



Adatbázis alapú web alkalmazás fejlesztése

Készítette

Verebélyi Valentin

Programtervező Informatikus BSc

Témavezető

Dr. Tajti Tibor Gábor

Egyetemi Docens

EGER, 2024

Tartalomjegyzék

Bevezetés	4
1. Tervezés	5
1.1. A tervezéshez alkalmazott eszközök és technológiák	8
1.1.1. Dbdiagram	8
1.1.2. PlantUML	8
2. Fejlesztés	12
2.1. A fejlesztéshez alkalmazott eszközök és technológiák	12
2.1.1. Laravel	12
2.1.2. PHP	13
2.1.3. MySQL	13
2.1.4. PhpMyAdmin	13
2.1.5. XAMMP	14
2.1.6. Visual Studio Code	14
2.1.7. GitHub	14
2.1.8. GitHub Desktop	14
2.1.9. Tailwind CSS	14
2.1.10. JavaScript	15
3. Tesztelés	16
3.1. A teszteléshez alkalmazott eszközök és technológiák	16
3.1.1. Manuális teszt	16
3.1.2. Cypress	17
4. Felhasználói dokumentáció	20
4.1. Admin	20
4.1.1. Felhasználókkal kapcsolatos művelet	20
4.1.2. Kategóriákkal kapcsolatos műveletek	21
4.1.3. Vármegyével és városokkal kapcsolatos műveletek	21
4.1.4. Hirdetéssel kapcsolatos műveletek	21
4.2. Felhasználó	22

4.2.1.	Hirdetéssel kapcsolatos műveletek	22
4.3.	Nézelődő	23
4.4.	Egyéb funkciók	23
5.	Fejlesztői dokumentáció	25
5.1.	Az alkalmazás felépítése	25
5.1.1.	Adatbázis kapcsolat létrehozása	25
5.1.2.	Migráció	26
5.1.3.	Modell	27
5.1.4.	Kontroller	28
5.1.5.	Web.php	29
5.1.6.	Blade	30
	Összegzés	33
	Irodalomjegyzék	34

Bevezetés

1. fejezet

Tervezés

A tervezés egy nagyon fontos szerepet tölt be egy szoftvereknél. Az én meglátásom és tudásom szerint a következő okok miatt szükséges tervezni:

1. Tervezési célok meghatározása: fontos az, hogy mind a fejlesztő, mind a megrendelő egyértelműen megértse, hogy pontosan mit kell elérni a szoftverrel.
2. Költség- és időmegtakarítás: implementálás előtt, ha kellő alaposággal tervezzük meg a szoftvert, akkor segíthet kiszűrni a későbbi fázisokban esetleges előforduló hibákat.
3. Bővíthetőség: ez alatt azt értem, hogy a szoftver úgy kell megtervezni, hogy fel legyen készítve arra, hogy lehessen hozzáadni könnyedén új funkciókat.
4. Funkciók és feladatok felosztása: ez azért lehet hasznos, mivel a fejlesztés során a fejlesztők pontosan csak az ő általuk elvállalt feladatokat valósítják meg.
5. Kommunikáció segítése: ha van egy jól kidolgozott terv, akkor ha a fejlesztők elakadnak valamiben vagy nem értik meg pontosan, hogy mit és hogyan kell implementálni, akkor elég, ha megnézik a tervben az adott dologhoz kapcsolódó részeket és ezáltal megtudják oldani a rájuk bízott feladatot.

Azonban az imént felsoroltakon kívül még számtalan ok miatt szükséges a tervezni. A következőben az én általam készített adatbázis alapú webalkalmazásomnak a követelménylistája látható. A követelménylistában azokat a funkciókat szükséges dokumentálni, amikkel a programoknak rendelkeznie kell. Ez általában a különböző projektek esetében a követelmény specifikációban található meg. A követelmény specifikáció egy olyan dokumentáció, amelyben az igények, követelmények kerülnek meghatározásra, ennek az alapja a riportok. A riportok segítségével lehetséges felmérni a megrendelő követelményeit. Az 1.1 táblázatban megtekinthető a projektem követelmény listája.

1.1. táblázat. Követelménylista

<i>Id</i>	<i>Modul</i>	<i>Név</i>	<i>Leírás</i>
K1	Jogosultság	Regisztráció	A webalkalmazásom teljes körű használatához regisztrációra van szükség.
K2	Jogosultság	Bejelentkezés	Email és jelszóval van lehetőség belépni. Hibák esetén visszajelzést kapnak a felhasználók.
K3	Jogosultság	Kijelentkezés	
K4	Jogosultság	Profillal kapcsolatos műveletek	A felhasználónak van lehetősége a nevét, email címét és jelszavát is módosítani, illetve törölni a saját profilját. Hibák esetén visszajelzést kapnak a felhasználók.
K5	Jogosultság	Jogosultsági szintek	<p><i>Admin</i>: vármegyékkel, városokkal, kategóriákkal kapcsolatos műveletek, azaz azok megtekintése, törlése, módosítása és hozzáadása. A hirdetések törlése, módosítása, megtekintése. A felhasználók törlése. Üzenetek küldése, megtekintése, sajátok módosítása és törlése.</p> <p><i>Felhasználó</i>: hirdetések megtekintése, sajátjai módosítása, törlése. Üzenetek küldése, megtekintése, sajátok módosítása és törlése. Kategóriák megtekintése.</p> <p><i>Nézelődő</i>: kategóriák és hirdetések megtekintése.</p>
K6	Felület	Regisztráció	Egy név, email cím és a jelszó kétszeri megadása szükséges. Hibák esetén visszajelzést kapnak a felhasználók.
K7	Felület	Bejelentkezés	Email és jelszóval van lehetőség belépni. Hibák esetén visszajelzést kapnak a felhasználók.
K8	Felület	Hirdetések	Az összes hirdetés kilistázva való megtekintése.
K9	Felület	Hirdetések szűrése, rendezése, keresése	Az összes hirdetés kilistázva való megtekintése a szűrésre, keresésre vagy rendezésre. A keresés az a hirdetés neve alapján történik.

Folytatódik a következő oldalon

táblázat 1.1 – folytatás

Id	Modul	Név	Leírás
K10	Felület	Saját hirdetések	A saját hirdetések kilistázva való megtekintése, törlése.
K11	Felület	Saját hirdetések módosítása	A hirdetés összes adatát lehetséges megváltoztatni, kivéve a hirdető nevét.
K12	Felület	Új hirdetés hozzáadása	
K13	Felület	Saját beszélgetések megtekintése	Az összes beszélgetés megtekintése, csak az utolsó üzenet látjuk.
K14	Felület	Saját üzenetek megtekintése, törlése	
K15	Felület	Saját üzenetek módosítása	
K16	Felület	Saját beszélgetések megtekintése	
K17	Felület	Kategóriák megtekintése, törlése	A törlés csak admin jogosultsággal lehetséges.
K18	Felület	Új kategória hozzáadása	
K19	Felület	Kategória módosítása	
K20	Felület	Vármegyék megtekintése, törlése	Csak admin jogosultsággal lehetséges.
K21	Felület	Vármegye hozzáadása	Csak admin jogosultsággal lehetséges.
K22	Felület	Vármegye nevének módosítása	Csak admin jogosultsággal lehetséges.
K23	Felület	Városok megtekintése, törlése	Csak admin jogosultsággal lehetséges.
K24	Felület	Városok hozzáadása	Csak admin jogosultsággal lehetséges.
K25	Felület	Városok nevének módosítása	Csak admin jogosultsággal lehetséges.

Folytatódik a következő oldalon

táblázat 1.1 – folytatás

Id	Modul	Név	Leírás
K26	Felület	Felhasználók megtekintése, törlése	Csak admin jogosultsággal lehetséges. Az adminok itt nem kerülnek kilistázásra.

1.1. A tervezéshez alkalmazott eszközök és technológiák

A következő alszakaszokban a tervezéshez használt eszközökről és technológiákról lesz majd szó, melyeknek bemutatásaival az a célom, hogy alaposabb megértést nyújtsak az adatbázis alapú webalkalmazáshoz kapcsoló alkalmazott eszközökről és technológiákról.

1.1.1. Dbdiagram

A dbdiagram segítségével van lehetőség arra, hogy egy alkalmazás adatbázisának a sémáját és struktúráját megtervezzük és ehhez ad egy vizuális képet számunkra. A DBML-t, azaz Database Markup Language használja, amit magyarul talán adatbázis jelölőnyelvként tudnék lefordítani. Amiért a véleményem szerint nagyon hasznos még az nem más, mint, hogy rengetegféle olyan opciót ad számunkra, amire szükségünk lehet. Ilyen opciók például azok, hogy van lehetőség importálni be kódot MySQL, PostgreSQL, Rails valamint SQL szerverről. Ugyanezeket a típusú kódokat kilehet exportálni az imént megemlített típusokba, kivéve a Rails, mivel oda nem lehetséges. Ezenkívül van még olyan opció is, hogy ki lehet exportálni PNG, SVG vagy akár PDF formátumba is az elkészült tervet. A dbdiagram alap változata ingyenesen használható mindenki számára. [9]

A következőkben az 1.2 képen látható adatbázis-terv kódjáról szeretnék egy keveset még írni, pontosabban arról, hogy hogyan épül fel az előbb bemutatott terv. Csak a tervről mutatott be pár részletet az érdekesség kedvéért.

Az 1.2 képen látható a dbdiagram-ban készített ábra az alkalmazáshoz.

1.1.2. PlantUML

Az UML a Unified Modeling Language rövidítése, azaz Egységes Modellezési Nyelv. Az UML arra jó, hogy szoftvert tervezzünk vele, a gondolataimat letudom vele rajzolni, skiccelni. Ezenkívül a kiforrott megoldások dokumentálására is használható. A PlantUML egy komponens, aminek a segítségével gyorsan tudunk létrehozni szekvencia-


```

// tábla létrehozása
Table Hirdetes
{
  hirdetes_id integer [pk, increment, not null]
  felhasznalo_id integer [not null]
  varos_id integer [not null]
  kategoria_id integer [not null]
  telefonszam varchar [unique]
  nev varchar [not null]
  ar varchar [not null]
  leiras text
}

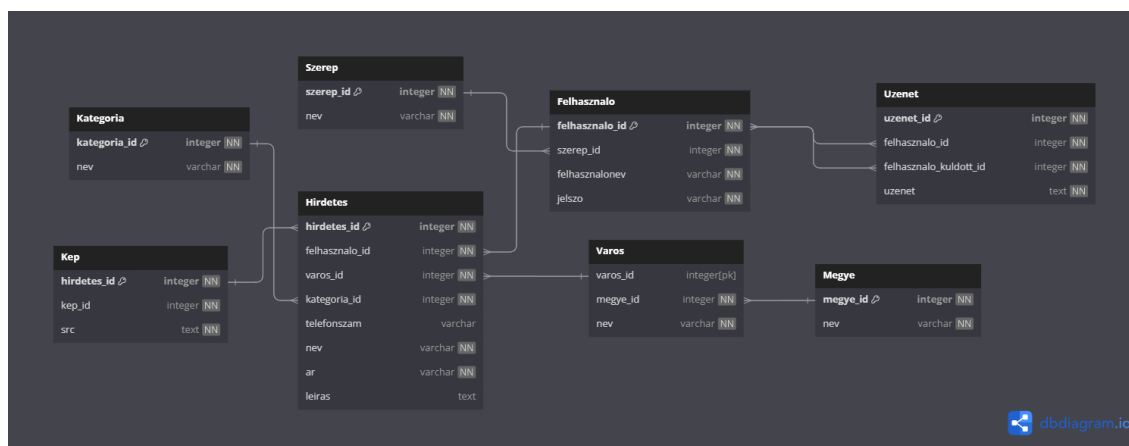
Table Varos
{
  varos_id integer[pk]
  megye_id integer [not null]
  nev varchar [unique, not null]
}

// táblák közötti kapcsolat megadása
Ref: Varos.varos_id < Hirdetes.varos_id

// < egy-a-többhöz kapcsolat
// > több-az-egyhez kapcsolat
// <> több-a-többhöz kapcsolat

```

1.1. ábra. DMBL kód részlet (Saját készítés)



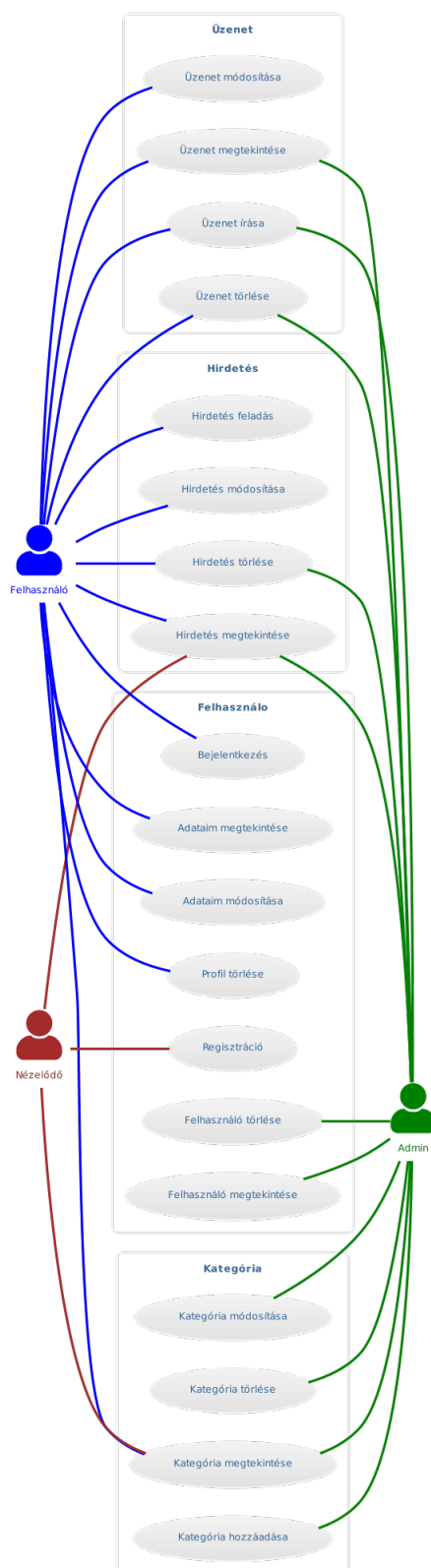
1.2. ábra. Adatbázisisterv (Saját készítés)

használati eset- és osztálydiagramokat. Ezenkívül még rengetegféle diagramot tudunk készíteni. [7]

Manapság a tervezés az használati eset központú. A következő pár mondatban az ilyenféle alapú tervezésről lesz szó. Tulajdonképpen itt azt a kérdést tehetjük fel, hogy ki (angolul: actor) mire (funkció) tudja használni az adott informatika rendszert. Az actor-ról azt érdemes tudni, hogy ő nem része az informatikai rendszernek, hame ő csak használja azt. Az actor gyakran lehet ember, AI vagy akár egy másik informatikai rendszer. Fontos az is, hogy minden ábra, amit készítünk az visszavezethető legyen a

már meghatározott követelmények szintjéig. Az használati alapú tervezés előnye az, hogy egyszerűen, érthetően írja le a rendszer funkcióit. Ebből kifolyólag, mind a megrendelő, mind a tervező számára könnyen érthető az ábra. Tulajdonképpen a használati eset alapú tervezés nem más, mint egy közös nyelv a megrendelő és a tervező között, amit mind a két fél ért.[2] Az 1.3 képen látható a plantUML-ben készített használati eset ábra az alkalmazásomhoz. Az 1.3 képen látható használati eset ábra magyarázata következik: Az ábrán három darab actor látható.

1. Nézelődő: ennek a szerepnek van a legkevesebb funkciója, ő az aki csak a már meglévő hirdetéseket és kategóriákat tudja megtekinteni, illetve képes még regisztráció, ami további meglévő funkciók használata miatt szükséges.
2. Felhasználó: ő képes a saját adatainak módosítására, a profil törlése is. Neki van már lehetősége saját hirdetéseket létrehozni és ezeket kezelni, valamint már van lehetősége üzenetet küldeni egy másik felhasználónak.
3. Admin: neki már van lehetősége a kategóriákkal kapcsolatos műveletek elvégzésére is. Szintén képes üzenetet küldeni bárkinek, ő képes már felhasználókat és hirdetések törölni az adott esetben.



1.3. ábra. Használati eset (Saját készítés)

2. fejezet

Fejlesztés

2.1. A fejlesztéshez alkalmazott eszközök és technológiák

A következő alszakaszokban a fejlesztéshez használt eszközökről és technológiákról lesz majd szó. Részletezem majd azokat az eszközöket és technológiákat, melyeket a fejlesztés során használtam és alkalmaztam. A következő alszakaszokban definiálni fogom a választott eszközöket és technológiákat, ezenkívül pedig megemlítem, hogy hol használtam őket, azaz miket valósítottam meg a segítségükkel.

2.1.1. Laravel

A Laravel egy nyílt forráskódú, PHP webkeretrendszer, ami MVC tervezési mintán alapszik. Az MVC a Model-View-Controller rövidítése. Ez az egyik legősibb tervezési minta.

1. Model (magyarul: modell): ez az adatokat kezelő réteg, az adatok tárolásáért és visszaolvasásáért felelős.
2. View (magyarul: nézet): ez a réteg felhasználói felületek megjelenítéséért illetékes.
3. Controller (magyarul: vezérlő, kontroller): a felhasználói műveletek megfelelő kezeléséért ez a réteg a felelős.

Az elképzelése röviden az, hogy van egy modell és ha ez módosul, akkor erről értesíteni kell az összes nézetet. Minden nézett pontosan ugyanazon a vezérlőn tudja módosítani a modellt. Mivel három részre van bontva, ezért van ennek a tervezési mintának néhány előnye is. Ilyen például ha lecseréljük az egyik réteget, akkor a többihez nem kell már hozzányúlni, amivel pénzt és időt is lehet spórolni. Előnye még az is, hogy

egy modellnek lehet több nézetes is. Pozitívumok közzé sorolható az még, hogy számos alapvetőnek tekinthető funkcióhoz lehetséges használni néhány szükséges parancs megadásával. Például a webes alkalmazásban használtam a laravel breezehez tartozó funkciókat. Ezzel különféle autentikációhoz kapcsolódó funkciókat lehet használni, ebbe beletartozik például a bejelentkezés, a regisztráció és a felhasználóval kapcsolatos adatok módosítása is. Még használtam egy olyan parancs által megadható funkciót is, ami a lapozás megvalósítását teszi lehetővé. Ezeknek a pozitívumoknak köszönhetően szerintem a laravel az nagyon megkönnyítheti a fejlesztők számára a munkavégzést. [1, 12]

2.1.2. PHP

A PHP egy szerveroldali szkriptnyelv, amelynek a segítségével dinamikus weboldalakat lehet vele létrehozni. Azt érdemes még róla tudni, hogy nyílt forráskódú. Ezen kívül még a következő feladatok megvalósítására lehet használni:

1. Parancssoros szkripttelés: itt lehetőségünk van arra, hogy úgy futtassunk egy php szkriptet, hogy nem használunk se böngészőt, se semmilyen szervert sem.
2. Asztali alkalmazások készítése: grafikus felhasználó felülettel ellátott asztali alkalmazások esetén a php nem a legjobb választás, azonban van lehetőségünk arra, hogy platformfüggetlen alkalmazások megvalósítsunk.

[11]

2.1.3. MySQL

A MySQL nem más, mint egy nyílt forráskódú adatbázis kezelő rendszer. A MySQL egy relációs adatbázis, ahol az adatokat különböző táblákban vannak eltárolva. A különböző táblákban szereplő mezők között lehetnek különféle kapcsolatok is. Ilyen lehet például az egy-az-egyhez kapcsolat vagy egy-a-többhöz kapcsolat. A MySQL adatbázisok szerverére jellemző, hogy gyorsak, megbízhatóak és skálázhatóak. [4]

2.1.4. PhpMyAdmin

A phpMyAdmin egy ingyenes szoftver, ami arra szolgál, hogy kezelje a MySQL-nek az adminisztrációját weben keresztül. A phpMyAdmin segítségével elvégezhetők a legtöbb adminisztratív feladatok, ideértve az adatbázis létrehozását, lekérdezések futtatását és felhasználói fiókok hozzáadását. [5]

2.1.5. XAMMP

Az XAMMP nem más, mint egy PHP fejlesztői környezet. Erről még azt érdemes tudni, hogy ingyenesen használható és rendkívül egyszerű telepítése, illetve a használata.

2.1.6. Visual Studio Code

Ez egy IDE (integrált fejlesztői környezet), ami ingyenes használható és az egyik legelterjedtebb és legnépszerűbb a fejlesztők körében. Fontosnak tartom megemlíteni, hogy a Visual Studio Code-ban van lehetőség különféle kiegészítők (angolul: extension) letöltésére is, például van lehetőség a python programozási nyelv vagy egy programozási nyelvhez tartozó szintaxis kiemelő letöltésére, illetve ezek használatára.

2.1.7. GitHub

A GitHub-ról azt érdemes tudni, hogy ez egy verziókövető rendszer. Ezek a rendszerek képesek állományok tartalmi változásait követni, azt is képesek megmondani, hogy ki és mikor módosította azokat, valamint van lehetőség arra is, hogy korábbi állapotokat is képes előállítani. A main branch-be feleltethető meg a fő ágnak, amiből van lehetőség elágazások (angolul: branch) is létrehozni. Az elágazásokat arra valóak, hogy a fejlesztési funkciókat elkülönítsük. Még arra is használhatjuk, hogy kísérletezzünk vagy akár hibajavítások elvégzésére is lehet használni.

2.1.8. GitHub Desktop

A GitHub Desktop egy ingyenesen használható alkalmazás, aminek a segítségével tudunk dolgozni a GitHub-on vagy Git-tárhely-szolgáltatásokon tárolt fájlokkal. Én ezt az eszközt azért szeretem használni, mivel megkönnyíti és felgyorsítja számomra a munkavégzés, illetve azért is, mert ennek a használatához nem kell a terminálban beírni a Git-hez tartozó parancsokat, hanem egy-két kattintás segítségével elvégezhetek egy konkrét parancsot. [6]

2.1.9. Tailwind CSS

Tailwind css használatával a felhasználói felületeket lehet kialakítani, ez csak külsőleges formázást jelent, tehát csak és kizárólag esztétikai szépítésre szolgál. Érdemes még róla tudni, hogy gyors, megbízható, rugalmas és nincsen futási ideje, azaz azonnal megtörténik a változás, nem kell rá várni semmit sem. [13]

2.1.10. JavaScript

A javascript az egyik szkript- vagy programozási nyelv, aminek a segítségével van lehetőség bonyolultabb funkciók megjelenítésére a weboldalakon. Lehetővé teszi például egy dinamikusan frissülő tartalom létrehozását vagy akár képek animálását is. Ezen kívül még nagyon sok mindenre használható még. A projektben használtam a javascriptet ahhoz, hogy amikor vármegyét választok ki egy legördülő listából, akkor egy másik listában csak az előzőleg kiválasztott vármegyébe tartozó város jelenjenek meg. A másik ahol szintén használtam az pedig a szűréshez tartozó részeket jelenítem meg vagy nem, annak a tükrében, hogy be van pipálva egy checkbox (magyarul: jelölőnégyzet) vagy nem.

3. fejezet

Tesztelés

A tesztelésre azért van szükség, hogy az alkalmazásomban megtaláljam az esetleges hibákat, amiket kijavítva növelhetem a szoftverem minőségét és megbízhatóságát. Abban sajnos nem lehetek biztos, hogy a tesztelés elvégzése után nem lesznek már hibák. A tesztelés során kettőféle tesztelési technikát alkalmaztam. Ezek pedig a következők:

1. Fehérdobozos (angolul: white-box): a forráskód alapján íródnak a tesztesetek.
2. Szürkedofozos (angolul: grey-box): amikor a forráskódnak csak egy rész ismert és csak ez alapján íródnak a tesztesetek.

Azonban van egy harmadik fajta is, amit nem alkalmaztam. Az nem más, mint a feketedofozos (angolul: black-box), amikor a tesztesetek a specifikáció alapján íródnak. [1, 26-29. oldal]

3.1. A teszteléshez alkalmazott eszközök és technológiák

A következő alszakaszokban a teszteléshez használt eszközökről és technológiákról lesz majd szó. Kitérek majd arra, hogy pontosan mit is jelentenek, miket lehet velük tesztelni és az előnyük és hátrányaikról is ejtek majd pár gondolatot.

3.1.1. Manuális teszt

Manuális teszt az azt jelenti, hogy a tesztelő végig próbálgatja saját magától a funkciókat és leellenőrzi azok működését, illetve helyességét. Az ilyenféle tesztelés végzésekor nem alkalmazunk semmilyen automatizált tesztet. Én fejlesztés közben folyamatosan végeztem manuális tesztelést, abból kifolyólag, hogy megbizonyosodjak az elkészült funkció helyességéről, illetve, hogy feltárjam vele az esetleges hibákat, amiket nem vettem észre vagy még nem gondoltam rá. Az én meglátásom szerint a manuális tesztelésnek a legnagyobb hibája az, hogy minden lehetséges kombinációt ki kell próbálnunk

ahhoz, hogy bebizonyosodjunk arról, hogy a készített funkció a megfelelő módon működik. Még szintén hátrány lehet az, hogy rendkívül költséges lehet a manuális tesztelés. Végül pedig az, hogy az emberek nagyobb százalékban hibáznak, ami érthető is, hiszen tévedni emberi dolog. Azonban vannak előnyei is, ezek pedig véleményem szerint a következők lennének. Az első ilyen az lenne, hogy, ha van valamilyen hiba, akkor azt nagy valószínűséggel hamarabb észre lehet venni, amivel pénzt és időt is lehet spórolni. A másik ilyen előny az pedig, hogy a tesztelőnek van megfelelő intelligenciája ahhoz, hogy kiszűrje az esetleges hibákat, ellenben az automatizált tesztekkel szemben.

3.1.2. Cypress

A Cypress egy NodeJS-ben írt front end tesztelési keretrendszer. Ahhoz, hogy tudjuk futtatni a Cypress-t, ahhoz előtte telepíteni kell a NodeJS-t. A Cypress segítségével tudunk készíteni végponttól végpontig tartó-, komponens-, integrációs- valamint a unitteszteket is. Az imént felsorolt teszt típusok jelentése a következő:

1. Végponttól végpontig tartó teszt: ennél a típusnál a szoftver funkcionalitását és teljesítményét teszteljük a kezdetektől a végéig, ami a vég felhasználók által megvalósított interakciókat szimulálja valós adatokkal.
2. Komponensteszt: ez nem más, mint, amikor a rendszernek csak egy önálló komponensét teszteljük.
3. Integrációs teszt: ez egy olyanféle teszt, aminél legalább kettő különböző komponens együttműködését teszteljük.
4. Unitteszt (magyarul: egységteszt): ez egy olyan teszt, ami a metódusokat teszteli le. Itt nézzük meg, hogy a tényleges visszatérési érték azonos-e az elvárttal. Ez a komponenteszt egyik fajtája.

Ezzel a tesztelési keretrendszer segítségével bármit lehet tesztelni ami a böngészőben fut. Nézzük meg néhány előnyét és hátrányát is ennek használatára. Kezdjük a negatívumokkal először. A teszteknek javascript nyelven kell íródniuk. Erről többet megtudhat itt a 2.1.10. Hátrány még az is, hogy egyszerre csak egy ablakban vagy csak egy böngészőben lehetséges a tesztek végrehajtása. Most nézzük meg a pozitívumait is. Az egyik legnyilvánvalóbb ha ránézzünk egy ilyen kódra az, hogy nagyon olvasható bárki számára. Előny még az is, hogy igazi böngészőt használ a tesztek végrehajtására. Ez azért hasznos, mivel a valódi végfelhasználók is igazi böngészőkben fogják használni az alkalmazást, ezért jól lehet velük szimulálni a működését. Még előnynek tekinthető az is, hogy a hálózattól érkező kéréseket lehetséges ellenőrizni a cypress segítségével. [8, 3, 10, 14]

A szemléltetés kedvéért szeretné, Önnek bemutatni, hogy hogyan is kell elképzelni egy ilyenféle tesztet. A 3.1 kódon látható.

3.1. kód. Egy cypress teszt egy hirdetés hozzáadására

```
1 describe('template_spec', () => {
2   it('passes', () => {
3
4     cy.visit('http://127.0.0.1:8000/advertisements/create')
5
6     cy.get('input[id=email]').type('v@gmail.com')
7     cy.get('input[id=password]').type('12345678')
8     cy.contains('Bejelentkezés').click()
9
10    cy.get('select[id=countySelect]').select('Heves_vármegye')
11    cy.get('select[id=citySelect]').select('Eger')
12    cy.get('select[id=category]').select('Autó')
13
14    const picture1 = 'background_image.png'
15    const picture2 = 'background.jpg'
16    cy.get('input[id=pictures1]').attachFile(picture1)
17    cy.get('input[id=pictures2]').attachFile(picture2)
18
19    cy.get('input[id=title]').type('Eladó_BMW_320d')
20    cy.get('input[id=price]').type('10000000')
21    cy.get('label[for=description]').next('textarea').type('Eladó_egy_BMW
    ↳ 320d, 2015-ös_évjárat, 200.000_km');
22    cy.get('input[id=mobile_number]').type('06301234567')
23
24    cy.contains('Hozzáadás').click()
25
26    cy.location('href').should('eq', 'http://127.0.0.1:8000/
    ↳ advertisements')
27
28   })
29 })
```

Itt a visit (magyarul: meglátogat) metódus segítségével lehetséges egy konkrét útvonalra eljutni. Ezután bejelentkezni szükség, mivel csak akkor van lehetőség hirdetés hozzáadására. Id, azaz valamilyen azonosító alapján szükséges megkeresni a formon az adatok megadásának a helyét. Ahol mi adunk meg valamilyen információt, ott a type (magyarul: begépel) metódus segítségével tehetjük meg. Ahol legördülő menüből kell kiválasztani a szükséges adatokat, ott pedig a select (magyarul: kiválaszt) metódussal lehetséges. Ezen kívül még két darab kép hozzáadása is látható még, ezt az attachFile (magyarul: fájl csatolása) metódussal történik, azonban ehhez szükséges még egy csomagot letölteni, de ezt csak megemlítettem, mivel többet erről nem lesz szó. Gom-

bokat lehetséges szöveg alapján megkeresni és utána rákattintani, ez pedig a `contains` (magyarul: tartalmaz) és `click` (magyarul: rákattint) metódusokkal történhet meg. Végetül pedig egy átirányítása megvalósítása látszik, ha minden sikerrel zárult. Tehát véleményem szerint kijelenthető, hogy cypress tesztek írása egyszerű és nagyon érthető még akár egy hétköznapi ember számára is.

4. fejezet

Felhasználói dokumentáció

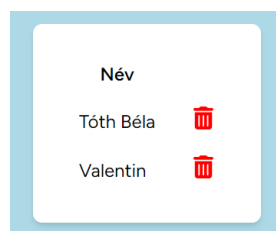
Ebben a fejezetben szó lesz majd, arról, hogy különböző jogosultságú felhasználóknak milyen lehetőségei vannak a webes alkalmazásom használata során. Minden ebben a fejezetben lévő alfejezetben szereplő jogosultsághoz szinte csak azokat a funkciókat említem majd meg, amiket csak ők tudnák használni. A különböző szerepű felhasználókról az 1.1.2 alfejezetben már megemlítettem róluk pár szót.

4.1. Admin

Az admin (magyarul: adminisztrátor) van néhány szükséges egyedi funkció az alkalmazásomnak. Az adminisztrátoroknak nagyon fontos szerepük bármilyen alkalmazás életében. Az webes projektben a következő admin funkciókat készítettem el.

4.1.1. Felhasználókkal kapcsolatos művelet

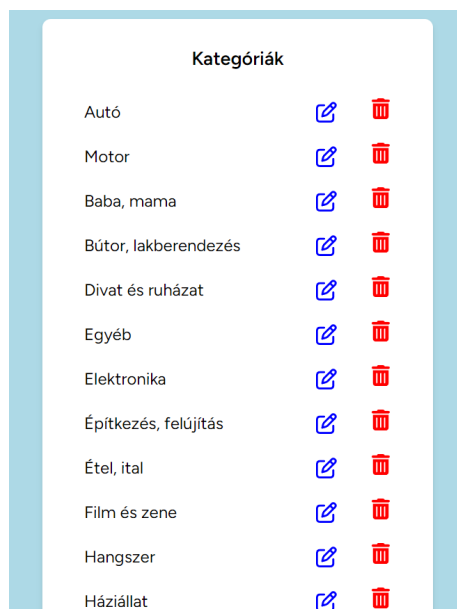
Az én meglévő tudásom birtokában úgy gondoltam, hogy a csak és kizárólag törölni tudjon már regisztrált felhasználókat. Admin jogosultságú felhasználókat nem lehetséges kitörölni, azaz amennyiben van több ilyen, akkor nem tudják kitörölni egymást. Ennek a jogosultságnak a birtokában képes új kategóriát hozzáadni, módosítani már meglévőt és még képes törölni is, amennyiben szükséges. A 4.1 ábrán a piros színű ikon segítségével törölni lehet a felhasználót.



























4.1. ábra. Felhasználók (Saját készítés)

4.1.2. Kategóriákkal kapcsolatos műveletek

Ennek a jogosultságnak a birtokában képes új kategóriát hozzáadni, módosítani már meglévőt és még képes törölni is, amennyiben szükséges. A 4.2 ábrán a kék színű ikon segítségével módosítani lehet a kategóriát, a piros színűvel pedig törölni. A képen nem látszik viszont van ott egy gomb, aminek a segítségével tudunk új kategóriát hozzáadni.



Kategóriák		
Autó		
Motor		
Baba, mama		
Bútor, lakberendezés		
Divat és ruházat		
Egyéb		
Elektronika		
Építkezés, felújítás		
Étel, ital		
Film és zene		
Hangszer		
Háziállat		






















4.2. ábra. Kategóriák (Saját készítés)

4.1.3. Vármegyével és városokkal kapcsolatos műveletek

Itt az admin jogosultságú felhasználó képes új vármegyét hozzáadni, meglévőt módosítani, illetve törölni. Szintén képes még különböző vármegyékhez hozzáadni új városokat, meglévőt módosítani és törölni. A 4.3 ábrán a zöld színű ikon segítségével lehet megtekinteni az adott vármegyéhez tartozó városokat megjeleníteni, ez ugyan úgy épül fel, mint az imént említett kategóriák a 4.1.2 alfejezetben. A kék színű ikon segítségével módosítani lehet a kategóriát, a piros színűvel pedig törölni. A képen nem látszik viszont van ott egy gomb, aminek a segítségével tudunk új vármegyét hozzáadni.

4.1.4. Hirdetéssel kapcsolatos műveletek

Ez lenne az egyetlen kivétel, itt a sima felhasználó is képes módosítani vagy törölni a saját hirdetéseit. Képes a már az összes meglévő hirdetéseket módosítani, illetve törölni azokat, ha valamilyen oknál fogva nem megfelelő nyelvezetet vagy képet használ.

Vármegye		
Bács-Kiskun vármegye		
Baranya vármegye		
Békés vármegye		
Borsod-Abaúj-Zemplén vármegye		
Csongrád vármegye		
Fejér vármegye		
Győr-Moson-Sopron vármegye		
Hajdú-Bihar vármegye		
Heves vármegye		
Jász-Nagykun-Szolnok vármegye		
Komárom-Esztergom vármegye		

4.3. ábra. Vármegyék (Saját készítés)

4.2. Felhasználó

A következő alfejezetekben csak a sima felhasználó egyedi funkcióijáról lesz szó.

4.2.1. Hirdetéssel kapcsolatos műveletek

Regisztrált felhasználóként van arra lehetőség, hogy a hirdetéseivel kapcsolatos műveleteket végezzen el, legyen szó akár csak azok megtekintéséről, módosításairól vagy törléseiről. Ő már képes új hirdetések létrehozni, ami a 4.4 ábrán lesz látható. A 4.4

<div>Vármegye</div> <div>Válassz vármegyét</div> <div>Város</div> <div>Válassz várost</div> <div>Kategória</div> <div>Válassz kategóriát</div> <div>Kép 1</div> <div>Fájl kiválasztása Nincs fájl kiválasztva</div> <div>Kép 2</div> <div>Fájl kiválasztása Nincs fájl kiválasztva</div> <div>Kép 3</div> <div>Fájl kiválasztása Nincs fájl kiválasztva</div>	<div>Cím</div> <div></div> <div>Ár</div> <div>Ft</div> <div>Leírás</div> <div></div> <div>Telefonszám</div> <div></div> <div>Hozzáadás</div>
---	--

4.4. ábra. Új hirdetés hozzáadása (Saját készítés)

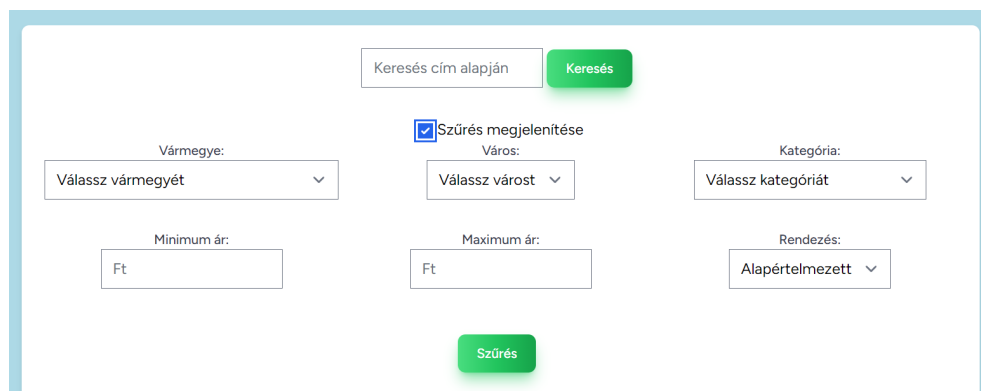
ábrán magyarázata következik. A vármegyét egy legördülő listából lehet kiválasztani, ha ezt megtettük csak utána tudjuk majd a várost kiválasztani szintén legördülő listából. A hirdetés kategóriáját is ilyenféle módon választhatjuk ki. A hirdetés címét, árát és leírását pedig a felhasználó adja majd meg. Ezek voltak a kötelezően megadandó információk. Telefonszámot, illetve képeket nem szükséges megadni. A megadható képek számát limitáltam 5 darabra.

4.3. Nézelődő

Ő nem rendelkezik semmilyen jogosultsággal sem, ezért is nincsen egyedi funkciója ennek a szerepkörnek. Ő csak a kategóriákat képes megtekinteni, illetve a meglévő hirdetéseket tudja megnézni. Keresésre és szűrésre is van itt lehetőség. Ezekről a 4.4 alfejezetben olvashat többet.

4.4. Egyéb funkciók

1. Keresés: a hirdetéseknel van arra lehetőség, hogy keresni lehessen, ez a hirdetés címe alapján lehetséges.
2. Szűrés: van arra lehetőség, hogy szűrjünk ára, vármegyére, városra vagy kategóriára is.
3. Rendezés: a hirdetéseket rendezhetjük ár szerint növekvő, csökkenőbe vagy lehet alapértelmezett is.



4.5. ábra. Keresés, szűrés és rendezés (Saját készítés)

4. Üzenetek: a regisztrált felhasználók vagy adminisztrátorok képesek üzenetek küldeni egymásnak, ezeket megtekinteni, módosítani vagy törölni a saját általuk korábban elküldött üzeneteket is.

5. Profil módosítása: a webes alkalmazásban adva van a lehetőség arra, hogy a regisztrált felhasználó tudják módosítani a saját adataikat, illetve képesek törölni a profiljukat is. Az adminisztrátor nem képes arra, hogy módosítsa a felhasználók adatait, csak törölni tudja őket.

5. fejezet

Fejlesztői dokumentáció

5.1. Az alkalmazás felépítése

Ebben az alfejezetben szeretném mutatni Önöknek az alkalmazásom fő komponenseit kódrészletek segítségével a könnyebb megértés reményében. Egy nagyon keveset már említettem megemlítettem korábban a 2.1.1 alfejezetben, de nem mindent, most lépésről lépésre elmesélem, hogy hogyan lehetséges megjeleníteni egy konkrét adatokat az adatbázisomból.

5.1.1. Adatbázis kapcsolat létrehozása

Magát az adatbázist többféle módon van lehetőségünk létrehozni, de ezekbe most nem mennék bele. Most itt még azt kell megemlítenem, hogy a projektben ezt hogyan szükséges beállítani. Ehhez meg kell keresni a projektben a .env nevű fájlt és meg kell adni néhány információt. A következőket szükséges megadni.

1. DB_CONNECTION (magyarul: adatbázis kapcsolat): itt magát az adatbázis fajtáját kell megadnunk. Alapértelmezetten mysql van megadva.
2. DB_HOST (magyarul: adatbázis kiszolgáló): ez az adatbázis IP-címe lenne.
3. DB_PORT (magyarul: adatbázis port): ennek a segítségével tudunk kapcsolódni az adatbázis szerverhez.
4. DB_DATABASE (magyarul: adatbázis neve): itt az magát az elnevezését kell megadnunk.
5. DB_USERNAME (magyarul: felhasználónév az adatbázishoz): meg kell adnunk egy felhasználónevet, amelyet az alkalmazás használ a belépéshez.
6. DB_PASSWORD (magyarul: jelszó az adatbázishoz): a belépéshez meg kell adnunk még egy jelszót is.

5.1.2. Migráció

A migrációk segítségével tulajdonképpen az adatbázis struktúráját tudjuk definiálni. Van arra lehetőség, hogy egy migrációs fájlt egy parancs segítségével le lehessen generálni struktúra szintjén, de már a benne szükséges mezőket és az esetleges különböző megszorításokat már nekünk kell definiálni. A következő parancs segítségével lehetséges egy migrációs fájlt generálni:

```
php artisan make:migration create_név_table .
```

Következőnek nézzük azt meg, hogy hogyan is épül fel egy migrációs fájlt, konkrétan az a hirdetéshez tartozó.

5.1. kód. Egy migrációra példa kód

```
1 <?php
2
3 use Illuminate\Database\Migrations\Migration;
4 use Illuminate\Database\Schema\Blueprint;
5 use Illuminate\Support\Facades\Schema;
6
7 return new class extends Migration
8 {
9     /**
10      * Run the migrations.
11      */
12     public function up(): void
13     {
14         Schema::create('advertisements', function (Blueprint $table) {
15             $table->id('advertisement_id');
16             $table->unsignedBigInteger('user_id');
17             $table->unsignedBigInteger('city_id');
18             $table->unsignedBigInteger('category_id');
19             $table->string('title');
20             $table->integer('price');
21             $table->text('description');
22             $table->string('mobile_number')->nullable();
23             $table->timestamps();
24
25             $table->foreign('user_id')->references('user_id')->on('users')
26                 ↳ ->onDelete('cascade');
27             $table->foreign('city_id')->references('city_id')->on('cities')
28                 ↳ ->onDelete('cascade');
29             $table->foreign('category_id')->references('category_id')->on('categories')
30                 ↳ ->onDelete('cascade');
31         });
32     }
33 }
```

```

31      * Reverse the migrations.
32      */
33      public function down(): void
34      {
35          Schema::dropIfExists('advertisements');
36      }
37 };

```

Látható tehát, hogy egy mezőt úgy szükséges megadni, hogy \$table->mező típusa, majd zárójelbe a mező nevét. A 15. sorban az elsődleges kulcs megadása látható. Az alatta lévő sorban pedig egy olyan mezőt hoztam létre, ami majd egy másik migrációs fájlból kapja majd meg az értékét, tehát ez egy idegen kulcs. A 25. sorban pedig az imént említett mezőre így lehetséges hivatkozni. Itt ennek a sornak végén lévő kód az azt jelenti, hogy ha kitörlődik az adott felhasználó, akkor a hirdetés is törlődni fog vele együtt. Alapértelmezetten egyik mező értéke sem lehet nulla, ha azonban szükségünk van rá, akkor meglehet tenni úgy, ahogyan én a 22. sorban csináltam. Ha elkészültünk a migrációs fájlokkal, akkor a következő parancs segítségével lehet majd végrehajtani azt:

php artisan migrate .

5.1.3. Modell

A laravel már tartalmazza az eloquentet, ami egy ORM (angoul: Object-Relational Mapping, magyarul: objektum-relációs leképzés). Amikor ezt használjuk, akkor minden egyes adatbázis táblához tartozik egy modell is, aminek a segítségével interaktálni tudunk a táblával. Az ilyenféle modellek használatával lehetséges beszúrni, módosítani vagy törölni az adott adatbázis táblából. Itt a modellekben még megadhatunk különféle metódusokat is és még itt van lehetőség az eloquent kapcsolatok megadására is. Ezekre még majd kitérek a példa bemutatása során.[12]

5.2. kód. Egy modellre példa kód részlet

```

1  <?php
2  class Advertisement extends Model
3  {
4      use HasFactory;
5
6      protected $table = 'advertisements';
7      protected $primaryKey = 'advertisement_id';
8
9      protected $fillable = [
10         'city_id',
11         'category_id',
12         'title',

```

```

13     'price',
14     'description',
15 ];
16 public function user()
17 {
18     return $this->belongsTo(User::class, 'user_id', 'user_id');
19 }
20 public function pictures()
21 {
22     return $this->hasMany(Picture::class, 'advertisement_id', '
        ↳ advertisement_id');
23 }
24 }

```

Az 5.2 kódrészleten a 6. és 7. sorban az adatbázis tábla neve és elsődleges kulcs megadása látható. Rögtön alatta pedig az látszik, hogy egy tömbbe megadom a kötelezően megadandó mezők nevét, amik nem lehetnek üresek. Itt található még egy belongsTo és egy hasMany kapcsolatra egy példa. Az első azt röviden az azt jelenti, egy modell egy másik modellhez tartozik, azaz, hogy egy hirdetés egy felhasználóhoz tartozik. Míg az utóbbi azt jelenti, hogy egy modell több modellhez tartozik, azaz egy hirdetéshez több kép is tartozhat. Egy modell létrehozása a következő parancs segítségével történhet meg:

php artisan make:model név .

5.1.4. Kontroller

A kontrollerek tulajdonképpen arra használatosak, hogy a különböző kéréséhez szükséges metódusokat itt definiálhatjuk. A kontrollerekben általában sokféle metódusokat írhatunk, azonban alap esetekben a következő lehetnek. Lehet index nevű, ami az összes adat (erőforrások) megjelenítésére szolgál, lehet még create (magyarul: létrehozás), ami az adatok bekéréséért felelős form, ehhez szorosan kapcsolódik a store (magyarul: tárol, mentés), ami pedig elmenti a kívánt erőforrást. Beszélhetünk még a show (magyarul: megnéz) és a destroy (magyarul: törlés) metódusokról, az előbbi egy konkrét erőforrást jelenít meg, míg utóbbi segítségével az erőforrást lehetséges kitörölni. Végezetül még két metódus van, az első az edit (magyarul: módosítás), ami a create metódushoz hasonlóan egy formot jelenít meg, aminek a segítségével lehetséges az adott erőforrás módosítása. A másik pedig az update (magyarul: módosítás), ez pedig konkrétan módosítja az imént említett edit formon az eszközölt változtatásokat. Példa képen a show metódust mutatom be Önnek. [12]

5.3. kód. Egy kontrollerben lévő metódus

```

1 <?php

```

```

2
3 namespace App\Http\Controllers;
4 use App\Models\Advertisement;
5 class AdvertisementController extends Controller
6 {
7     public function show($id)
8     {
9         $advertisement = Advertisement::find($id);
10
11         if (!$advertisement)
12         {
13             return redirect()->route('advertisements.index')->with('error'
14                 ↪ , 'Nincsen ilyen hirdetés!');
15         }
16
17         return view('advertisements.show', compact('advertisement'));
18     }
19 }

```

A következőben röviden megmagyarázom az 5.3 kódot. Látható, hogy a hirdetés-hez tartozó kontrollerben található a metódus, ami paraméterként kér egy id-t, az id segítségével tudom, hogy pontosan melyik hirdetést szeretném megjeleníteni. Ezután megkeresem az adott bekért id alapján a hirdetést, majd leellenőrzöm, hogy egyáltalán létezik-e, ha nem akkor hibát adok vissza, amit majd a blade tudunk majd felhasználni, ahhoz, hogy értesítsem a felhasználót a hibáról. Végül pedig, ha nincsen hiba, akkor megjelenítem egy másik blade-n a felhasználó számára. A compact nevű metódusra, azért van szükségem, hogy a blade-n majd megjeleníteni tudjam a kívánt hirdetést. A bladekről többet az 5.1.6 alfejezetben tudhat meg. Egy kontroller létrehozása a következő parancs segítségével történhet meg:

php artisan make:controller névController .

5.1.5. Web.php

A web.php nevű fájl, arra szolgál, hogy a kontrollerekben elkészített különböző metódusokhoz route-t (magyarul: útvonal) rendeljük hozzá. Ez azért szükséges, hogy a böngészőkben el tudjuk érni magát a konkrét blade-t, ami a kontrollerben definiált metódust fogja megvalósítani. Itt még annyit érdemes megemlítenem, hogy a middleware (magyarul: köztes szoftver) segítségével van lehetőség szabályozni, azt hogy ki férhesen hozzá az adott útvonalhoz. A következőben pedig egy útvonalat fogok mutatni, hogy el tudják képzelni, hogy hogyan is nézz ki.

5.4. kód. Egy web.php lévő útvonal

```

1 <?php

```

```

2
3 use Illuminate\Support\Facades\Route;
4 use App\Http\Controllers\AdvertisementController;
5
6 Route::get('/advertisements/{advertisementId}/show', [
    ↪ AdvertisementController::class, 'show']->name('advertisements.
    ↪ show');
7
8 require __DIR__ . '/auth.php';

```

Röviden az látható az 5.4 kód részletén, hogy a route és két darab kettőspont után van lehetőségünk a kérést típusát megadni, attól függően, hogy melyik metódust hívjuk meg. Jelen esetben most get (magyarul: kap) található ott, ami tulajdonképpen a listázásért felelős. Gyakran használatosak még a post (magyarul: hozzáadás, beszúrás), a put (magyarul: módosítás) és a delete (magyarul: törlés) elnevezések. Ezt követően következik az első paraméter, ami az elérési útvonal, a másik pedig azon az útvonalon található metódust lesz. Még megtalálható a végen ott egy elnevezés metódus, aminek a segítségével könnyebben tudunk majd hivatkozni a kívánt blade-n. Ennek a használata azért előnyös, mivel így nem kell a teljes útvonalat megadni, ha hivatkozni szeretnénk a metódusra és ha esetlegesen változtatni kellene az útvonalon, akkor azt csak egy helyen elég megtenni, itt a web.php-ben. Ha nem neveznénk el és több blade is használná ezt az útvonalat, akkor mindenhol meg kellene változtatni, ami nem lenne túl szerencsés. A bladekről többet az 5.1.6 alfejezetben tudhat meg.

5.1.6. Blade

A bladek segítségével azokat nézeteket tudjuk definiálni, amiket látni lehet a böngészőben. Tulajdonképpen ez lenne a UI (user interface), azaz a felhasználói felület. Egy hirdetés megjelenítéséért felelős felület szeretném bemutatni, amiről már szó esett az előző két alfejezetben. Ezen az 5.5 kód részletén láthatja.

5.5. kód. Egy hirdetés megjelenítésére szolgáló felület

```

1 @extends('layouts.app')
2 @section('content')
3     <div class="min-h-screen container mx-auto mt-8">
4         <div class="flex justify-center">
5             <div class="w-2/3">
6                 <div class="bg-white p-6 rounded-lg shadow-md">
7                     <table class="table-auto w-full">
8                         <thead>
9                             <tr>
10                                <th class="px-4 py-2">Kép</th>
11                                <th></th>
12                                <td class="px-4 py-2">

```

```

13         <div class="grid grid-cols-5 gap-2">
14             @foreach ($advertisement->pictures as
15                 ↪ $i => $picture)
16                 <a href="{\asset('storage/' . \
17                     ↪ $picture->src)}" data-
18                     ↪ title="Image{\$i+\1}"
19                     ↪ >
20                     
22                 </a>
23             @endforeach
24         </div>
25     </td>
26 </tr>
27 <tr>
28     <th class="px-4 py-2">Cím</th>
29     <td class="px-4 py-2">{{ $advertisement->
30         ↪ title }}</td>
31 </tr>
32 <tr>
33     <th class="px-4 py-2">Város</th>
34     <td class="px-4 py-2">{{ $advertisement->
35         ↪ city->name }}</td>
36 </tr>
37 <tr>
38     <th class="px-4 py-2">Vármegye</th>
39     <td class="px-4 py-2">{{ $advertisement->
40         ↪ city->county->name }}</td>
41 </tr>
42 <tr>
43     <th class="px-4 py-2">Kategória</th>
44     <td class="px-4 py-2">{{ $advertisement->
45         ↪ category->name }}</td>
46 </tr>
47 <tr>
48     <th class="px-4 py-2">Ár</th>
49     <td class="px-4 py-2">{{ $advertisement->
50         ↪ price }}</td>
51 </tr>
52 <tr>
53     <th class="px-4 py-2">Leírás</th>
54     <td class="px-4 py-2 break-all">{{
55         ↪ $advertisement->description }}</td>
56 </tr>
57 <tr>
58     <th class="px-4 py-2">Eladó</th>
59     <td class="px-4 py-2">{{ $advertisement->
60         ↪ user->name }}</td>
61 </tr>
62 </tbody>
63 </table>
64 </div>

```

```

49         </tr>
50         <tr>
51             <th class="px-4_py-2">Telefonszám</th>
52             <td class="px-4_py-2">{{ $advertisement->
53                 <!-- user->name }}</td>
54             </tr>
55         </thead>
56     </table>
57 </div>
58 </div>
59 </div>
60 @endsection

```

Erről csak röviden szeretnénk beszélni, rögtön az elején van ként részlet, ami említésre méltó. Az `extends` (magyarul: kibővít) kulcsszó segítségével tudjuk bővíteni egy másik felületet, a másik kulcsszó pedig a `section` (magyarul: szekció), ami pedig a kibővített felületen belüli content (magyarul: tartalom) helyére kerül be a taglalt felület. Táblázatos formában jelenítettem meg a hirdetést, és ennek a `tailwind css` (2.1.9) segítségével való formázás található még ott. Az 5.1.4 alfejezetben esik szó a `compact` nevű metódusról, aminek a használata azért volt szükséges, hogy elérjük a `$advertisement` (magyarul: hirdetés) változót. Még egy dolgot szeretnénk kiemelni, ami pedig a 36. sorban olvasható. Erről már volt szó ebben az 5.2 alfejezetben. Itt a modellek miatt tudom ezt megtenni, hogy így megtudjam jeleníteni a hozzá tartozó kategória nevét, ennek a hiányában a hirdetéshez tartozó kategória `id`-t tudnám csak megjeleníteni.

Összegzés

Irodalomjegyzék

- [1] KUSPER GÁBOR: *Programozási technológiák*, Eger, 2015.
- [2] KUSPER GÁBOR: *Informatikai rendszerek tervezése*, Eger, 2023, 0.8.7.2-es verzió
- [3] KUSPER GÁBOR: *Szoftvertesztelés*, Eger, 2023, 2023.10.14-es verzió
- [4] ORACLE: <https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>, 2024-es verzió.
- [5] PHPMYADMIN: <https://www.phpmyadmin.net/>, 2024-es verzió.
- [6] GITHUB DESKTOP: <https://docs.github.com/en/desktop/overview/about-github-desktop>. 2024-es verzió.
- [7] PLANTUML: *PlantUML Language Reference Guide*, <https://plantuml.com/guide>, 2023. novemberi verzió.
- [8] CYPRESS: <https://docs.cypress.io/guides/overview/why-cypress>, 2024-es verzió.
- [9] DBDIAGRAM: <https://dbml.dbdiagram.io/docs/>, 2024-es verzió.
- [10] KATALON: <https://katalon.com/resources-center/blog/end-to-end-e2e-testing>, 2024-es verzió.
- [11] PHP: <https://www.php.net/docs.php>, 2024-es verzió.
- [12] LARAVEL: <https://laravel.com/docs/10.x>, 2024-es verzió.
- [13] TAILWIND CSS: <https://tailwindcss.com/>, 2024-es verzió.
- [14] MEDIUM: <https://skakarh.medium.com/advantages-and-disadvantages-of-cypress-end> 2024-es verzió.