Final Capstone Project

1. Project Overview

Build a complete **Personal Finance Manager** that combines the main skills learned throughout the Python project series.

The application includes:

- User registration and login
- JWT authentication
- Password hashing
- Expense management
- Income management
- Budget tracking
- Financial summaries
- SQLite database
- REST API
- HTML/CSS/JavaScript frontend

FastAPI provides built in security utilities for OAuth2 and Bearer authentication, and its documentation demonstrates JWT based authentication with secure password hashing.

SQLite is suitable for this capstone because it is lightweight, file based, and supported directly by Python.

2. Features

Authentication

- Register
- Login
- Password hashing

- JWT access tokens
- Protected API endpoints

Finance

- Add income
- Add expenses
- View transactions
- Delete transactions
- Filter by type
- Calculate balance
- Calculate total income
- Calculate total expenses

Budget

- Set monthly budget
- View current budget
- Compare spending against budget

Frontend

- Dashboard
- Login form
- Registration form
- Transaction form
- Transaction list
- Financial summary
- Logout

3. Technologies

- Python 3

- FastAPI
- Pydantic
- SQLite
- PyJWT
- pwdlib
- Uvicorn
- HTML
- CSS
- JavaScript

FastAPI's own full stack template uses concepts including FastAPI, SQL databases, authentication, JWT, frontend integration and testing, making these appropriate technologies for a larger full stack Python project.

4. Folder Structure

```
personal_finance_manager/
□□□ main.py
□□□ database.py
□□□ auth.py
□□□ schemas.py
□□□ requirements.txt
□□□ README.md
□□□ templates/
□   □□□ login.html
□   □□□ dashboard.html
□□□ static/
    □□□ style.css
    □□□ app.js
```

5. How the Project Works

```
User
 □
```

Frontend

FastAPI

Authentication

JWT Token

Protected API

SQLite Database

JSON Response

Frontend Dashboard

After login:

Username + Password

Verify Account

Create JWT

Frontend

Authorization: Bearer TOKEN

Protected Endpoints

6. Database Design

The application uses three tables:

```
users
□□□ id
□□□ username
□□□ password_hash

transactions
□□□ id
□□□ user_id
□□□ title
□□□ amount
□□□ type
□□□ category
□□□ date

budgets
□□□ id
□□□ user_id
□□□ amount
□□□ month
```

7. Database Code

database.py

```
import sqlite3


class Database:

    def __init__(self, database_name="finance.db"):
        self.database_name = database_name
        self.create_tables()

    def connect(self):
        return sqlite3.connect(self.database_name)
```

```
  def create_tables(self):
      connection = self.connect()
      cursor = connection.cursor()

      cursor.execute("""
          CREATE TABLE IF NOT EXISTS users (
              id INTEGER PRIMARY KEY AUTOINCREMENT,
              username TEXT UNIQUE NOT NULL,
              password_hash TEXT NOT NULL
          )
      """)

      cursor.execute("""
          CREATE TABLE IF NOT EXISTS transactions (
              id INTEGER PRIMARY KEY AUTOINCREMENT,
              user_id INTEGER NOT NULL,
              title TEXT NOT NULL,
              amount REAL NOT NULL,
              type TEXT NOT NULL,
              category TEXT NOT NULL,
              date TEXT NOT NULL,
              FOREIGN KEY (user_id)
                  REFERENCES users(id)
          )
      """)

      cursor.execute("""
          CREATE TABLE IF NOT EXISTS budgets (
              id INTEGER PRIMARY KEY AUTOINCREMENT,
              user_id INTEGER NOT NULL,
              amount REAL NOT NULL,
              month TEXT NOT NULL,
              UNIQUE(user_id, month),
```

```
            FOREIGN KEY (user_id)
                REFERENCES users(id)
        )
    """)

    connection.commit()
    connection.close()

def create_user(self, username, password_hash):
    connection = self.connect()
    cursor = connection.cursor()

    cursor.execute("""
        INSERT INTO users
        (username, password_hash)
        VALUES (?, ?)
    """, (username, password_hash))

    connection.commit()
    user_id = cursor.lastrowid
    connection.close()

    return user_id

def get_user(self, username):
    connection = self.connect()
    connection.row_factory = sqlite3.Row
    cursor = connection.cursor()

    cursor.execute("""
        SELECT id, username, password_hash
        FROM users
        WHERE username = ?
    """, (username,))
```

```
        user = cursor.fetchone()
        connection.close()

        return dict(user) if user else None

    def add_transaction(
        self,
        user_id,
        title,
        amount,
        transaction_type,
        category,
        date
    ):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            INSERT INTO transactions
            (user_id, title, amount, type, category, date)
            VALUES (?, ?, ?, ?, ?, ?)
        """, (
            user_id,
            title,
            amount,
            transaction_type,
            category,
            date
        ))

        connection.commit()
        transaction_id = cursor.lastrowid
        connection.close()
```

```
        return transaction_id

    def get_transactions(self, user_id):
        connection = self.connect()
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()

        cursor.execute("""
            SELECT id, title, amount, type, category, date
            FROM transactions
            WHERE user_id = ?
            ORDER BY id DESC
        """, (user_id,))

        transactions = [
            dict(row)
            for row in cursor.fetchall()
        ]

        connection.close()

        return transactions

    def delete_transaction(self, user_id, transaction_id):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            DELETE FROM transactions
            WHERE id = ? AND user_id = ?
        """, (transaction_id, user_id))

        connection.commit()
```

```
        deleted = cursor.rowcount > 0
        connection.close()

        return deleted

    def get_summary(self, user_id):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            SELECT
                COALESCE(
                    SUM(
                        CASE
                            WHEN type = 'income'
                            THEN amount
                            ELSE 0
                        END
                    ), 0
                ),
                COALESCE(
                    SUM(
                        CASE
                            WHEN type = 'expense'
                            THEN amount
                            ELSE 0
                        END
                    ), 0
                )
            FROM transactions
            WHERE user_id = ?
        """, (user_id,))

        income, expenses = cursor.fetchone()
```

```
        connection.close()

        return {
            "income": income,
            "expenses": expenses,
            "balance": income - expenses
        }

    def set_budget(self, user_id, amount, month):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            INSERT INTO budgets
            (user_id, amount, month)
            VALUES (?, ?, ?)
            ON CONFLICT(user_id, month)
            DO UPDATE SET amount = excluded.amount
        """, (user_id, amount, month))

        connection.commit()
        connection.close()

    def get_budget(self, user_id, month):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            SELECT amount
            FROM budgets
            WHERE user_id = ? AND month = ?
        """, (user_id, month))
```

```
        result = cursor.fetchone()
        connection.close()

        return result[0] if result else 0
```

8. Authentication Code

auth.py

```
from datetime import datetime, timedelta, timezone

import jwt
from pwdlib import PasswordHash


SECRET_KEY = "change-this-secret-key"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 60


password_hash = PasswordHash.recommended()


def hash_password(password):
    return password_hash.hash(password)


def verify_password(password, hashed_password):
    return password_hash.verify(
        password,
        hashed_password
    )


def create_access_token(user_id):
```

```
    expires = datetime.now(
        timezone.utc
    ) + timedelta(
        minutes=ACCESS_TOKEN_EXPIRE_MINUTES
    )

    payload = {
        "sub": str(user_id),
        "exp": expires
    }

    return jwt.encode(
        payload,
        SECRET_KEY,
        algorithm=ALGORITHM
    )


def decode_token(token):
    try:
        return jwt.decode(
            token,
            SECRET_KEY,
            algorithms=[ALGORITHM]
        )
    except jwt.InvalidTokenError:
        return None
```

For a real application, the secret key should be generated securely and stored outside the source code, such as an environment variable. FastAPI's JWT guidance also recommends using a secure random secret key.

9. Schemas

schemas.py

```
from pydantic import BaseModel, Field


class RegisterRequest(BaseModel):
    username: str = Field(
        min_length=3,
        max_length=50
    )

    password: str = Field(
        min_length=8,
        max_length=100
    )


class LoginRequest(BaseModel):
    username: str
    password: str


class TransactionCreate(BaseModel):
    title: str = Field(
        min_length=1,
        max_length=200
    )

    amount: float = Field(gt=0)

    type: str

    category: str = Field(
        min_length=1,
        max_length=100
    )
```

```
    date: str


class BudgetCreate(BaseModel):
    amount: float = Field(gt=0)
    month: str
```

10. Complete API

main.py

```
from fastapi import (
    Depends,
    FastAPI,
    HTTPException
)
from fastapi.security import OAuth2PasswordBearer

from auth import (
    create_access_token,
    decode_token,
    hash_password,
    verify_password
)

from database import Database

from schemas import (
    BudgetCreate,
    LoginRequest,
    RegisterRequest,
    TransactionCreate
)
```

```
app = FastAPI(
    title="Personal Finance Manager",
    description="Full stack Python finance application.",
    version="1.0.0"
)

database = Database()

oauth2_scheme = OAuth2PasswordBearer(
    tokenUrl="/login"
)


def get_current_user(
    token: str = Depends(oauth2_scheme)
):
    payload = decode_token(token)

    if not payload:
        raise HTTPException(
            status_code=401,
            detail="Invalid or expired token."
        )

    user_id = payload.get("sub")

    if not user_id:
        raise HTTPException(
            status_code=401,
            detail="Invalid token."
        )

    return int(user_id)
```

```
@app.get("/")
def home():
    return {
        "message": "Personal Finance Manager is running"
    }


@app.post("/register")
def register(data: RegisterRequest):

    if database.get_user(data.username):
        raise HTTPException(
            status_code=400,
            detail="Username already exists."
        )

    user_id = database.create_user(
        data.username,
        hash_password(data.password)
    )

    return {
        "message": "Account created successfully.",
        "user_id": user_id
    }


@app.post("/login")
def login(data: LoginRequest):

    user = database.get_user(data.username)
```

```
    if not user or not verify_password(
        data.password,
        user["password_hash"]
    ):
        raise HTTPException(
            status_code=401,
            detail="Invalid username or password."
        )

    token = create_access_token(
        user["id"]
    )

    return {
        "access_token": token,
        "token_type": "bearer"
    }


@app.post("/transactions")
def add_transaction(
    data: TransactionCreate,
    user_id: int = Depends(get_current_user)
):

    if data.type not in {"income", "expense"}:
        raise HTTPException(
            status_code=400,
            detail="Type must be income or expense."
        )

    transaction_id = database.add_transaction(
        user_id,
        data.title,
```

```
        data.amount,
        data.type,
        data.category,
        data.date
    )

    return {
        "id": transaction_id,
        "message": "Transaction added successfully."
    }


@app.get("/transactions")
def get_transactions(
    user_id: int = Depends(get_current_user)
):
    return database.get_transactions(user_id)


@app.delete("/transactions/{transaction_id}")
def delete_transaction(
    transaction_id: int,
    user_id: int = Depends(get_current_user)
):

    deleted = database.delete_transaction(
        user_id,
        transaction_id
    )

    if not deleted:
        raise HTTPException(
            status_code=404,
            detail="Transaction not found."
```

```
        )

    return {
        "message": "Transaction deleted successfully."
    }


@app.get("/summary")
def summary(
    user_id: int = Depends(get_current_user)
):
    return database.get_summary(user_id)


@app.post("/budget")
def set_budget(
    data: BudgetCreate,
    user_id: int = Depends(get_current_user)
):

    database.set_budget(
        user_id,
        data.amount,
        data.month
    )

    return {
        "message": "Budget saved successfully."
    }


@app.get("/budget/{month}")
def get_budget(
    month: str,
```

```
    user_id: int = Depends(get_current_user)
):

    return {
        "month": month,
        "budget": database.get_budget(
            user_id,
            month
        )
    }
```

FastAPI's `OAuth2PasswordBearer` integrates Bearer token authentication into the API and its OpenAPI documentation.

11. API Endpoints

```
POST    /register
POST    /login

POST    /transactions
GET     /transactions
DELETE  /transactions/{transaction_id}

GET     /summary

POST    /budget
GET     /budget/{month}
```

Protected endpoints require:

```
Authorization: Bearer YOUR_ACCESS_TOKEN
```

12. Frontend

templates/login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta
    name="viewport"
    content="width=device-width, initial-scale=1.0"
  >
  <title>Finance Manager</title>

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

<body>

<div class="container">

  <div class="card">

    <h1>Finance Manager</h1>

    <p>Login to manage your finances.</p>

    <form id="login-form">

      <input
        id="username"
        placeholder="Username"
        required
      >
```

```
        <input
          id="password"
          type="password"
          placeholder="Password"
          required
        >

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

      </form>

      <p id="message"></p>

    </div>

</div>

<script src="/static/app.js"></script>

</body>
</html>
```

templates/dashboard.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">

  <meta
    name="viewport"
    content="width=device-width, initial-scale=1.0"
  >
```

```
    <title>Finance Dashboard</title>

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

<body>

<div class="container">

    <div class="card">

        <h1>Finance Dashboard</h1>

        <div class="summary">

            <div>
                <span>Income</span>
                <strong id="income">0</strong>
            </div>

            <div>
                <span>Expenses</span>
                <strong id="expenses">0</strong>
            </div>

            <div>
                <span>Balance</span>
                <strong id="balance">0</strong>
            </div>
```

```
    </div>

    <form id="transaction-form">

      <input
        id="title"
        placeholder="Title"
        required
      >

      <input
        id="amount"
        type="number"
        step="0.01"
        placeholder="Amount"
        required
      >

      <select id="type">

        <option value="expense">
          Expense
        </option>

        <option value="income">
          Income
        </option>

      </select>

      <input
        id="category"
        placeholder="Category"
        required
```

```
        >

        <input
          id="date"
          type="date"
          required
        >

        <button type="submit">
          Add Transaction
        </button>

      </form>

      <h2>Transactions</h2>

      <ul id="transactions"></ul>

      <button id="logout">
        Logout
      </button>

    </div>

</div>

<script src="/static/app.js"></script>

</body>
</html>
```

13. JavaScript

static/app.js

```
const token = localStorage.getItem("token");


async function login(event) {

    event.preventDefault();

    const username =
        document.getElementById("username").value;

    const password =
        document.getElementById("password").value;

    const response = await fetch("/login", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            username,
            password
        })
    });

    const data = await response.json();

    if (!response.ok) {
        document.getElementById("message")
            .textContent = data.detail;

        return;
    }

    localStorage.setItem(
```

```
        "token",
        data.access_token
    );

    window.location.href = "/dashboard";
}


async function loadSummary() {

    const response = await fetch("/summary", {
        headers: {
            Authorization: `Bearer ${token}`
        }
    });

    if (!response.ok) {
        window.location.href = "/";
        return;
    }

    const data = await response.json();

    document.getElementById("income")
        .textContent = data.income.toFixed(2);

    document.getElementById("expenses")
        .textContent = data.expenses.toFixed(2);

    document.getElementById("balance")
        .textContent = data.balance.toFixed(2);
}
```

```
async function loadTransactions() {

  const response = await fetch("/transactions", {
    headers: {
      Authorization: `Bearer ${token}`
    }
  });

  const transactions = await response.json();

  const list =
    document.getElementById("transactions");

  list.innerHTML = "";

  transactions.forEach(transaction => {

    const item =
      document.createElement("li");

    item.className = "transaction";

    item.innerHTML = `
      <span>
        ${transaction.title}
        <small>
          ${transaction.category}
          • ${transaction.date}
        </small>
      </span>

      <strong>
        ${transaction.type === "income" ? "+" : "-"}
        ${transaction.amount}
```

```
            </strong>

            <button
              onclick="deleteTransaction(
                ${transaction.id}
              )"
            >
              Delete
            </button>
          `;

          list.appendChild(item);
        });
}


async function addTransaction(event) {

    event.preventDefault();

    const data = {
        title: document.getElementById("title").value,
        amount: Number(
            document.getElementById("amount").value
        ),
        type: document.getElementById("type").value,
        category:
            document.getElementById("category").value,
        date: document.getElementById("date").value
    };

    await fetch("/transactions", {
        method: "POST",
        headers: {
```

```
            "Content-Type": "application/json",
            Authorization: `Bearer ${token}`
        },
        body: JSON.stringify(data)
    });

    event.target.reset();

    await loadSummary();
    await loadTransactions();
}


async function deleteTransaction(id) {

    await fetch(`/transactions/${id}`, {
        method: "DELETE",
        headers: {
            Authorization: `Bearer ${token}`
        }
    });

    await loadSummary();
    await loadTransactions();
}


const loginForm =
    document.getElementById("login-form");

if (loginForm) {
    loginForm.addEventListener(
        "submit",
        login
```

```
    );
}


const transactionForm =
    document.getElementById("transaction-form");

if (transactionForm) {

    transactionForm.addEventListener(
        "submit",
        addTransaction
    );

    loadSummary();
    loadTransactions();
}


const logout =
    document.getElementById("logout");

if (logout) {

    logout.addEventListener(
        "click",
        () => {
            localStorage.removeItem("token");
            window.location.href = "/";
        }
    );
}
```

14. CSS

static/style.css

```
* {
    box-sizing: border-box;
}

body {
    margin: 0;
    font-family: Arial, sans-serif;
    background: #0f172a;
    color: #f8fafc;
}

.container {
    max-width: 900px;
    margin: auto;
    padding: 50px 20px;
}

.card {
    background: #1e293b;
    padding: 30px;
    border-radius: 16px;
}

h1 {
    margin-top: 0;
}

form {
    display: grid;
    gap: 12px;
    margin: 25px 0;
}
```

```
input,
select,
button {
    padding: 12px;
    border-radius: 8px;
    border: none;
}

input,
select {
    background: #0f172a;
    color: white;
    border: 1px solid #475569;
}

button {
    cursor: pointer;
    background: #38bdf8;
    color: #082f49;
    font-weight: bold;
}

.summary {
    display: grid;
    grid-template-columns:
        repeat(3, 1fr);
    gap: 15px;
}

.summary div {
    padding: 20px;
    background: #0f172a;
    border-radius: 10px;
```

```
}

.summary span {
    display: block;
    color: #94a3b8;
    margin-bottom: 8px;
}

.summary strong {
    font-size: 24px;
}

#transactions {
    list-style: none;
    padding: 0;
}

.transaction {
    display: flex;
    align-items: center;
    justify-content: space-between;
    gap: 15px;
    padding: 15px;
    margin-bottom: 10px;
    background: #0f172a;
    border-radius: 8px;
}

.transaction small {
    display: block;
    color: #94a3b8;
    margin-top: 5px;
}
```

```
.transaction button {
    background: #ef4444;
    color: white;
}

@media (max-width: 650px) {

    .summary {
        grid-template-columns: 1fr;
    }

    .transaction {
        flex-direction: column;
        align-items: flex-start;
    }
}
```

15. Dashboard Route

Add these imports to `main.py`:

```
from pathlib import Path

from fastapi.responses import HTMLResponse
from fastapi.staticfiles import StaticFiles
from fastapi.templating import Jinja2Templates
from fastapi import Request
```

Then add:

```
BASE_DIR = Path(__file__).resolve().parent

app.mount(
    "/static",
```

```
    StaticFiles(directory=BASE_DIR / "static"),
    name="static"
)

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


@app.get(
    "/dashboard",
    response_class=HTMLResponse
)
def dashboard(request: Request):
    return templates.TemplateResponse(
        request=request,
        name="dashboard.html"
    )
```

16. Requirements

requirements.txt

```
fastapi
uvicorn
jinja2
pyjwt
pwdlib[argon2]
python-multipart
```

17. How to Run

Create a virtual environment:

```
python -m venv venv
```

Activate it on Windows:

```
venv\Scripts\activate
```

Install dependencies:

```
pip install -r requirements.txt
```

Run the application:

```
uvicorn main:app --reload
```

Open:

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

API documentation:

```
http://127.0.0.1:8000/docs
```

18. Basic Testing

Test the complete application in this order:

```
1. Register a new account
2. Login
3. Receive JWT token
4. Add an income
5. Add an expense
6. View transactions
7. Check financial summary
8. Delete an expense
9. Set a monthly budget
```

10. Check the budget
11. Logout
12. Try accessing protected endpoints without a token

Expected summary:

```
{
   "income": 100000,
   "expenses": 35000,
   "balance": 65000
}
```

19. Final Capstone Skills

This project combines:

Python

OOP

File / Project Structure

SQLite

REST APIs

FastAPI

Pydantic

Authentication

Password Hashing

JWT

HTML

CSS

JavaScript

Frontend + Backend Integration

20. Production Improvements

This project is an educational capstone. Before production use, improve:

- Move secrets to environment variables
- Use PostgreSQL instead of SQLite
- Add database migrations
- Add automated tests
- Add refresh tokens
- Add stronger authorization
- Add rate limiting
- Add HTTPS
- Add CSRF protection where applicable
- Validate all user input
- Add proper error logging
- Containerize with Docker
- Add CI/CD
- Deploy backend and frontend separately when appropriate

FastAPI's production-oriented documentation covers security, OAuth2, JWT and database integration as separate areas that can be expanded from this foundation.

Visit haas.dev for more resources and guides.