          theatre_name = item.theatre_name,
112                          )
113     db.add(obj)
114     db.commit()
115     db.refresh(obj)
116     return obj
117
118 @app.get('/getseats/', response_model=List[schemas.SeatsSchema])
119 def show_seats(db:Session=Depends(get_db)):
120     obj = db.query(models.Seats).all()
121     return obj
```

The status bar at the bottom indicates the current position is Ln 94, Col 15, with 4 spaces, UTF-8 encoding, CRLF line endings, Python 3.10.4 interpreter, and various extensions like Go Live and Prettier are active. The system tray shows the date and time as 10-06-2022, 22:19.

OUTPUT:

127.0.0.1:8000/docs#/default/book_ticket_createticket_post

ExecuteClear

Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/getusers/' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/getusers/
```

Server response

CodeDetails

200

Response body

```
{
  "id": 1,
  "username": "sonal",
  "email": "sonal@gmail.com"
}
```

Response headers

```
content-length: 55
content-type: application/json
date: Fri, 10 Jun 2022 15:33:07 GMT
server: uvicorn
```

127.0.0.1:8000/docs#/default/book_ticket_createticket_post

PUT /updateuser/{user_id} Update Users

ParametersCancelReset

Name	Description
user_id * required	
integer	2
(path)	

Request body requiredapplication/json

```
{
  "email": "s@gmail.com"
}
```

DELETE /deleteuser/{user_id} Delete Users

Parameters

Cancel

Name	Description
user_id * required	
integer	
(path)	

Execute

Clear

Responses

Curl

```
curl -X 'DELETE' \
  'http://127.0.0.1:8000/deleteuser/2' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/deleteuser/2
```

Curl

```
curl -X 'GET' \
  'http://127.0.0.1:8000/getmovies/' \
  -H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/getmovies/
```

Server response

Code	Details
200	<div><div>Response body</div><div><pre>[{ "movie_id": 101, "movie_name": "shiddat2", "movie_description": "nice", "movie_duration": "120mins", "rating": 5, "type": "emotional" }]</pre></div><div>Download</div></div> <div><div>Response headers</div><div><pre>content-length: 126 content-type: application/json date: Fri, 10 Jun 2022 15:34:27 GMT server: uvicorn</pre></div></div>

Responses

Code	Description	Links
200	Successful Response	No links

127.0.0.1:8000/docs#/default/show_users_getshows_get

Execute

Clear

Responses

Curl

```
curl -X 'GET' \
'http://127.0.0.1:8000/gettheatre/' \
-H 'accept: application/json'
```

Request URL

```
http://127.0.0.1:8000/gettheatre/
```

Server response

Code

Details

200

Response body

```
[
  {
    "theatre_id": 1,
    "theatre_name": "Supermall"
  },
  {
    "theatre_id": 2,
    "theatre_name": "vr"
  }
]
```

Download

Response headers

```
content-length: 82
content-type: application/json
date: Fri,10 Jun 2022 15:35:03 GMT
server: uvicorn
```

Curl

```
curl -X 'POST' \
'http://127.0.0.1:8000/createticket/' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "ticket_id": 11,
  "movie_id": 101,
  "id": 1,
  "theatre_id": 1,
  "timeslot_id": 1,
  "seat_id": 1,
  "show_id": 1
}'
```

Request URL

```
http://127.0.0.1:8000/createticket/
```

Server response

Code

Details

200

Response body

```
{
  "data": " Housefull"
}
```

Download

Response headers

```
content-length: 21
content-type: application/json
date: Fri,10 Jun 2022 15:36:06 GMT
server: uvicorn
```

```
-d '{
  "ticket_id": 1,
  "movie_id": 101,
  "id": 1,
  "theatre_id": 1,
  "timeslot_id": 1,
  "seat_id": 1,
  "show_id": 1
}'
```

Request URL

http://127.0.0.1:8000/createticket/

Server response

Code	Details
------	---------

200	
-----	--

Response body

```
{
  "data": " Housefull"
}
```



Download

Response headers

```
content-length: 21
content-type: application/json
date: Fri, 10 Jun 2022 15:36:22 GMT
server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response	No links

Responses

Curl

```
curl -X 'POST' \
'http://127.0.0.1:8000/createticket/' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "ticket_id": 1,
  "movie_id": 101,
  "id": 1,
  "theatre_id": 1,
  "timeslot_id": 1,
  "seat_id": 1,
  "show_id": 55
}'
```

Request URL

http://127.0.0.1:8000/createticket/

Server response

Code	Details
------	---------

200	
-----	--

Response body

```
{
  "error": "invalid show id"
}
```



Download