Mérési jegyzőkönyv – Adatbázisok Laboratórium

IV. mérés: SOA

|  |  |
| --- | --- |
| Név: | **Szabó Bence Farkas** |
| Neptun kód: | **RF57V5** |
| Feladat kódja: | **30 - VASUT** |
| Mérésvezető neve: | **Csapó Tamás** |
| Mérés időpontja: | **2018-04-09 12:15** |
| Mérés helyszíne: | **HSZK N** |
| Megoldott feladatok: | **1,2,3,4,5,6** |
| Elérhető pontszám (plusz pontok nélkül): | **40p** |

# Mérési feladatok megoldása

## feladat

Az első feladatban az **/allomasok.json**, illetve a **/allomasok/<id>.json** szolgáltatások implementálása volt.

### A megoldáshoz használt SQL utasítás

**SELECT** id, nev, varos **FROM** allomas;

**SELECT** \* **FROM** allomas **WHERE** id = :allomas\_id;

ahol az allomas\_id a paraméterként kapott érték

### A megoldás menete

A szolgáltatásokat **service.py** file-ban kellett megvalósítani. A mintaprogramhoz hasonlóan, két függvényt hoztam létre. Az első az **@app.route(’/allomasok.json’)**, melynem kilistázzuk az összes állomást az adatbázisból. Ehhez a fentbb látható SQL első lekérdezést hajtjuk végre, majd a kurzor által visszaadott eredménytömbből feltöltjük a visszatérési tömbünket a már JSON formátumra alakított állomásokkal.

Ehhez hasonló az **@app.route(’/allomasok/<id>.json)** függvény is, itt azonban egy paraméteres SQL lekérdezést (fentebb a második SQL parancs) hajtunk végre a kapott ID-t használva paraméterként. A visszakapott eredménytömb első elemét kiolvassuk (mivel az ID egyedi nem is lehet benne több elem), majd JSON formátumra alakítva visszatérünk vele.

### Tesztelés menete

A teszteléshez használható a curl parancs is, de böngészőben tesztelve is jól látható az eredmény

/allomasok.json tesztje:



vagy a curl paranccsal:

$ curl <https://localhost:20918/allomasok.json>

{

"allomasok": [

{

"id": 1,

"nev": "Budapest-Keleti",

"varos": "Budapest"

},

{

"id": 2,

"nev": "Budapest-Nyugati",

"varos": "Budapest"

},

{

"id": 3,

"nev": "Budapest-D\u00e9li",

"varos": "Budapest"

},

{

"id": 4,

"nev": "Budapest-Kelenf\u00f6ld",

"varos": "Budapest"

},

{

"id": 5,

"nev": "Budapest-Ferencv\u00e1ros",

"varos": "Budapest"

},

{

"id": 6,

"nev": "T\u00f6r\u00f6kb\u00e1lint",

"varos": "T\u00f6r\u00f6kb\u00e1lint"

},

{

"id": 7,

"nev": "Biatorb\u00e1gy",

"varos": "Biatorb\u00e1gy"

},

{

"id": 8,

"nev": "Bicske",

"varos": "Bicske"

},

{

"id": 9,

"nev": "Tatab\u00e1nya",

"varos": "Tatab\u00e1nya"

},

{

"id": 10,

"nev": "V\u00e9rtessz\u0151l\u0151s",

"varos": "V\u00e9rtessz\u0151l\u0151s"

},

{

"id": 11,

"nev": "Tata",

"varos": "Tata"

},

{

"id": 12,

"nev": "Alm\u00e1sf\u00fczit\u0151",

"varos": "Alm\u00e1sf\u00fczit\u0151"

},

{

"id": 13,

"nev": "Kom\u00e1rom",

"varos": "Kom\u00e1rom"

},

{

"id": 14,

"nev": "\u00c1cs",

"varos": "\u00c1cs"

},

{

"id": 15,

"nev": "Gy\u0151r",

"varos": "Gy\u0151r"

},

{

"id": 16,

"nev": "Hegyeshalom",

"varos": "Hegyeshalom"

},

{

"id": 17,

"nev": "Wien-S\u00fcdbahnhof",

"varos": "B\u00e9cs"

},

{

"id": 18,

"nev": "Enese",

"varos": "Enese"

},

{

"id": 19,

"nev": "Eternitgy\u00e1r",

"varos": "L\u00e1batlan"

},

{

"id": 20,

"nev": "Csorna",

"varos": "Csorna"

},

{

"id": 21,

"nev": "Kapuv\u00e1r",

"varos": "Kapuv\u00e1r"

},

{

"id": 22,

"nev": "Fert\u0151szentmikl\u00f3s",

"varos": "Fert\u0151szentmikl\u00f3s"

},

{

"id": 23,

"nev": "Balf",

"varos": "Balf"

},

{

"id": 24,

"nev": "Sopron",

"varos": "Sopron"

},

{

"id": 25,

"nev": "Dunakeszi",

"varos": "Dunakeszi"

},

{

"id": 26,

"nev": "G\u00f6d",

"varos": "G\u00f6d"

},

{

"id": 27,

"nev": "Sz\u0151dliget",

"varos": "Sz\u0151dliget"

},

{

"id": 28,

"nev": "V\u00e1c",

"varos": "V\u00e1c"

},

{

"id": 29,

"nev": "Ver\u0151cemaros",

"varos": "Ver\u0151cemaros"

},

{

"id": 30,

"nev": "Nagymaros",

"varos": "Nagymaros"

},

{

"id": 31,

"nev": "Zebeg\u00e9ny",

"varos": "Zebeg\u00e9ny"

},

{

"id": 32,

"nev": "Szob",

"varos": "Szob"

},

{

"id": 33,

"nev": "Debrecen",

"varos": "Debrecen"

},

{

"id": 34,

"nev": "Elhagyatott",

"varos": "Elhagyatott"

},

{

"id": 35,

"nev": "Kisalf\u00f6ld szerv\u00edz",

"varos": null

},

{

"id": 36,

"nev": "Szerv\u00edz",

"varos": null

},

{

"id": 37,

"nev": "Sz\u00e9kesfeh\u00e9rv\u00e1r",

"varos": "Sz\u00e9kesfeh\u00e9rv\u00e1r"

},

{

"id": 38,

"nev": "Si\u00f3fok",

"varos": "Si\u00f3fok"

},

{

"id": 39,

"nev": "Keszthely",

"varos": "Keszthely"

},

{

"id": 40,

"nev": "Zalaegerszeg",

"varos": "Zalaegerszeg"

},

{

"id": 41,

"nev": "Szombathely",

"varos": "Szombathely"

}

]

}

/allomasok/<id>.json tesztje:

$ curl <https://localhost:20918/allomasok/1.json>

{

"atlagutas": 9000,

"nev": "Budapest-Keleti",

"sztrajkutas": 2000,

"varos": "Budapest"

}

## feladat

A második feladatban az első feladat részletező szolgáltatását kellett kibővíteni úgy, hogy kiírjuk az város koordinátáit, amelyben az állomás található.

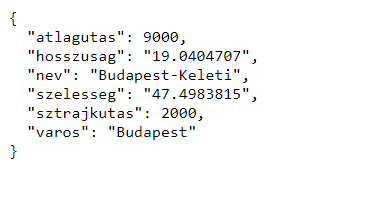
### A megoldás menete

A feladat megoldásához az @app.route(’/allomasok/<id>.json) függvényt kellett kiegészíteni. A megoldáshoz a <http://nominatim.openstreetmap.org/search> szolgáltatást használtam, mellyel lekérdezhető a paraméterben elküldött város számos adata. Nekem ebből a földrajzi szélességre és hosszúságra volt szükségem (lat és lon mezők az eredményhalmazban). Ezeket megkapva csak hozzá kellett fűzni őket az állomáshoz a jsonify() függvényben.

### Tesztelés menete

Böngészőben tesztelve:

localhost:20918/allomasok/1.json



## feladat

A harmadik feladatban egy html oldalt kellett létrehozni, melyen táblázatosan megjelennek az állomások adatai.

### A megoldás menete

A feladat megoldásához a static mappában létrehoztam egy allomasok.html file-t a minta (ajaxdemo.html) alapján. Itt módosítottam táblátat fejlécét a feladatnak megfelelően, majd a táblázat feltöltése következett. Itt a kapott adatokból beállítjuk és feltöltjük a táblázat sorait a $ objektumot CSS szelektorként használva (minden állomás elemnél egy új sort adunk a táblázathoz). Az adatok feltöltése függvényt a $.ajax függvényben hívjuk meg (amikor az oldal többi része már betöltődött). Itt lekérdezzük a már kész /allomasok.json szolgáltatás segítségével az állomásokat JSON formátumban, amit majd átadunk a load\_data\_into\_table függévnynek.

### Tesztelés menete

Böngészőben a localhost:20918/static/allomasok.html oldalt megnyitva:



## feladat

A negyedik feladatban az állomások keresését kellett megvalósítani név illetve város szerint, ahol a keresés paraméterét url-ben adjuk meg.

### A megoldáshoz használt SQL utasítás

"SELECT id, nev, varos FROM allomas WHERE nev LIKE :n ESCAPE '@'", n= "%" + nev + "%"

("SELECT id, nev, varos FROM allomas WHERE varos LIKE :var ESCAPE '@';"), var="%" + varos + "%"

### A megoldás menete

A feladat megoldásához a service.py állományban kellett megvalósítani két metódust, az egyik a név szerinti, a másik a város szerinti keresést végzi. Először a név szerinti kereséssel kezdtem. Ehhez az @app.route(’/allomasok/nev-szerint/<nev>.json’) függvényt valósítottam meg. Itt paraméterként megkapjuk a keresés kulcsát (név), melyet egy paraméteres SQL lekérdezésnek aduk át úgy, hogy elé és mögé fűzzük a „%” karaktert, így biztosítva a szó eleji, közepi és végi illeszkedést (fent az első SQL parancs). A speciális karakterek kezelését az ESCAPE ’@’-el valósítom meg.

A város szerinti keresést az @app.route(’/allomasok/varos-szerint/<varos>.json’) függvénnyel valósítottam meg, mely szinte teljesen ugyan úgy működik mint a név szerinti keresés, csak itt a várost használjuk keresési kulcsként (fent a második SQL parancs).

### Tesztelés menete

A megoldás tesztelhető böngészőből a [https://localhost:20918/allomasok/nev-szerint/<nev>.json](https://localhost:20918/allomasok/nev-szerint/%3cnev%3e.json) illetve [https://localhost:20918/allomasok/varos-szerint/<varos>.json](https://localhost:20918/allomasok/varos-szerint/%3cvaros%3e.json), ahol a **nev** és **varos** helyére írhatjuk a keresendő kulcsszót, vagy a curl paranccsal is:

pl.:

$curl <https://localhost:20918/allomasok/nev_szerint/szeg.json>

{

"allomasok": [

{

"id": 40,

"nev": "Zalaegerszeg",

"varos": "Zalaegerszeg"

}

]

}

$curl <https://localhost:20918/allomasok/varos_szerint/uda.json>

{

"allomasok": [

{

"id": 1,

"nev": "Budapest-Keleti",

"varos": "Budapest"

},

{

"id": 2,

"nev": "Budapest-Nyugati",

"varos": "Budapest"

},

{

"id": 3,

"nev": "Budapest-D\u00e9li",

"varos": "Budapest"

},

{

"id": 4,

"nev": "Budapest-Kelenf\u00f6ld",

"varos": "Budapest"

},

{

"id": 5,

"nev": "Budapest-Ferencv\u00e1ros",

"varos": "Budapest"

}

]

}

## feladat

Az 5. feladatban a http verbeket kellett megvalósítani, amelyekkel manipulálhatjuk az adatbázis tartalmát (DELETE, POST, PUT).

### A megoldáshoz használt SQL utasítás

1. "SELECT id FROM allomas WHERE id = :id ", id=id
2. "DELETE FROM allomas WHERE id = :id", id=id
3. "UPDATE allomas SET nev = :nev, varos = :varos, atlagutas = :atlagutas, sztrajkutas = :sztrajkutas WHERE id = :id", nev=**data**['nev'], varos=**data**['varos'], atlagutas=**data**['atlagutas'], sztrajkutas=**data**['sztrajkutas'], id=id
4. "SELECT MAX(id) as MAX FROM allomas"
5. "INSERT INTO allomas VALUES(:id, :nev, :varos, :atlagutas, :sztrajkutas)", id=new\_id, nev=**data**['nev'], varos=**data**['varos'], atlagutas=**data**['atlagutas'], sztrajkutas=**data**['sztrajkutas']

### A megoldás menete

A feladat megoldásához két metódust kellett megvalósítani. Az egyik, amely az **/allomasok/<id>** útvonalon érhető el. Ez valósítja meg a DELETE és PUT metódusokat, melyet a függvény **methods** paraméterében kapunk meg. Ezek után a fent látható első SQL lekérdezéssel megnézzük, hogy létezik-e a kapott id-jú állomás az adatbázisban. Ha nem 404-es Not Found hibával térünk vissza.

Ha létezik az elem és a methods-ban kapott érték DELETE, akkor törlést kell végrehajtanunk. Ehhez a fent látható második, paraméteres lekérdezést használjuk. A commit parancsot csak a lekérdezés lefutása után adjuk ki. Ha közben valahol hiba keletkezne 500-as Internal Server Error hibával térünk vissza és rollback műveletet hajtunk végre.

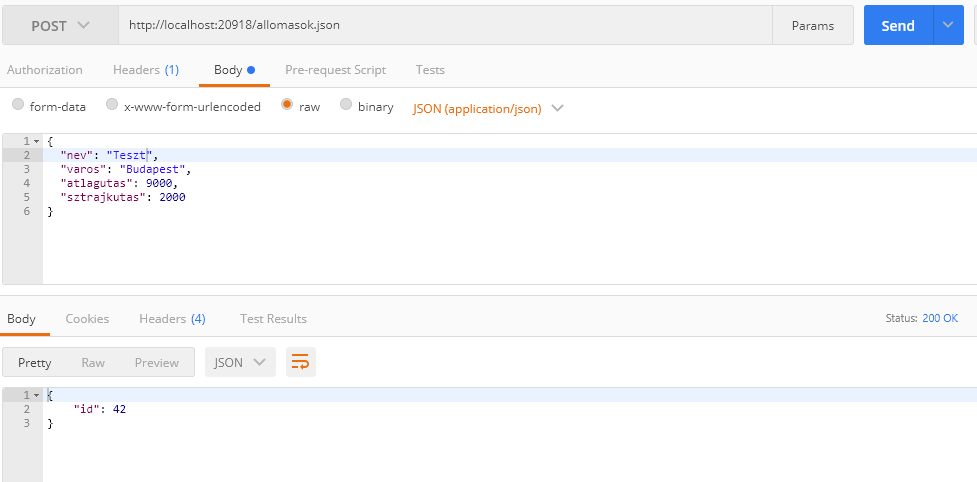
Ha a methods-ban kapott érték PUT, akkor frissítés műveletet kell végrehajtanunk. Ekkor számítunk arra, hogy a kérés törzsében szerepelnek a frissítendő állomás adatai JSON formátumban (ezt kiolvassuk innen). A frissítéshez a harmadik SQL utasítást használjuk, ahol a kapott adatokból feltöltjük a lekérdezés paramétereit. Commit-ot szintén csak a frissítés sikeres lefutása után adunk ki, ha hiba keletkezik 500-as hibaüzenettel térünk vissza. A tranzakció sikeres lefutása esetén visszaadjuk a frissített állomás id-ját JSON formátumban.

A második metódus a POST művelet megvalósítására szolgál, mely a /allomasok.json útvonalon érhető el, ahol a kérés methods paramétere a POST értéket kapja. Itt is ellenőrizzük hogy a methods értéke POST e. A beszúrandó állomás adatait a kérés törzse tartalmazza JSON formátumban, amit lekérdezünk. Az új elemnek id-t kell generálnunk. Ehhez lekérdezzük a legnagyobb id-t az adatbázisból (negyedik SQL lekérdezés), majd ezt egyel megnövelve biztosan egyedi id-t kapunk. Ezek után a kapott adatok és az új id felhasználásával beszúrjuk az új állomást az adatbázisba az ötödik, paraméteres SQL parancs segítségével. Itt is csak a tranzakciók végén adjuk ki a commit parancsot, hiba esetén pedig 500-as hibaüzenettel térünk vissza. Sikeres tranzakció esetén visszaadjuk az újonnan beszúrt elem id-ját JSON formátumban.

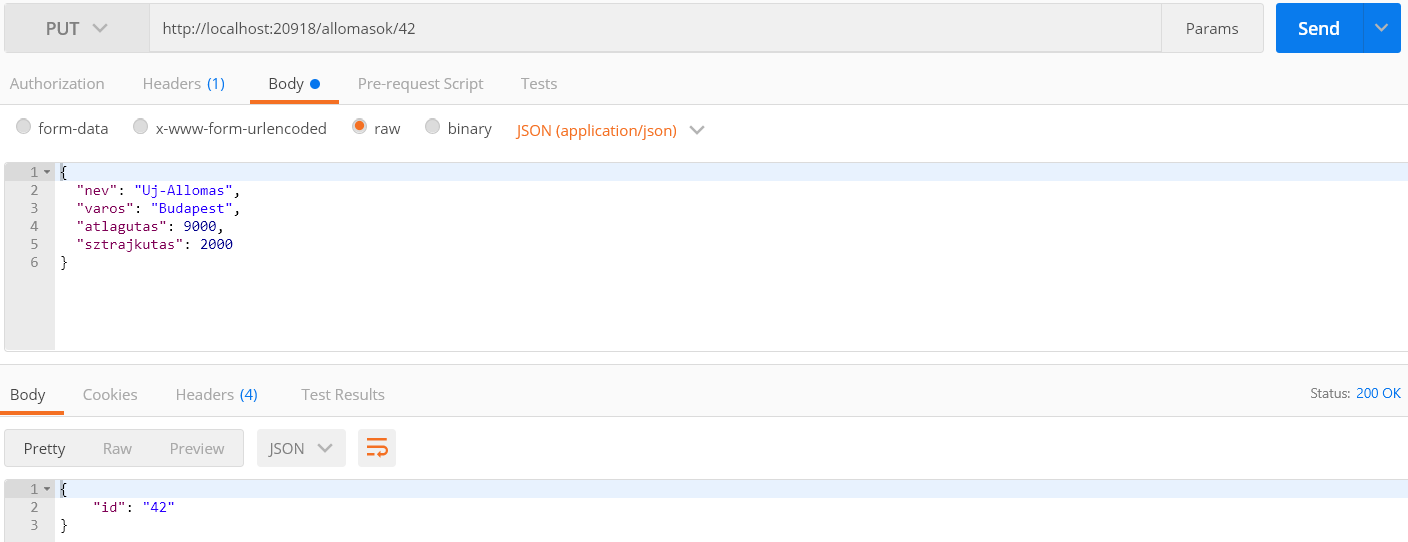
### Tesztelés menete

A tesztelést a Google Chrome Postman[[1]](#footnote-1) bővítményével végeztem, mellyel könnyedén generálhatunk különböző kéréseket.

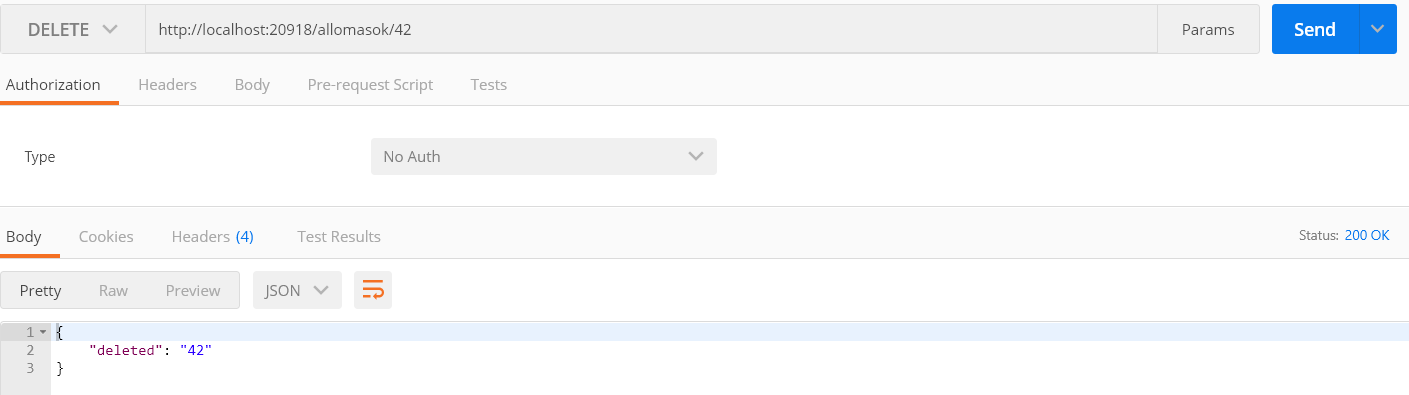
Beszúrás művelet tesztelése:



Frissítés művelet tesztelése:



Törlés művelet tesztelése:



## feladat

A 6. feladatból csak első részfeladatot oldottam meg, az állomások részleteinek kiírását egy alert ablakban.

### A megoldás menete

A megoldáshoz az **allomasok.html** file-ban kellett dolgozni. Először itt létrehoztam a táblázatban a részletező gombokat a minta megoldás alapján. Beállítottam a gomb attribútumait (nevét, ID-ját, stb.), majd létrehoztam egy kattintásra reagáló eseménykezelő függvényt szintén a minta megoldás alapján **button\_click\_handler** néven. Ez a függvény a $ objektum segítségével lekérdezi az adott id-jú állomás adatait a már kész /allomasok/<id>.json szolgáltatás segítségével. Ezek után a kapott adatokkal megívunk egy függvényt, mely összefőzi azokat egy szöveggé, majd megjeleníti egy alert() ablakban.

### Tesztelés menete

Böngészőben tesztelve:



1. https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop [↑](#footnote-ref-1)