Web programozás projekt

*Building REST APIs*

Fekete Csanád, Gazdasági informatika

2021

# Bevezető

A legtöbb nem webfejlesztő valószínűleg úgy gondolja, hogy az alkalmazások létrehozása azt jelenti, hogy megépítjük a szoftvert PC-re vagy Mac-re, játékokra vagy weboldalakra, mert nekik csak ennyi látni és használni tudnak. De ha egyszer csatlakozol a webfejlesztői közösséghez, akár önfejlesztésen ekresztül, akár szakmailag végül rájössz, hogy, mennyi munka van az alkalmazásokban és egyébb eszközökben, amelyek nem rendelkeznek felhasználói felülettel.

Gondolkozotál már valaha azon, hogy valaki webhelye hogyan férhet hozzá a Facebook-profilodhoz, és később automatikus üzenetet küld a faladra? Vagy hogyan kezelik a webhelyek az információk küldését/fogadását az oldal tartalmának frissítése érdekében, anélkül hogy frissíttenének vagy küldenének bármilyen űrlapot? Mindezen funkciók, és még sok más érdekesebb alkalmazás mögött meghuzódó integrációnak köszönhetően lehetségesek Használatuk ismerete megnyitja az ajtót az érdekesebb és hasznosabb webalkalmazások felé.

Ebben a fejezetben a következőkről fogsz tanulni:

* Bevezetés az API-ba és a REST API-ba, valamint használatukba
* A REST API-k alapja
* Harmadik féltől származó API-k használata
* Eszközök REST API fejlesztők számára
* REST API-k desing-ja és irása Laravel segítségével
* A REST API-k tesztelésének különböző módjai

# Bevezetés az API-kba

Az **API** az **Application Program Interface** rövidítése. Célja, hogy olyan felületet biztosítson hogy más programok olyan parancsokat küldhessenek, amelyek valamilyen folyamatot indítanak el a alkalmazásban, esetleg valamilyen kimenetet ad vissza. A fogalom kissé elvontnak tűnhet, de valójában mindenben van API ami valamilyen módon kapcsolódik számítógépekhez. Lássunk néhány példát a való életből:

* Az operációs rendszerek avagy OS, mint a Windows vagy Linux azok a programok, amelyek lehetővé teszik számodra a számítógépek használatát. Ha bármilyen alkalmazást használsz a számítógépeden valószínűleg valamilyen módon beszélnie kell az operációs rendszerrel, mintpéldául egy bizonyos fájl lekérésével, hanganyag elküldésével a hangszórókba, és igy tovább. Mindezek az interakciók az alkalmazás és az operációs rendszer között lehetségesek API-knak köszönhetően amiket operációs rendszer biztosított . Ily módon az alkalmazásnak nem kell direkt kapcsolatot létesitenie a hardverrel, ami nagyon fárasztó feladat lenne.
* A felhasználóval való interakcióhoz egy mobilalkalmazás grafikus felhasználói felületet(GUI) biztosít. A felület sorrendben rögzíti a felhasználó által elindított összes eseményt, például kattintást vagy gépelést hogy elküldje őket a szerverre. A GUI szerverrel kommunikál egy API ugyanúgy ahogy a program az operációs rendszerrel, ahogy azt fentebb elvolt magyarázva.
* Amikor olyan webhelyet hozol létre, amelynek meg kell jelenítenie a felhasználó tweetjeit a Twitter fiókjáról, kommunikálnia kell a Twitterrel. Olyan API-t biztosítanak amely HTTP-n keresztül érhető el. Amint megvolt a hitelesítés utána a megfelelő HTTP kérések elküldésével frissítheted és/vagy lekérheted az adatokat az alkalmazásukból.

Amint láthatod, az API-k különböző helyeken hasznosak. Általában mikor van egy rendszered, amelyhez kívülről kell hozzáférni, lehetőség szerűen meg kell adnod a felhasználóknak egy API-t. Amikor azt mondjuk, hogy kívülről, akkor egy másik alkalmazást vagy könyvtárat értjük, de nagyon jól lehet akár ugyanabban a gépben.

## Bevezetés a rest API-kba

A REST API-k az API-k egy specifikus típusát jelentik. Protokollként a HTTP-t használják hogy kommunikálj velük, így elképzelheted, hogy ők lesznek a legtöbbet használt webes alkalmazások. Valójában nem sokban különböznek az Ön által használt webhelyektől már be van építve, mivel a kliens HTTP kérést küld, a szerver pedig ezzel válaszol HTTP-válasz. A különbség az, hogy a REST API-k nagymértékben használják a HTTP-t állapotkódokat, hogy megértse, mi a válasz, és ahelyett, hogy HTML-t adna vissza CSS-sel és JS-sel rendelkező erőforrásokat, a válasz JSON-t, XML-t vagy bármely más dokumentumot használ formátumban, csak információkkal, és nem grafikus felhasználói felülettel.

Vegyünk egy példát. A Twitter API hitelesítés után lehetővé teszi a fejlesztők számára egy adott felhasználó tweetjeit kapja meg egy HTTP GET kérés elküldésével a https://api. twitter.com/1.1/statuses/user\_timeline.json címre. A válasz erre a kérésre a HTTP-üzenet, amelynek gerimce a tweetek JSON-térképe, és a 200-as állapotkód. Az állapotkódot már említettük a 2. fejezetben, Webalkalmazások PHP-vel, de rövidesen felidézzük őket.

A REST API azt is lehetővé teszi a fejlesztők számára, hogy tweeteket tegyenek közzé a felhasználó nevében. Ha te már hitelesítve vagy, mint az előző példában, csak el kell küldened a POST-kérést a https://api.twitter.com/1.1/statuses/update.json címre a megfelelő POST paramétereket a törzsben, például a tweetelni kívánt szöveg. Annak ellenére, hogy ez a kérés nem GET, és így nem adatot kérsz, hanem küldesz, a kérésnek a válasza nagyon fontos. A szerver fogja a válasz állapotkódjait, hogy tudassa a kérelmezővel, ha a tweet közzétételre került sikeresen, vagy ha nem tudták megérteni a kérést, akkor volt egy belső szerver hiba, vagy a hitelesítés nem volt érvényes, és így tovább. Ezen forgatókönyvek mindegyikére rendelkezik egy állapotkód, amely minden alkalmazásban ugyanaz. Ez teszi nagyon könnyűvé kommunikálást a különböző API-kkal, mivel nem kell új állapotkód listát tanulnod minden alkalommal. A szerver további információkat is hozzáadhat a törzshöz annak érdekében, hogy megvilágítsa, miért történt a hiba, de ez az alkalmazástól is függ.

Elképzelheted, hogy ezeket a REST API-kat a fejlesztők biztosítják, így azok integrálni tudják őket alkalmazásaikkal. Nem felhasználóbarátak, ellenben HTTP-barát.

## A REST API-k alapjai

Bár a REST API-knak nincs hivatalos szabványuk, a legtöbb fejlesztő egyetért ugyanazzal az alappal. Ez segít a HTTP-n, amely a technológia protokollját kommunikációra tudja használni, mert van szabványa. Ebben a részben megpróbáljuk körül irni hogyan kellene működjenek a REST API-k.

# HTTP “request” metódusok

A HTTP metódusok ötletét a 2. fejezetben, Webalkalmazások PHP-val már bemutattuk. Elmagyaráztuk, hogy a HTTP metódus csak a kérés igéje, amely meghatározza milyen műveletet próbál végrehajtani. Ezt a módszert már meghatároztuk, amikor HTML űrlapokkal dolgoztunk: a form tag kaphat egy opcionális attribútumot, metódust(method), amely elküldi az űrlapot az adott HTTP-módszerrel.

Ha REST API-kkal dolgozol, nem fogsz űrlapokat használni, de továbbra is pontosithatod a kérés módját. Valójában két kérelem érkezhet ugyanahhoz a végponthoz ugyanazon paraméterekkel, fejlécek és így tovább, és mégis teljesen eltérő viselkedést mutatnak módszereik miatt, ami a kérés nagyon fontos részévé teszi őket.

Mivel olyan nagy jelentőséget tulajdonítunk a HTTP metódusoknak, hogy azonosítsuk, mit is akar kérés tenni, természetes, hogy szükségünk lesz egy maroknyira belőlük. Eddig bemutattuk a GET-et és a POST-ot, de valójában nyolc különböző módszer létezik: GET, POST, PUT, DELETE, OPTIONS, HEAD, TRACE, és CONNECT. Általában csak néggyel fogsz dolgozni. Nézzük meg őket részletesen.

## GET

Ha egy kérés a GET metódust használja, az azt jelenti, hogy információt kér egy adott entitásról. A végpontnak információt kell tartalmaznia arról, hogy mi ez az entitás, mint egy könyv azonosítója. A GET használható objektumok listájának lekérdezésére is, vagy az összesre szűrve vagy lapozva.

A GET kérések szükség esetén további információkat adhatnak a kéréshez. Például, ha megpróbáljuk visszakeresni az összes könyvet, amely tartalmazza a "gyűrűk" karakterláncot, vagy ha azt akarjuk, hogy a a teljes könyvjegyzék 2. oldala. Mint már tudod, ez az extra információ GET paraméterként hozzáadódik a lekérdezési karakterlánchoz és jellel (&) összefűzve, amely a kulcs-érték párok listája. Tehát ez azt jelenti, hogy a http://bookstore.com/books?year=2001&page3 kérés valószínűleg a harmadik oldala a 2001-ben megjelent könyvek listájának.

A REST API-k kiterjedt dokumentációval rendelkeznek az elérhető végpontokról és paraméterekől, így könnyen megtanulhatod a helyes lekérdezést. Ennek ellenére mégis dokumentálva lesz, intuitív nevű paraméterekre kell számítanod, mint a példában szereplők.

## POST és PUT

A POST a második típusú HTTP-módszer, amelyről már tudsz. Használtad űrlapokban ahhoz hogy adatokat "postolj ", vagyis megpróbáltál egy erőforrást frissíteni a szerver oldalon. Amikor új könyvet akart hozzáadni vagy frissíteni, POST kérést küldtél a könyv adataival, mint POST paraméterek.

A PUT metódus nagyon hasonlít a POST metódushoz. Ez is megpróbálja hozzáadni vagy frissíteni adatokat a szerver oldalon, és ebből a célból további információkat is hozzáad a kérelem törzséhez. Miért kellene legyen két különböző módszerünk, amelyek ugyanazt teszik? Valójában két fő különbség van e módszerek között:

* A PUT kérések vagy létrehoznak egy erőforrást, vagy frissítik azt, de csak azt az érintett erőforrás amelyik a végpont által meghatározott, és semmi más. Ez azt jelenti, hogy ha mi könyvet szeretne frissíteni, a végpont kijelentené, hogy az erőforrás egy könyv, és azt adja meg, például: http://bookstore.com/books/8734. A másrészt, ha nem azonosítja a létrehozandó vagy frissítendő erőforrást a végponton, vagy egyidejűleg más erőforrásokat is érint, érdemes használni POST kéréseket.
* Az idempotencia egy bonyolult szó egy egyszerű fogalomra. Az idempotenciai HTTP metódus sokszor meghívható, és az eredmény is mindig ugyanaz lesz. Például, ha megpróbálod frissíteni a címét egy könyvnek "Don Quijote"-ra, nem számít, hányszor hívod meg, a az eredmény mindig ugyanaz lesz: az erőforrás címe "Don Quijote". Másrészt a nem idempotens módszerek eltérő eredményeket adhatnak ugyanazon kérés végrehajtásakor. Példa lehet egy végpont amely gyarapítja valamely könyv állományát. Minden alkalommal, amikor hívod, növeli a készletet, egyre többször és többször, és így az eredmény nem ugyanaz. PUT kérések idempotensek, míg a POST kérések nem.

Még ezt a magyarázatot szem előtt tartva is elég gyakori a POST és a PUT helytelen használata a fejlesztők körében, különösen, ha nincs elég tapasztalatuk REST API-k fejlesztésében. Mivel a HTML-ben lévő űrlapok csak a POST-al küldenek adatot és a nem PUT-al, az első népszerűbb. Találhatsz REST API-kat, ahol az összes az adatokat frissítő végpont POST, de némelyiküknek PUT-nak kéne lennie.

## DELETE

A DELETE HTTP metódusnak elég beszédes neve van. Akkor használják, amikor ki akarsz törölni egy erőforrást a szerveren. A PUT kérésekhez hasonlóan a DELETE végpontoknak is azonosítani kell a törölni kívánt erőforrást. Példa erre, amikor szeretnénk hogy eltávolítsunk egy könyvet az adatbázisunkból. DELETE kérést küldhetünk egy végpontra, hasonlóan a:http://bookstore.com/books/23942-hez.

A DELETE kérések csak az erőforrásokat törlik, és azokat már előre meghatározza az URL. Ennek ellenére, ha további információkat kell küldened a szervernek, használhatod a kérés törzsét ugyanúgy, mint a POST vagy PUT esetén. Valójában mindig küldhetsz információkat a kérés törzsében, beleértve a GET kéréseket is, de ez nem jelenti azt, hogy a jó gyakorlni ezt.

# Állapotkódok a válaszokban

Ha a HTTP metódusok nagyon fontosak a kérésekhez, az állapotkódok szinte nélkülözhetetlenek a válaszokhoz. Egyetlen számmal az ügyfél tudja, mit történt a kéréssel. Ez különösen akkor hasznos, ha tudod az állapot kódok szabványait, és hogy ezeket széles körben dokumentálják az interneten.

A legfontosabbakat már leírtuk a 2. fejezetben, Webalkalmazások ezzel PHP, de soroljuk fel újra, és adjunk hozzá még néhányat, amelyek fontosak a REST API-k számára. Az állapotkódok teljes listáját a https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html oldalon találod.

## 2xx – sikeres

Az összes 2-vel kezdődő állapotkódot a rendszer azon válaszokhoz használja, ahol a kérés sikeresen felvolt dolgozva, függetlenül attól, hogy GET vagy POST volt. Néhány, ebben a kategóriában a leggyakrabban használtak közül:

* **200 OK**: Ez az általános "minden rendben volt" válasz. Ha lekérdeznél egy erőforrást a választ törzsében kapnád meg, és ha frisstitenél egyet, ez azt jelenté, hogy az új adatok sikeresen el lettek mentve.
* **201 létrehozva**: Az erőforrások sikeres létrehozásakor használt válasz POST vagy PUT segítségével.
* **202 elfogadva**: Ez a válasz azt jelenti, hogy a kérést elfogadták, de még nem dolgozták fel. Ez akkor lehet hasznos, ha az ügyfélnek szüksége van az egyértelmű válaszra egy nagyon nehéz művelethez: a szerver elküldi az elfogadva a választ, majd elkezdi feldolgozni.

## 3xx – átirányítás

Bár azt gondolhatod, hogy csak egyfajta átirányítás létezik, itt van néhány pontositás:

* **301 véglegesen áthelyezve**: Ez azt jelenti, hogy az erőforrás át lett helyezve egy eltérő URL-címre, szóval onnantól kezdve próbáld meg elérni ezen az URL-en keresztül ami a válasz törszében van.
* **303 lásd máshol**: Ez azt jelenti, hogy a kérést feldolgozták, de annak érdekében hogy lásd a választ szükséges csatlakoznod a válasz törzsében elküldött URL-re.

## 4xx – ügyfélhiba

Ez a kategória állapotkódokkal rendelkezik, amelyek leírják, hogy mi történt az ügyfél kérése miatt/alatt:

* **400 rossz kérés**: Ez egy általános válasz egy rosszul formázott kérésre, azaz szintaktikai hiba van a végpontban vagy néhány várt paraméterben nem pontositottak.
* **401 jogosulatlan**: Ez azt jelenti, hogy az ügyfél nincs még sikeresen hitelesítve, és az erőforrásnak, amelyhez hozzá akar férni, szüksége van erre hitelesítés.
* **403 tiltott**: Ez a hibaüzenet azt jelenti, hogy bár a kliens hitelesítve lett, nincs elegendő jogosultsága/engedélye a hozzáféréshez az adott erőforráshoz.
* **404 nem található**: A megadott erőforrás nem található.
* **405 nem engedélyezett metódus**: Ez azt jelenti, hogy a végpont létezik, de nem fogadja el a kéréshez használt HTTP metódust, például mi próbáltuk használni a PUT-ot, de a végpont csak a POST kéréseket fogad el.

## 5xx – szerverhiba

Akár 11 különböző hiba is előfordulhat a szerver oldalon, de minket csak az egyik érdekel: az **500-as belső szerver hiba**. Használhatod ezt az állapotkódot, amikor valami váratlan, például adatbázishiba történik a kérés feldolgozása közben.

# REST API biztonság

A REST API-k hatékony eszközt jelentenek, mivel lehetővé teszik a fejlesztők számára a visszakeresést és/vagy frissíti az adatokat a szerverről. De a nagy erő nagy felelősség is jár, és a REST API tervezésekor gondolj arra, hogy adatait olyan biztonságossá tedd, amennyire csak lehetséges. Képzeld el, bárki közzétehet tweeteket a te nevedben egy egyszerű HTTP kérés használatával!

A webes alkalmazások használatához hasonlóan itt is két fogalom létezik: **hitelesítés** és **felhatalmazást**. Valaki hitelesítése azt jelenti, hogy azonosítjuk, ki ő, azaz kérését összekapcsolja egy felhasználóval az adatbázisban. Másrészt felhatalmazni valakit az az hogy megengedünk bizonyos felhasználónak bizonyos műveletek végrehajtását. Gondolhatsz a hitelesítésre a felhasználó bejelentkezési adataiként, és a jogosultságra, mint engedélyek megadása.

A REST API-knak nagyon óvatosan kell kezelniük ezt a két fogalmat. Csak azért mert a fejlesztő hitelesítette magá nem jelenti azt, hogy hozzáférhet az összes adathoz szerveren. Néha a felhasználók csak a saját adataikhoz férhetnek hozzá, míg néha te olyan szereprendszert szeretnél megvalósítani, ahol minden szerepkörnek különböző hozzáférési szintjei vannak. Ez mindig az építendő alkalmazás típusától függ.

Bár az engedélyezés a szerver oldalon történik, vagyis a szerveré adatbázis, dönti el, hogy egy adott felhasználó hozzáfér-e egy bizonyos erőforráshoz avagy sem, A hitelesítést az ügyfélnek kell elindítania. Ez azt jelenti, hogy az ügyfélnek kell tudnia kell, hogy a REST API milyen hitelesítési rendszert használ a művelet folytatásához hitelesítés. Minden REST API saját hitelesítési rendszert valósít meg, de van néhány jól ismert megvalósítás.

Alapszintű hozzáférési hitelesítés

Az alapvető hozzáférési hitelesítés – röviden BA (Basic access authentication) –, ahogy a neve is sugallja, alapvető. Az ügyfél minden kérés fejlécébe hozzáadja a felhasználó adatait, azaz a felhasználónevet és jelszót. A probléma az, hogy ezeket az információkat csak a BASE64 kódolja de nem titkosítva, így a behatoló rendkívül könnyen dekódolja a fejlécet és tiszta szövegként szerezze meg a jelszót. Ha valaha is használnod kell, hiszen őszintén szólva ez a nagyon egyszerű módja valamilyen hitelesítés megvalósításának, javasoljuk HTTPS használatát.

Ennek a módszernek a használatához össze kell kapcsolnod a felhasználónevet és a jelszót mint felhasználónév:jelszó, kódold az eredményül kapott karakterláncot Base64 használatával, és add hozzá a engedélyezési fejlécet igy:

Authorization: Basic <encoded-string>

## OAuth 2.0

Ha az alapvető hitelesítés nagyon egyszerű volt és nem biztonságos, akkor az OAuth 2.0 a legbiztonságosabb rendszer, amelyet a REST API-k használnak a hitelesítéshez, és így volt az előző is OAuth 1.0 esetében is. Ennek a szabványnak valójában különböző változatai vannak, de mindegyik ugyanazon az alapon dolgoznak:

1. Nincsenek felhasználónevek és jelszavak. Ehelyett a REST szolgáltatója az API egy pár hitelesítő adatot – egy tokent és a titkot – rendel a fejlesztőhöz.
2. A hitelesítéshez a fejlesztőnek POST kérést kell küldenie a "token" végponthoz, amely minden REST API-ban más, de ugyanaz a koncepció. Ennek a kérésnek tartalmaznia kell a kódolt fejlesztői hitelesítő adatokat.
3. A szerver az előző kérésre egy munkamenet tokennel válaszol. Ezt (és nem az említett hitelesítő adatokat első lépésben) minden kérésnek tartalmaznia kell amit a REST API-hoz készítesz. A munkamenet token biztonsági okokból lejár, ezért meg kell ismételned a második lépést, amikor ez megtörténik.

Annak ellenére, hogy ez a szabvány elég friss (2012-től), több nagy cég például a Google vagy a Facebook már megvalósította az ő REST API-aikhoz. Talán kissé túlbonyolítottnak tűnik, de hamarosan elkezded használni, sőt végrehajtani is.

# 3-ik féltől származó API-k használata

Ennyi elmélet elég is volt a REST API-kkal kapcsolatban; itt az ideje, hogy belemerüljünk egy valós példába. Ebben a részben egy kis PHP alkalmazást fogunk írni, amely kölcsönhatásba lép a következővel: a Twitter REST API-ja; amely magában foglalja a fejlesztői hitelesítő adatok kérését, a hitelesítést, és kérések küldését. A cél az, hogy megszerezd az első munkatapasztalatot REST API-al, és megmutatjuk, hogy ez egyszerűbb, mint várnád. Az is segíteni fog hogy jobban megértsd, hogyan működnek, így később könnyebb lesz sajátot építeni.

## Az alkalmazást hitelesítő adatok megszerzése

A REST API-k általában rendelkeznek az alkalmazás fogalmával. Az alkalmazás olyan, mint egy fiók a fejlesztői webhelyükön, amely azonosítja, hogy ki használja az API-t. Az általad megadott hitelesítő adatokat használja az API eléréséhez, ehhez az alkalmazáshoz lesz kapcsolva, ami azt jelenti, hogy több alkalmazás is társítható ugyanahhoz a fiókhoz.

Feltéve, hogy van Twitter-fiókod, nyissd meg a https://apps.twitter.com webhelyet új alkalmazás létrehozásához. Ehhez kattints az **Új alkalmazás létrehozása** gombra azért hogy elérd jelentkezés részleteit. A mezők nagyon magától értetődőek – csak a az alkalmazás neve, leírása és a webhely URL-címe. A visszahívási URL itt nem szükséges, mivel ez csak azokhoz az alkalmazásokhoz lesz használva, amelyekhez hozzáférés szükséges valaki más fiókjához. A folytatáshoz fogadd el a feltételeket.

Miután át lettél irányítva az alkalmazás oldalára, mindenfélét szerkeszthető információt fogsz látni. Mivel ez csak egy példa, menjünk egyenesen oda ami számít: a hitelesítő adatok. Kattints a **Kulcsok és hozzáférési tokenek** fülre hogy lásd **a fogyasztói kulcs (API-kulcs)** és a **fogyasztói titok (API-titok)** értékeit. Nincs semmi más, amire szükségünk lenne innen. Mentheted őket a fájlrendszereder ugy mint ~/.twitter\_ néven php7.json, például:

{

"key": "iTh4Mzl0EAPn9HAm98hEhAmVEXS",

"secret": "PfoWM9yq4Bh6rGbzzJhr893j4r4sMIAeVRaPMYbkDer5N6F"

}

A hitelesítő adataid biztosítása

A REST API hitelesítési adataidnak védelmét komolyan kell venni. Valójában, gondoskodnod kell mindenféle hitelesítő adatról, mint például az adatbázisról. De a különbség az, hogy az adatbázist általában a szervereden tárolod, ami kissé megnehezíti a dolgokat annak, aki támadni akar. Másrészről a harmadik féltől származó REST API nem része a rendszerednek, és valaki a hitelesítő adataiddal szabadon használhatja fiókodat a nevedben.

Soha ne foglald bele a hitelesítő adataid a kódbázisba, különösen, ha van kódod rajta van GitHub-on vagy más tárolóban. Az egyik megoldás az lenne legyen egy fájl a szervereden, a kódon kívül, a hitelesítési adatokkal; ha az fájl titkosítva van, ez még jobb. És próbáld meg frissíteni a hitelesítő adataid rendszeresen, amit valószínűleg megtehetsz a szolgáltató honlapján.

Az alkalmazás beállítása

Alkalmazásunk rendkívül egyszerű lesz. Egy osztályból áll majd, amely lehetővé teszi hogy tweeteket szerezzünk be. Ezt az app.php szkriptünk fogja kezelni. Mivel HTTP kéréseket kell küldenünk, vagy saját függvényeket írhatunk, amelyek használnak **cURL-t** (a PHP natív függvényeinek készlete), vagy használja a híres PHP könyvtárat, a **Guzzle**-t. Ez a könyvtár megtalálható a **Packagist**-ben, ezért Composert fogunk ahsználni, hogy annak a segítségével belefoglaljuk:

**$ composer require guzzlehttp/guzzle**

Lesz egy Twitter osztályunk, amely a konstruktortól kapja meg a hitelesítő adatokat, és egy nyilvános módszer a: fetchTwits. Egyelőre csak hozd létre a vázat, hogy tudjuk dolgozni vele; a későbbi részekben olyan módszereket fogunk megvalósítani. Add hozzá a következő kódot az src/Twitter.php-hez:

<?php

namespace TwitterApp;

class Twitter {

private $key;

private $secret;

public function \_\_construct(String $key, String $secret) {

$this->key = $key;

$this->secret = $secret;

}

public function fetchTwits(string name, int $count): array {

return [];

}

}

Mivel beállítottuk a TwitterApp névteret, frissítenünk kell a composer.json fájlunkat a következő kiegészítéssel. A frissítéshez ne felejtsd el futtatni a composer update-t hogy frissitsd az automatikus betöltő.

"autoload": {

"psr-4": {"TwitterApp\\": "src"}

}

Végül létrehozunk egy alap app.php fájlt, amely tartalmazza a Composer automatikus betöltőt, beolvassa a hitelesítő adatfájlokat, és létrehoz egy Twitter-példányt:

<?php

use TwitterApp\Twitter;

require \_\_DIR\_\_ . '/vendor/autoload.php';

$path = $\_SERVER['HOME'] . '/.twitter\_php7.json';

$jsonCredentials = file\_get\_contents($path);

$credentials = json\_decode($jsonCredentials, true);

$twitter = new Twitter($credentials['key'], $credentials['secret']);

## Egy hozzáférési token kérése

Egy valós alkalmazásban valószínűleg el szeretnéd választani a hitelesítéshez kapcsolódó kódot attól, amely olyan műveletekkel foglalkozik, mint az adatok lekérése vagy feladása. Az egyszerűség kedvéért hagyjuk a Twitter osztályt, hogy tudja hogyan hitelesítse magától.

Kezdjük azzal, hogy hozzá adunk egy $client tulajdonságot az osztályhoz, amely egy példányt fog tartalmazni Guzzle Client osztályából. Ez a példány tartalmazza a Twitter API alap URI-jét, amelyet a TWITTER\_API\_BASE\_URI konstansként használhatunk. Példányosítsa ezt a tulajdonságot a konstruktorban, hogy a többi metódus ki tudja használni. Te is hozzá tudsz adni egy $accessToken tulajdonságot, amely tartalmazza a Twitter API hitelesítéskor visszaadott hozzáférési tokenjét. Mindezek a változások itt vannak kiemelve:

<?php

namespace TwitterApp;

use Exception;

**use GuzzleHttp\Client;**

class Twitter {

**const TWITTER\_API\_BASE\_URI = 'https://api.twitter.com';**

private $key;

private $secret;

**private $accessToken;**

**private $client;**

public function \_\_construct(String $key, String $secret) {

$this->key = $key;

$this->secret = $secret;

**$this->client = new Client(**

**['base\_uri' => self::TWITTER\_API\_BASE\_URI]**

**);**

}

//...

}

A következő lépés az lenne, hogy írjunk egy metódust, amely a kulcs és a titok alapján hozzáférési tokent kér a szolgáltatótól. Pontosabban:

* Kösd össze a kulcsot és a titkot egy “:”-al. Kódold az eredményt a Base64 segítségével.
* Küldj egy POST kérést az /oauth2/token címre a kódot hitelesítő adatokkal mint Authorization fejléc. Tartalmazzon egy Content-Type fejlécet és egy törzset is (További információért ellenőrizd a kódot).

Most meghívjuk a Guzzle client példányának post módszerét, amely két argumentumot küld: a végpont karakterlánc (/oauth2/token) és egy tömböt opciókkal. Ezek a lehetőségek magukban foglalják a fejlécek és a kérés törzse, amint azt hamarosan látni fogod. Ennek a felhívásnak a válasza egy olyan objektum, amely azonosítja a HTTP választ. Kibonthatod a válasz tartalmát(törzs) a getBody-val. A Twitter API-válasza egy JSON, néhány értékkel. Amivel a leginkább fogsz törődni, az az access\_token, azt a token minden további API-kérelemben szerepeltetned kell. Bontsd ki és mentsd el. A teljes módszer a következőképpen néz ki:

private function requestAccessToken() {

$encodedString = base64\_encode(

$this->key . ':' . $this->secret

);

$headers = [

'Authorization' => 'Basic ' . $encodedString,

'Content-Type' => 'application/x-www-form-urlencoded;charset=UTF-8'

];

$options = [

'headers' => $headers,

'body' => 'grant\_type=client\_credentials'

];

$response = $this->client->post(self:: OAUTH\_ENDPOINT, $options);

$body = json\_decode($response->getBody(), true);

$this->accessToken = $body['access\_token'];

}

Ezt a kódot már kipróbálhatod, ha a konstruktor végéhez hozzáadod a következő két sort:

$this->requestAccessToken();  
var\_dump($this->accessToken);

Futtassd az alkalmazást a szolgáltató által adott hozzáférési token megtekintéséhez a következő parancsot használva. Ennek érdekében ne felejtsd el eltávolítani az előző két sort annak érdekében, hogy folytasd a szekciót.

**$ php app.php**

Ne felejtsd, hogy még akkor is, ha van kulcsod, titkod és hozzáférésed a token ugyanaz minden OAuth-hitelesítésnél, a ki-kódolás sajátos módja hogy, a használt végpont és a szolgáltatótól kapott válasz kizárólagosan A Twitter API-jától származhat. Lehetséges, hogy több másik is pontosan ugyanaz, de mindig ellenőrizd mindegyikhez a dokumentációt.

# Könyvészet

1. Antonio Lopez, Learning PHP 7, Packt publishing, 2016