# FastAPI Banking System - Complete Documentation

## 1. Case Study Story

This application is a simple Banking System built using FastAPI. It allows users to register, log in, deposit funds, and view their transaction history. JWT authentication is used to secure user sessions, and Jinja2 is used for rendering web templates.

## 2. Setup and Execution Steps

* 1. Install Python (if not installed).
* 2. Create a virtual environment and activate it.
* 3. Install required dependencies using `pip install -r requirements.txt`.
* 4. Run `python -m app.database` to initialize the database.
* 5. Start the FastAPI server using `uvicorn app.main:app --reload`.
* 6. Open `http://127.0.0.1:8000/register` to register a new user.
* 7. Log in at `http://127.0.0.1:8000/login`.
* 8. Test deposit functionality from the dashboard.
* 9. Logout and ensure the authentication works correctly.

## 3. Folder Structure and Files

📂 Lab - FastAPI  
 ├── 📂 app  
 │ ├── 📂 templates (Jinja2 templates for frontend)  
 │ │ ├── base.html (Main layout template)  
 │ │ ├── index.html (Home page)  
 │ │ ├── login.html (Login page)  
 │ │ ├── register.html (Registration page)  
 │ │ ├── dashboard.html (User dashboard)  
 │ ├── 📂 static (CSS, images, etc.)  
 │ │ ├── style.css (Frontend styling)  
 │ ├── main.py (Application entry point)  
 │ ├── routes.py (API routes for authentication and transactions)  
 │ ├── database.py (Database connection and initialization)  
 │ ├── models.py (Database models for users and transactions)  
 │ ├── auth.py (JWT authentication functions)  
 │ ├── schemas.py (Pydantic models for request validation)  
 ├── requirements.txt (Python dependencies)  
 ├── README.md (Setup and usage instructions)  
 ├── bank.db (SQLite database)

## 4. Code Snippets

### 🔹 main.py

from fastapi import FastAPI

from fastapi.staticfiles import StaticFiles

from app.routes import router

from app.database import engine, Base

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

app = FastAPI()

app.mount("/static", StaticFiles(directory="app/static"), name="static")

app.include\_router(router)

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

    import uvicorn

    uvicorn.run("app.main:app", host="127.0.0.1", port=8000, reload=True)

### 🔹 routes.py

from fastapi import APIRouter, Depends, HTTPException, Request, Form, Cookie, Response

from sqlalchemy.orm import Session

from fastapi.templating import Jinja2Templates

from fastapi.responses import RedirectResponse

from app.database import SessionLocal

from app.models import User, Transaction

from app.auth import create\_jwt\_token, verify\_jwt\_token

router = APIRouter()

templates = Jinja2Templates(directory="app/templates")

# ✅ Database Dependency

def get\_db():

    db = SessionLocal()

    try:

        yield db

    finally:

        db.close()

# ✅ Home Page

@router.get("/")

def home(request: Request):

    return templates.TemplateResponse("index.html", {"request": request})

# ✅ Dashboard (Requires JWT Authentication)

@router.get("/dashboard")

def dashboard(request: Request, access\_token: str = Cookie(None), db: Session = Depends(get\_db)):

    if not access\_token:

        raise HTTPException(status\_code=401, detail="Not authenticated")

    token = access\_token.replace("Bearer ", "")  # ✅ Remove "Bearer " prefix

    user\_data = verify\_jwt\_token(token)

    user = db.query(User).filter(User.username == user\_data["sub"]).first()

    if not user:

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

    transactions = db.query(Transaction).filter(Transaction.user\_id == user.id).all()

    return templates.TemplateResponse("dashboard.html", {"request": request, "user": user, "transactions": transactions})

# ✅ Login Page

@router.get("/login")

def login\_page(request: Request):

    return templates.TemplateResponse("login.html", {"request": request})

# ✅ Handle Login & Set JWT Cookie

@router.post("/login")

def login(response: Response, username: str = Form(...), password: str = Form(...), db: Session = Depends(get\_db)):

    user = db.query(User).filter(User.username == username, User.password == password).first()

    if not user:

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

    token = create\_jwt\_token({"sub": user.username})

    response = RedirectResponse(url="/dashboard", status\_code=303)

    response.set\_cookie(key="access\_token", value=f"Bearer {token}", httponly=True)  # ✅ Store JWT in cookie

    return response

# ✅ Register Page

@router.get("/register")

def register\_page(request: Request):

    return templates.TemplateResponse("register.html", {"request": request})

# ✅ Handle Registration

@router.post("/register")

def register(response: Response, username: str = Form(...), password: str = Form(...), db: Session = Depends(get\_db)):

    existing\_user = db.query(User).filter(User.username == username).first()

    if existing\_user:

        raise HTTPException(status\_code=400, detail="Username already exists")

    new\_user = User(username=username, password=password, balance=0.0)

    db.add(new\_user)

    db.commit()

    return RedirectResponse(url="/login", status\_code=303)

# ✅ Logout (Clears JWT Cookie)

@router.get("/logout")

def logout():

    response = RedirectResponse(url="/login", status\_code=303)

    response.delete\_cookie("access\_token")  # ✅ Remove JWT cookie

    return response

# ✅ Deposit Money

@router.post("/deposit")

def deposit(response: Response, amount: float = Form(...), access\_token: str = Cookie(None), db: Session = Depends(get\_db)):

    if not access\_token:

        raise HTTPException(status\_code=401, detail="Not authenticated")

    token = access\_token.replace("Bearer ", "")

    user\_data = verify\_jwt\_token(token)

    user = db.query(User).filter(User.username == user\_data["sub"]).first()

    if not user:

        raise HTTPException(status\_code=401, detail="User not found")

    if amount <= 0:

        raise HTTPException(status\_code=400, detail="Deposit amount must be greater than zero")

    # ✅ Update Balance

    user.balance += amount

    # ✅ Record Transaction

    transaction = Transaction(user\_id=user.id, amount=amount, type="deposit")

    db.add(transaction)

    db.commit()

    return RedirectResponse(url="/dashboard", status\_code=303)

### 🔹 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)

    username = Column(String, unique=True, index=True)

    password = Column(String)

    balance = Column(Float, default=0.0)

class Transaction(Base):

    \_\_tablename\_\_ = "transactions"

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

    user\_id = Column(Integer, ForeignKey("users.id"))

    amount = Column(Float)

    type = Column(String)

    user = relationship("User")

### 🔹 auth.py

from fastapi import Depends, HTTPException

from fastapi.security import OAuth2PasswordBearer

from datetime import datetime, timedelta

import jwt

SECRET\_KEY = "secret"

ALGORITHM = "HS256"

oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="login")

def create\_jwt\_token(data: dict):

    expire = datetime.utcnow() + timedelta(hours=1)

    data.update({"exp": expire})

    return jwt.encode(data, SECRET\_KEY, algorithm=ALGORITHM)

def verify\_jwt\_token(token: str = Depends(oauth2\_scheme)):

    try:

        payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])

        return payload

    except jwt.ExpiredSignatureError:

        raise HTTPException(status\_code=401, detail="Token expired")

    except jwt.InvalidTokenError:

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

### 🔹 schemas.py

from pydantic import BaseModel

# ✅ User Schema for API Responses

class UserResponse(BaseModel):

    id: int

    name: str

    email: str

    class Config:

        from\_attributes = True  # ✅ Correct for Pydantic v2

# ✅ User Schema for API Requests (Registration)

class UserCreate(BaseModel):

    name: str

    email: str

    password: str

# ✅ Transaction Schema for API Responses

class TransactionResponse(BaseModel):

    id: int

    amount: float

    user\_id: int

    class Config:

        from\_attributes = True  # ✅ Correct for Pydantic v2

# ✅ Transaction Schema for Creating a New Transaction

class TransactionCreate(BaseModel):

    amount: float

    user\_id: int

### 🔹 database.py

from sqlalchemy import create\_engine

from sqlalchemy.orm import sessionmaker, declarative\_base

DATABASE\_URL = "sqlite:///./bank.db"

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

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

Base = declarative\_base()

def init\_db():

    from app.models import User, Transaction  # ✅ Import models

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

## 5. Template Files (Frontend)

### 🔹 base.html

<!DOCTYPE html>

<html>

<head>

    <title>{% block title %}Banking System{% endblock %}</title>

    <link rel="stylesheet" href="/static/style.css">

</head>

<body>

    <nav>

        <a href="/">Home</a> | <a href="/dashboard">Dashboard</a> | <a href="/login">Login</a> | <a href="/register">Register</a>

    </nav>

    <div>{% block content %}{% endblock %}</div>

</body>

</html>

### 🔹 index.html

{% extends "base.html" %}

{% block title %}Home{% endblock %}

{% block content %}

    <h2>Welcome to FastAPI Home Page</h2>

    <p>This is the main entry page.</p>

{% endblock %}

### 🔹 login.html

{% extends "base.html" %}

{% block title %}Login{% endblock %}

{% block content %}

    <h2>Login</h2>

    <form action="/login" method="post">

        <label for="username">Username:</label>

        <input type="text" id="username" name="username" required>

        <label for="password">Password:</label>

        <input type="password" id="password" name="password" required>

        <button type="submit">Login</button>

    </form>

{% endblock %}

### 🔹 register.html

{% extends "base.html" %}

{% block title %}Register{% endblock %}

{% block content %}

    <h2>Register</h2>

    <form action="/register" method="post">

        <label for="username">Username:</label>

        <input type="text" id="username" name="username" required>

        <label for="email">Email:</label>

        <input type="email" id="email" name="email" required>

        <label for="password">Password:</label>

        <input type="password" id="password" name="password" required>

        <button type="submit">Register</button>

    </form>

{% endblock %}

### 🔹 dashboard.html

{% extends "base.html" %}

{% block content %}

    <h2>Welcome to Your Dashboard</h2>

    <p>Hello, <strong>{{ user.username }}</strong>! Your balance is <strong>${{ user.balance }}</strong></p>

    <h3>Deposit Funds</h3>

    <form method="POST" action="/deposit">

        <input type="number" name="amount" step="0.01" placeholder="Enter amount">

        <button type="submit">Deposit</button>

    </form>

    <h3>Transaction History</h3>

    <table border="1">

        <tr>

            <th>Type</th>

            <th>Amount</th>

        </tr>

        {% for transaction in transactions %}

        <tr>

            <td>{{ transaction.type }}</td>

            <td>${{ transaction.amount }}</td>

        </tr>

        {% endfor %}

    </table>

    <br>

    <a href="/logout"><button>Logout</button></a>

{% endblock %}

### 🔹 static/style.css

body {

    font-family: Arial, sans-serif;

    text-align: center;

    background-color: #f8f9fa;

    margin: 50px;

}

h2 {

    color: #007bff;

}

## 6. README

### FastAPI Banking System  
\*\*Installation & Setup:\*\*  
1. Install dependencies: `pip install -r requirements.txt`  
2. Initialize database: `python -m app.database`  
3. Start server: `uvicorn app.main:app --reload`  
  
\*\*Endpoints:\*\*  
- `/register`: Register a new user  
- `/login`: User login  
- `/dashboard`: View balance & transactions (requires authentication)  
- `/deposit`: Deposit funds (requires authentication)  
- `/logout`: Logout user

Code Explanation

Main.py

| Line of Code | Explanation |
| --- | --- |

|  |  |
| --- | --- |
| from fastapi import FastAPI | This imports the FastAPI class, which is used to create a web application. |

|  |  |
| --- | --- |
| from fastapi.staticfiles import StaticFiles | This imports StaticFiles, which helps serve static files (e.g., CSS, images, JavaScript) in the app. |

|  |  |
| --- | --- |
| from app.routes import router | This imports the router from routes.py, where all API endpoints are defined. |

|  |  |
| --- | --- |
| from app.database import engine, Base | This imports engine (database connection) and Base (used for defining database tables) from database.py. |

|  |  |
| --- | --- |
| Base.metadata.create\_all(bind=engine) | This creates all tables in the database if they don't already exist. |

|  |  |
| --- | --- |
| app = FastAPI() | This initializes a FastAPI app instance, which will handle incoming web requests. |

|  |  |
| --- | --- |
| app.mount("/static", StaticFiles(directory="app/static"), name="static") | This tells FastAPI to serve static files from the app/static folder when a user accesses /static. |

|  |  |
| --- | --- |
| app.include\_router(router) | This adds all API endpoints from routes.py to the FastAPI application. |

|  |  |
| --- | --- |
| if \_\_name\_\_ == "\_\_main\_\_": | This ensures the application runs only when executed directly and not when imported as a module. |

|  |  |
| --- | --- |
| import uvicorn | This imports uvicorn, a server used to run FastAPI apps. |

|  |  |
| --- | --- |
| uvicorn.run("app.main:app", host="127.0.0.1", port=8000, reload=True) | This starts the FastAPI server on localhost (127.0.0.1) at port 8000 with auto-reload enabled (useful for development). |

\_\_init\_\_.py

(Nothing needed as a content, presence of file itself should sufficently work)

auth.py

| Line of Code | Explanation |
| --- | --- |

|  |  |
| --- | --- |
| from fastapi import Depends, HTTPException | Imports FastAPI dependencies: Depends is used for dependency injection, and HTTPException is used to raise errors when authentication fails. |

|  |  |
| --- | --- |
| from fastapi.security import OAuth2PasswordBearer | Imports OAuth2 scheme: OAuth2PasswordBearer is used to extract and validate the token from incoming requests. |

|  |  |
| --- | --- |
| from datetime import datetime, timedelta | Imports time-related functions: datetime handles current timestamps, and timedelta helps set token expiration time. |

|  |  |
| --- | --- |
| import jwt | Imports the PyJWT library to create and decode JSON Web Tokens (JWT). |

|  |  |
| --- | --- |
| SECRET\_KEY = "secret" | Defines the secret key used to sign JWT tokens. (In real applications, store this in an environment variable for security!) |

|  |  |
| --- | --- |
| ALGORITHM = "HS256" | Defines the encryption algorithm for signing the token. HS256 (HMAC with SHA-256) is commonly used for JWT authentication. |

|  |  |
| --- | --- |
| oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="login") | Creates an OAuth2 password bearer token scheme, where users authenticate using the /login endpoint to obtain a token. |

|  |  |
| --- | --- |
| def create\_jwt\_token(data: dict): | Defines a function to generate a JWT token using user data. |

|  |  |
| --- | --- |
| expire = datetime.utcnow() + timedelta(hours=1) | Sets token expiration: Adds 1 hour to the current time, ensuring the token expires after this duration. |

|  |  |
| --- | --- |
| data.update({"exp": expire}) | Adds expiration time (exp) to the token payload. |

|  |  |
| --- | --- |
| return jwt.encode(data, SECRET\_KEY, algorithm=ALGORITHM) | Encodes the JWT token using the secret key and algorithm, returning the signed token as a string. |

|  |  |
| --- | --- |
| def verify\_jwt\_token(token: str = Depends(oauth2\_scheme)): | Defines a function to verify the JWT token by extracting it from the request using Depends(oauth2\_scheme). |

|  |  |
| --- | --- |
| payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM]) | Decodes the token using the secret key and algorithm, extracting user information. |

|  |  |
| --- | --- |
| return payload | Returns the decoded payload if the token is valid. |

|  |  |
| --- | --- |
| except jwt.ExpiredSignatureError: | Handles expired tokens: If the token is expired, an exception is raised. |

|  |  |
| --- | --- |
| raise HTTPException(status\_code=401, detail="Token expired") | Raises a 401 Unauthorized error when the token has expired. |

|  |  |
| --- | --- |
| except jwt.InvalidTokenError: | Handles invalid tokens: If the token is malformed or invalid, an exception is raised. |

|  |  |
| --- | --- |
| raise HTTPException(status\_code=401, detail="Invalid token") | Raises a 401 Unauthorized error for invalid tokens. |

database.py

| Line of Code | Explanation |
| --- | --- |

|  |  |
| --- | --- |
| from sqlalchemy import create\_engine | Imports create\_engine, which is used to establish a connection with the database. |

|  |  |
| --- | --- |
| from sqlalchemy.orm import sessionmaker, declarative\_base | Imports SQLAlchemy ORM components: sessionmaker is used for managing database sessions, and declarative\_base is used to define database models. |

|  |  |
| --- | --- |
| DATABASE\_URL = "sqlite:///./bank.db" | Defines the database URL: Uses SQLite as the database engine and stores the database file as bank.db in the current directory. |

|  |  |
| --- | --- |
| engine = create\_engine(DATABASE\_URL, connect\_args={"check\_same\_thread": False}) | Creates a database engine that connects to bank.db. The check\_same\_thread=False argument allows multiple threads to use the same connection. |

|  |  |
| --- | --- |
| SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine) | Creates a session factory: Each time SessionLocal() is called, it creates a new database session. |

|  |  |
| --- | --- |
| Base = declarative\_base() | Creates a Base class: This is used to define database models by inheriting from Base. |

|  |  |
| --- | --- |
| def init\_db(): | Defines a function (init\_db) to initialize the database by creating required tables. |

|  |  |
| --- | --- |
| from app.models import User, Transaction | Imports database models (User & Transaction) inside the function to avoid circular imports. |

|  |  |
| --- | --- |
| Base.metadata.create\_all(bind=engine) | Creates all tables in the database based on the models defined in app.models. |

Models.py

| Line of Code | Explanation |
| --- | --- |

|  |  |
| --- | --- |
| from sqlalchemy import Column, Integer, String, Float, ForeignKey | Imports SQLAlchemy types used for defining database columns. |

|  |  |
| --- | --- |
| from sqlalchemy.orm import relationship | Imports relationship, which is used to define relationships between tables. |

|  |  |
| --- | --- |
| from app.database import Base | Imports Base from the database module to define models. |

|  |  |
| --- | --- |
| class User(Base): | Defines a User model that represents a users table in the database. |

|  |  |
| --- | --- |
| \_\_tablename\_\_ = "users" | Specifies the table name in the database as "users". |

|  |  |
| --- | --- |
| id = Column(Integer, primary\_key=True, index=True) | Defines id as the primary key (unique identifier) for the users table. |

|  |  |
| --- | --- |
| username = Column(String, unique=True, index=True) | Defines username as a string column, ensuring each username is unique and indexed for faster queries. |

|  |  |
| --- | --- |
| password = Column(String) | Defines password as a string column to store user passwords. (In real-world applications, store hashed passwords instead of plain text!) |

|  |  |
| --- | --- |
| balance = Column(Float, default=0.0) | Defines balance as a floating-point number, with a default value of 0.0. |

|  |  |
| --- | --- |
| class Transaction(Base): | Defines a Transaction model that represents a transactions table in the database. |

|  |  |
| --- | --- |
| \_\_tablename\_\_ = "transactions" | Specifies the table name in the database as "transactions". |

|  |  |
| --- | --- |
| id = Column(Integer, primary\_key=True, index=True) | Defines id as the primary key for the transactions table. |

|  |  |
| --- | --- |
| user\_id = Column(Integer, ForeignKey("users.id")) | Defines user\_id as a foreign key that links each transaction to a specific user in the users table. |

|  |  |
| --- | --- |
| amount = Column(Float) | Defines amount as a floating-point number representing the transaction amount. |

|  |  |
| --- | --- |
| type = Column(String) | Defines type as a string indicating the type of transaction (e.g., "deposit", "withdrawal"). |

|  |  |
| --- | --- |
| user = relationship("User") | Establishes a relationship between Transaction and User, allowing access to user details from a transaction record. |

Routes.py

| Line of Code | Explanation |
| --- | --- |

|  |  |
| --- | --- |
| from fastapi import APIRouter, Depends, HTTPException, Request, Form, Cookie, Response | Imports FastAPI modules for handling API routes, authentication, and request processing. |

|  |  |
| --- | --- |
| from sqlalchemy.orm import Session | Imports Session from SQLAlchemy to interact with the database. |

|  |  |
| --- | --- |
| from fastapi.templating import Jinja2Templates | Enables Jinja2 template rendering for HTML pages. |

|  |  |
| --- | --- |
| from fastapi.responses import RedirectResponse | Allows redirection after login, registration, or logout. |

|  |  |
| --- | --- |
| from app.database import SessionLocal | Imports the database session to interact with the SQLite database. |

|  |  |
| --- | --- |
| from app.models import User, Transaction | Imports User and Transaction models. |

|  |  |
| --- | --- |
| from app.auth import create\_jwt\_token, verify\_jwt\_token | Imports JWT authentication functions for user authentication. |

|  |  |
| --- | --- |
| router = APIRouter() | Creates a FastAPI router to define multiple endpoints. |

|  |  |
| --- | --- |
| templates = Jinja2Templates(directory="app/templates") | Specifies the folder where HTML templates are stored. |

|  |  |
| --- | --- |
| def get\_db(): | Defines a function to create and close a database session. |

|  |  |
| --- | --- |
| db = SessionLocal() | Creates a new database session. |

|  |  |
| --- | --- |
| yield db | Allows the session to be used in API calls before closing it. |

|  |  |
| --- | --- |
| db.close() | Closes the session when the request is completed. |

|  |  |
| --- | --- |
| @router.get("/") | Defines a route for the home page. |

|  |  |
| --- | --- |
| def home(request: Request): | Receives an HTTP request. |

|  |  |
| --- | --- |
| return templates.TemplateResponse("index.html", {"request": request}) | Renders index.html and passes the request object. |

|  |  |
| --- | --- |
| @router.get("/dashboard") | Defines a route for the user dashboard. |

|  |  |
| --- | --- |
| def dashboard(request: Request, access\_token: str = Cookie(None), db: Session = Depends(get\_db)): | Requires an authentication token stored in cookies. |

|  |  |
| --- | --- |
| if not access\_token: | Checks if the user is logged in. |

|  |  |
| --- | --- |
| raise HTTPException(status\_code=401, detail="Not authenticated") | Returns an error if the user is not authenticated. |

|  |  |
| --- | --- |
| token = access\_token.replace("Bearer ", "") | Removes "Bearer " from the token. |

|  |  |
| --- | --- |
| user\_data = verify\_jwt\_token(token) | Verifies and decodes the JWT token. |

|  |  |
| --- | --- |
| user = db.query(User).filter(User.username == user\_data["sub"]).first() | Fetches user details from the database. |

|  |  |
| --- | --- |
| transactions = db.query(Transaction).filter(Transaction.user\_id == user.id).all() | Fetches transaction history for the logged-in user. |

|  |  |
| --- | --- |
| return templates.TemplateResponse("dashboard.html", {"request": request, "user": user, "transactions": transactions}) | Renders the dashboard.html page with user and transaction data. |

|  |  |
| --- | --- |
| @router.get("/login") | Defines a route to display the login page. |

|  |  |
| --- | --- |
| return templates.TemplateResponse("login.html", {"request": request}) | Renders login.html. |

|  |  |
| --- | --- |
| @router.post("/login") | Defines a route to handle login form submission. |

|  |  |
| --- | --- |
| def login(response: Response, username: str = Form(...), password: str = Form(...), db: Session = Depends(get\_db)): | Accepts username and password from the login form. |

|  |  |
| --- | --- |
| user = db.query(User).filter(User.username == username, User.password == password).first() | Checks if the user exists in the database. |

|  |  |
| --- | --- |
| if not user: | If the user does not exist, raise an error. |

|  |  |
| --- | --- |
| raise HTTPException(status\_code=401, detail="Invalid credentials") | Returns an error for incorrect login details. |

|  |  |
| --- | --- |
| token = create\_jwt\_token({"sub": user.username}) | Generates a JWT token. |

|  |  |
| --- | --- |
| response = RedirectResponse(url="/dashboard", status\_code=303) | Redirects the user to the dashboard after login. |

|  |  |
| --- | --- |
| response.set\_cookie(key="access\_token", value=f"Bearer {token}", httponly=True) | Stores the JWT token in cookies. |

|  |  |
| --- | --- |
| return response | Returns the response object. |

|  |  |
| --- | --- |
| @router.get("/register") | Defines a route for the registration page. |

|  |  |
| --- | --- |
| return templates.TemplateResponse("register.html", {"request": request}) | Renders register.html. |

|  |  |
| --- | --- |
| @router.post("/register") | Defines a route to handle user registration. |

|  |  |
| --- | --- |
| def register(response: Response, username: str = Form(...), password: str = Form(...), db: Session = Depends(get\_db)): | Accepts username and password from the registration form. |

|  |  |
| --- | --- |
| existing\_user = db.query(User).filter(User.username == username).first() | Checks if the username already exists. |

|  |  |
| --- | --- |
| if existing\_user: | If the user already exists, raise an error. |

|  |  |
| --- | --- |
| raise HTTPException(status\_code=400, detail="Username already exists") | Returns an error if the username is taken. |

|  |  |
| --- | --- |
| new\_user = User(username=username, password=password, balance=0.0) | Creates a new user with a balance of 0.0. |

|  |  |
| --- | --- |
| db.add(new\_user) | Adds the new user to the database. |

|  |  |
| --- | --- |
| db.commit() | Saves the changes. |

|  |  |
| --- | --- |
| return RedirectResponse(url="/login", status\_code=303) | Redirects to the login page after successful registration. |

|  |  |
| --- | --- |
| @router.get("/logout") | Defines a route to log out the user. |

|  |  |
| --- | --- |
| response = RedirectResponse(url="/login", status\_code=303) | Redirects the user to the login page. |

|  |  |
| --- | --- |
| response.delete\_cookie("access\_token") | Removes the JWT token from cookies. |

|  |  |
| --- | --- |
| return response | Returns the response object. |

|  |  |
| --- | --- |
| @router.post("/deposit") | Defines a route for depositing money. |

|  |  |
| --- | --- |
| def deposit(response: Response, amount: float = Form(...), access\_token: str = Cookie(None), db: Session = Depends(get\_db)): | Accepts deposit amount from the form. |

|  |  |
| --- | --- |
| if not access\_token: | Checks if the user is logged in. |

|  |  |
| --- | --- |
| raise HTTPException(status\_code=401, detail="Not authenticated") | Returns an error if the user is not authenticated. |

|  |  |
| --- | --- |
| token = access\_token.replace("Bearer ", "") | Removes "Bearer " from the token. |

|  |  |
| --- | --- |
| user\_data = verify\_jwt\_token(token) | Verifies the JWT token. |

|  |  |
| --- | --- |
| user = db.query(User).filter(User.username == user\_data["sub"]).first() | Fetches the logged-in user. |

|  |  |
| --- | --- |
| if not user: | If the user does not exist, raise an error. |

|  |  |
| --- | --- |
| raise HTTPException(status\_code=401, detail="User not found") | Returns an error if the user is not found. |

|  |  |
| --- | --- |
| if amount <= 0: | Checks if the deposit amount is valid. |

|  |  |
| --- | --- |
| raise HTTPException(status\_code=400, detail="Deposit amount must be greater than zero") | Returns an error if the deposit amount is invalid. |

|  |  |
| --- | --- |
| user.balance += amount | Updates the user's balance. |

|  |  |
| --- | --- |
| transaction = Transaction(user\_id=user.id, amount=amount, type="deposit") | Creates a new deposit transaction. |

|  |  |
| --- | --- |
| db.add(transaction) | Adds the transaction to the database. |

|  |  |
| --- | --- |
| db.commit() | Saves the transaction. |

|  |  |
| --- | --- |
| return RedirectResponse(url="/dashboard", status\_code=303) | Redirects the user to the dashboard after deposit. |