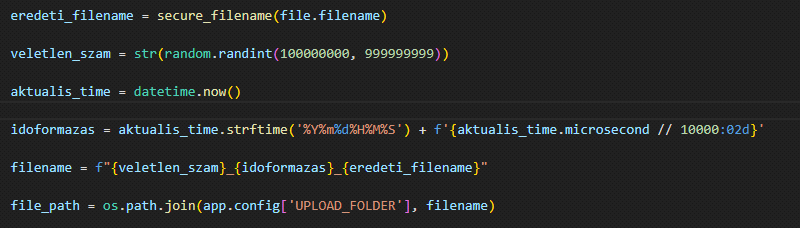
**Phyton Flask:**

A Flask egy nyílt forráskódú webes keretrendszer, amit Phyton nyelven írtak meg.

A Flask keretrendszert weboldalak, REST API-k gyors fejlesztéséhez. Egyik nagy előnye, hogy könnyen összekapcsolható más Frontend keretrendszerekkel.

A mi weblapunkon ezt arra használtuk, hogy ezzel könnyedén tudjunk képeket feltölteni. Hiszen lehetővé teszi, hogy a felhasználó a saját számítógépéről bárhonnan tud képeket feltölteni, és az elérhető lesz akkor is, ha azt letörli onnan, ahol volt vagy pedig valaki más egy másik eszközről szeretné azt megtekinteni. Mivel ez egy virtuális adatbázison keresztül bekerül a React mappába.

Mikor a képfeltöltés történik, akkor a python kód automatikusan a feltöltött kép neve elé beszúr egy 9számjegyből álló számsort, valamint az aktuális időpontot mikró szekundum pontosággal. Ezekre azért van szükség, hogy biztosítsuk, hogy a lép neve egyedi legyen, ugyanis nem lehet a kettő képnek ugyan az a neve.



* ábra. kép nevének egyedivé tétele

Backend:

Az alkalmazás azon része, amely a háttérben fut. Ez kezeli az authentikációt, adatbázist, illetve a teljesíti a frontendtől érkező kéréseket.

Ilyen például a Web API 2 ami Entity Framework 6-ot használ.

Ez úgy történik, hogy a backendben implementálva vannak metódusok, amelyeket a frontend egy végponton keresztül tud elérni.

Leggyakrabban implementált metódusok álltalános leírása:

Paraméter nélküli Get metódus:

Egy olyan Get eljárás, amely adatot kér a szervertől a végponton keresztül, anélkül, hogy bármit is küldene az adatbázisnak. Ezért az összes adatot visszaadja, ami azon a végponton található.

Felépítése:   
 Annotáció: GET

Útvonal: api/…

Válasz típusa: pl. (IHttpActionResult)

Válasz: küldi az objektumot JSON-ként

Leggyakrabban használt státuszkódok:

200 – OK

204 – No Content

400 – Bad Request

404 – Not Found

500 – Internal Server Error

Paraméteres Get metódus:

Egy olyan Get metódus, amely adatokat kér a szervertő úgy, hogy egy vagy akár több paramétert is küld a szervernek, ami alapján csak egy konkrét adatot is képes visszaadni. Az alábbi felépítési példánál Id alapján szeretnénk megkeresni egy konkrét adatot

Felépítése:

Annotáció: GET

Útvonal: api/valami/{id}

Válasz típusa: pl. (IHttpActionResult)

Válasz: küldi az objektumot JSON-ként

a leggyakrabban használt státuszkódok megegyeznek a paraméter nélküli Get metóduséval

Post metódus:

Egy olyan eljárás, amelyet leginkább adatok küldésére használunk a végpont felé. Ezt a műveletet használják akkor, mikor egy új adatot szeretnénk létrehozni az adatbázisban.

Felépítése:

Annotáció: Post

Útvonal: api/valami

Body: Adatok küldése általában JSON vagy XML formátumban

Leggyakrabban használt státuszkódok:

200 - OK

201 - Created

400 - Bad Request

409 - Conflict

500 – Internal Server Error

Patch metódus:

Egy olyan metódus, amely a már meglévő erőforrásokat tudja részlegesen módosítani. A fejlécben megadott adatok alapján tudja beazonosítani, a változtatni kívánt adatokat.

Felépítés:

Annotáció: Patch

Útvonal: api/valami/{id}

Body: a módosítani kívánt részek

Válasz Tipusa: pl. (IHttpActionResult)

Válasz: státuszkód

Legygakrabban használt státuszkódok:

200 - OK

204 - No Content

400 - Bad Request

404 – Not Found

409 – Conflict

500 – Internal Server Error

Put metódus:

Ez egy olyan modosító metódus, amely már a meglévő erőforrásokat tudja teljesen módosítani, miután a fejlécben kapott adatok után beazonosítja az erőforrást. Put eljárás hátránya, hogy a patchel szemben ez nem képes az adatok részleges módosítására. Ezért ha a felhasználó nem küld valamilyen adatot akkor az null értékre változik meg az adatbázisban.

Felépítés  
 Annotáció: Patch

Útvonal: api/valami/{id}

Body: összes adat küldése

Válasz Tipusa: pl. (IHttpActionResult)

Válasz: státuszkód

Ugyan az mint a patchnél

Delete metódus:

Ez egy olyan eljárás, amely arra szolgál, hogy egy adott erőforrást véglegesen eltávolítson a szerverről. A sikeres törléshez szükség van arra, hogy a fejlécbe küldjünk megfelelő adatot/adatokat, ami alapján be lehet azonosítani a megfelelő adatot

Felépítése:

Annotáció: Patch

Útvonal: api/valami/{id}

Válasz Tipusa: pl. (IHttpActionResult)

Válasz: státuszkód

Leggyakrabban használt státuszkod:

200 - OK

204 - No Content

400 - Bad Request

404 – Not Found

500 – Internal Server Error

Frontendtől a kérés sikeresen megérkezése, utána feldolgozza a kérésben szereplő információkat, és azokat továbbítja az adatbázis felé, ahonnan egy választ vár vissza, amit státuszkódba csomagolva továbbít a frontend felé.

Státuszkódok öt osztályra bonthatók:

tájékoztató: 100-199

sikeres: 200-299

átirányítás: 300-399

ügyfélhiba: 400-499

kiszolgáló hiba: 500-599

Leggyakrabban használt státuszkódok összefoglalva:

200 - OK: Sikeresen végrehajtódott, a kérés

201 - Created: Új elem létrehozása sikerült

204 - No Content: A kérés befejeződött, de nem szeretnénk megjeleníteni semmit

400 - Bad request: A kérésben szintaktikai hiba található

401 - Unauthorized: Hozzáféréshez kötött

404 - Not Found: Az erőforrás nem található

409 - Conflict: Ütközés történt  
 500 - Internal Server Error: probléma történt a szerveren

A backendben megtalálhatók modellek, ugyanis ezek fognak megfelelni az adatbázisban szereplő mezőknek.

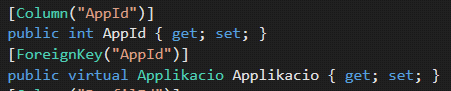
A modellekben, ha az elsődleges kulcsot nem Id-nak nevezzük, akkor ezt annotálni kell a key kulcs szóval.



Illetve érdemes a többi mezőnek ugyan azt a nevet adni, mint ami az adatbázisban szerepel, különben azokat pedig egy schemat kell beállítani, ami az adatbázisban szereplő névre mutat.



Abban a modellben, ahol egy kapcsolat idegen kulcsa található, ott azt is jelölni kell, hogy melyik modell-vel van összekapcsolni:



Mint ahogy már korábban szó volt róla, hogy a backendbe vannak implementálva azok az eljárások, amiken szükségesek azok, hogy a kliens oldalról eljussanak a kérések az adatbázishoz. Ezeket a Controllerben helyezik el.