győri szakképzési centrum

Jedlik Ányos
Gépipari és Informatikai
Technikum és Kollégium

9021 Győr, Szent István út 7.
+36 (96) 529-480
+36 (96) 529-448

OM: 203037/003
jedlik@jedlik.eu
www.jedlik.eu

Záródolgozat feladatkiírás

Tanuló(k) neve[1]:	Blasek Balázs, Luksa Laura, Venter Alex
Képzés:	nappali munkarend
Szak:	54 213 05 Szoftverfejlesztő
	(2016-tól érvényes kerettanterv)

A záródolgozat címe:

House of Swords

Konzulens:	Horváth Norbert
Beadási határidő:	2023. 04. 14.

Győr, 2023. 10. 03

Módos Gábor
igazgató

[1] Szakmajegyzékes záródolgozat esetében több szerzője is lehet a dokumentumnak, OKJ-s záródolgozatnál egyetlen személy ad le záródolgozatot.

győri szakképzési centrum

Jedlik Ányos
Gépipari és Informatikai
Technikum és Kollégium

9021 Győr, Szent István út 7.
+36 (96) 529-480
+36 (96) 529-448

OM: 203037/003
jedlik@jedlik.eu
www.jedlik.eu

Konzultációs lap

	A konzultáció		Konzulens aláírása
	ideje	témája	
1.	2022.10.28.	Témaválasztás és specifikáció	
2.	2023.02.24.	Záródolgozat készültségi fokának értékelése	
3.	2023.04.14.	Dokumentáció véglegesítése	

Értékelés

A záródolgozat százalékos értékelése: ……

Legalább 51%-ot elérő előzetes értékelés és három igazolt konzultáció esetén a záródolgozat megfelelt.

Győr, 2023. április 14.

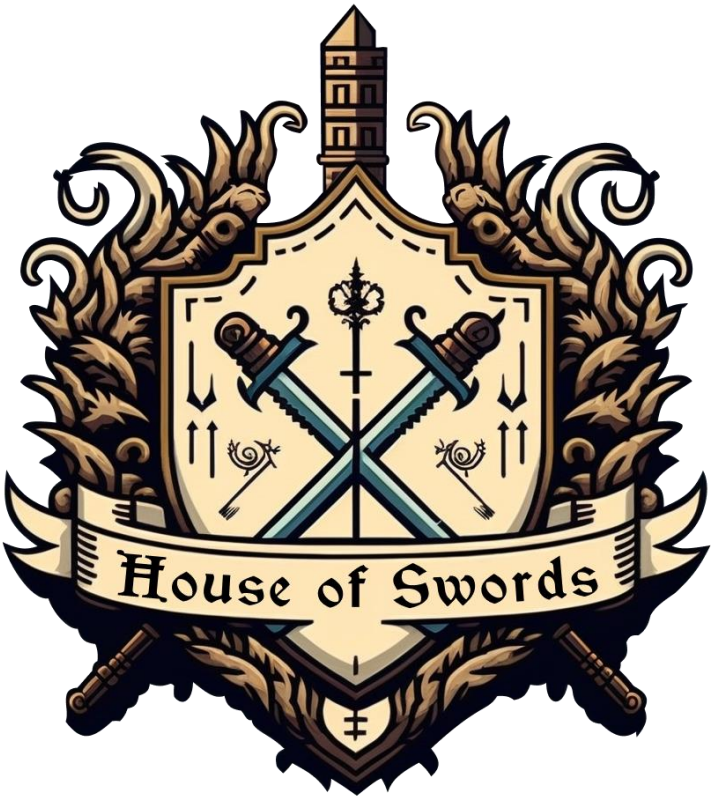
értékelő aláírása

Tulajdonosi nyilatkozat

Ez a dolgozat a saját munkánk eredménye. Dolgozatunk azon részeit, melyeket más szerzők munkájából vettünk át, egyértelműen megjelöltük.
Ha kiderülne, hogy ez a nyilatkozat valótlan, tudomásul vesszük, hogy a szakmai vizsgabizottság a szakmai vizsgáról kizár és szakmai vizsgát csak új záródolgozat készítése után tehetünk.

Győr, 2023. április 14.

Blasek Balázs | Luksa Laura | Venter Alex



A House of Swords egy izgalmas középkori világban játszódó stratégiai játék, ahol a játékosoknak taktikusan kell irányítaniuk a hadseregüket és ügyesen kezelniük erőforrásaikat a győzelem érdekében..

Taralomjegyzék

1. Specifikáció 6
1.1. Játék leírása 6
1.2. Frontend megoldás 6
1.3. Backend megoldás 6
1.4. Mobil megvalósítás 6
1.5. Weblap 6
2. Csapatmunka 6
2.1. Tervek 6
2.2. Projekt alatt 7
3. Használati útmutató 8
3.1. Hogyan tudunk játszani a játékkal? 8
3.2. Regisztráció 8
3.3. Profil oldal 8
3.4. A játék letöltése, belépés 9
3.5. Játékmenet 10
3.6. Mozgás a városban 11
3.7. Kilépés 11
3.8. Játék célja 11
4. Regisztráció és belépés - backend 11
4.1. Regisztráció 11
4.2. Bejelentkezés 12
4.3. CSRF védelem 12
4.4. Felhasználó jogosultság 13
5. Adatbázis 13
5.1. Adatbázis fontossága 13
5.2. Hogyan jön létre az adatbázis? 13
5.3. Mi a helyzet a központi szerveren lévő adatbázissal? 14
5.4. Hogyan lehet megtekinteni, illetve módosítani az adatbázist és a 14
5.5. Milyen táblák, modellek és kapcsolatok vannak az adatbázisban? 14
6. Sablonok használata 16
6.1. Prefabok 16
6.2. Layouts szerkezet 17
7.1. Domain beállítása 18
7.2. Szerver beállítása 18
7.3. GitHub használata 19
8. Email szerver 20

8.1. Szerver beállítása 20
8.2. Laravel kapcsolódása a szerverhez 22
9. Kriptográfia és biztonság 22
9.1. Titkosítás 22
9.2. Salt & Pepper 23
10. Frontend API hívások 24
10.1. Hálózati kommunikáció 24
10.2. Bejelentkezés API kéréssel 25
10.3. API titkosítás tokennel 26
11. Adatbázis automatikus frissítése 27
11.1. Tervezések 27
11.2. Projektbe ágyazás 27
12. Pathfinding 28
13. Tesztelés 29
13.1. API tesztelés 29
14. Források 30

1. Specifikáció

1.1. Játék leírása

A játék témája: Egy középkori, többjátékos, stratégiai, idle farming játék. A játékos egy középkori város irányítója és célja minél magasabb szintre megerősíteni a városát (katonailag, tudományilag, társadalmilag).

A játék egy képzeletbeli világban játszódik, ahol az összes játékos városa található. A játékosok tudnak egymással interakcióba lépni (támadás, fosztogatás, kereskedelem, technológia-lopás, szövetség).

1.2. Frontend megoldás

A projekt "Frontend" része maga a Unity-s játék alkalmazás (C# programozás), illetve a szükséges adatok szerverről való lekérésének megvalósítása a kliens részére.

1.3. Backend megoldás

A Backend része a szerveren az adatbázis kezelése, illetve a kérések kiszolgálásának megvalósításából állna. Az adatbázist *MySQL*-lel képzeltük el egy szerveren.

A játék nem valós idejű lenne, vagyis a játékosok egymás akcióit nem közvetlenül a végrehajtások után láthatnák. Minden akció az adatbázist frissítené, amit a kliensek később lekérhetnének, de ez a megoldás nem lenne elegendő egy valós idejű kapcsolat megvalósításához, ráadásul egy ilyen játéknál nem feltétlenül lenne rá igény.

1.4. Mobil megvalósítás

A Unity motornak köszönhetően a játék mobil eszközökre is kompatibilis lenne, hasonló felhasználói élményt nyújtva, az asztali eszközökre fejlesztett játéktól annyi eltéréssel, hogy nem *.exe,* hanem *.apk* lenne a kiterjesztése.

1.5. Weblap

A játékunkat kísérné egy weblap is, amely mind mobil, mind asztali eszközökön használható lenne. Ezen az oldalon lehetne egy bemutatót találni a játékról, regisztrálni a rendszerbe, letölteni a játékot, valamint a felhasználói adatokat módosítani/törölni.

2. Csapatmunka

2.1. Tervek

A projekt kezdetén felmerült a munkamegosztás kérdése. Mindhárman szeretnénk kivenni a részünket minden területen, azonban fontosnak tartottuk, hogy részekre

bontsuk a feladatot. Mindannyian elvállaltunk egy egységet. Természetesen segítünk egymásnak mindenben, de így minden feladatrésznek van vezetője, aki felelősséget vállal a rész időben való megvalósításáért, minőségéért.

Alex legfőképpen a unity-vel szeretne foglalkozni, mivel korábbi projektjei alatt sok tapasztalatot szerzett benne.

Az adatbázist Balázs vállalta el. Ennek az oka, hogy mindig is érdekelte a hálózatok világa.

Laurát a weblap fejlesztés és a GUI design érdekli leginkább. A grafikai mindig is meghatározó része volt az életének, és ezt a programozással ötvözni szeretné későbbi tanulmányai során is.

2.2. Projekt alatt

Most, hogy nagyjából a projekt közepén tartunk, sok tapasztalatot szereztünk a közös munkák során. Szerencsésen tapasztaljuk, hogy a csapatmunka megfelelően működik. Viszont volt néhány dolog, amire nem gondoltunk a tervezésnél.

Alapvetően felosztottuk három egységre a munkát de mindegyik még mindig nagy és komplex volt. Ezt a következő módon orvosoltuk. Kisebb feladatokat, célokat tűztünk ki magunknak, hogy ezt könnyen nyomon követhessük a „*trello*” című weboldalt használtuk. Ez jelentősen megkönnyítette a munkát. Itt öt táblára osztottuk a projektet (általános, adatbázis, játék, weblap, dokumentáció).

Hogy párhuzamosan tudjunk dolgozni fontos volt a branchek használata. A githubot használtuk a weblap forráskódjának tárolására. A Unity-hez *Plastic SCM-t, (*ami egy többplatformos kereskedelmi elosztott verzióvezérlő eszköz) használtuk. Mind a kettővel könnyen és átláthatóan tudtuk kezelni a brancheinket.

Tapasztaltuk magunkon, hogy amikor elakadtunk valamiben több időre, nagyon jót tett egy kis környezet változás képpen a Unity-ről Laravel-re és fordítva válltani, így újult erővel tudtunk nekilátni a munkának.

A kommunikációhoz *Discord*-ot használtunk. Mindannyiunk számára már korábban is ismert volt a program, így nem okozott fennakadást, tudtunk a feladatainkra koncentrálni. *Bot*-ok segítségével megoldottuk, hogy minden egyes *github commit*-ot és *Unity*-ban történő *chekin*-t egy *text channel*-be kiírjunk, így értesítést kaptunk róla, ezért ezzel pozitívan hatottunk egymásra mivel láttuk, hogy valaki épp dolgozik.

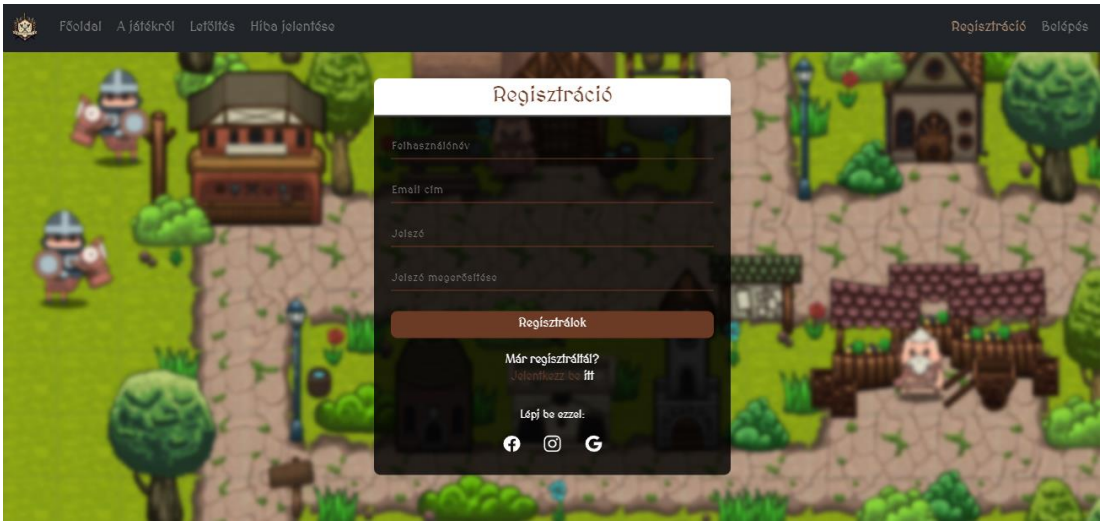
3. Használati útmutató

3.1. Hogyan tudunk játszani a játékkal?

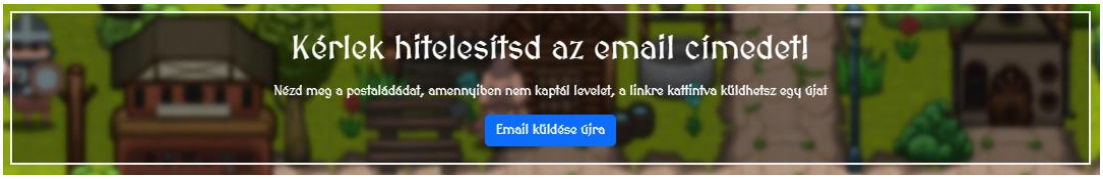
A játékszoftver használata hasonlít korunk játékainak felhasználásához. Ahhoz, hogy játszani tudjunk, először regisztrálnunk kell egy fiókot az oldalon.

3.2. Regisztráció

Szabadon választott böngészőben meg kell nyitnunk a https://houseofswords.hu weboldalt. A navigációs sáv jobb oldalán válasszuk a „*Regisztráció*” lehetőséget.



Miután helyesen megadtuk az adatainkat, a szervertől egy *email üzenetet fogunk kapni* a megadott email címünkre. Ellenőrizzük a postaládánkat, és kattintsunk az üzenetben kapott hivatkozásra, hogy *megerősítsük az email címünk hitelességét*. Amíg ezt nem tesszük meg, az oldal többi szolgáltatását nem tudjuk igénybe venni. Ha az email nem érkezett meg, vagy elveszett a többi levél között, a kék gombra kattintva újra tudjuk kérni annak elküldését.



3.3. Profil oldal

Miután ezt megtettük, a weboldalon elérhetővé válik számunkra a *Profil* oldal. Itt frissíteni tudjuk a felhasználói adatainkat, felhasználónevet és jelszót válthatunk, valamint akár profilképet tölthetünk fel.

Admin, valamint *tulajdonos* jogosultsággal rendelkező felhasználóknak a profil oldal mellett megjelenik az admin felület, ahol karbantartásokat, javításokat lehet végezni a rendszerben.

3.4. A játék letöltése, belépés

Miután regisztráltunk, menjünk át a *Letöltés* oldalrészre (https://houseofswords.hu/download). Ezen az oldalon a játékszoftvert letölthetjük Windows 10, illetve Android operációs rendszerrel működő készülékekre.

Miután letöltöttük, és telepítettük a játékot a készülékünkre, megnyitás után a *bejelentkezés felület* fogad minket.

Miután bejelentkeztünk, kiválaszthatjuk, hogy melyik városunkkal szeretnénk játszani.

3.5. Játékmenet

Egy rövid töltés után a városunk megjelenik. A felső sávban láthatjuk a városunkban rendelkezésre álló nyersanyagokat, illetve a lakosok boldogság-szintjét a város neve alatt.

Egy épületre kattintás után megjelenik az épület interakciós ablaka. Például az alábbi képen látható templom ablaka. Az itt lévő gombbal például egy istentisztelet indítható, ami növeli a város boldogságát.

3.6. Mozgás a városban

A *"W", "A", "S", "D"* billentyűkkel, avagy a *bal klikk* nyomva tartásával mozgathatjuk a kamerát a város körül.

3.7. Kilépés

Az "*Escape*" billentyű lenyomására megjelenik az oldalsó menü a játékban, ahol a játék beállításai módosíthatóak, ki lehet lépni a menübe, illetve a teljes játékból.

3.8. Játék célja

A játék célja, hogy a többi játékos által irányított városok közül a legerősebbek legyünk -- ezt több értelemben is lehet venni, például lehet célja valakinek, hogy nagy hadserege legyen, míg más a szövetkezést preferálja, és hatalmas közösségeket alakíthatnak, játékstílustól függ.

4. Regisztráció és belépés - backend

4.1. Regisztráció

Az űrlap az elküldése után a *CreationRequest* osztály segítségével ellenőrizve vannak az adatok. Az alábbi validálási szabályokat használjuk:

```
public function rules()
    {
        return [
            'Username' => 'required|unique:users|min:6|max:20|alpha_dash',
            'EmailAddress' => 'required|unique:users|email',
            'PwdHash' => 'required|min:8|max:24|confirmed',
            'PwdHash_confirmation' => 'required',
            'Role' => 'integer|min:0|max:2'
        ];
    }
```

Amennyiben a beírt adatszerkezetben hiba található, azt jelezzük a felhasználónak. Minden mező esetében a következő módszerrel jártunk el:

```
@if ($errors->has('Username'))
    <span class="text-danger text-left">{{ $errors->first('Username') }}
</span>
@endif
```

A *$errors* változót a Illuminate\View\Middleware\ShareErrorsFromSession middleware teszi elérhetővé az összes *.blade.php* fájlban.

Amennyiben minden adat megfelelő, létrejön egy új felhasználó és automatikusan bejelentkeztetjük, ezzel is felhasználó barátabbá téve az oldalt.

4.2. Bejelentkezés

A bejelentkezés menüpontban a felhasználónak meg kell adnia a *felhasználónevét* és beállított *jelszavát*.

Az esetlegesen *elfelejtett jelszó* esetében egy jelszó visszaállítási link is be van ágyazva, melynek segítségével a *hitelesített email* címre a felhasználó kap egy levelet, amelyben található linkre kattintva új jelszót tud létrehozni.

A bejelentkezésnél a program megkeresi, hogy létezik-e az adott nevű felhasználó a mentett adatbázisban és a jelszó titkosítása után keres egyezést a jelszó mezőnél. Hiba esetén természetesen értesítjük a felhasználót, hogy a felhasználónév vagy a jelszó helytelen. Ha a program talált az adatbázisban egyezést az adott felhasználóra akkor ellenőrzi, hogy *hitelesítve* van-e a felhasználó *email címe*. Ha igen, tovább engedi őt a profil oldalra, ahol a felhasználó tudja kezelni személyes adatait (pl.: jelszó, felhasználónév, profilkép szerkesztése).

Abban az esetben, ha a felhasználónak *nincs megerősítve* az email címe, nem engedi tovább a program, megragad egy köztes oldalon, ami kéri, hogy az emailjei közt keresse meg a kapott emailt és az abban található linkre kattintva hitelesítse email címét. Amennyiben a felhasználó nem találja azt a levelet, tud önmagának új levelet küldeni egy gombra kattintva.

4.3. CSRF védelem

Az oldalon használt összes `<form>`, ami POST kérést használ CSRF-el van védve. A `<form>`-ba beírt kódrészlet az `App\Http\Middleware\VerifyCsrfToken` middleware működéséhez szükséges. Ez alapértelmezés szerint a webes *middleware* csoportban található és automatikusan ellenőrzi, hogy a kérelem bemenetében lévő token megegyezik-e a munkamenetben tárolt tokennel. Ha ez a két token egyezik, akkor tudjuk, hogy a hitelesített felhasználó az, aki a kérést kezdeményezi. Példa:

```
<form action="/register" method="post">
    @csrf
    <input type="hidden" name="_token" value="{{ csrf_token() }}" />
```

A *CSRF token* megakadályozza a *Cross-Site támadást* a *cookie token* és a *szerver token* összehasonlításával. A *CSRF middleware* és *template* tag (`@csrf`) védelmet nyújt a

Cross Site Request Forgeries ellen. Ez a fajta támadás akkor következik be, amikor egy rosszindulatú weboldal tartalmaz egy linket, egy űrlapgombot vagy valamilyen *JavaScript*-et, amelynek célja, hogy valamilyen műveletet hajtson végre a mi weboldalunkon, a bejelentkezett felhasználó hitelesítő adatainak felhasználásával, aki a rosszindulatú weboldalra látogat a böngészőjében. A támadás becsapja a felhasználó böngészőjét, hogy valaki más hitelesítő adataival jelentkezzen be egy webhelyre.

4.4. Felhasználó jogosultság

Az adatbázisban egy új mezőt kellett létrehozni, amit *Role*-nak hívunk. Itt alapértelmezetten minden regisztrált felhasználó 0-ás értéket kap. A 0 a *sima felhasználó* jogait reprezentálja. Ezt a mezőt az *adminisztrátorok* illetve a *tulajdonosok* tudják változtatni. Az előbbiek az 1-es, az utóbbiak pedig a 2-es értéket kapják a *Role* mezőben. Az 1-es vagy 2-es jogokkal rendelkező felhasználók plusz menüpontokat és szerkesztési lehetőségeket kapnak az weboldalon.

5. Adatbázis

5.1. Adatbázis fontossága

Egy játékszoftver működésének kiemelkedően fontos eleme egy megbízható, jól megtervezett adatbázis, tervezés hiányában ugyanis a szoftver karbantartása körülményesebbé válhat, bővíthetősége megnehezül.

A mi adatbázisunkat a *Laravel* keretrendszer beépített adatbáziskezelő megoldásaival hoztuk létre, ugyanis a *migrációk* és az *adatbázis-seederek* rendkívül egyszerűvé teszik a fejlesztést bárhonnan.

5.2. Hogyan jön létre az adatbázis?

Egy lokális adatbázist egy újonnan beüzemelt fejlesztői számítógépen nagyon egyszerűen létre lehet hozni az előbb említett eszközökkel. Miután elindítottuk az *SQL szerverünket* (általában *xampp*) és megnyitottuk a *Laraveles projektet*, azután szimplán nyitnunk kell egy terminálablakot, majd beírni a következő parancsot:

```
php artisan migrate --seed
```

Ez által létrejön az *SQL szerveren* egy „*houseofswords”* nevű adatbázis. Az adatbázis létrejöttekor a táblák az *adatbázis-seedereknek* köszönhetően automatikusan feltöltődnek tesztadatokkal, így például az *„admin”* nevű felhasználó már egyből – egy játszható várossal, épületekkel, és kikutatott egységekkel – létrejön.

Ha már létezik az adatbázis lokálisan, de elavultak a benne lévő adatok egy egyszerű paranccsal frissíthetjük is az adatbázisunkat.

```
php artisan migrate:refresh --seed
```

5.3. Mi a helyzet a központi szerveren lévő adatbázissal?

A központi szerveren lévő adatbázis is hasonlóképpen módosítható, annyi különbséggel, hogy az adatbázishoz tartozó környezeti változókat a *.env* fájlban át kell írnunk a szerver adataira. A képen látható a különbség a változók között mindkét esetben. Ha lokális adatbázist szeretnénk használni, akkor a fájlban a szerver adataihoz tartozó 6 sort ki kell kommentezni, és vice versa.

```
# LOCAL DATABASE
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=houseofswords
DB_USERNAME=root
DB_PASSWORD=

# LIVE DATABASE
DB_CONNECTION=mysql
DB_HOST=79.121.11.122
DB_PORT=3306
DB_DATABASE=houseofswords
DB_USERNAME=houseof
DB_PASSWORD=************
```

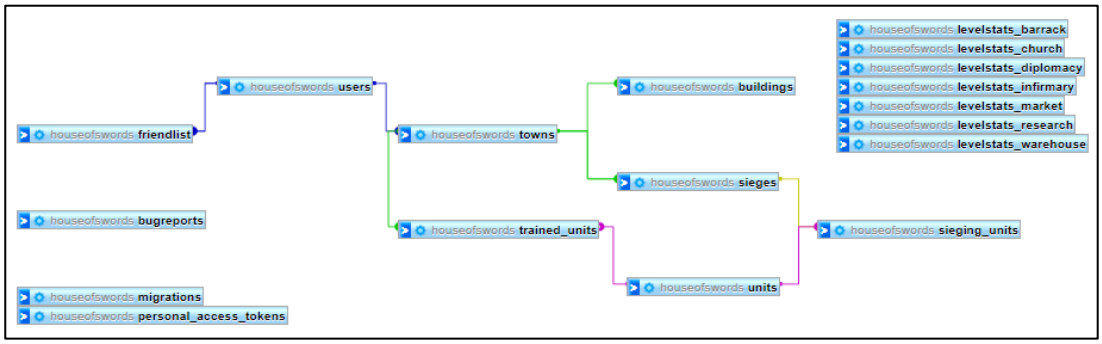
5.4. Hogyan lehet megtekinteni, illetve módosítani az adatbázist és a benne lévő adatokat?

Lokális adatbázis esetében egyszerűen – *DBForge* vagy *phpMyadmin* segítségével.

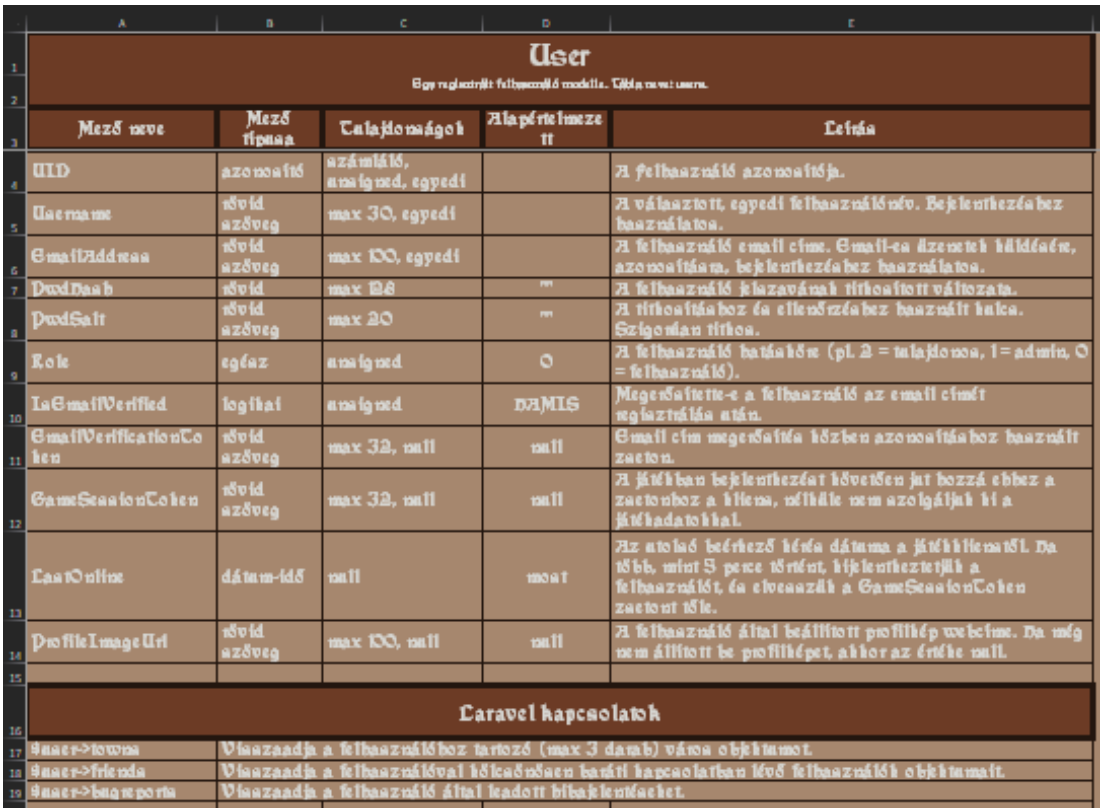
A szerveren lévő adatbázist a https://db.houseofswords.hu domain című oldalon lehet elérni, a bejelentkezési adatok megadása után. A *phpMyadmin* felületen a szokásos módon lehet felvenni, módosítani és törölni táblákat és rekordokat

5.5. Milyen táblák, modellek és kapcsolatok vannak az adatbázisban?

Az alábbi képen látható az adatbázisunk összes táblája, illetve a köztük lévő kapcsolatok. Az ábra a *phpMyadmin* felület vizuális szerkesztőjéből lett fotózva.



A dokumentációhoz tartozik egy *.xlsx* kiterjesztésű állomány *„Modellek gyűjteménye”* néven. Ebben a táblázatban minden lapon egy-egy adatbázis-modell mezői, illetve kapcsolatai láthatóak, az alábbi módon. (Megtekintéshez a mellékelt betűtípus, a Vinque Rg telepítése ajánlott, de nem kötelező.)



User				
Egy regisztrált felhasználó modellje. Tábla neve: users.				
Mező neve	Mező típusa	Tulajdonságok	Alapértelmezett	Leírás
UID	azonosító	számláló, unsigned, egyedi		A felhasználó azonosítója.
Username	rövid szöveg	max 30, egyedi		A választott, egyedi felhasználónév. Bejelentkezéshez használatos.
EmailAddress	rövid szöveg	max 100, egyedi		A felhasználó email címe. Email-es üzenetek küldéséhez, azonosításra, bejelentkezéshez használatos.
PwdHash	rövid	max 128	""	A felhasználó jelszavának titkosított változata.
PwdSalt	rövid szöveg	max 20	""	A titkosításhoz és ellenőrzéshez használt hulla. Szigorúan titkos.
Role	egész	unsigned	0	A felhasználó hatásköre (pl. 2 = tulajdonos, 1 = admin, 0 = felhasználó).
IsEmailVerified	logikai	unsigned	FALSE	Megerősítette-e a felhasználó az email címét regisztrálás után.
EmailVerificationToken	rövid szöveg	max 32, null	null	Email cím megerősítés közben azonosításhoz használt zseton.
GameSessionToken	rövid szöveg	max 32, null	null	A játékban bejelentkezést követően jut hozzá ehhez a zsetonhoz a kliens, vélhetően nem szolgáljuk ki a játékadatokkal.
LastOnline	dátum-idő	null	most	Az utolsó beérkező kérés dátuma a játékkliensről. Ha több, mint 5 perce történt, kijelentkeztetjük a felhasználót, és elvesszük a GameSessionToken zsetont tőle.
ProfileImageUrl	rövid szöveg	max 100, null	null	A felhasználó által beállított profilkép webcíme. Ha még nem állított be profilképet, akkor az értéke null.
Laravel kapcsolatok				
$user->towns	Visszaadja a felhasználóhoz tartozó (max 3 darab) város objektumot.			
$user->friends	Visszaadja a felhasználóval kölcsönösen baráti kapcsolatban lévő felhasználók objektumait.			
$user->bugreports	Visszaadja a felhasználó által leadott hibajelentéseket.			

Amennyiben a tábla egyik mezője kapcsolódik egy másikhoz, úgy a *tulajdonságok* mezőben az is meg lesz jelölve. Például az *egy-a-többhöz kapcsolat* miatt a *„towns”* táblában lévő *„Users_UID”* mező, ami annak a felhasználónak az azonosítójára mutat, akié a város.

Mező neve	Mező típusa	Tulajdonságok	Alapértelmezett	Leírás
Users_UID	idegen kulcs	users - 1:M - towns		A várost birtokló felhasználó azonosítója. A felhasználó törlése esetén kaszkádoltan törlődik a rekord.

A vizuális szerkesztőből fotózott képen látható kapcsolatok nem minden tábla között állnak fent. Néhol olyan egyedi, logikai kapcsolat áll fent két tábla között, amit a *szabványos SQL* szabályaival nem lehet megvalósítani. Például a *„bugreports”* tábla arra szolgál, hogy a felhasználók által jelentett hibaüzeneteket tárolja. Ha a felhasználó a rögzítés időpontjában be volt jelentkezve, akkor elmentjük az *email címét*, ami alapján beazonosítható, hogy ki jelentette a hibát. Ellenkező esetben ha nincs bejelentkezve a felhasználó, akkor *null* érték kerül az email cím mezőbe.

Az *SQL* nem engedi olyan kapcsolatok létrejöttét, ahol lehet *null* az idegen kulcs, ezért a kapcsolat csak *logikai* lesz, és a *Laravel modell* osztályában valósítható meg.

A *„Laravel kapcsolatok”* részben a táblázat alsó részén az kerül leírásra, hogy egy *modell objektumnak* milyen függvénye van, ami megvalósít egy-egy ilyen kapcsolatot.

$user->bugreports	Visszaadja a felhasználó által leadott hibajelentéseket.

Például egy *„user”* típusú objektumnak *Laravelen* belül van egy *„bugreports”* nevű függvénye, ami visszaadja az összes olyan rekordot a *„bugreports”* táblából, ahánynál az *email cím mező* egyezik a *felhasználó email címével*.

Ilyen logikai kapcsolat áll fent még például a *„buildings”* és a *„levelstats”* táblák között. Egy *„building”* objektum csak akkor kapcsolódik egy *„levelstats_épületTípus”* táblához, ha a *„building”* objektum típus mezője egyezik az *„épületTípus”*-sal. Vagyis egy *templom* típusú épület a *„levelstats_church”* táblához kapcsolódik, míg egy *raktár* típusú a *„levelstats_warehouse”* táblához.

A táblázatban több információ van az egyes modellekről, mezőikről és kapcsolataikról.

6. Sablonok használata

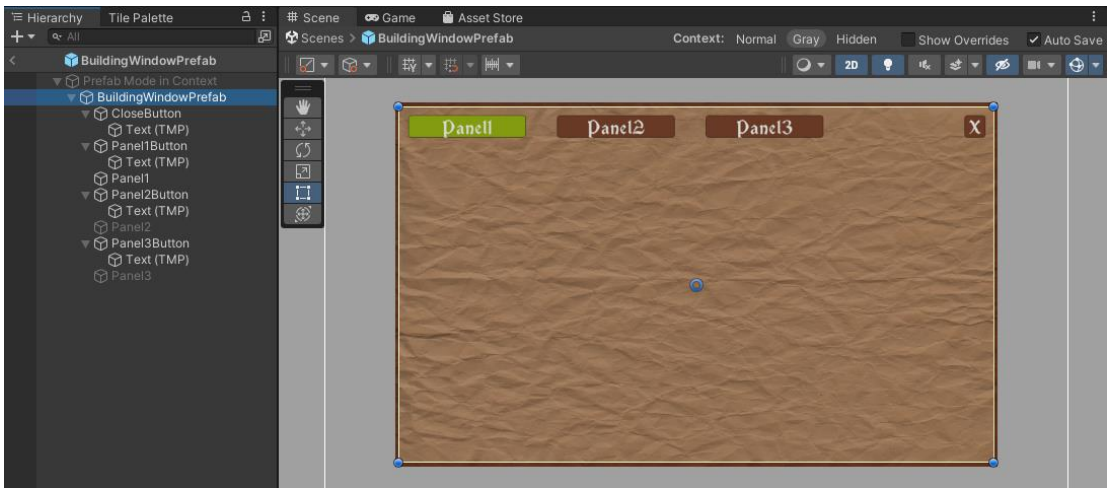
6.1. Prefabok

A frontend kinézet kiépítését egységes sablonok, úgy nevezett „prefabok” létrehozásával kezdtük el, a játék egységes kinézete és a munkafolyamat gyorsítása érdekében. A Unity által könnyen létrehozható prefabok olyan előre elkészített játékobjektumok, amelyeket testre szabásuk után bármennyiszer felhasználhatóak. Szerkesztése esetén az összes belőle létrehozott elem frissül, ezért könnyen módosítható.

Az összes többször előforduló GUI elemre létrehoztunk egy-egy ilyen sablont. (szöveg, gomb, beviteli mező, csúszka, stb…).

Példa:

Mivel a játékban minden egyes épületnél felugrik egy ablak, ezért érdemes volt létrehoznunk erre egy prefabot:



Hogy a képen is jól látható egy prefab tartalmazhat egyéb sablonokat, ebben az esetben az ablak gombokat és paneleket. A bezáró gombok funkcióját elég volt egyszer megírni, hiszen ezt megtehettük magán a prefabon.

Egy ablak megnyitását és bezárását egy függvénnyel oldottuk meg, ami egy bekért logikai változó alapján állítja a megjelenést. Ezzel egyszerre az épületek elérhetőségét is változtatjuk.

```
3 references
public static void enableBuildingWindows(bool value)
{
    GameObject.Find("WindowCanvas").GetComponent<Canvas>().enabled = value;

    BuildingsManager buildingsManager = GameObject.Find("BuildingsManager").GetComponent<BuildingsManager>();
    buildingsManager.setBuildingCollidersTo(!value);
    buildingsManager.windowIsOpen = value;
}
```

Tehát egy új *BuildingWindow* létrehozásánál már nincs egyéb dolgunk a *CloseButton*-nal, egyből neki is lehet állni a GUI felület kialakításához.

6.2. Layouts szerkezet

A prefabokhoz hasonlóan a laravelben is sablonokat hoztunk létre. A minden oldalon ismétlődő részeket a könnyű átláthatóság érdekében külön *.blade.php* állományokba rendeztük. Ezeket az *app.blade.php*-ba egyszerűen beimportálhatjuk `@include` segítségével. Ilyen fájlok a n*avigáció, lábléc,* a *css* linkek és *script*-ek beillesztése.

Ezzel így egy teljes weblapkép összeáll, tehát már csak az oldal `<main>` részét kell változtatnunk a route-tól függően a `@yield` segítségével.

7. Webszerver

A weboldal éles tesztelése és a játék kipróbálása miatt egy folyamatosan elérhető számítógépre volt szükségünk, hogy az adatbázis is elérhető legyen a nap bármelyik pillanatában. Erre egy *Synology DS418Play NAS (Network Attached Storage),* Linux alapú hálózati adattárolót választottunk. A webszervert telepíteni kellett a *Synology* saját linuxos csomagközpontjából, illetve a *PHP* egyes verzióit is (jelen esetben a 8.0-ás verziót használjuk, a telepítés pillanatában ez volt a legfrissebb támogatott verzió). Továbbá szükség volt még a *phpMyAdmin* és *MariaDB MySQL* adatbázis szerverre, illetve az *Apache HTTP* szerverre, melynek a 2.4-es verzióját használjuk. Tartalék szerverként a *NGINX szerver* is be lett állítva.

7.1. Domain beállítása

A megvásárolt *houseofswords.hu* domain címet a domain szolgáltató oldalán átirányítottuk a hálózati adattároló hálózatára, melyen az otthoni routeren a megfelelő beállításokkal ki kellett oldali a használatos portokat, jelenleg a 80, 81, 443, 3306 portokat (email szerverhez egyéb portokat is használunk).

7.2. Szerver beállítása

A *NAS*-on futó webszerveren meg kellet adni a weboldalunk fő fájljához vezető útvonalat (*index.php*), illetve a portokat be kellett állítani, hogy melyeket használja. Továbbá a *HTTP szervert* és a *PHP* verzióját és engedélyeit is be kellett állítani (HTTP 2.4-es verzió, PHP 8.0-ás környezet teljes körű hozzáféréssel).

A *Laravel* keretrendszert, melyet a weboldalhoz és a backendhez egyaránt használunk, parancssorból kellett telepíteni, melyet a *PuTTY* programmal *SSH*-n keresztül értünk el (Ehhez a szokásos 20-as portot használtuk. Csak belső hálózatról érhető el a konzol).

A `/usr/local/bin` mappába kellett telepíteni a *Composer* csomagközpontot a többi program mellé a következő paranccsal:

```
sudo curl -s http://getcomposer.org/installer | sudo php80
```

Elérési út beállítása a *Composer* parancsnak:

```
sudo vi composer
```

Beillesztettük a fájlba az elérési utat (path):

```
#!/bin/bash
php80 /usr/local/bin/composer.phar $*
sudo chmod --reference=composer.phar composer
```

Végül ellenőriztük a telepítés sikerességét:

```
composer –version
```

(2.4.4-es verzió)

Ezután elő kellett készíteni a weboldalt egy minta telepítésével:

```
composer create-project --prefer-dist laravel/laravel <appnév>
```

Illetve egyéb függőségeket is telepítettünk pl: `php80 artisan ui bootstrap –auth` (*Bootstrap CSS* és *JavaScript* keretrendszer).

Mivel a telepítés egy admin jogokkal rendelkező felhasználó alatt történt a megfelelő biztonság érdekében és hogy a *HTTP* felhasználó írni és olvasni is tudja a számár

szükséges mappákat a mappahozzáférésnél a weboldalt tartalmazó mappának a jogosultságait és tulajdonosát módosítani kell!

Nagyon fontos, a *Bootstrap cache* és *storage* mappáinak a tulajdonosát és hozzáférési beállításait módosítani kell a megfelelő működés érdekében egy sima felhasználóra.

```
- sudo chown -R $USER:http bootstrap/cache
- sudo chown -R $USER:http storage
- chmod -R 775 bootstrap/cache
- chmod -R 775 storage
```

Hogy az *NGINX* tartalék webszerverrel is működjön a weboldal a következő beállítást kellett elvégezni, hogy ne csak a főoldal, hanem az egyéb fájlokra történő routolás is működjön:

```
sudo vi /etc/nginx/app.d/server.webstation-vhost.conf
```

Ebbe a fájlba hozzá kellett adni a következő sort :

```
location / { try_files $uri $uri/ /index.php?$query_string; }
```

Ezután újra kellett indítani az *NGINX* webszervert a következő paranccsal:

```
sudo nginx -s reload
```

Hasznos parancsoka cache törlésére:

```
- php80 artisan view:clear
- php80 artisan route:clear
- php80 artisan cache:clear
- php80 artisan config:cache
```

7.3. GitHub használata

Ezzel beállításra került a *NAS*-on a webszerver és a Laravelles project. Viszont a weboldal frissítése és fejlesztése közben rájöttünk, hogy rendkívül macerás és nehézkes, ha a *GitHubon* lévő brancheket külön-külön másoljuk fel a *NAS*-ra. Ez kompatibilitási és adatvesztési problémákhoz vezetett és újra kellett csinálnunk néhány funkciót, mivel felülíráskor elvesztek a korábban írt sorok. Erre a problémára azt sikerült kitalálnunk, hogy a *NAS*-ra is telepítettük a *Git*-et és a beépített feladatidőzítő segítségével percenként lefuttatunk egy `git pull` parancsot. Ezzel tökéletesen kiküszöböltük az adatvesztést. A *GitHub* segítségével a különböző brancheken tudtunk dolgozni a különböző részfeladatokon, és a különbségeket a fő mappába mergeljük ami percenként fetchelve van a *NAS*-sal. Amint tesztelve és ellenőrizve lett egy részmechanika *main branchre* történő *mergelés* után egy percen belül láttuk is a

változtatásokat a szerveren és élesben lehetett használni. Ez az automatizálás rengeteg időt megspórolt nekünk.

8. Email szerver

8.1. Szerver beállítása

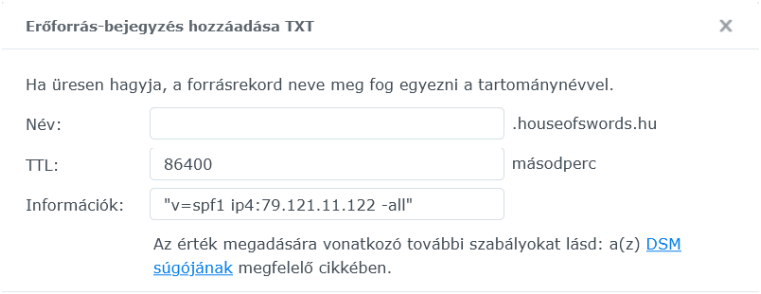
A weblapon történő regisztrációnál a felhasználónak hitelesítenie kell az email címet. Később ha elfelejtené a jelszavát erre a hitelesített email címre tud a rendszer küldeni egy jelszó visszaállító linket. Továbbá a hiba jelentéseknél kapunk egy értesítő emailt hogy valaki bejelentett egy hibát. Ezekhez a funkciókhoz szükségünk volt egy *email szerverre*, hogy a saját domainünket fel tudjuk használni. A *Synology NAS*-on a csomagkezelő központban telepítettük a *Synology MailPlus Server*-t és egy séma alapján konfiguráltuk a levelező szerver beállításait.

Ezután a domain szolgáltató oldalán be kellett állítanunk a megfelelő *DNS* rekordokat:

Ezzel a rekorddal a bejövő levelek címét beállítottuk *mail.houseofswords.hu*-ra ami a *NAS* külső *IP-címére* mutat.

Ezután a egy *MX* rekordot beállítottunk amely a domain névhez a levelezésért felelős szervereket mutatják meg. Így már tudunk leveleket fogadni és küldeni a szerverről,

viszont hitelesítések híján csak a spam mappába érkeznek meg a levelek. Így be kellett állítani a következőket:



Egy *SPF* rekorddal beállítottuk az egyetlen email szerverünket hogy az *IP-címe* alapján hiteles legyen, így már a levelező rendszerek nem szemét levélnek érzékelik az általunk küldött leveleket.

Ezzel az utolsó beállítással pedig egy *DKIM mezőt* hoztunk létre amely egy digitális aláírásnak felel meg. Így már a levelek és a szerver is megkapta kellő hitelesítéseket. Ezután az *otthoni router*-en a következő portokat oldottuk ki, hogy kívülről is minden elérhető legyen, ne csak a belső hálózaton:

- 25 – SMTP
- 465 – SMTP/SSL
- 587 – SMTP/TLS
- 143 – IMAP
- 993 – IMAPS
- 110 – POP3
- 995- POP3S

Ezután a *MailPlus* szerveren létre kellett hozni a megfelelő usereket a megfelelő domain névhez és kész is a szerver beállítása. Egy felhasználót korábban létrehoztunk már „*houseof*” néven az adatbázis eléréshez a *PhpMyAdminhoz*. Ezt a felhasználót használjuk itt is, egy *no-reply@houseofswords.hu* és egy *info@houseofswords.hu* email címet rendeltünk hozzá. Ezután a kapcsolatot bármilyen emailkezelő alkalmazással le lehet tesztelni, mi *Windows Mail, Outlook* és *Apple Mail*-en teszteltük a kapcsolatot és mindenhol sikeresnek bizonyult. A bejövő levelek szerverneve:

mail.houseofswords.hu, a kimenő levelek szervre: *smtp.houseofswords.hu*. Természetesen elsősorban 993-as és 587-es portokat használjuk a titkosított kapcsolatok végett.

8.2. Laravel kapcsolódása a szerverhez

.env- fájlba a következőket állítottuk be:

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.houseofswords.hu
MAIL_PORT=587
MAIL_USERNAME=houseof
MAIL_PASSWORD=$PASSWORD
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS="no-reply@houseofswords.hu"
MAIL_FROM_NAME="House of Swords"
```

/config/mail.php:

```
    'mailers' => [
        'smtp' => [
            'transport' => 'smtp',
            'host' => env('MAIL_HOST'),
            'port' => env('MAIL_PORT'),
            'encryption' => env('MAIL_ENCRYPTION'),
            'username' => env('MAIL_USERNAME'),
            'password' => env('MAIL_PASSWORD'),
            'timeout' => null,
            'local_domain' => env('MAIL_EHLO_DOMAIN'),

            'auth_mode' => null,
            'verify_peer' => false,
        ],
```

Az utolsó kettő sorra azért volt szükség, mert a laravel és a szerver közti hitelesítés nem volt lehetséges, de mivel fizikailag egy szerveren van, nem jelent gondot.

9. Kriptográfia és biztonság

9.1. Titkosítás

Mivel projektünkben intenzív hálózati kommunikációt használunk, és felhasználók adatait, jelszavát kezeljük, ezért kiemelten fontos a biztonság kérdése.

Alapvető tény, hogy felhasználók jelszavát nem szövegként kell tárolni egy adatbázisban, hiszen ha egy támadó fél hozzájutna az adatokhoz, akkor egyből tudná a bejelentkezési adatokat. Ennek elkerülése érdekében komplex, *SHA512-es titkosítást* használunk.

Ez egy olyan algoritmus neve, amely egy bemeneti karakterláncot látszólag véletlenszerűen átalakít egy hexadecimális értékké. Azonban ez a folyamat koránt sem véletlenszerű, hiszen ha a folyamatot többször hajtjuk végre ugyanazzal a bemenettel, mindig ugyanazt a hexadecimális eredményt fogjuk kapni, valójában egy egyértelmű hozzárendelésről beszélünk.

Minden eltérő bemeneti karakterlánc esetén egy teljesen más hexadecimális értéket (ún. *hash*-t) kapunk, ha ezekre ránézünk, kizárt dolog, hogy az eredeti jelszót megismerjük. Például:

“House of Swords”:
c8eb916e211f7e9062ec5a367bd4d756fea4a3d2462dc567b32a0fcfaf509050e71dacd6
1ba48296d3ce8b2bbe9fb558d47f5d38efb35294bb613d78981ecff9
“House of Sword”:
f05d153fba12b96b5428ec75345ece77518af0bc9ba9afacd779cfcca76f1831fdd0d046
39304ad8309c7d7cce15ddea4ef8d376b7af2068e8d1880b5ffde36a

Ahogy az jól látszik, ha a bemenet csak egy karakterrel tér el, már akkor is teljesen különböző eredményt kapunk.

9.2. Salt & Pepper

Azonban ez önmagában nem teljesen megfelelő a véleményünk szerint, hiszen az egyszerűbb jelszavakat egy “*Dictionary attack*”, avagy egy “*Szótár támadás*” segítségével így is könnyedén fel lehet törni.

Ez ellen úgy tudunk védekezni, hogy a jelszó titkosítása előtt a bemenet végéhez ún. “*Salt*” és “*Pepper*” karakterláncokat fűzünk.

A “*Salt*” karakterlánc az egy (a mi esetünkben 20 karakter hosszú) véletlenszerűen generált, mindenféle karakterből álló lánc, amelyet a titkosított jelszó mellett tárolunk. A “*Pepper*” egy véletlenszerű, kis- vagy nagybetűs karakter az angol ábécéből, amit nem tárolunk el. Miután hozzáfűztük a jelszó végéhez ezt a két karakterláncot, az után következik a titkosítás, majd az eltárolás.

Ezzel azt érjük el, hogy ha a felhasználó egy egyszerűbb jelszót ad meg, ami szerepel egy adathalászok által ismert adatbázisban, akkor sem fogják tudni egyből feltörni, hiszen az adathalászok ezeket a jelszavakat és a hozzájuk tartozó *hexadecimális hash*-t tárolják, de ezek a szavak a “*Salt*” nélkül lettek titkosítva, így nem fognak egyezni.

A “*Pepper*” ugye nem került tárolásra, de akkor honnan tudjuk, hogy melyik karaktert fűztük a jelszó végére titkosítás előtt? Igazából nem tudjuk, erre a megoldást a “*Brute-*

force” módszer jelenti, avagy minden esetben, amikor a felhasználó be akar jelentkezni, a beírt jelszó végére hozzáfűzzük a “*Salt*”-ot, és az így keletkezett szó végére egyesével kipróbáljuk az 52 lehetséges betűt, majd egyesével az 52 szót titkosítjuk. Ha valamelyik hexadecimális érték egyezik az adatbázisban lévővel, akkor tudjuk, hogy jó jelszót írt be a felhasználó, ellenkező esetben pedig téveset. Ez a módszer programidőben nem szignifikáns, hiszen egy karakterláncot összehasonlítani egy másikkal a másodperc töredéke alatt lehetséges, 52-vel megcsinálni ugyanezt pedig a másodperc 52 töredéke alatt lehet, ez még mindig elenyésző időben.

Ez úgy javít a biztonságon, hogy ha a támadó is “*Brute-force*” módszerrel szeretné feltörni a jelszót, akkor 52-szer több időbe fog neki telni, hiszen a végén lévő karaktert is fel kell törniük, amire 52 lehetőség van (a-Z).

Így a végleges jelszó, illetve a hozzátartozó *hash* valahogy így néz ki:

“House_of_Swords” + “1aGtPGDRvCDpShGZyjTR” + “M” = “House_of_Swords1aGtPGDRvCDpShGZyjTRM”

76a2539baa13634355853bec9f5984e09eb5b8b1db49f6cd5bbdcb801f527f97d28009
13d8f004a6a03b21dd544a83b96e3b57348a5407c92a881e57aba9626a

10. Frontend API hívások

10.1. Hálózati kommunikáció

Fontos része a szoftver működésének, hogy a Unity-ban elkészített játék tudjon kommunikálni a szerverrel, hogy az adatok mentésre kerüljenek, illetve hogy a kliensek egymással tudjanak interaktálni. A szervernek a kliens kérésére ki kell szolgálnia azt adatokkal, illetve létre kell hozni, módosítani kell adatbázis rekordokat.

A frontenden a statikus *APIHelper* nevű osztály felelős ezért. Ezen osztály bármely másik szkriptből elérhető, publikus. Metódusaival aszinkron API kéréseket tud küldeni a szerver felé.

Fontos, hogy a kérések aszinkronok legyenek, hiszen a megfelelő játékélmény érdekében (pl. egy “ostrom indítása” gomb megnyomására) nem szabad várakoznia, megállnia a programnak. Így simább, minőségibb érzete lesz a szoftvernek.

Az *APIHelper* metódusai a *System.Net* és a Unity beépített *UnityEngine.Networking* osztályokat használják a kommunikáció megvalósítására. Ha adatot kér le az egyik metódus a szerverről (ez lehet például egy felhasználó, vagy egy város adatai, de akár ezekből több is egyszerre), akkor azt a Scripts/Models mappában található megfelelő

modell formájában adja vissza (pl. egy felhasználó adatait a Scripts/Models/User modellben).

10.2. Bejelentkezés API kéréssel

Tökéletes példa egy API kérésre a bejelentkezés folyamata. Az *APIHelper* osztály *postTryLogin()* metódusa ezt valósítja meg. A folyamat try-catch szintaxissal van felkészítve az esetleges hálózati kommunikációs hibákra.

Első lépésben a paraméterként kapott *felhasználónevet* és *jelszót* állítjuk be a kérés tartalmának. Ezután a 114. sorban történik a kérés elküldése, és a várakozás a válaszra.

```
1 reference
104  public static async Task<User> postTryLogin(string username, string password)
105  {
106      try
107      {
108          FormUrlEncodedContent content = new FormUrlEncodedContent(new Dictionary<string, string>
109          {
110              { "Username", username },
111              { "Password", password }
112          });
113
114          HttpResponseMessage response = await client.PostAsync(loginRoute, content);
115          response.EnsureSuccessStatusCode();
116
117          string responseString = await response.Content.ReadAsStringAsync();
118          Debug.Log(responseString);
119
120          User user = JsonConvert.DeserializeObject<User>(responseString);
121          APIHelper.gameSessionToken = user.GameSessionToken;
122
123          return user;
124      }
125      catch (Exception ex)
126      {
127          Debug.LogError(ex);
128          return null;
129      }
130  }
```

A client nevű objektum egy *HttpClient* típusú mezője az *APIHelper* osztálynak. Ez az objektum tartja fent a kapcsolatot a hálózattal a játék teljes menete során.

```
26 references
public static class APIHelper
{
    // HTTP CLIENT FOR COMMUNICATION
    private static readonly HttpClient client = new HttpClient();
```

A *loginRoute* a kérés útvonala, ez is egy statikus tulajdonsága az *APIHelper* osztálynak, hiszen ezek a játék közben nem változnak. A könnyebb átláthatóság érdekében ezeket egy helyre gyűjtöttük (hiszen nem csak egy útvonal van, amire API kérések lesznek küldve), és csoportosítottuk

```
#region GLOBAL API ROUTES
//[SerializeField] private static string usersAPIRoute = "https://houseofswords.hu/api/users";
//[SerializeField] private static string townsAPIRoute = "https://houseofswords.hu/api/towns";
//[SerializeField] private static string buildingsAPIRoute = "https://houseofswords.hu/api/buildings";

//[SerializeField] private static string unitStatsAPIRoute = "http://houseofswords.hu/api/stats/units";

//[SerializeField] private static string churchStatsAPIRoute = "https://houseofswords.hu/api/stats/church";
//[SerializeField] private static string researchStatsAPIRoute = "https://houseofswords.hu/api/stats/research";
//[SerializeField] private static string warehouseStatsAPIRoute = "https://houseofswords.hu/api/stats/warehouse";
//[SerializeField] private static string marketStatsAPIRoute = "https://houseofswords.hu/api/stats/market";
//[SerializeField] private static string diplomacyStatsAPIRoute = "https://houseofswords.hu/api/stats/diplomacy";
//[SerializeField] private static string barrackStatsAPIRoute = "https://houseofswords.hu/api/stats/barrack";
//[SerializeField] private static string infirmaryStatsAPIRoute = "https://houseofswords.hu/api/stats/infirmary";

//[SerializeField] private static string actionsAPIRoute = "https://houseofswords.hu/api/actions";

//[SerializeField] private static string loginRoute = "https://houseofswords.hu/api/createGameSession";
#endregion

#region API ROUTES FOR LOCAL TESTING
[SerializeField] private static string usersAPIRoute = "http://localhost:8000/api/users";
[SerializeField] private static string townsAPIRoute = "http://localhost:8000/api/towns";
[SerializeField] private static string buildingsAPIRoute = "http://localhost:8000/api/buildings";

[SerializeField] private static string unitStatsAPIRoute = "http://localhost:8000/api/stats/units";

[SerializeField] private static string churchStatsAPIRoute = "http://localhost:8000/api/stats/church";
[SerializeField] private static string researchStatsAPIRoute = "http://localhost:8000/api/stats/research";
[SerializeField] private static string warehouseStatsAPIRoute = "http://localhost:8000/api/stats/warehouse";
[SerializeField] private static string marketStatsAPIRoute = "http://localhost:8000/api/stats/market";
[SerializeField] private static string diplomacyStatsAPIRoute = "http://localhost:8000/api/stats/diplomacy";
[SerializeField] private static string barrackStatsAPIRoute = "http://localhost:8000/api/stats/barrack";
[SerializeField] private static string infirmaryStatsAPIRoute = "http://localhost:8000/api/stats/infirmary";

[SerializeField] private static string actionsAPIRoute = "http://localhost:8000/api/actions";

[SerializeField] private static string loginRoute = "http://localhost:8000/api/createGameSession";
#endregion
```

őket aszerint, hogy *helyi hálózaton* (*localhost*) teszteljük, vagy az éles *szerveren futó backenddel* (*houseofswords.hu*).

A *response.EnsureSuccessStatusCode()* sor megvizsgálja a kérés hatására visszaérkezett választ, és ha a kérés státuszkódja nem sikeres, akkor hibát dob, és a függvény a *catch* ágon folytatódik.

A *response.ReadAsStringAsync()* metódussal olvashatjuk ki az érkezett válasz törzsét, így jutunk hozzá a kért adatokhoz.

A kért adatokat ezek alatt a sorok alatt feldolgozzuk (például jelen esetben egy *User* objektumot készítünk a *JSON válaszból*, amit vissza fogunk adni, és beállítjuk az *APIHelper gameSessionToken* mezőjét, erről a következő pontban lesz szó), majd visszaadjuk a kívánt adatokat.

Ha a függvény hibába ütközik, akkor kiíratjuk az esetet a *Debug konzolra*, és általában null-t adunk vissza eredményül (de ez igény szerint módosítható).

10.3. API titkosítás tokennel

```
private static string _gameSessionToken;
1 reference
[SerializeField] public static string gameSessionToken {
    private get => _gameSessionToken;
    set
    {
        _gameSessionToken = value;
        client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", value);
    }
}
```

Amikor a felhasználó bejelentkezett, a felhasználó objektumban létrejött egy *GameSessionToken* nevű karakterlánc. Ezt csakis a felhasználó és a szerver ismerheti, így ez a karakterlánc használható a két fél közötti kommunikáció hitelesítésére.

Amikor a token beállításra kerül az *APIHelper* osztályban, a további kérések részére automatikusan beállítjuk, hogy *Bearer tokenként* csatolják a kérés fejlécébe. Ez alapján fogja tudni a szerver, hogy hiteles kérés érkezett-e be, vagy sem. A szerver csak azokat a kéréseket hajlandó teljesíteni, amikkel be szeretnénk jelentkezni, vagy amelyek már tartalmazzák fejlécben az azonosító tokent.

Alternatív megoldásként az is működik, ha a kérés útvonalának végére ún. *query paraméterként* megadjuk a tokent „*gamesessiontoken*” kulccsal.

Erre a karakterláncra azért van szükség, hogy azonosítani tudjuk a felhasználót, miután már egyszer beírta a felhasználónevét és a jelszavát. Ezzel a módszerrel nem kell

minden kérésben benne lennie a *felhasználónévnek* és a *jelszónak*, hogy tudjuk, a küldő a hiteles felhasználó. Így a biztonság a szerver és a kliens között megerősödik.

Fontos kiemelni, hogy egy ilyen token nem érvényes örökké. A kiadás után számolt 5 percig érvényes, amennyiben abban az 5 percben nem érkezik egy újabb kérés, amelyik ezt a tokent tartalmazza. Ha érkezik egy kérés ezzel a tokennel, akkor az 5 perc újra kezdődik, mielőtt elavul a karakterlánc, és újra be kell jelentkezni. Ezzel elkerülhetjük az örökké működő tokenek létrejöttét, és ha a felhasználó nem küld kérést 5 percig, akkor automatikusan kijelentkeztetjük.

11. Adatbázis automatikus frissítése

11.1. Tervezések

A megoldásához egy Linux szerveren is futó automatikus parancsindító programot kellett keresnünk. Hosszas kutatás és utánajárás következményeként A *Cron* nevű feladatidőzítőt találtuk a legoptimálisabbnak. Szerencsénkre a *Synology NAS* adattároló (amin az adatbázis és a webszerver is fut) önmagába foglal egy beépített feladatidőzítőt, aminek grafikus felülete miatt egyszerűen lehet kezelni és konfigurálni. Ez a beépített *Task Scheduler* egy *Cron* alapú szoftver, szóval tökéletesnek bizonyult számunkra.

Mindezek után utánajártunk a *Laravel* beépített függvényeinek, hogy találunk-e egy célnak megfelelőt, hogy ne kelljen a teljesen magunktól megírni. Rá is bukkantunk a *Laravel* beépített *Task Scheduling* funkciójára. Ehhez mindössze az *app/Console/Kernel.php* fájlban kellett egy függvényt létrehoznunk:

```
protected function schedule(Schedule $schedule)
    {
        $schedule->command('TownResourcesUpdate')
            ->everyMinute();
    }
```

A függvény feladata csupán, hogy a megadott commandok alapján minden percben futtassa le a kódot, ami egy update parancs az adatbázisnak.

A függvényt hosszasan teszteltük és ellenőriztük és beépítettük a Laravel projektbe.

11.2. Projektbe ágyazás

Ezt a php artisan schedule:run paranccsal tudjuk megtenni.

Az alábbi parancsot kell a *Synology NAS* beépített feladatidőzítőjében beállítani percenkénti futtatásra.

/volume4/@appstore/PHP8.0/usr/local/bin/php80
/volume4/web/HouseOfSwords/artisan schedule:run

Így a parancs az általunk is használt *php 8.0*-ás verzióval fog futni és a későbbiekben elkerülve a kompatibilitási problémákat, ha esetleg automatikusan frissülne a php verzió.

Ezután egy problémába ütköztünk. Egy *„access denied for user 'root'@'localhost'”* hibakódú oldalt kaptunk válaszul. Ezt hosszas tesztelgetés után meg tudtuk oldani. A változtatások miatt a Linux szerveren az alapértelmezett http-felhasználónak újra kellett osztani az egész mappára és almappáira az írás és olvasás jogot, ezek után hiba nélkül futott a weboldal is és az adatbázis frissítő is sikeresen implementálva lett.

12. Pathfinding

A *Unity navigációs rendszer* alapvetően nem támogatja a 2d-s *patfinding*-ot, ezért egy *package*-et kellett letölteni a https://arongranberg.com/astar/download oldalról. Már az ingyenes verzió is tartalmazott mindent, ami nekünk kellett.

Az *A** nevű objektum a *pathfinding* közepe. Ennek a *Pathfinder componens*-e határozza meg melyik területen engedélyezett a mozgás. Ehhez szüksége van egy *Obstacle Layer Mask*-ra, ami az akadályokat és azok *collider*-ét tartalmazza, a mi esetünkben ez az *npc_path*. Lényegében ezután végignéz minden csomóponton, hogy keletkezne e átfedés, és ez alapján létrehoz egy *navigation gride*-et, ami a képen látható. A kék részeken engedélyezett, a piros négyzettel jelölt részeken nem engedélyezett a mozgása.

A grafika és a logika el van különítve két objektumba. Ez azért hasznos, mivel így könnyen ki lehet cserélni a grafikát anélkül hogy a logikába bele kéne nyúlni.

A logikához hozzá van adva az *AIPath* és a *Seeker componens*. Ez a kettő olyan *scriptek*, amik minden *AI objektum* működéséhez szükségesek. A *Seeker* felelős a célhoz vezető útgenerálásért. Az *AIPath* pedig a *Seekert* fogja kontrollálni és mozgatni a grafikát.

Cél megadásához hozzá kell csatolni az *AI Destination Setter - Target*-jához a kívánt objektumot. Ahhoz, hogy a grafika mindig a cél felé nézzen az x sebesség alapján kell tükröznünk. Ha ez pozitív, akkor jobbra, ha negatív, akkor balra halad a karakter. Ez az alábbi kódrészletben látható:

```
void Update()
{
    // flipping based on X speed
    if(aiPath.desiredVelocity.x >= 0.01f) // moving to the right
    {
        transform.localScale = new Vector3(-1f, 1f, 1f);
    }
    else if (aiPath.desiredVelocity.x <= -0.01f) // moving to the left
    {
        transform.localScale = new Vector3(1f, 1f, 1f);
    }
}
```

13. Tesztelés

13.1. API tesztelés

Az backendről meghívott API-kat kétféleképpen teszteltük. Egyrészt *Visual Studioban* a *Thunder Client* nevezetű kiegészítő csomaggal az éppen létrehozott végpont helyességét azonnal tudtuk tesztelné manuálisan. Másrészt egy automata tesztet is írtunk *Postman*-ben. Itt az összes API-t egy kollekcióba összegyűjtöttük és szortíroztuk, és ezután a teszt menüpontban egy speciális JavaScript nyelven metódusokat írtunk a válaszidő és válaszüzenet tesztelésére. Ezeket a teszteket egyben automatikusan le tudtuk futtatni, így gyorsabbá téve a különböző platformon történő teszteléseket.

14. Források

14.1. Általános

- https://stackoverflow.com/
- https://laravel.com/

14.2. Csapatmunka

- https://gist.github.com/jagrosh/5b1761213e33fc5b54ec7f6379034a22

14.3. Regisztráció és belépés - backend

- https://laravel.com/docs/10.x/csrf
- https://laravel.com/docs/10.x/validation
- Bólya Gábor tanár úr óráin tanultak

14.4. Webszerver

- https://community.synology.com/enu/forum/1/post/133463
- https://www.rackhost.hu/tudasbazis/online/dns-rekordok/
- https://kb.synology.com/hu-hu/DSM/help/Git/git?version=7

14.5. Email szerver

- https://kb.synology.com/hu-hu/DSM/help/MailPlus-Server/mailplus_server_creation?version=7
- https://kb.synology.com/en-au/DSM/tutorial/How_to_set_up_MailPlus_Server_on_your_Synology_NAS
- https://laravel.com/docs/10.x/mail
- https://www.tutorialspoint.com/laravel/laravel_sending_email.htm
- https://www.cloudways.com/blog/send-email-in-laravel/

14.6. Kriptográfia és biztonság

- https://www.youtube.com/watch?v=DMtFhACPnTY

14.7. Pathfinding

- https://arongranberg.com/astar/download
- https://docs.unity3d.com/Manual/Navigation.html
- https://www.youtube.com/watch?v=jvtFUfJ6CP8

14.8. Felhasznál grafikák

14.8.1. Tileset

- https://graphicriver.net/item/summer-medieval-city-game-tileset/23381324

14.8.2. Erőforrás ikonok:

- https://assetstore.unity.com/packages/2d/gui/icons/resources-and-craft-icon-pack-40750

14.8.3. Karakterek:

- https://assetstore.unity.com/packages/2d/characters/tasty-characters-castle-pack-108881
- https://assetstore.unity.com/packages/2d/characters/tasty-characters-village-pack-108879

14.8.4. Logó, brigád kártya:

- https://openai.com/product/dall-e-2