Mérési jegyzőkönyv – Szoftver Laboratórium 5

4. mérés: SOA

Név:	**Páli Márton**
Neptun kód:	**Z2PWV9**
Feladat kódja:	**28 – REND**
Mérésvezető neve:	**Kiss Máté Levente**
Mérés időpontja:	**2016-04-26 10:15**
Mérés helyszíne:	**HSZK B**
Megoldott feladatok:	**1,2,3,4,5,6**
Elérhető pontszám (plusz pontok nélkül):	**73p**

Mérési feladatok megoldása

1. feladat

Magyarázat

Az első feladatban két egyszerű Python kódot kellett írni, amelyeknek a „/eszkozok.json” és a „/eszkozok/<eszk_azon>.json” címeknél kellett a megfelelő adatokat kilistázniuk, JSON formátumban. A megoldáshoz a „@app.route(„/eszkozok.<n>”)” és az „@app.route(„/eszkozok/<eszk_azon>.json”)” kezdetű metódusok tartoznak.

Először az eszközlistát csak JSON formátumban tettem lekérhetővé, később a 7-es feladattal próbálkozva egészítettem ki. Ahhoz, hogy az adatok elérhetők legyenek, egy SQL lekérdezést kellett megírni:

```
SELECT eszk_azon, nev, vasarlas FROM eszkozok
```

Ennek az SQL lekérdezésnek a segítségével és a cur.execute() metódussal már meglett az eredménytábla, amelynek a sorait belemásoltam egy JSON tömbbe, melynek sorai szótárak. Ez a tömb pont megfelelő formátumú a visszaadáshoz. Annyit kellett rajta módosítani, hogy a dátumra isoformat() függvvényt lehessen hívni. Ehhez először sima, időpont nélküli dátummá konvertáltam a date() metódussal, majd erre hívtam meg az isoformat()-ot, hogy a kellő formátumú legyen.

A feladat második részénél egy paraméteres metódust írtam meg, az egyetlen paramétere az eszköz azonosítója (eszk_azon). A feladathoz használt paraméteres SQL utasítás:

```
'SELECT nev, marka, tipus, napi_ksg, vasarlas FROM eszkozok WHERE eszk_azon = :eszk_azon', eszk_azon = eszk_azon
```

Ha ennek a lekérdezésnek az eredménytáblája None, akkor abort(404) üzenetet kap a felhasználó, ha nem, akkor a megoldást JSON formátumúra írom, az előző résznél látott módon, viszont itt már elég az eredménytábla egyetlen sorát visszaadni (több nem is lehet), mivel az eszk_azon kulcs az eszkozok táblában.

2. feladat

Magyarázat

A második feladatnál a Request távoli szolgáltatás-elérő funkciót kellett használni. Az aktuális HUF-to-EUR árfolyamot az alábbi linken tudtam lekérdezni: http://currencies.apps.grandtrunk.net/getlatest/huf/eur. Erről

a linkről a GET verbbel már lekérhető az aktuális váltószám, amelyet csak meg kellett szorozni a termék napi költségével, majd a feladat kérése szerint string típusúvá váltani, és meg is volt az eredmény euróban. Az előző feladatban használt második metódust kellett ezzel kiegészíteni.

3. feladat
Magyarázat

A harmadik feladatnál egy statikus .html weblapot kellett készíteni, amely tartalmazta a segédlet összes $-al kapcsolatos funkcióját. A weblap betöltődésekor a loadData() metódus hívódik meg, mely egy $.ajax aszinkron hívással a „../eszkozok.json” címről lekérdezi a táblázatba az adatokat, majd az eszkozok_into_table() metódussal lehet ezeket az adatokat a táblázatba felvenni. Ezek mind <script> és a </script> közé kerültek, amelyet a <body></body>-ba tettem. Ezen kívül még felvettem az oldalra a táblázatot, „eszkozok” ID-vel, hogy hivatkozni lehessen rá, és felvettem a táblázat fejlécét. A táblázatot kicsit kiszépítettem, hogy vonalakkal legyenek a cellák elválasztva.

4. feladat
Magyarázat

A negyedik feladathoz két Python kódot írtam meg, melyek a „@app.route("/eszkozok/tipus-szerint/<tipus>.json")” és az „@app.route("/eszkozok/nev-szerint/<nev>.json")” kezdetű metódusokhoz tartoznak. A kettő igencsak hasonló volt, mivel mind a kettő egy-egy paraméteres SQL lekérdezést hajtottak végre, és ez a paraméter a keresendő szó volt. A megadott paramétert ahhoz, hogy a speciális karakterek kereshetőségét megteremtsem, a Python replace() string-metódusával variáltam meg, minden speciális karakter elé beírtam egy „@” jelet, majd az SQL lekérdezést annyival módosítottam, hogy a „@” jelet „escape-elje”, vagyis a mögötte lévő karakter karakterként értelmeződjön. Az így kapott paraméteres SQL lekérdezések:

```
"SELECT eszk_azon, nev, vasarlas FROM eszkozok WHERE nev LIKE :nev
ESCAPE '@'", nev = nev
```

```
"SELECT eszk_azon, nev, vasarlas FROM eszkozok WHERE tipus
LIKE :tipus ESCAPE '@'", tipus = tipus
```

Ahhoz, hogy a csonkolt szavak is kereshetők legyen, a keresendő szavakat kiegészítettem: az elejére és végére is tettem egy-egy „%” karaktert, így már bármelyik olyan szót, amely tartalmazza a keresett kifejezést, visszaadja a kereső. Ha az SQL lekérdezés által adott eredménytábla None, akkor 404-es kódú hibaüzenettel juttatjuk a felhasználó tudtára, hogy a keresett szöveg, nem szerepel a táblába megfelelő oszlopában. A választ ismét JSON formátumban várta el a feladat, így azt az első feladathoz hasonlóan valósítottam meg.

5. feladat
Magyarázat

Az ötödik feladatot a DELETE verb megírásával kezdtem, szerintem ez volt a legegyszerűbb. A service.py – ba vettem fel egy új metódust, amely akkor hívódik meg, hogyha a „/eszkozok/<eszk_azon>” címen szeretnének valamilyen műveletet végrehajtani, a megengedett műveletek: DELETE és PUT.

A Python metódus legelején egy vizsgálatot csinálok, megnézem, hogy létezik-e a megadott azonosítóval rendelkező eszköz az adatbázisban, és ha nem akkor azt jelezze a felhasználónak (404-es hiba), ha pedig igen, akkor a megadott utasítástól függően hajtsa végre a DELETE vagy PUT szálat. A DELETE szál SQL utasítása:

```
"DELETE FROM eszkozok WHERE eszk_azon = :eszk_azon", eszk_azon = eszk_azon
```

Ha nem sikerült a törlés, akkor 500-as kódú hibát dob a program, egyébként pedig nem tér vissza semmivel.

A PUT művelethez az alábbi SQL kódot használtam fel:

```
"UPDATE eszkozok SET nev = :nev, marka = :marka, tipus = :tipus,
napi_ksg = :napi_ksg, vasarlas = TO_DATE(:vasarlas, 'yyyy-MM-dd')
WHERE eszk_azon = :eszk_azon", nev=data['nev'], marka=data['marka'],
tipus=data['tipus'], napi_ksg=data['napi_ksg'],
vasarlas=data['vasarlas'], eszk_azon=eszk_azon
```

Ha nem sikerült a beillesztés, 500-as kódú hibát dob a program, egyébként pedig a frissített elem azonosítójával tér vissza, JSON formátumban.

A POST művelethez egy külön metódust írtam meg, amely az „/eszkozok.json” hivatkozásakor elérhető. A metódus részletes magyarázata megtalálható a service.py file kommentjei között, itt csak a lényegesebb dolgokra térek ki. Ahhoz, hogy az újonnan felvett eszközhöz azonosítót tudjon rendelni a függvény, generálni kell egy ilyen azonosítót. Ez a gen_azon() metódus feladata, amely végigiterál az azonosítókon, és a legnagyobbhoz hozzáadva egyet, megteremti az új, egyedi azonosítót. Miután ez megvan, már csak végre kell hajtani az alábbi SQL utasítást:

```
"INSERT INTO eszkozok VALUES
(:eszk_azon, :nev, :marka, :tipus, :napi_ksg, TO_DATE(:vasarlas,
'yyyy-MM-dd'))”, eszk_azon=eszk_azon, nev=data['nev'],
tipus=data['tipus'], marka=data['marka'], napi_ksg=data['napi_ksg'],
vasarlas=data['vasarlas']
```

Ha nem sikerült a beillesztés, 500-as kódú hibát dob a program, egyébként pedig az új elem azonosítójával tér vissza, JSON formátumban.

6. feladat
Magyarázat

A hatodik feladathoz az eszkozok.html oldalt kellett kiegészíteni gombokkal, valamint az adatok frissítésével. Először a gombokat vettem fel a táblába, az ajaxdemo.html file mintájára, kiegészítve az ID megadásával a táblafeltöltő-ciklusban. A gombok nevének a sorhoz tartozó eszköz eszk_azon mezőjét adtam meg, hogy arra később könnyedén hivatkozhassak. A clicked() metódusát a gombnak úgy írtam meg, hogy egy $.ajax() lekérdezéssel a korábban már megírt „eszkozok/eszk_azon.json” paraméteres oldalról szedje le az adatokat, és ha ez sikeres volt, akkor azokat egy alert ablakban jelenítse meg. A $.ajax()-ban úgy adtam

meg az eszköz azonosítóját, hogy lekérdeztem a megnyomott button nevét.

A második dolog amivel foglalkoztam, az az adatok frissítése volt. Ehhez a window.setInterval() metódust használtam, ami a paraméterként megadott függvényt a (másik) paraméterként megadott időközönként meghívja. Ez a metódus ismét a loadData() volt, mivel ez tölti be az adatokat. A success függvényét még ki kellett annyival egészíteni, hogy a már meglévő sorokat törölje ki, azokra már nem lesz szükség, ezt egy for ciklussal meg lehetett oldani. Ahhoz hogy az eredeti, még frissítés előtti, sorszámot eltároljam, felvettem egy globális tableData változót.

A változás jelzéséhez eleinte összehasonlítottam a tábla eredeti, frissítés előtti, és frissítés utáni változatát a compare() függvénnyel, ami ellenőrzi a két tábla minden egyes sorának minden cellájának az egyezőségét, és ha valahol eltérést talál, akkor az eltérésnek megfelelően alert()-et küld a felhasználónak.