Telecom Web Application

23.03.2023

Sagar Pandey
STL Digital
8th Floor, Karle Town Centre
Bangalore

Overview

We are building a Telecom Web Application using React as frontend and FastApi as backend. In this project, react uses Javascript as a language and FastApi uses Python as a language.

Goals

1. **Frontend using:** Angular or React based on Training undergone

   (i) UI Microservice

2. **Backend using:** Python FastAPI Microservices

   (i) Facility of Login and Signup for Customers
   (ii) Facility to view Account/Profile Details (Mobile Number, current active plan, past plans, plan expiry date)

   (iii) Facility to view plans

   (iv) Facility to view plan details

   (v) Facility to recharge with a plan

   (vi) Facility to pay the bill (simulation)

   (vii) Facility to apply for new broadband connection.

Project Requirements

1. For the front end, we used React which helps to build its User Interface.
2. For the back end, we used FastApi which helps to build its backend.
3. For the database, we used the known database MySQL which helps store data.

Approach

I. Backend - FastApi

Introduction

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python-type hints.

Features

Based on open standards

1. for API creation, including declarations of path operations, parameters, body requests, security, etc.
2. Automatic data model documentation with **JSON Schema** (as OpenAPI itself is based on JSON Schema).**OpenAPI**
3. Designed around these standards, after a meticulous study. Instead of an afterthought layer on top.
4. This also allows using automatic **client code generation** in many language

First Step

The simplest FastAPI file could look like this:

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/")
async def root():
    return {"message": "Hello World"}
```

FastApi uses **Uvicorn** as a server to run. To run your fastapi file we have to run the following command in a terminal window:-

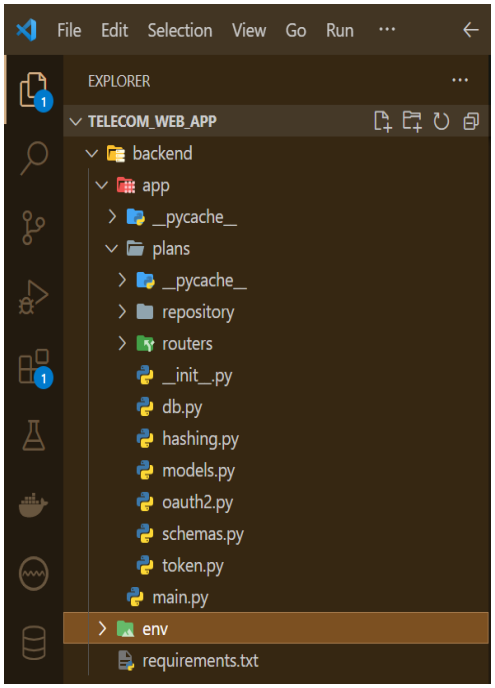
“ uvicorn main:app --reload ”

Project Use

FastApi is used as backend platform and also used for Api calling and testing.

In this project, I created 4 tags in which APIs are called and tested.

The following is the file structure of FastApi:-



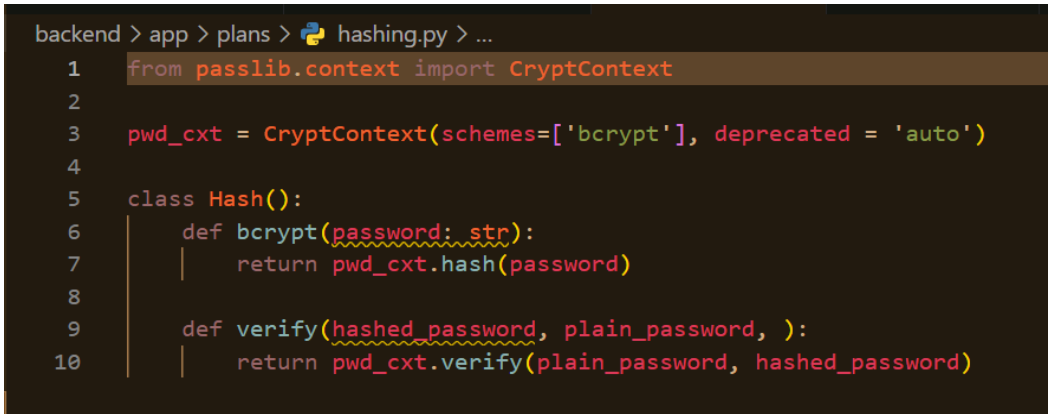
It contains folders in which working files are stored. The backend folder containsapp folder which contains **plans**, **repository**, **routers**, and **env** folders. It also contains a most important file called **main.py**.

(i) /plans – It contains most of the important files which are **db.py**, **hashing.py**, **models.py**, **schemas.py**, **oauth2.py**, and **token.py.**

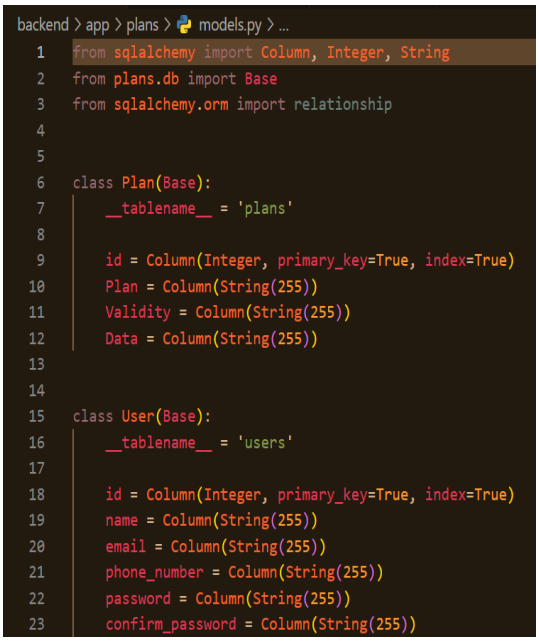
- **db.py -** This file contains all database configurations for the project. In this file, we have to declare our database instance, engine, and Session in which we have to store our data. This is a snap of the db.py file of the project:-

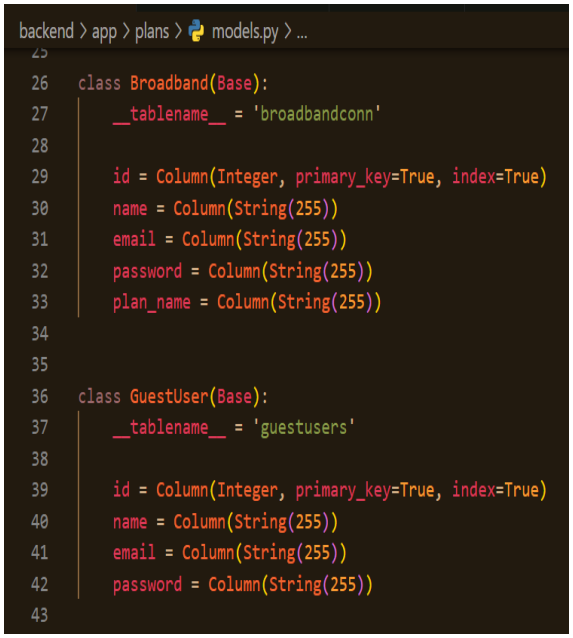
```python
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

SQLALCHEMY_DATABASE_URL = "mysql://root:Sagar0666@127.0.0.1:3306/userdb"

engine = create_engine(SQLALCHEMY_DATABASE_URL)

SessionLocal = sessionmaker(bind=engine,autocommit=False, autoflush=False)

Base = declarative_base()

def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

- **hashing.py :-** This file contains configuration for hashing of passwords generated by users. In this file, I am declaring a class called Hash which is used in other files for hashing passwords. This is a snap of the hashing.py file for the project:-



- **models.py**:- This file contains the structure of the tables we have to create, and what columns and rows we have to create. In this file, I declare four tables users, broadbandconn, plans, and guestusers. This is a snap of the models.py file for the project:-





- **schemas.py:-** This file contains the configuration for the schemas of the input and output that takes in FastApi. It allows us to declare the schema for User, Plan, Broadband, and GuestUser. This is the snap of the schemas.py file for the project:-

user.py ...\repository hashing.py 2

backend > app > plans > schemas.py > ...

```python
from pydantic import BaseModel
from typing import List


class BlogBase(BaseModel):
    Plan: str
    Validity: str
    Data: str


class Plan(BlogBase):
    class Config():
        orm_mode = True


class User(BaseModel):
    name: str
    email: str
    password: str
    confirm_password: str
    phone_number: str


class ShowUser(BaseModel):
    name: str
    email: str

    class Config():
        orm_mode = True
```

backend > app > plans > schemas.py > ...

```python
class Login(BaseModel):
    username: str
    password: str


class Token(BaseModel):
    access_token: str
    token_type: str


class TokenData(BaseModel):
    email: str | None = None


class BroadBase(BaseModel):
    name: str
    email: str
    password: str
    plan_name: str


class Broadband(BroadBase):
    class Config():
        orm_mode = True


class ShowBroadband(BaseModel):
    name: str
    email: str
```

- **token.py :-** This file contains the **JWT Token** for the project. In this file, I declare tokens that are used in the login process. This is a snap of the token.py file in the project.

backend > app > plans > token.py > ...

```python
from datetime import timedelta, datetime
from jose import JWTError, jwt
from plans import schemas


SECRET_KEY = "09d25e094faa6ca2556c818166b7a9563b93f7099f6f0f4caa6cf63b88e8d3e7"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 30

def create_access_token(data: dict):
    to_encode = data.copy()
    expire = datetime.utcnow() + timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    to_encode.update({"exp": expire})
    encoded_jwt = jwt.encode(to_encode, SECRET_KEY, algorithm=ALGORITHM)
    return encoded_jwt


def verify_token(token:str, credentials_exception):
    try:
        (variable) email: str ken, SECRET_KEY, algorithms=[ALGORITHM])
        email: str = payload.get("sub")
        if email is None:
            raise credentials_exception
        token_data = schemas.TokenData(email=email)
    except JWTError:
        raise credentials_exception
```

- **oauth2.py:-** This file contains the OAuth scheme for authentication of Fastapi's API. It verifies the token and authorizes the Apis. This is a snap of the oauth2.py file for the project.

```python
from fastapi import Depends, HTTPException, status
from fastapi.security import OAuth2PasswordBearer
from plans import token


outh2_scheme = OAuth2PasswordBearer(tokenUrl = "login")

def get_current_user(data: str = Depends(outh2_scheme)):
    credentials_exception = HTTPException(
        status_code=status.HTTP_401_UNAUTHORIZED,
        detail="Could not validate credentials",
        headers={"WWW-Authenticate": "Bearer"},
    )

    return token.verify_token(data, credentials_exception)
```

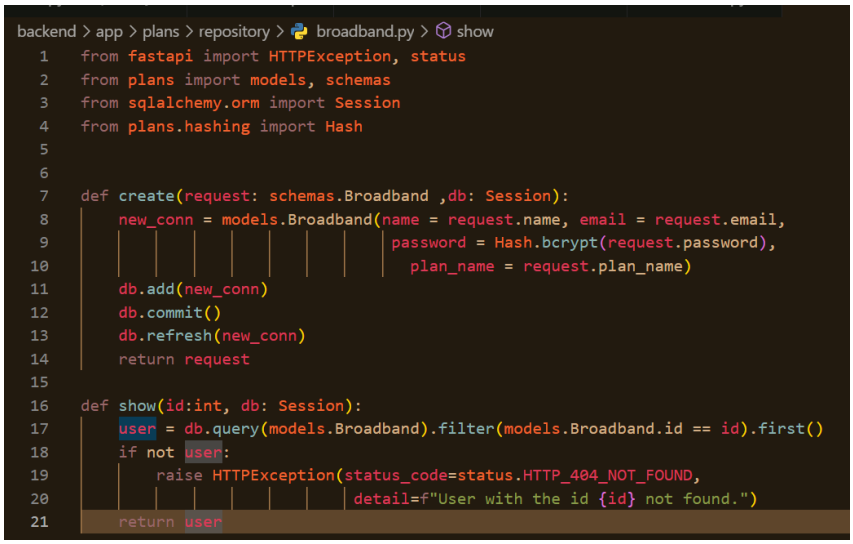
(ii) /repository - This folder contains **guestuser.py**, **broadband.py**, **user.py**, and **plan.py** in which I define all the functionalities of APIs and how API works and show output.

- **guestuser.py:-** In this file, I define functions for creating, showing, and deleting guest users. This is a snap of the guestuser.py file for the project.

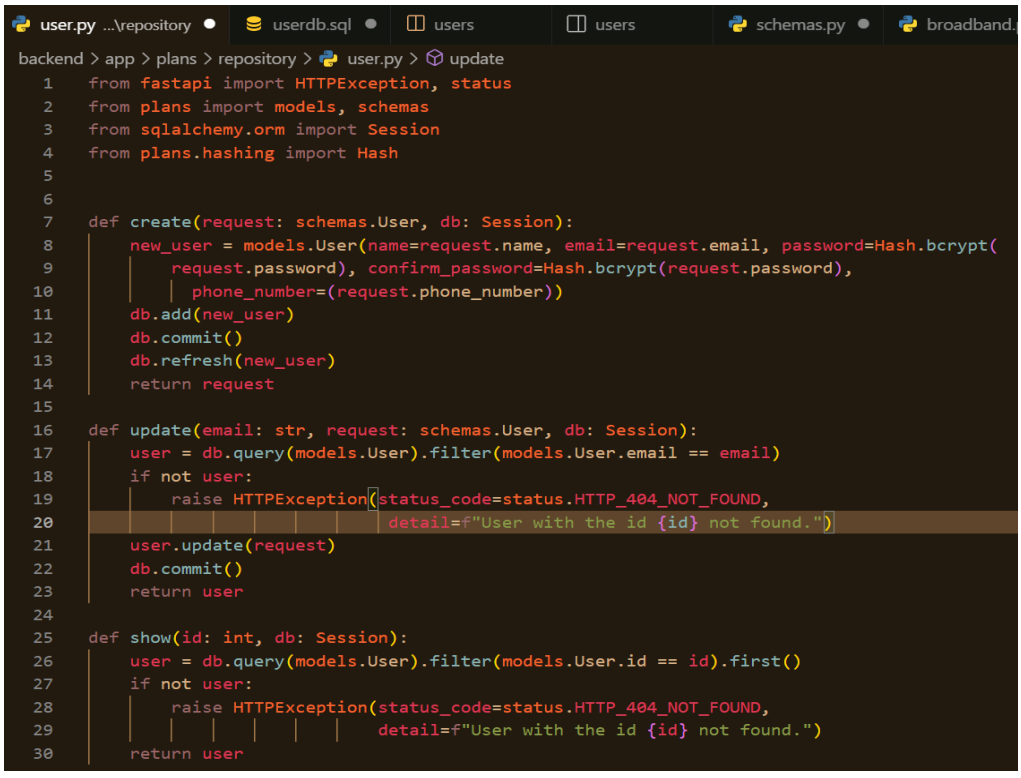
backend > app > plans > repository > guestuser.py > ...

```python
from fastapi import HTTPException, status
from plans import models, schemas
from sqlalchemy.orm import Session
from plans.hashing import Hash


def create(request: schemas.GuestUser ,db: Session):
    new_guest_user = models.GuestUser(name = request.name, email = request.email, password = Ha
    db.add(new_guest_user)
    db.commit()
    db.refresh(new_guest_user)
    return request

def show(email:str, db: Session):
    guest_user = db.query(models.GuestUser).filter(models.GuestUser.email == email).first()
    if not guest_user:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail=f"User with the email
    return guest_user

def delete_guest_user(id:int,db: Session):
    guest_user = db.query(models.GuestUser).filter(models.GuestUser.id == id)

    if not guest_user.first():
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
                            detail=f"Blog with id {id} not found")

    guest_user.delete(synchronize_session=False)
    db.commit()
    return 'done'
```

- **broadband.py:-** In this file, I define functions for creating and showing of broadband connections. This is a snap of the broadband.py file for the project:-



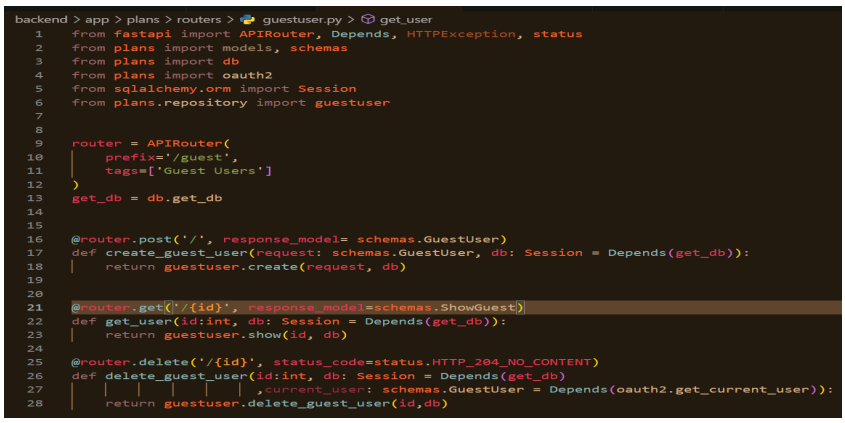
- **user.py:-** In this file, I define functions for creating, updating, and showing users in the project. This is a snap of the user.py file for the project:-



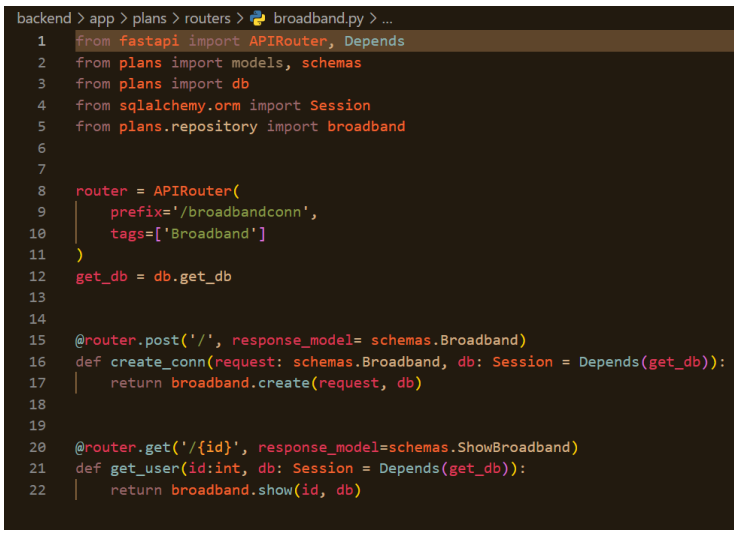
- **plan.py:-** In this file, I define functions for creating, showing, and showing all users in the project. This is a snap of the user.py file for the project:-

(iii) /routers - This folder contains **guestuser.py**, **broadband.py**, **user.py**, **plan.py, and authentication.py** files in I call the request **GET**, **POST**, **PUT** and **DELETE** with help of decorators (**@routers**).

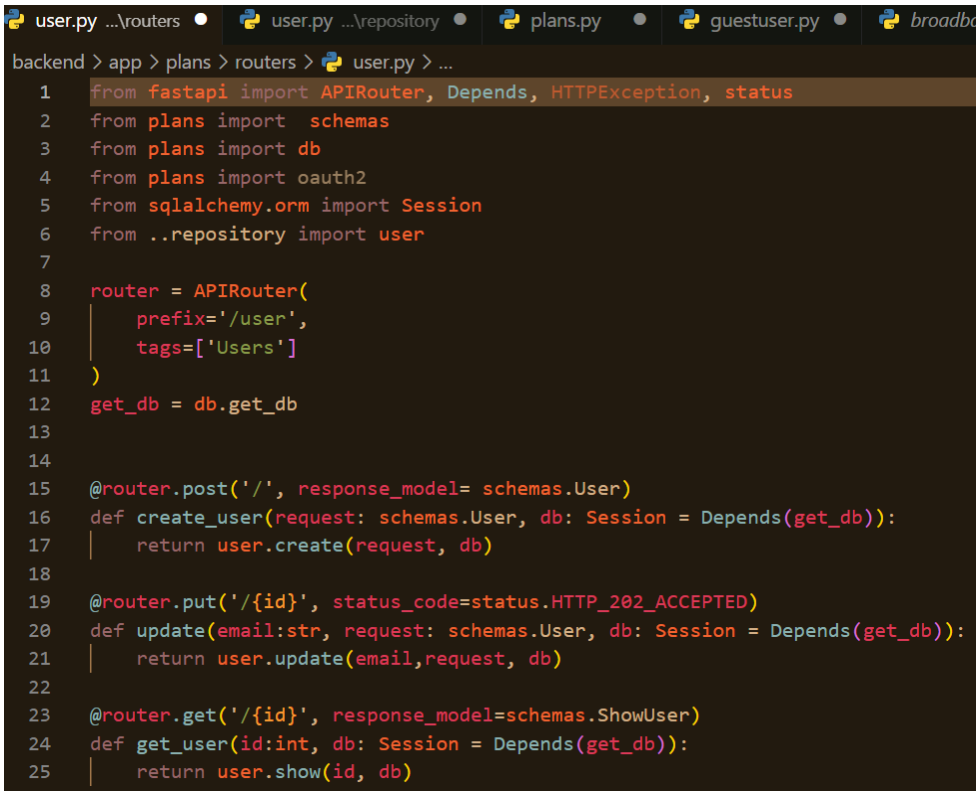
- **guestuser.py:-** In this file, I define routes for the API calling of guest users I call three APIs which are **GET** for getting guest users, **POST** for creating guest users, and **DELETE** for deleting guest users as guest users are temporary users. This is a snap of the guestuser.py file for the project:-



- **broadband.py:-** In this file, I define routes for the API calling of broadband connection I call two APIs which are **GET** for getting broadband users and **POST** for creating broadband connections. This is a snap of the broadband.py file for the project:-

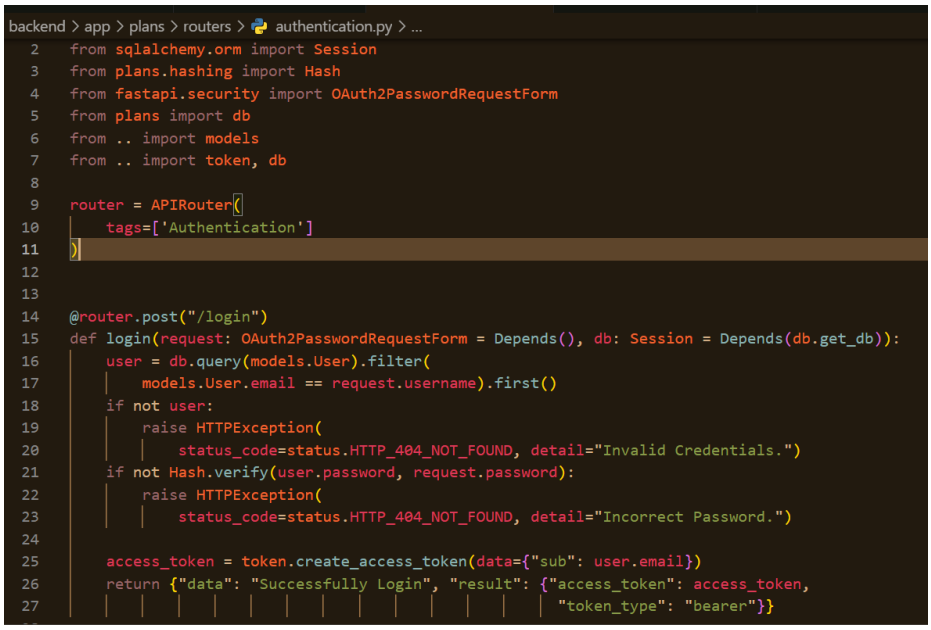


- **user.py:-** In this file, I define routes for the API calling of users I call three APIs which are **GET** for getting users, **POST** for creating new users, and **PUT** for updating the information of users. This is a snap of the user.py file for the project:-



- **plans.py:- I**n this file, I define routes for the API calling of users I call three APIs which are **GET** for getting the user with id and all users and **POST** for creating new plans. This is a snap of the plans.py file for the project.

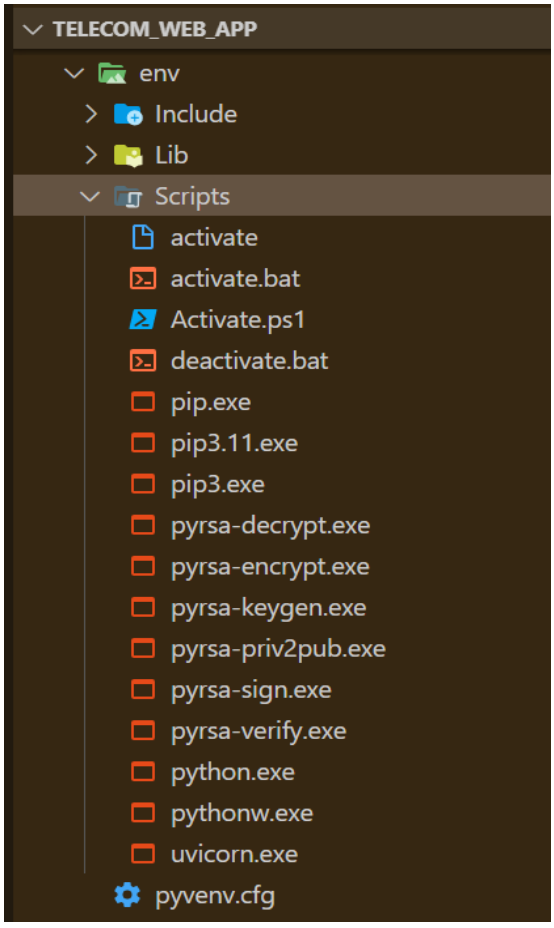
- **authentication.py:- I**n this file, I define routes for the API calling of users I call two APIs which are **POST** for login of users and **POST** for login of guest users. This is a snap of the plans.py file for the project:-



backend > app > plans > routers > authentication.py > ...

```python
@router.post("/guest_login")
def login(request: OAuth2PasswordRequestForm = Depends(), db: Session = Depends(db.get_db)):
    guest_user = db.query(models.GuestUser).filter(
        models.GuestUser.email == request.username).first()
    if not guest_user:
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND, detail="Invalid Credentials.")
    if not Hash.verify(guest_user.password, request.password):
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND, detail="Incorrect Password.")

    access_token = token.create_access_token(data={"sub": guest_user.email})
    return {"access_token": access_token, "token_type": "bearer"}

```

(iv) /env - This folder was created when we create a virtual environment in python using the following command: "**python -m venv env**". This folder contains subfolders which contain folders like Scripts which contain important files called to activate and specified an interpreter for the project. This is a snap of the file structure of the env folder:-



(v) main.py - This file is the most important file for FastApi, it consists of all routing of APIs and consist of function **app** which is used in uvicorn command to run the server. This is a snap of the main.py file for the project:-

backend > app > main.py > ...

```python
from fastapi import FastAPI
from starlette.middleware.cors import CORSMiddleware
from plans import models
from plans.db import get_db, engine
from plans.routers import plans, broadband, guestuser, user, authentication


app = FastAPI()

models.Base.metadata.create_all(engine)


app.include_router(user.router)
app.include_router(authentication.router)
app.include_router(plans.router)
app.include_router(broadband.router)
app.include_router(guestuser.router)


origins = [
    "https://localhost:3000",
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"]
)
```

Running FastApi

We can run our port using uvicorn server.

Uvicorn is an ASGI web server implementation for Python. Until recently Python has lacked a minimal low-level server/application interface for async frameworks.

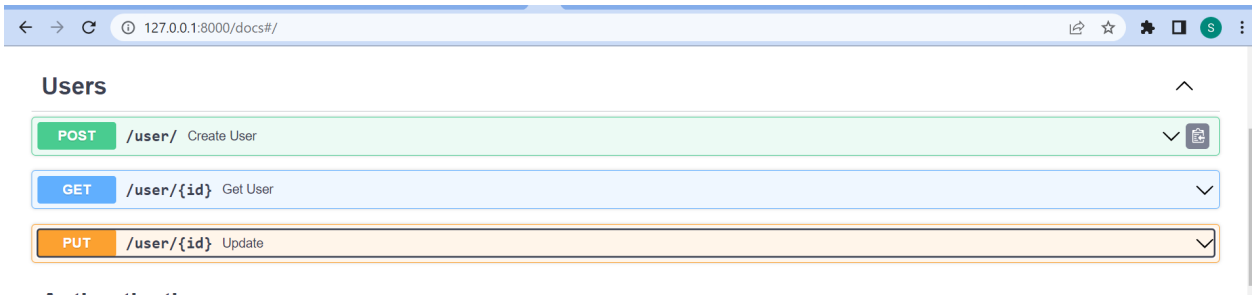
The ASGI specification fills this gap and means we're now able to start building a common set of tooling usable across all async frameworks. Uvicorn currently supports HTTP/1.1 and WebSockets. We can run our server using the following command:-

“**uvicorn main:app –reload**”

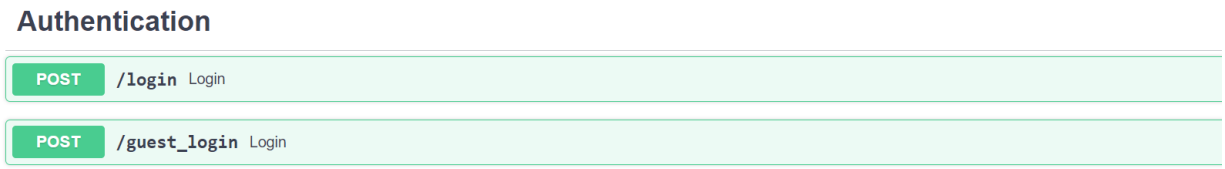
```
INFO:     Stopping reloader process [15080]
(env)
sagar.pandey@BAN-LL-SagarPan MINGW64 ~/Desktop/Telecom_Web_App/backend/app
$ uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\sagar.pandey\\Desktop\\Telecom_
Web_App\\backend\\app']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [3788] using StatReload
INFO:     Started server process [20124]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

After calling APIs through VisualCode Fastapi got the amazing feature of testing APIs directly through the web browser by applying **/docs** behind the local server URL i.e. http://127.0.0.1:8000/docs#/ This is snap of called APIs in fastapi's **Swagger UI:-**

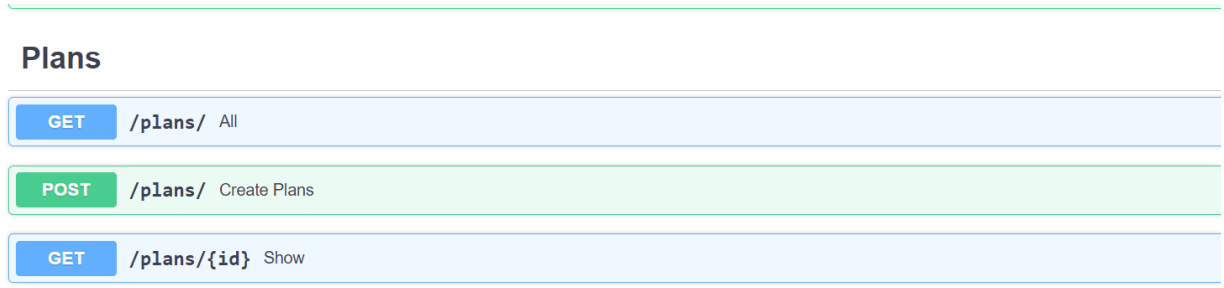
- **Users -**



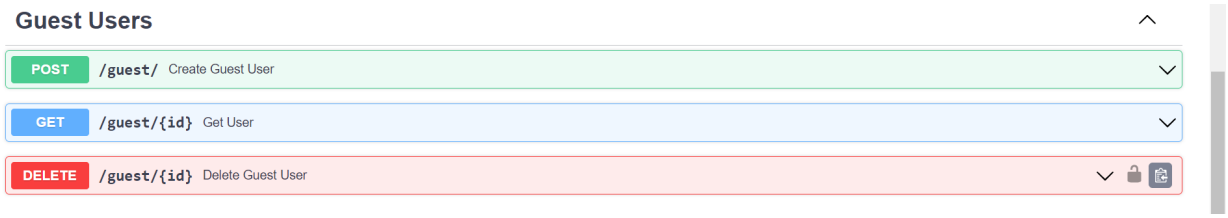
- **Authentication -**



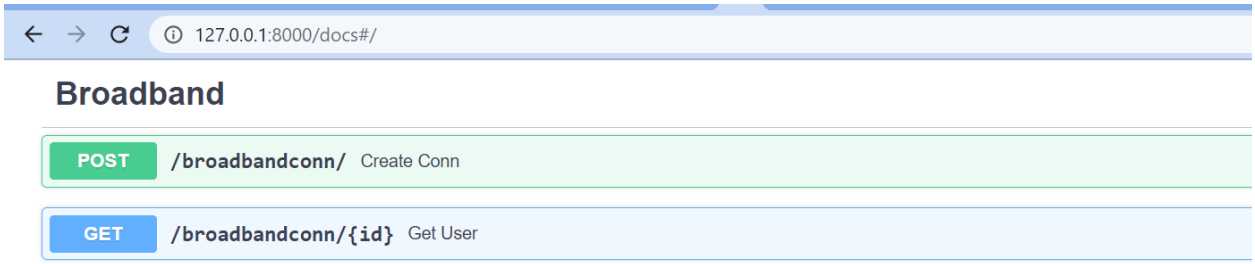
- **Plans -**



- **Guest Users -**

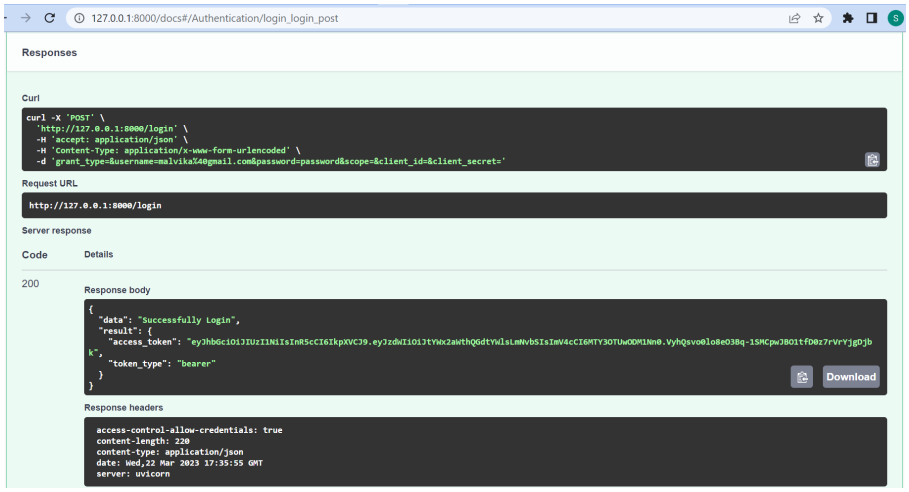


- **Broadband -**



We can test our called APIs by clicking on try it out, we can test create, show, update, and delete requests.

This is a snap of the result of the testing login API:-



II. Frontend - React

Introduction

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScriplibrary for building user interfaces based on components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies React can be used as a base in the development of single-page, mobile, or server-rendered applications with frameworks like Next.js. However, React is only concerned with the user interface and rendering components to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

Features

1. **Declarative:-** React adheres to the declarative programming paradigm (a programming paradigm). Developers design views for each state of an application and React updates and renders components when data changes. This is in contrast with imperative programming.
2. **Components:-** React code is made of entities called components. These components are reusable and must be formed in the SRC folder following the Pascal Case as its naming convention (capitalize camelCase).
3. **React Hooks:-** React provides a few built-in hooks like useState, useContext, useNavigate, and useEffect. Others are documented in the Hooks API Reference. useState and useEffect which are the most commonly used, are for controlling state and side effects respectively.
4. **Virtual-DOM:-** Another notable feature is the use of a virtual Document Object Model or virtual DOM. React creates an in-memory data-structure cache, computes the resulting differences, and then updates the browser's displayed DOM efficiently.

First Step

- **To create a new react application we have to run the following commands in our terminal window:-**

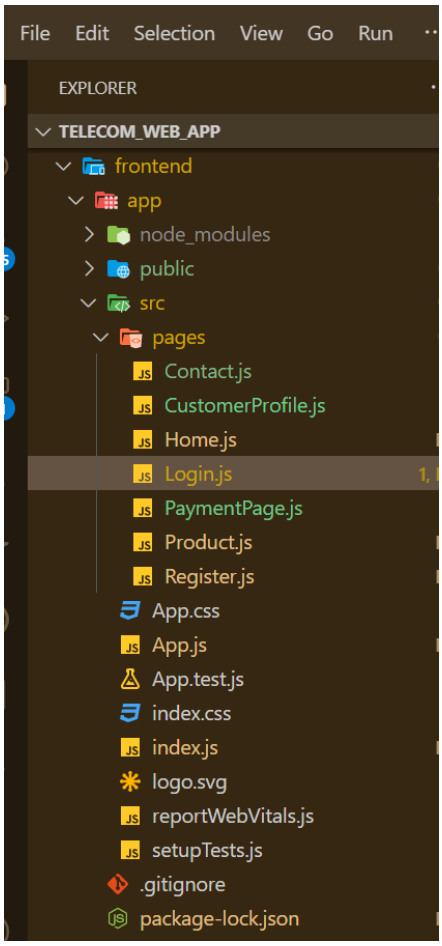
“npx create-react-app ‘app-name’” => “cd app-name” => “npm start”

This will create a blank react app from we can start our developing.

Project Use

React is used as a frontend platform for developing User Interface for the telecom web application.

The following is the file structure of frontend for the project:-



It contains folders in which frontend working files are stored. The frontend folder contains the app folder which contains **node_modules**, **public**, and **src** folder. In frontend folder’s src folder contains the **pages** folder which contains pages which are JavaScript files used to develop the pages of Telecom Web Application.

(i) /node_modules – A node_modules directory contains all the React dependencies packages: react, react-dom, and their transitive dependencies like webpack , babel, rxjs , ESLint, etc., to build and run a React project.

(ii) /public - The public folder contains the HTML file so you can tweak it, for example, to set the page title. The <script> tag with the compiled code will be added to it automatically during the build process.

(iii) /src – The src folder contains the pages folder which contains all javascript files which are used in developing the react application. In the src folder, the pages folder contains **Home.js**, **Contact.js**, **Login.js**, **Register.js**, **PaymentPage.js**, **CustomerProfile.js**, and **Product.js.**

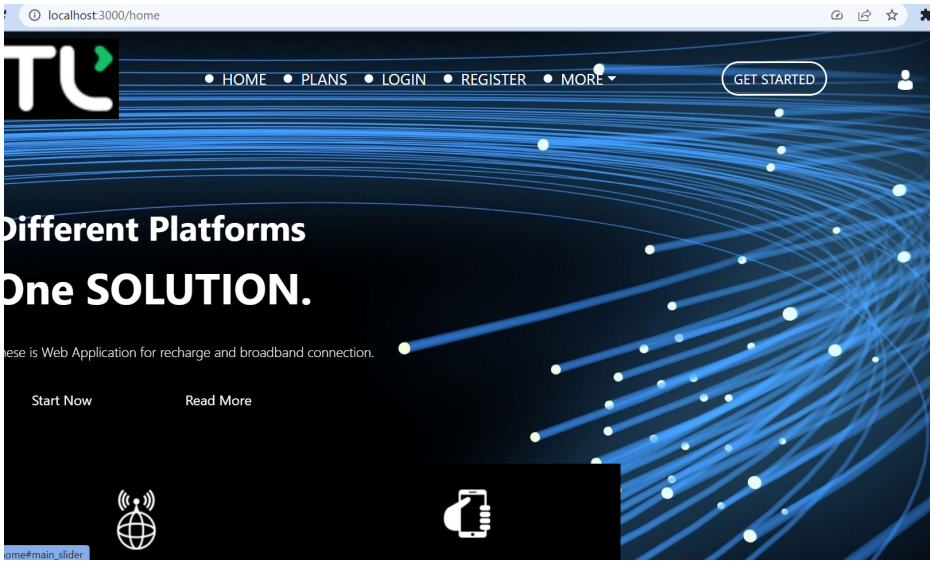
- **Home.js** - This file contains react hooks and JSX code for the UI of the home page. This is the link for the code of Home.js.
- **Contact.js -** This file contains react hooks and JSX code for the UI of the contact us page. This is the link for the code of Contact.js.
- **Login.js -** This file contains react hooks and JSX code for the UI of the login page. This is the link for the code of Login.js.
- **Register.js -** This file contains react hooks and JSX code for the UI of the register page. This is the link for the code of Register.js.
- **PaymentPage.js -** This file contains react hooks and JSX code for the UI of the payment page. This is the link for the code of PaymentPage.js.
- **CustomerProfile.js -** This file contains react hooks and JSX code for the UI of the profile page. This is the link for the code of CustomerProfile.js.
- **Product.js -** This file contains react hooks and JSX code for the UI of the plans page. This is the link for the code of Product.js.
- **App.js -** This file is the most important file for React App. It contains a function defined in which routing is done for redirecting the page and connecting them to each other. This file is responsible for running the react application. This is the link for the code of App.js.

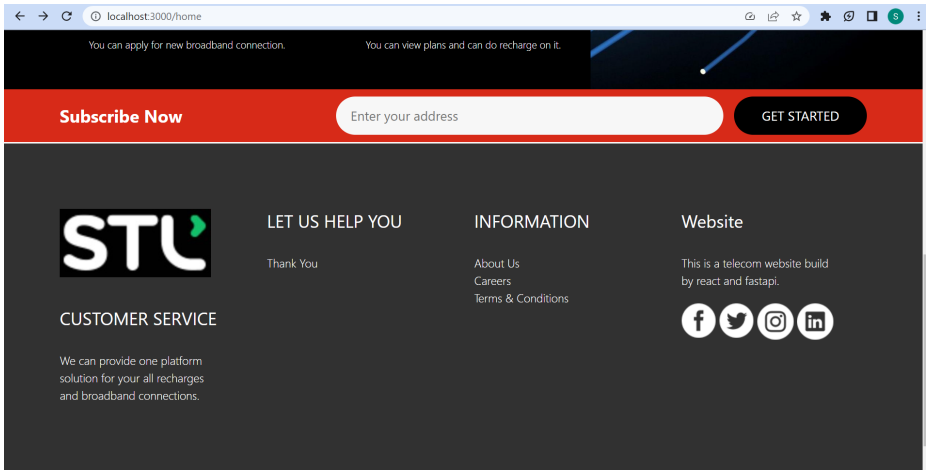
Running React

We can use the command "**npm start**" to start the server and run our application.

- npm is the world's largest software registry. Open source developers from every continent use npm to share and borrow packages, and many organizations use npm to manage private development as well.
- npm consists of three distinct components: the website, the Command Line Interface (CLI) and the registry.

Snap of Home page of Telecom Web Application.





III. Database - MySQL

Introduction

MySQL Database Service is a fully managed Oracle Cloud Infrastructure native service, which automates tasks such as backup and recovery, and database and operating system patching.

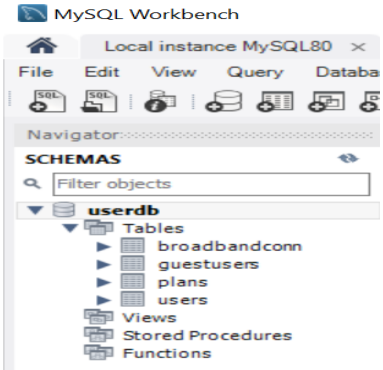
Features

1. **Easy to Use:-** MySQL is easy to use. We have to get only the basic knowledge of SQL. We can build and interact with MySQL by using only a few simple SQL statements.
2. **Client/Server Architecture:-** MySQL follows the working of a client/server architecture. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicate with the server; that is, they can query data, save changes, etc.
3. **Compatible on many OS:-** MySQL is compatible to run on many operating systems, like Novell NetWare, Windows* Linux*, many varieties of UNIX* (such as Sun* Solaris*, AIX, and DEC* UNIX), OS/2, FreeBSD*, and others.
4. **It is scalable:-** MySQL supports multi-threading which makes it easily scalable. It can handle almost any amount of data, up to as much as 50 million rows or more.

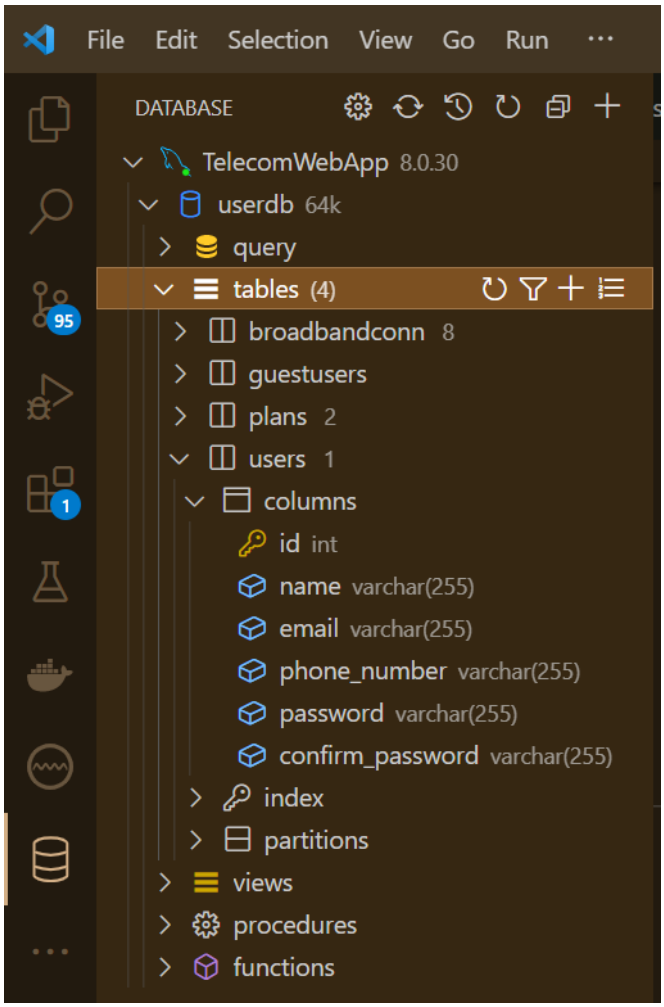
Project Use

In this project MySQL is used as a database. It is configured in the backend as db.py.

I am using userdb as a database instance.



Snap of Database conncection in VisualCode:-



Security

In this project **JWT** and **Oauth2** are used for login and authorization in the backend and I also used **Hashing** for encryption of passwords that makes login secure.

Basically, It generates tokens at the time of login and makes the site safe.

- JWT - JSON web token (JWT), pronounced "jot", is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. Again, JWT is a standard, meaning that all JWTs are tokens, but not all tokens are JWTs.
- Oauth - OAuth 2.0, which stands for “Open Authorization”, is a standard designed to allow a website or application to access resources hosted by other web apps on behalf of a user.
- Hashing - Hashing is the process of transforming any given key or a string of characters into another value. This is usually represented by a shorter, fixed-length value or key that represents and makes it easier to find or employ the original string.

Conclusion

In this way, I created a telecom web application using react as frontend, fastAPI as backend, and MySQL as the database.

In this project, my major learnings in the backend are how can we use fastapi as the backend and call the APIs and test the APIs on fastapi and with the help of JWT and Oauth2 we can provide security to the backend.

The front end is how can we use react as the front end and by using Javascript and major features of react like react hooks, and vDOM we can create a working web application.