



Jelenlét és hiányzás követő alkalmazás fejlesztése

Készítette

Györkis Tamás

Programtervező informatikus BSc

Témavezető

Dr. Király Roland

Egyetemi docens

EGER, 2024

Tartalomjegyzék

Bevezetés	4
1. Alkalmazás bemutatása	5
1.1. Adatbázis	5
1.2. Általános információk	6
1.2.1. Felhasználók	6
1.2.2. Általános funkciók	7
1.3. Adminisztrátori nézet, funkciók	8
1.3.1. Felhasználók kezelése	8
1.3.2. Alkalmazás konfiguráció	8
1.3.3. Félévek, termék, tantárgyak, kurzusok, órák kezelése	9
1.4. Tanári nézet, funkciók	10
1.4.1. Oktatott tárgyak	10
1.4.2. Igazolások	11
1.5. Hallgatói nézet, funkciók	12
1.5.1. Tantárgyak, kurzusok	12
1.5.2. Hallgató által létrehozott igazolások	12
2. Felhasznált technológiák, csomagok	13
2.1. Laravel	13
2.1.1. Keretrendszer alapjai	13
2.1.2. Adatbázis és modellek	15
2.1.3. Eloquent ORM	17
2.1.4. Blade sablon fájlok	18
2.1.5. Autentikáció	19
2.1.6. Jogosultság kezelés	19
2.1.7. Email küldés	21
2.1.8. Naptár exportálása	22
2.2. Tailwind	23
2.2.1. DaisyUI	23
2.3. Livewire	24
2.3.1. Komponensek és data binding	24

2.3.2.	Livewire és SPA	25
2.3.3.	Események	25
2.3.4.	WireToast: Livewire alapú értesítések	27
2.4.	Websocket és Pusher	27
2.4.1.	Websocket	27
2.4.2.	Pusher channels	28
2.4.3.	Csatornák, események létrehozása	29
2.4.4.	Események fogadása JavaScript, LiveWire esetén	29
2.5.	QR-kód generálás	29
2.6.	Naptár - FullCalendar.js	29
2.6.1.	Alpine.js: PHP kód eredményének lekérése JavaScript-ben . . .	29
2.6.2.	Beállításai	29
2.6.3.	Kapcsolat a keretrendszerrel LiveWire segítségével	29
2.7.	Diagramok - Chart.js	29
2.7.1.	Beállításai	29
2.7.2.	Kapcsolat a keretrendszerrel LiveWire segítségével	29
2.8.	Progressive Web Apps	29
2.8.1.	A PWA jelentése, jelentősége	29
2.8.2.	Laravel PWA csomag	29
2.8.3.	A manifest fájl	29
2.8.4.	Service Worker	29
3.	Az alkalmazás tesztelése	30
3.1.	Manuális tesztelés	30
3.2.	Automatizált tesztelés	30
3.3.	Terheléses tesztelés	30
3.4.	Laravel Pint és Github actions	30
4.	Alkalmazás telepítése	31
4.1.	Laravel Forge	31
4.2.	Kézi telepítés lépései	31
	Összegzés	34
	Irodalomjegyzék	35

Bevezetés

Amikor szakdolgozati témaválasztás előtt álltam, sokat gondolkoztam a témán. Mindenképpen egy olyan alkalmazást szerettem volna elkészíteni, mely ténylegesen hasznos is lehet. Egyetemi éveim alatt demonstrátorként tanítottam több féléven át az egyetemen, és innen jött a felismerés, hogy jó lenne, ha a hallgatók jelenlétét, hiányzásait ne táblázatokban kelljen vezetni, hanem egy külön eszköz legyen rá készítve. Innen származik az alkalmazás ötlete.

A megvalósítás során törekedtem arra, hogy egy jól átlátható, ergonomikus weboldalt készítsek, amit – esetleges kisebb módosításokkal – ne csak egyetemi környezetben lehessen használni. Ezekből kifolyólag egy webalkalmazást készítettem, mivel ezt a leg-egyszerűbb elérni, hiszen csak egy webböngésző szükséges hozzá. A tanulók és tanárok által használt oldalak reszponzívan lettek elkészítve, ezáltal biztosítva, hogy mobiltelefonon is használni lehessen. Illetve az alkalmazás PWA¹ funkciókkal rendelkezik, aminek jelentőségére később térek ki, de előjáróban annyit érdemes megemlíteni róla, hogy lehetővé teszi a weboldal alkalmazásként való telepítését számítógép és telefon esetén is, ezáltal egy rendes alkalmazás érzését keltve.

A megvalósításhoz a Laravel keretrendszert használtam, mely egy PHP alapú, MVC² keretrendszer, amivel tanulmányaim során találkoztam, és egyből megkedveltem. Nem titkolt célom a szakdolgozatommal, hogy bemutassam, hogy a mai, JavaScript preferált világban, továbbra is lehetséges modern, a mai kort kielégítő weboldalt készíteni PHP segítségével, amihez különböző csomagokat alkalmaztam, amiknek működését, illetve egymással való működését a későbbiekben fogom kifejteni.

Szakdolgozatom négy részre osztottam: az 1. fejezetben magát az alkalmazást mutatom be. A 2. fejezetben magát a keretrendszert, illetve a használt csomagokat, a technikai részleteket tárgyalom. Végül a 3. és a 4. fejezetben a tesztelést és a telepítést részletezem.

¹ Jelentése: Progressive Web App: egy olyan alkalmazás, ami webes technológiákat használ, de platform specifikus alkalmazásként viselkedik.[3]

² Jelentése: Model-View-Controller.[9]

1. fejezet

Alkalmazás bemutatása

1.1. Adatbázis

Az 1.1 ábrán látható az alkalmazás által használt adatbázis, egyed-kapcsolat diagrammal ábrázolva, melyről szeretnék néhány szót ejteni.



1.1. ábra. Az alkalmazás által használt adatbázis.

- *users*, *user_roles*: ez a tábla tárolja a felhasználók adatait, illetve a felhasználókhoz tartozó jogosultsági köröket. Minden felhasználóról tárolunk egy azonosítót,

egy Neptun[1] kódot¹, nevet, email címet, egy titkosított jelszót, egy profilképet, a felhasználó által használt nyelvet, illetve egy UUID-t², azaz egy egyedi azonosítót, amit az órarend exportálásához lehet használni, erről később. Az azonosító oszlopra azért van szükség, mivel az alkalmazás fel van készítve arra, hogy Neptun kód nélkül rendelkező felhasználókat is tudjon kezelni.

- *terms, places*: a félévek és termek (helyek) tárolására szolgáló táblák.
- *subjects, courses, course_classe, class_login_links*: ezen táblák tárolják a tantárgyakat (amik többek között rendelkeznek egy azonosítóval, névvel, leírással, kreditértékkel, és egy tantárgyfelelőssel), a tantárgyakhoz tartozó kurzusokat (amihhez tárolok egy azonosítót, ami minden esetben egyedi, egy kurzus azonosítót, ami minden félév és tantárgy esetében kell egyedinek lennie, tartozik hozzá egy félév, illetve egy létszám limit), illetve kurzusokhoz pedig órák (amiknek van egy kezdete, vége, illetve egy terem, ahol tartják). Az utolsó tábla pedig az órákhoz tartozó bejelentkező linkeket tartalmazza, ennek jelentőségéről kicsit később.
- *attendances*: itt tárolom az egyes kurzusok résztvevőinek az órai jelenléteit, melyek lehetnek: nincs kitöltve, jelen, késés (mely esetben percben lehet tárolni a mértékét), hiányzás, igazolt hiányzás.
- *justification, justification_acceptances, justification_pictures*: ezek a táblák tárolják az igazolásokat, az igazoláshoz tartozó feltöltött képeket, illetve az igazolásban érintett tanárok válaszát, hogy elfogadják-e ó, vagy sem.
- Az egyéb, nem említett táblák inkább kapcsolótábla funkciót töltenek be, vagy a keretrendszernek vannak rá szükségei. Például a *jobs* tábla a háttérben végrehajtandó feladatokat (job) tartalmazza.

Itt érdemes még megemlíteni, hogy a fejlesztés során a MySQL nevezetű, relációs adatbázist használtam, viszont a Laravel keretrendszerből adódóan sok más típusú adatbázis szoftverrel használható az alkalmazás, mivel a táblák felépítése a keretrendszer nyújtotta módon van elkészítve.

1.2. Általános információk

1.2.1. Felhasználók

Az alkalmazás 4 különböző jogkört különböztet meg a felhasználók esetén, melyekből egyszerre többet is birtokolhat a felhasználó:

¹ A neptun kód egy 6 karakter hosszú, egyedi azonosító.
²

- Szuper adminisztrátor: képes a felhasználók adatainak – és jogköreinek – szerkesztésére, illetve az alkalmazás alapbeállításainak módosítására.
- Adminisztrátor: a félévek, termek, tantárgyak, kurzusok, órák, készítésére, módosítására, törlésére jogosult.
- Tanár: megtekintheti a tanított óráit, kurzusait, hallgatóit. Adminisztrálni tudja az órákon való részvételt, illetve megtekintheti és bírálhatja a hozzá érkezett igazolásokat.
- Hallgató: a hozzárendelt óráit, tantárgyait látja, megtekintheti minden kurzus esetén az egyes órák státuszát, illetve igazolásokat hozhat létre

Az alkalmazás természetesen rendelkezik bejelentkezés, regisztráció funkciókkal. A felhasználók Neptun[1] kódjuk vagy email címük, illetve a jelszavuk megadásával tudnak bejelentkezni. Regisztrálni csak abban az esetben tudnak, ha az oldal beállításai-ban ezt engedélyezték, egyéb esetben a szuper adminisztrátorok tudnak felhasználókat létrehozni vagy importálni. Belépés után a profil szerkeszthető, a jelszó cserélhető.

1.2.2. Általános funkciók

Bejelentkezés után a főoldalon a felhasználók pár, számukra fontos vagy érdekes információ láthatnak. Tanulók esetében a mai napi óráikat, illetve adminisztrálásra váró igazolásaikat. Tanárok esetén szintén a mai óráikat, illetve a kapott igazolásaikat láthatják. Az adminisztrátorok pár statisztikát láthatnak az oldal kapcsán.

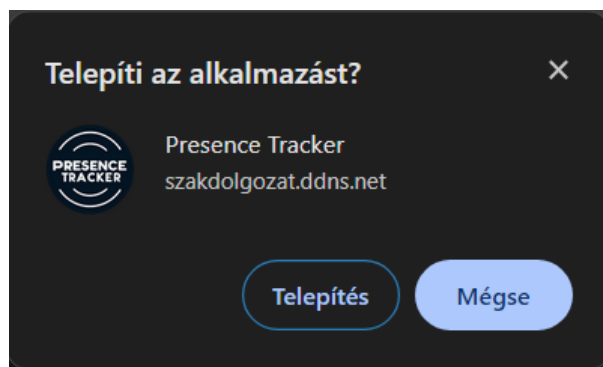


1.2. ábra. A főoldal kinézete abban az esetben, ha a felhasználó minden szerepkörrel rendelkezik. A két bal oldali oszlop a hallgatói nézethez tartozik, a két jobb oldali pedig az oktatóihoz, míg a statisztika adminisztrátorok esetén jelenik meg.

A profil kép melletti gombra kattintva a felhasználó meg tudja változtatni az oldal nyelvét, Angol és Magyar nyelv közül választva, illetve az oldalt kinézetét: világos és sötét mód között váltogatva. A nyelv beállítása eltárolódik a felhasználóhoz, így újabb bejelentkezés esetén automatikusan beállításra kerül.

Tanuló vagy tanár esetén megjelenik az *Órarend* menüpont is, ahol egy helyen láthatják a heti óráikat a felhasználók. Az egyes órák különböző színekkel jelennek meg, ezzel jelezve, hogy az adott órát a felhasználó tanítja-e, vagy hallgató esetén az órán jelen volt, hiányzott, késett, igazoltan hiányzott, vagy még nincs kitöltve a jelenlét. Lehetőség van az órarend exportálásra is, mely során egy *.ics* kiterjesztésű fájl áll elő. Ez egy széles körben használt általános naptár formátum, amit a felhasználók be tudnak importálni a gyakran használt naptár programokba[2]. Ezáltal kedvenc naptár alkalmazásukban is nyomon tudják követni óráikat.

Lehetőség van a weboldalt alkalmazásként telepíteni, ahogy az az 1.3 ábrán is látható, így gyorsan hozzáférhetnek az alkalmazás funkcióihoz, egy „rendes” alkalmazás érzését keltve ezzel. Ezt a PWA³ teszi lehetővé, melynek működésére későbbiekben térek ki.



1.3. ábra. Az oldal telepítésére szolgáló felugró ablak.

1.3. Adminisztrátori nézet, funkciók

1.3.1. Felhasználók kezelése

Szuper adminisztrátor esetén a felhasználó képes kezelni az alkalmazásban tárolt felhasználókat, kivéve saját magát (személyes adatait továbbra is meg tudja változtatni). A listában képes szűrni a felhasználók között, megváltoztatni profil adataikat, jelszavukat alaphelyzetbe állítani, szerkeszteni jogosultsági szintjeiket, illetve akár törölni is őket, ha szükséges. Tud új felhasználókat létrehozni, illetve képes fájlból is importálni adatokat.

1.3.2. Alkalmazás konfiguráció

Szintén a szuper adminisztrátor körébe tartozik az oldal beállításainak kezelése. Az alábbi beállításokat tudja kezelni:

³ Jelentése: Progressive Web App: egy olyan alkalmazás, ami webes technológiákat használ, de platform specifikus alkalmazásként viselkedik.[3]

- Oldal neve: az oldalon megjelenő név megváltoztatása.
- Regisztráció engedélyezése: ezzel lehet engedélyezni, hogy a felhasználók maguktól is tudjanak-e felhasználói fiókot készíteni az oldalon.
- Saját Neptun[1] kód megváltoztatása: beállítható, hogy a felhasználó meg tudja magának változtatni ezt az értéket, vagy sem.
- Kötelező Neptun kód: az oldal használható ezen kód megadása nélkül is, ez itt szabályozható.
- Alkalmazás képe: itt tölthető fel új kép, ami a fő oldalon jelenik meg látogatók esetén.

1.3.3. Félévek, termek, tantárgyak, kurzusok, órák kezelése

Minden adminisztrátor képes ezen adatok kezelésére.

Félévek esetén egy nevet, illetve egy kezdő és vég dátumot szükséges megadni. Ellenőrizve van, hogy a félévek nem ütközhetnek egymással. A listában egy pipa jelzi a jelenlegi félévet, ha van ilyen.

A termeknél elegendő egy nevet megadni, ezeket lehet az egyes órákhoz hozzárendelni.

Tantárgyak esetén egy tárgy kódot, egy nevet, egy opcionális leírást, egy kredit értéket, illetve egy tantárgyfelelőst tárol, akinek rálátása van a tárgy összes kurzusára. Ezen belül minden kurzus egy tárgyhoz kapcsolódik. Ezek szintén rendelkeznek egy kóddal, ami minden tárgy és félév esetén egyedi, szintén egy opcionális leírással, illetve nulla, egy, vagy több tanárral. Minden kurzus egy félévhez van rendelve. A kurzusokhoz hozzárendelhetők a hallgatók, és csak hallgatók.

A kurzusokhoz órák hozhatóak létre, amik a féléven belül lehetnek. Minden óra külön kezdés és vég időponttal rendelkezik, illetve egy teremmel, ahol tartják. Az óra létrehozásakor lehetőség van arra is, hogy a félév végig heti ismétléssel hozza létre az órákat – természetesen később egyenként lehet őket szerkeszteni, törölni –, ezáltal megkönnyítve az adminisztrációt.

1.4. ábra. Új óra hozzáadása.

1.4. Tanári nézet, funkciók

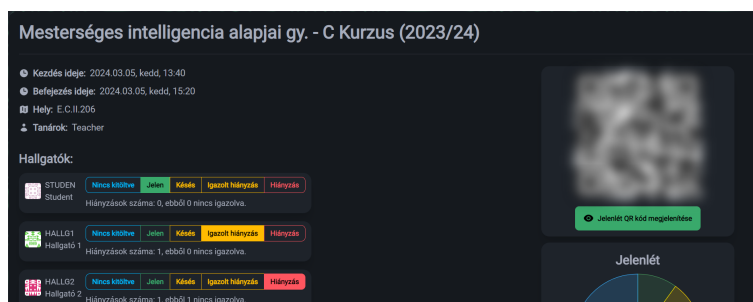
1.4.1. Oktatott tárgyak

A tanárok megtekinthetik az általuk oktatott – vagy tantárgyfelelősként hozzáadott – tárgyakat, és az azokhoz kapcsolódó adatokat. Igény szerint szűrhetnek kód, név, vagy félévre is. Minden kurzus esetén megtekinthetik az alap adatokat, a kurzushoz tartozó órákat, hallgatókat. Itt van lehetősége a tanárnak hozzáadni hallgatókat a kurzushoz, illetve egy listában megtekinteni diákokra lebontva, hogy mennyit hiányoztak, mennyit késtek, ebből mennyi az igazolatlan.

A kurzus órái listában érhetik el az egyes órákat, ahol adminisztrálhatják a jelenlétet. Egy órát megnyitva láthatják az ahhoz kapcsolódó adatok, illetve bal oldalon egy listát a kurzus tanulóiról, ahol beállíthatják a hallgató státuszát. Amennyiben az adott hallgató a tanár által elfogadott igazolással rendelkezik, akkor ebben az esetben csak „jelen” státusz – ha esetleg mégis megjelent az órán –, vagy pedig „igazol hiányzás” státusz állítható be, illetve egy figyelmeztető üzenet is megjelenik. Ezek mellett megjelenik az adott kurzuson való összes hiányzás és igazolt hiányzások száma is. Hiányzás rögzítése esetén email üzenet formájában is értesítést kap a hallgató.

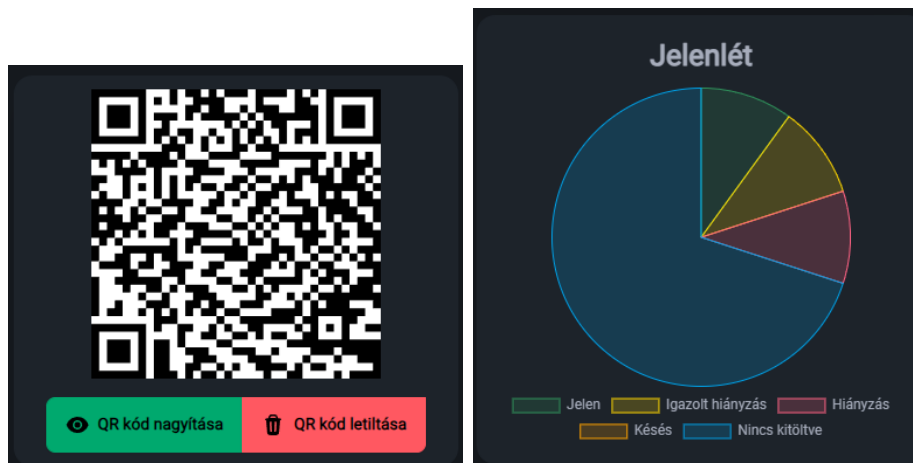
A jobb oldalt kettő doboz foglal helyet. Az első egy Qr⁴ kódot tartalmaz. Ennek segítségével, ha az oktató látható teszi, a diákok a kódot a telefonjukon beolvasva, majd bejelentkezve is „beírhatják magukat” az órára, ezzel felgyorsítva a folyamatot, és a tanárnak sem kell végig mennie a listán. A kód csak az óra végéig működik, illetve ha az oktató úgy sejt, hogy a hallgatók visszaélnék ezzel, akkor hamarabb is letilthatja, illetve generálhat egy másikat. Amikor a hallgató új módon bejelentkezik az órára, a tanári felületen automatikusan frissítésre kerül a státusz. A hallgató csak abban az esetben tudja ezt a funkciót használni, hogy ha fel van iratkozva a kurzusra, és még nem állították be az adott órára a státuszát, ezzel elkerülve, hogy magától átírja esetleges hiányzását, késését.

A másik ilyen doboz egy kis statisztikát mutat kör diagram formájában, megszámlolva a hallgatók státuszát az órán.



1.5. ábra. Egy óra nézete.

⁴ A Qr kód egy információt tartalmazó kép, amit kamerával lehet leolvasni.[4]



1.6. ábra. Egy óra kártyái.

1.4.2. Igazolások

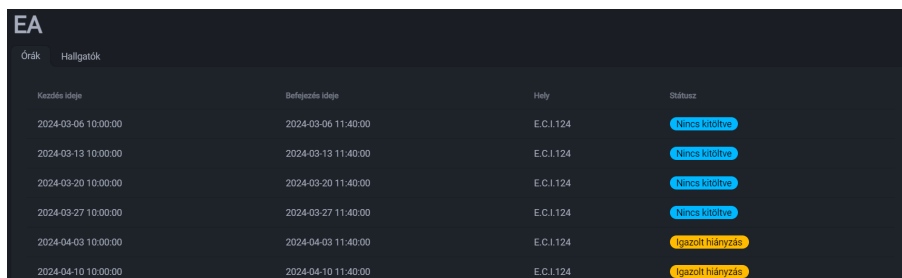
Amikor egy hallgató igazolást ad le, akkor azt minden érintett tanár megkapja, és reagálhat rá; vagy elfogadja, vagy indoklással elutasítja. A lista alaphoz szűrésre kerül, hogy csak a még nem megválaszolt igazolásokat mutassa az oktatónak. Amikor a tanár megnyit egy ilyen igazolást, láthatja, hogy ki küldte be az igazolást, a típusát (orvosi vagy egyéb), az kezdeti és vég időpontot, illetve az esetlegesen feltöltött képeket is megtekintheti. Ezek mellett listába szedve, és tantárgy, majd kurzus alapján kategorizálva látja az általa oktatott és az igazolásban érintett órákat, azok idejét, illetve a hallgató jelenlegi státuszát az órán. Az igazolás elfogadása esetén azon órák, ahol nem „jelen”-re van állítva az állapot, automatikusan „igazolt hiányzás” állapotra kerülnek beállításra. Elfogadás és elutasítás esetén is üzenet kap a hallgató, illetve a főoldalon megtekintheti a még függőben lévő (tehát nem minden érintett tanár által megválaszolt) igazolásait is.

1.7. ábra. Egy elfogadott igazolás, ahogy a tanár látja.

1.5. Hallgatói nézet, funkciók

1.5.1. Tantárgyak, kurzusok

A hallgatók természetesen láthatják a tantárgyaikat, amiket csakugyan szűrhetnek név, kód, félév szerint. Itt egy helyen megtekinthetik kurzusokra lebontva, hogy melyik órán milyen státusz van rögzítve, illetve egy diagramon ezeknek az eloszlását. Megtekinthetik még a kurzuson lévő hallgatótársaikat is.



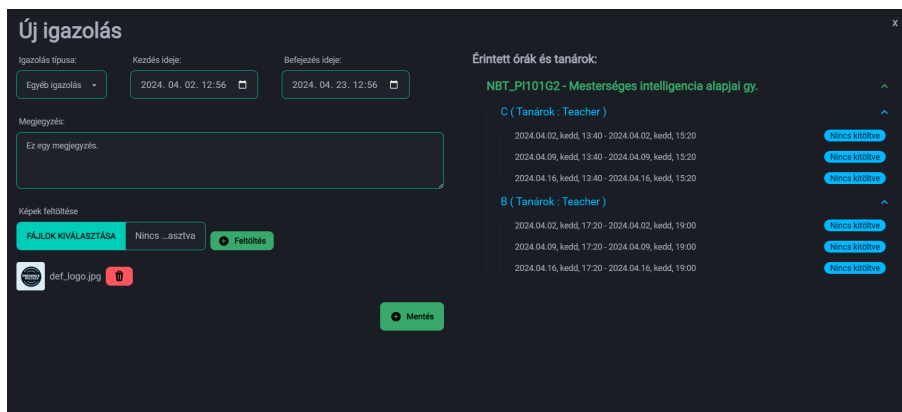
Kezdési idője	Befejezési idője	Hely	Státusz
2024-03-06 10:00:00	2024-03-06 11:40:00	E.C.1.124	Nincs kitöltve
2024-03-13 10:00:00	2024-03-13 11:40:00	E.C.1.124	Nincs kitöltve
2024-03-20 10:00:00	2024-03-20 11:40:00	E.C.1.124	Nincs kitöltve
2024-03-27 10:00:00	2024-03-27 11:40:00	E.C.1.124	Nincs kitöltve
2024-04-03 10:00:00	2024-04-03 11:40:00	E.C.1.124	Igazolt hiányzás
2024-04-10 10:00:00	2024-04-10 11:40:00	E.C.1.124	Igazolt hiányzás

1.8. ábra. A hallgató által látott egy kurzus részlete.

1.5.2. Hallgató által létrehozott igazolások

A hallgató a hiányzásainak igazolására igazolásokat adhat le a rendszerben. A létrehozáskor meg kell adnia a típusát (orvosi vagy egyéb), a kezdeti és vég dátumát, illetve szükség szerint megjegyzést fűzhet, illetve képeket tölthet fel, például az orvosi igazolást lefényképezheti. A dátumok megadása után a hallgató is látja az érintett óráit listába szedve.

Helyes adatok megadása és mentés után a listában is megjelenik az igazolás, amit lenyitva látja a megadott részleteket, illetve az egyes tanárok visszajelzését is, hogy elfogadták-e vagy sem. Amennyiben úgy látja, hogy elrontott valamit, akkor itt törölheti az igazolást, és hozhatja újra létre.



Új igazolás

Igazolás típusa: Egyéb igazolás

Kezdési idője: 2024. 04. 02. 12:56

Befejezési idője: 2024. 04. 23. 12:56

Megjegyzés: Ez egy megjegyzés.

Képek feltöltése: FÁJLOK KIVÁLASZTÁSA Nincs asztva Feltöltés

Érintett órák és tanárok: NBT_PH101G2 - Mesterséges intelligencia alapjai gy.

C (Tanárok : Teacher)

2024.04.02, kedd, 13:40 - 2024.04.02, kedd, 15:20	Nincs kitöltve
2024.04.09, kedd, 13:40 - 2024.04.09, kedd, 15:20	Nincs kitöltve
2024.04.16, kedd, 13:40 - 2024.04.16, kedd, 15:20	Nincs kitöltve

B (Tanárok : Teacher)

2024.04.02, kedd, 17:20 - 2024.04.02, kedd, 19:00	Nincs kitöltve
2024.04.09, kedd, 17:20 - 2024.04.09, kedd, 19:00	Nincs kitöltve
2024.04.16, kedd, 17:20 - 2024.04.16, kedd, 19:00	Nincs kitöltve

Mentés

1.9. ábra. Új igazolás létrehozása képernyő.

2. fejezet

Felhasznált technológiák, csomagok

2.1. Laravel

Ahogy a bevezetőben említettem, a megvalósításhoz a Laravel nevezetű keretrendszert használtam. A keretrendszer sok funkciót alpból elkészít, amit hosszadalmas lenne újra és újra megírni, ezáltal a programozó a lényegre fókuszálhat: az elkészítendő programra.[5]

2.1.1. Keretrendszer alapjai

MVC architektúra

A keretrendszer az MVC architektúrát követi, ami segíti a kód karbantartását, áttekinthetőségét.[9] Ez azt jelenti, hogy logikailag a kód három része bomlik:

- Modellek (Model): ezek tartalmazzák az adatbázis táblák modelljeit, ezeken keresztül tudjuk elérni az adatbázisunkat, illetve az adatbázisban tárolt adatokat is modellek formájában kapjuk meg, így ezeknek követni kell a táblák felépítését. Ezek mellett kapcsolatokat is meg tudunk adni a modellek között.
- Nézet (View): itt magára a nézetekre kell gondolni, amit látni fog a felhasználó. Ezek jelen esetben a Blade[16] fájlok, amire a 2.1.4. alfejezetben térek ki.
- Kontroller (Controller): az üzleti logikát tartalmazza. Itt kapnak helyet a függvények például, amik egy kérés fogadásakor futnak le.

Útvonalak

A keretrendszerben az útvonalakat a *routes* mappában található fájlokban találjuk. Mivel a keretrendszer képes nem csak webes kéréseket kiszolgálni, így több fájlra vannak szedve, viszont én csak a webes útvonalakhoz tartozó fájlt használtam.

Útvonalakat a *Route* osztály metódusaival tudunk létrehozni. Ezek után meg kell adnunk a útvonal típusát (tehát hogy *Get*, *Post*, *Put* vagy *Delete* típusú útvonalat szeretnénk létrehozni), magát az elérési útvonalat, illetve a meghívandó programkódot, ami ki fogja szolgálni a kérést. Opcionálisan megadhatunk egy nevet is, amivel referálni tudunk rá a kódunk többi részéből, illetve adhatunk meg egy vagy több köztes program kódot (angolul: *middleware*), mellyel kontrollálhatjuk az útvonal elérést és egyéb más dolgokat tehetünk meg a kérés kiszolgálása előtt.[7] Erre láthatunk egy példát a 2.1 kódban.

Amennyiben az olvasó megnézi az elkészült programban található útvonalakat, azt láthatja, hogy csak *Get* típusú útvonalak találhatók benne. Ennek fő oka a Livewire[6] használata, ahol útvonalak létrehozása helyett metódushívásokkal történik sok interakció, erről a 2.3. alfejezetben lesz szó.

Middleware

A *middleware* arra alkalmas, hogy a kérés feldolgozása előtt végezzünk el teendőket, akár szűrjük őket.[8] A program során én három fő dologra használtam ezt fel: bejelentkezés vizsgálata, hozzáférési jogosultság vizsgálata, illetve a kérés elején az alkalmazás nyelvének beállítása. Az utolsóra azért volt szükség, mivel Laravel esetén a nyelv beállítása mindig csak az adott kérésre vonatkozik. Így minden esetben meg kellett vizsgálni, hogy a felhasználó átállította-e a nyelvet, és ha igen, akkor alkalmazzuk a kiválasztott nyelvet a kérésre. A *middleware* futása során dönthetünk úgy is, hogy a kérést megszakítjuk, például ha a felhasználó nem rendelkezik kellő jogosultsággal. Hozzá is tudjuk rendelni ezeket az egyes útvonalakhoz, ahogy az a 2.1. kódrészleten látható, vagy akár beállíthatjuk, hogy globálisan, minden kérés esetén lefusson az adott kód. A 2.8. kódrészletben egy *middleware*-ből látható egy részlet, ami adminisztrátorok esetén engedélyezi a kérést.

2.1. kódrészlet. Egy *Get* típusú útvonal, ami az órarendet nyitja meg. Az útvonal meghívásakor a *Controller* nevű osztály *getTimetable* metódusa fut le. Az útvonalat csak bejelentkezett felhasználók ('auth') és diákok vagy tanárok ('studentorteacher') érhetik el.

```
1 Route::middleware(['auth', 'studentorteacher'])->group(function () {  
2     Route::get('timetable', [Controller::class, 'getTimetable'])  
3         ->name('timetable');  
4 });
```

Konfiguráció

A *config* mappában található a konfigurációs fájlok. Ha jobban megnézzük, ezek igazából PHP fájlok, amik egy tömb-ként adják vissza a konfigurációs értékeket. Ezeket

módosíthatjuk, illetve mi is hozhatunk újakat létre. Én is így tettem: az olyan beállításokat, mint például, hogy engedélyezve van-e a regisztráció, vagy hogy a felhasználók módosíthatják-e a kódjukat, innen kérdezem le. Sok esetben az értékeket a környezeti konfigurációs fájlból olvassa be a program, ezáltal a gyakran változó értékek egy helyen módosíthatók anélkül, hogy keresni kéne őket, hogy hol vannak.

Nyelvesítés

Az alkalmazás két nyelvet támogat jelenleg: Angolt és Magyar, ezekhez készítettem el a fordítást. A nyelvesítést tartalmazó fájlokat a *lang* mappában találhatjuk, nyelvek szerint almappákra bontva. Az egyes kifejezések kulcs-érték párokként vannak eltárolva, ezen kulcs segítségével tudjuk elérni őket a `___('kulcs')` segédfüggvény segítségével, ami globálisan elérhető a Kontroller és a Nézet fájlokban is. A keretrendszer automatikusan a beállított nyelv alapján visszaadja a megfelelő fordítást. Ha esetleg hiányozna az egyik fordítás, akkor a konfigurációban alapértelmezettként megadott nyelvből próbálja kikeresni az értéket. Ha ott sem találja meg, akkor a paraméterként megadott kulcsot adja vissza.

2.1.2. Adatbázis és modellek

Az adatbázis felépítése a keretrendszer adta módon van megoldva. Ez azt jelenti, hogy minden, a Laravel által támogatott adatbázis disztribúcióval működni fog a felépített adatbázis séma. A táblák úgynevezett migrációk segítségével vannak leírva. Ez egyrészt segíti az adatbázis megosztását a fejlesztők között, hiszen nem kell külön adatbázis fájlokat küldözgetni egymás között, másrészt egyfajta verziókezelést tesz lehetővé: hogyha valamit módosítunk az adatbázison, akkor új migrációt hozunk létre. Amikor egy másik fél letölti a változtatásokat, és elindítja a migrációt, a keretrendszer felismeri melyik változtatásokat kell alkalmaznia, és melyek azok, amik már alkalmazva lettek.[11]

Migrációk

A migrációk a *database* mappán belül a *migrations* mappában találhatók. Egy fájl általában egy tábla leírását vagy módosítását tartalmazza a jobb átláthatóság érdekében. Minden migráció két eljárást tartalmaz: az *up()* eljárás az adatbázis migráció futtatásakor fut le, míg a *down()* az adatbázis törlése vagy teljes újraépítése esetén, tehát az előbbi eljárás ellentettjét végzi el.[11]

A táblákat a *Schema* osztály *create* metódusával tudjuk létrehozni. Ez két paramétert vár: a tábla nevét, illetve a tábla felépítését leíró függvényt. Ezen függvény belül írhatjuk le a tábla mezőit.

A 2.2. kódrészletben egy ilyen migrációt láthatunk, ahol egy kurzusok tárolására szolgáló táblát hozom létre. Itt több, különböző mező létrehozása megfigyelhető:

- Az egyik első lépés szokott lenni a tábla elsődleges kulcsának meghatározása. Éppen ezért a Laravel biztosít erre egy gyorsabb módszert: a második sorban látható `$table->id()` segítségével létrehozható egy `id` nevű, szám típusú, automatikusan növekvő mező.
- Természetesen sima mezőket is létrehozhatunk, ehhez a típusát, a nevét, és egyéb opciókat adhatunk meg, mint például az alapértelmezett értékét, vagy hogy a mező null-ra állítható-e. Ilyet például a negyedik vagy a nyolcadik sor.
- Tudunk idegen kulcsokat is definiálni, ahogy azt a hatodik és hetedik sorban láthatjuk. Ehhez a `foreign()` függvényt kell használni, megadni a mező nevét, illetve hogy melyik tábla melyik mezőjét referáljuk ezzel az idegenkulccsal. Megadható az is, hogy törlés vagy módosítás esetén mi történjen. A példán egyik esetben törlése kerül a rekord, más esetben null-ra állítódik a mező.
- Laravel esetében szokás két dátumot is adni a táblákhoz: egyet, hogy mikor jött létre az egyed, és egyet, hogy mikor módosították utoljára. Ezt könnyen meg lehet tenni a kilencedik sorban látható módon.

2.2. kódrészlet. A kurzusokat tároló tábla migrációja.

```

1 Schema::create('courses', function (Blueprint $table) {
2     $table->id();
3     $table->string('course_id');
4     $table->string('description')->nullable();
5     $table->string('subject_id');
6     $table->foreign('subject_id')->references('id')->on('subjects')->
        ↳ cascadeOnDelete();
7     $table->foreignId('term_id')->nullable()->constrained()->nullOnDelete
        ↳ ();
8     $table->integer('course_limit')->default(20);
9     $table->timestamps();
10 });

```

Lehetőségünk van már létező táblákat módosító migrációt is írni. Ilyenkor a `create` helyett a `table` függvényt kell használni. A `down()` függvény általában a tábla törlésére szolgáló kódsort tartalmazza.

Modellek

Az adatbázis modellek a `app` könyvtárban belül a `Models` mappában találhatók. Minden modellt egy osztállyal lehet leírni. Amennyiben követjük a Laravel elnevezési konvencióját[12], akkor nem szükséges megadnunk a modellhez kapcsolódó tábla nevét, egyébként igen.

A modellen belül megadhatunk több információt, mint a tábla neve, ha szükséges, illetve beállíthatjuk, hogy az elsődleges kulcs az automatikusan növekedjen-e vagy sem. Megadhatjuk azt is, hogy mely mezőket lehet kitölteni a *\$fillable* mező segítségével. A *\$hidden* mezővel megadhatjuk, hogy mely attribútumokat nem szeretnénk szerializálni (ilyen lehet például a jelszó), és azt is megadhatjuk, hogy ha egyes adatbázis mezőket szeretnénk átkonvertálni más típusúra (ilyen például ha a tárolt dátumot a keretrendszer által biztosított dátum objektummá szeretnénk alakítani). Az adatbázis mezőit ezeken felül nem kell megadnunk, azokat úgynevezett „magic method”-on[13] keresztül tudjuk elérni.

A másik nagyobb „kategória”, amit meg szoktak írni a modellek esetén, azok a kapcsolatok. A keretrendszer lehetőséget nyújt a szokásos kapcsolatok (egy az egyhez, egy a többhöz stb.) kialakítására függvények segítségével. A 2.3. kódrészletben egy a többhöz kapcsolatot lehet látni. A 2.4. kódrészletben pedig egy több a többhöz kapcsolatot, ami kettő, egy a többhöz kapcsolattal van feloldva, ahol a kapcsoló tábla a *course_students*. Érdemes megfigyelni, hogy nincsenek megadva a kapcsolatoknál az azokat megvalósító idegenkulcsok nevei. Ahogy az fentebb említettem, ha követjük az elnevezés konvenciót, akkor azokat itt sem kell megadni. Ez alól a több a többhöz kapcsolat esetén a kapcsolótábla neve jelent kivételt.

2.3. kódrészlet. Egy a többhöz kapcsolat: egy tantárgynak több kurzusa van.

```
1 public function Courses(): HasMany
2 {
3     return $this->hasMany(Course::class);
4 }
```

2.4. kódrészlet. Több a többhöz kapcsolat: egy kurzusnak több hallgatója van, egy hallgató több kurzusra van felíratkozva.

```
1 public function Students(): BelongsToMany
2 {
3     return $this->belongsToMany(User::class, 'course_students');
4 }
```

2.1.3. Eloquent ORM

Az ORM, avagy az Object Relational Mapping rendszerek abban nyújtanak segítséget, hogy összekössék az adatbázist a modelleinkkel, ezáltal a modelleket használhatjuk az adatbázissal való kommunikáció során SQL parancsok írása helyett.[14] A Laravel egy nagyszerű ORM rendszerrel készül, amit Eloquent-nek[15] hívnak. Segítségével bonyolult lekérdezéseket, frissítéseket, létrehozásokat tudunk elvégezni a modelleken keresztül. Bár a bonyolultabb lekérdezések megírása elsőre nehézséget jelentett, könnyen megtanulható és megszokható volt ez az ORM, és gyorsan meg is szerettem.

A 2.5 kódrészletben egy ilyen lekérdezést láthatunk: lekérdezem a tantárgyakat, illetve szűröm őket, ha adott meg szűrési feltételeket a felhasználó. Emellett látható az is, hogy a lapozás is keretrendszer szinten támogatott a *paginate* függvény segítségével. Ehhez meg kell adni, hogy hány elemet szeretnénk megjeleníteni egy oldalon, és opcionálisan egy nevet is adhatunk neki, ami akkor hasznos, ha több ilyen lapozó szakaszt is tartalmaz az oldal.

2.5. kódrészlet. Tantárgyak lekérdezése, szűrése ha adtak meg értéket.

```
1 $subjects = Subject::where(function ($query) {
2     if ($this->idSearch != '') {
3         $query->where('id', 'like', '%'.$this->idSearch.'%');
4     }
5     if ($this->nameSearch != '') {
6         $query->where('name', 'like', '%'.$this->nameSearch.'%');
7     }
8 }->paginate(10, 'pageName: 'subjectsPage'));
```

2.1.4. Blade sablon fájlok

A nézetek és komponensek létrehozására is megoldást nyújt a keretrendszer a Blade[16] fájlok formájában. Ezek szokványos HTML kódot tartalmaznak, illetve a Blade által nyújtott direktívákat, de akár egyszerű PHP kódot is tartalmazhat, mivel végső soron ezekből PHP fájlokat készít a keretrendszer.

A Blade különböző direktívák használatát teszi lehetővé, ami megkönnyíti a nézetek létrehozását. Ezek általában egy @ jellel kezdődnek. Segítségükkel feltételeket rakhatunk a HTML kódunkba, ciklusokat a kollekciók kiírására, de például gyors megoldást ad arra is, hogy az oldal egyes részeit csak bejelentkezett felhasználóknak jelenítsük meg a *@auth* direktíva segítségével, csak hogy párat említsek. Ilyen direktívák láthatók a 2.6. kódrészletben példaképpen.

2.6. kódrészlet. Bejelentkezett felhasználó esetén a jelenleg nem aktív félévek nevének kiírása.

```
1 @auth
2     @foreach ($terms as $term)
3         @if (! $term->active())
4             <h1>{{ $term->name }}</h1>
5         @endif
6     @endforeach
7 @endauth
```

Hasznos segítséget nyújt a kinézet felépítésében is: az egyes nézetek kibővíthetnek másik nézeteket, ezáltal könnyedén létrehozható egy egységes nézet, mivel a főbb formázásokat és részleteket (például a navigációs sávot) elegendő egyszer megírni, a többi

oldalunkat pedig ebből a fájlból kiterjeszteni.

Ezek mellett komponenseket is létrehozhatunk, amik újra és újra felhasználható kódokat tartalmaznak. Én ezt első sorban az ikonoknál használtam ki, a többi esetben egy speciális, Livewire komponenseket használtam, erről többet a 2.3. alfejezetben olvashatnak. Minden olyan ikon esetében, amit többször használtam, egy komponenszt hoztam létre neki, ami tartalmazta a ikont leíró SVG kódot, így azt csak be kellett hivatkozni, ahogy az a 2.7. kódrészletben látható. A komponens egy HTML tag-ként jelenik meg, illetve egy *x-* előtaggal rendelkezik.

2.7. kódrészlet. Komponens meghívása.

```
1 <button class="btn btn-success" wire:click="createJustification">
2     <x-icons.plus_fill_small/>{{ __('general.save')}}
3 </button>
```

2.1.5. Autentikáció

A modellek és az adatbázis elkészítése után az egyik első dologom volt az autentikáció kialakítása. Ez egy elég repetitív folyamat tud lenni, ezért ennek felgyorsítására a *Laravel Fortify*[17] nevű csomagot alkalmaztam.

Ezen csomag abban nyújt segítséget, hogy előre elkészíti az autentikációhoz szükséges útvonalakat, funkciókat, mint a bejelentkezés, regisztráció, jelszó visszaállítás. Magát a nézetek továbbra is nekünk kell létrehozni, illetve szükség esetén személyre szabhatjuk a különböző folyamatokat is. Gondolok itt arra, ha mondjuk nem email cím segítségével jelentkeznek be a felhasználók (például az én programomban beléphetnek email vagy Neptun kód segítségével is), vagy ha több mezőt kell megadni regisztrációkor, mint amit a Fortify alapértelmezetten felajánl.

Amennyiben egy egyszerűbb alkalmazást készítünk, akkor lehetőségünk van használni több olyan csomagot, amik telepítéstől kezdve egy félkész alkalmazást adnak, amiben sok általános funkció (bejelentkezés, felhasználókezelés stb.) alpból el vannak készítve. Bár csábítóan hangzik, véleményem szerint ezeket nem érdemes használni, mivel egy nagyon általános, nem túl egyedi élményt nyújtanak, amit nehézkes lehet személyre szabni. Emiatt döntöttem inkább a Fortify használata mellett, és alakítottam ki a én a nézeteket.

2.1.6. Jogosultság kezelés

Szerepkörök

Ahogy azt az 1.2.1. alfejezetben említettem, több szerepkört különböztetek meg a programban: szuper adminisztrátor, sima adminisztrátor, tanár és hallgató. Ezeknek a gyors létrehozására a *Laravel Ladder*[18] csomagot alkalmaztam. Ennek segítségével gyorsan

definiálhatók szerepkörök. A felhasználó modellt módosítja úgy a csomag, hogy gyorsan tudjuk ellenőrizni, hogy milyen szerepkör(ök)kel rendelkezik a felhasználó. Ezt elsősorban három helyen használtam: a middleware-ek esetén, hogy egy útvonal kinek elérhető, a nézetekben, hogy mely menüpontokat jelenítsem meg a felhasználónak, és a Policy-k esetén, erről bővebben lentebb.

2.8. kódrészlet. Admin middleware részlet: az ez által védett útvonalak csak adminoknak érhetők el.

```
1 public function handle(Request $request, Closure $next): Response
2 {
3     if (Auth::user()->hasRole('admin') || Auth::user()->hasRole('
4         ↳ superadmin')) {
5         return $next($request);
6     }
7     abort(403);
8 }
```

Jogosultság ellenőrzése

Ahogy azt fentebb említettem, a middleware csak az egyik hely, ahol a szerepköröket használtam. A Policy-k[19] segítségével egy helyre lehet gyűjteni, hogy a felhasználó egy adott modell esetén milyen jogokkal rendelkezik: megtekintheti, újat hozhat létre, esetleg módosíthatja vagy törölheti. Minden modellhez készíthető ilyen szabály, egy szóval itt lehet definiálni a felhasználó jogait. Ez nálam a legtöbb esetben úgy nézett ki, hogy adminisztrátor (vagy szuper adminisztrátor) esetén mindenhez joga van, a modellek megtekintéséhez be kell jelentkezni, többi esetben a szituációtól függ a dolog, például tantárgyakat csak adminisztrátorok tudnak létrehozni, kezelni, de bármely bejelentkezett felhasználó megtekintheti.

A 2.9. kódban egy részletet láthatunk a jogosultság vizsgálatra: a *before* függvény minden függvény előtt lefut, itt vizsgálom meg, hogy adminisztrátorról van szó. Ha nem, akkor megy tovább a vizsgálat. Megtekinteni mindenki megtudja, létrehozni pedig csak adminisztrátor tud, viszont ezt már az előbb említett függvény levizsgálja, így itt elegendő egy hamis értéket vissza adni. A 2.10. részletben pedig magát a vizsgálatot láthatjuk egy konkrét esetben: a felhasználó rendelkezik egy *can* és a fordítottja, egy *cannot* függvénnyel, ezekkel lehet eldönti, hogy van-e jogosultsága a felhasználónak az adott művelethez, vagy sem. Ha nincs, akkor megjeleníték egy hibaüzenetet, és visszalépek a függvényből.

2.9. kódrészlet. Részlet a tárgyakhoz tartó szabályból.

```
1 public function before()
2 {
3     if (Auth::user()->hasRole('superadmin') || Auth::user()->hasRole('
4         ↪ admin')) {
5         return true;
6     }
7     return null;
8 }
9 public function viewAny(User $user): bool
10 {
11     return true;
12 }
13
14 public function create(User $user): bool
15 {
16     return false;
17 }
```

2.10. kódrészlet. Jogosultság vizsgálata tárgy létrehozásakor.

```
1 if (Auth::user()->cannot('create', Subject::class)) {
2     toast()->danger(__('general.noPermission', __('general.error')))->
3         ↪ push();
4     return;
5 }
```

Ezen szabályok nagy előnye, hogy ha bármely okból kifolyólag módosítani kell a kialakított szabályrendszeren, akkor ezt csak egy helyen kell megtenni, és nem kell végig nézni a kódot, hogy hol kell módosítást végezni.

2.1.7. Email küldés

A programomban több helyen is használok email küldést értesítésék küldésére, például hiányzás rögzítésekor. Egy email küldéséhez egy értesítés osztályt kell létrehozni, ahol definiálni kell, hogy az értesítést email-ként kell elküldeni, és hogy mi legyen az üzenet tartalma. Ehhez a Laravel biztosít egy leírónyelvhez hasonló módszert, így nem szükséges minden fajta levél esetén leírni annak kinézetét, csak a tartalmát kell definiálni.[20] Az elkészül emailt a felhasználó modelljén keresztül tudjuk elküldeni, ahol akár a levél nyelvét is beállíthatjuk. Én például a modellben tárolt *lang* mező alapján tettem ezt, így a leveleket a beállított nyelvük alapján kapják meg a felhasználók.

A levelek egy sorra kerülnek rá, így azok a háttérben kerülnek elküldésre. Ezáltal elkerülhető, hogy a küldés során az oldal sokáig töltsön, vagy esetleg hibára fusson, ha

nem sikerült elküldeni a levelet.

2.11. kódrészlet. Egy példa az email felépítésére. A szövegek helyén a fordítás behelyettesítésére szolgáló függvények és kulcsok láthatók.

```
1 $mail = new MailMessage();
2
3 $mail->subject(__('teacher.absenceRecorded'))
4     ->line(__('teacher.absenceRecordedLine', ['name' => $this->user->name
5         ↳ ]))
6     ->line(__('teacher.subject').':_'. $this->class->Course->Subject->name
7         ↳ )
8     ->line(__('teacher.course').':_'. $this->class->Course->course_id)
9     ->line(__('general.startTime').':_'. $this->class->start_time)
10    ->line(__('general.endTime').':_'. $this->class->end_time)
11    ->line(__('teacher.recordedBy', ['name' => $this->recordingUser->name
12        ↳ ]));
13
14 return $mail;
```

2.1.8. Naptár exportálása

Az első fejezetben említettem, hogy az órarendet lehetőség van exportálni. Ha a felhasználó engedélyezi ezt, akkor egy egyedi azonosító kerül létrehozásra. Az ezt az azonosítót tartalmazó útvonalat szükséges megadni a naptár programban. Az importálás során, amikor meglátogatja ezt az útvonalat a naptár, a programom egy ICS kiterjesztésű fájlt hoz létre. Ebbe a fájlba menti ki a program a felhasználó naptárában található órákat a fájl típus által megadott struktúrában.[21] Mivel ez egy gyakori fájlformátum, így a legtöbb naptár program kezelni tudja ezt. A megadott linken keresztül periodikusan frissíteni is tudja a naptár magát, azáltal hogy megint lekérdezi ezt a fájlt a programtól.

2.12. kódrészlet. Részlet egy generált fájlból: egy esemény (jelen esetben óra) leírása.

```
1 BEGIN:VEVENT
2 UID:65ff156e83c3a
3 DTSTAMP:20240323T184622Z
4 DTSTART:20240508T090000Z
5 DTEND:20240508T104000Z
6 SUMMARY:Szoftverjog és biztonságtechnika - EA
7 LOCATION:E.C.I.124
8 END:VEVENT
```

2.2. Tailwind

Amikor a nézetek kialakítását elkezdtem, elgondolkoztam azon, hogy milyen segédeszközöket használhatnék. Szaktársaim ajánlására kipróbáltam a *Tailwind* nevezetű keretrendszert, ami gyorsan elnyerte a tetszésemet. A Tailwind egy úgynevezett „utility-first” keretrendszer.[22] Ez gyakorlatban azt jelenti, hogy CSS osztályok és szelektorok írása helyett a kívánt formázási szabályokat közvetlen az adott elemen adhatjuk meg, a Tailwind pedig létrehozza az ezekhez szükséges osztályokat, mely sok esetben 1-1 sorból állnak.

2.13. kódrészlet. Tailwind példa: az elemre flex megjelenítést alkalmaz, horizontális elrendezéssel, 4-es térközzel az elemek között.

```
1 <div class="flex flex-row gap-4">
2   <!-- Tartalom -->
3 </div>
```

Az online fórumokon több helyen is láttam, hogy kritika éri a Tailwind-et abból a szempontból, hogy pont a CSS fájlok lényegét veszi el, és ezáltal átláthatatlanabb HTML kód keletkezik. Bár valamilyen szinten egyet értek ezekkel a kritikákkal, a program készítése során úgy éreztem, hogy nagyban meggyorsította és könnyebbé tette a fejlesztést ezen keretrendszer használata, mivel nem kellett elhagynom a fájlt, amiben dolgoztam. Azt, hogy átláthatatlanabb lenne a fájl, kétféleképpen is ki lehet védeni: egyrészt érdemes komponensekre szedni az alkalmazást, másrészt egyéb, a Tailwind-be beépülő komponens könyvtárak használata is segíthet.

Itt emelném még ki, hogy a programomban használt ikonok a csakugyan a Tailwind készítőitől származó *Heroicons*[24] csomagból kerültek felhasználásra.

2.2.1. DaisyUI

Azért, hogy egységesen megjelenő oldalt készítsek, a *DaisyUI*[23] nevezetű komponens könyvtár használata mellett döntöttem. Ez a könyvtár kifejezetten a Tailwind-hez készült. A komponensek között megtalálható egyszerű gombok, bementi mezőktől kezdve kártyák, összeecsukható menükön át egészen felugró ablakokig sok fajta komponens. Ezek mellett a DaisyUI témák kialakítását is lehetővé teszi, ezért is rendelkezik az alkalmazásom sötét és fehér kinézettel. Ezeket a témákat személyre is lehet szabni. Mivel a front-end területén nem rendelkezem sok tapasztalattal, így sok segítséget nyújtott ez a könyvtár, hogy egy esztétikus oldalt alakítsak ki.

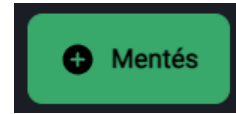
2.14. kódrészlet. DaisyUI: gomb létrehozása

```

1 <button class="btn btn-success">
2   <x-icons.plus_fill_small/>{{__('
      ↪ general.save')}}
3 </button>

```

2.1. ábra. A megjelenő gomb.

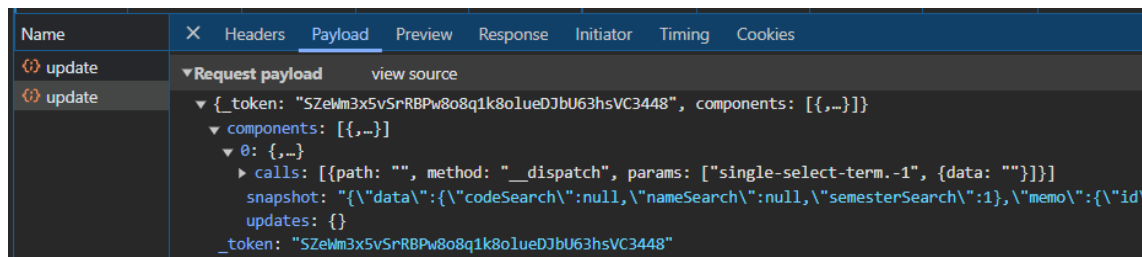


2.3. Livewire

A program megtervezésekor célom volt, hogy egy modernnek érződő alkalmazást készítek. Ezért esett egy rövidebb keresés után a Livewire-re a választásom, ami egy dinamikus oldalak létrehozására szolgáló keretrendszer, kifejezetten a Laravel-hez készítve.[6]

Nagy előnye a csomagnak, hogy nem kell kilépnünk a PHP nyelvből, hiszen a Livewire által létrehozott komponensek nagyon hasonlítanak a 2.1.4. fejezetben említett komponensekhez. Az így létrehozott komponensek dinamikusan viselkednek. Ez a felhasználók szemszögéből abban jelenik meg, hogy az oldallal való interakció esetén (gombnyomás, mező kitöltése stb) nem töltődik újra a teljes oldal, csak azon részei, amik változtak. Ezt a livewire úgy oldja meg, hogy ha olyan interakciót végzünk az oldalon, ami változtatna a komponens állapotán, akkor a háttérben egy kérést küld a szerver felé, ami tartalmazza a komponens (illetve a komponenshez tartozó PHP osztály) új állapotát, amit a szerver kiértékel, és az visszaküldi a komponens új állapotát. Ekkor a beérkezett adatok alapján megváltoztatja a DOM-ot¹ a Livewire, ezáltal kikerülve az oldal teljes újratöltését.

Így bár a háttérben továbbra is JavaScript alapú kérések történnek, a programozó továbbra is a PHP kényelmében maradhat a programozás során.

2.2. ábra. Livewire komponens frissítése: interakció esetén egy ilyen kérés kerül létrehozásra, ahol a *snapshot* tartalmazza a komponens állapotát, amit újra ki kell értékelni.

2.3.1. Komponensek és data binding

Ahogy ez fentebb említettem, a csomag használatához komponenseket kell létrehozni, melyek egy PHP osztályból, és az ahhoz tartozó nézetből állnak. Az osztályban létrehozott mezőkhöz és függvényekhez hozzá tudunk férni a nézetünkben, és akár hozzá

¹ Jelentése: Document Object Model, avagy Dokumentum Objektum Modell[25]

is tudjuk ezeket az értékeket kötni egyes beviteli mezőkhöz a *wire:model* segítségével. A függvények meghívásait pedig gombnyomásokhoz, vagy űrlapok elküldéséhez is köthetjük a *wire:click* és *wire:submit* használatával. Így amikor a felhasználó begépel valamit egy mezőbe, és/vagy megnyom egy gombot, a mögöttes PHP osztály állapota frissül, ez kerül elküldésre a 2.2. kódrészletben látható módon, amit kiértékel a szerver, és visszaküldi. Ezek a komponensek egyébként Blade fájlok egyben, így a 2.1.4. fejezetben írt direktívák továbbra is használhatók. Amennyiben azt szeretnénk, hogy minden gombnyomásra frissüljön az oldal – ez akkor hasznos például, ha az űrlap kitöltése során egyből szeretnénk frissíteni az oldalt esetleges hibaüzenetekkel –, akkor ezt is megtehetjük, ha a direktívát kiegészítjük a *live* kulcsszóval.

2.15. kódrészlet. A komponens osztályának egy mezője és egy bemeneti mező összekapcsolása, ami a harmadik sorban található.

```
1 <input type="datetime-local" placeholder="Type here"
2     class="input input-bordered input-accent w-full max-w-xs"
3     wire:model.live="end"/>
```

A programomban főként ilyen Livewire komponenseket használtam. Ez alól egyedül a bejelentkezés és a regisztráció képez kivételt, mivel azokhoz a 2.1.5. fejezetben írt csomagot használtam, és ott sima Blade fájlokkal kellett dolgozni.

2.3.2. Livewire és SPA

A fentebb említettek alapján sikerült kialakítani dinamikusan frissülő nézeteket, viszont egy probléma továbbra is fent állt: az oldalak közötti navigáció esetén továbbra is az egész oldal újra töltődött. Ennek megoldására született a Livewire *Navigate*[26] funkciója. Használata egyszerű: a navigálásra szánt gombokra, hivatkozásokra a *wire:navigate* direktívát kell alkalmazni. Ezután amikor rákattint a felhasználó, az oldal teljes újratöltése helyett a háttérben kéri le a weboldalt a szervertől, majd a kapott adatok alapján módosítja a dokumentum struktúráját a Livewire.

Ezáltal egy SPA típusú oldal hozható létre, melynek jelentése: Single Page Application, avagy Egy oldalból álló alkalmazás. Ahogy a neve is sugallja, ezek olyan alkalmazások, amik egy oldalt töltenek be teljesen, a későbbiek során ennek az oldalnak a felépítését módosítják háttérben történő kérésekkel, mint ami a 2.2. ábrán is látható.[28]

2.3.3. Események

A csomag lehetőséget ad események küldésére és fogadására is. Ezt én elsősorban kettő funkció megoldására használtam fel. Az egyik a legördülő kiválasztó mezők esetén volt. Ezek a mezők külön komponensek, így amikor annak az értéke frissül, akkor egy eseményt hozok létre, amit fogad az a komponens, amibe a mező be van ágyazva, ezáltal

módosítva az értéket. A másik ilyen a felugró ablakok bezárása: ha valamilyen modell létrehozására szolgáló felugró ablakot nyit meg a felhasználó, majd a mezők megfelelő kitöltése után a mentésre kattint a felhasználó, akkor – amennyiben nem lép közbe validációs probléma – a mentést megvalósító függvény egy eseményt küld, hogy záródjon be az ablak.

Ilyen eseményeket a komponensben belül a *dispatch()* függvény segítségével küldhetünk, ahol adni kell egy nevet az eseménynek, illetve igény szerint egyéb adatokat, amiket szeretnénk mellékelni az eseményhez.

Az eseményeket fogadni többféleképpen tudjuk. Például a legördülő mezők esetén közvetlen a komponens osztályában fogadtam az eseményt. Ehhez egy függvényt kellett írni, aminél az *On* attribútumot kellett alkalmazni. Amennyiben egyéb adatok is várunk az eseménynél, akkor azt a függvény paraméterein keresztül tudjuk megkapni. Ezt felhasználva, amikor változott a legördülő mező értéke, akkor egy eseményben kiküldtem a szülő komponensnek az új értéket, azt ő fogadta, és eltárolta azt.

Eseményt nem csak a PHP kódban, hanem a nézetekben, JavaScript-ben belül is tudunk fogadni. Ez a felugró ablakok bezárásánál volt előnyös, hiszen a nézetből férék hozzá az egyes HTML elemekhez az oldalon. Ilyenkor a JavaScript kódunkban a *Livewire.on* függvényt kell használni, megadni az esemény nevét, illetve hogy mi történjen az esemény bekövetkezésekor, tehát jelen példában záródjon be az ablak.

2.16. kódrészlet. Események küldése és fogadása.

```
1 // Esemény küldése adatokkal:
2
3 $this->dispatch('multiple-select-students', data: array_column($this->
    ↪ selected_items, 'id'));
4
5 // Esemény fogadása komponensen belül:
6
7 #[On('single-select-teacher')]
8 public function setManager($data)
9 {
10     $this->subjectManager = $data;
11 }
12
13 // Események fogadása JavaScripten belül:
14
15 <script>
16     Livewire.on('closeSubjectCreateModal', () => {
17         subjectCreateModal.close();
18     })
19 </script>
```

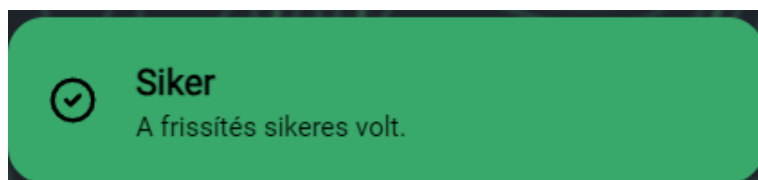
2.3.4. WireToast: Livewire alapú értesítések

Az utolsó Livewire-el kapcsolat téma, amit szeretnék megemlíteni, az az oldalon megjelenő értesítések egyes interakciók után, mint például létrehozás. Ehhez a *Tall toasts*[29] nevű csomagot használtam. Ez a Livewire-rel együtt működő csomag, így csak ilyen komponenseken belül lehet használni. Működéséhez a komponensek használnia kell a *WireToasts* nevezetű trait-et, majd pedig a *toast()* függvény meghívásával hozható létre ilyen értesítés. Ezután a típusát kell megadni (információ, siker, esetleg hiba), illetve a megjelenítendő szöveget és címet. Majd a *push()* függvénnyel tudjuk kiküldeni az értesítést.

2.17. kódrészlet. Oldalon megjelenő sikeres értesítés küldése.

```
1 toast()->success(__('general.subjectCreated'), __('general.success'))->  
  ↪ push();
```

Az értesítések kinézete természetesen személyre szabható, én például a DaisyUi által szolgáltatott értesítés komponensre írtam át az eredeti nézeteket.



2.3. ábra. A létrejövő értesítés kinézete.

2.4. Websocket és Pusher

Amikor az 1.5. képen látható oldalt készítettem, akkor mindenképpen szerettem volna megoldani, hogy amikor a tanuló a Qr kód segítségével bejelentkezik az órára, a státusza automatikusan frissüljön a tanár által látott oldalon is. Első gondolatom az volt, hogy periodikusan, pár másodpercenként frissítem az oldalt, viszont ezt az ötletet hamar elvetettem, mivel ez rengeteg fölösleges kéréssel járna, ami sok töltést jelentene a felhasználóknak, illetve terhet a szervernek. Így másik megoldás után néztem, ekkor találtam rá a Websocket-ekre és a Pusher szolgáltatásra.

2.4.1. Websocket

A Websocket technológia kétirányú kommunikációt tesz lehetővé a szerver és a kliensek között. Ezáltal a folytonos frissítést, amit először terveztem, eseményekkel lehet kiváltani.[30]Amikor valaki bejelentkezik az órára, akkor egy esemény jön létre, ezt a tanári oldal fogadja, és innen tudja, hogy frissíteni kell az adott tanulóhoz tartozó sort. Ezek nem azok az események, amiket a 2.3.3. fejezetben taglaltam. Azok az események

az adott felhasználói munkamenetre korlátozódnak, míg ezek az események küldése, fogadása eszközök, felhasználók között történnek.

2.4.2. Pusher channels

A Pusher Channels[31] tulajdonképpen egy megvalósítása a Websocketeknek, amit kifejezetten webalkalmazásokhoz terveztek. Ingyenes fiókot lehet hozzá létrehozni, ami bár korlátozott, a szakdolgozat készítéséhez és kipróbálásához bőven elegendő. A Laravel keretrendszer is támogatja a Pusher technológiát, így könnyen integrálható volt a programba. Mondhatni a Pusher a szerver és a kliensek között áll: ide érkeznek be az események, és innen kerülnek kiküldésre.

Természetesen más, akár nyílt forrású megoldások is elérhetők, illetve a dolgozat készítésekor is fejlesztés alatt volt egy kifejezetten a Laravel-hez készülő változat, viszont a fejlesztők később lettek készek vele ahhoz, hogy fel tudjam használni szakdolgozatomban, így a Pusher lecserélése erre a megoldásra egy jövőbeli feladat lesz.

2.4.3. Csatornák, események létrehozása

A Websocketek konfigurálása és használata két részre tehető: a szerver és a kliens beállítása.

2.4.4. Események fogadása JavaScript, LiveWire esetén

2.5. QR-kód generálás

2.6. Naptár - FullCalendar.js

2.6.1. Alpine.js: PHP kód eredményének lekérése JavaScript-ben

2.6.2. Beállításai

2.6.3. Kapcsolat a keretrendszerrel LiveWire segítségével

2.7. Diagramok - Chart.js

2.7.1. Beállításai

2.7.2. Kapcsolat a keretrendszerrel LiveWire segítségével

2.8. Progressive Web Apps

2.8.1. A PWA jelentése, jelentősége

2.8.2. Laravel PWA csomag

2.8.3. A manifest fájl

2.8.4. Service Worker

3. fejezet

Az alkalmazás tesztelése

3.1. Manuális tesztelés

3.2. Automatizált tesztelés

3.3. Terheléses tesztelés

3.4. Laravel Pint és Github actions

4. fejezet

Alkalmazás telepítése

Az alkalmazás a szakdolgozat védés, illetve a záróvizsga időszak alatt elérhető a `https://szakdolgozat.ddns.net` címen, ahol ki lehet próbálni az alkalmazást. Pár alapértelmezett felhasználó elérhető:

- Szuper adminisztrátor:
 - Felhasználónév: *SADMIN*
 - Jelszó: *superadmin*
- Adminisztrátor:
 - Felhasználónév: *ADMIN0*
 - Jelszó: *admin*
- Tanár:
 - Felhasználónév: *TEACHE*
 - Jelszó: *teacher*
- Hallgató:
 - Felhasználónév: *STUDEN*
 - Jelszó: *student*

4.1. Laravel Forge

4.2. Kézi telepítés lépései

1. Töltsük le az alkalmazás fájljait.

2. A futtatáshoz szükségünk van egy adatbázis szerverre. Javasolt MySQL vagy PostgreSQL használata. Ezeknek a telepítéséről az adott adatbázisszerver web-oldalán lehet tájékozódni.
3. Telepítsük a PHP-t számítógépünkre, legalább a 8.1-es verziót, melyet a Laravel keretrendszer 10-es verziója követel meg. Alternatívaként telepíthetjük a XAMPP nevezetű programot is, mely feltelepít egy adatbázis és PHP disztribúciót is.
4. Konfiguráljuk az adatbázist: hozzunk létre egy felhasználót és egy sémát, amihez hozz fog tudni férni az oldalunk.
5. Az alábbi php kiegészítő csomagok engedélyezése szükséges: Ctype, cURL, DOM, Fileinfo, Filter, Hash, Mbstring, OpenSSL, PCRE, PDO, Session, Tokenizer, XML.
6. Telepítsük a Composer nevű PHP csomagkezelő rendszert.
7. Telepítsük a Node.js nevű JavaScript futtatókörnyezetet.
8. A sikeres telepítések után navigáljunk el a *thesisproject* nevű mappába a letöltött fájlok között. Ez tartalmazza a program fájljait, a továbbiakban itt dolgozunk, itt adunk ki parancsokat.
9. Készítsünk egy másolatot a *.env.example* nevű fájlról és nevezzük azt át *.env* névre. Nyissuk meg, és töltsük ki az alábbi adatokat:
 - APP_URL: amennyiben nem helyi környezetben, kipróbálásra telepítjük, akkor itt adjuk meg az alkalmazás elérési útját.
 - DB prefixummal kezdődő értékek: itt állítsuk be az adatbázis kapcsolat adatait.
 - BROADCAST_DRIVER: értékét állítsuk át *pusher*-re.
 - QUEUE_CONNECTION: értékét állítsuk át *database*-re, amennyiben az adatbázist szeretnénk használni a sor-hoz.
 - MAIL prefixummal kezdődő értékek: itt állíthatjuk be a levelezéshez használt hozzáférési értékeket. Ezekről az email szolgáltatótól kaphatunk információt.
 - PUSHER prefixummal kezdődő értékek: itt a Pusher hozzáférési adatok beállítása szükséges. Ezekhez fiókot kell regisztrálnunk, miután elérhetővé válnak a szükséges értékek.

A többi értéket az alkalmazáson belül tudjuk módosítani.

10. Ezek után a csomagok telepítése szükséges. Adjuk ki a terminálban a **composer install --optimize-autoloader --no-dev** parancsot.
11. Majd pedig a **npm ci** és az **npm run build** parancsokat.
12. Ezek után az alábbi parancsok kiadása szükséges:
 - **php artisan key:generate**: alkalmazás kulcs generálása.
 - **php artisan migrate --seed**: adatbázis felépítése.
 - **php artisan route:cache**: útvonalak gyorsítótárazása.
 - **php artisan config:cache**: konfiguráció gyorsítótárazása.
 - **php artisan storage:link**: a képek tárolására használt könyvtár elérhetővé tétele.
 - A keretrendszerben jártas olvasó észreveheti, hogy kihagytam a nézetek gyorsítótárazására szolgáló parancsot. A Livewire általam használt verziójában van egy olyan probléma, miszerint gyorsítótárazás esetén az egyes elemek nem működnek megfelelően. Ennek kiküszöbölésére döntöttem úgy, hogy a nézeteket nem gyorsítótárazom.
13. A fejlesztői szerver elindítható a **php artisan serve** paranccsal. Ugyan csak szükséges a sor elindítása is egy másik terminál ablakban: **php artisan queue:work**.
14. Az alkalmazást megnyithatjuk a böngészőben, az alapértelmezett **SADMIN/superadmin** kombinációval tudunk belépni.

Összegzés

Irodalomjegyzék

- [1] SDA INFORMATIKA ZRT: *Neptun alkalmazás leírása*
<https://sdainformatika.hu/termekek>, Megtekintés dátuma: 2024.03.21.
- [2] FILEINFO.COM: *ICS fájl leírása.*
<https://fileinfo.com/extension/ics>, Megtekintés dátuma: 2024.03.21.
- [3] MDN WEB DOCS: *Progressive web apps*
https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps,
Megtekintés dátuma: 2024.03.21.
- [4] KASPERSKY: *QR Code Security: What are QR codes and are they safe to use?*
<https://www.kaspersky.com/resource-center/definitions/what-is-a-q-r-code-how-to-scan>, Megtekintés dátuma: 2024.03.21.
- [5] LARAVEL: *Meet Laravel*
<https://laravel.com/docs/11.x#meet-laravel>, Megtekintés dátuma:
2024.03.24.
- [6] LIVEWIRE: *Főoldal*
<https://livewire.laravel.com/>, Megtekintés dátuma: 2024.03.24.
- [7] LIVEWIRE: *Routing*
<https://laravel.com/docs/10.x/routing>, Megtekintés dátuma: 2024.03.24.
- [8] LIVEWIRE: *Middlewares*
<https://laravel.com/docs/10.x/middleware>, Megtekintés dátuma:
2024.03.24.
- [9] MDN WEB DOCS: *MVC*
<https://developer.mozilla.org/en-US/docs/Glossary/MVC>, Megtekintés
dátuma: 2024.03.24.
- [10] LARAVEL: *Blade templates*
<https://laravel.com/docs/10.x/blade>, Megtekintés dátuma: 2024.03.24.

- [11] LARAVEL: *Database: Migrations*
<https://laravel.com/docs/10.x/migrations>, Megtekintés dátuma: 2024.03.24.
- [12] LARAVEL: *Eloquent: Table Names*
<https://laravel.com/docs/10.x/eloquent#table-names>, Megtekintés dátuma: 2024.03.24.
- [13] PHP: *Magic Methods*
<https://www.php.net/manual/en/language.oop5.magic.php>, Megtekintés dátuma: 2024.03.24.
- [14] FREECODECAMP: *What is an ORM – The Meaning of Object Relational Mapping Database Tools*
<https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/>, Megtekintés dátuma: 2024.03.25.
- [15] LARAVEL: *Eloquent*
<https://laravel.com/docs/10.x/eloquent>, Megtekintés dátuma: 2024.03.25.
- [16] LARAVEL: *Blade Templates*
<https://laravel.com/docs/10.x/blade>, Megtekintés dátuma: 2024.03.25.
- [17] LARAVEL: *Laravel Fortify*
<https://laravel.com/docs/10.x/fortify>, Megtekintés dátuma: 2024.03.25.
- [18] GITHUB: *Ladder*
<https://github.com/eneadm/ladder>, Megtekintés dátuma: 2024.03.25.
- [19] LARAVEL: *Creating Policies*
<https://laravel.com/docs/10.x/authorization#creating-policies>, Megtekintés dátuma: 2024.03.25.
- [20] LARAVEL: *Mail Notifications*
<https://laravel.com/docs/10.x/notifications#mail-notifications>, Megtekintés dátuma: 2024.03.25.
- [21] WEBDAVSYSTEM: *Calendar (.ics) File Structure*
https://www.webdavsystem.com/server/creating_caldav_carddav/calendar_ics_file_structure/, Megtekintés dátuma: 2024.03.25.
- [22] TAILWIND: *Főoldal*
<https://tailwindcss.com/>, Megtekintés dátuma: 2024.03.25.

- [23] DAISYUI: *Főoldal*
<https://daisyui.com/>, Megtekintés dátuma: 2024.03.25.
- [24] HEROICONS: *Főoldal*
<https://heroicons.com/>, Megtekintés dátuma: 2024.03.25.
- [25] MDN WEB DOCS: *Introduction to the DOM*
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction, Megtekintés dátuma: 2024.03.26.
- [26] LIVEWIRE: *Navigate*
<https://livewire.laravel.com/docs/navigate>, Megtekintés dátuma: 2024.03.26.
- [27] MDN WEB DOCS: *SPA (Single-page application)*
<https://developer.mozilla.org/en-US/docs/Glossary/SPA>, Megtekintés dátuma: 2024.03.26.
- [28] LIVEWIRE: *Events*
<https://livewire.laravel.com/docs/events>, Megtekintés dátuma: 2024.03.26.
- [29] GITHUB: USERNOTNULL: *Tall toasts*
<https://github.com/usernotnull/tall-toasts>, Megtekintés dátuma: 2024.03.26.
- [30] MDN WEB DOCS: *The WebSocket API (WebSockets)*
https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API, Megtekintés dátuma: 2024.03.26.
- [31] PUSHER: *Pusher channels*
<https://pusher.com/channels/>, Megtekintés dátuma: 2024.03.26.