**Mutual Fund Broker Web Application with RapidAPI Integration**

**1. Project Structure**

The project will have the following structure:

mutual-fund-broker/

├── backend/

│ ├── app/

│ │ ├── \_\_init\_\_.py

│ │ ├── models.py

│ │ ├── routes.py

│ │ ├── services/

│ │ │ ├── rapidapi\_integration.py

│ │ ├── tests/

│ │ │ ├── test\_endpoints.py

│ ├── migrations/

│ ├── .env

│ ├── requirements.txt

│ ├── config.py

│ ├── run.py

├── frontend/

│ ├── public/

│ ├── src/

│ │ ├── App.js

│ │ ├── components/

│ │ ├── services/

│ ├── package.json

├── README.md

├── postman\_collection.json

└── .gitignore

**2. Backend Code**

**Backend Setup with FastAPI**

**backend/app/\_\_init\_\_.py**

from fastapi import FastAPI

from app.routes import router

from app.database import Base, engine

# Initialize the database

Base.metadata.create\_all(bind=engine)

# Initialize FastAPI app

app = FastAPI(title="Mutual Fund Broker")

# Register routes

app.include\_router(router)

**backend/app/database.py**

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

import os

DATABASE\_URL = os.getenv("DATABASE\_URL", "sqlite:///./test.db")

engine = create\_engine(DATABASE\_URL, connect\_args={"check\_same\_thread": False})

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

Base = declarative\_base()

# Dependency to get DB session

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

**backend/app/models.py**

from sqlalchemy import Column, Integer, String, Float, ForeignKey

from sqlalchemy.orm import relationship

from app.database import Base

class User(Base):

\_\_tablename\_\_ = 'users'

id = Column(Integer, primary\_key=True, index=True)

email = Column(String, unique=True, index=True, nullable=False)

password = Column(String, nullable=False)

class Portfolio(Base):

\_\_tablename\_\_ = 'portfolios'

id = Column(Integer, primary\_key=True, index=True)

user\_id = Column(Integer, ForeignKey('users.id'), nullable=False)

fund\_name = Column(String, nullable=False)

fund\_value = Column(Float, nullable=False)

user = relationship("User", back\_populates="portfolio")

User.portfolio = relationship("Portfolio", back\_populates="user")

**backend/app/routes.py**

from fastapi import APIRouter, HTTPException, Depends

from sqlalchemy.orm import Session

from app.models import User, Portfolio

from app.services.rapidapi\_integration import fetch\_fund\_data

from app.database import get\_db

from passlib.context import CryptContext

from pydantic import BaseModel

router = APIRouter()

pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

# Request models

class UserRequest(BaseModel):

email: str

password: str

class FundRequest(BaseModel):

family: str

# User Registration

@router.post("/register")

def create\_account(user: UserRequest, db: Session = Depends(get\_db)):

hashed\_password = pwd\_context.hash(user.password)

new\_user = User(email=user.email, password=hashed\_password)

db.add(new\_user)

db.commit()

return {"message": "Account created successfully"}

# User Login

@router.post("/login")

def login(user: UserRequest, db: Session = Depends(get\_db)):

db\_user = db.query(User).filter(User.email == user.email).first()

if not db\_user or not pwd\_context.verify(user.password, db\_user.password):

raise HTTPException(status\_code=401, detail="Invalid credentials")

return {"message": "Login successful"}

# Fetch Funds

@router.post("/funds")

def get\_funds(fund\_request: FundRequest, db: Session = Depends(get\_db)):

funds = fetch\_fund\_data(fund\_request.family)

if not funds:

raise HTTPException(status\_code=404, detail="No funds found")

return funds

**backend/app/services/rapidapi\_integration.py**

import requests

import os

API\_KEY = os.getenv("RAPIDAPI\_KEY")

def fetch\_fund\_data(family):

url = "https://latest-mutual-fund-nav.p.rapidapi.com/fetchOpenEndFunds"

headers = {

"X-RapidAPI-Key": API\_KEY,

"X-RapidAPI-Host": "latest-mutual-fund-nav.p.rapidapi.com"

}

params = {"family": family}

response = requests.get(url, headers=headers, params=params)

if response.status\_code == 200:

return response.json()

return None

**backend/.env**

makefile

DATABASE\_URL=sqlite:///./test.db

RAPIDAPI\_KEY=<Your-RapidAPI-Key>

**backend/requirements.txt**

css

fastapi

uvicorn

sqlalchemy

passlib[bcrypt]

pydantic

python-dotenv

requests

**backend/run.py**

import uvicorn

if \_\_name\_\_ == "\_\_main\_\_":

uvicorn.run("app.\_\_init\_\_:app", host="0.0.0.0", port=8000, reload=True)

**2. Database Migration (Optional with Alembic)**

**Alembic Setup**

alembic init migrations

**Alembic env.py Changes**

Update the env.py to include:

from app.database import Base, engine

target\_metadata = Base.metadata

**Initial Migration**

alembic revision --autogenerate -m "Initial migration"

alembic upgrade head

**backend/migrations/versions/initial\_migration.py**

from alembic import op

import sqlalchemy as sa

revision = 'initial\_migration'

down\_revision = None

branch\_labels = None

depends\_on = None

def upgrade():

op.create\_table(

'users',

sa.Column('id', sa.Integer, primary\_key=True),

sa.Column('email', sa.String, unique=True, nullable=False),

sa.Column('password', sa.String, nullable=False),

)

op.create\_table(

'portfolios',

sa.Column('id', sa.Integer, primary\_key=True),

sa.Column('user\_id', sa.Integer, sa.ForeignKey('users.id'), nullable=False),

sa.Column('fund\_name', sa.String, nullable=False),

sa.Column('fund\_value', sa.Float, nullable=False),

)

def downgrade():

op.drop\_table('portfolios')

op.drop\_table('users')

**3. Frontend Code**

**React App (frontend/src/App.js)**

javascript

import React, { useState } from 'react';

function App() {

const [email, setEmail] = useState('');

const [password, setPassword] = useState('');

const [family, setFamily] = useState('');

const [funds, setFunds] = useState([]);

const register = async () => {

const response = await fetch('http://localhost:8000/register', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ email, password }),

});

const data = await response.json();

console.log(data);

};

const fetchFunds = async () => {

const response = await fetch('http://localhost:8000/funds', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ family }),

});

const data = await response.json();

setFunds(data);

}

return (

<div>

<h1>Mutual Fund Broker</h1>

<h2>Register</h2>

<input type="email" onChange={(e) => setEmail(e.target.value)} placeholder="Email" />

<input type="password" onChange={(e) => setPassword(e.target.value)} placeholder="Password" />

<button onClick={register}>Register</button>

<h2>Fetch Funds</h2>

<input type="text" onChange={(e) => setFamily(e.target.value)} placeholder="Fund Family" />

<button onClick={fetchFunds}>Fetch</button>

<h3>Funds</h3>

<ul>

{funds.map((fund, index) => (

<li key={index}>{fund.scheme\_name}</li>

))}

</ul>

</div>

);

}

**4. Git Setup**

**.gitignore**

bash

# Python

\*.pyc

\_\_pycache\_\_/

env/

venv/

.env

# Node.js

node\_modules/

build/

**Git Commands**

1. Initialize Git:

bash

git init

1. Add files:

bash

git add .

1. Commit:

bash

git commit -m "Initial commit for Mutual Fund Broker"

1. Push to GitHub:

bash

git branch -M main

git remote add origin <repository-url>

git push -u origin main

**6. Testing**

Use pytest or FastAPI's TestClient for testing:

**backend/app/tests/test\_endpoints.py**

python

from fastapi.testclient import TestClient

from app.\_\_init\_\_ import app

client = TestClient(app)

def test\_register():

response = client.post("/register", json={"email": "test@example.com", "password": "test123"})

assert response.status\_code == 200

def test\_login():

response = client.post("/login", json={"email": "test@example.com", "password": "test123"})

assert response.status\_code == 200