



Webes vizsgáztató rendszer

Készítette

Kecse Károly Dániel
Programtervező informatikus

Témavezető

Dr. Király Roland
egyetemi docens

EGER, 2023

Tartalomjegyzék

Bevezetés	4
1. A hasonló alkalmazásokról	6
1.1. Alternatívák és jellemzők, észrevételek	6
1.1.1. Moodle	6
1.1.2. Pegazus	6
1.1.3. Redmenta	7
1.1.4. Quizizz	7
1.2. Következtetések	7
2. Felhasznált technológiák	9
2.1. Laravel	9
2.2. Livewire	10
2.3. MySQL	10
2.4. Tailwind	11
3. Adatbázis	12
3.1. A kurzusok felépítése	13
3.1.1. Kurzus	14
3.1.2. Modul	14
3.1.3. Teszt	14
3.1.4. Vizsga feladatsor kitöltése	15
3.2. A csoportok felépítése	15
4. A projekt megvalósítása	17
4.1. A Laravel alap működése	17
4.1.1. Migration	17
4.1.2. Model	18
4.1.3. N+1 Query Problem	19
4.1.4. Controller	20
4.1.5. Refaktorálási kísérletek	21
4.1.6. View és Blade fájlok	23

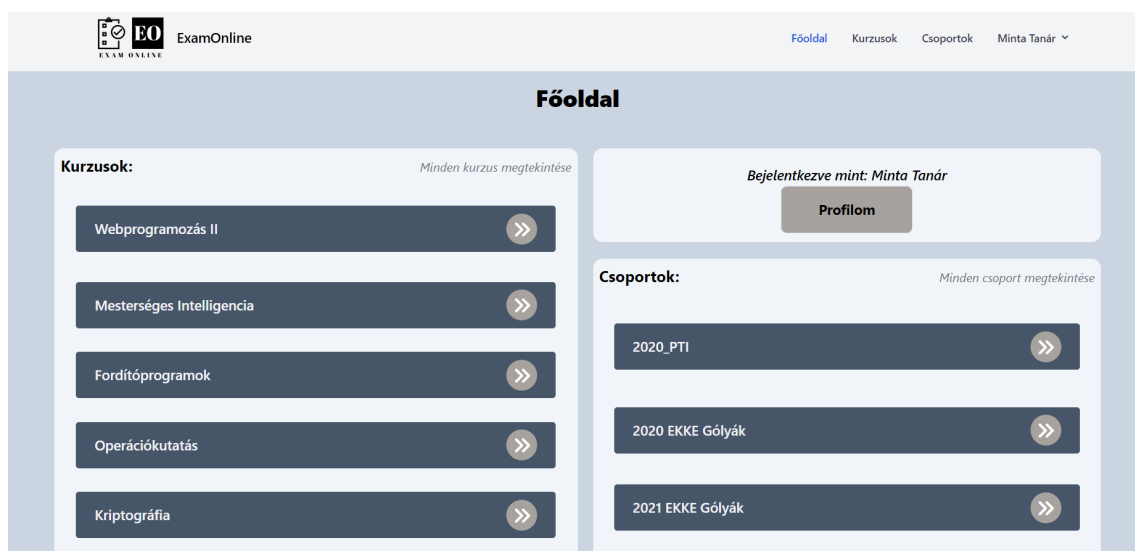
4.2.	Autentikáció	25
4.2.1.	Laravel/UI	25
4.2.2.	Middleware	25
4.2.3.	Szabályzatok (Policies)	26
4.3.	Livewire működésének megértése	28
4.3.1.	Oldalak feldolgozása	28
4.3.2.	Tulajdonságok (Properties)	29
4.3.3.	Műveletek és Események (Actions & events)	29
4.3.4.	Sortable csomag	30
4.3.5.	Valós idejű validáció	31
5.	A csoportok bemutatása	33
5.1.	Felhasználók hozzárendelése a csoportjainkhoz	33
5.2.	Felhasználók eltávolítása a csoportjainkból	34
5.3.	Chat: Pusher és Echo	35
6.	A kurzusok bemutatása	37
6.1.	Modulok	38
7.	A tesztek felépítése és kitöltése	39
7.1.	Service-ek	39
7.2.	Feladatlapok megvalósítása	40
7.3.	Teszt kitöltésének folyamata	41
7.3.1.	Kvízek	41
7.3.2.	Vizsgák	42
8.	Az alkalmazás tesztelése	44
9.	Külső komponensek, technológiák	46
9.1.	SweetAlert	46
9.2.	Spatie süti engedélyező	46
9.3.	TinyMCE	47
9.4.	Font Awesome	47
9.5.	Mailtrap	48
10.	Telepítési útmutató	50
	Irodalomjegyzék	53

Bevezetés

A szakdolgozat célja, hogy egy olyan rendszert biztosítsak az iskoláknak, amivel lehetőséget adok a tanároknak, hogy egy helyen tárolják az órákhoz létrehozott tananyagot az ezekhez kapcsolódó számonkérési és gyakorlási lehetőségekkel együtt, amiknek a kitöltésére egy jól szabályozható felületet biztosítunk a diákoknak.

Egy másik elvárásom az volt, hogy a szoftvert tetszőleges eszközön lehessen használni úgy, hogy a megjelenítés minden esetben komfortos legyen és ne érezzük kényelmetlen egyetlen funkciónak a használatát sem, legyen az akár egy kisebb kijelzőjű mobil vagy egy szélesebb monitor. Így a választásom a webes megjelenítésre esett, aminek egyetlen feltétele a megfelelő működéshez, hogy stabil internet hozzáféréssel rendelkezünk.

A fenti elvárásoknak megfelelően megszületett a *webes vizsgáztató rendszer*, amely egy olyan webes alkalmazás, amelyben lehetőségünk van arra, hogy csoportokat kezeljünk, esetlegesen annak tagjaival csevegjünk, vagy kurzusokon vegyünk részt. A bejelentkezést követően egy olyan főoldal (1. ábra) fogad minket, amely kilistázza az elérhető kurzusainkat illetve csoportjainkat.



1. ábra. Bejelentkezést követő főoldal

A projekt egyik fő részét a csoportok képzik. Itt nincsen semmilyen korlátozás arra, hogy ki hozhat létre, kezelhet csoportokat. Egyetlen megkötés, hogy az adott csoporthoz való műveletek (frissítés és törlés) csak az azt létrehozó felhasználónak vannak

biztosítva. A csoportok bemutatásáról, funkcióiról és képernyőképeiről részletesebb leírást az 5. fejezetben olvashatnak.

A rendszer másik, legnagyobb és legjelentősebb egységét a kurzusok jelentik. Ezek segítségével tudtam tananyagokat és tesztek készíteni a különböző tantárgyakhoz. Ennek az egységnek a működése nagyban függ attól, hogy milyen a bejelentkezett felhasználónak a jogosultsági köre.

Amennyiben tanárok vagyunk, lehetőségünk nyílik arra, hogy kurzusokat készítsünk, ahol tananyagokat illetve tesztek hozhatunk létre. Megszabhatjuk, hogy mely csoportok és milyen diákok érhetik el az itt található tartalmakat, illetve milyen időintervallumban férhetnek hozzá bizonyos feladatsorokhoz.

Tanulóként megtekinthetjük a kurzus tananyagát, kitölthetjük a tesztek, viszont a vizsgák kitöltésének eredményei közül csak a sajátunkat láthatjuk.

Ezzel ellentétben a tanár megtekintheti a kurzus minden tagjának az eredményét. Akár azt is megteheti, hogy bizonyos próbálkozásokat töröl, ilyenkor növelve a lehetséges kitöltések számát az adott diáknak. A kurzusokról később bővebben a 6. fejezetben fogok írni. Az ezekhez tartozó különböző tesztek (felépítés, működés, különbségek) a 7. fejezetben olvashatnak.

Egy harmadik, kisebb, ám annál jelentősebb funkció a tanárok regisztrálása. Mivel a navigációs sávban mindenki által elérhető regisztrációs felülettel csak tanulóként regisztrálhatunk, szükségessé vált az, hogy legyen egy olyan jogosultsági kör, aminek egyetlen feladata a tanárok felvitele a rendszerbe. Ez az admin felhasználó lesz (akit a Seeder segítségével adhatunk az adatbázishoz), akinek a „Kurzusok” és „Csoportok” helyett egy „Tanár hozzáadása” menüpont lesz látható a bejelentkezés után, amely segítségével oktatókat vihet fel a rendszerbe.

A szakdolgozat további részében megtalálható azoknak az alkalmazásoknak az elemzése, amelyek alapján született az alap elképzelés a rendszerről (1. fejezet).

Ezt követően az adatbázis felépítését fogom tárgyalni a 3. fejezetben, ahol a két fő egység elkészítéséhez szükséges táblákat, illetve azok szükségességét fogom taglalni.

Ezek után felvezetem a 2. fejezetben azokat a fontosabb keretrendszereket és technológiákat, amelyek nagyban hozzájárultak a projekt elkészítésében. Az itt tárgyalt Laravel és Livewire keretrendszerekről a 4.1. és a 4.3. fejezetben fogok részletesebben írni, ahol a felhasznált elemek működési elvét ismertetem példakódok segítségével.

A 9. fejezetben a külső komponensekről fogok említést tenni, amiben különböző csomagok, könyvtárak és tesztelési környezetek leírásáról olvashatnak.

1. fejezet

A hasonló alkalmazásokról

A fejezetben összegyűjtöttem azokat az alkalmazásokat, amiket a témakörben relevánsnak ítélt meg. Ezek közül a legtöbbhöz volt szerencsém eddigi tanulmányaim alatt, ezért van némi tapasztalatom is velük. Az alábbi rendszerek tüzetesebb elemzése után igyekeztem kigyűjteni azokat a funkciókat, amikkel kapcsolatban pozitív volt az élményem illetve elengedhetetlennek ítélt meg.

1.1. Alternatívák és jellemzők, észrevételek

1.1.1. Moodle

Az Egyetem elearning rendszerét[1] használtuk a legtöbb órán, ezért erről van az egyik legjobban kirajzolódott véleményem. A projektben belüli kurzusok felépítése ehhez a rendszerhez hasonlít, hiszen kifejezetten tetszett, hogy a számonkérés és a tananyagnak az elérése egy egységes rendszerben elérhető. Ennek a rendszernek talán két dolgot lehetne felróni: rengeteg olyan felesleges információt tár elénk az oldal, amelyet valószínűleg soha nem fogunk használni, ez pedig meglehetősen csökkenti a felület átláthatóságát. A másik, ettől sokkal kisebb probléma, hogy a számonkéréseknek a feladattípusai számomra eléggé egyhangúak.

1.1.2. Pegazus

A Pegazus[2] egy általunk kevésbé használt alkalmazás, így sajnos csak minimális személyes tapasztalatom van vele kapcsolatban, éppen ezért hallgató társaimtól is érdeklődtem, hogy milyen véleményük van a használatával kapcsolatban. Az első dolog, amit észrevettem, hogy a moodle-höz képest rendkívül letisztult és szép megjelenéssel rendelkezik az oldal. Emellett nagyon sokféle feladattípus közül lehet választani, ami nagy előny. A másik pozitívum, hogy itt is elérhető a kurzusokhoz tartozó tananyag a számonkéréssel együtt (bár azt hallottam, hogy a tesztek használata nem túl elterjedt

a rendszeren belül). Az én célom egy ehhez hasonló rendszer kialakítása volt, amely gyorsabb oldalelérést tesz lehetővé, és amely a diákok számára lehetővé teszi, hogy közelebb érezzék magukat egymáshoz a használat során.

1.1.3. Redmenta

A Redmenta[3] egy olyan webes alkalmazás, amely a Pegazus sokszínű feladattípus kínálatával vetélkedik, így biztosítva egy elképesztően változatos vizsgázási élményt. Felhasználási köre leginkább az általános és középiskolák körében elterjedt, viszont volt már rá példa, hogy itt, az Egyetemen is ezt alkalmaztuk egy online vizsga keretén belül. Egyedül tesztek készítésére és kitöltésére használható, ezért a Redmenta tudása számomra kissé korlátozott.

1.1.4. Quizizz

A Quizizz[4] egy olyan alkalmazás, amelyet leginkább kvízek írására fejlesztettek ki, és hasonlóan a Redmentához, lehetővé teszi a tesztek készítését. A Quizizz különböző animációkkal, effektekkel, játékos elemekkel próbálja szórakoztatóvá tenni a tanulást, ezért is elterjedtebb általános iskolák körében. Sajnos a legtöbb funkció előfizetéshez kötött, így az alkalmazásban való tevékenységünk kissé korlátozottabb lehet, mint például a Redmenta vagy a Pegazus esetében.

1.2. Következtetések

A fentiekben felsoroltak alapján felállítottam néhány elvárást, amiknek a szem előtt tartásával terveztem meg és implementáltam a projekt egyes funkcióit. Ezek az alábbiak, amiknek a leírásáról a többi fejezetben esik majd szó:

- Letisztult, felhasználó barát felület (mobilra is)
- Tananyag elérése a számonkéréssel együtt egy rendszerben
- Változatos feladattípusok (jelenleg 4 típus)
- Gyakorlási lehetőség biztosítása az egyes órákhoz
- Olyan funkciók készítése és megjelenítése, amelyeket én is szívesen látnék, mint diák

Sajnos a fentiekben felsorolt alkalmazásokhoz képest ez a projekt eléggé gyerekcipőben jár, viszont ezeknek a jól bevált módszereit igyekeztem úgy implementálni, hogyha nem is módosítás nélkül, de minimális átdolgozással problémamentesen bővíthető legyen.

Ilyen például több feladattípus felvitele, tananyaghoz fájlok feltöltésének biztosítása, statisztikák lekérdezése a megadott válaszoknak megfelelően.

2. fejezet

Felhasznált technológiák

2.1. Laravel

A Laravel[5] egy PHP nyelvhez készített nyílt forráskódú, ingyenes backend keretrendszer, amely MVC tervezési mintát használ. Ez annyit tesz, hogy a rendszerünk működése 3 rétegre van felosztva:

- **Model:** Osztályok használatával biztosít adatbázison végezhető műveleteket metódushívásokon keresztül.
- **View:** Egy olyan nézet, ahol általában a kontrollerből érkező adatokat jelenítjük meg. Az oldalon biztosíthatunk a felhasználónak navigációs lehetőségeket különböző útvonalak használatával, amelyek egy-egy Controller függvényét hívják meg.
- **Controller:** A View és a Model között biztosít összeköttetést. Az oldalakról esetlegesen érkező adatok segítségével végezhet műveleteket egy adott modellen, vagy a megjelenítendő oldalnak előállíthatja a szükséges adatokat. Egy kontrollerben lefutott utasítássorozat után vagy visszaad egy nézetet az összeállított adatokkal együtt, vagy átnavigál minket egy másik útvonalra.

A Laravel egyik legnagyobb előnye a számomra, hogy rendkívül jól dokumentált, így nem jelentett különösebb problémát a felmerülő hibák megoldását megtalálni, illetve eddig ismeretlen lehetőségek után keresgélni (például Queued Job, Broadcast). Folyamatosan fejlesztik, hiszen 2023 februárjában került kiadásra a Laravel 10-es verziója (még ha az eddigi egyik legkisebb frissítés is). Egyszerű módszerrel biztosítja az egyes útvonalak (route-ok) kezelését különböző HTTP kérési típusok megkülönböztetésével együtt. Másik pozitívuma, hogy a felhasználói autentikációhoz is van rengeteg már elkészített és megfelelően integrált megoldás, így sem a bejelentkezést, sem a munkamenetkezelést nem kell az alapjaitól kezdve megvalósítanunk.

A Laravel alaphól érkezik egy Artisan-nal, ami egy CLI (Parancssori Felhasználói Felület). Ennek segítségével a projektünkben generálhatunk fájlokat (Migration, Model, Controller, Resource, Job, Policy), lefuttathatjuk a megírt Migration fájljainkat illetve a projektnek a szerveren való futtatását is elvégzi.

2.2. Livewire

Szerettem volna egy olyan frontend-kezelő keretrendszert használni, amihez nem kell több hónapos tapasztalat ahhoz, hogy bonyolultabb feladatokat is viszonylag gyorsan és hatékonyan tudjak implementálni, illetve könnyedén tudjam a Laravel-es projekten belül használni. Így választásom a Livewire-re[6] esett, ami egy kifejezetten Laravel-hez írt fullstack keretrendszer. Egyszerűvé teszi a dinamikus felületek létrehozását, az összetettebb űrlapkezelést – mindezt úgy, hogy közben maradhatunk a jól megszokott Laravel szintaxisoknál.

Az ilyen módon készített fájlok sem különböznek túlságosan az egyszerű Laraveles működéstől: egy LiveWire komponens létrehozása során kapunk egy külön Livewire Controller-t illetve egy Blade fájlt, amelyek szoros kapcsolatban állnak egymással. A használatához ezt kell majd beimportálnunk azokon az oldalakon, ahol szeretnénk a már kész komponenseinket használni.

Ilyen technikával készültek a projekt tesztjeinek létrehozását, szerkesztését, kitöltését, valamint a csoportok és felhasználók keresését biztosító elemek. Egyszerű asszociatív tömbökkel tudunk dolgozni, amiknek az értékeit hozzárendelhetjük az űrlapunk egyes beviteli mezőinek értékéhez.

A backend és frontend kommunikációja a „*render()*” metóduson keresztül biztosított, amit az oldalak betöltésénél illetve minden, a komponensen végrehajtott módosításkor lefut (hacsak másként nem határoztunk), így biztosított a felhasználói interakciónak a megfelelő lereagálása.

A Livewire komponensek működésének részletesebb leírásáról a 4.3. fejezetben olvashat.

2.3. MySQL

Az adatbázis rendszer kiválasztásánál törekedtem arra, hogy az ingyenes felhasználás mellett a teljesítménye elég legyen viszonylag nagy számú felhasználók esetén is, illetve a Laravel-lel való támogatottsága is biztosított legyen.

A MySQL egy ingyenesen használható, nyílt forráskódú adatbázis-kezelő. Egyike a legszélesebb körben használt rendszereknek, így a támogatottsága is biztosított a Laravel-en belül. Bármilyen hiba vagy probléma esetén könnyen utána tudok nézni

a dokumentációban és a StackOverflow oldalon is, mivel rengeteg hasznos bejegyzés található ezeken a platformokon.

Annak ellenére, hogy nem kell konkrét lekérdezéseket írni, a fejlesztés során egy rendkívül hasznos eszköz a phpMyAdmin, ami egy grafikus felületen jeleníti meg a web-szerverünkön futtatott adatbázisokat. Megnézhetjük az adatbázis tábláit, azok szerkezetét, illetve az azokon belül eltárolt rekordok adatait is megtekinthetjük csupán néhány kattintással.

2.4. Tailwind

A Tailwind[7] egy olyan széleskörűen elterjedt CSS keretrendszer, amely használatával előre definiált osztályok segítségével tudunk egyszerű, a CSS-hez nagyon hasonló – gyakran annak csak rövidítése – hivatkozások használatával új megjelenítést adni az oldalainknak.

Könnyedén biztosíthatjuk az oldalaink megfelelő megjelenítését különböző méretű kijelzőkön (small, medium, large, extra-large). Ezt előre meghatározott szélességű töréspontokkal teszi lehetővé, így mindig a jelenlegi képernyőméretnek megfelelő attribútumot értelmezi az oldalon megtalálható stílusok közül.

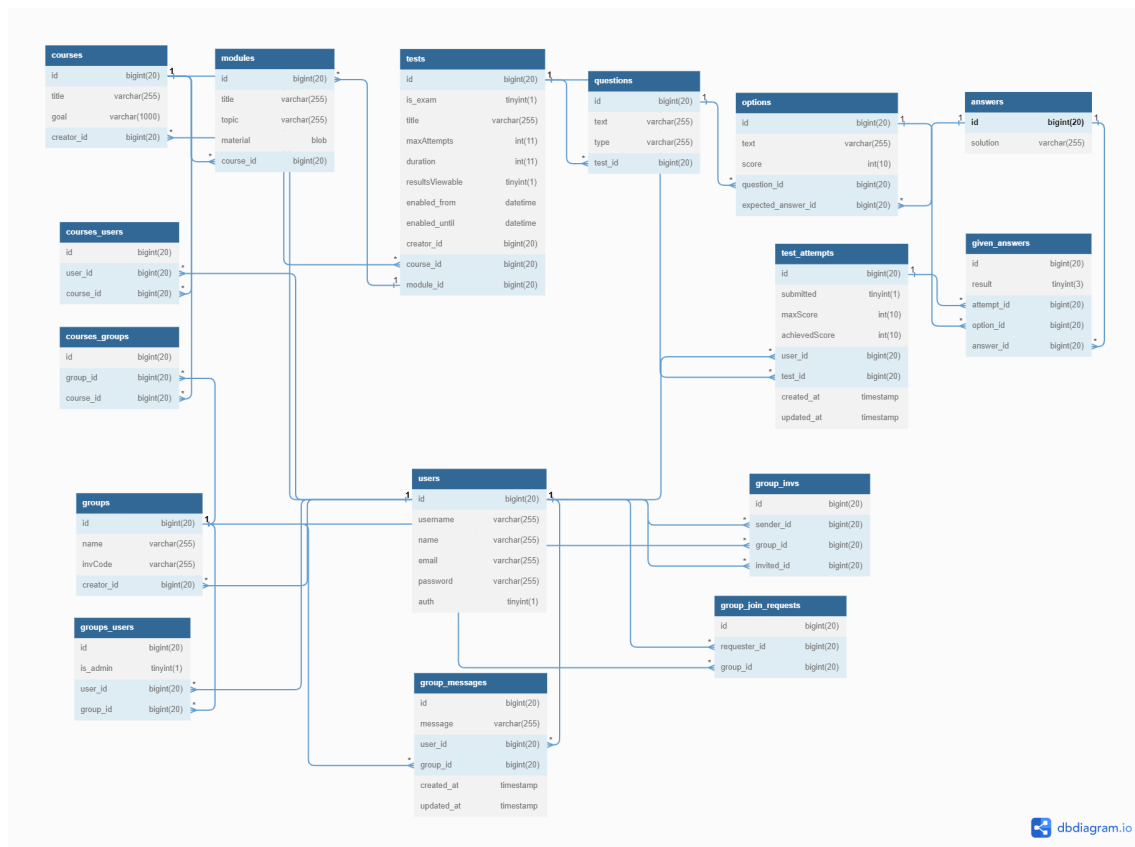
A töréspontok definiálása mellett lehetőséget kapunk arra is, hogy az egyes elemeken végbemenő események közül lekezeljük azokat, amik valamilyen jelentőséggel bírnak az oldal megjelenítése során. Ilyen például amikor a kurzort ráhelyezzük egy nyomógombra és annak háttérszíne megváltozik. Erre egy példa, amikor a „*hover:bg-green-700*” osztály írja felül az eddig aktív „*bg-green-500*” Tailwind osztályt, így zöldről sötétzöldre vált a háttérszíne az adott elemnek.

Másik hatalmas előnye, hogy rengeteg példa kód és komponens található a Tailwind dokumentációján belül, ahol szinte minden attribútum esetén megtalálhatjuk, hogy milyen intervallumban és értékekkel hivatkozható, valamint hogyan hozhatunk létre saját igényeinknek megfelelő, egyedi osztályokat a projektjeinken belül.

3. fejezet

Adatbázis

Ebben a fejezetben az adatbázis felépítését fogom ismertetni. Az itt látható ábrák a dbdiagram[8] segítségével készültek. A 3.1 képen látható a teljes adatbázis felépítése.

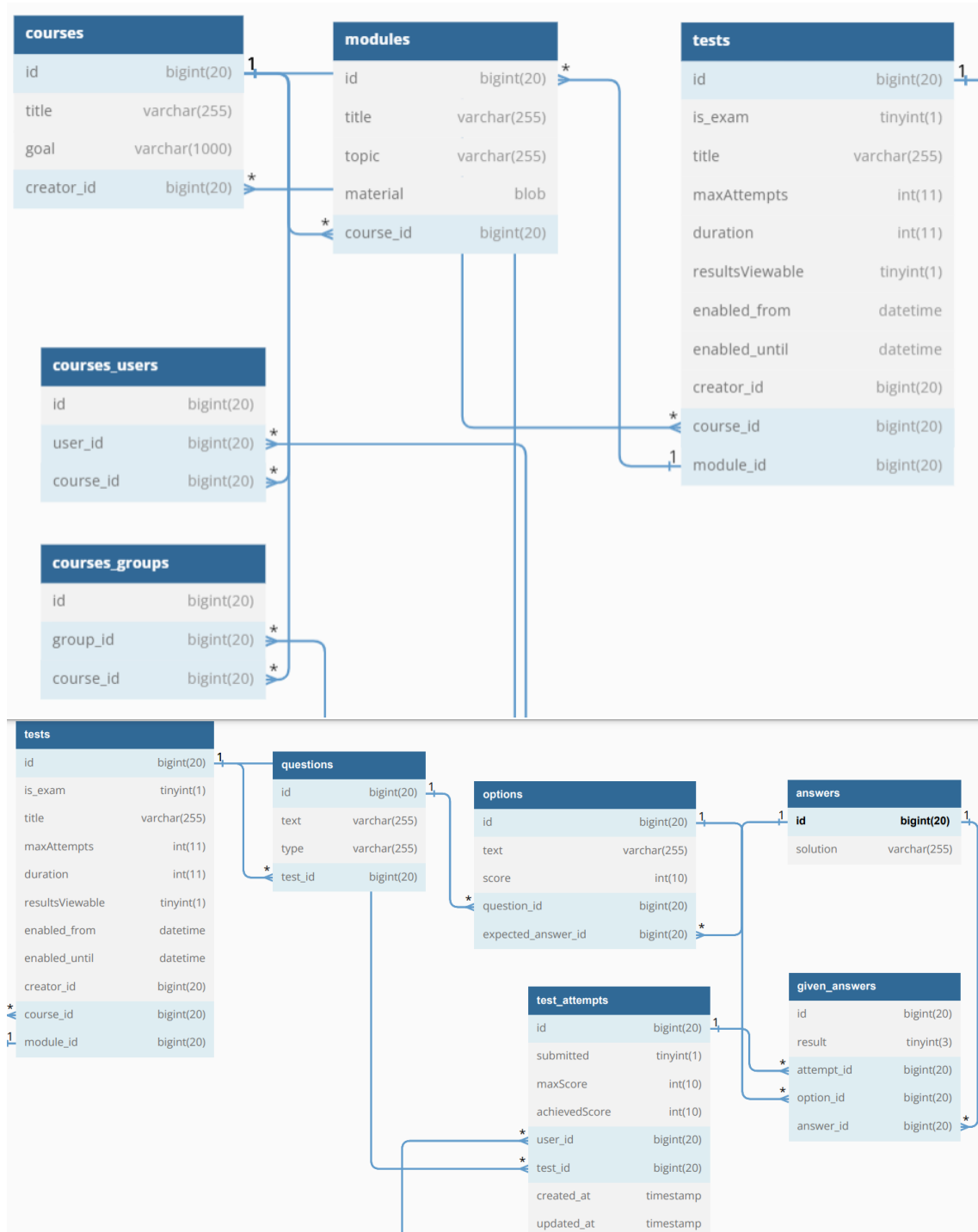


3.1. ábra. Az adatbázis felépítése

A következő fejezetekben részletezni fogom a táblák közti kapcsolatokat úgy, hogy felosztom két nagyobb szekcióra a projektet.

3.1. A kurzusok felépítése

A projekt egyik fő logikai egységét képezik a kurzusok, amelyek azzal a céllal jöttek létre, hogy legyen egy olyan gyűjtő egységünk, amely magába foglalja az egyes órákhoz tartozó tananyagokat (modulok), gyakorló feladatokat (kvízek) és vizsga feladatsorokat.



3.2. ábra. A kurzusokkal kapcsolatos táblák az adatbázisból

3.1.1. Kurzus

Minden kurzus rendelkezik egy megnevezéssel, illetve a képzés céljának leírásával. Itt eltároljuk a kurzust létrehozó felhasználó azonosítóját, hiszen csak neki fogunk biztosítani lehetőséget a módosításhoz, a résztvevők kezeléséhez és a törléshez.

A kurzusokhoz hozzárendelhetünk embereket – ez a kapcsolat a „*courses_users*” táblában fog eltárolódni, ami a *több-a-többhöz* kapcsolathoz létrehozott kapcsolótábla-, illetve csoportokat – ez pedig a „*courses_groups*”-ban fog rögzítésre kerülni.

3.1.2. Modul

Modulokat tudunk létrehozni a kurzusainkhoz, amik tartalmazzák az egyes órák tananyagait. Az elkészítés során meg kell adni az óra címét, a témakör megnevezését illetve a tananyagot. Itt biztosítjuk az *egy-a-többhöz* kapcsolatot a „*modules*” tábla „*course_id*” mezőjében.

3.1.3. Teszt

A kétféle teszt típus miatt két idegen kulcsot használunk ebben a táblában:

- Kvízhez: rögzítjük, hogy a teszt melyik modulhoz készült (ez lehet null is, ilyenkor csak szimplán, esetlegesen egy vizsga feladat gyakorlására készülhetett) a „*test.module_id*” mezőjében.
- Vizsga feladatsorhoz: rögzítjük, hogy melyik kurzushoz készült a „*test.course_id*” mezőjében.

Azt, hogy éppen milyen az adott teszt típusa, az „*is_exam*” boolean mező értékével adhatjuk meg.

Ha vizsga feladatsort hozunk létre, megadhatjuk, hogy a kurzushoz rendelt felhasználók és a csoportok tagjai mettől-meddig érhetik el a teszt kitöltését (év, hónap, nap, óra, perc-re pontosan) az „*enabled_from*” és „*enabled_until*” mezők segítségével.

További, kötelezően megadandó teszt attribútumok: megnevezés (*title*), láthatóak-e a megoldások (*resultsViewable*), kitöltéshez rendelkezésre álló idő percben (*duration*).

Minden teszthez tartozik legalább 1 kérdés (*question*), aminek van:

- Szövege
- Típusa (TrueFalse, OneChoice, MultipleChoice, Sequence)
- Több válaszlehetősége (*option*), amiknek attribútumai:
 - „*question_id*”-vel eltárolom, hogy melyik kérdéshez tartozik az opció,
 - Szöveg (*text*),

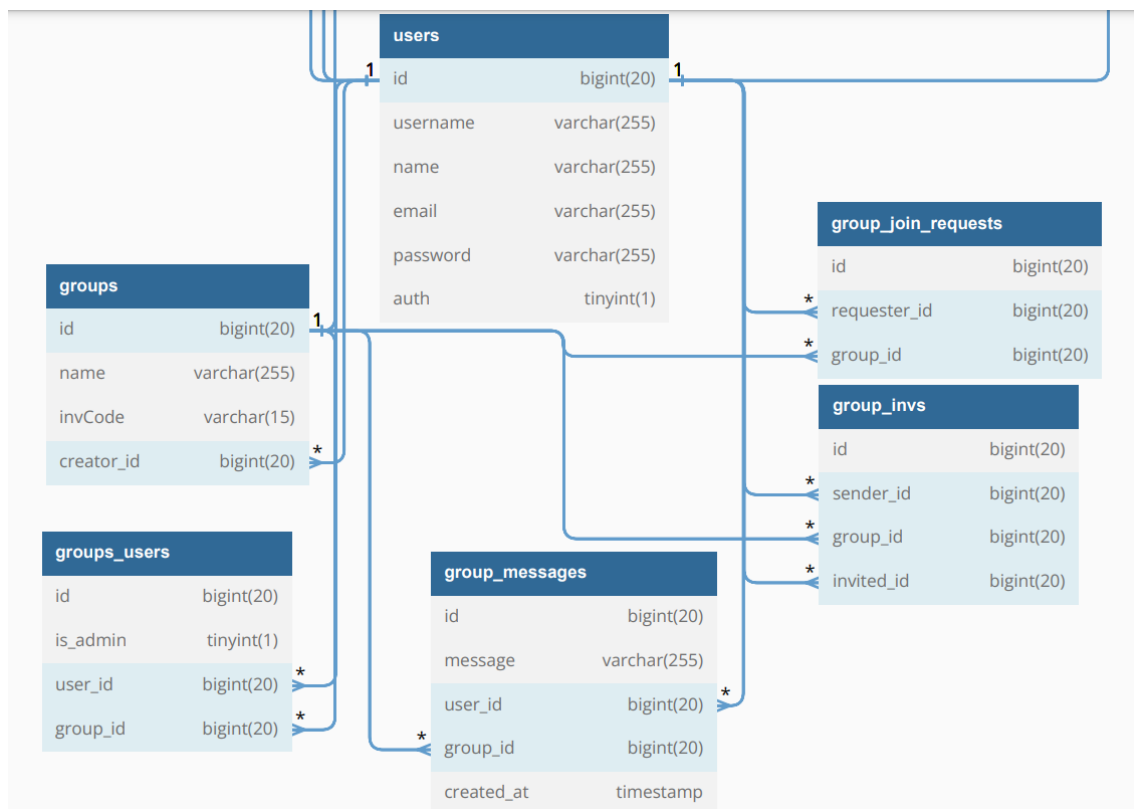
- Hány pontot ér (0 vagy 1, *score*),
- Melyik kérdéshez tartozik (*question_id*)
- Elvárt válasz azonosítója (*expected_ans_id*). Ezt egy külön táblában, az „*answers*”-ben tároltuk el, hiszen így elkerülhető a válaszok nagy mértékű redundanciája

3.1.4. Vizsga feladatsor kitöltése

Egy próbálkozás indításánál létrehozunk egy új rekordot a „*test_attempts*” táblában, ahol eltároljuk, hogy *ki*, *melyik teszthez*, *mikor* kezdte meg a kitöltést. A feladatlap kitöltésének végeztével kiszámoljuk a maximálisan elérhető illetve az elért pontszámokat, amiket szintén eltárolunk.

Az eredmények későbbi ellenőrzéséhez a feladatlapon rögzített válaszokat elmentjük a „*given_answers*” táblában, ahol megadjuk, hogy melyik próbálkozáshoz adtuk le az egyes válaszainkat, mi az a válaszlehetőség, amihez tartozik a válaszuk, illetve a diák által adott érték helyes volt-e az elvártakhoz képest, vagy sem.

3.2. A csoportok felépítése



3.3. ábra. A csoportokkal kapcsolatos táblák az adatbázisból

A másik fő egységet képezik a csoportok, amik segítségével egy valódi közösséget tudunk teremteni mind a diákok, mind a tanárok körében úgy, hogy egy csevegő felület segítségével kommunikálhatnak egymással a csoport tagjai. Ezek mellett arra is egyszerű megoldás, hogy egyszerűsítsük a tanulóknak a kurzusokhoz való hozzárendelés folyamatát, hiszen ennek segítségével egyszerre több tanulónak is elérhetővé tehetjük a tananyagainkat.

A csoportról csupán néhány információt tárolunk el a *groups* táblában:

- Mi a neve? (*name*)
- Meghívó kód (automatikusan generált, *invCode*)
- Ki hozta létre? (*creator_id*)

Két csatlakozási lehetőséget biztosítottam a felhasználók számára:

- A felhasználó egy adott csoport egyedi meghívó kódját elküldve rögzíti jelentkezési kérelmét a „*group_join_requests*” táblában, amit majd később a csoport tulajdonosa vagy elfogad, vagy elutasít.
- A csoportot létrehozó felhasználó a kereső segítségével meginvitálja az adott csoportba a kikeresett felhasználó(ka)t. Ilyenkor a kérelem/kérelmek a „*group_inv*” táblában kerül eltárolásra, ahol megtudjuk, hogy ki hívott meg (*sender_id*) kit (*invited_id*) és milyen csoportba (*group_id*).

Amint a csoport tagjaivá válunk, elérhetővé válik számunkra egy „Üzenetek” felület, aminek adatait a „*group_messages*” táblában tároljuk.

4. fejezet

A projekt megvalósítása

Ebben a fejezetben a projektben alkalmazott technológiát mutatom be úgy, hogy a Laravel alap működését magyarázom el példák segítségével, kitérve arra, hogy mit, miért és hogyan implementáltam.

4.1. A Laravel alap működése

4.1.1. Migration

A Laravel Migration-je[9] egy egységes sémát biztosít az adatbázisunk tábláinak a felépítésére, amelyet megoszthatunk a projekten dolgozó többi fejlesztővel. Ez azért kifejezetten hasznos, mert a manuális tábla-módosítás helyett csupán csak a módosított migration-t kell futtatni, így amellett, hogy energiát és időt spórolunk meg, sokkal kisebb az esetleges emberi pontatlanságokból adódó hibák száma.

Egy migration-t legegyszerűbben a *php artisan* segítségével generálhatunk le. A létrehozott fájlunkon belül található „up” metódusban meghatározzuk a táblánk nevét, illetve annak egyes attribútumait úgy, hogy a kívánt típusnak megfelelő metódus hívást elvégezzük, majd paraméterként átadjuk a mezőnek szánt nevet illetve egyéb megkötéseket (például hossz).

4.1. Programkód. Példa egy migration-re a projektből.

```
1 public function up()  
2 {  
3     Schema::create('group_messages', function (  
4         ↳ Blueprint $table) {  
5         $table->id();  
6         $table->string('message');  
7         $table->foreignId('user_id')->constrained  
8         ↳ ()->onDelete('cascade');
```

```

7      $table->foreignId('group_id')->
      ↪ constrained()->onDelete('cascade');
8      $table->timestamps();
9  });
10 }
```

Az egyszerűbb kapcsolatkezelés végett a Laravel egyik alap konvenciója, hogy a külső kulcsokat az összetartozó táblának egyes számú nevével és az „_id” felirattal lássuk el, erre a példa migration-ből:

A „*group_message*” tábla kapcsolatban van a „*groups*” és „*users*” táblával, ezért az idegen kulcsok „*group_id*” és „*user_id*”, így a modelljeinken belüli kapcsolatokat meghatározásánál nincsen szükség további pontosítások megadására.

A projektben több olyan eset is volt, amikor nem követtem ezt a konvenciót, nemes egyszerűséggel az olvashatóság végett. Ez semmi negatívummal nem jár azon kívül, hogy a kapcsolatok meghatározásánál meg kell még adnunk néhány külső és helyi kulcsot.

Az elkészített migration-jeinket szintén a php artisan-nak a *migrate* parancsával tudjuk futtatni, s így létrejönnek a migrációs állományokban meghatározott táblák.

4.1.2. Model

A keretrendszerünk biztosít számunkra egy Eloquent ORM-et[10], ami segít abban, hogy általunk definiált osztályokon tudjunk elvégezni különböző metódus hívásokkal adatbázis-műveletek sokaságát a kontrollereinken belül.

Alapértelmezetten a modellünk a tábla-összeköttetést az osztály nevének megfelelően úgynevezett „snake_case” konvenció szerint fogja összekötni, ami annyit tesz, hogy egy *PéldaOsztály* modellt egy *példa_osztály* nevű táblával kötné össze. A rekordjaink azonosítóját (primary key) is *id*-ként fogja keresni, hacsak nem határozzuk meg explicit módon, hogy eltérünk a megszokottaktól.

A „*\$fillable*” protected mezőben kell felsorolnunk azokat a mezőket, amelyeket kulcsokként kívánunk alkalmazni a *create* és *update* metódusoknak paraméterként átadott asszociatív tömbön belül.

A modellen belül szükségeltetik megadni azokat a kapcsolatokat is, amelyek az adott modellünkhöz tartoznak. Legyen Egy-az-Egyhez, Több-az-Egyhez vagy Több-a-Többhöz kapcsolat, könnyedén definiálhatjuk a megfelelő metódusok hívásának (hasOne - belongsTo, hasMany - belongsTo, belongsToMany) a megfelelő paraméterezésével, ahol az elnevezési konvenciók követésével csak a kapcsolódó modell nevére van szükségünk, egyébként pedig általában egy esetleges kapcsoló tábla, külső- és helyi kulcs megadására lesz szükségünk.

4.1.3. N+1 Query Problem

Az összetettebb kapcsolatok megjelenítését eleinte a nézeteken használt ciklusok magjában oldottam meg úgy, hogy a kívánt kapcsolatot lekérdeztem minden vizsgált elemen, ezzel egy úgynevezett „Lazy Loading”-ot megvalósítva, ami performancia hibákhoz vezethet sok adat esetén. Később a Laravel Debugbar¹ segítségével kiderült, hogy sokkal több lekérdezést hajtok végre az oldal betöltése során, mint amennyire igazából szükség lenne.

A probléma magyarázata:

- Végrehajtok **1** lekérdezést az adatok kinyerésére
- Az **N** db visszaadott rekordnál szintén kiadok 1-1 lekérdezést a ciklusok lefutása során

Ekkor az a probléma, hogy egy összetettebb lekérdezés (például INNER JOIN) helyett $N + 1$ kisebb query-t küldünk az adatbázisunknak, ami a visszakapott adatok mennyiségével arányos módon ronthatja az oldalunk teljesítményét.

A probléma elhárításához a Laravel biztosít egy kapcsolatot, ami az *Eager Loading*. Ennek segítségével a modellek kinyerésekor megszabhatjuk, hogy milyen további, már meghatározott kapcsolatot kívánunk vele összekötni – ilyenkor a háttérben egy egyszerű INNER JOIN-t használ –, így egyetlen lekérdezéssel kinyerhetünk minden szükséges adatot a megjelenítéshez, amit módosítás nélkül meg tudunk jeleníteni az oldalon.

Ha egy lekérdezést akarunk elvégezni a táblánkon, akkor a *Model* osztály *with* statikus függvényét kell meghívunk, ahol paraméterben átadjuk a modellünk kapcsolatát/kapcsolatait. Itt lehetőségünk van arra, hogy felsoroljuk a szükséges relációkat, valamint a „Nested Eager Loading”-gal az is, hogy mélyebb kapcsolatokat is egyetlen paraméterrel lekérdezzünk.

Tehát ha ki akarom nyerni a legutolsó csoportüzenetet a küldő adataival együtt, akkor a 4.2. kódban leírtakat kell tennem:

4.2. Programkód. Eager Loading-ra példa.

```
1 GroupMessage::with('user')->latest()->first();
```

Ha egy lekérdezést akarunk elvégezni egy *meglévő modellen*, akkor a *Model* osztály *load* példányszintű függvényét hívjuk, ahol a paraméterezés ugyanúgy működik, mint a fentebb említett metódusnál.

A 4.3. kódban egy, a paraméterből érkező kurzusnak kérdezem le a hozzárendelt felhasználóit, csoportjait és kvízeit:

4.3. Programkód. Lazy Eager Loading példa.

```
1 $course->load('users', 'groups', 'quizzes');
```

¹ Laravel Debugbar: <https://github.com/barryvdh/laravel-debugbar>

4.1.4. Controller

A *web.php* fájlokban előre definiált útvonalaknak megfelelően végezhetünk anonim metódushívásokat, aminek a törzsében elvégezhetjük az útvonalnak meghatározott feladatokat, viszont ez túlságosan átláthatatlan kódhoz vezet. A jobban struktúrált kód végett hasznos, ha a kérélmeket kezelő logikát Controller-ekbe[12] helyezzük el és ezeknek az egyes metódusait hívjuk meg a route-ok kezelésekor.

Jellemző, hogy a legtöbb Migration-höz egyszerre generálunk Model-t és Controller-t is, így az egyes objektumokon végzendő logika mindig jól meghatározott és elkülönült helyen található úgy, hogy a rendszer elnevezésekkel kapcsolatos konvenciókjait is betartja.

A Controller-ben tipikusan egy (vagy több, a vizsgált objektumunkhoz tartozó) Model osztállyal dolgozunk. Ennek biztosítjuk az alapvető CRUD műveleteket:

- Create - 2 metódus:
 - `create()`: a létrehozáshoz szükséges űrlapot tartalmazó nézetet adja vissza.
 - `store(Request $request)`: a create űrlapjáról érkező adatokat kiolvassuk *\$request* változóból és azok alapján létrehozunk egy objektumot.
- Read - 2 metódus:
 - `index()`: lekérdezzük a kontrollerben kezelt modell minden, az adatbázisban található példányát, majd ezeket átadjuk megjelenítésre az index által visszaadott nézetten.
 - `show(Model $model)`: a paraméterből érkező példányt küldjük át megjelenítésre a nézetre.
- Update - 2 metódus:
 - `edit(Model $model)`: a paraméterből érkező példányt adjuk át a nézetten elhelyezett űrlapra, ami általában megegyezik, vagy legalább is nagyon hasonló a *create* oldalhoz, viszont az átadott változó mezőit betöltjük az *input mezőink* értékeibe.
 - `update(Request $request, Model $model)`: az edit űrlapjáról érkező adatokat kiolvassuk a *\$request* változóból és azok alapján a második paraméter mezőit frissítjük és mentjük.
- Delete - 1 metódus:
 - `destroy(Model $model)`: a paraméterből érkező modellt kitöröljük az adatbázis rekordjai közül.

Egy bevált módszer az is, hogy a Controller helyett egy Resource-t generáltatunk a *php artisan*-nal. Ez biztosítja a fentebb említett 7 metódust, importálja és paraméterezi a használni kívánt Model-lel a metódusainkat és kommentjeinket. A generált Resource-unkat lehetőségünk van behivatkozni a *route.php*-n belül, ahol alapból kapni fog minden egyes route egy egyedi nevet, jellemzően a metódus nevének megfelelően.

A route-okhoz biztosított egy lehetőség, amivel elkerülhetjük a hosszas útvonalak leírását azzal, hogy a hivatkozás meghatározásánál egy *name* függvény paraméterében megadjuk, hogy milyen egyedi névvel szeretnénk rá hivatkozni a későbbiekben.

4.1.5. Refaktorálási kísérletek

A fejlesztés során beleestem egy olyan hibába, amit a hozzám hasonló, kevés szakmai gyakorlattal rendelkező programozók is gyakran elkövetnek: túlságosan hosszú és átláthatatlan kódot írtam úgy, hogy nem használtam ki a keretrendszer által biztosított lehetőségeket.

Az őszi félév elején elkezdtem dolgozni a projekten, viszont inkább az óráimra készültem, ezért nem tudtam olyan sűrűn foglalkozni vele, ahogy kellett volna. Amikor kisebb-nagyobb kihagyások után visszatértem, előfordult, hogy elszörnyülködtem, hogy a 2-3 hete írt kódomat egyszerűen nem tudom értelmezni. Nem értettem, hogy mit csinál, milyen logikát követ, mit hivatott elintézni.

Hosszú, egymásba ágyazott ciklusokkal teli kontrollerek, nem megfelelő elhelyezése a különböző logikáknak, nem teljesen átgondolt elvárások a programmal szemben. Szerencsére rátaláltam egy Laravel Daily nevű csatornára, ahol egy közel 20 részes lejátszási lista volt elérhető Junior Code Review[13] különböző aspektusokra fókuszálva.

Ennek a tanulmányozása előtt már megkezdtem az Eager Loading használatát, amivel már jó pár foreach-től megszabadultam, viszont még mindig elfogadhatatlan volt a kód minősége, így továbbra is szükséges volt a kontrollerek újradolgozása.

Events & Listeners

A videók többségének megtekintésével illetve a témakör után olvasgatva úgy véltem, hogy két opcióm van olyan problémák megoldására, amik bizonyos események lefutásához kapcsolódnak: Observer vagy Event-Listeners.

Megismerkedtem a Laravel *Observerével*, viszont hosszú távon nem tűnt túlságosan átláthatónak használata ahhoz, hogy fenntartható kódot írjak. Tegyük fel, hogy egy hónapokkal későbbi debugolás vagy refaktorálás során valószínűleg meggyűlne a bajunk, ha egy tőlünk valamennyire „elrejtett” logikát kellene észrevennünk, hogy az valami módosítást hajt végre a programunk valamely más részén.

Ezért is döntöttem úgy, hogy inkább az *Event-Listener*[14, 15] megvalósítást használom, aminek a működése felettéből hasonlít az előbb említett Observer-éhez azzal a

különbséggel, hogy itt nekem kell explicit módon egy eseményt kiváltani egy általam meghatározott helyen, viszont ezeknek hívását ugyanúgy lekövetik a Listener osztályaink.

Ezt például a csoportok létrehozásánál használtam, ahol is:

- A `GroupController store` metódusában kiváltok egy „*GroupCreated*” eseményt, aminek paraméterül átadom a létrehozott csoport példányt.
- Az „*EventServiceProvider.php*”-ben ehhez az eseményhez hozzárendeltem egy *listener* osztályt, aminek a neve „*AddFirstUserToGroup*”.
- Az „*AddFirstUserToGroup.php*”-on belül található *handle* metódus paraméterében található esemény kiváltása után kiolvasom az eseményből érkező adatokat (csoport azonosító), majd az alapján elvégzem a létrehozó felhasználó hozzáadását a csoporthoz.

Services és Actions

Olyan hosszadalmas programkódot, ami nem feltétlen kellett az egyes controllerekbe – mert vagy nem az adott controller modelljén hajtott végre valamilyen műveletet, vagy inkább segédszámítások és átalakítások elvégzéséhez kellett –, kiszerveztem vagy egy Service, vagy egy Action osztályba. Ilyen volt például a tesztek összeállítását végző eljárás, ami olyan formában adta vissza a modellt, amely átadható a Livewire komponenseknek, vagy egy teszt kitöltését rögzítő művelet. Így egy több tíz soros logika helyett elintézhető egyetlen sorral a művelet, növelve ezzel is a kontrollerem függvényének átláthatóságát, valamint az egyes komponensek újrahasználhatóságát is megvalósítottam ezzel.

Requests

A form-okból érkező adatok feldolgozása illetve a kontrollerben lévő validációs folyamatok szintén megnövelték a kontrollerjeim méretét – mint utólag kiderült, teljesen feleslegesen.

A dokumentációt olvasva megtudtam, hogy egyszerűbben meghatározhatom az egyes validációs lépéseket és a hozzájuk tartozó hibaüzeneteket is egy saját Request osztályban. Ezért a legtöbb formokat feldolgozó metódusok paraméterében az alap „*Request*” osztály helyett az általam létrehozott egyedi kérésvalidációs állományt használtam.

Itt meghatároztam különböző, az input mezőkhöz tartozó korlátozásokat (required, minimum-, maximum hossz, típus) és azok hibaüzeneteit. Viszont most szükség volt arra is, hogy ezeket megjelenítsem a formjaim oldalán is, hiszen másként nem tudom informálni a felhasználót az általa elkövetett hibákról.

Ezt a legtöbb oldalon úgy végeztem el, hogy a form tetején kiíratom az összes, `$errors->all()` által visszaadott hibaüzenetet, ha az `$errors->any()` igazzal tér vissza, tehát van lekezelendő validációs hiba.

Másik lehetőség a Laraveles `@error` direktíva, amiről a 4.3.5. bekezdésben fogok még írni.

A modelljeink létrehozását és frissítését végző programkódjaink rövidítésére van egy további alternatívánk is egy saját Request létrehozásával. Ilyenkor a paraméter „`$request`” változójával egyetlen sorban elvégezhetjük az utasításokat, mégpedig úgy, hogy a „`$request->validated()`” által visszaadott asszociatív tömböt átadjuk paraméterként a frissítést vagy létrehozást végző függvényünknek.

Például: `$course->update($request->validated());`

4.1.6. View és Blade fájlok

Az MVC által használt nézetekhez a Laravel biztosít úgynevezett Blade sablonokat. Ezek az oldalak a legtöbb esetben a Controller-ek által kerülnek megjelenítésre a felhasználóinknak, viszont nem idegen dolog az sem, amikor a route-ok lekezelésekor adunk vissza egy-egy nézetet.

A nézeteinket a „`resources/views`” könyvtárban találhatjuk meg. Észrevehető, hogy ezen belül is van egy „`layouts`” directory, amin belül kerülnek elhelyezésre azok az alap sablon fájlok, amik tartalmazzák a teljes projektre tartozó hivatkozásokat (például külső CSS és JS fájlok), növelve ezzel a komponenseink újrahasználhatóságát és csökkentve a kódismétlést.

Mivel ezek a blade fájlok is PHP kiterjesztésűek, ezért tartalmazhatnak a megszokott módon PHP-n kívül HTML, CSS és JavaScript kódokat is.

A kontrollerből érkező adat(oka)t egy sajátos szintaxissal, az „echo statement”-tel tudjuk megjeleníteni úgy, hogy „`{{ }}`” zárójelpárok között megadjuk a megjelenítendő változó nevét, ami a következő módon nézne ki kiíratásnál:

4.4. Programkód. Példa az echo statement-re

```
1 <h1>{{ $title }}</h1>
```

A blade fájlok további előnye, hogy lehetővé teszik a fejlesztők számára, hogy egyszerűbb módon alkalmazzanak vezérlési szerkezeteket (például ciklusokat vagy elágazásokat) a beérkező adatok dinamikus megjelenítéséhez. Ezeket az utasításokat előre meghatározott direktívák[17] alkalmazásával hajthatjuk végre. Rendkívül pozitív a használatuk során, hogy a PHP-ban már jól megszokott szintaxisokat kell itt is használnunk, valamint a kód olvashatóságát is növelik, hiszen nem kell külön bajlódni a PHP tag-ek alkalmazásával a különböző hívások elvégzéséhez.

4.5. Programkód. Példa egy if-es blade direktívára

```

1  @if($condition1)
2      ...
3  @else if($condition2)
4      ...
5  @else
6      ...
7  @endif

```

Egy jól bevált szokás, hogy kialakítunk egy „szülő-gyermek” struktúrát[18] az oldalak között úgy, hogy egy alap sablon (egyik layout-beli view) különböző részeibe hivatkozunk be más, a kontrollerjeink által visszaadott nézetnek a különböző, alap sablonnak megfelelő szekcióit.

Ezt úgy tudjuk elvégezni, hogy definiálunk egy „szülő” fájlt, amely tartalmazza a projektünk főbb hivatkozásait és tulajdonságait (például háttérszín, behúzások, külső fájlok). Itt megszabunk bizonyos részeket, amiket a „gyermek” nézetektől várunk el, hogy átadjanak. Ilyen mondjuk az oldalaink címei vagy a megjelenítendő tartalmuk.

Tegyük fel, hogy létezik egy alap sablon nézet, amely a „resources/views/layouts”-ban található *app_example.blade.php* fájl, amelyben elvárjuk a „@yield(...)” direktívával, hogy adott kulccsal beszúrásra kerüljenek a gyermek nézetnek az elemei.

4.6. Programkód. app_example.blade.php példa kódja

```

1  <!DOCTYPE html>
2  <html>
3      <head>
4          <link rel="stylesheet" href="___ .css">
5          <script src="___ .js"></script>
6          <title>@yield('title')</title>
7      </head>
8      <body class="bg-red-500">
9          @yield('content')
10     </body>
11 </html>

```

Ehhez írunk kell egy „gyermek” oldalt, amely szolgáltatni fogja a „szülő” nézet hiányos részeit a „@section(...)” direktívával miután megmondtuk, hogy melyik oldalt is terjeszti ki (@extends(...)).

4.7. Programkód. app_example.blade.php példa kódja

```

1  @extends('layouts.app_example')
2  @section('title', 'Pelda_nezet')
3  @section('content')

```



```
4 ...  
5 @endsection
```

4.2. Autentikáció

4.2.1. Laravel/UI

Mivel egy picit korábbi projektem folytatása ez a webes vizsgáztató rendszer, ennél fogva egy akkoriban általam jól megszokott csomagot használtam a felhasználói azonosításra, ami a *laravel/ui* package. A telepítésének és használatának a részletes leírása az online dokumentációban[19] volt elérhető, tehát nem jelentett nagyobb gondot a használata. Egyetlen problémája az volt, hogy Bootstrap-et használt a Tailwind helyett, viszont az osztályok átírása gyorsan megoldható volt, így nem kellett akkoriban alternatívák után kutatgatni.

A szakdolgozat fejlesztése közben láttam, hogy már az új, Laravel 10.x-es online dokumentáció[20] a *Laravel Breeze* csomagot ajánlja. Ez is a *laravel/ui*-hoz hasonlóan blade fájlokat használ és szinte ugyanolyan eljárású kontrollereket. Mivel a UI és Breeze ugyanolyan azonosítást használ és a végrehajtott eljárásaik is nagyon hasonlóak, maradtam a UI-nál.

Nagy előnye, hogy kapunk jó néhány előre megírt komponenst:

- Controller-ek: Auth könyvtárban 6 controller, pl. Login-, RegisterController.php
- View-ok: szintén 6 nézet az auth directory-ban, pl. login, register, verify.blade.php

Ezek anélkül is teljesen működnek, hogy akármit is módosítani kellene rajtuk, viszont igény szerint bővíthetők is. Sajnos ilyenkor mind a User modellt, mind a „users” migration-t is módosítanunk kell a változtatások helyes lekövetéséhez, valamint a csomaghoz tartozó PHP fájlokban is módosításokat kell végrehajtanunk.

Szóba jöhetett volna még a Fortify, esetleg a Jetstream[21] csomag is, viszont nem igazán tudtam volna kihasználni az általuk biztosított lehetőségeket, illetve mindenképpen maradni szerettem volna a blade fájlok használatánál a Livewire komponensek helyett.

4.2.2. Middleware

A Laravel Middleware-je egy köztes átjáróként (Gate)-ként szolgál a HTTP kérések és az őket feldolgozó route-ok között. Különböző feltételeket szabhatunk az applikációnk felé érkező kéréseknek annak érdekében, hogy azokat egyáltalán feldolgozzuk és eljuttassuk a web.php-nk felé.

Az előre definiáltak közül az egyik ilyen middleware, amit használtam az „*auth*”, amely a felhasználói azonosításért felel. Ez biztosítja, hogy az oldalaimat csak akkor érhessek el a userok, ha sikeresen bejelentkeztek, egyébként visszadobja őket a login page-re. A másik ilyen a „*verified*”, amely megkötelezi az összes usert, hogy az email címükre érkező levélben található hivatkozással erősítsék meg a regisztrációjukat.

Az egyes állapotok megkülönböztetéséhez a Laravel biztosít nekünk például egy *@guest* direktívát, ami igazából egy elágazásként is szolgálhat. Ennek a törzsébe akkor lépünk be, ha még nem történt felhasználói bejelentkezés, az ehhez írható *@else* ágba pedig akkor, ha sikeres volt az azonosítás. Ilyen módszerrel vannak meghatározva a navigációs sávon belül elérhető oldalhivatkozások is.

Az alkalmazásunk bejelentkezett felhasználóinak elérésére kapunk egy helper function-t is, ami az *auth()*. Ezzel lekérdezhetjük a szerver oldalon eltárolt munkamenetből a böngészőnk sütijei segítségével a bejelentkezett User Model-jét (*auth()->user()*), vagy csak egyszerűen annak azonosítóját (*auth()->id()*). Ez azért ajánlatos a Facade-ból beimportálható Auth osztállyal szemben, mert így biztos nem fordul elő, hogy elfelejtünk egy hivatkozást, valamint a blade-jeinken belül is szabadon használható.

További lehetőségünk van arra is, hogy a route-jainkat csoportosítsuk a rájuk vonatkozó middleware-eknek megfelelően, ahol akár felsorolásszerűen is megadhatóak a szabályrendszerek, elkerülve ezzel a további kódismétlést.

4.8. Programkód. web.php-ban található Middleware hívás

```
1 Route :: middleware ([ 'auth', 'verified' ]) ->
    ↳ group(function() {
2     Route::get( '/', [HomeController::class, '
        ↳ index'] ) -> name( 'home' );
3     Route::resource( 'groups', GroupController::
        ↳ class );
4 ...
5 }
```

A 4.8. kódban kötelezzük a felhasználókat a bejelentkezésre, illetve a bejelentkezett userünket arra, hogy megerősítsék az email címüket az oldal útvonalainak teljes eléréséhez.

4.2.3. Szabályzatok (Policies)

A felhasználók által végrehajtható műveleteknek a szabályozására a keretrendszerünk biztosít egy eszközt a fejlesztőknek, amit Policy-nek[23] neveznek.

Ezek szintén generálhatóak a php artisannal, amely minden egyes CRUD művelethez létrehoz 1-1 ellenőrző függvényt, amelynek beérkező *\$user* paraméterével eldönthetjük, hogy a megadott (bejelentkezett) felhasználóra milyen szabályrendszert sze-

retnénk felállítani az adott művelet elvégzéséhez. Az ilyen fájlokat az „*app/Policies*” könyvtárban találhatjuk meg.

Egy policy függvénynek visszatérésének két kimenetele lehet. Vagy egy engedélyező (*Response::allow()*) választ kapunk a policy-tól, vagy elutasítót, amelyben egy megadott üzenetet is meghatározhatunk (*Response::deny('Elutasító üzenet')*).

Az alábbi példával egy try-catch segítségével próbálkozni fogunk egy teszt törlésével. Ha nem megengedett a megadott művelet, visszatereljük a felhasználót egy index oldalra egy SweetAlert-es felugró ablakkal, ahol informáljuk, hogy nem engedélyezett műveletet próbált meg végrehajtani.

Amennyiben viszont egy felhatalmazott személy próbálja elvégezni a törlést, a tesztet eltávolítjuk és visszaküldjük az előző oldalra egy megerősítő üzenettel.

4.9. Programkód. TestPolicy delete metódusa

```
1 public function delete(User $user , Test $test)
2 {
3     return $user->id === $test->creator_id
4         ? Response::allow()
5         : Response::deny('Nem engedélyezett
6             ↳ művelet! Nem Onhoz tartozik
7             ↳ ez a teszt!');
```

4.10. Programkód. A TestPolicy függvényét felhasználó TestController destroy metódusa

```
1 public function destroy(Test $test) {
2     try{
3         $this->authorize('delete', $test);
4         $test->delete();
5         Alert::success('A vizsgasor
6             ↳ sikeresen törölve!');
7         return back();
8     }
9     catch (AuthorizationException
10         ↳ $exception) {
11         Alert::warning($exception->
12             ↳ getMessage());
13         return redirect()->route('courses.
14             ↳ index');
```

A blade fájljainkon belül is található ezeknek használatához egy direktíva. A „`@can`” paraméteréül meg kell adnunk egy Policy-ra való hivatkozást, így a visszatérő értékeknek megfelelően felhasználói felületen megjelenő elemek közül el tudjuk dönteni, hogy a jogosultságnak megfelelően megjelenítünk-e például egy szerkesztést vagy törlést biztosító gombot, vagy sem.

Például: `@can('delete', $course)`, ahol leellenőrzöm, hogy a bejelentkezett felhasználó törölheti-e az adott kurzust.

Minden olyan művelethez használtam a projekten belül, ahol ez indokolt volt.

4.3. Livewire működésének megértése

Mint korábban említettem, a Livewire egy kifejezetten Laravelhez fejlesztett full-stack keretrendszer, amely lehetővé teszi, hogy saját komponensek készítése után beépített AJAX hívások segítségével dinamikusan tudjak oldalakat kezelni anélkül, hogy bonyolultabb JavaScript kódokat írnék.

4.3.1. Oldalak feldolgozása

A Livewire komponensek egy nézetből és egy kontrollerből épülnek fel, amelyek szoros összeköttetésben állnak egymással. A komponensünket be kell hivatkozni egy olyan helyen ahol fel szeretnénk azt használni, méghozzá az alábbi módon: `@livewire('group-chat', ['group'=>$group])`. A példában meghívtuk a csoportos üzenetküldést biztosító komponenst az „`@livewire`” direktívával. Első paraméterben átadásra kerül a komponensünk neve, majd ezt követi a változók felsorolása, ami jelen esetben az, hogy melyik csoport üzeneteit szeretnénk látni.

A komponens hívásakor egyszer lefut a „`mount()`” függvény, amely arra szolgál, hogy a komponenseinket alaphelyzetbe állítsuk, illetve a komponens hívásánál beérkező adatokat kimentsük egy-egy publikus változóba.

Ezután pedig a „`render()`” függvény a szerveroldalon előállítja a megjelenítendő oldalt, majd ezt visszaadjuk a blade sablonfájlunknak. Minden alkalommal, amikor a felhasználó érintkezik az oldallal, egy kérés kerül elküldésre a szerver felé, aminek hatására újra meghívódik a `render` metódus, így biztosítva azt, hogy mindig az aktuális adatokat jelenítjük meg a felületeinken.

A Livewire további függvényeket is biztosít, amelyek az oldal egyes életciklusaihoz tartoznak. Ilyen például az „`updated()`” függvény. Ezt általában akkor szoktuk alkalmazni, amikor valós idejű validációt szeretnénk megvalósítani az oldalon. Amikor a felhasználóink a bemeneti mezők értékeit változtatják, először ez a függvény hívódik meg, ahol elvégezzük a szükséges műveleteket, majd annak megfelelően újra rendereljük az oldalt.

4.3.2. Tulajdonságok (Properties)

[24] A komponensek nézetein megjelenítendő bemeneti mezők értékeit összeköthetjük 2 irányú „data binding”-gal, ami annyit tesz, hogy ha van egy publikus változónk a Livewire kontrollerünkön belül, akkor azt az oldalon lévő input mezővel összeköthetjük a „`wire:model=“variable_name”`” attribútum segítségével. Ilyenkor minden, az input mező értékén végrehajtott módosítás esetén egy-egy AJAX kérés kerül elküldésre a szerver felé, aminek határása a kontrollerben lévő, összekötött változó értéke leköveti a változásokat az aktuális értékre. Fontos megemlíteni, hogy ez a kapcsolat visszafelé is igaz, hiszen ha a PHP kódunkban a változó kezdőértékét módosítjuk, abban az esetben az oldalon is leköveti az input mezőnk értéke a változást.

Annak érdekében, hogy ne történjen meg az, hogy minden egyes billentyű leütésre küldünk egy kérést a szerver felé, – ezzel fölösleges hívásokkal terhelve azt –, lehetőségünk van arra, hogy úgynevezett „*debouncing*”-gal lecsökkentsük a kérelmek számát. Ez úgy valósul meg, hogy csak azután kerül elküldésre az AJAX kérés, amiután egy előre meghatározott (alapból 150 ms) ideig a felhasználó nem módosít az input mezőnk értékén. Ennek használata a fentebb említett „`wire:model`” kiegészítésével történik: „`wire:model.debounce.150ms=“variable_name”`”

Abban az esetben, ha csak akkor szeretnénk adat-összeköttetést végezni, amikor már biztosan befejezte a felhasználó a bemenet módosítását, használjuk a „*lazy updating*”-ot, ami az alábbi: „`wire:model.lazy=“variable”`”. Ebben az esetben csak akkor küldjük el a kérést a szerver felé, amennyiben a bemeneti mezőből kikattintott a user vagy az ENTER billentyű lenyomásra kerül.

Ha a lehető legkevesebb üzenetküldést szeretnénk kivitelezni, akkor használjuk a „`wire:model.defer=“variable_name”`” módszert. Ebben az esetben csak akkor történik meg az értékek szinkronizálása, amikor egyéb okok miatt egy AJAX hívás kerül elküldésre a Livewire felé. Ezzel el tudjuk végezni, hogy eltároljuk az összes módosítást, majd a legvégén egyetlen nagyobb AJAX hívást küldve a szerver felé validálunk minden adatot és azoknak megfelelően intézkedünk a lefutás további részében.

4.3.3. Műveletek és Események (Actions & events)

[25, 26] Az input mezők összeköttetésén kívül van egyéb lehetőségünk is Livewire-es hívást intézni. Ezek közül egyik ilyen az „*action*”, ami segítségével könnyedén ki tudunk váltani kommunikációt a szerver és kliens között.

Három alap műveletet biztosít a keretrendszer:

- Click: ennek segítségével egy gomb lenyomásához hozzárendelhetünk függvény-hívásokat akár paraméterekkel is.

Például: `<button wire:click=“foo(param1)”>..</button>`

- Submit: form-jainknak adható livewire attribútum, további módosítónak adható például a „prevent”, aminek hatására nem kerül rögzítésre a megadott űrlap, hanem helyette egy meghatározott függvény kerül végrehajtásra.

Például: `<form wire:submit.prevent="foo">`

- Keydown: egy megadott billentyű lenyomására előre meghatározott függvény kerül meghívásra.

Például: `<input type="text" wire:keydown.enter="foo">`

Előfordult, hogy bizonyos eseményekre el kellett indítanom JavaScript-es híváson keresztül egy Livewire komponensen belüli függvényt. Erre kifejezetten akkor volt szükség, amikor a tanulók teszt kitöltésénél figyelni kellett a hátralévő időre. Amennyiben a számláló eléri a nulla másodpercet, le kell zárni a tesztet, amit alapvetően a Livewire controllerjében található `endTest()` eljárás végez el.

Ahhoz, hogy ezt JavaScript-en belül meg tudjam hívni (ami az óra megjelenítését végezte el), a „`@this.emit('timeRanOut');`”-t kellett használnom, aminek segítségével egy „timeRanOut” eseményt váltottam ki, amire ráállítottam egy *listener*-t a Livewire Controller-en belül, ami végül meghívta az `endTest()` függvényt.

4.3.4. Sortable csomag

A tesztek során készíteni akartam egy olyan kérdés típust, amely lehetővé teszi a kitöltő számára, hogy sorrendbe helyezze az egyes elemeket. Ehhez szükségem volt egy olyan könyvtárra vagy package-re, amely biztosítja a „Drag&Drop” funkcionalitást az oldalnak. Szerencsére a Livewire dokumentációjában[28] rátaláltam egy már létező, githubra feltöltött package-re[29], aminek a neve *Sortable*.

Ahhoz, hogy használni tudjuk ezt a megoldást, a CDN-es hivatkozás vagy az NPM-es telepítés után szükségünk van egy olyan tag-re, amely magába foglalja azokat az elemeket, amelyeket mozgathatóvá szeretnénk tenni (legyen ez egy div, vagy egy ul/ol). Ennek kell egy „`wire:sortable="foo"`” attribútumot kell adni, amelynek értékének a sorrendváltozást feldolgozó függvénynek a nevével kell lennie (jelen esetben „foo”).

Az itt megadott feldolgozó függvénynek minden esetben tartalmaznia kell egy formális paramétert, ami a hívás során értékül fog kapni egy tömböt. Ez annyi elemet tartalmaz, ahány elemet eltároltunk ebben a konténer tag-ben. Minden egyes elem értéke egy-egy további tömb lesz, aminek két eleme van:

- Az első az „order”, amely megadja, hogy az adott elem hanyadik volt az átrendezett sorrendben. Itt a számozás 1-estől indul, egészen az elemszámig.
- A második a „value”, amiből kiolvashatóak azok az értékek, amelyeket a mozgatható elemeinkhez rendeltünk a „`wire:sortable-item="value"`” attribútummal.

Ennek a paraméternek a feldolgozásával megoldható az elemek sorrendjének a megfelelő módosítása.

A sorrendbe helyezendő elemeinknek egy „*wire:sortable-item*=*\"value\"*” attribútumot kell adnunk. Ezzel fogunk tudni a sorrendbe helyezendő elemünkhöz valamilyen értéket rendelni (jellemzően ID, vagy index). Az itt szereplő érték fontos, mivel ez fog átadásra kerülni a sorrendet kezelő függvénynek.

További, opcionálisan megadható attribútum a „*wire:sortable.handle*”: az elemünkhöz rendelhetünk egy szimbólumot, vagy speciális karaktert. Ha megjelöljük ezzel az attribútummal, akkor a teljes elem helyett csak erre a megjelölt tag-re kattintva tudjuk mozgathatóvá tenni.

4.11. Programkód. Minta kód projektből.

```
1 <div wire:sortable="updateOptionOrder">
2   @foreach ($question['options'] as
3     ↳ $optionIndex => $option)
4     <div class="..."
5       wire:sortable.item="{{ $questionIndex
6         ↳ . '_' . $optionIndex }}"
7       wire:sortable.handle>
8       {{ $option['text'] }}
9     </div>
10  @endforeach
11 </div>
```

A projektben a 4.11.-es kódban láthatjuk a kicsit leegyszerűsített változatát a projektben található implmenetációnak.

Láthatjuk, hogy minden egyes átrendezés során meghívódik egy „*updateOptionOrder*”, aminek paraméterében elérhető lesz az összes válaszlehetőség, amely az adott kérdéshez tartozik. Minden opcióhoz megvan, hogy a kérdése hanyadik a sorban, illetve maga az opció milyen index-szel rendelkezik.

Ezt úgy dolgoztam fel, hogy létrehoztam egy üres array-t, ami majd tartalmazni fogja az új sorrendben lévő válaszlehetőségeket. A paraméterben kapott elemeken végig iteráltam, majd a kapott értékekből kinyert indexeknek megfelelően mindig a „*\$questionIndex*”-edik kérdés „*\$optionIndex*”-edik opcióját hozzáadtam az új sorrend tömbjéhez. A ciklus végeztével a kérdés opcióinak listáját felülírtam az újonnan létrehozott, már átrendeztet opciók listájára.

4.3.5. Valós idejű validáció

A Laravel-hez hasonló módon itt is lehetőségünk van a beérkező adatok ellenőrzésére különböző szabályoknak a felsorolásával, illetve ezeknek a megjelenítendő hibaüzenek-

teiknek a megadásával.

A Livewire támogatja az egyszerű blade fájlokban használatos `@error` direktívák használatát is, amiben az eltárolt hibaüzenetek értékét a `$message` változóból tudjuk kiolvasni. A különbség csupán annyi, hogy az `@error` hivatkozásánál paraméterben nem az input mező `name` attribútumának kell megadni a helyes hivatkozását, hanem a vele összekapcsolt, kontrollerben található változó nevével. Például:

4.12. Programkód. A teszt létrehozásánál alkalmazott hibakezelésből

```
1 @error('testTitle') {{ $message }} @enderror
2 <input type="text" wire:model="testTitle">
```

Valós idejű hibakezelést a már korábban említett `updated()` „lifecycle hook” segítségével tudjuk elvégezni. A függvényen belül választhatunk, hogy vagy az oldal teljes tartalmát validáljuk, vagy csak a legutóbb módosított input mezőt. Ha úgy járunk el, ahogy a dokumentáció is ajánlja és az utóbbit választjuk, a függvénynek paraméterül kell adnunk egy „`$propertyName`” változót, amelyben átadásra kerül a validálandó változó neve. A függvény törzsében egyetlen sort kell elhelyeznünk, ami elvégzi annak az egyetlen mezőnek a validálását, amelynek nevét átadjuk paraméterként az alábbi módon: „`$this->validateOnly($propertyName);`”. Ha probléma adódik, egy `ValidationException`-nel fog visszatérni úgy, hogy minden hibát egy-egy `$error` változóba helyez el, amit majd az oldalunkon belül feldolgozhatunk a már korábban tárgyalt módszerekkel. Amennyiben a formunkat elküldjük, automatikusan lefut majd a „`$this->validate();`”, amely hatására minden bemeneti mezőre megtörténik az ellenőrzés.

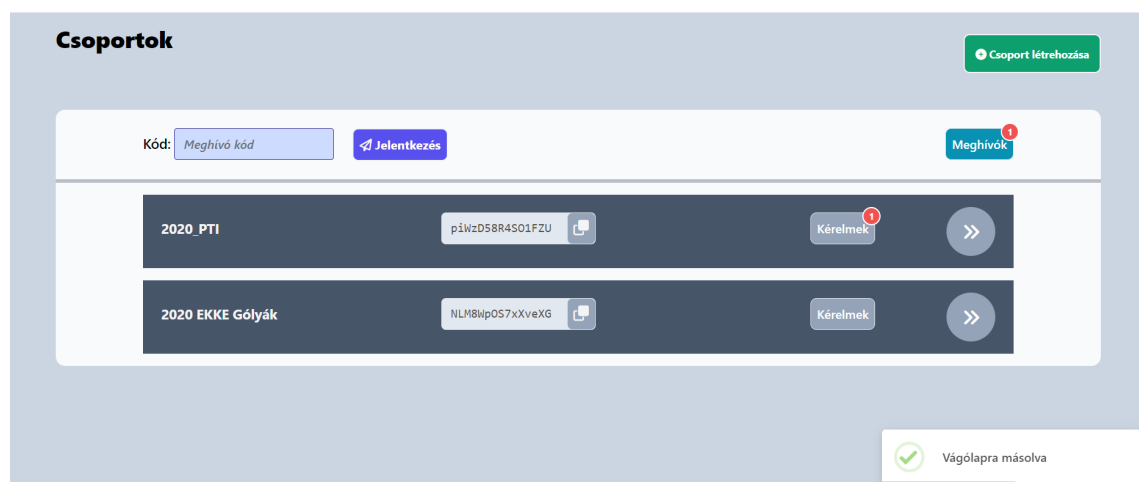
5. fejezet

A csoportok bemutatása

Egy kisebb közösség létrehozásához készítettem ezt a funkciót, mint például egy tanulócsoporthoz vagy osztályhoz. A csoportok segítségével több tanulót vagy felhasználót foglalhatunk egy egységbe, megkönnyítve ezzel a diákoknak a kurzusokhoz való hozzárendelését.

A valódi közösség érzet kialakításához, megvalósítottam egy chat szobát is, ahol a csoport tagjai szabadon tudnak egymással kommunikálni az alkalmazáson belül.

5.1. Felhasználók hozzárendelése a csoportjainkhoz



5.1. ábra. A csoportjainkat listázó oldalról képrenyőkép

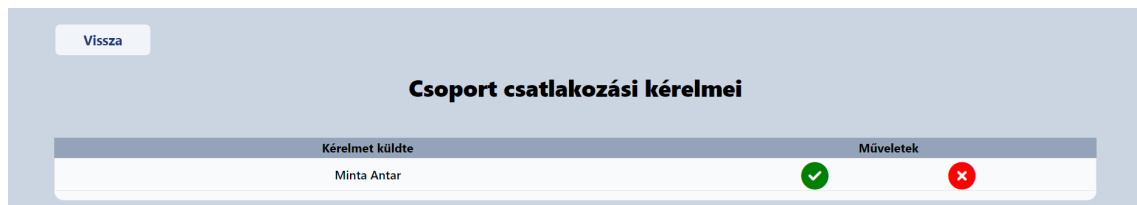
Ahhoz, hogy egy felhasználó tagja lehessen egy csoportnak, két lehetőség áll rendelkezésre:

- Minden csoport létrehozásakor generálódik egy egyedi, 15 karakter hosszúságú kód, amely állhat számokból illetve kis- és nagykapitális betűkből. Ennek a kódnak a kimásolásához minden olyan csoporthoz biztosítottam egy gombot, amelynek tagjai vagyunk, ezáltal ismerősöket is meghívhatunk a közösségünkbe. A

jelentkezés elküldéséhez csupán annyi a teendőnk, hogy a csoportok főoldalán található beviteli mezőbe beillesztjük a kimásolt kódot, majd elküldjük a csatlakozási kérelmünket.

Minden csoport adminisztrátorának (aki létrehozta azt) látható egy-egy gomb „Kérelmek” felirattal. Itt láthatjuk a jobb felső sarkában, hogy jelenleg hány jelentkezési kérelmünk váratkozik elbírálásra a csoporthoz.

Erre kattintva kilistázhathatjuk az összes jelentkezni kívánt felhasználót, ahol eldönthetjük, hogy elfogadjuk vagy elutasítjuk a kérelmet



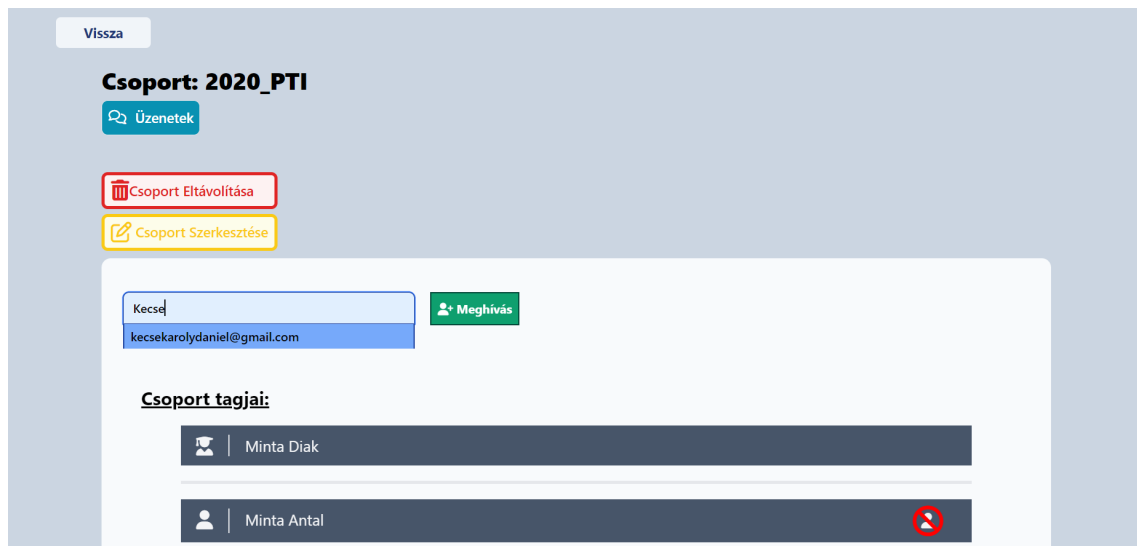
5.2. ábra. Kép egy csoport csatlakozási kérelmeit listázó oldalról

- Amennyiben mi hoztuk létre a csoportot, az online dokumentációban[30] található módon létrehozott Livewire kereső komponens segítségével kereshetünk e-mail cím szerint a létrehozott felhasználók között. Abban az esetben, ha a megadott e-mail címre kapunk találatot, kislistázzuk, majd kattintással kijelölhetjük. Ilyenkor a kereső mező alatt megjelenik a kijelölt elemnek egy kis kártyája. Akár több felhasználót is kijelölhetünk egyszerre. A kérelmek kiküldéséhez a „Meghívás” gombra kell kattintanunk, amely során átküldjük a szerver felé, ahol rögzítünk a felhasználókhöz 1-1 meghívót a csoportunkba. Ebben az esetben szintén a főoldal jobb felső sarkában található „Meghívók” gombra kattintva találhatjuk meg ezeket a meghívási kérelmeket. Ez majdnem teljesen olyan, mint a csatlakozási kérelmek oldala, viszont itt azt is látjuk, hogy mi a csoport neve, amelybe meginvitáltak minket.

5.2. Felhasználók eltávolítása a csoportjainkból

Amennyiben a csoport adminisztrátora távolít el minket valamilyen oknál fogva a csoportból, kapunk egy értesítést az e-mail címünkre, amelyben láthatják, hogy mely csoportból lettek kirúgva.

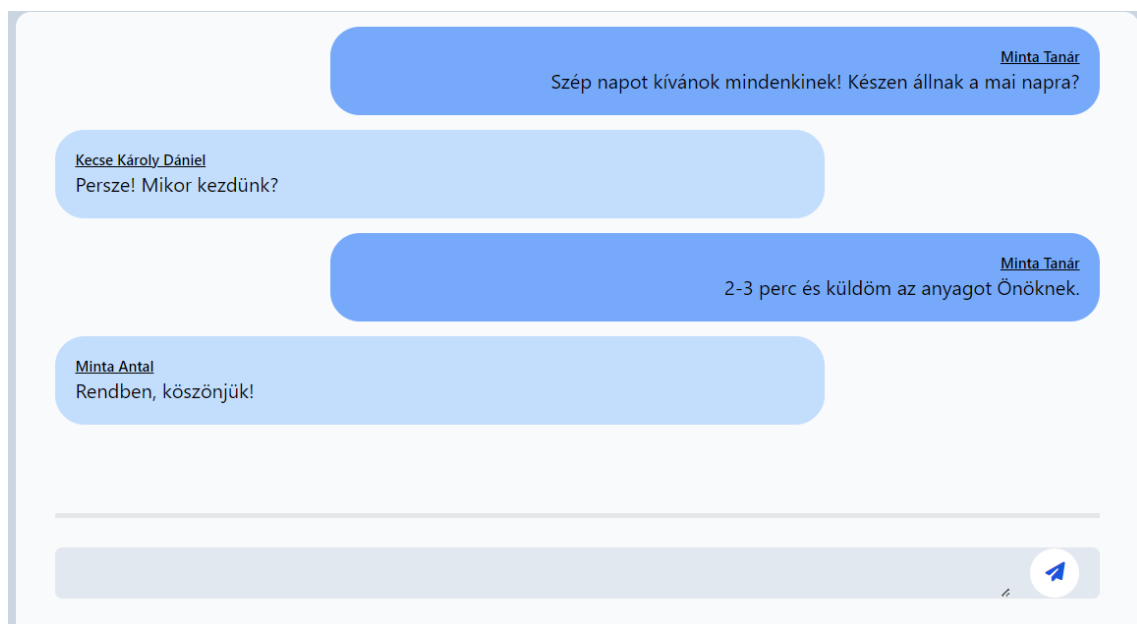
Másik lehetőség, hogy a felhasználó saját maga lép ki a csoportból, ilyenkor nem kap senki üzenetet.



5.3. ábra. Egy csoport kiválasztásánál kapott oldal

5.3. Chat: Pusher és Echo

A chat felületét minden olyan felhasználó elérheti – aki tagja a csoportunknak – az „Üzenetek” gombra kattintva. Ekkor megjelenik az alábbi felület:



5.4. ábra. Példa egy csoportos csevegésről

Mivel egy alap elvárás egy ilyen funkciónál, hogy a felhasználóknak ne kelljen mindig frissíteni az oldalt, hogy megnézzék, küldött-e valaki egy új üzenetet, ezért valami WebSocket-hez hasonló megoldást kellett keresnem. Mivel nem voltam járatos a témában, ezért további alternatívák felé kutattam és megtaláltam a Laravel által ajánlott Broadcastingot. Két JavaScript-en alapuló package segítségével biztosítottam, hogy a szerver oldalon kiváltott eseményt figyelni tudjam a kliens oldalon is egy meghatározott

csatornán, ahol le tudom reagálni az egyes változásokat.

Ennek a működése a következő képpen történik nagy vonalakban: A felhasználók a felületen feliratkoznak egy adott csatornára, majd ebbe a csatornába küldjük a backend egyes eseményeit, amit a kliens oldalon feldolgozunk.

A szerveroldali üzenet broadcastolásához használtam a *Pusher-Channels*[31] csomagot. Ennek használatához egy-egy üzenet létrehozása után elindítok egy broadcast-ot, aminek átadok egy eseményt, amelyet felparamétereztem a csoport azonosítójával. Ezen az eventen belül pedig egy „*broadcastOn()*” metódusban létrehozok egy új privát csatornát, amelynek a neve „groupMessagesX”, ahol X a csoport ID-je, értesítve ezzel a csatorna többi tagját, hogy egy új üzenet került elküldésre és ezt le kellene kérdezniük.

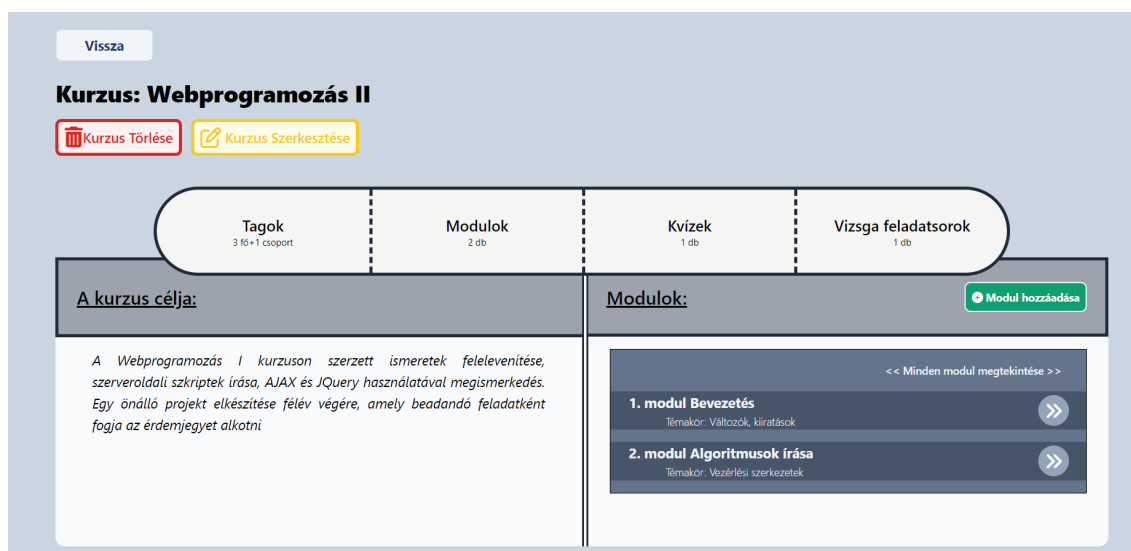
Erre a csatornára a kliens oldalon a *Laravel-Echo*[32] package segítségével feliratkozunk úgy, hogy megadjuk, hogy milyen csatornára szeretnénk „ráállni” és azon belül is milyen eseményt figyelünk. Amennyiben kiváltásra került az event, indítunk egy GET-es kérést a szerver felé, ahol lekérdezzük a csoportunkba legutoljára felvitt üzenet szövegét, amit megjelenítünk a Livewire komponensünk segítségével.

6. fejezet

A kurzusok bemutatása

A projekt kurzusai szolgálnak az egyes tanórák gyűjtő egységének. Lehetősége van a kurzus tulajdonosának felhasználókat és csoportokat hozzárendelni a kurzusokhoz, valamint tananyagoknak (moduloknak) a létrehozásához. Készíthetnek kvízeket, amelyeket vagy magához a kurzushoz, vagy egy-egy modulhoz kapcsolnak. A jegyszerzés biztosításához pedig készíthetnek vizsga feladatsorokat, amiket egy megadott indőintervallumban kitölthetnek a tanulóink.

Minden olyan kurzushoz hozzáférhetünk, amihez vagy személyesen, vagy csoporton keresztül el kellene tudnunk érni. A tagok hozzárendelésénél ugyanaz a Livewire komponens került felhasználásra, amely végzi a felhasználók csoportokhoz rendelését. A képernyőképen látható gombok megjelenítése és működése itt is a policy-k segítségével valósul meg.

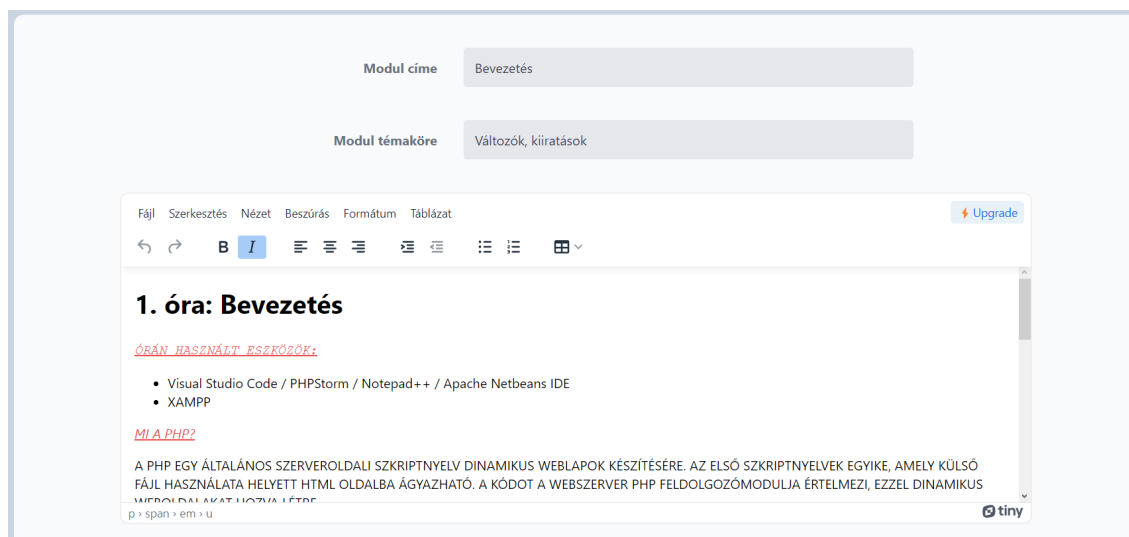


6.1. ábra. Egy kurzus megtekintésénél ez az oldal fogad minket

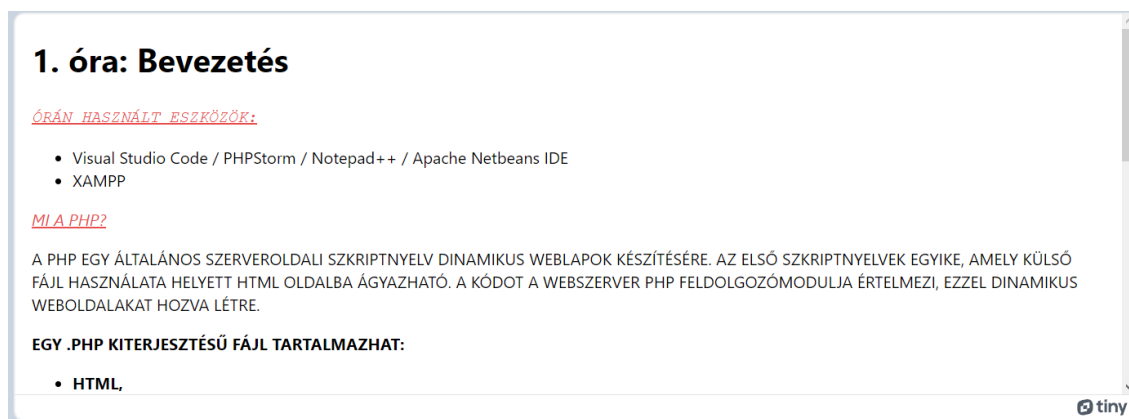
6.1. Modulok

Ezek az egységek tartalmazzák a kurzusok tananyagait. Itt egy TinyMCE felület található, amely biztosítja az összes olyan alapvető eszközt, amely a tananyag megírásához és összeállításához szükséges lehet.

A szerkesztő felület abban az esetben jelenik meg, ha a kurzus tulajdonosa (tehát a tanár) létrehozásra vagy szerkesztésre nyitja meg a modult. Ilyenkor módosíthatja a témakört, az óra megnevezését illetve a tananyag tartalmának szövegét.



6.2. ábra. Egy adott modul tananyagának szerkesztése



6.3. ábra. Egy adott modul tananyagának megtekintése

A tananyag egésze egy „textarea”-ban van eltárolva, ennek az értékét fogjuk eltárolni egy „blob” típusú mezőben az adatbázisunkban. Ez tartalmaz minden, a tananyagot képező HTML taget, annak osztályát és inline CSS beállítását. Azért, hogy ezek megfelelően jelenjenek meg az oldalon a megtekintés során, ugyanolyan szerkesztőfelületbe töltjük bele a „material” mező értékét, mintha szerkesztenénk, viszont readonly módban nyitjuk meg és a szerkesztői fejléct elrejtjük.

7. fejezet

A tesztek felépítése és kitöltése

A projektben található tesztek segítségével tudjuk biztosítani, hogy a tanulók visszacsatolást szerezzenek – kvízek segítségével – a jelenlegi tudásukról, illetve, hogy számot adjanak arról – vizsga feladatsorok kitöltésével –, hogy mennyire tudták elsajátítani az egyes kurzusokhoz tartozó tananyagot.

A tesztek létrehozásához és szerkesztéséhez 1-1 Livewire komponens került elkészítésre. Ezeknek a használatakor elkülönül az oldalak működése oly módon, hogy ha tudjuk, hogy kvízt készítünk, letiltunk (Kitöltések száma, kitöltésre szánt idő megadása) illetve hozzáadunk (Modul kiválasztása) néhány funkciót, míg vizsga feladatsor összeállításánál ennek a fordítottját hajtjuk végre.

Ezt a létrehozásnál úgy tudjuk megtenni, hogy a komponens hívásakor egy paraméterben átadom a teszt típusát, szerkesztésnél pedig a komponensnek átadott teszt típusa alapján tudjuk, hogy mivel is kell majd dolgoznunk a feldolgozás során.

7.1. Service-ek

Annak érdekében, hogy a teszteket a Livewire komponenseimen belül megfelelően tudjam kezelni, szükséges volt egy hosszadalmasabb átalakítása a modelljeimnek. Emiatt szükségessé vált, hogy kiszervezzem a meglévő metódusaim bizonyos részleteit egy külön osztályba/osztályokba, hiszen túlságosan hosszúvá váltak a kontrollerjeim ahhoz, hogy átlátható és tiszta legyen a kód.

Mivel minden ilyen blokk a „Test” modellen végzett valamilyen műveletet, ezért úgy döntöttem, hogy az „Action”, „Job” és „Service” osztályok közül inkább az utóbbit választom. Amíg az előző két osztály lényegében 1-1 metódus elvégzésére használatosak, addig a Service-be elhelyezhetek több, logikailag összetartozó metódust.

Ebbe a „TestService” osztályba tartoznak azok az eljárások és függvények, amelyek elvégzik a tömb adatszerkezetben eltárolt tesztjeim mentését illetve módosítását, valamint a megkapott „Test Model” (és az ahhoz tartozó kapcsolatok) átalakítását szintén

tömb adatszerkezeté, így megkönnyítve a későbbi data bindingot illetve a megjelenítést.

7.2. Feladatlapok megvalósítása

Feladatlap címe:
Minta feladatlap

Lehetséges kitöltések száma:
2

Teszt kitöltési ideje
5 perc

Eredmény elérhető: ☒ Igen

Igaz/Hamis
Igaz/Hamis
1 megoldásos választás
Több megoldásos választás
Sorrend

...ap tegnapi?

☒ Igaz ☐ Hamis

Új kérdés Mentés

7.1. ábra. Egy vizsga feladatsor létrehozása

Miután a teszt fejlécét kitöltöttük, fel kell töltenünk a tesztünket tartalommal, aminek az elvégzéséhez található a bal alsó sarokban egy „Új kérdés” feliratú gomb.

Egy kérdés hozzáadása után meg kell adnunk, hogy milyen legyen a típusa. Itt 4 lehetőség közül választhatunk – *Igaz/Hamis*, *Egy-*, *Több lehetséges megoldás* kiválasztása, *Sorrend*) –, ahol egy opció kijelölése után „emitelünk” egy eseményt a Livewire backend felé, ahol ezt figyeljük egy „listener” segítségével, majd az alábbiakat tesszük:

- Ha a kiválasztott típus *Igaz/Hamis*, az adott kérdéshez kapunk két választási lehetőséget. Ezek az „Igaz” és „Hamis” feliratot kapják. Itt rá kell kattintanunk arra, amelyet a kérdés megoldásának szánunk.
- Minden egyéb esetben nekünk kell újabb válaszlehetőséget rendelnünk a kérdéshez, ahol megadjuk a legalább 5 karakter hosszúságú feliratát, majd itt is kijelöljük a helyes megoldásokat, vagy megfelelő sorrendbe állítjuk őket abban az esetben, ha sorrend típusú feladatról beszélünk.

A dinamikus hozzáadáshoz, törléshez, a szövegek eltárolásához és a helyes megoldások kijelöléséhez a már korábban emlegetett *data-binding*-ot és *action*-t használjuk úgy, hogy a 3.1.3. bekezdésben említett struktúrának megfeleljünk.

Minden egyes kérdéshez tartozik egy szöveg, típus, válaszlehetőségek tömbje illetve az *Igaz/Hamis* és az *Egy megoldásos* feladattípusok megoldásainak eltárolásához egy „right_option_index” kulcs-érték pár. Ilyenkor a kérdések megoldásainak eltárolása

miatt a „right_option_index”-en belül eltároljuk annak a válaszlehetőségnek az indexét, amelyet a felhasználó helyes megoldásként jelölt meg.

Minden válaszlehetőség egy szövegből, pontszámból és egy „solution” kulcs-érték párból áll. A pontszám abban az esetben lesz 1, amennyiben az helyes megoldásként van megjelölve a feladatlap elkészítése során. A „solution”-be kerül eltárolásra, hogy az adott opció helyes megoldásként van-e megjelölve, vagy sem. Ezt kulcsot a *Sorrend* és a *Több megoldásos* feladatoknál használjuk fel úgy, hogy a checkbox-okat és diveket az adott opció „solution” kulcsához rendeljük. Ez abban az esetben kap értéket, ha a felhasználó bepipálja azt a *Több megoldásos* feladatnál az adott választ, a *Sorrend* kérdéseknél pedig automatikusan a sorban elfoglalt indexét kapja.

Amikor egy-egy gombra kattintunk, legyen az törlést vagy hozzáadást végrehajtó, mindig egy „wire:click” action-t hívunk meg, aminek határása lefut egy-egy eljárás. Ezek a függvények biztosítják az alábbi műveleteket:

- a kérdés hozzáadását: a „*\$questions*” array-hez ad hozzá egy elemet,
- a kérdés eltávolítását: a „*\$questions*” array-ből eltávolítja a paraméterből érkező „*\$questionIndex*”-edik elemet,
- opció hozzáadását: a „*\$questions*” tömbnek a paraméterből érkező „*\$questionIndex*”-edik elemének „*\$options*” tömbjéhez ad egy elemet,
- opció eltávolítását: a „*\$questions*” tömbnek a paraméterből érkező „*\$questionIndex*”-edik elemének „*\$options*” tömbjének „*\$optionIndex*”-edik elemét távolítja el,
- teszt mentését: létrehozok a fejlécben megadott adatokkal egy „*Test*” modellt, majd azt átadom a *TestService store* metódusának az eltárol „*\$questions*” tömbbel együtt.

7.3. Teszt kitöltésének folyamata

Amennyiben egy adott kurzus tesztjét szeretnénk kitölteni, először is kötelezően tag-nak kell lennünk, egyébként a Policy miatt nem fogunk tudni hozzáférni. Ezután a *TestService* „getTestToWrite” függvényével átalakítjuk tömb adatstruktúrává a Test modellünket minden relációjával együtt, majd ezt átadjuk a kitöltést elvégző Livewire komponensünknek.

7.3.1. Kvízek

A teszt létrehozásánál tárgyalt módon történik itt is az adatoknak az összeköttetése, viszont itt eltároljuk a helyes válaszok indexét is, hiszen ebben az esetben nem

probléma, ha valahogy kiderülne, hogy mik a helyes megoldások. A teszt kitöltése után megtekinthetik a megadott válaszaik eredményét, illetve újra megpróbálkozhatnak a kitöltéssel, amennyiben nem elégedettek a teljesítményükkel. Ebben az esetben az összes kérdést, illetve az *Igaz/Hamis* típusú feladatok kivételével az összes válaszlehetőséget is összekeverjük, majd megjelenítjük ismételten a kvízt. Itt fontos, hogy a próbálkozások eredményei nem mentődnek, illetve akárhányszor kitölthető tetszőleges időn keresztül, korlátozások nélkül.

7.3.2. Vizsgák

Vizsgák esetén azon a feltételen felül, hogy kurzus tagnak kell lennünk, van egy másik korlátozás is. A kurzus tulajdonosa megadhatja, hogy a vizsga feladatsorok a diákok számára milyen időintervallumban lehessenek elérhetőek. Ennek megfelelően egy kissé összetettebb policy-n is át kell esniük a felhasználóknak, amely ellenőrzi, hogy jogosult-e a kurzushoz, elérhető-e a jelenlegi dátumon a teszt kitöltése, és van-e még további kitöltésre lehetőségünk. Mivel a vizsga feladatsorok csak korlátozott számmal tölthetőek ki, ezért előfordulhat, hogy kifutunk a próbálkozásokból.

Amennyiben biztosak vagyunk abban, hogy elindítjuk a teszt kitöltését, rögzítünk egy „testAttempt” rekordot, amellyel jelezzük, hogy van egy aktív próbálkozása az adott diáknak egy adott teszthez. Ilyenkor ezzel egy időben egy *Queued Job*-ot is indítunk, amely biztosítja azt, hogy egy próbálkozást akkor is lezárunk, ha a diák valamilyen oknál fogva kilépne a rendszerből.

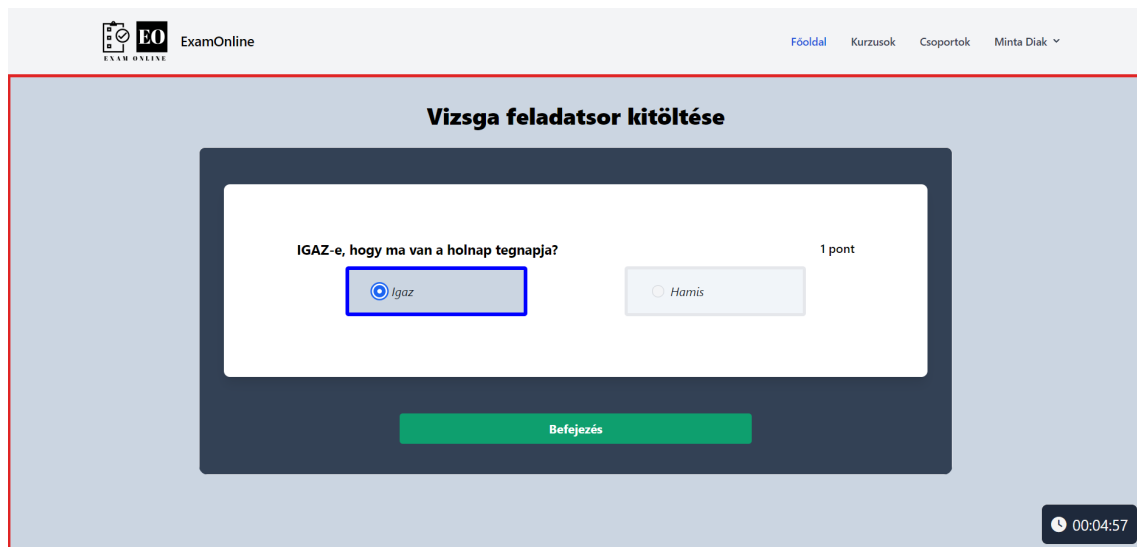
Azért, hogy ne keveredjenek össze a szálak, egy minimális időeltolást alkalmazunk. Ha van egy teszt, aminek a kitöltésére 5 percünk van, akkor ezt a Job-ot 5 perc és 1 másodperccel időzítem, hiszen amennyiben a diák felhasználja mind az 5 percet, amely a rendelkezésére áll, akkor a rendszer automatikusan lezárja a kitöltést, majd ezután a beérkező Job észreveszi, hogy a próbálkozást lezárunk és be is fejeződik. Amennyiben mindketten azonosan 5 perc után hajtódának végre, előfordulhatna az, hogy érvénytelen próbálkozásnak veszi a Job miatt és nem kerülne kiértékelésre a dolgozat.

Annak érdekében, hogy eltároljuk a teszt kitöltésének előrehaladását, a vizsga megkezdésekor az összeállított teszt tömbjét eltárolom a session-ben a kezdés időpontjával együtt (ebből is kiszámolhatom a hátralévő időt) a próbálkozás azonosítójának segítségével.

Amikor megpróbálkozunk a vizsga kitöltésével, megnézzük, hogy van-e az adott felhasználónak egy már megnyitott, de be nem fejezett próbálkozása az adott teszthez. Ha van, kiolvassuk az adott ID-vel eltárolt session változóból, viszont amennyiben ez nem található, sajnos elveszett az elmentett adat, ezért egy újabb Service hívással összeállítunk egy tesztet és megkezdődik előlről a kitöltés a hátralévő idővel.

A teszt kitöltése alatt 5 másodpercenként felülíródik az eddig eltárolt előrehaladás a

munkameneten belül. Ezt a „*wire:poll.5s=SaveDataToSession*”-nel tudjuk elvégezni, amely 5 másodpercenként meghívja a megadott eljárásom.



The screenshot displays the ExamOnline web application. At the top, the logo and name 'ExamOnline' are visible, along with navigation links for 'Fórdal', 'Kurzusok', 'Csoportok', and 'Minta Diák'. The main content area is titled 'Vizsga feladatsor kitöltése'. It features a question: 'IGAZ-e, hogy ma van a holnap tegnappja?' with a value of '1 pont'. There are two radio button options: 'Igaz' (selected) and 'Hamis'. A green 'Befejezés' button is at the bottom of the question box. A timer in the bottom right corner shows '00:04:57'.

7.2. ábra. Egy vizsga feladatsor kitöltésére példa

A teszt kitöltése közben látható egy számláló ahol láthatjuk még rendelkezésre álló időnket, illetve egy vörös keretet a teszt körül. Amennyiben a kitöltő kurzora 1 másodpercnél tovább marad kint a megengedett tartományon kívül, a tesztre visszahúzza a kurzorát befejeződik a kitöltés és kiértékelődik a tesztje az eddigi megadott válaszai alapján.

Mivel az online oktatás során megtanultam, hogy a diákok bármire képesek annak érdekében, hogy megkönnyítsék a digitális számonkérést, tudom, hogy ez sem teljes megoldás, viszont megpróbálkoztam az egyértelmű csalásokat kiszűrni.

8. fejezet

Az alkalmazás tesztelése

A projekt tesztelésére kiemelt hangsúlyt fektettem az alkalmazás fejlesztése során, hiszen mindenképpen egy teljesen jól funkcionáló, hibáktól mentes rendszert akartam kiadni a kezeim közül. Az egyes funckiók elkészítése után mindig elvégeztem egy, vagy több ahhoz tartozó manuális tesztet. Csak abban az esetben léptem át egy következő fejlesztési folyamatra, amennyiben az általam elvárt eredményeket sikerült produkálnia a rendszernek.

Az alkalmazás növekedésével egyre több időbe telt, hogy az oldal egészét leteszteljem, ezért szükség volt egy automatizálható tesztre. Mivel sok jót hallottam a Cypress-ről, ezért úgy döntöttem, hogy ezt fogom használni a továbbiak során. A legtöbb funckiót így sikerült letesztelnem, viszont a feladatsorok kitöltését mindeképpen manuálisan kell még mindig tesztelni, hiszen minden egyes feladatlap más sorrendben generálja a kérdéseket és az azokhoz tartozó válaszlehetőségeket.

A Cypress egy nyílt forráskódú, JavaScript nyelven íródott keretrendszer, amely segítségével automatizált tesztek végrehajtását el a fontendünkön végezhető műveletek. Az egyik legnagyobb előnye, hogy könnyen és gyorsan tesztelhetjük a rendszerünket úgy, hogy a tesztelő integrálása is egyszerűen megoldható.

A Laracasts Cypress GitHub[33] oldalán található telepítési útmutatóval néhány perc alatt elérhetjük, hogy használhassuk a rendszerünkön belül. Az általuk készített Youtube videó[34] segítségével készültek el a projekt egyes tesztjei.

Szinte minden teszt az alábbi utasításokat tartalmazza:

- „*cy.login({username: '...'})*”: bejelentkezünk a rendszerbe egy olyan felhasználóval, akinek a felhasználóneve a paraméterben megadott értéknek megfelel. Amennyiben létezik már ilyen értékű mezővel rekord, azzal belépünk, egyéb esetben a paraméterben meghatározott értékekkel létrehozunk egy rekordot, majd azzal bejelentkezünk. Az utóbbi esetben a meg nem határozott, kötelezően kitöltendő mezőknek véletlenszerűen generált értékeket ad a keretrendszer.
- „*cy.visit('...')*”: a paraméterben meghatározott útvonalra navigál. Az itt található

elemekre írhatunk DOM-os vizsgálatokat, amikhez a következő 2 pontban leírtakat használtam.

- „*cy.get(...)*”: paraméterben megadunk DOM kijelölőket, mint például HTML tag típusa, osztály vagy azonosító.
- „*cy.contains(...)*”: a paraméterben megadott szöveggel keresünk az oldalon egy elemet.
- Amennyiben a kijelölt elemünk egy gomb, a „click” metódust meghívhatjuk. Ezzel szimuláljuk az elemre kattintást. Ha egy szöveges mezőnek szeretnénk értéket adni, a „type” eljárást használjuk, aminek paraméterének meg kell adnunk, hogy mit helyezzünk el az adott beviteli mezőben. Egyéb input tageken végezhető eljárások a „check” és a „select('select_option_szovege)’”.

Az „*npx cypress open*” parancs kiadása után az E2E (End To End) teszt és a böngésző kiválasztása után meg kell adnunk, hogy melyik az a teszt fájlt, amit szeretnénk futtatni. Ezután láthatjuk a tesztelő által végrehajtott lépéseket.

A 8.1. kódban látható egy Cypress tesztre példa. Bejelentkezek egy már létrehozott, „mintamarton” nevű felhasználóval és létrehozok egy új csoportot, majd vizsgálom, hogy visszakapom-e a megerősítő üzenetet az oldalon.

8.1. Programkód. Cypress példa csoport létrehozásához

```
1 describe('Groups', () => {
2   it('mintamarton creates a group', () => {
3     cy.login({username: 'mintamarton'});
4     cy.visit('/groups');
5     cy.get('#createGroup').click();
6     cy.get('#name').type('Minta_Marton_
    ↳ csoportja');
7     cy.get('#create').click();
8     cy.contains('Csoport sikeresen_
    ↳ létrehozva!');
9   });
10 }
```

9. fejezet

Külső komponensek, technológiák

9.1. SweetAlert

A SweetAlert egy rendkívül elterjedt JavaScript könyvtár, amely segítségével felugró ablakban üzenhetünk a felhasználónak. Számos konfigurációs lehetőség áll rendelkezésünkre, ahol többek között a megjelenített szöveg témájához tudjuk igazítani az ablakban megjelenő animációt, ikont illetve gombokat. Megadhatjuk, hogy hogyan viszonuljon az oldal többi eleméhez (ne engedje a usert kikattintani belőle).

Kétféleképpen szerepel a projektben, a Laravel-ből[36] kiváltott ablakok a composer-en keresztül kerültek telepítésre package formájában, a JavaScript-beli használatát pedig egy CDN-es[35] hivatkozás segítségével biztosítottam.

Laravel-ben annyi a feladatunk a használata során, hogy behivatkozzuk a „*Real-Rashid\Facades\SweetAlert*” névtérben található „*Alert*” osztályt, majd ennek hívjuk a statikus metódusait. Ezek közül kettőt használtam: a „*success*”-t és a „*warning*”-ot. Egy használat így néz ki: „*Alert::success('Sikeres módosítás!')*;”.

JavaScript-ben legfőképpen a törlést és frissítést biztosító form-ok elküldésének megerősítésére használtam. Amikor rögzíteni kívánjuk az űrlapunkat, megállítom a küldést, és megjelenítek a Swal.fire metódussal egy felugró ablakot, ahol megerősítés után továbbküldjük a formot, egyéb esetben elvetjük a műveletet.

9.2. Spatie süti engedélyező

A Spatie egy jól ismert szoftverfejlesztő cég, amely nyílt forráskódú csomagokat készít. Széleskörű webfejlesztési problémákra kínálnak megoldást, amikkel általánosságban meggyűlik a bajuk a programozóknak, de ezek akár lehetnek akár gyakran implementálandó funkciók is.

Egy általuk készített package, amit felhasználtam a projekt elkészítése során a „*Laravel Cookie Consent*”[37], amely kifejezetten a süti használatának tényének közlésére

és elfogadására használatos. Erre azért volt szükséges, mert a teszt kitöltése során a session-be helyezzük el a kitöltött teszt pillanatbeli állapotát.

Ennek a megoldása roppant egyszerű volt, hiszen a csomag telepítése után csak be kellett hivatkozni az oldal tartalma alá az alábbi utasítással „`@include("cookie-consent:index")`”, illetve a „`.env`” fájlunkon belüli engedélyezés után csak a megjelenő szöveget kellett személyre szabni.

9.3. TinyMCE

A TinyMCE[38] egy olyan nyílt forráskódú, ingyenes WYSIWYG (What You See Is What You Get) szövegszerkesztő, amely biztosítja a felhasználók számára a weboldalakon lévő tartalmak szerkesztését és formázását oly módon, hogy a végrehajtott módosításokat folyamatosan láthatják.

Minden olyan funkciót magában hordoz, amelyre szükségünk lehet egy tananyag összeállításához (bekezdések, betűméret, kiemelések, listák használata, táblázatok, YouTube videók beszúrása). Mivel rendkívül könnyű a használata, ezért nem igényel semmi magyarázatot, hiszen nagyon hasonlít a Microsoft Word-ben található funkciók megjelenítéséhez.

A szerkesztő testreszabásához különböző kiegészítőket telepíthetünk, amelyek kiegészítik az editor funkcionalitását, mint például képek, videók beszúrása. A pluginok mellett különféle paraméterezési lehetőségünk is van, mint például a szerkesztői felület csak olvashatóvá tétele, a szerkesztői eszközöket tartalmazó menüsév eltávolítása. Mindezek meghatározására a tinymce inicializációjánál van lehetőségünk. Sajnos az editor ingyenessége miatt a lehetőségeink száma eléggé szűkös, hiszen rengeteg elérhető módosítás fizetős licenz esetében van csak biztosítva. Ezért is az alkalmazásom a legalapvetőbb verziót tartalmazza – ezért is jelenik meg a jobb felső sarkában az „Upgrade” felirat.

Mivel a szerkesztett szöveget Hypertext-ként tárolja egy *textarea*-n belül, így könnyedén elmenthető adatbázisba, majd ebbe hasonló könnyedséggel vissza is tölthető, biztosítva ezzel a megfelelő megjelenítést.

9.4. Font Awesome

A Font Awesome[39] főoldalán az alábbi bemutatkozás található: „A Font Awesome az internet az ikonkönyvtára és eszközkészlete, amelyet tervezők, fejlesztők és tartalomgyártók milliói használnak”. Hatalmas választékot kínál vektorgrafikus ikonok körében, amiknek elég nagy része fizetős, viszont az ingyenes kategóriában is egészen biztosan találhatunk számunkra megfelelő elemeket.

Az ikonjainkat könnyedén beilleszthetjük egy egyszerű „toolkit” legenerálásával, aminek a *script* hivatkozását kell elhelyeznünk azokon az oldalakon, ahol szeretnénk azokat alkalmazni. Az oldalon kikereshetjük a számunkra megfelelő ikont, majd a számunkra szimpatikus elem kijelölése után egy kattintással kimásolhatjuk annak HTML tagjét, amely tartalmazza azokat a szükséges osztályokat, amelyekkel az oldalon láthatóvá válik. Amellett, hogy igényes megjelenítést ad az oldalunknak, intuitívabbá is tehetjük az oldalunk funkcióit az ilyen ikonok felhelyezésével.

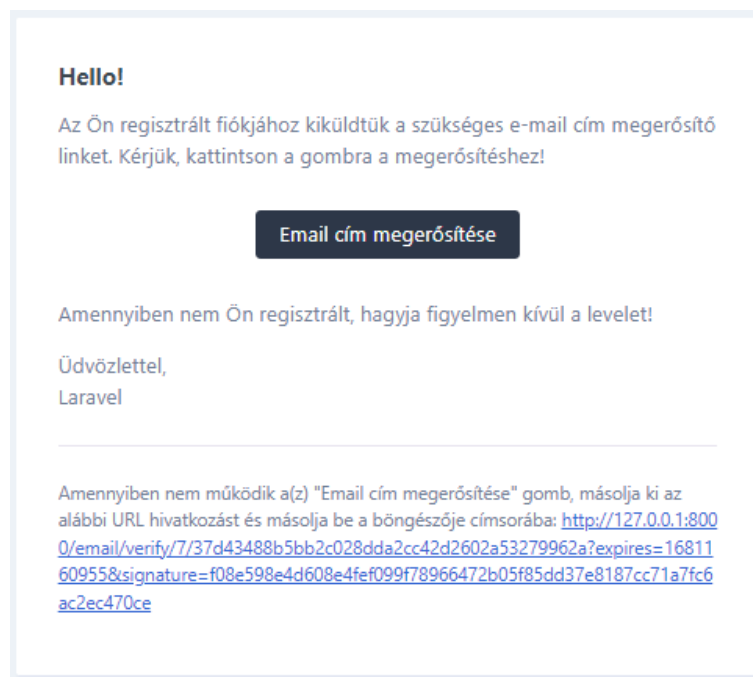
Az egyszerű használat mellett lehetőség van a szimbólumok személyre szabására is. A megfelelő CSS utasítások segítségével a betűméret, betűszín, valamint a háttérszín is saját belátásunk szerint módosíthatjuk.

9.5. Mailtrap

A Mailtrap[40] egy tökéletes megoldást nyújt az e-mail küldések teszteléséhez egy hamis SMTP szerver biztosításával. Egy felhasználói fiók létrehozása után egyedi kulcsokat (felhasználónév és jelszó) kapunk, amiket a *.env* fájlunkon belül a „MAIL” szekció megfelelő részeinek értékül adunk. Eközben ügyeljünk arra, hogy az API azonosítók lekérését ne zárjuk be addig, amíg biztosak nem vagyunk abban, hogy a szerverünk megfelelően működik.

A helyes konfigurálás után elérhetjük azt, hogy a fejlesztés során elküldendő leveleket ne egy más által esetlegesen használatban lévő e-mail-re küldjük ki, hanem a küldésnek csak a funkcionalitását teszteljük úgy, hogy összes elküldött e-mail-t egyetlen helyen összegyűjtjük, azokat elemezzük és utána elvégezzük az esetleges finomításokat a programunkon.

A tesztelés során havonta 100 e-mail-t küldhetünk az ingyenes verzióban, így bőven biztosított számunkra, hogy akár több megjelenítést is kipróbáljunk a fejlesztés során.



9.1. ábra. Regisztrációt megerősítő e-mail



9.2. ábra. A felhasználót értesítő e-mail arról, hogy eltávolították egy csoportból

10. fejezet

Telepítési útmutató

Hajtsuk végre az alábbi lépéseket az alkalmazás telepítése során:

1. Töltsük le/klónozzuk a GitHub repository-t!
2. A konzol felületen navigáljunk el a klónozott/kicsomagolt állományba!
3. Futtassuk az ***npm install*** parancsot!
4. Adjuk ki a ***composer install*** utasítást!
5. ***composer require pusher/pusher-php-server*** és ***npm install --save-dev laravel-echo pusher-js*** utasításokkal telepítsük az üzenetek küldéséhez szükséges csomagokat!
6. Generáljuk le az alkalmazás egyedi kulcsát a ***php artisan key:generate***-tel!
7. A „*env.example*” fájlnak megfelelően **készítsünk egy új fájlt *.env* névvel!**
8. Az **APP_URL** értéke legyen a következő: **http://localhost:8000!**
9. Az adatbázis beállításokat a saját adatbázisunknak megfelelően írjuk át!
10. A **BROADCAST_DRIVER** értékét írjuk át „*log*”-ról „*pusher*”-re!
11. Adjuk hozzá az alábbi sort a Laravel Debugbar kikapcsolásához: **DEBUGBAR_ENABLED=false!**
12. A **QUEUE_CONNECTION** értékét írjuk át „*sync*”-ról „*database*”-re!
13. A „MAIL” szekció beállításait konfiguráljuk az SMTP szererünknek megfelelően. Amennyiben Mailtrap-et szeretnénk használni tesztelés miatt, a **MAIL_HOST**-ot írjuk át „*sandbox.smtp.mailtrap.io*”-re, a **MAIL_PORT**-ot „*2525*”-re, a **MAIL_ENCRYPTION**-t „*tls*”-re. Ezután hozzunk létre egy fiókot a **https:**

`//mailtrap.io`-n, majd a `https://mailtrap.io/api-tokens` oldalon generáljunk le egy API kulcsot. A `MAIL_USERNAME` és `MAIL_PASSWORD`-hoz másoljuk be az itt kapott értékeket.

14. Az üzenetek küldéséhez regisztráljunk a `https://pusher.com`-ra, majd hozzunk létre egy Channel-t. Ezután a csatorna beállításai között megtalálható egy „*App Keys*” opció, ahol megtalálhatjuk azokat a beállításokat és azoknak értékeit, amiket be kell írunk a „*.env*” fájlban található **PUSHER** megfelelő részeihez.
15. Az adatbázis tábláit készítsük el a migrációk futtatásával, amit a ***php artisan migrate*** paranccsal érhetünk el. Ezután futtassuk az alábbi utasítást: ***php artisan db:seed --class=UserSeeder***! Ezzel hozzáadjuk a felhasználók táblájához a kiinduló admin felhasználót.
16. A felhasználónévvel való bejelentkezéshez navigáljunk el a projekten belül található „`vendor/laravel/ui/auth-backend`” könyvtárba, majd az ott található **`AuthenticatesUsers.php`**-ban a „*username*” függvényben-ben írjuk át az „*email*”-t „*username*”-re
17. A futtatáshoz adjuk ki az ***npm run dev***, ***php artisan serve*** és ***php artisan queue:work*** parancsokat!

A fenti parancsok és konfigurálások elvégzésével futtathatóvá tettük az alkalmazást. Bejelentkezhetünk az „admin” felhasználónévű és „admin” jelszavú felhasználóval, hogy létrehozzunk tanárokat, illetve regisztrálhatunk diákként is.

Összegzés

Az alkalmazás fejlesztése közben törekedtem arra, hogy egy igényesen összeállított és jól funkcionáló rendszert adjak ki a kezem közül. A fejlesztéssel párhuzamosan folyamatosan kerestem olyan tartalmakat (akár cikkek, akár Youtube videósorozatok, mint a már korábban említett „Laravel Daily” csatorna), amikből megtanulhattam, hogy a háttérben zajló folyamatokat hogyan lehet a lehető legjobb módon optimalizálni és fenntarthatóvá tenni, így a kód is folyamatos átalakuláson ment keresztül a fejlesztés alatt.

Úgy gondolom, hogy ez a projekt egy nagyszerű lehetőséget adott arra, hogy átlássam egy nagyobb volumenű alkalmazásnak a fejlesztését úgy, hogy közben egyre átfogóbb tudást szereztem a Laravel működéséről és használatáról. Külön örülök annak, hogy amennyiben ilyen téren szeretnék elhelyezkedni, nyugodtan referálhatnak erre a munkámra, mert úgy gondolom, hogy ez egy szépen összeállított és lefejlesztett produktum.

A szakdolgozat megírásának végeztével szerintem mindent sikerült megvalósítani, amit úgy éreztem, hogy szükséges egy ehhez hasonló rendszerbe. Elképzelhetőnek tartom, hogy a jövőben bővítem a funkcionalitását különböző feladattípusokkal, egyéb formátumú tananyagok hozzáadásának lehetőségével. Amennyiben úgy érzem, hogy olyan szinten áll a szoftver, hogy végfelhasználók is szívesen használnák, kiadhatnám egy ingyenes, nyílt forráskódú alkalmazásként.

A szakdolgozat forráskódját az alábbi GitHub-os repository-n keresztül érhetik el: <https://github.com/KecseKaroly/Vizsgaztato>

A működést bemutató videót pedig az alábbi hivatkozásra kattintva tekinthetik meg: <https://youtu.be/16AfIONGfNQ>

Irodalomjegyzék

- [1] EKKE MOODLE *elérhetősége*
<https://elearning.uni-eszterhazy.hu>, 2023.03.12.
- [2] EKKE PEGAZUS: *elérhetősége*
<https://pegazus.uni-eszterhazy.hu>, 2023.03.28.
- [3] REDMENTA *elérhetősége*
<https://redmenta.com/hu>, 2023.03.28.
- [4] QUIZIZZ *elérhetősége*
<https://quizizz.com/join>, 2023.03.28.
- [5] ARTTURI JALLI: *What Is Laravel?*, 2022.
<https://tinyurl.com/WhatIsLaravel>, 2023.03.28.
- [6] LARAVEL LIVEWIRE *játékos leírása*
<https://laravel-livewire.com>, 2023.04.03.
- [7] TAILWIND *CSS dokumentációja*
<https://tailwindcss.com/docs/installation>, 2023.04.10.
- [8] DBDIAGRAM
<https://dbdiagram.io>, 2023.04.13.
- [9] LARAVEL *Migrations dokumentációja*
<https://laravel.com/docs/10.x/migrations#introduction>, 2023.03.30.
- [10] LARAVEL *Eloquent dokumentációja*
<https://laravel.com/docs/10.x/eloquent>, 2023.03.30.
- [11] LARAVEL *Eloquent: Relationships, Eager Loading és N + 1 Query Problem*
<https://tinyurl.com/LaravelEagerLoading>, 2023.03.31.

- [12] LARAVEL *Controllers dokumentációja*
<https://laravel.com/docs/10.x/controllers>, 2023.03.30.
- [13] POVILAS KOROP [„LARAVEL DAILY”] *Code Review játszási listája*
<https://tinyurl.com/LaravelDailyCodeReview>, 2023.03.31.
- [14] LARAVEL *Events & Listeners dokumentációja*
<https://laravel.com/docs/10.x/events>, 2023.03.31.
- [15] KINGSLEY OKPARA *Laravel 8 Events and Listeners with Practical Example*, 2020.12.25.
<https://tinyurl.com/LaravelEventListenersExample>, 2023.03.31.
- [16] LARAVEL *Blade Sablon validációs hibakezelése*
<https://laravel.com/docs/10.x/blade#validation-errors>, 2023.03.31.
- [17] LARAVEL *Bladet Template-ek direktívái*
<https://laravel.com/docs/10.x/blade#blade-directives>, 2023.04.04.
- [18] LARAVEL *Blade Template-ek kialakítása öröklődéssel*
<https://tinyurl.com/LaravelLayoutInheritance>, 2023.03.31.
- [19] LARAVEL *7-es verzió autentikációs dokumentációja*
<https://laravel.com/docs/7.x/authentication>, 2023.04.01.
- [20] LARAVEL *10-es verzió autentikációs dokumentációja*
<https://laravel.com/docs/10.x/authentication>, 2023.04.01.
- [21] ALEX GARRETT-SMITH: *Laravel Authentication: UI vs Jetstream vs Fortify vs Breeze*, 2020.12.27.
<https://tinyurl.com/Laravel-AuthComp>, 2023.04.01.
- [22] LAREVEL *Middleware dokumentáció*
laravel.com/docs/10.x/middleware, 2023.04.01.
- [23] LAREVEL *Authorization - Policy dokumentáció*
<https://tinyurl.com/LaravelPolicy>, 2023.04.01.
- [24] LARAVEL LIVEWIRE *Properties*
<https://laravel-livewire.com/docs/2.x/properties>, 2023.04.03.

- [25] LARAVEL LIVEWIRE *Actions*
<https://laravel-livewire.com/docs/2.x/actions>, 2023.04.03.
- [26] LARAVEL LIVEWIRE - EVENTS
<https://laravel-livewire.com/docs/2.x/events>, 2023.04.03.
- [27] LARAVEL LIVEWIRE *Validáció*
<https://laravel-livewire.com/docs/2.x/input-validation>, 2023.04.03.
- [28] LIVEWIRE *Drag & Drop Sorting A List*
<https://laravel-livewire.com/screencasts/s8-dragging-list>, 2023.04.03.
- [29] LIVEWIRE/SORTABLE *GitHub repository*
<https://github.com/livewire/sortable>, 2023.04.03.
- [30] LIVEWIRE SCREENCASTS *Basic Search*
<https://laravel-livewire.com/screencasts/s7-search>, 2023.04.02.
- [31] PUSHER CHANNELS *GitHub repository*
<https://pusher.com/channels/>, 2023.04.02.
- [32] LARAVEL ECHO *GitHub repository*
<https://github.com/laravel/echo>, 2023.04.02.
- [33] LARACASTS *Laravel Cypress repository:*
<https://github.com/laracasts/cypress>, 2023.04.13.
- [34] LARACASTS *Cypress and Laravel Crash Course*
www.youtube.com/watch?v=Td8o2LA9gI0, 2023.04.13.
- [35] SWEETALERT2 *documentáció (JavaScript)*
<https://sweetalert2.github.io>, 2023.04.03.
- [36] RASHID ALI *SweetAlert package (Laravel) repository*, 2023.04.03.
<https://github.com/realrashid/sweet-alert>, 2023.04.03.
- [37] SPATIE *Laravel Cookie Consent repository*
<https://github.com/spatie/laravel-cookie-consent>, 2023.04.03.
- [38] TINYMCE *dokumentációja Laravelhez*
<https://tinymce.com/TinyMCELaravel>, 2023.04.14.

[39] FONTAWESOME *ingyenes ikonjai*

<https://fontawesome.com/search?o=r&m=free>, 2023.04.14

[40] MAILTRAP *bemutatása*

<https://mailtrap.io>, 2023.04.03.

NYILATKOZAT

Alulírott KECSE KÁROLY DÁNIEL....., büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, WEBES.....
VIZSGA'ZTATÓ RENDSZER..... című szakdolgozat (diplomamunka) önálló szellemi termékem. Amennyiben mások munkáját felhasználtam, azokra megfelelően hivatkozom, beleértve a nyomtatott és az internetes forrásokat is.

Tudomásul veszem, hogy a szakdolgozat elektronikus példánya a védés után az Eszterházy Károly Katolikus Egyetem könyvtárába kerül elhelyezésre, ahol a könyvtár olvasói hozzájuthatnak.

Kelt: EGER....., 2023 év április..... hó 15..... nap.

Kecse Károly.....

aláírás