Python 1. beadandó feladat

ELTE IK PNYFT – 2025-26 1

REST API készítése FastAPI alapokon

A beadandó feladatban egy team által implementált program kiegészítését kell elvégeznünk. Az alkalmazás már majdnem futáskész, de egy-két funkció nincs implementálva és a felhasználó kezelést át kell alakítani a lentebb megfogalmazott feltételek alapján.

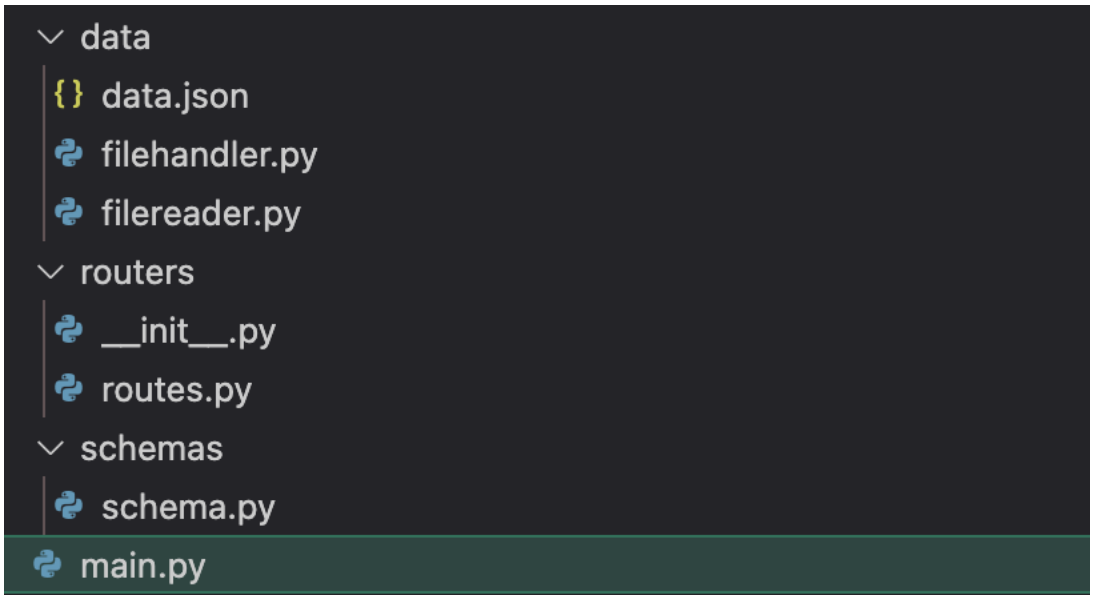
Az API egyes részei már elkészültek, vagyis a feladat, amit meg kell oldanunk, egy FastAPI implementációjának a kiegészítése. Az eddig elkészült részeket mellékeljük a feladathoz. A kész kódokat tanulmányozni kell ahhoz, hogy a maradék részeket el lehessen készíteni. (A csatolt fájlokban instrukciók vannak az adott modul funkcióinak a megírásához és véglegesítéséhez.)

Az API egy webbolt felhasználóinak és vásárlásainak a kezelését teszi lehetővé. Tartalmazza a felhasználók bolthoz rendelését, a kosár hozzáadását egy adott vásárlóhoz, valamint a vásárló kosarába tudunk helyezni termékeket a végpontok segítségével. A vásárló kosarába helyezett termékeket lehet módosítani vagy törölni. A megvásárolt tételek összeértékét, vagyis a fizetendő összeget is le lehet kérdezni egy adott vásárló kosarában található termékek árai alapján. Az API lehetőséget biztosít egy adott vásárló adatainak a lekérdezésére is, valamint az összes vásárló, vagy egy adott vásárlói kosár tartalmának a megjelenítésére.

Az API ezekkel a funkciókkal segítségünkre lehet egy webbolt elkészítéséhez, de a felhasználói felületet nem tartalmazza. A program a felhasználói adatokat a users.json, a kosár adatokat a data.json nevű fájlban tárolja (a csatolt megvalósításban ez nem így van implementálva, vagyis ezzel dolgunk lesz a feladat megoldása során).

A rendszert localhost-on kell futtatni virtual environment használatával (uv, vagy venv).

A mellékelt fájlok tartalma



- A data.json a JSON az adatok tárolására szolgál. Ebben elhelyeztünk néhány teszt adatot, amelyeket fel lehet használni az alkalmazás kipróbálásához (jelenleg a felhasználók kezelés is itt van).
- A filehandler.py és a filereader.py a JSON adatok fájlból történő beolvasására és fájlba írására használható, de csak a data.json írásának lehetőségeit kell, hogy tartalmazza, vagyis a users.json fájl elkészítése fontos feladatunk. A függvényeket ki kell egészíteni ahhoz, hogy képesek legyenek adatokat írni és olvasni, valamint a user funkciókat teljesen külön kell választani. A szétválasztáshoz használhatunk egy másik, a user adatokat tartalmazó fájl írására és olvasására alkalmas modult, de implementálhatjuk a jelenlegi modulokban is az új funkciókat.
- A routes.py tartalmazza az alkalmazás végpontjait. Ezeket a függvényeket kell kiegészíteni a megfelelő funkciókkal, valamint kiegészíteni új endpointokkal, ha erre szükség van.
- A schemas.py a JSON adatok kliens és szerver közötti protokollját, valamint a JSON adatok kezelésére alkalmas osztályokat tartalmazza. Ezeket az osztályokat ki kell egészíteni a megfelelő mezőkkel a JOSN adatok alapján. Ügyelni kell az adatok helyességére is a fájlban és a feladatban leírtak alapján (lásd. később).
- A main.py tartalmazza a FastAPI main modulját, amelyet futtatni kell az alkalmazás elindításához a uvicorn python csomag segítségével a futtatáshoz készített „venv” környezetben.

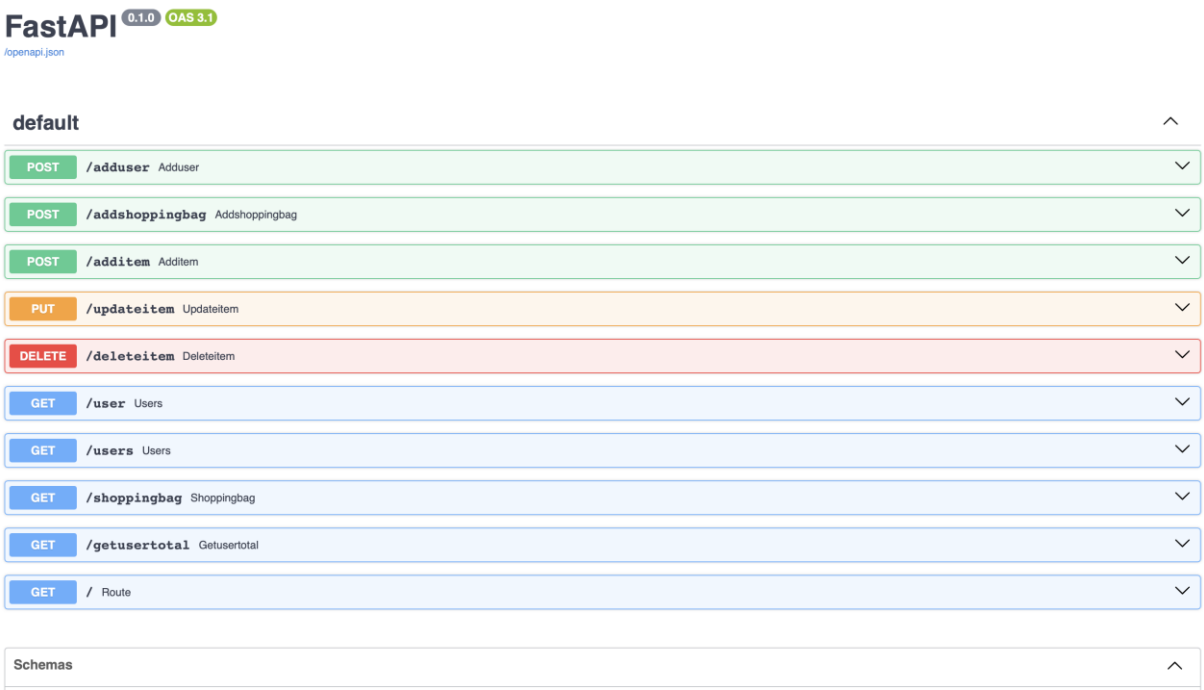
Feladatok

Indítsuk el virtual environment-ben (venv, uv) a REST API-t és nézzük meg az endpointok listáját a FastAPI általautomatikusan generált UI felületén. Szerverként használjuk a uvicorn csomagot. Az API elindításakor a rendszer generálni fog egy URL-t és egy portot rendel a futó programhoz. Ennek az URL-nek a kiegészítésével elérjük az alkalmazás /doc endpointjával a webes felületet, de használhatunk postman-t, vagy más API tesztelő eszközt is.

uv run uvicorn (mainfileneve:appneve) --reload --port (port)

(5 pont)

Az alábbi képernyőhöz hasonló képet kell látnunk a webböngészőben:



- Az itt látható végpontokat kell elkészítenünk és kiegészítenünk, valamint igény esetén újakat készítenünk.
- Az adduser segítségével egy felhasználót tudunk hozzáadni a webshophoz. Ehhez a funkcióhoz egy token segítségével lehet hozzáférni, amit a http header-ben kell küldeni. A token egy egyszerű string, amit a szerver oldalon tárolunk egy fájlban, és ezzel hasonlítjuk össze az adott endpointon megadott tokent. (Nem kell token kezelőt alkalmazni, csak egy egyszerű string-et eltárolni és azt várni az endpointokon.)
  (10 pont)
- A user.json fájl elkészítése és megfelelő funkciók és a jogosultság implementálása. **(5 pont)**
- Felhasználói funkciók:
  - Az additem egy terméket ad az adott felhasználó kosarához. **(5 pont)**
  - Az updateitem módosítja az adott termék attribútumait egy felhasználó kosarában. **(5 pont)**
  - A deleteitem töröl egy terméket a kosárból. készítsünk egy olyan változatot is, ami az összes terméket kiüríti a kosárból (Ez a kibővített /deletall ednpoint adminisztrátori funkció lesz!). Ez a funkció a vásárlás megvalósításához is jól jöhet, de ezt a funkciót nem nekünk kell elkészíteni, hanem a team egy másik tagjának, vagyis ezzel nincs dolgunk. **(5 pont)**

    - A shoppingbag egy kosárban található összes termék adatait adja vissza. **(5 pont)**
    - A getusertotal egy vásárló kosarában található termékek értékét adja vissza. **(5 pont)**
- Adminisztrátori funkciók, amelyek csak a mentett token küldésével érhetők el:
    - A user egy adott felhasználó adatait adja vissza. **(5 pont)**
    - A users visszaadja az összes felhasználót. **(5 pont)**
    - Az adduser egy felhasználót visz fel a rendszerbe **(5 pont)**
    - Az addshoppingbag egy kosarat rendel egy felhasználóhoz. **(5 pont)**
    - A kész rendszert egészítsük ki egy olyan funkcióval is, ami törli a megadott felhasználót, de csak akkor, ha annak a kosara üres. **(5 pont)**
- A Route az üzlet nevét adja vissza. Ez a funkció már implementálva van a main.py fájlban

Az alkalmazás webes felületén (/docs) a fenti feladatokat el lehet végezni. Minden egyes funkció a router fájlban leírt módon viselkedik, és az ott meghatározott adatokat adja vissza Szabványos JSON formátumban, a JSONResponse függvénnyel, státusz kóddal együtt.

A végpontok megírása során az alábbi szabályokat kell betartani **(10 pont)**:

- Minden végpontnál - ahol lehet - adjuk meg a response_modell értékét (típus).
- Ügyeljünk a típusok megadására a függvényekben is.
- Az admin funkciókat tartalmazó endpointokat csak a token küldésével lehet elérni.
- A függvények visszatérési értéke lehetőség szerint JSONResponse() legyen.
- Minden függvény tartalmazzon hibakezelést, hiba esetén dobjon egy HTTPException-t és a megfelelő status code-ot (ahol ez értelmezhető).
- A függvények a JSON adatok mentéséhez és visszaolvasásához a filehandler.py és a filereader.py, a userek kezeléséhez az általunk készített funkciókat is használják. Tegyük elérhetővé ezeket az alkalmazásban! Az termék adatokat a data.json fájlba, a felhasználói adatokat a users.json fájlba kell menteni.
- A titkos token egy külön fájlban legyen, amit kívülről, vagyis az endpointokon keresztül nem lehet elérni.
- A HTTP válaszok minden esetben tartalmazzák a megfelelő status code-ot, pl 404 - Not found, vagy 200 – OK.
- A végpontok minden esetben tájékoztassanak arról, hogy milyen funkciót valósítanak meg. Ez dokumentáció jelenjen meg a „redoc” dokumentációban, de érdemes elhelyezni kommenteket is a függvények elé a kódban.
- A funkciók kisebb módosítása megengedett, de amennyiben valamelyik funkciót megváltoztatjuk, vagy módosítjuk a működését, akkor a módosítás magyarázata is legyen dokumentálva a kódban, hogy azt a védésen meg tudjuk mutatni a gyakorlat vezetőjének.

Az alkalmazás használata

A main.py fájl a REST API main modulja indítja el. A futtatásához telepíteni kell a pip csomagkezelővel, vagy az uv add-al a uvicorn és a fastapi csomagokat:
pip install uvicorn, fastapi, vagy uv add fastapi
esetleg
pip3 install uvicorn, fastapi
Ezután az alkalmazás a következő paranccsal futtatható a terminálban:
uv run **uvicorn main:app --reload --port 9000**

A port nem kötelező opció.

A futó alkalmazás a következő URL-en érhető el:
127.0.0.1:9000

A webes UI pedig az alábbi URL használatával:
127.0.0.1:9000/docs

A dokumentáció pedig itt:
127.0.0.1:9000/redoc

Az adatok kezelésére használható osztályokat a megadott schema alapján ki kell dolgozni. A schema.py tartalmazza azok küldésére és fogadására készített osztályokat. Az osztályokban az adatok legyenek validálva! **(10 pont)**

- Az int adatok nem lehetnek negatívak.
- Az email mező csak e-mail formátumot fogadhat el.
- Hiba esetén ValueErrort kell dobni és lehetőség szerint a kliens oldalon is jelezni kell a hibát.

A fájlkezelők implementálása

A data.json fájlban található adatok kezelésére két Python modult készítettünk. Az egyik, a filehandler.py, ami a fájlok írására használható függvényeket, a másik a filereader.py, ami a JSON adatok olvasására készült függvényeket tartalmaz. A függvények törzse nincs implementálva, ezért a beadandó elkészítéséhez ezeket a függvényeket is implementálni kell az alábbi leírás alapján. A users.json fájl tartalmának az írásához használható függvényeket, vagy osztályokat a fent megadott fájlokban, vagy saját modulban kell elkészíteni. **(10 pont)**:

filehandler.py

Új felhasználó hozzáadása:

```
new_user = {
   "id": 4,  # Egyedi felhasználó azonosító
   "name": "Szilvás Szabolcs",
   "email": "szabolcs@plumworld.com"
}
```

Felhasználó hozzáadása a JSON fájlhoz:

```
add_user(new_user)
```

Hozzáadunk egy új kosarat egy meglévő felhasználóhoz:

```
new_basket = {
   "id": 104,  # Egyedi kosár azonosító
   "user_id": 2,  # Az a felhasználó, akihez a kosár tartozik
   "items": []  # Kezdetben üres kosár
}

add_basket(new_basket)
```

Új termék hozzáadása egy felhasználó kosarához:

```
user_id = 2
new_item = {
   "item_id": 205,
   "name": "Szilva",
   "brand": "Stanley",
   "price": 7.99,
   "quantity": 3
}
```

Termék hozzáadása a kosárhoz:

```
add_item_to_basket(user_id, new_item)
```

Hogyan használjuk a fájlt? Importájuk a függvényeket a filehandler.py modulból:

```
from filehandler import (
   add_user,
   add_basket,
   add_item_to_basket,
)


# A JSON fájl elérési útja
JSON_FILE_PATH = ""
```

```
def load_json() -> Dict[str, Any]:
  with open(JSON_FILE_PATH, "r", encoding="utf-8") as file:
    pass

def save_json(data: Dict[str, Any]) -> None:
  pass

def add_user(user: Dict[str, Any]) -> None:
  pass

def add_basket(basket: Dict[str, Any]) -> None:
  pass

def add_item_to_basket(user_id: int, item: Dict[str, Any]) -> None:
  pass
```

filereadr.py

Felhasználó adatainak lekérdezése:

```
user_id = 1
user = get_user_by_id(user_id)
```

Felhasználó kosarának tartalmának lekérdezése:

```
user_id = 1
basket = get_basket_by_user_id(user_id)
```

Összes felhasználó lekérdezése:

```
users = get_all_users()
```

Felhasználó kosarában lévő termékek összárának lekérdezése:

```
user_id = 1
total_price = get_total_price_of_basket(user_id)
```

Hogyan futtassuk a fájlt? Importájuk a függvényeket a filehandler.py modulból:

```
from filereader import (
  get_user_by_id,
  get_basket_by_user_id,
  get_all_users,
  get_total_price_of_basket
)
```

```
# A JSON fájl elérési útja
JSON_FILE_PATH = ""

def load_json() -> Dict[str, Any]:
  pass

def get_user_by_id(user_id: int) -> Dict[str, Any]:
  pass

def get_basket_by_user_id(user_id: int) -> List[Dict[str, Any]]:
  pass

def get_all_users() -> List[Dict[str, Any]]:
  pass

def get_total_price_of_basket(user_id: int) -> float:
  pass
```

Ne feledjük, a felhasználó kezelését le kell választani a data.json fájlról, vagyis el kell készíteni a megfelelő file és adatkezelő funkciókat.

A végpontok implementációja

Az alábbi végpontokat kell kidolgozni:

Felhasználó hozzáadása a bolthoz. A bemenet egy User típus, a visszatérési érték a felvitt felhasználó rekordja. A token kezelést is implementálnunk kell.

```
@routers.post('/adduser', response_model=User)
def adduser(user: User) -> User:
  pass
```

Kosár hozzáadása egy User-hez. A bemenő paraméter a felhasználó azonosítója, a visszatérési érték a következő szöveg: „Sikeres kosár hozzárendelés.” Itt se felejtsük el a token kezelését!

```
@routers.post('/addshoppingbag')
def addshoppingbag(userid: int) -> str:
  pass
```

Termék berakása a felhasználó kosarába. A bemenő paraméterek a felhasználó azonosítója és a termék. A visszatérési érték a kosár tartalma.

```
@routers.post('/additem', response_model=Basket)
def additem(userid: int, item: Item) -> Basket:
  pass
```

Egy adott termék attribútumainak módosítása. A bemenő paraméter a felhasználó azonosítója, valamint a termék azonosítója és új attribútumai. Termék cserével is megoldható a feladat. A visszatérési érték a kosár tartalma.

```
@routers.put('/updateitem')
def updateitem(userid: id, itemid: int, updateItem: Item) -> Basket:
  pass
```

Egy termék törlése az adott felhasználó kosarából. A bemenő paraméterek a felhasználó azonosítója, valamint a termék azonosítója. A visszatérési érték a kosár tartalma.

```
@routers.delete('/deleteitem')
def deleteitem(userid: int, itemid: int) -> Basket:
  pass
```

Egy adott felhasználó adatainak a megjelenítése. A bemenő paraméter a felhasználó azonosítója, a visszatérési érték a felhasználó rekordja.

```
@routers.get('/user')
def user(userid: int) -> User:
  pass
```

Az összes felhasználó lekérdezése az adatbázisból. Nincs bemenő paraméter és a kosarak nem kerülnek megjelenítésre. A visszatérési érték a felhasználók listája.

```
@routers.get('/users')
def users() -> list[User]:
  pass
```

Egy adott felhasználó kosarának megjelenítése. A paramétere a felhasználó azonosítója. A visszatérési érték a termékek.

```
@routers.get('/shoppingbag')
def shoppinbag(userid: int) -> list[Item]:
  pass
```

Egy adott felhasználó kosarában lévő termékek értékét adja vissza. A bemenő paramétere a felhasználó azonosítója, a visszatérési érték az összeg.

```
@routers.get('/getusertotal')
def getusertotal(userid: int) -> float:
  pass
```

Kérdés esetén a gyakorlatvezetőhöz, vagy az előadás vezetőjéhez lehet fordulni.

A feladat értékeléséhez minden végpontnak működnie kell. A pontszám a megvalósítás minőségétől is függ, vagyis, ha egy végpont működik, de a megadott feltételeknek nem felel meg, akkor kevesebb pontot ér.

Az elérhető maximális pontszám: **100 pont**
Jó munkát!