Szakdolgozat

Miskolci Egyetem

Közösségi webalkalmazás ételek elkészítéséhez

Készítette:

Csikos Gábor

Programtervező Informatikus szak

Témavezető:

Dr. Házy Attila

Piller Imre

Miskolc, 2016

Miskolci Egyetem
Gépészmérnöki és Informatikai Kar
Alkalmazott Matematikai Tanszék **Szám:**

Szakdolgozat Feladat

Csikos Gábor (E6OBHQ) programtervező informatikus jelölt részére.

A szakdolgozat tárgyköre: Web technológiák

A szakdolgozat címe: Közösségi webalkalmazás ételek elkészítéséhez

A feladat részletezése:

Számos receptek közzétételét segítő alkalmazás érhető el, viszont ezek elsősorban a receptek tárolását, keresését és másokkal való megosztását teszik lehetővé. A szakdolgozatban bemutatott webalkalmazás az említett funkciókon túl segíti a felhasználókat az egészségesebb életmód elérésében, az ételek közötti hatékonyabb keresésben, illetve fényképek alapján értékelhetik majd egymás receptjeit.

A webalkalmazás a Laravel PHP-s, illetve a Vue.js JavaScript keretrendszerben készül. Az adatok tárolását MySQL adatbázis biztosítja. A stíluslapokat SASS előfordító segítségével generálja.

Témavezető: Dr. Házy Attila, intézetigazgató helyettes, egyetemi docens

Konzulens: Piller Imre, tanársegéd

A feladat kiadásának ideje:

. . . . . .
szakfelelős

Eredetiségi Nyilatkozat

Alulírott .....; Neptun-kód: .....
a Miskolci Egyetem Gépészmérnöki és Informatikai Karának végzős .....
szakos hallgatója ezennel büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom és aláírásommal igazolom, hogy .....
című szakdolgozatom/diplomatervem saját, önálló munkám; az abban hivatkozott szakirodalom felhasználása a forráskezelés szabályai szerint történt.

Tudomásul veszem, hogy szakdolgozat esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

Alulírott kijelentem, hogy a plágium fogalmát megismertem, és tudomásul veszem, hogy plágium esetén szakdolgozatom visszautasításra kerül.

Miskolc, ...év ..... .hó ..... .nap

.....
Hallgató

1.

szükséges (módosítás külön lapon)

A szakdolgozat feladat módosítása

nem szükséges

....

dátum témavezető(k)

2. A feladat kidolgozását ellenőriztem:

témavezető (dátum, aláírás): konzulens (dátum, aláírás):

....

....

....

3. A szakdolgozat beadható:

....

dátum témavezető(k)

4. A szakdolgozat ..... szövegoldalt

..... program protokollt (listát, felhasználói leírást)

..... elektronikus adathordozót (részletezve)

.....

..... egyéb mellékletet (részletezve)

.....

tartalmaz.

....

dátum témavezető(k)

5.

bocsátható

A szakdolgozat bírálatra

nem bocsátható

A bíráló neve: .....

....

dátum szakfelelős

6. A szakdolgozat osztályzata

a témavezető javaslata: .....

a bíráló javaslata: .....

a szakdolgozat végleges eredménye: .....

Miskolc, ...

a Záróvizsga Bizottság Elnöke

Tartalomjegyzék

1. fejezet

Bevezetés

Napjainkban, egyre nagyobb figyelmet kell fordítanunk az egészséges táplálkozásra. A hazai piacon számos olyan alkalmazást találunk, amelyekkel a kedvenc ételreceptjeinket összegyűjthetjük és megoszthatjuk másokkal. Bár ezen alkalmazások többsége kevés funkciót tartalmaz, mégis egyik fő hiányosságuk, hogy nem veszik figyelembe a felhasználók életmódját, és nem kínálnak fel alternatívákat olyan esetekre, amikor a felhasználó vegetáriánus vagy vegán életmódot folytat, esetleg valamely alapanyagokra allergiás.

Az elkészítésével elsősorban szeretném megmutatni, hogy egy recept alkalmazás is alkothat egy közösségi hálózatot, akár az *Instragram* vagy a *Facebook*. A növekvő felhasználói igény és a hazai piacon lévő hiánya inspirált az alkalmazás elkészítésére. A dolgozatban szó lesz a felhasznált technológiákról és a jelen lévő alkalmazásokról, valamint bemutatásra kerül az alkalmazás, és áttekintjük néhány fejlesztési megvalósítást[1].

A dolgozat célja egy olyan közösségi webalkalmazás elkészítése, amely többletfunkcióival túlmutat a jelenleg elérhető weboldalak, alkalmazások funkcióin.

[1]A kutató munka a Miskolci Egyetem stratégiai kutatási területén működő Mechatronikai és Logisztikai Kiválósági Központ keretében valósult meg.

2. fejezet

Receptek nyilvántartása

2.1. A célcsoport meghatározása

A hazai és a nemzetközi piacon számos mobil és webes ételrecept alkalmazást megtalálhatunk, mégis az emberek többsége jelenleg is a közösségi alkalmazások lehetőségeit használja receptjeinek megosztásához. A Pinterest, az Instragram valamint a Facebook lehetővé teszi a felhasználói számára, hogy azonnali visszajelzést kapjanak, amint valamilyen esemény bekövetkezik amely a felhasználót érdekelheti.

Az említett közösségi alkalmazások ilyen célú használatának egyik hátránya, hogy nem rendelkeznek megfelelő szűrési lehetőségekkel. Ilyen esetben olyan receptről is értesítést kaphat a felhasználó, amely az egyéni étrendje vagy életmódja miatt nem megfelelő. A keresési lehetőségek pontos találatot biztosítanak a felhasználóik számára, azonban ezen alkalmazások nem étel receptek keresésére lettek optimalizálva, ezáltal a megadott kulcsszavak alapján olyan találatokat is felkínál, melyek nem fedik le például a recept hozzávalóit.

Az alkalmazás elkészítése során a fentebb említett hiányosságok mellett, a növekvő felhasználói igényre is tekintettel voltam, ennek során számos felhasználói célcsoportot határoztam meg, melyekbe beletartoznak azok, akik

- ételallergiában szenvednek,
- változatosabban és tudatosabban szeretnének táplálkozni,
- tudatosabban szeretnének vásárolni,
- keresik a motivációt,
- időt szeretnének megtakarítani az étrendjük összeállításával.

Sokan a közösségi hálózatok lehetőségeit kihasználva szeretik megmutatni, hogy éppen mit esznek. Erre való tekintettel ez a felhasználói csoport is lehetőséget kapott az alkalmazás használatára azáltal, hogy a recept feltöltése során enyhe szabályok és megkötések lettek alkalmazva.

2.2. Az alkalmazások összevetése

A hazai piacon jelenlévő recept alkalmazások (Apróséf, NoSalty, ReceptNeked, stb.) használatának egyik hátránya lehet, hogy olyan tartalomkezelő rendszereket (Content

Management System) használnak, amik ki vannak téve a folytonos hackertámadásoknak. Az ilyen támadások során az illetéktelen felhasználók teljes körű hozzáférést szerezhetnek, ezáltal hozzáférhetnek a felhasználók személyes adataihoz, valamint jelentős vagyoni károkat okozhatnak.

A legtöbb alkalmazásnál a bejelentkezés és a regisztráció alap funkciónak számít, mégis ezek jelenthetik a leginkább felmerülő gondot. A regisztráció során a felhasználót sok olyan adat megadására kötelezik, amik kitöltése felesleges mind a felhasználó mind pedig a fejlesztő számára, mivel a későbbiekben ezen adatok felhasználására és elemzésére nem kerül sor. Az alkalmazások esetén a bejelentkezési lehetőség kiválasztása illetve hibás működése a felhasználók elvesztésével járhat. Utóbbi a Facebook fiókkal történő bejelentkezésnél felmerülő gond, amely során a funkció nem az elvárásoknak megfelelően működik. Ennek egyik oka lehet, hogy az első bejelentkezés után az alkalmazás nem regisztrálja a felhasználót, ezáltal rákényszerítve arra, hogy másik bejelentkezési lehetőség után nézzen. A fejlesztés során ezen gondokra ügyelve készítettem el egy, a felhasználók által elvárható bejelentkezési és regisztrálási lehetőséget, amikről bővebben a `Bejelentkezés és regisztráció` alfejezetben lesz szó.

A vizsgált alkalmazásokban további közös vonás, hogy a felhasználóik számára elérhetővé teszik a recept feltöltést. Ennek ellenére találhatunk olyan alkalmazásokat, amelyek a feltöltés során további adatok megadására kötelezik a felhasználókat. A feltöltés során a recept megjelenéséhez szükséges elvárások megadásával korlátozzák a felhasználók lehetőségeit. Ilyen megkötés lehet az ételről elkészített kép méretének, minőségének és fájlméretének a figyelembe vétele, ami által a felhasználót más alkalmazások használatára kényszeríti. További hátránnyal járhat, hogy a feltöltött receptek csak jóváhagyás után jelennek meg, ezáltal a felhasználók nem kapnak azonnali visszajelzést a receptjeikről.

A receptjeink módosítására általában ezekben úgy van lehetőségünk, hogy a módosítás szintén jóváhagyás után jelenik csak meg. A receptjeink eltávolítására nem minden esetben kapunk lehetőséget.

A recept feltöltésnél számos olyan funkciót megtalálunk, melyek továbbfejlesztésével egyszerűbbé és jobbá válik az alkalmazás használata a felhasználók számára. Ilyen funkció a recept alapján az étel elkészítési költségének a megadása. Amíg a hazai alkalmazások a felhasználót kérik meg a recept elkészítési költségének a megadására, addig a külföldi alkalmazások a hozzávalók alapján számolnak ki egy becsült értéket. Mivel a hozzávalók árai eltérőek lehetnek és változhatnak, emiatt a felhasználók nem kapnak pontos összeget a recept elkészítésénél. Ennek megoldására részletesebben a `Közeljövőbeli fejlesztések` alfejezetben olvashatunk.

Ezen alkalmazások használata során számos alapfunkcióbeli hiányosságot és hibát tapasztaltam. Annak ellenére, hogy a legtöbb ilyen alkalmazás a közösségre épül, az alkalmazáson belül nincs lehetőségünk a felmerülő hibák és hiányosságok jelentésére.

2.3. Technológiai megvalósítás

Az elkészítés során fontos folyamat volt a megfelelő fejlesztői keretrendszerek és könyvtárak kiválasztása. A jelenlévő technológiák alkalmazásával nem csak a fejlesztési idő rövidül, de ezáltal tisztább kódot kaphatunk, aminek köszönhetően a későbbiekben könnyebben tudjuk az alkalmazásunkat módosítani. A fejlesztés során elsőként az alkalmazás szerver oldali részét kellet megtervezni. Ebben a támpontot a kiválasztott programozási nyelv és a hozzá elérhető keretrendszerek jelentettek. Az alkalmazás PHP

programozási nyelvben készült, amihez a Laravel keretrendszer állt a rendelkezésemre. A keretrendszer bemutatásáról a `A Laravel keretrendszer` fejezetben olvashatunk bővebben.

A kliens oldali rész több különböző komponensből tevődik össze. A Laravel-ben megtalálható Blade Template Engine-re alapozva és a Laravel által is támogatott Vue.js JavaScript keretrendszer együttes alkalmazásával egy dinamikus kliens oldalt alakítottam ki. A Vue.js lehetővé teszi a direktívák alkalmazását, ezáltal nem követeli meg, hogy a kliens kifejezetten JavaScript keretrendszerből álljon. A direktívák alkalmazása nem jár szigorú megkötésekkel, így együtt tudjuk használni a jQuery könyvtárral is.

Az alkalmazás SPA (Single Page Application) alapon készült el, amely a későbbi fejlesztések során a Vue.js által átalakítható MPA (Multi Page Application) alapra, ezáltal jobb felhasználói élményt biztosíthat az alkalmazás felhasználóinak. A kliens oldali stíluslap Sass stíluslapra épül, amihez egy Bourbon mixin is párosul, így egy letisztult stíluslapot kapunk.

2.4. Elérhető funkciók

A fejlesztése során elsődleges célom volt a megfelelő autentikáció biztosítása. Az autentikációt követően a felhasználónak lehetősége van receptek feltöltésére, módosítására és törlésére mindenféle jóváhagyás nélkül. A mások által megosztott recepteket a kedvelés funkció által elmentheti, így a későbbiek során bármikor visszanézheti, hogy mely recepteket kedvelte. A kedvelés mellet lehetősége van más felhasználókat követni, ezáltal a követett felhasználók receptjei jelennek meg a bejelentkezés utáni kezdőoldalon. A kedvelés és a követés során az érintett felhasználó értesítést kap.

A kereső funkció által a felhasználó rákereshet receptekre és személyekre, ahol utóbbinál a profil oldalra eljutva információkat tudhat meg az illetőről. A beállítások oldalon az alapbeállítások mellett, a felhasználónak lehetősége van az ételallergia megadására. Az ételallergia beállításával a felhasználó nem fog olyan recept találatot kapni, amelyek hozzávalóira allergiás. Emellett lehetősége van a testtömegindex megadására, ahol egy grafikonon nyomon tudja követni a változást, így egy tudatosabb táplálkozást kialakítva. A grafikonok által jobb követési és viszonyítási lehetőséget kap, így akár egy pozitív visszajelzésben is részesülhet, amely tovább segíti őt a célja elérésében.

2.5. Közeljövőbeli fejlesztések

Az alkalmazások folyamatos fejlődésben vannak, a fejlődő technológiai lehetőségek és a felhasználói közösség változó elvárásai által. A közeljövőben számos új funkciót fog kapni az alkalmazás, ami által jobb és letisztultabb lesz.

A kliens oldali recept feltöltés lépésekre lesz felbontva, ezáltal a felhasználók egyszerűbben tudnak feltölteni új recepteket. A szűrési feltételek fejlesztése, valamint a felhasználói tevékenységek követése lehetővé fogják tenni, hogy a felhasználók a számukra leginkább megfelelő receptet kapják.

Az animált képek lejátszása és a videó feltöltés jelenleg nem támogatott. Ezen formátumok kezelése szintén bekerül a fejlesztések közé, ezáltal támogatást nyújtva azon felhasználóknak, akik jobban szeretnek ilyen tartalmakat használni.

A világpiacon számos kiváló recept alkalmazás megtalálható, viszont ezen alkalmazások egyik hátránya a nyelvesítés. Ennek hiányában egy recept alkalmazás számos

felhasználót elveszít, mivel a felhasználó nem beszéli az adott nyelvet, vagy ha mégis, számára egyszerűbb az adott ételreceptet az anyanyelvén elolvasni. A nyelvesítés hozzáadásával a felhasználók nem csak a hazai ízek receptjeit ismerhetik majd meg, hanem első kézből kaphatják meg a külföldi recepteket.

További fejlesztés terv, hogy az alkalmazás a hűtőben vagy kamrában lévő hozzávalók alapján tud majd felkínálni recepteket, ezáltal elkerülve azt, hogy a felhasználónak közvetlenül a főzés előtt még be kelljen vásárolnia. Ezzel együtt olyan receptet is felkínálhat, amelynek az elkészítéséhez csak pár darab további hozzávaló szükséges. A bevásárlás során felkínálja a közelben lévő üzleteket és a termékek árait. Abban az esetben, ha a felhasználó kezdene kifogyni a hozzávalókból, egy értesítést kap a felsorolt hozzávalók bevásárlására. A vásárlás során lehetőségünk lesz az árak összehasonlítására és a kiadásaink követésére.

A recept elkészítésének költségeit a hozzávalók ára alapján fogja majd becsülni. Ehhez viszont elengedhetetlen az olyan keresőrobotok alkalmazása, amelyek felkeresik az adott termékeket az üzletek oldalain, és a termékhez tartozó ár alapján mindig a recept elkészítési költségének az aktuális árát kapjuk meg.

A későbbiekben akár egy mobilalkalmazás is készíthető, ahol egy időzítő alkalmazásával a felhasználó riasztást kap az étel elkészítésének idejével kapcsolatban. Ennek a megvalósítása webes alkalmazás esetén is megoldható, viszont a felhasználók így nehezebben jutnak hozzá időben ezen riasztáshoz.

A követések blokk áthelyezését követően az üzenet/beszélgetés veszi át a helyét, ezáltal lehetőség nyílik a kommunikációra a felhasználók között, amivel együtt a görgetősáv is megújul. A beszélgetéssel együtt feltöltési szűrők is helyet kapnak, ami által a nem megfelelő tartalmakat a rendszer kiszűri.

Biztosítva lesz a valósidejű üzenet és értesítés megjelenítésének a továbbfejlesztése, ami által a kliens és a szerver oldal is kevesebb terhelést fog kapni. Ezzel együtt, és a letisztultabb forráskóddal várhatóan az alkalmazás teljesítménye is javulni fog.

3. fejezet

A Laravel keretrendszer

3.1. A Laravel története

A Laravel egy ingyenesen elérhető, nyílt forráskódú PHP keretrendszer. Fejlesztője Taylor Otwell, az Arkansas-i Műszaki Egyetem elvégzését követően internetes vállalkozásait szerette volna megvalósítani. A terveinek véghezviteléhez szüksége volt egy olyan PHP keretrendszerre, amivel hatékonyan tud fejleszteni webes alkalmazásokat. Az akkoriban elérhető keretrendszerekből hiányolta az olyan alapvető közös funkciókat, mint például a felhasználói hitelesítés és engedélyezés, illetve a nyelvesítés. Ennek következménye képpen Taylor Otwell saját keretrendszer fejlesztésébe kezdett, és 2011 júniusában kiadta a Laravel béta verzióját, és még ugyanebben a hónapban megjelent a stabilabb verziója, a Laravel 1.

A Laravel 1-ben megtalálható volt a Model és a View, viszont a Controller hiánya miatt nem felelt meg az MVC szerkezeti mintának. Ennek ellenére a fejlesztők gyorsan megkedvelték az egyszerűsége, a letisztult szintaxisa és a jól dokumentáltsága miatt. Az autentikáció integrálása lehetővé tette a fejlesztők számára, hogy egy adott projektet hamarabb tudjanak elkészíteni.

Az adatbázis kapcsolatot létrehozó osztályok (*Database Connectors*) lehetővé teszik a MySQL, a PostgreSQL és az SQLite relációs adatbázis-kezelőkhöz való kapcsolódást, valamint az Eloquent objektum-relációs leképzés biztosítja az adatbázissal való hatékonyabb munkát azáltal, hogy minden egyes adatbázis táblához hozzárendel egy megfeleltetett modell osztályt. A fejlesztők munkáját szintén segítette a Pagination és a Helper metódusok bevezetése.

A Laravel 2-ben megjelent a Controller, amivel együtt a Laravel már megfelelt az MVC szerkezeti mintának. Ezzel együtt megjelent a HTTP kérés validáció (*Request Validation*). A validáció a felhasználók által felvitt adatokat a fejlesztő által megadott szabály szerint ellenőrzi, és érvénytelen input esetén hibaüzenettel tér vissza. A modellek továbbfejlesztése és a route-ok megújulása tovább segítette azon fejlesztők munkáját, akik a Laravel 2-t választották.

Mialatt a Laravel 1-ben egy route átadása esetén tartalmazni (include) kellett a PHP fájlt elérési útvonallal együtt, addig a Laravel 2-ben az átadott Controller és metódus egyezőség alapján hívta meg az adott metódust és adta át az útvonalat. A Session Driver-ek beépítésével lehetővé vált a felhasználók adatainak tárolása a különféle driver-ekben, mint például az adatbázis vagy a Redis. Szintén ezen verzióban került be a Blade Template Engine, ami által a fejlesztők kezdték jobban megkedvelni a Laravel egyszerűségét.

A Laravel 3 számos újítást hozott a fejlesztői közösségnek. Bekerült az Events, az egységtesztelés (*Unit testing*), az Inversion of Control Container valamint egy parancssoros felhasználói felület (Command Line Interface), ami az Artisan nevet kapta. A Database Connector kiegészült az SQL Server-hez való kapcsolódással és további újítás volt a Migration, amivel a fejlesztők könnyedén megtudták tervezni az adatbázis sémáikat. Ehhez hozzátartozott még a Database Schema Builder, amely lehetővé tette ezen sémák integrálását az adatbázisba. Ezen verzióban jelent meg a Bundles is, ami lehetővé tette a fejlesztők számára, hogy a Laravel által készített kódjaikat megoszthassák egymással, mások kódjait felhasználhassák.

A Laravel 4-ben megjelent a Packages (a Bundles utódja) ami engedélyezte a fejlesztői csomagok önállóságát, ezáltal képes volt egy másik Laravel alapú keretrendszerrel is együtt működni. Megjelent a Database Seeding, ami lehetővé tette az adatbázis táblák adatokkal való feltöltését. A Mail támogatása biztosította az alkalmazás különféle levelező rendszerekkel való összekapcsolását. A verzióban megjelenő üzenetsorok (Queues) által lehetőség nyílt az időigényes kérések egy későbbi időpontra való elhalasztására. Szintén ebben a verzióban jelent meg a Facades.

A Laravel 5-ben (korábban 4.3) a Flysystem csomag által könnyebben érhetjük el a távoli tárolóinkat. Az Elixir által a csomageszközeinket könnyebben tudjuk kezelni, valamint a Socialite megjelenése lehetőséget biztosít az OAuth autentikáció használatára. A Notification által az e-mail üzenetek küldése mellett, lehetőségünk van SMS és Slack értesítések küldésére is. A Laravel 5 fejlődésével párhuzamosan, a Laravel 4-el készült alkalmazások is megkapják ugyanazon hibajavításokat és sebességbeli fejlesztéseket, amik által az alkalmazásuk biztonságosabb és stabilabb lesz. A Laravel az évek elteltével számos változáson esett át. Az alábbi kód a route-ok kezelésére szolgáló `routes.php` szintaxisának átalakulását szemlélteti Laravel 1-ben és Laravel 5-ben.

Routing Laravel 1-ben

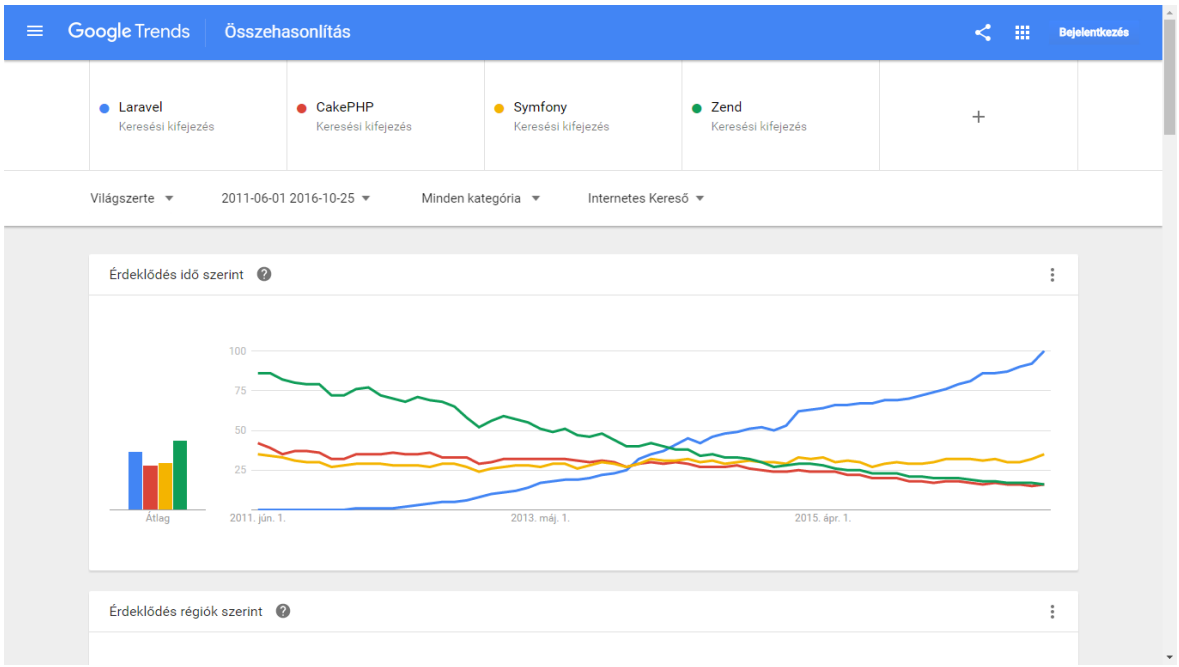
```
<?php
    'GET /' => array('name' => 'home', 'do' => function() {
        return View::make('home/index');
    });
```

Routing Laravel 5-ben

```
<?php
    Route::get('/', function() {
        return view('home.index');
    })->name('home');
```

Laravel megjelenésével együtt számos korábbi Zend keretrendszert használó fejlesztő váltott át Laravel-re, mivel előnyt jelentett a Zend-hez képest az egyszerűsége és a jól dokumentáltsága. Mivel a Laravel-ben megtalálhatóak a Symfony keretrendszer komponensei, ezért az akkori Symfony közösséget nem befolyásolta az átállás. A közös komponensek által a PHP fejlesztők könnyen tudnak váltani a két keretrendszer között, nem jelent különösebb nehézséget az átállás.

Az alábbi 3.1. ábra a Symfony, a CakePHP és a Zend keretrendszerek Google keresési népszerűségét szemlélteti a Laravel keretrendszerrel szemben a Laravel megjelenésétől kezdődően.



3.1. ábra. Keretrendszerek népszerűsége - Google Trends

3.2. Az alkalmazás felépítése

Amikor kérést küldünk a böngészőn (Browser) keresztül egy Laravel alkalmazásnak, a kérés számos komponensen megy keresztül, mire az alkalmazás visszatér a megfelelő nézettel.

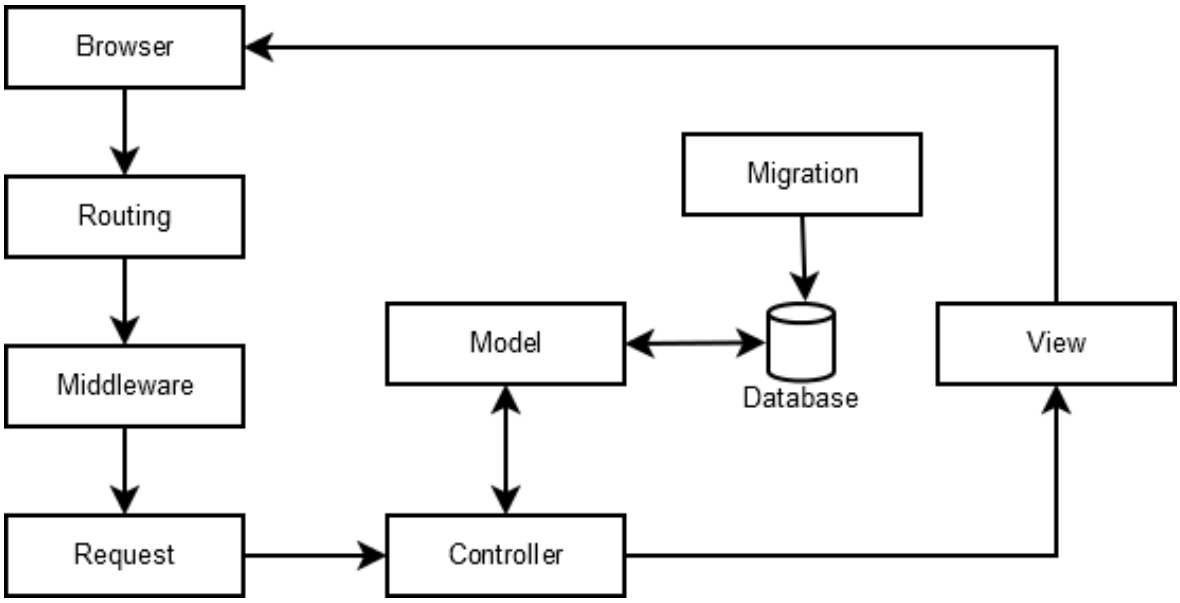
A kérés feldolgozása során a szerver fogadja a kérést és átadja a Routing komponensnek. A Routing ellenőrzi az átadott útvonal létezését a route fájlokban, és meghívja az útvonalnak megfelelő metódust. Abban az esetben, ha az útvonal nem létezik, akkor egy kivétellel tér vissza és ennek megfelelően generálja le a nézetet.

Az alkalmazás útvonalai alapértelmezetten a `routes\web.php` vannak meghatározva, melyek automatikusan betöltődnek a keretrendszerrel együtt. A jobb keresési lehetőség és áttekinthetőség érdekében az útvonalak csoportosítva lettek a megfelelő útvonal fájlokban, amiket a *route.php* fájl hív meg.

A Middleware komponens biztosítja, hogy egy adott útvonalat csak a megfelelő jogosultsággal lehessen megtekinteni. Mellette szűrési lehetőségeket is beállíthatunk, ezáltal a szűrni kívánt felhasználókkal műveleteket tudunk végezni. Ugyanilyen szűrés van biztosítva az alkalmazás során, ami vizsgálja hogy a felhasználó a regisztráció során megadta-e a nevét vagy a felhasználónevét. Ha ezen adatok hiányosak, akkor egy űrlap jelenik meg számára, ahol meg kell adnia a kitöltendő adatokat. A Middleware-k a `app\Http\Middleware` mappában érhetőek el. Miután definiálunk egy Middleware-t, át kell adnunk a Kernel-nek, hogy azt a Routing segítségével a keretrendszer azt meg tudja hívni.

Egy útvonal elérése esetén az útvonalhoz tartozó osztály és metódus kerül meghívásra. Amikor POST kérést küldünk a szerver felé a Request komponens fogja a kérésünket feldolgozni. A Controller-ben meghívhatjuk a Request-et és validálhatjuk annak adattagjait. A sikeres validáció során kérést küldhetünk a Model-en keresztül, amely a kért adatokkal tér vissza az adatbázisból, vagy éppen adatokkal tölti fel az adatbázist. Adatokat a Migration által is lehetőségünk van feltölteni.

Miután a metódus végez a feladataival az adatokat továbbadja a View-nak, amely a kérés feldolgozása során az adatok alapján renderel egy nézetet és elküldi azt a böngészőnek.



3.2. ábra. Az architektúra

3.3. Eszközök

A Laravel számos eszközt és csomagot biztosít, amelyek segítik megadni a kezdő lépéseket a fejlesztőknek. Akár egy egy könnyűsúlyú akár egy komplex alkalmazást kell elkészíteni, a készen elérhető csomagok által gyorsabban és hatékonyabban lehet fejleszteni.

Az alkalmazások fejlesztése esetén meg kell határozni, hogy az adott projektben milyen funkciók legyenek elérhetőek. Abban az esetben, ha az alkalmazás csak egy töredékét használná fel a Laravel keretrendszer által adott funkcióknak, a Laravel biztosítja a keretrendszere könnyűsúlyú változatát, a Lumen PHP mikro-keretrendszert. A Lumen előnye a Laravel-lel szemben a sebességében mutatkozik meg. A komponensei a Laravel-en alapszanak, viszont használata néhol eltérhet, mint például a Routing esetén.

Routing Lumen 5-ben

```
<?php
    $app->get('/', ['as' => 'home', function() {
        return view('home.index');
    }]);
```

3.4. Routing

Az útvonalak lehetővé teszik, hogy az alkalmazásnak küldött kérések a nekik megfelelő válasszal térhessenek vissza. Az útvonalainknak átadhatunk paramétereket, csoportok-

ba foglalhatjuk és nevesíthetjük őket. A Laravel hivatalos dokumentációjában ezek használatáról többet tudhatunk meg a `https://laravel.com/docs/routing` oldalon.

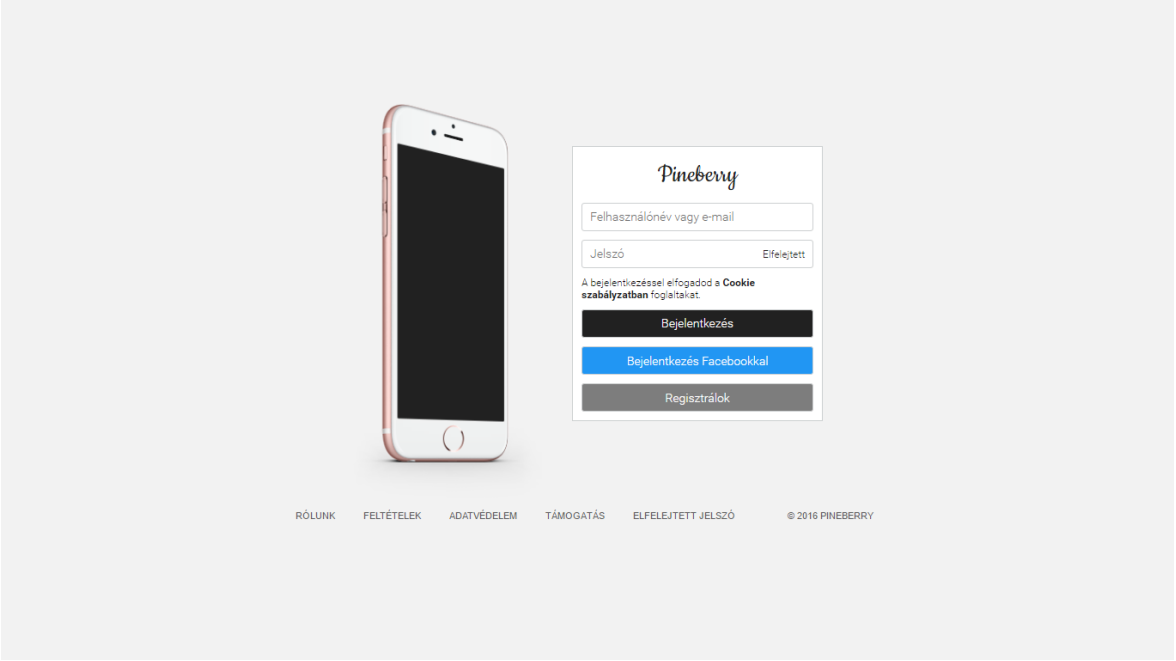
Az alábbi kódban egy post és egy get kérésre látunk példát, ahol a `group` függvénynek átadunk egy tömböt és egy Closure-t. A tömb paramétere egy *account* prefix lesz, ahol a prefix a csoporton belüli post és get kérések első paraméter elejét helyettesít. Ebben a példában az *account/general* és az *account/delete* URL-t váltja ki. Hasonlóan a prefix-hez a namespace egy *Auth* névteret tartalmaz. A *name* függvénnyel nevesítjük az útvonalainkat, ami által könnyen hivatkozhatunk egy útvonalra a `route` függvénnyel. Például ha az *account/delete* oldalt szeretnénk átadni hívás során, akkor használhatjuk a `route('general')` függvényt. A get és a post második paramétere hasonlóan a `group` függvényhez, tartalmazhat egy *Closure*-t vagy egy osztály/metódus hívást.

```
Route::group(['prefix' => 'account'], function () {

    Route::get('general', 'HomeController@getSettings')
        ->name('general');

    Route::group(['namespace' => 'Auth'], function() {

        Route::post('delete', 'AccountController@destroy')
            ->name('account.destroy');
    });
});
```

4. fejezet

Felhasználói dokumentáció

Az alkalmazást egy, az alkalmazás által létrehozott felhasználói fiókkal lehet igénybe venni, melyre kétféle bejelentkezési lehetőség biztosított a felhasználó számára. Az egyik a Facebook fiók által való bejelentkezés, míg a másik az alkalmazás által létrehozott felhasználói fiók.

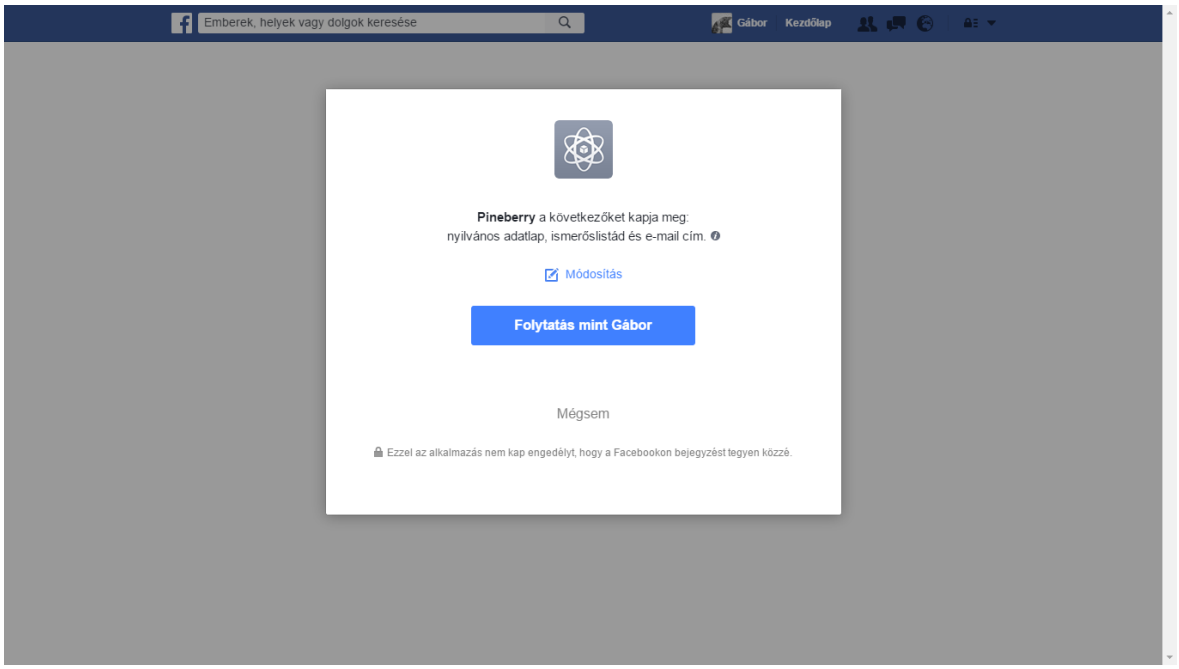


4.1. ábra. Az alkalmazás bejelentkező felülete

Ha rendelkezünk Facebook fiókkal és a későbbiekben is ezzel a bejelentkezési lehetőséggel szeretnénk élni, akkor kattintsunk a `Bejelentkezés Facebookkal` gombra (4.1. ábra). Az első bejelentkezés során engedélyeznünk kell az alkalmazás által kért adatokat a Facebook-on keresztül, ahogy a 4.2. ábrán is látható. Ezáltal létrehozzuk a fiókunkat. A későbbiekben a `Bejelentkezés Facebookkal` való gombra kattintás során könnyedén elérjük az alkalmazás szolgáltatásait.

Ellenben, ha az alkalmazás összekapcsolása esetén a `Mégsem` gombra kattintunk, a regisztráció meghiúsul és a felhasználó át lesz irányítva egy hibaoldalra.

Ha nem rendelkezünk Facebook fiókkal vagy csak nem szeretnénk használni, akkor az alkalmazás erre az esetre is biztosít egy lehetőséget. Az alkalmazáson hozzunk létre egy felhasználói fiókot, amivel a későbbiekben a felhasználónév és jelszó vagy e-mail és



4.2. ábra. Bejelentkezés Facebook felhasználói fiókkal

jelszó párossal be tudunk jelentkezni. Ehhez kattintsunk a `Regisztrálok` gombra, ahol a felhasználónév, e-mail cím és jelszó megadása után a `Regisztráció` gombra kattintva létrehozhatjuk a felhasználói fiókunkat. Ehhez a későbbiekben a Facebook fiókunkat is hozzákapcsolhatjuk.

Sikertelen bejelentkezés esetén az alábbi hibaüzenetek jelenhetnek meg:

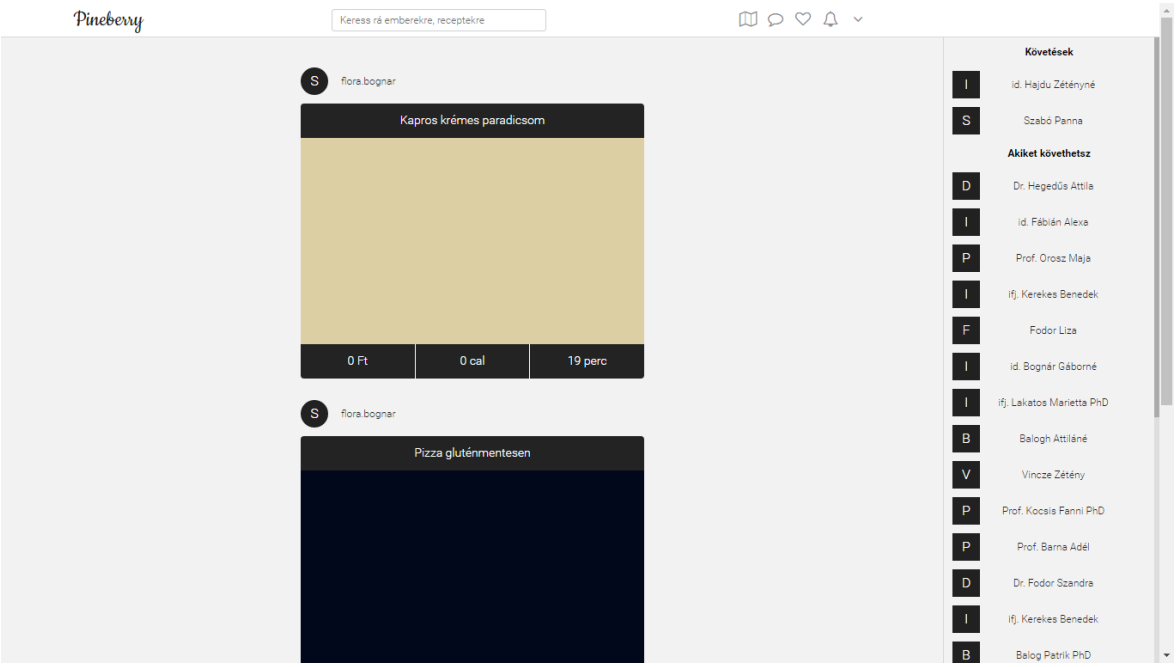
- *A felhasználóneved vagy e-mail címed megadása kötelező!*
- *A jelszavad megadása kötelező!*
- *Az adataid nem megfelelőek!*

Ha ezen üzenetek egyike sem jelenik meg, akkor a belépési adataink újbóli megadása során ellenőrizzük, hogy jól töltöttük ki az űrlapot. Ha a sikertelen bejelentkezés továbbra is fennáll, lehetőségünk van új jelszó igénylésére. Ehhez a láblécben található `Elfelejtett jelszó` hivatkozásra vagy a jelszó mezőtől jobbra található Elfelejtett hivatkozásra kell kattintani. Az e-mail cím megadásával egy üzenetet kap a felhasználó a korábban megadott e-mail címére, ahol a hivatkozásra kattintva létrehozhatja a fiókjához az új jelszavát.

A sikeres bejelentkezést követően egy üres kezdőoldal fogadja a felhasználót, ami a későbbiek során az általuk követett más felhasználók receptjeit fogja megjeleníteni, ahogy az az alábbi 4.3. ábrán is látható.

A jobb oldali sávban a követett és a követhető felhasználók listája jelenik meg. A követhető felhasználók véletlenszerűen jelennek meg, és csak azok a felhasználók, akik töltöttek fel már receptet. A sávban a névre vagy a név mellett megjelenő profilképre kattintva a felhasználó profilját érjük el.

A profil oldalon a felhasználó olyan nyilvános adatait tekinthetjük meg, mint például, hogy hány receptet töltött fel, mely recepteket kedvelte, kiket követ és őt kik követik. Ezen az oldalon tudjuk követni az adott felhasználót a `Követés` gombra kattintva, amit



4.3. ábra. Az alkalmazás kezdőoldala

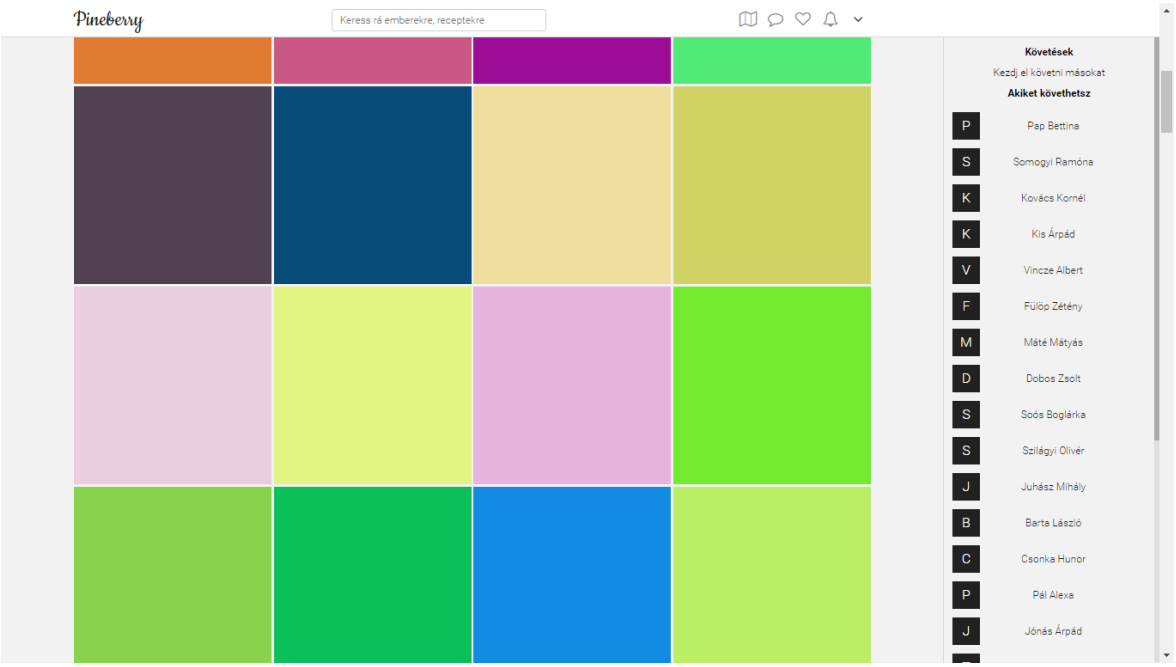
követően a jobb oldali sávban megjelenik a Követések alatt. A profil oldalon lehetőségünk van a felhasználó által feltöltött recepteket fényképesen is megtekinteni, ahol a képre kattintva elérjük a képhez tartozó receptet.

Lehetőségünk van receptek és felhasználók keresésére. Ehhez használjuk a fejlécben található keresőmezőt. A keresés során az első öt releváns találatot jeleníti meg különválasztva a felhasználó és a recept találatokat. A találat során a felhasználóra kattintva eljutunk annak profiljához, míg a receptre kattintva a recepthez. Ha a keresés során a *Nincs találat* üzenet jelenik meg, akkor a keresés során nem talált releváns felhasználót vagy receptet.

A keresőmezőtől jobbra található ikonok sorban a Felfedezés, a Kedvelési értesítő, a Követési értesítő és a felhasználói menü. A térkép ikonra kattintva megjelenik az összes feltöltött recept képe, amit a 4.4. ábrán láthatunk. A képre kattintva az adott receptet érhetjük el, ez a 4.5. ábrán látható. A recepteknél megjelenő tartalom a `Fejlesztői dokumentáció` fejezetben ismertetett generálási módon történik. A szív ikonra kattintva arról tudhatunk meg információt, hogy melyik receptünket kik kedvelték. Mellette jobbra található a harang ikon, ami a követőinkről ad tájékoztatást. A találatok megjelenítése során az utolsó öt eseményt jeleníti meg.

A lefele nyíl ikonra kattintva megjelenik a felhasználó által elérhető menü. Fentről lefelé haladva a Profil hivatkozásra kattintva a saját profilunkat érjük el. A saját profil oldalon lehetőségünk van megtekinteni a feltöltött receptjeinket, a kedveléseinket, a követéseket és a követőket. A fejlécben található Recept feltöltése és a Receptjeim gomb. A recept feltöltésre kattintva tudunk feltölteni receptet, a receptjeim gombra kattintva pedig elérni a feltöltött receptjeinket, amit a menüből szintén elérhetünk a Receptek hivatkozásra kattintva.

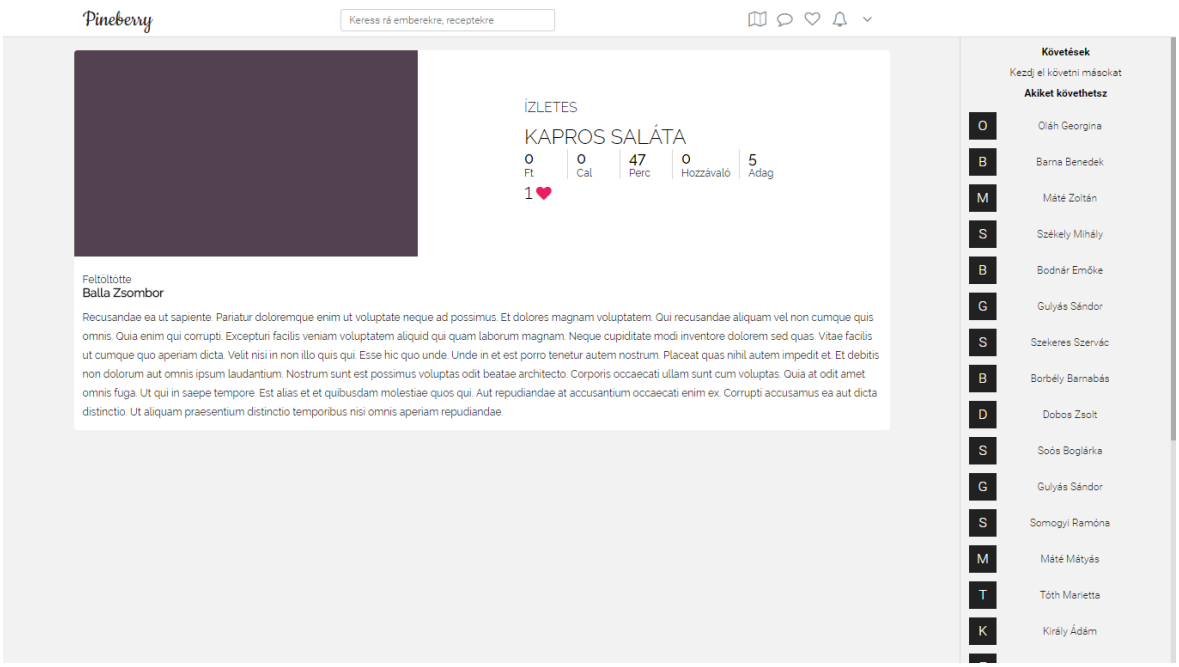
A receptek oldalon szintén lehetőségünk van megtekinteni a feltöltött receptjeinket, viszont ez abban különbözik a profil oldalon lévőtől, hogy itt lehetőségünk van a receptjeinket törölni illetve módosítani. Az eltávolítás ikonra kattintva a recept törlésre kerül, amit a későbbiekben nem vonhatunk vissza.



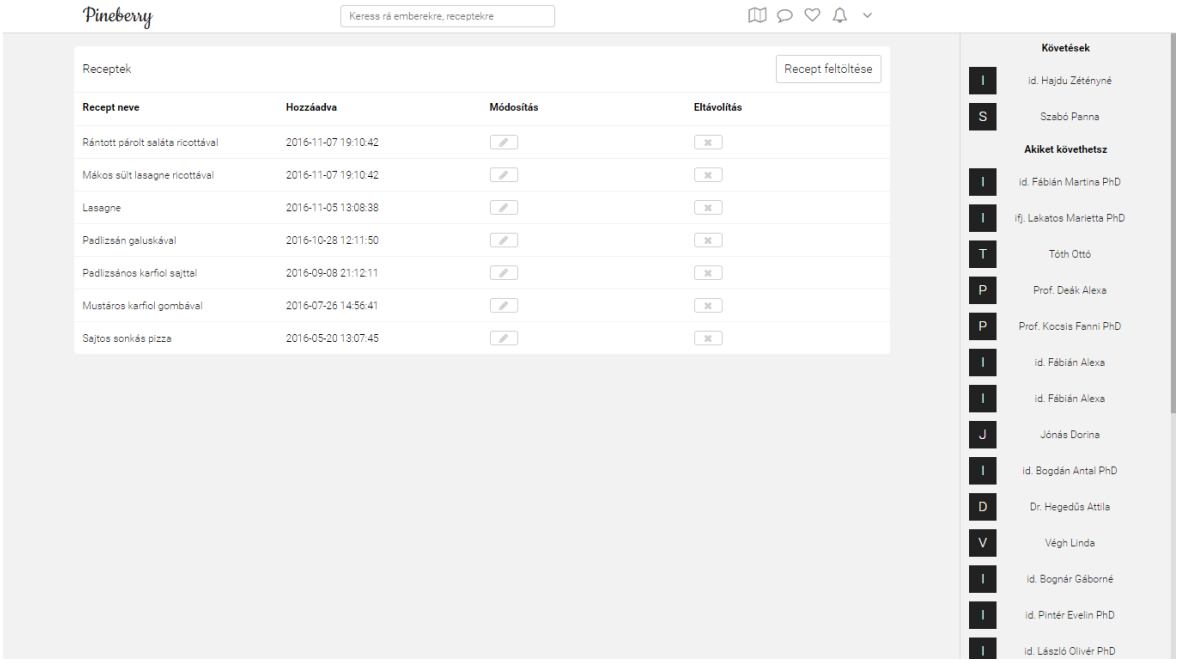
4.4. ábra. A felhasználók által feltöltött receptek képei

A *Általános* beállítások oldalon a fiók regisztrációtól függően lehetőségünk van a nevünk és a felhasználónevünk módosítására, illetve a fiókunk törlésére, amit a 4.8. ábrán láthatunk. A *Testtömegindex* oldalon a testmagasságunk és a testsúlyunk megadását követően, a *Statisztika* hivatkozásra kattintva egy diagramon láthatjuk az eredményt, ami a 4.9. látható. A diagram mindig az adott nap utolsó eredményét mutatja, így helytelen testmagasság és testsúly megadása esetén elég csak újból megadnunk azokat a helyes értékkel.

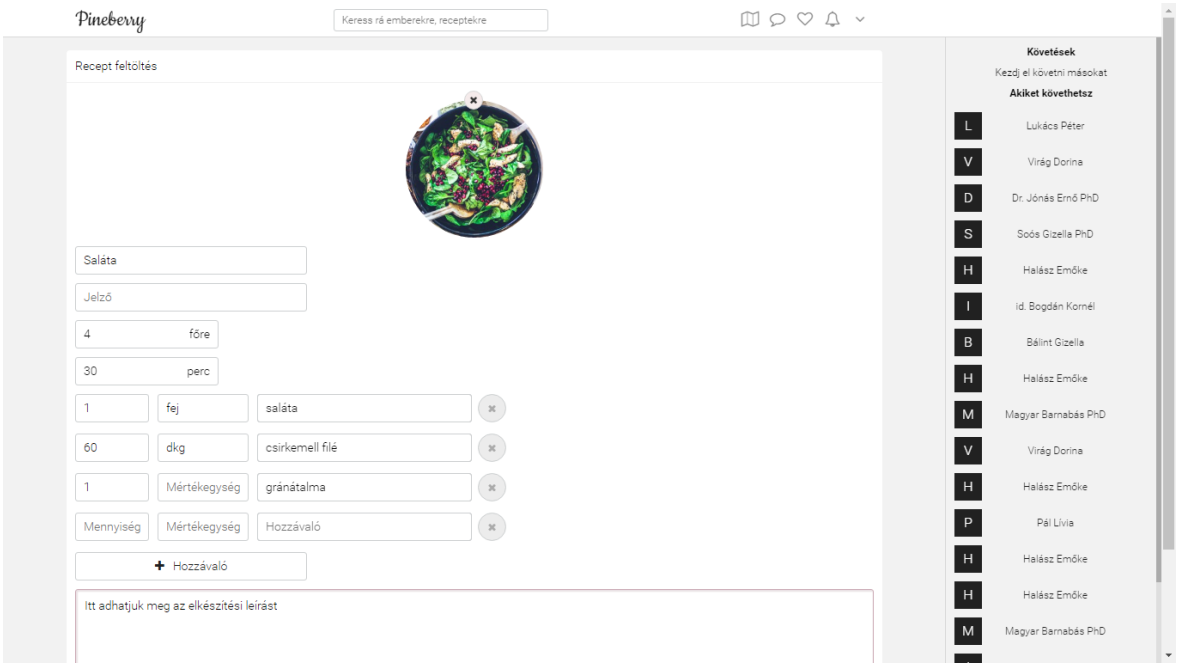
A megfelelő jogosultsággal rendelkező felhasználóknak lehetőségük van a statisztikai adatok megtekintésére, ami a 4.10., a 4.11. és a 4.12. ábrán látható.



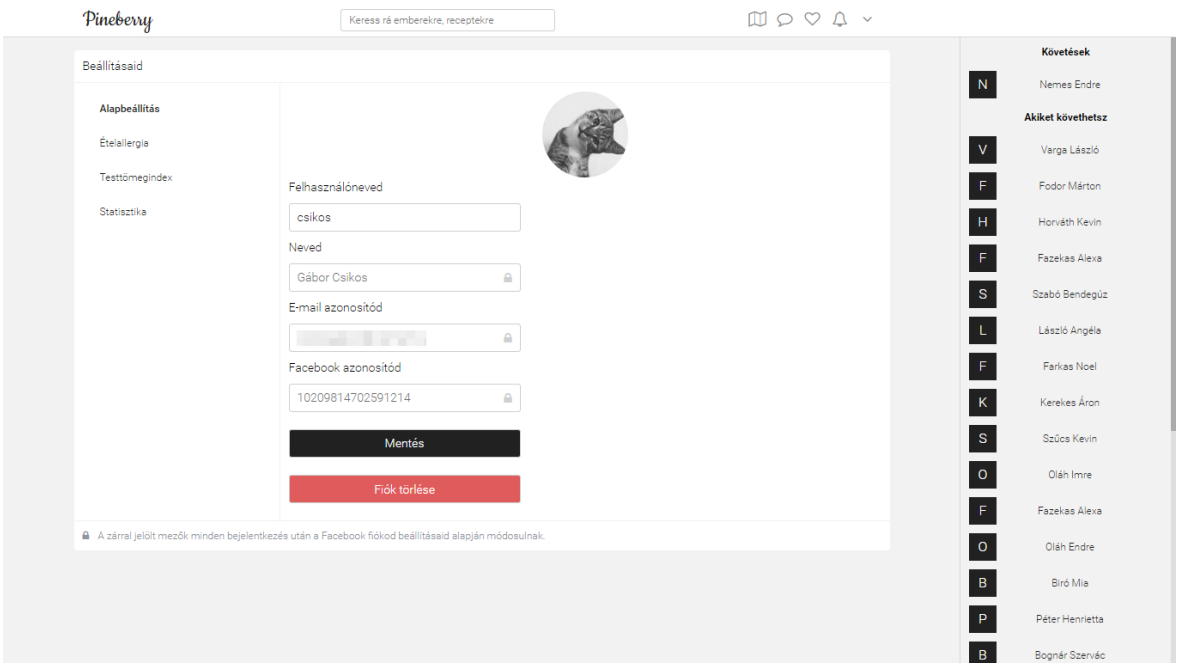
4.5. ábra. Recept oldal a recept részletes leírásával, a fontosabb adataival és az ételről készített képpel



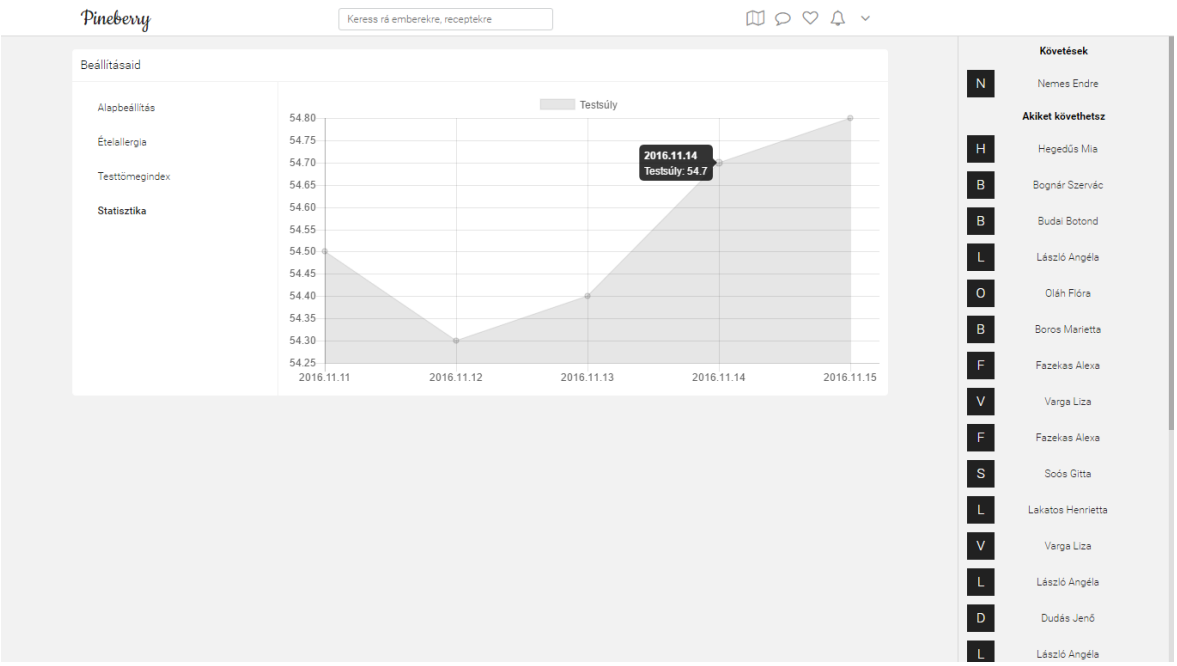
4.6. ábra. A felhasználó saját receptjeinek listája



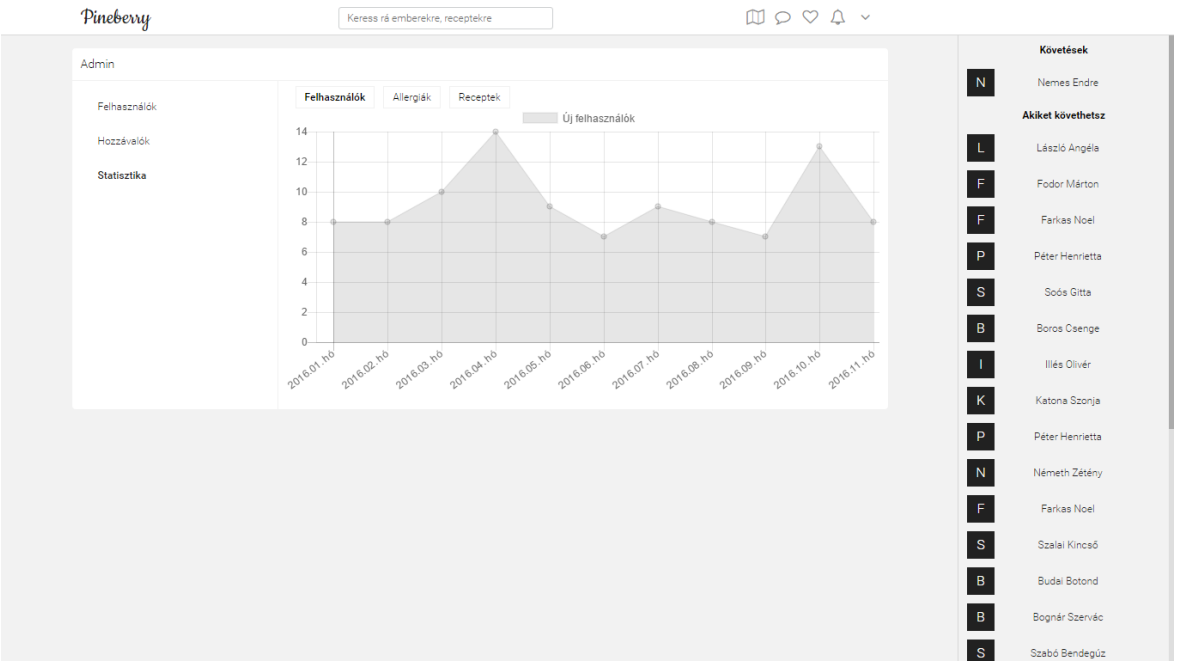
4.7. ábra. Recept feltöltő oldal a recept adatainak megadásához és a kép kiválasztásához



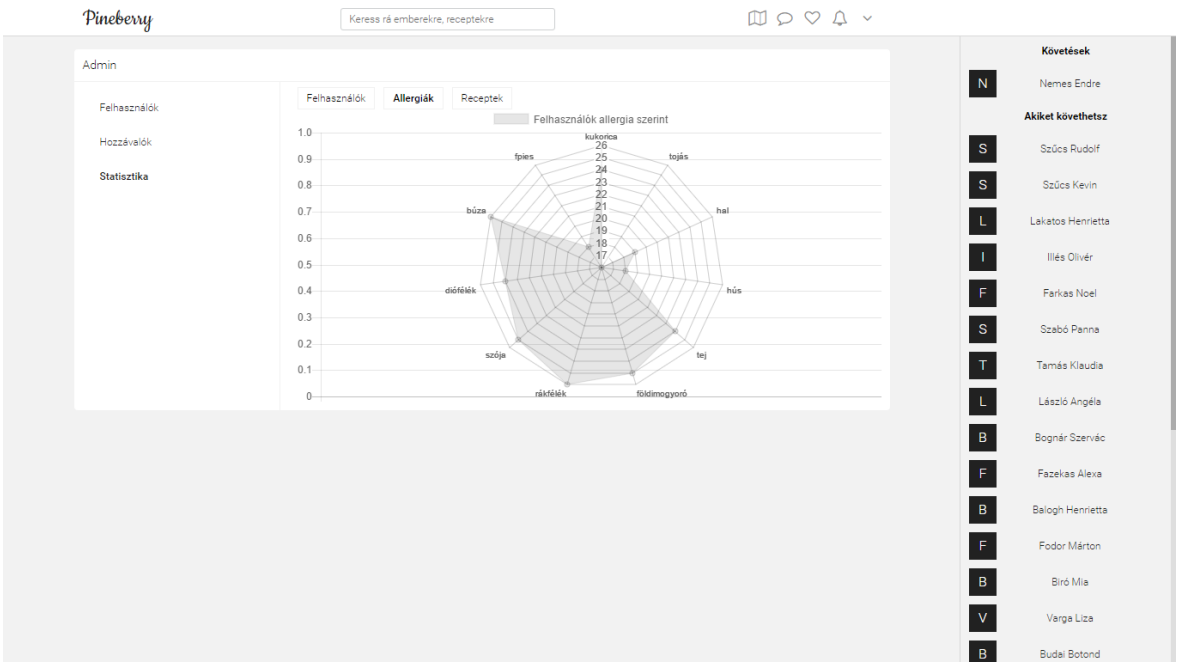
4.8. ábra. A felhasználói fiókhoz tartozó általános beállítások



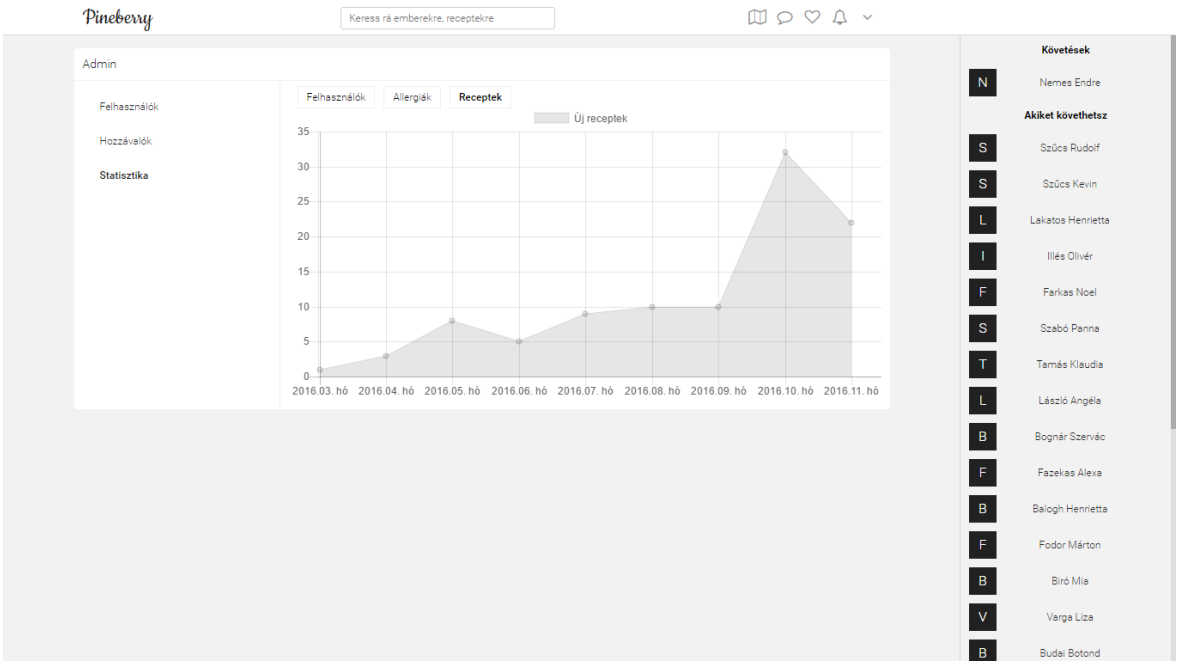
4.9. ábra. Statisztika a testtömegindex alakulásáról



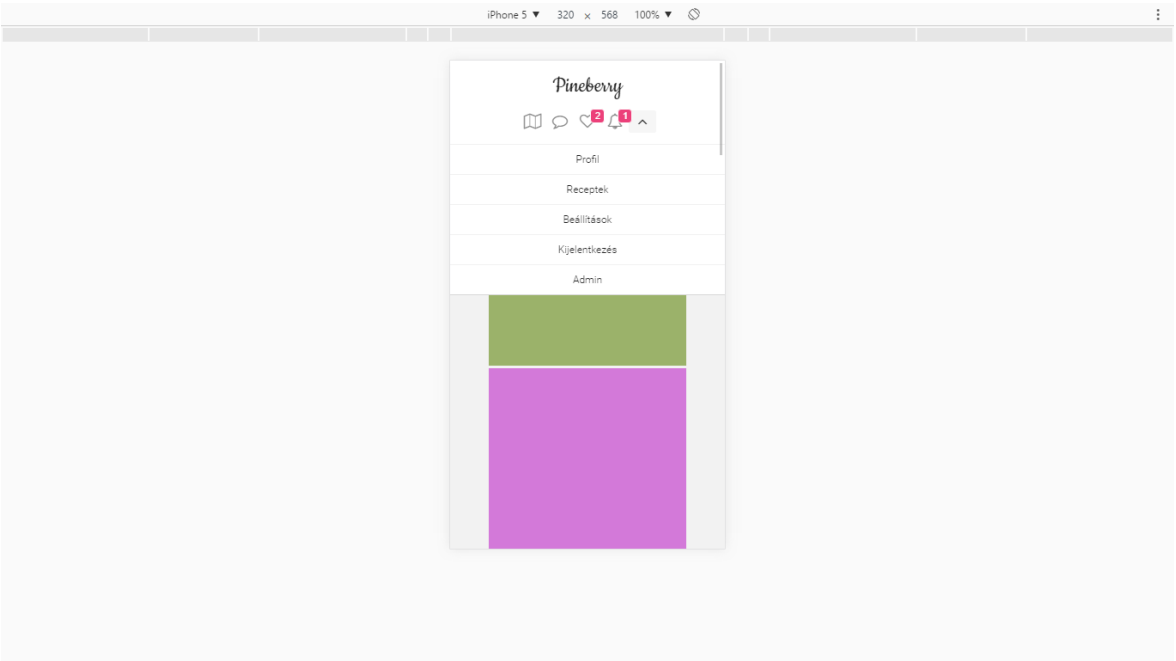
4.10. ábra. Az új regisztrációk száma havi bontásban



4.11. ábra. Az egyes allergiában szenvedő felhasználók száma



4.12. ábra. Az új receptek száma havi bontásban



4.13. ábra. Az alkalmazás reszponzív menüjének a megvalósítása

5. fejezet

Fejlesztői dokumentáció

5.1. Controller-ek

Ebben a fejezetben ismertetésre kerülnek az alkalmazás során használt Controller-ek, és ezen Controller-ek elérése szolgáló útvonal paraméterek, valamint az újraírt autentikációról és a tesztelési fázisról lesz szó. A tesztelés kapcsán kifejtésre kerül az e-mail és az adatbázis tesztelés, ahol utóbbinál a tesztelés fontossága mellett a tesztadatok generálásáról is olvashatunk, amelyek a későbbi fejlesztési folyamatok során a segítségünkre lesznek.

Az alkalmazás az alábbi táblázatokban szereplő útvonalak elérését teszi lehetővé. A *Method* oszlopban a G|H a GET|HEAD, a P a POST metódus rövidítése, az *URI* oszlopban az útvonal, a *Middleware* oszlopban az útvonalhoz tartozó szűrő, és az *Action* oszlopban az útvonalhoz tartozó jellel ellátott metódus található. Minden metódus az adott táblázat fejlécében található osztályhoz tartozik.

Az *AdminController* `chartDataAllergies`, `chartDataRecipes` és `chartDataUsers` metódusai által megkapjuk az ételallergiás felhasználók, a havi feltöltött receptek és a havi regisztrált felhasználók számát. A `setIngredientAllergy` metódussal beállíthatjuk az egyes hozzávalók ételallergiai besorolását, ami a felhasználói recept szűrés során fontos.

App\Http\Controllers\AdminController			
Method	URI	Action	Middleware
G|H	chart/data/allergies	@chartDataAllergies	web,auth,cert
G|H	chart/data/recipes	@chartDataRecipes	web,auth,cert
G|H	chart/data/users	@chartDataUsers	web,auth,cert
P	admin/ingredient/set	@setIngredientAllergy	

Az *AccountController* `destroy` metódusa a felhasználói fiók törlésére szolgál.

App\Http\Controllers\Auth\AccountController			
Method	URI	Action	Middleware
P	account/delete	@destroy	web,auth,cert

A *FacebookController* `login` metódus a Socialite által létrehozza az alkalmazás és a Facebook közötti kommunikációt. Sikeres kapcsolat létrehozását követően az adatokat a `callback` metódus fogja megkapni. A `destroy` a kapcsolat bontására szolgál.

App\Http\Controllers\Auth\FacebookController			
Method	URI	Action	Middleware
G|H	facebook-login-callback	@callback	web
P	facebook-destroy	@destroy	web,auth,cert
G|H	login/facebook	@login	web

A *ForgotPasswordController* `sendPasswordResetEmail` metódusa sikeres e-mail cím megadása esetén egy jelszó visszaállító üzenetet küld el a megadott e-mail címre.

App\Http\Controllers\Auth\ForgotPasswordController			
Method	URI	Action	Middleware
P	password/email	@sendPasswordResetEmail	web,guest

A *LoginController* `login` metódusa a felhasználónév/jelszó vagy e-mail/jelszó páros bejelentkezési lehetőségét biztosítja.

App\Http\Controllers\Auth\LoginController			
Method	URI	Action	Middleware
P	login	@login	web

A *NotificationController* `getFollowNotification` által a követési értesítéseket, a `getLikeNotification`-el pedig a kedvelési értesítéseket kapjuk meg. Ezen értesítések megtekintése során a `setFollowSeen` és a `setLikeSeen` az értesítések értékét beállítja igazra, így a későbbiek során nem fognak megjelenni az új értesítéseink között.

App\Http\Controllers\Auth\NotificationController			
Method	URI	Action	Middleware
G|H	notification/follow	@getFollowNotification	web,auth,cert
G|H	notification/like	@getLikeNotification	web,auth,cert
P	notification/follow/seen	@setFollowSeen	web,auth,cert
P	notification/like/seen	@setLikeSeen	web,auth,cert

A *RegistrationController* `createUser` a felhasználói fiók elkészítésére szolgál.

App\Http\Controllers\Auth\RegistrationController			
Method	URI	Action	Middleware
P	registration	@createUser	web

A *ResetPasswordController* `showResetForm` által kapjuk meg a jelszó visszaállító oldalt, ahol a `reset` metódus dolgozza fel a jelszó visszaállítási kérelmünket.

App\Http\Controllers\Auth\ResetPasswordController			
Method	URI	Action	Middleware
P	password/reset	@reset	web,guest
G|H	password/reset/token?	@showResetForm	web,guest

Az *IngredientController* `getIngredientById` által a jogosult felhasználók a migration által feltöltött hozzávalókat tekinthetik meg.

App\Http\Controllers\Collect\IngredientController			
Method	URI	Action	Middleware
G|H	admin/ingredient/id	@getIngredientById	web,auth,cert

A *ProductController* `index` metódusa a termék oldal nézetét adja vissza, ami a későbbi fejlesztésekbe fog bekerülni.

App\Http\Controllers\Collect\ProductController			
Method	URI	Action	Middleware
G|H	product	@index	web,auth,cert

A *RecipeController* a receptek kezelésére szolgál. Az `index` által a recept lista oldalt kapjuk meg, ahol a `destroy` által lehetőségünk van a recept törlésére is. A `getRecipeByID` a kérés során az átadott azonosító alapján adja vissza a receptet nézettel együtt. A `getRecipeIngredient` a migration és a felhasználók által hozzáadott hozzávalók együttesét adja vissza. A `createIndex` által a recept feltöltő oldalt kapjuk meg, és a `newRecipe` metódus által töltjük fel a receptünket az adatbázisba. Az `update` és `updateIndex` a recept frissítésére szolgáló nézetet adja vissza.

App\Http\Controllers\Collect\RecipeController			
Method	URI	Action	Middleware
G|H	recipe/create	@createIndex	web,auth,cert
P	recipe/delete	@destroy	web,auth,cert
G|H	recipe/recipe	@getRecipeByID	web,auth,cert
P	recipe/id/update	@getRecipeIngredient	web,auth,cert
G|H	recipe	@index	web,auth,cert
P	recipe/create	@newRecipe	web,auth,cert
P	recipe/update	@update	web,auth,cert
G|H	recipe/recipe/update	@updateIndex	web,auth,cert

A *ExploreController* `index` a felhasználók által feltöltött receptek nézetének az elkészítésére szolgál.

App\Http\Controllers\ExploreController			
Method	URI	Action	Middleware
G|H	explore	@index	web,auth,cert

A *FollowController*-ben a felhasználói követések be- és kikapcsolása valósul meg a `setFollow` és a `setUnfollow` metódusok által. A kezdőoldalon megjelenő recepteket a `getRecipesByFollow` metódus kéri le, melyek az általunk követett felhasználók közül kerülnek ki.

App\Http\Controllers\FollowController			
Method	URI	Action	Middleware
G|H	follow-recipes	@getRecipesByFollow	web,auth,cert
P	follow	@setFollow	web,auth,cert
P	unfollow	@setUnfollow	web,auth,cert

A *SearchController* a `getResult` metódus által a keresés során a találatnak megfelelő felhasználókat és recepteket adja vissza. A `getIngredients` a későbbi fejlesztések során a keresőmezőben megadott hozzávalók alapján fogja majd a találatokat visszaadni.

App\Http\Controllers\Fragment\SearchController			
Method	URI	Action	Middleware
G|H	ingredients	@getIngredients	web,auth,cert
G|H	search	@getResult	web,auth,cert

A *LeadController* `clean` metódusával lehetőségünk van kijelentkezni és törölni a munkameneteket, vagy visszaállíthatjuk a softDelete során törölt adatokat a `restore` által. A `mod`-al jogosultságot szerezhetünk és használhatjuk az *Admin* funkciókat, míg az `unmod`-al visszaadhatjuk a jogosultságot.

App\Http\Controllers\LeadController			
Method	URI	Action	Middleware
G|H	clean	@clean	web
G|H	mod	@mod	web
G|H	restore	@restore	web
G|H	unmod	@unmod	web

A *HomeController* generálja le az egyes útvonalak nézetét.

App\Http\Controllers\HomeController			
Method	URI	Action	Middleware
G|H	about-us	@getAboutUs	web
G|H	admin/analytics	@getAdmin	web,auth,cert
G|H	admin	@getAdmin	web,auth,cert
G|H	admin/users	@getAdmin	web,auth,cert
G|H	admin/ingredients	@getAdmin	web,auth,cert
G|H	account-delete	@getDeleteAccount	web,auth,cert
G|H	we-will-miss-you	@getDeleteSuccess	web
G|H	registration/facebook	@getFacebookRegistration	web
G|H	blog	@getHome	web
G|H	login	@getLogin	web
G|H	/	@getLogin	web,cert
G|H	logout	@getLogout	web,auth
G|H	privacy	@getPrivacy	web
G|H	registration	@getRegistration	web
G|H	password/reset	@getReset	web
G|H	account/body-mass-inde	@getSettings	web,auth,cert
G|H	account/food-allergy	@getSettings	web,auth,cert
G|H	account/general	@getSettings	web,auth,cert
G|H	account	@getSettings	web,auth,cert
G|H	account/analytics	@getSettings	web,auth,cert
G|H	registration/success	@getSuccessRegistration	web
G|H	password/reset/succes	@getSuccessReset	web
G|H	support	@getSupport	web
G|H	terms	@getTerms	web

A *LikeController* `setLike` által lehetőségünk van kedvelni egy receptet, míg a `setDislike` a korábban kedvelt recept állapotát állítja vissza alaphelyzetbe.

App\Http\Controllers\LikeController			
Method	URI	Action	Middleware
P	dislike	@setDislike	web,auth,cert
P	like	@setLike	web,auth,cert

A *PhotoController* `getOld` metódusa lehetővé teszi a korábban feltöltött kép azonosítója átadását abban az esetben, ha a recept feltöltése sikertelen volt. A képfeltöltést az `upload` metódus teszi lehetővé.

App\Http\Controllers\PhotoController			
Method	URI	Action	Middleware
P	photo/old	@getOld	web,auth,cert
P	photo/upload	@upload	web,auth,cert

A *SettingsController* `save` metódusa a felhasználói profil, a `setBodyMassIndex` testtömegindex és a `setFoodAllergy` az ételallergiák beállítására szolgál.

App\Http\Controllers\SettingsController			
Method	URI	Action	Middleware
P	account	@save	web,auth,cert
P	account/body-mass-index/set	@setBodyMassIndex	web,auth,cert
P	account/allergy/set	@setFoodAllergy	web,auth,cert

A *UserController* `chartDataBMIs` metódusa a testsúly, a `getFollowers` a követők, a `getFollowing` a követések és a `getProfile` a profil oldal adatait adják vissza. A felhasználói fiókok megerősítésére az `update` metódus áll rendelkezésre, ahol a regisztrációtól függően egy nevet vagy egy felhasználónevet kell megadni.

App\Http\Controllers\UserController			
Method	URI	Action	Middleware
G|H	chart/data/body-mass-indexes	@chartDataBMIs	web,auth,cert
G|H	user/user/followers	@getFollowers	web,auth,cert
G|H	user/user/following	@getFollowing	web,auth,cert
G|H	user/user/likes	@getLikes	web,auth,cert
G|H	user/user	@getProfile	web,auth,cert
P	confirm	@update	web,auth

Az alábbi *Closure* `setup` metódusa lehetővé teszi az adatok generálását, míg a `clear` ezen adatok kitakarítására szolgál. A `store` és a `report` a későbbi fejlesztések számára van fenntartva, míg a `test` metódus az aktuálisan fejlesztett programrész műveleteinek kipróbálására szolgál.

Method	URI	Action	Middleware
G|H	setup	Closure	web
G|H	clear	Closure	web
G|H	test	Closure	web
G|H	store	Closure	web,auth,cert
G|H	report	Closure	web,auth,cert

5.2. Bejelentkezés és regisztráció

Az alkalmazás használatához a felhasználónak rendelkeznie kell egy felhasználói fiókkal. A felhasználói fiók létrehozása kétféleképpen biztosított. A felhasználó a regisztrációs oldalon elkészítheti a fiókját név, e-mail cím és jelszó megadásával vagy használhatja a Laravel Socialite által biztosított Facebook bejelentkezési lehetőséget. Ez utóbbinál az első bejelentkezéskor a fiókja elkészül. A Laravel Socialite az OAuth autentikáció által egy tokent hoz létre a kiválasztott szolgáltatónál, úgy mint például a Facebook, Google, Twitter. Ezáltal az alkalmazás hozzáfér a felhasználó adataihoz, amit a szolgáltató megoszt.

Azok a felhasználók, akik nem rendelkeznek vagy nem szeretnék használni a külső szolgáltatón keresztüli bejelentkezési lehetőséget, azok számára egy e-mail/jelszó vagy egy felhasználónév/jelszó párossal való bejelentkezési lehetőség is biztosított. A felhasználó összekapcsolhatja az alkalmazás által létrehozott fiókját a Facebook fiókjával, ha a Facebook bejelentkezési lehetőséget választja.

A Laravel rendelkezik egy autentikációval, melyet a `php artisan make:auth` paranccsal tudunk igénybe venni. Mivel az alkalmazás bejelentkezési lehetősége eltér a Laravel által implementált autentikációtól, ezért saját autentikációs osztályt kellett készítenem. Ilyen esetben is támogatást nyújt a Laravel az *attempt* függvény által, ahol az átadott értékek alapján hitelesíti a felhasználót.

A bejelentkezésnél és a regisztrációnál is fontos szempont a felhasználó által megadott adatok hitelessége. Ezen adatokat ellenőrzésére a *Validator* osztály által meghívott trait-ek szolgálnak. Az alábbi trait-ben a bejelentkezés során használt `validateLogin` metódus látható, ami ellenőrzi a felhasználó által megadott felhasználónév/e-mail cím és jelszó párost.

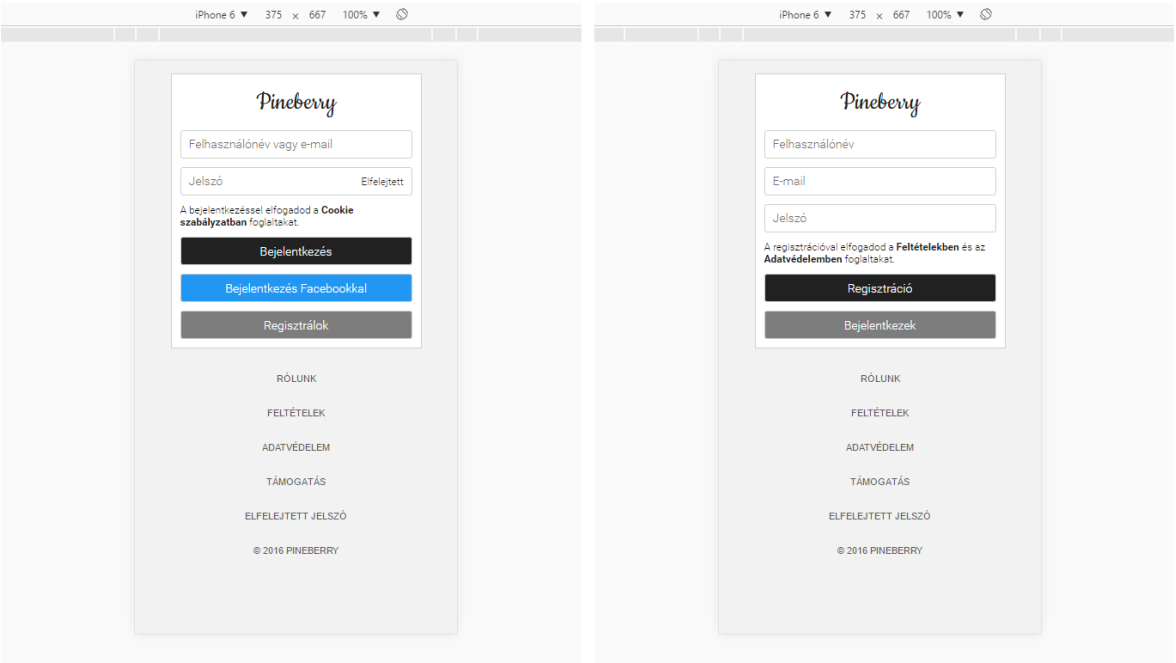
```
/**
 * app\Http\Controllers\Validation\UserValidation.php
 */

<?php

namespace App\Http\Controllers\Validation;

use Illuminate\Http\Request;

use Validator;

trait UserValidation {

    public static function validateLogin(Request $request) {

        $input = [
            'user'     => $request->user,
            'password' => $request->password
        ];

        $rules = [
            'user'     => 'required',
            'password' => 'required'
        ];

        $messages = [
            'user.required'     => 'A felhasználóneved vagy e-mail
```

```
                                    címed megadása kötelező',
            'password.required' => 'A jelszavad megadása kötelező'
        ];

        return Validator::make($input, $rules, $messages);
    }
}
```

A trait meghívásához használjuk a *use* operátort a meghívandó osztályon belül. Mivel egy osztály csak egy osztályt örökölhet, így a trait-ek által lehetőségünk van a trait-ben meghatározott metódusokat egy osztályon belül felhasználni.

```
/**
 * app\Http\Controllers\Validator.php
 */

<?php

namespace App\Http\Controllers;

class Validator {

    use Validation\UserValidation;
}
```

Miután a felhasználó bejelentkezett választania kell egy nevet vagy egy felhasználónevet. A választás módja függ attól, hogy melyik bejelentkezési lehetőséget választotta előzőleg. Ha a felhasználó a Facebook bejelentkezést választotta, akkor egy felhasználónevet kell megadnia, egyébként pedig a nevét. Ennél a bejelentkezési lehetőségnél a Facebook biztonsági szempontból nem adja ki a felhasználóneveket, ezért az alkalmazásba regisztrált felhasználókat külön kell megkérni, hogy válasszanak egy számukra tetszőleges felhasználónevet.

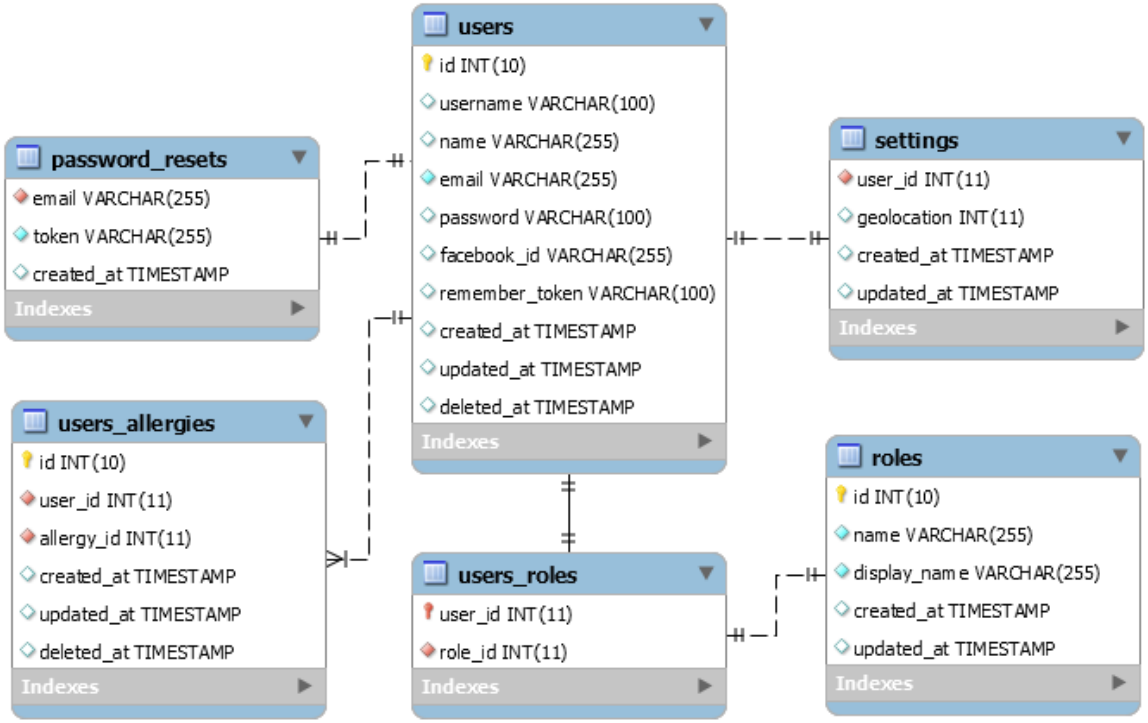


5.1. ábra. Autentikáció - Developer Tools - Google Chrome

5.3. Az adatbázis megtervezése

A felhasználók esetében az ételallergiát okozó hozzávalók szűrésénél lényeges feladat volt az adatbázis modelljének megtervezése. A modell elkészítését követően, a modell alapján írt SQL kód segítségével le tudtam kérdezni a megfelelő rekordokat. Ezt követően az SQL kódot implementálva és kiegészítve, a felhasználó azokat a recepteket kapta vissza, amelyek összetevőire nem allergiás.

A tervezés első lépése volt a felhasználók adatainak a tárolása, amit a *users* tábla tartalmaz. A *users* táblához 3 tábla kapcsolódik közvetlenül és 1 közvetetten. Minden felhasználó rendelkezik alapbeállításokkal amit a *settings* táblában tárolunk. A *roles* tábla, ami tartalmazza a felhasználók jogosultságait, közvetlenül kapcsolódik a *users* táblához a *users_ roles* kapcsoló tábla által. Ez a kapcsolótábla tárolja, hogy melyik felhasználót milyen jogosultsághoz rendelünk. A felhasználóknak lehetőségük van jelszavuk visszaállítására, amely során a *password_ resets* táblában tárol le az alkalmazás, illetve minden felhasználó beállíthatja, hogy a felsorolt 11 féle ételallergia közül melyik befolyásolja az étkezését. A felhasználó réteg modellje az 5.2. ábrán látható.

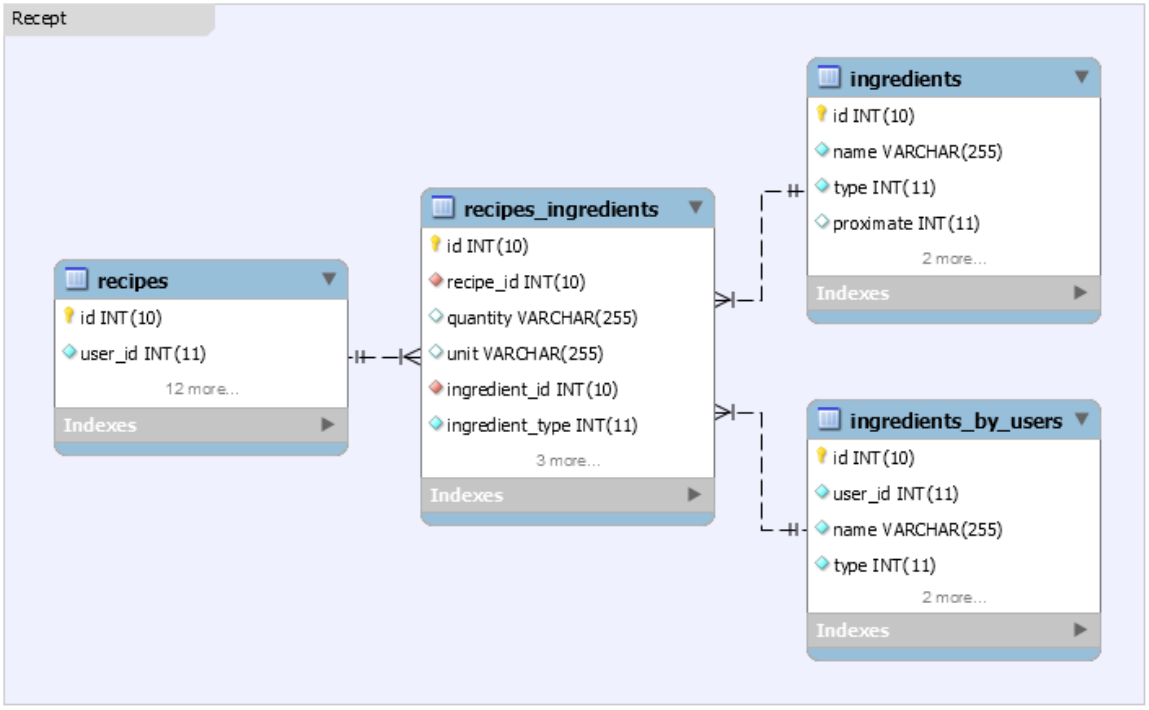


5.2. ábra. A felhasználó réteg EER modelle

Minden felhasználónak lehetősége van a receptek megosztására, amiknek az adatait a *recipes* tábla tárolja. A receptek hozzávalóit az *ingredients* és az *ingredients_ by_ users* táblák tartalmazzák. Az *ingredients* táblában azok a hozzávalók vannak, amiket a `database seed` során adtunk hozzá, míg az *ingredients_ by_ users* tábla a felhasználók által felvitt hozzávalókat tartalmazza.

A hozzávalók külön táblában való tárolását a későbbi fejlesztések miatt választottam. A recept feltöltés során egy hozzávaló megadásánál a hozzávaló automatikusan kiegészül, ami ebből a táblából keresi ki a feltételnek megfelelő szót. A másik előnye, hogy az eltérő dialektusok és a későbbi nyelvesítés során könnyebben lehet áttervezni a modellt.

A recept rétegen belül a receptek és a hozzávalók kapcsolatát a *recipes_ ingredients* tábla tartalmazza. Ahhoz, hogy a megfelelő táblából kérje le a hozzávalót, el kellett látni egy további mezővel (*ingredient_ type*), ahol az 1-es jelzi az *ingredients* táblát, míg a 2-es az *ingredients_ by_ users* táblát. A modell az 5.3. ábrán látható.



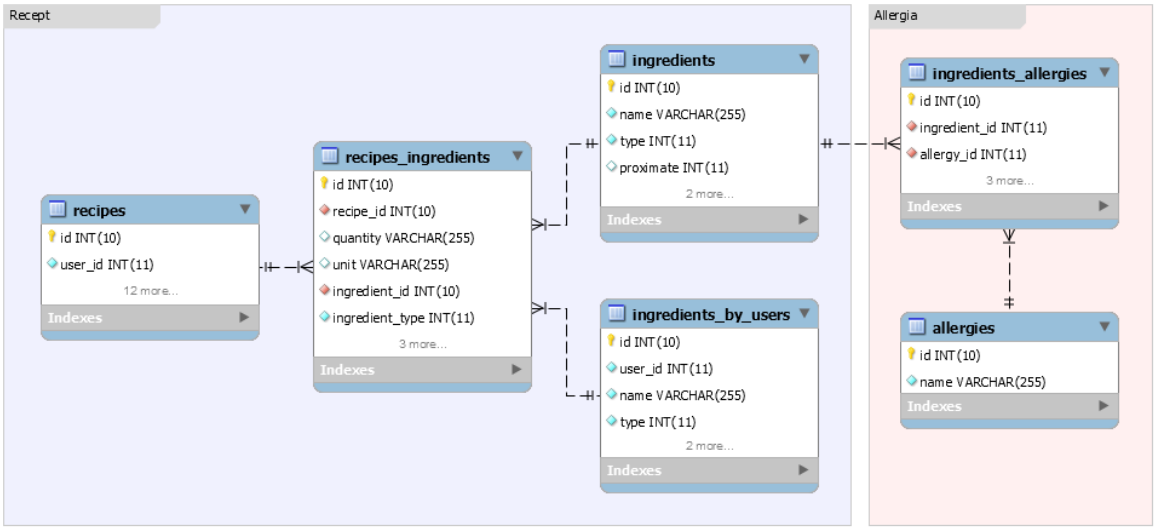
5.3. ábra. A recept réteg EER modelle

Az előző recept réteghez kapcsoljuk hozzá az allergia réteget az *ingredients* és az *ingredients_ allergies* táblák segítségével, ahol az allergia rétegben az *allergies* tábla az allergiákat, az *ingredients_ allergies* tábla a hozzávalók és allergiák kapcsolatát tartalmazza. Az 5.4. ábrán a recept és az allergia réteg kapcsolata látható.

Ehhez az allergia réteghez kapcsoljuk hozzá az előbb bemutatott felhasználó réteget. Az 5.5. ábrán a 3 réteg kapcsolata látható, ahol a felhasználó rétegnek csak egy részét ábrázoltam.

Az EER modell alapján az alábbi SQL lekérdezéssel visszakaptam azon recepteket, amelyekre az adott felhasználó allergiás.

```
SELECT * FROM `recipes`
INNER JOIN recipes_ingredients
    ON recipes.id = recipes_ingredients.recipe_id
INNER JOIN `ingredients`
    ON recipes_ingredients.ingredient_id = ingredients.id
INNER JOIN `ingredients_allergies`
    ON ingredients.id = ingredients_allergies.ingredient_id
INNER JOIN `allergies`
    ON allergies.id = ingredients_allergies.allergy_id
INNER JOIN `users_allergies`
    ON allergies.id = users_allergies.allergy_id
INNER JOIN `users`
```



5.4. ábra. Az allergia réteg EER modelle

5.5. ábra. A teljes allergia EER modell

```
    ON users_allergies.user_id = users.id
WHERE recipes_ingredients.ingredient_type = 1
AND users.id = 101
GROUP BY recipes.id
```

Ennek megfelelően került megvalósításra az SQL kód az alkalmazásban, ahol a *join* a kapcsolt tábla nevét és a kulcsokat, a *where* a feltételt és a *grouBy* tagfüggvény a csoportosítási feltételt kapta meg.

```
$recipe_allergy =
Recipe::join('recipes_ingredients as ri', 'recipes.id', 'ri.recipe_id')
    ->join('ingredients as i', 'ri.ingredient_id', 'i.id')
    ->join('ingredients_allergies as ia', 'i.id', 'ia.ingredient_id')
    ->join('allergies as a', 'a.id', 'ia.allergy_id')
    ->join('users_allergies as ua', 'a.id', 'ua.allergy_id')
    ->join('users as u', 'ua.user_id', 'u.id')
    ->where('ri.ingredient_type', 1)
    ->where('u.id', Auth::user()->id)
    ->select('recipes.id')
    ->groupBy('recipes.id')->get();
```

Mivel a lekérdezés csak azokat a recepteket adta vissza amelyekre a felhasználó allergiás, ezért készítenem kellett egy másik lekérdezést is, ezáltal azokat a recepteket adta vissza, amelyek hozzávalóira a felhasználó nem allergiás. A *Recipe* modell lekérdezése során a *whereNotIn* tagfüggvény biztosította, hogy a lekérdezés ne tartalmazza azokat a recepteket, amelyeket az első lekérdezés során kaptunk.

```
$data = Recipe::whereNotIn('id', $recipe_allergy)
    ->orderBy('created_at', 'desc');
```

5.4. Adatbázis tesztelés

Az alkalmazás tesztelése során elengedhetetlen lépés az adatbázis feltöltése a megfelelő adatokkal. Ilyen esetekben derül ki, hogy a tábláink megfelelnek-e a céloknak, valamint az oszlopaink a megfelelő adattípussal rendelkeznek-e, továbbá hogy az alkalmazásunk tud-e kapcsolódni az adatbázishoz.

A következő részben az adatok generálásáról lesz szó, amely a Laravel *model factory*, a Faker PHP könyvtár és az Artisan együttes használatát mutatja be véletlenszerű adatok generálására.

5.4.1. Adat szinkronizáció

A *factory* elkészítésénél elsődleges szempont volt, hogy az adatok szinkronban legyenek, vagyis egy recept generálása folyamán csak olyan felhasználói azonosítót (*id*) válasszon, ami a `users` táblában megtalálható, valamint az időpontot is ehhez igazítsa. Más szóval a generálás során a recept feltöltési időpontját, a felhasználó regisztráció időpontja és a jelenlegi időpont alapján véletlenszerűen határozza meg.

5.4.2. Az adatfeltöltés ismertetése

Ahhoz, hogy ne kelljen egyesével feltöltenünk az adatbázist, a Laravel lehetővé teszi a *model factory*-k használatát. Az alábbi programkód a *RecipeFactory* működését mutatja be, ami által a recept táblánkat töltjük fel.

```
/**
 * database\factories\RecipeFactory.php
 */

<?php

use Carbon\Carbon;
use Faker\Generator;
use App\Models\User;
use App\Models\Photo;
use App\Models\Recipe;
use App\Http\Controllers\Faker\Provider\hu_HU\Recipe as RecipeProvider;

$factory->define(Recipe::class, function (Generator $faker) {

    $faker->addProvider(new RecipeProvider($faker));

    $user_count = User::count();
```

```
    $photo_count = Photo::count();

    $user_id = $faker->numberBetween(1, $user_count);

    $user = User::find($user_id);
    $startDate = $user->created_at;
    $endDate = Carbon::now();
    $timezone = 'Europe/Budapest';
    $dateTime = $faker->dateTimeBetween($startDate, $endDate, $timezone);

    return [
        'user_id'     => $user->id,
        'name'        => $faker->recipeName,
        'modifier'    => $faker->recipeModifier,
        'photo_id'    => $faker->numberBetween(1, $photo_count),
        'description' => $faker->text($maxNbChars=1000),
        'serve'       => $faker->numberBetween(1, 8),
        'cost'        => 0,
        'cal'         => 0,
        'time'        => $faker->numberBetween(1, 100),
        'checked'     => 0,
        'created_at'  => $dateTime,
        'updated_at'  => $dateTime
    ];
});
```

Ahhoz, hogy jobban megértsük a *RecipeFactory* működését bontsuk fel a kódot részekre és ez alapján vizsgáljuk meg az egyes lépéseket. A *use* operátorral importáljuk a névtereket, amelyek közül az utolsó névteret a `RecipeProvider` álnévvel látunk el. A *Provider* egy osztály, amely a `Faker\Provider\Base` osztálynak a kiterjesztése. Egy *Provider* létrehozása során ügyelnünk kell arra, hogy a formázóknak (formatters) nevei eltérőek legyenek a Faker előre definiált formázóitól, különben felülírjuk őket. Például adott a `Faker\Provider\Person::name()` amit a `$faker->name`-el hívhatunk meg. Abban az esetben, ha létrehozunk egy új *Provider*-t névtől függetlenül, ami szintén tartalmaz egy name metódust, akkor a `$faker->name` meghívása esetén a `Person::name()` helyett ezen új *Provider* name metódusa lesz végrehajtva.

A *define* metódusnak átadunk egy osztályt valamint egy callback-et, másnéven egy *Closure*-t (név nélküli függvényt). Osztály esetén a *Recipe* modellt adjuk át, míg Closure esetén a Faker egy példányát a *Generator* osztályt. Ez által lehetőségünk lesz a Faker alap formázóinak a használatára. Mivel a `Faker\Generator` nem képes az általunk meghatározott formázók alapján az adatok generálására, ezért hozzá kell adnunk a `faker` változóhoz a *Provider*-ünket, aminek a segítségével lehetőségünk lesz a saját formázók használatára. Jelen esetben az `addProvider` metódust meghívva a *RecipeProvider*-t átadjuk a *Generator*-nak, ami által képesek leszünk használni a saját formázóinkat is.

```
$factory->define(Recipe::class, function (Generator $faker) {

    $faker->addProvider(new RecipeProvider($faker));

});
```

A következő részben a változóinkat határozzuk meg, amikkel majd vissza fogunk térni. Mivel a recepteket csak regisztrált felhasználók tölthetnek fel és minden recept

tartalmaz egy fényképet, ezért lényegesen ügyelni kell arra, hogy ezen felhasználók és fényképek is létezzenek. A *RecipeFactory* osztályunk minden `factory(Recipe::class)->create()` hívás esetén lefut, ezáltal lehetőségünk van egy-egy meghívás során meghatározni a jelenlegi felhasználó és fénykép darabszámot. A Laravel az Eloquent modellhez a `count` metódus által támogatást ad a count aggregáció használatára, ami által meghatározhatjuk a darabszámot. A *User* és a *Photo* egy Eloquent modellnek felel meg, így a count aggregációt szabadon alkalmazhatjuk.

A felhasználói azonosító kiszámításához a Faker a `numberBetween` formázója által generál egy számot 1 és a jelenlegi felhasználó szám között. Ezt követően a *User* modellben megkeressük a korábban generált felhasználót az azonosítója alapján. Ennek az eredménye egy *User* objektum lesz. A `created_at` adattag által lehetőségünk van elérni ezen objektum létrehozásának a dátumát, amely majd a recept feltöltési idejének a meghatározásához lesz szükséges.

A Carbon DateTime PHP API bővítmény által lehetőségünk van az idő és a dátum generálására. A Carbon `now` metódusa által létrehozza a jelenlegi időt, amit a későbbiek során felhasználunk. A `Carbon::now()` helyett lehetőségünk van a Carbon osztály példányosítására is, aminek az értéke meg fog egyezni az előbbi `Carbon::now()`-al. A Faker `dateTimeBetween` metódusa egy kezdődátumot, egy végdátumot és egy időzóna paramétereket vár. Mivel a kezdő- és a végdátumot már korábban meghatároztuk, így az időzóna maradt hátra, aminek a meghatározásához csak át kell adnunk egy PHP által támogatott időzóna karakterláncot.

```
$user_count = User::count();
$photo_count = Photo::count();

$user_id = $faker->numberBetween(1, $user_count);

$user = User::find($user_id);
$startDate = $user->created_at;
$endDate = Carbon::now();
$timezone = 'Europe/Budapest';
$dateTime = $faker->dateTimeBetween($startDate, $endDate, $timezone);
```

Miután ezzel megvagyunk, a Closure számára át kell adnunk egy visszatérési tömböt. A *user_id*-t a korábban kapott *User* modell alapján a `$user->id` adattaggal határozzuk meg. A saját *RecipeProvider* által meghatározzuk a *name* és a *modifier* kulcsot, amely a `recipeName` és a `recipeModifier` metódus szerint lesz generálva. A *photo_id*-nek és a *serve*-nek a korábban is használt Faker `numberBetween` metódusa alapján generált véletlen számot adunk át, ahol egy kezdő és egy végparamétert vár, míg a *description*-nek egy Lorem Ipsum szöveget generálunk a paraméterben átadott karakterszám szerint.

```
return [
    'user_id'     => $user->id,
    'name'        => $faker->recipeName,
    'modifier'    => $faker->recipeModifier,
    'photo_id'    => $faker->numberBetween(1, $photo_count),
    'description' => $faker->text($maxNbChars=1000),
    'serve'       => $faker->numberBetween(1, 8),
    'cost'        => 0,
    'cal'         => 0,
    'time'        => $faker->numberBetween(1, 100),
    'checked'     => 0,
```

```
        'created_at'  => $dateTime,
        'updated_at'  => $dateTime
    ];
```

A következőkben részletesen megnézzük, hogy hogyan használhatjuk fel a *RecipeFactory* kódunkat, a `factory` metódus használatától kezdődően egy terminál parancs elkészítésével bezárólag.

A Laravel `factory` metódusa lehetővé teszi, hogy átadjunk a *FactoryBuilder*-nek egy modellt, így előkészítve a modellünket a generálási folyamatra. A következő kódrészletben a `factory` metódusnak átadunk egy *User* modellt és elmentjük a `user` változóba. A `dd` metódus által kiírjuk a `user` változó értékét és a további végrehajtási folyamatokat megszakítjuk.

```
public function factoryTest() {

    $user = factory(App\Models\User::class);

    dd($user);
}
```

Hogy generálni tudjunk egy véletlenszerű adatot, a `factory` metódusunkhoz hozzá kell láncolni egy paraméter nélküli `make` metódust.

```
public function factoryTest() {

    $user = factory(App\Models\User::class)->make();

    dd($user);
}
```

A `make` metódus használata akkor lenne a megfelelő számunkra, hogy ha az adott függvényen belül használnánk fel az így generált adatokat. Ahhoz, hogy elmentsük az adatbázisunkba a `make` helyett használnunk kell a `create` metódust, továbbá megadjuk hogy hány példányt hozzon létre. A következő kódrészletben létrehozunk 10 véletlenszerű felhasználót és elmentjük az adatbázisunkba.

```
public function factoryTest() {

    $user = factory(App\Models\User::class, 10)->create();

    dd($user);
}
```

Most, hogy már tudjuk hogyan kell hamis adatokat generálni át is adhatnánk a függvényünket egy útvonalnak (route), hogy generáljon számunkra ezer felhasználót.

```
Route::get('test', function() {

    $user = factory(App\Models\User::class, 1000)->create();

});
```

Annak ellenére, hogy ez a megoldás jónak tűnhet, több hátránya is van. Amellett, hogy a kliens oldalt leterheli, bármelyik pillanatban megszakadhat a felhasználók generálása, és amellett hogy nem látjuk hol tart a folyamat a PHP végrehajtási idő is

általában korlátozva van. Ebből az okból a tesztelés során készítenem kellett egy olyan osztályt, amely lehetővé teszi a konzolos parancsok használatát, és a parancs használata során meghívja az adott osztályt, ami a elvégzi a műveletet. A következő kód a *FakerSeed* osztály egy részlete, amit az előző *RecipeFactory*-hoz hasonlóan szintén végigveszünk a korábban ismertetett *factory* használatával együtt.

```
/**
 * app\Console\Commands\FakerSeed.php
 */

<?php

namespace App\Console\Commands;

use Illuminate\Console\Command;
use Symfony\Component\Console\Helper\ProgressBar;

use App\Models\Photo;
use App\Models\Recipe;

class FakerSeed extends Command {

    protected $signature = 'faker:seed';

    protected $description = 'Seed the database with random records';

    public function __construct() {

        parent::__construct();
    }

    public function handle() {

        $this->steps = 1;

        $i = 0;
        $units = 1000;

        $progress = $this->output->createProgressBar($units);
        $progress->setFormat('%bar% %current%/%max%');

        $progress->start();
        $progress->setRedrawFrequency(1);

        while ($i++ < $units) {

            $photo = factory(Photo::class, $this->steps)->create();

            $recipe = factory(Recipe::class, $this->steps)->create([
                'user_id' => $photo->user_id,
                'photo_id' => $photo->id
            ]);

            $progress->advance();
        }

        $progress->finish();
        $this->line('');
```

```
    }
}
```

A *namespace* által meghatározzuk, hogy a fájlunk az `App\Console\Commands` útvonalon érhető el. A *use* által importáljuk a névtereket, és az egyiket el is látunk egy álnévvel. A `use DB, File` szintaktikailag helyes, mivel ott két eltérő névteret importálunk. A következőkben elkészítjük a *FakerSeed* parancsunkat. Egy konzol parancs létrehozása során használhatjuk a `php artisan make:command FakerSeed` parancsot a terminálon keresztül. A parancs használata során létrejön az `app\Console\Commands` könyvtár és azon belül a `FakerSeed.php`. Mivel a létrehozáskor számos felesleges adattal és hozzászólással tölti ki, így egyszerűbb kimásolni a fontosabb részeket a `https://laravel.com/docs/5.3/artisan#command-structure` oldalról. A következő kódrészlet a minta alapján a *FakerSeed* osztály egy részletét tartalmazza.

```
class FakerSeed extends Command {

    protected $signature = 'faker:seed';

    protected $description = 'Seed the database with random records';

    public function __construct() {

        parent::__construct();
    }

    public function handle() {

    }
}
```

A *FakerSeed* osztályunk a *Command* osztályból fog származni, amely egy kiterjesztett Symfony komponens. A `signature` változónak egy konzol parancsot, míg a `description` változónak egy, a parancshoz tartozó leírást adunk, és beállítjuk a láthatósági szintet `protected`-re. Az utána lévő `__construct` metódus fogja beállítani a megadott paramétereket és fogja meghívni a `handle` metódust. A `handle` metódusban állítjuk be, hogy a megadott parancs alapján milyen folyamatot végezzen el. Ezután megadjuk a korábban bemutatott `factory` metódust, és a `$this->steps` változóban eltároljuk a lépések számát, vagyis hogy hány rekordot generáljon.

```
public function handle() {

    $this->steps = 1000;

    $photo = factory(Photo::class, $this->steps)->create();

    $recipe = factory(Recipe::class, $this->steps)->create([
        'user_id' => $photo->user_id,
        'photo_id' => $photo->id
    ]);
}
```

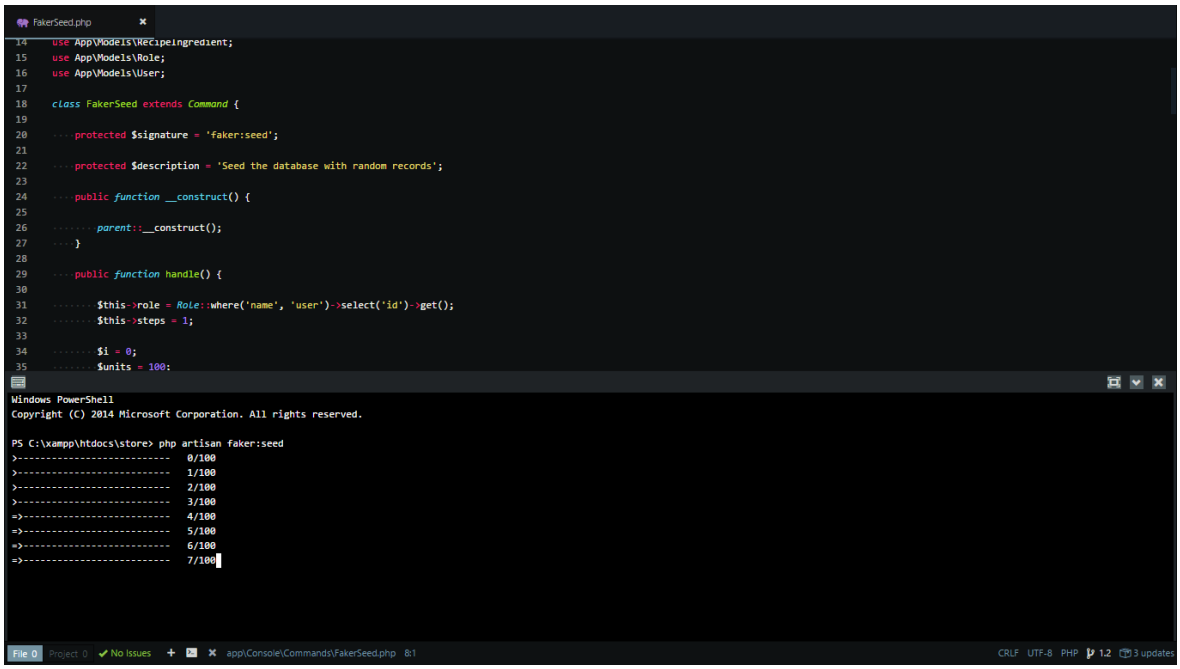
Mivel a futtatás során a megoldás nem mutatta, hogy hol tart a folyamat, ezért a Symfony *ProgressBar* komponensét felhasználva a kódot a következőkkel egészítettem ki.

```
public function handle() {

    $this->steps = 1;

    $i = 0;
    $units = 1000;

    $progress = $this->output->createProgressBar($units);
    $progress->setFormat('%bar% %current%/%max%');

    $progress->start();
    $progress->setRedrawFrequency(1);

    while ($i++ < $units) {

        $photo = factory(Photo::class, $this->steps)->create();

        $recipe = factory(Recipe::class, $this->steps)->create([
            'user_id' => $photo->user_id,
            'photo_id' => $photo->id
        ]);

        $progress->advance();
    }

    $progress->finish();
    $this->line('');
}
```

A kezdőértéket beállítjuk 0-ra és a `unit`-nak átadjuk a létrehozandó rekord számot. A `createProgressBar` tagfüggvénynek átadjuk a `units` értékét és láncoljuk a `$this->output` adattaghoz. A `setFormat` tagfüggvénnyel beállítjuk a formátumot és `start` tagfüggvénnyel elindítjuk és megjelenítjük a folyamat sávunkat, ahol a frekvenciát 1-re állítjuk be. Így minden rekord feltöltésnél megjeleníti, hogy éppen hol tart a folyamat. Az `advance` tagfüggvény segítségével léptetjük a folyamatot. Amint végighalad a cikluson a `finish` tagfüggvénnyel zárjuk. Ez megbizonyosodik arról, hogy a folyamat valóban véget ért, ahol a `line` tagfüggvénnyel a végére beszúrunk egy új sort, ezáltal a kurzort a következő sorba léptetjük. A ciklusunk ennek megfelelő rekordot fog létrehozni a `factory` metódus segítségével és a folyamatot a Symfony *Progress Bar* komponense által jeleníti meg. Az eredmény az alábbi 5.6. ábrán látható. Ennek megfelelően az adatbázis tisztítását is konzol paranccsal végezhetjük el, ahol átadjuk a kiürítésre szánt tábláinkat.

Az adatbázis tábláit a `php artisan faker:seed` terminál paranccsal tudjuk feltölteni. Ha a feltöltés során a terminál hibaüzenettel tér vissza, használjuk a `php artisan faker:clear` parancsot. Hibaüzenetek olyan esetekben fordulnak elő, ha egy korábbi *unique* adattípussal meghatározott oszlopban ismét létre szeretné hozni ugyanazt az adatot.

A `php artisan faker:clear` paranccsal lehetőségünk van az `app\Console\Commands\FakerClear.php` fájlban található `models` változóban meghatározott táblák adatainak a törlésére. Ha a feltöltött adatok mennyiségét szeretnénk meghatározni, akkor módosítsuk a `units` változó értékét az `app\Console\Commands\FakerSeed.php` fájlban.



5.6. ábra. A faker:seed parancs használata Atom fejlesztői környezetben

5.5. E-mail tesztelés

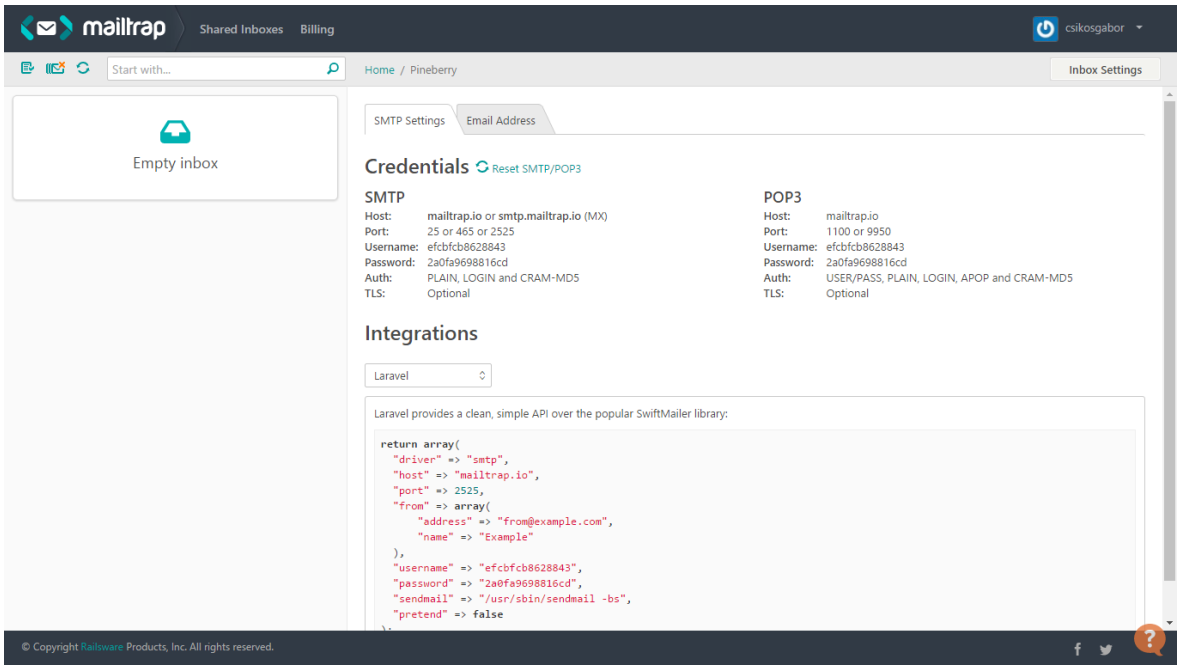
A Laravel a SwiftMailer könyvtáron alapuló API által lehetőséget ad a fejlesztők számára, hogy helyi vagy felhő alapú szolgáltatásokat vegyenek igénybe e-mail üzenetek elküldésére. Az API által az üzenetek elküldésére használhatjuk az SMTP, a Mailgun, a SparkPost vagy az Amazon SES szolgáltatásokat valamint támogatást ad a PHP `mail` és a `sendmail` függvények használatára.

A felhasználóknak szánt e-mail üzeneteket az elküldés előtt tesztelnünk kell. Ezen tesztfolyamathoz nyújt segítséget a Mailtrap. A Mailtrap egy hamis SMTP (Simple Mail Transfer Protocol) szerver, ami támogatást ad a fejlesztők számára az alkalmazásuk által kiküldött üzenetek tesztelésében. Ezáltal lehetőségük van az e-mail üzeneteik tartalmának az ellenőrzésére, mielőtt azokat elküldenék a felhasználóiknak.

A Mailtrap továbbá támogatást nyújt a megfelelő integráció kiválasztására, amellyel egyszerűen lehet integrálni az általunk választott keretrendszerbe. Az alábbi 5.7. ábrán a Mailtrap SMTP beállításokra vonatkozó információk láthatóak. Az integráció során kiválaszthatjuk a nekünk optimális keretrendszer SMTP integrációját.

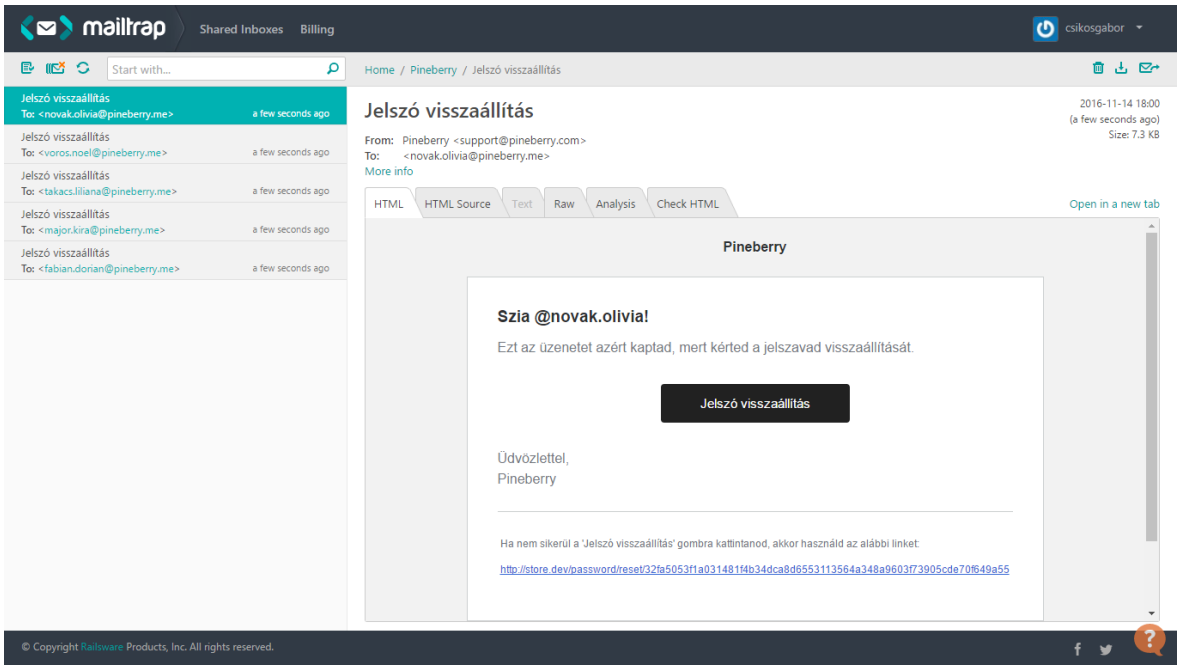
A Laravel esetében az SMTP-t kétféleképpen tudjuk beállítani. Az egyik megoldás, hogy kiválasztjuk az integrációs legördülő listából a Laravel-t, ahol visszatérésként egy tömböt láthatunk, amit a `config\mail.php` fájlba kell bemásolnunk. A másik egyszerűbb megoldás, hogy az SMTP felhasználó és jelszó párost a `.env` fájlba bemásoljuk és beállítjuk a titkosítást *tls*-re.

A beérkező e-mail során lehetőségünk van megtekinteni azokat HTML, HTML forráskód, szöveg vagy RAW formátumban. Az analitika során megtudhatjuk, hogy az üzenetünk tartalmaz-e vírust vagy valamelyik feketelistán szerepel-e. A HTML ellenőrzés egy összegzést ad nekünk arról, hogy a felsorolt HTML és CSS szabályok közül, melyek nem jelennek meg megfelelően az adott asztali, mobil vagy egyéb webes eszközökön. A Mailtrap által számunkra adott információkkal lehetőségünk van optimalizálni az e-mail üzenetünket és minden optimalizálás előtt tesztelni. Az 5.8. ábrán egy jelszó



5.7. ábra. Mailtrap - SMTP beállítások

visszaállító e-mailt láthatunk.



5.8. ábra. Mailtrap - Jelszó visszaállító e-mail

6. fejezet

Összefoglalás

A dolgozat egy receptalkalmazást, annak funkcióit, szerkezetét és elkészítésének fő lépéseit mutatja be. A `Receptek nyilvántartása` fejezetben az alkalmazás során meghatározott célcsoportról és a jelenlévő alkalmazások összehasonlításáról olvashatunk. A fejezet bemutatja az alkalmazott technológiákat és az ezen technológiák segítségével elkészített funkciókat. A fejezet végén sorra vettem az alkalmazás fejlesztésének további lehetőségeit, terveit.

Az `A Laravel keretrendszer` című fejezetben az alkalmazás által használt keretrendszer aktuális változata mellett bemutatásra került a keretrendszer története, és a kisebb, kevesebb erőforrást igénylő változata, a *Lumen* keretrendszer. Szintén ebben a fejezetben kapott helyet az alkalmazás felépítésének, a routing megvalósításának bemutatása.

A `Felhasználói dokumentáció` fejezetben képernyőképekkel kiegészítve bemutatásra kerültek az alkalmazás funkciói. Ebben különös figyelmet fordítottam a bejelentkezési módok, az ezzel kapcsolatos hibajelzések leírására.

A `Fejlesztői dokumentáció` fejezetben ismertettem az alkalmazás funkcióit és az ezekhez tartozó elérési pontokat a kontrollerek bemutatásán keresztül. Az adatbázis felépítését szemléltető EER modelleket követően néhány komplikáltabb lekérdezést, és az adatbázis véletlenszerű adatokkal való feltöltésének módját írtam le. Fontosnak tartottam, hogy az email küldés jól le legyen tesztelve, mivel az alkalmazás működése során erősen építkezik ezen funkcionalitásra.

Az alkalmazás elkészítése során számos tapasztalatot szereztem mind a Laravel mind pedig a Vue.js alkalmazása esetén. A legtöbb kihívást az alkalmazás kliens oldali részének elkészítése jelentette, mivel a technológia és egyes az egyes részek elkészítése is újnak bizonyult számomra. Ilyen rész volt például a képfeltöltés, amely kezdetben JavaScript nélkül került megvalósításra, és csak a későbbiekben írtam át belőle az elegánsabb JavaScript változatot. Szintén a JavaScript-nek köszönhetően kényelmesebbé vált az új hozzávalók felvitele az automatikus kiegészítés alkalmazásával. Ezen kihívások rámutattak arra, hogy érdemes minden feladatrészt alaposan előre megtervezni, mielőtt megoldanánk. Tervezés hiányában egy komponens elkészítése során időt veszíthetünk, így ezt az időt jobb inkább a tervezésre fordítani, mintsem hogy elpazaroljuk. A részletes és pontos dokumentáció segíthet a későbbi fejlesztések során, így az annak megírására fordított idő előbb vagy utóbb megtérül.

A szerver oldali fejlesztés során sok újdonsággal ismerkedtem meg és használtam fel azokat az alkalmazás elkészítése során. Ezen újdonság egyike az adatbázis lekérdezések, amelyekhez kezdetben *Query Builder*-t használtam, és csak később tértem át

az *Eloquent ORM*-re. Az Eloquent ORM még újnak bizonyul számomra, de a dolgozat megírása során sikerült a szükséges ismereteket megszereznem a használatához. Az Eloquent lekérdezések a megfelelő adatokkal térnek vissza, de a későbbiekben érdemes lesz majd még időt fordítani a lekérdezések optimalizálására is.

Akárcsak a Notification-ök, a factory-k elkészítése, és ezen factory-k konzolos alkalmazása is lehetőséget adtak új módszerek és technológiai elemek megismerésére. A fejlesztés során, az alkalmazás egyes részeinek a megvalósítása és a Laravel komponenseinek az alkalmazása, számos járatlan utat tartogatott számomra. Remélem, hogy az így megszerzett új ismereteket majd a későbbi alkalmazások megírása során is hatékonyan fel fogom tudni használni.

Bízom benne, hogy az alkalmazás által hozzá tudok járulni ahhoz, hogy az emberek életmódja egészségesebbé váljon.

Irodalomjegyzék

[1] Laravel - The PHP Framework For Web Artisans,
https://laravel.com/
Internet - 2016.11.15.

[2] Laravel - Wikipedia,
https://en.wikipedia.org/wiki/Laravel
Internet - 2016.11.15.

[3] Taylor Otwell - Inviting Arkansas,
http://www.invitingarkansas.com/featured/taylor-otwell
Internet - 2016.11.15.

[4] A Brief History of Laravel - Vehikl News,
https://medium.com/vehikl-news/a-brief-history-of-laravel-5d55970885bc
Internet - 2016.11.15.

[5] History of Laravel PHP framework, Eloquence emerging - Maxoffsky,
https://maxoffsky.com/code-blog/history-of-laravel-php-framework-eloquence-emerging
Internet - 2016.11.15.

[6] Laravel 5 - Laravel News
https://laravel-news.com/2015/01/laravel-5
Internet - 2016.11.15.

[7] How to modularize pages in Laravel. - Forrst,
http://zurb.com/forrst/posts/How_to_modularize_pages_in_Laravel-YjJ
Internet - 2016.11.15.

[8] What are the most promising PHP 5.3 frameworks? - Quora,
https://www.quora.com/What-are-the-most-promising-PHP-5-3-frameworks
Internet - 2016.11.15.

[9] Hogyan küldhetsz be receptet? - NoSalty,
http://www.nosalty.hu/mithogyan/t/hogyan-kuldhetsz-be-receptet/471
Internet - 2016.11.15.

Adathordozó használati útmutató

Az alkalmazás telepítés lépései

1. NGINX vagy Apache telepítése.
2. Adatbázis-kezelő telepítése.
3. PHP telepítése.
4. Composer telepítése.
5. Másoljuk át a fájlokat a hosztolt mappába.
6. Töltsük le a Laravel framework-ot a GitHub-ról, ami a https://github.com/laravel/framework címen érhető el és másoljuk a mappába, ahol az alkalmazás található.
7. Hozzunk létre egy *.env* fájlt a következő adatokkal az alkalmazás mappájában.

```
APP_ENV=local
APP_KEY=
APP_DEBUG=true
APP_LOG_LEVEL=debug
APP_URL=http://localhost

DB_CONNECTION=mysql # Adatbázis-kezelő megadása
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=store   # Adatbázis neve
DB_USERNAME=root    # Adatbázis felhasználóneve
DB_PASSWORD=        # Adatbázis jelszava

BROADCAST_DRIVER=log
CACHE_DRIVER=file
SESSION_DRIVER=file
QUEUE_DRIVER=sync

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

# A következő adatok megadásához szükség van egy Mailtrap fiókra, amit
```

```
# a https://mailtrap.io/ oldalon tudunk létrehozni

MAIL_DRIVER=smtp
MAIL_HOST=mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null  # Mailtrap SMTP felhasználóneve
MAIL_PASSWORD=null  # Mailtrap SMTP jelszava
MAIL_ENCRYPTION=tls

PUSHER_APP_ID=
PUSHER_KEY=
PUSHER_SECRET=

APP_NAME='Pineberry'                # Alkalmazás neve
SESSION_COOKIE='pineberry_session'  # Cookie neve

# A következő adatok megadásához szükség van egy Facebook
# alkalmazásra, amit a Facebook https://developers.facebook.com/
# tudunk létrehozni

FACEBOOK_CLIENT_ID=null         # Facebook alkalmazás azonosítója
FACEBOOK_CLIENT_SECRET=null     # Facebook alkalmazás jelszava
FACEBOOK_CALLBACK_URL='{URL}/facebook-login-callback'
# Az {URL} helyére kerül az alkalmazás URL-e
```

Az adatbázis telepítés lépései

1. Hozzunk létre egy adatbázist store néven utf8_general_ci illesztéssel.
2. A parancssoros értelmezőnek adjuk meg a `php artisan migrate --seed` parancsot.
3. Ezután adjuk meg a `php artisan faker:seed` parancsot.

Fontos, hogy ha szeretnénk ismételten alkalmazni a `php artisan faker:seed` parancsot, akkor használjuk előtte a `php artisan faker:clear` parancsot az adatbázis megtisztítására, ellenkező esetben kivétel keletkezik.