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]:	Pintér Ármin, Pirik Mihály, Obsitos Dominik
Képzés:	nappali munkarend
Szak:	5 0613 12 03 Szoftverfejlesztő és tesztelő technikus

A záródolgozat címe:

Forgotten Kingdom

Konzulens:	Bólya Gábor
Beadási határidő:	2024. 04. 22.

Győr, 2024. 04. 21.

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.	2024.02.28.	Témaválasztás és specifikáció	
2.	2024.03.24.	Záródolgozat készültségi fokának értékelése	
3.	2024.04.14.	Dokumentáció véglegesítése	

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, 2024. 04. 21.

Pintér Ármin	Pirik Mihály	Obsitos Dominik

A Forgotten Kingdom egy középkorban játszódó szerepjáték, stratégiai elemekkel vegyítve. A játékos saját karakterével fedezhet fel egy rég elfeledett királyságot.

1. Specifikáció

1.1. A játékról:

A játék környezete középkori időkben játszódik, egy dinamikus és izgalmas szerepjáték keretein belül. A játékosok egy elhagyatott királyságba csöppennek, ahol a fő feladatuk az elrabolt lakosok megmentése a szörnyek karmaiból, miközben fegyverüket egyre erősebbre fejlesztik.

A hely rejtélyes és veszélyes, tele kihívásokkal és lehetőségekkel. A játékosoknak harcolniuk kell a szörnyekkel, bányászniuk kell az értékes nyersanyagokat, aratniuk kell a földeken, és teljesíteniük kell különböző mellékküldetéseket.

A játékosoknak lehetőségük van egymással interakcióba lépni a piac funkció segítségével, ahol nyersanyagokat adhatnak és vehetnek egymástól, hogy elősegítsék egymás fejlődését.

1.2. A Frontend-ről:

A frontendünk összetett struktúrával rendelkezik, amely két fő részből áll. A fő oldalunk fejlesztése során a Vue.js keretrendszert alkalmaztuk, hogy dinamikus és felhasználóbarát élményt nyújtsunk.

Ugyanakkor a játék maga, natív JavaScript segítségével lett elkészítve. Ez azért fontos, mert a natív JavaScript lehetővé teszi a pontosabb és optimalizáltabb működést a játék logikájának és mechanizmusainak kezelésében.

1.3. A Backend-ről:

A backendünk kialakításakor az Express.js keretrendszert használtuk, amely a Node.js egyik legnépszerűbb keretrendszere. Az Express.js lehetővé teszi számunkra, hogy stabil és megbízható háttérszolgáltatást készítsünk a játék számára.

A backend fő feladata az adatbázis kezelése (MySQL), amelyben a játék minden fontos információját tároljuk. Az Express.js segítségével könnyedén kezelhetjük az adatbázis kapcsolatát, lekérdezéseket hajthatunk végre, és frissíthetjük az adatokat a játékosok tevékenységeinek megfelelően. Emellett a backend felelős különböző kérések kiszolgálásáért, például a felhasználó regisztrációja, belépése és a játékos statisztikák lekérdezése.

1.4. Mobil:

A játékunk komplexitása miatt nagy kihívás lenne annak átalakítása mobilalkalmazássá, ezért úgy döntöttünk, hogy a főoldalunkon (ami teljes mértékben reszponzív, és alkalmazkodik a különböző monitorok és mobil eszközök méretéhez) egy figyelőt helyezünk el a bejelentkezés után. Ez a figyelő érzékeli, ha mobil eszközt használ a felhasználó, és egy üzenettel tájékoztatja arról, hogy a játék eléréséhez használjon számítógépet. Így biztosítva azt, hogy megőrizzük a teljes projektünk mobil reszponzivitását és a játékosok is csak olyan környezetben játszhassanak, ahol a legjobb élményt kapják.

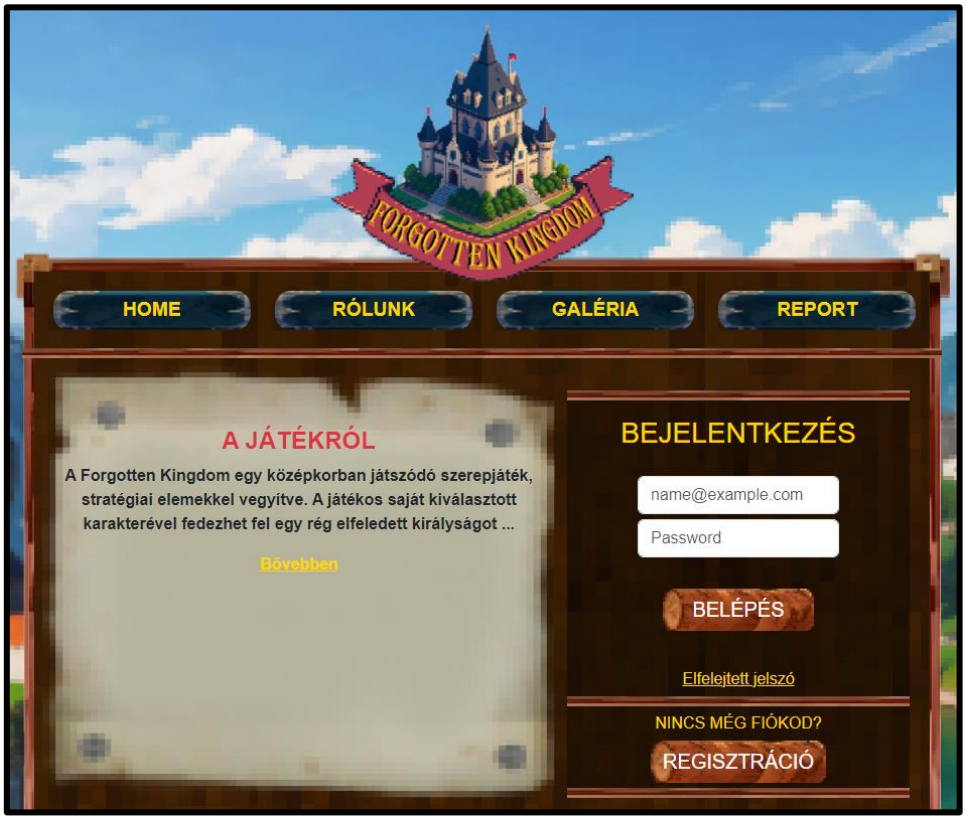
1.5. A főoldal:

A főoldalon lényegében minden megtalálható, ami a játékunkhoz kapcsolódik. Itt található a regisztrációs felület, amely lehetővé teszi a játékosok számára, hogy létrehozzanak egy fiókot. Emellett leírás is elérhető a játékról és a fejlesztő csapatról, hogy a felhasználók teljes képet kapjanak a projekt hátteréről és tartalmáról.

A galéria fül segítségével a játékosok megtalálhatják egy rövid leírással körítve, a már játékban is található különféle helyszíneket, karaktereket és egyéb fontos elemeket. Lehetővé téve számukra, hogy betekintést nyerjenek a játék világába akár még az elkezdése előtt is.

Egy hibajelentés funkció is rendelkezésre áll, hogy a játékosok könnyen és gyorsan jelenteni tudják az esetleges problémákat vagy hibákat a fejlesztő csapatnak, így segítve a játék folyamatos fejlesztését és javítását.

És természetesen, a belépési lehetőség, amely lehetővé teszi a játékosok számára, hogy elindítsák a játékot.



2. Csapat

2.1.Tervek:

A tervezéskor fontos volt számunkra, hogy mindenki teljes mértékben kibontakozhasson és kihasználhassa képességeit, ezért úgy döntöttünk, hogy részekre bontjuk a feladatokat, miközben mégis együttműködünk és besegítünk egymásnak.

Ármin főként a frontend fejlesztésével foglalkozik, mivel számára fontos tapasztalatokat szerezni ezen a területen.

Mihály vállalta a backend fejlesztését, mivel mindig is érdekelte ez a terület és szeretné mélyebben megismerni azt.

Dominik a játék grafikai tervezésével foglalkozik, mivel régebben is érdekelte őt más játékok grafikai felépítése.

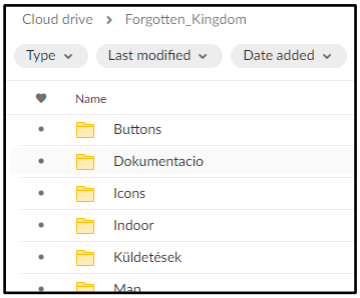
Ez a munkamegosztás lehetővé teszi számunkra, hogy mindenki a saját erősségeire és érdeklődési területére összpontosítson, miközben együtt dolgozunk azért, hogy létrehozzunk egy összehangolt játékélményt.

2.2. Fejlesztés:

Ahogy haladunk előre a projektben, egyre több tapasztalatot gyűjtünk munkánk során. A csapatmunka hatékonyan működik, és mindenki aktívan haladunk előre a végleges cél eléréséhez. Ugyanakkor felmerültek olyan dolgok, amelyekre kezdetben nem gondoltunk a tervezés során.

A munkát a főoldalunk elkészítésével kezdtünk. Az alapok lerakása után képekkel díszítettük, majd reszponzívvá tettük azt. Azonban amikor eljött az ideje a játék fejlesztésének, rá kellett döbbennünk, hogy ez a programozás egy más oldala, és viszonylag több idő kell ahhoz, hogy megszerezzük a szükséges tudást, mint azt gondoltuk.

A fejlesztés során a GitHubot és a MEGA Limited által biztosított 50 GB-os felhőtárhely szolgáltatást használtuk. Mindkettő segítségével könnyen és átláthatóan tudtuk kezelni az adatokat és megosztani azokat egymás között. Ez lehetővé tette számunkra, hogy hatékonyan dolgozzunk.



Később, a kellő tudás birtokában képesek voltunk elkezdeni a játék alapjait, de még így is számos problémába botlottunk, amik megoldást igényeltek.

Az iskolán kívüli kommunikációhoz a Discord szolgáltatásait használtuk. Ha szükség volt rá, akkor a hang- és videóhívás funkciójának köszönhetően könnyedén megtudtuk értetni magunkat és még a saját eszközünk képernyőjét is megoszthattuk egymás között.

3. Használati útmutató

3.1.A játék elérése

Mint a legtöbb webes játékot, úgy ezt is regisztráció után érhetjük el, amit a belépés követ.

3.2. Regisztráció

Ahhoz, hogy elkezdjük a Forgotten Kingdom által nyújtott élményeket, először is meg kell nyitnunk a böngészőnket és a címsorba beírnunk a https://bgs.jedlik.eu/forgottenkingdom weboldal URL-jét. Ezután keressük meg az oldal jobb felében található „Regisztráció” lehetőséget és válasszuk ki.

Miután átkattintottunk a regisztrációs folyamatra, következő lépésként töltsük ki az adatlapot a szükséges információkkal. Fontos, hogy adatainkat pontosan adjuk meg, hogy a regisztráció sikeres legyen. Miután minden adatot helyesen megadtunk, akkor már csak a belépés van hátra.

3.3. Belépés

Miután sikeresen regisztráltunk és meguntuk a galéria böngészését, következhet a játékba való belépés. Egyszerűen csak meg kell adnunk a regisztráció során rögzített adatokat, és máris kezdődhet a játék.

Sikeres belépést követően kezdetét veheti az elfeledett királyság története.

3.4. Felület

A játék kezdetekor számtalan fontos információ tárul elénk. A képernyő bal felén azonnal szembetűnik a kiválasztott karakterünk profilképe és az életerejét jelző csík. Ezek alatt látható pénzünk száma és a fegyverünk szintje. Ha rákattintunk a kis táska ikonra, akkor pedig a nyersanyagaink és mennységük mutatkozik meg, amelyekre szükségünk lehet a kalandok során. A sarokban található térkép ikonra kattintva pedig szemünk elé tárulnak a királyság különböző helyszínei.

Az oldal jobb szélén tájékozódhatunk az aktuális küldetéseinkről, és a sarokban lehetőségünk van kilépni a játékból, ha szükségét érezzük.

Az oldal közepén pedig természetesen a vidéket láthatjuk *darabokra osztva,* amelyeket fő küldetéseink során járhatunk be és fedezhetünk fel.

3.5. Irányítás

A kis karakterünket, a "W", "A", "S", "D" gombok lenyomásával irányíthatjuk a pályán, melyek segítségével felfedezhetjük a környezetet. Továbbá lehetőségünk van a bal egér gombbal támadni, hogy mindig készen álljunk az előttünk álló kihívásokra. Az 1, 2, 3, 4, és az 5 gombok lenyomásával pedig bekaphatunk egy almát, ihatunk egy kortyot, integethetünk megkedvelt szereplőknek és akár még táncolhatunk is a legyőzött szörnyek örömére!

3.6. Tájékozódás

Minden pályán irányt jelző táblák helyezkednek el, segítve ezzel utunkat és mutatva az elérhető útvonalakat a következő pályákra. Emellett a térképünk mindig kéznél van, hogy pontosan tudjuk, melyik darabkán vagyunk éppen.

3.7. Interakció

A játék során felfedezhetünk olyan helyeket, ahol interakcióba léphetünk a környezettel és más karakterekkel. Ezek a cselekedetek lehetnek épületbe való belépések, melyhez csak az ajtó elé kell állnunk, vagy karakterekkel való párbeszédek, amihez értelemszerűen közel kell lennünk hozzájuk. A nyersanyagok gyűjtése is különféle pontokon kezdhető el, ahol logikus módon találhatjuk meg és gyűjthetjük össze őket.

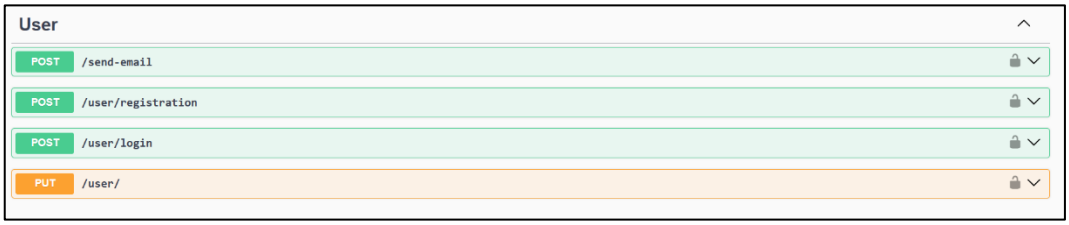
3.8. A játék célja

A játék célja az, hogy felfedezzük a fő történetet és eljussunk annak végére, ahol karakterünk méltó jutalmat kap a megmentett barátoktól és lakosoktól. Ehhez folyamatosan fejlesztenünk kell fegyverünket és harcolnunk kell a még erősebb ellenfelekel, akik az egyre mélyebbre vezető erdőben várnak ránk.

4. Backend

4.1. Auth nélküli végpontok

4.1.1. Felhasználói végpontok



A userRoutes.js - ban találhatók a felhasználóval kapcsolatos végpontok, amelyek a felhasználók regisztrációját, bejelentkezését és a felhasználói adatok módosítását kezeli.

A **POST user/registration** végpont fogadja és kezeli az új felhasználók regisztrációját. A felhasználó által megadott adatok alapján létrehoz egy új felhasználót az adatbázisban, majd, ha a kliens által beküldött adat minden validáción átment, visszaküld egy üzenetet a sikeres regisztrációról 201 státusz kóddal, különben egy 400 státuszkódot, és egy hibaüzenetet ad vissza.

A **POST user/login** végpont a felhasználó bejelentkezését végzi. Ellenőrzi az általa megadott e-mail címet és jelszót, majd, ha megegyezik az adatbázisban tároltakkal, generál egy JWT (JSON Web Token) tokent és visszaküldi azt a kliensnek. Ha a bejelentkezés sikeres, a státuszkód ’200 OK’.

A **PUT /user** végpont a felhasználói adatok módosítását végzi. A kérést a felhasználó által megadott token alapján azonosítja, majd frissíti az adatokat az adatbázisban. Ha minden sikerült, visszaküld egy visszaigazolást a sikeres módosításról a 200 OK státusszal.

A végpontok figyelmet fordítanak az adatok validálására és a hibakezelésre, például ellenőrzik a jelszó helyességét és figyelembe veszik a mezők egyediségét is. Az alkalmazás JWT tokent használ azonosításra és hitelesítésre, és bcrypt segítségével biztonságosan tárolja a jelszavakat hash formátumban.

4.1.2. Middleware hibákra

Az errorController.js egy olyan hibák szempontjából általános middleware, amely az alkalmazásban bekövetkezett hibák kezeléséért felel. A fájl egyetlen függvényt, az errorHandler-t tartalmazza.

Ez a middleware lényegében egy egyszerű hibakezelő, amely a hibát visszaküldi a kliensnek a válasz részeként, egy 500-as státuszkóddal és egy JSON objektummal, ami tartalmazza a hibaüzenetet ("Valami hiba történt, próbáld meg később!"). Bármely kérés során, ha valami olyan hiba következne be, ami nincs lekezelve, ez a middleware elkapja azokat, és jelzi a felhasználónak, az fentebb említett hibaüzenettel és státuszkóddal.

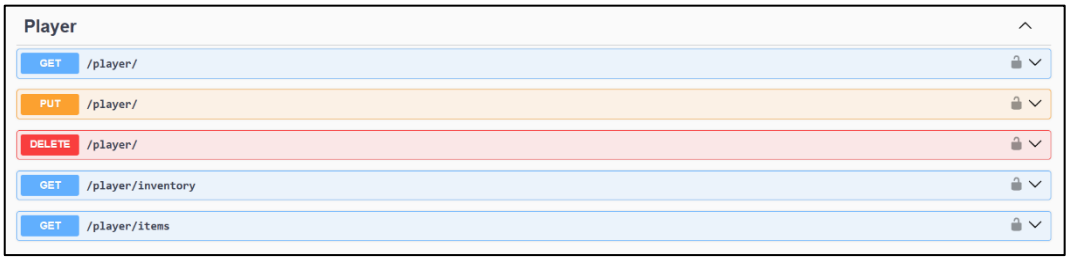
```
const errorHandler = (err, req, res, next) => {
    res.status(500).json({ message: "Valami hiba történt, próbáld meg később!" });
    console.log(err);
};

module.exports = { errorHandler };
```

4.2. Autentikációs végpontok

Minden végpont csak JSON formátumban kommunikál, azaz csak ilyen formátumú kéréseket fogad, és csak ilyen formátumban is válaszol. A REST-API architektúrának megfelelően nincsen server session, minden kérés egyedinek, és teljesen különbözőnek tekint a szerver.

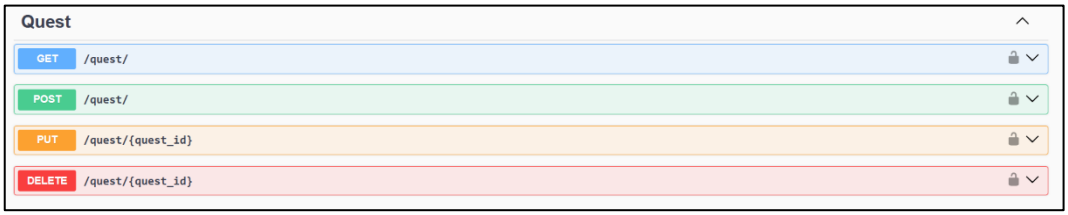
4.2.1. Játékos végpontjai



A playerController.js a játékosokkal kapcsolatos végpontokat tartalmazza, amelyek a játékos adatait kezeli az alkalmazásban.

A **GET /player** végpont visszaadja a játékos adatait, beleértve, melyik blokkon tartózkodik, eszközöket, életerejét, pénzét, nevét. Ezek az adatok segítenek a játékoshoz közvetlenül, és a játékost leíró adatok.
A **GET /inventory** végpont segítségével a játékos az aktuális készletét kaphatja meg, beleértve az egyes elemek mennyiségét is. Ez hasznos lehet, amikor a játékosnak ellenőriznie kell, mennyi erőforrása van rendelkezésre a továbbiakban.
A **PUT /player** végpont lehetővé teszi a játékos adatainak frissítését, például a nevét, életerejét, pénzét stb. Emellett lehetőséget biztosít az alapanyagainak mennyiségének frissítésére.
A **DELTE /player** végpont lehetővé teszi a játékos törlését az adatbázisból.

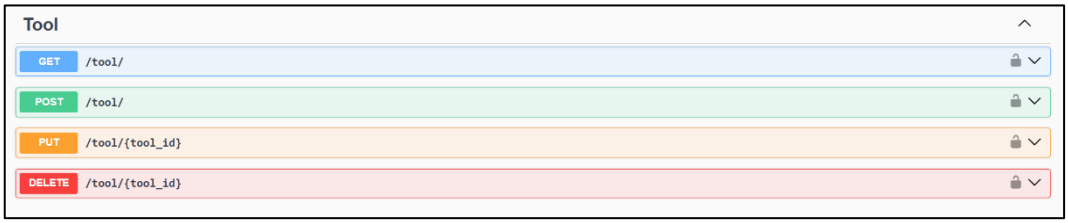
4.2.2. Küldetés végpontok



A questController.js az alkalmazás küldetéseivel kapcsolatos kéréseket kezeli.
A **GET /quest** végpont a játékosok küldetéseinek lekérdezését kezeli. Lekérdezi a játékoshoz tartozó összes küldetést, szűrési feltételekkel biztosítva.
A **POST /quest** végpont új küldetés felvételét kezeli a játékoshoz. A kliens által küldött adatok alapján létrehoz egy új rekordot a QuestStat táblában, 201 – as státuszkóddal jelezve a sikeres műveletet.
A **PUT /quest/:quest_id** végpont a küldetés állapotának módosítását kezeli. A játékos által elvégzett módosításokat alapul véve frissíti a küldetés állapotát a QuestStat táblában.
A **DELETE /quest/quest_id** végpont egy a játékoshoz tartozó küldetés törlését végzi el. Ellenőrzi, hogy a küldetés létezik-e, ha igen,

törli azt a QuestStat táblából, ha nincs ilyen küldetés 404 státuszkáddal tér vissza.

4.2.3. Eszköz végpontok



A toolController.js a játékosokhoz kapcsolódó eszközöket kezeli egy adatbázisban.

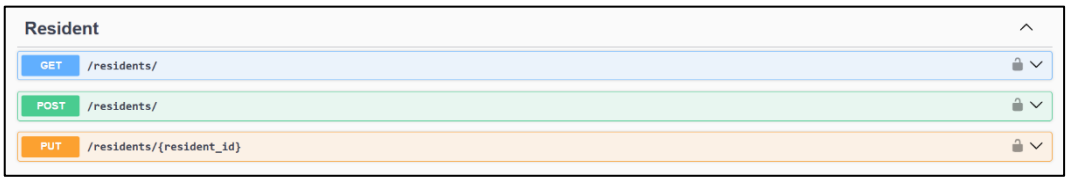
A **GET /tool** végpont a játékoshoz tartozó eszközöket lekérdezi az adatbázisból. A játékos azonosítóját a req.token.id segítségével kapja meg. A visszatérő adatokat 200-as státuszkóddal küldi vissza JSON formátumban.

A **POST /tool** végpont új eszközt hoz létre a játékos számára az adatbázisban. A játékos azonosítóját a req.token.id segítségével kapja meg, míg az új eszköz típusát a req.body.tool_type_id-ből olvassa ki. Sikeres létrehozás esetén 201-es státuszkóddal és egy üzenettel válaszol.

A **PUT /tool/:tool_id** végpont frissíti egy meglévő eszköz típusát az adatbázisban. A frissítendő eszköz azonosítóját a req.params.tool_id-ből kapja meg, míg az új típust a req.body.tool_type_id-ből olvassa ki. A frissítés után 200-as státuszkóddal és egy üzenettel válaszol.

A **DELETE /tool/:tool_id** végpont töröl egy meglévő eszközt az adatbázisból. Az eszköz azonosítóját a req.params.tool_id-ből kapja meg. Ha az eszköz nem létezik, akkor 404-es státuszkóddal válaszol "Ilyen fegyver nem létezik!" üzenettel, egyébként pedig 200-as státuszkóddal és egy sikeres törlési üzenettel.

4.2.4. Lakos végpontok



A residentController.js a játék lakosaival kapcsolatos műveletek végrehajtásáért felel.

A **GET /residents** végpont felelős a játékoshoz tartozó összes lakos lekérdezéséért az adatbázisból. Itt is szintén, mint néhány másik végpontnál, lehetőséget biztosítunk különböző szűrésekre. Közülük a legfontosabb a blockX és a blockY mezők alapján szűrés a frontend számára, hogy egy adott blokkon lévő lakosokat le tudjuk kérdezni.

A **POST /resident** végpont új lakos létrehozásáért felelős az adatbázisban. Fogadja az új lakos adatait, például az elhelyezkedést, a nevet, a foglalkozást stb. Majd ezeket az adatokat felhasználva létrehoz egy új lakost az adatbázisban. Válaszként visszaküldi a sikeres művelet megerősítését.

A **PUT /resident/:resident_id** függvény a meglévő lakos adatainak frissítését végzi. Fogadja az új adatokat, amelyekkel frissíteni szeretnénk a lakost, és az azonosító alapján megtalálja és frissíti a megfelelő lakost az adatbázisban. A válaszként visszaküldött üzenet jelzi a módosítás sikerességét.

4.2.5. Piac végpontok

A **GET /market** végpont az összes ajánlat lekérdezését végzi el, figyelembe véve a kérésben megadott szűrőket.

A **GET /market/playerOffer** végpont a bejelentkezett játékoshoz tartozó összes ajánlatot kéri le.

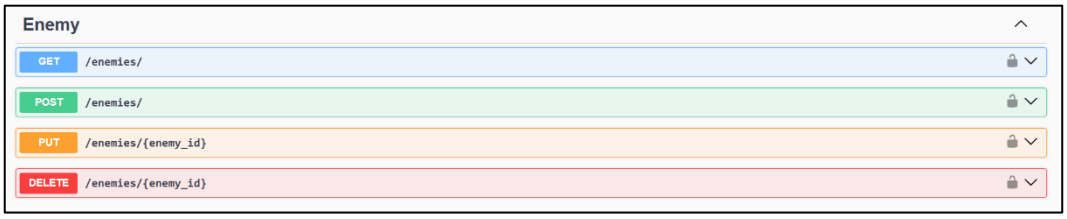
A **POST /market** végpont létrehoz egy új ajánlatot a beérkezett adatok alapján, és sikeres létrehozás esetén 204 – es státuszkóddal tér vissza.

A **PUT /market/:offer_id** végpont módosítja egy meglévő ajánlat adatait az ajánlat azonosítója alapján.

A **DELETE /market/:offer_id** végpont törli az adott. Ha az ajánlat nem található, akkor 404-es hibaüzenetet ad vissza. A játékos csak a saját ajánlatait tudja törölni.

A **PUT /market/buy/:offer_id** végponttal egy adott ajánlatot tudunk megvásárolni. A paraméterben kapott offer_id alapján lekérdezi az ajánlatot, és módosítja a vevő és a tulajdonos alapanyagait megfelelően. Ha a vásárlónak nincsen egy olyan alapanyaga, amivel meg szeretne vásárolni egy adott ajánlatot, valamint, ha a játékosnak nincs elég alapanyaga, hogy megvásárolja, mindkét esetben egy 400 – as státuszkóddal jelezzük a felhasználó számára a hibát a megfelelő hibaüzenettel.

4.2.6. Ellenfél végpontok



Az enemyController.js az ellenségek kezelését végzi a játékban.

A **GET /enemies** végpont az adott játkoshoz tartozó összes ellenség lekérdezését végzi el. Csak úgy, mint a lakosoknál, itt is lehet szűrni, és szintén a leggyakrabban használt szűrési feltétel az, hogy egy adott blokk koordinátái alapján kérdezi le az összes ellenséget, ami a játékoshoz van rendelve.

A **POST /enemy** végpont új ellenségek hozzáadását végzi az adatbázishoz. A kliens oldalról érkező adatok alapján létrehozza az új

ellenséget a játékoshoz. A válasz tartalmazza a sikeres művelet üzenetét.

A **PUT /enemy/:enemy_id** végpont meglévő ellenségek módosítását végzi az adatbázisban. Az enemy_id alapján azonosítja az adott ellenséget, majd frissíti az adatokat a kliens oldalról érkező új adatokkal. A válasz tartalmazza a sikeres művelet üzenetét.

A **DELETE /enemy/:enemy_id** végpont az ellenségek törlését végzi az adatbázisból. Az enemy_id alapján azonosítja az adott ellenséget, majd törli azt az adatbázisból. Ha az ellenség nem létezik, akkor hibaüzenetet küld vissza. A válasz tartalmazza a sikeres művelet üzenetét.

4.3. Middleware

4.3.1. Hitelesítés

A userAuth middleware feladata a felhasználók hitelesítése és azoknak az elérésének korlátozása, akik nincsenek bejelentkezve. A middleware feladata, hogy ellenőrizze a kérés fejlécében található JWT tokent, és megerősítse annak érvényességét a szerveren.

Ha a kérésben nincs token, vagy a token érvénytelen, akkor a middleware visszautasítja a kérést megfelelő státuszkóddal és üzenettel.

```
const userAuth = async (req, res, next) => {
  try {
    const token = req.headers.authorization?.split(" ")[1];
    if (!token) {
      return res.status(401).json({ message: "Jelentkezz be!" });
    }
    jwt.verify(token, process.env.SECRET_KEY, {}, (err, decodedToken) => {
      if (err) {
        return res.status(403).json({ message: "Nincs jogod a következőre!" });
      } else {
        req.token = decodedToken;
        next();
      }
    });
  } catch (error) {
    next(error);
  }
};
```

Amennyiben a token érvényes, a middleware továbbítja a kérést a következő lépéshez, miközben hozzáférést biztosít a dekódolt token adataihoz a req.token segítségével.

Ezáltal a middleware növeli a biztonságot, biztosítva, hogy csak az azonosított felhasználók érjék el a játékot.

4.3.2. Játkos Kezdő állapot

A játékosok amikor beregisztrál az adatbázis nem tartalmazza azokat az adatokat, amelyekkel még a játékos elkezdhet játszani, ilyen közöttük például, hogy a játkos inventory táblája üres, illetve nincs elindítva a fő küldetés szál. Ennek megfelelően létrehoztunk egy olyan életciklus horgot amikor a játékos regisztrál, a horog automatikusan feltölti a játékoshoz kapcsolódó adatokat az adatbázisba.

Három egyszerű bulkCreate utasítás megy ki az adatbázis felé, az inventory, residents, és a quest_stat táblákat tölti fel, az inventoryt feltölti a játékban lévő összes alapanyaggal, természetesen nulla lesz mindegyikből. Egy kezdeti lakost ad hozzá, valamint a játékban lévő összes küldetésnek felveszi a statisztikáját.

4.3.3. Szűrés middleware

Sok lekérdezés igényel szűrési funkciót mint például lakosok, szörnyek szűrése mezőkön, azonban mindegyik modell más és más tulajdonságokkal rendelkezik, így önmagában a req.query paramétert átadni változatlanul nem garantálja, hogy mindegyik lekérdezés jól fog működni.

Erre a célra hoztuk létre egy olyan szűréseket lekezelő middlewaret ami a modellnek megfelelően kiszűri azokat a tulajdonságokat a query objektumból amik nem része a modellnek, valamint ha nem csak egyenlőség szerint hanem más operátor mint például kisebb, nagyobb szerint akarunk szűrni azt is lekezeli a middlweare és olyan formátumban adja át a megfelelő kontrollernek, ahogyan a sequelize szerint szintaktikailag helyes.

5. Frontend – Fő oldal

5.1. Bevezetés

A weblap a Vue.js egy progresszív, reaktív adatokat tartalmazó, és viszonylag rugalmas keretrendszerében készült. Azért ezt a megoldást választottuk, mert úgy láttuk, hogy rövid időn belül elsajátítható, a dokumentációja terjedelmes, informatív, és könnyen áttekinthető. A fő oldal egy Single Page (SPA) oldal, a Vue.js által nyújtott routingot használtuk, és ennek megfelelően definiáltunk négy útvonalat, amelyekhez mindegyikhez tartozik egy komponens.

5.2. Komponensek

Minden egyes komponens egy oldalt fog magába foglalni, és mindegyikük saját logikáját tartalmazza. Az App.vue, vagyis fő komponensünk, tartalmazza a navigációs felületet, amely mindig megjelenik. Alatta található a RouterView, amely a megfelelő útvonalak szerint jeleníti meg a hozzájuk tartozó komponenseket.

A Home komponens a főoldalt tartalmazza. Ide került a bejelentkezési űrlap is, amellyel be tudunk jelentkezni.

A Registration komponens a regisztrációs űrlapot tartalmazza, ahol a felhasználó létrehozhat egy profilt a játékhoz.

Természetesen, ha bármilyen észrevétel lenne a játék vagy akár a weboldal során, megadtuk a lehetőséget, hogy a felhasználó bejelentsen egy megtapasztalt problémát. Ezt a funkciót tartalmazza a Report komponensünk.

Habár nem sok szerepe van, létrehoztunk egy Not Found Page komponenst is, hogy jelezzük a felhasználónak, hogy az adott útvonal nem létezik.

5.3. Szerviz

Az űrlapok kezelésére létrehoztunk egy különálló szervizt is, amely axios kérésekkel elküldi az űrlap adatait a backend felé. Természetesen a kérés elküldése esetén az adatok végigmennek a kliensoldali validáción, és ha mégsem megfelelő az adat, akkor a backendtől kapott hibaüzenetet megjelenítjük az űrlap alatt. A regisztrációnál, ha sikeres, akkor az űrlap alatt megjelenik, hogy Sikeres regisztráció. A bejelntkezésnél ha sikeres akkor átnavigál a játék oldalra.

6. Frontend - Játék

6.1. A játék

A játék fejlesztése során nehézséget okozott számunkra, hogy milyen technológiát válasszunk a programozáshoz, de végül úgy döntöttünk, hogy a Canvas 2D API-t fogjuk felhasználni natív JavaScript környezetben. Fontolóra vettük a WebGL API-t is, de mivel annak megtanulása sok időt igényelne, elvetettük ezt az ötletet, és inkább az előbbi mellett döntöttünk.

6.2. Programozási modell

A tervezés folyamán felismertük, hogy a játék programozásához teljesen más programozási modellt és megközelítést kell alkalmaznunk. A funkcionális, és az objektum orientált programozási modellt raktuk össze. A játék állapotát, mint például a játékost, az entitásokat és az akadályokat, egy osztályhierarchián belül helyeztük el, ahol az entitásokat, vagyis a játékban lévő élőlényeket egy alap Entity osztályból származtatjuk. Más élőlények által nem áthidalható akadályok, mint például a Barrier és a Collision, nem rendelkeznek közös alap osztállyal, mivel bár hasonló funkcionalitásokkal rendelkeznek, az ütközésdetektálás miatt mindegyik más algoritmust használ.

A játékban szinte minden a játékos interakciójától függ, mint például kattintások, másik blokkra való átlépés, bizonyos helyekre való lépés, lakosokkal való interakció, vagy paneleken való kattintások. Ezen interakciók kezelésére létrehoztunk egy View és egy Controller komponenst/komponenseket. A View tartalmazza azokat a függvényeket, amelyek a játékos nézetét módosítják, például panel eltüntetése vagy megjelenítése, küldetések panelének módosítása, inventory és eszköz panel megjelenítése kattintásra. Emellett a szükség esetén más komponenseknek is átadhat paramétereket, és az if feltételek alapján határozhatja meg, mit jelenítsen meg a felhasználónak.

A Controllerek pedig olyan függvényeket tartalmaznak, amelyek a játék állapotának változását kezelik.

6.3. Ciklus és teljesítmény

A játéknak két fő komponense van: az egyik a játék ciklus, a másik pedig a Story controller.

A játék ciklus a teljesítmény szempontjából a legfontosabb, mivel minden tickben ismétlődik, és magában foglalja a canvas vászon újrarajzolását, az entitások új pozíciójának beállítását, az ütközés detektálását, valamint a panelokhoz való közelség érzékelését a játékoshoz viszonyítva.

A Story controller akkor játszik szerepet, amikor a játékos interakcióba lép a lakosokkal. Ilyenkor az isInConversation boolean változóját igazra állítjuk, ami megállítja a fő ciklust, így a Story controller veszi át az irányítást a játék felett. Ez lehetővé teszi, hogy a fő küldetés szál egy-egy pontján a párbeszédek alatt a lakosokat áthelyezzük a vászonon, vagy akár egy másik sprite-ot cseréljünk ki az éppen aktuális helyett, így még élethűbb hatást érve el a játékos szemszögéből.

Mivel a játék böngészőben fut, arra törekedtünk, hogy optimalizáljuk a játék teljesítményét, különösen a játék ciklusát, hogy minél szélesebb körű eszközön élvezhető legyen. Ehhez a requestAnimationFrame beépített JavaScript animációs függvényét használtuk, egy kis kiegészítéssel. Ez a függvény automatikusan igazodik a kijelző frissítési rátájához, és csak akkor hívja meg a következő framet, amikor a böngésző erre készen áll.

Az FPS optimalizálásához a fő JavaScript fájlban, ahol az 'animate' nevű függvény található, a requestAnimationFrame függvény első hívásával indítja el a ciklust. A requestAnimationFrame függvény vár egy callback függvényt, amely egy olyan paramétert kap, ami azt jelzi, hogy a játék hány ezredmásodperccel ezelőtt

```
let previousTime = 0
  const animate = (timeStamp) => {
    const deltaTime = timeStamp - previousTime
    previousTime = timeStamp
    game.gameLoop(deltaTime)
    requestAnimationFrame(animate)
  }
  animate(0)
}
```

kezdődött. Ennek segítségével kiszámítjuk a 'deltaTime'-ot, azaz, hogy az előző és a jelenlegi frame között mennyi idő telt el, és ezt a 'deltaTime'-ot minden frame híváskor átadjuk a játék ciklusnak paraméterként.

A játék osztálynak a timer tulajdonságon keresztül, amely egy köztes jelenlegi időt tárol, vezérli az intervallumonkénti frissítést. Csak akkor indítjuk el a játék ciklusát az adott képkockában, ha a timer eléri az intervallumot, amely a képkockák másodpercenkénti számát jelenti ezredmásodpercben. A ciklus végén, ha nem engedélyeztük a frissítést, növeljük a timer értékét a deltaTime nevű paraméterrel. Ha a timer értéke nagyobb, mint az intervallum értéke, a ciklus lefut, és a timer értékét nullára állítjuk.

```
this.fps = 60
this.timer = 0
this.interval = 1000 / this.fps
```

6.4. Entitás modell

A könnyebb kezelhetőség érdekében az entitásokhoz, hasonlóan a mai modern játékokhoz, hozzárendeltünk egy ütközésdetektáló dobozt (hitboxot) és egy ütközési sugarat. Ezeket leszámítva, még van az entitásoknak a saját sprite sheet eredeti méretei, szélessége és magassága, amelyek nem láthatók az ábrán.

6.5. Ütközés detektálás

Az ütközésdetektálás szintén az egyik legnehezebb feladat volt. A tervezés során olyan algoritmusokat próbáltunk alkalmazni, amelyek hatékonyak és viszonylag jól absztraktálhatók, hogy ne terheljék le a számítógépet.

A játékban kétféle akadálytípust alkalmazunk, bár voltak terveink másféle akadályokkal is.

Az algoritmusok mellett fontos megemlíteni egy másik, a modern játékokban viszonylag kedvelt módszert, nevezetesen az AABB-t (Axis Aligned Bounding Box). Ennek lényege, hogy minden játékbeli objektumot, entitást vagy akadályt egy tengelyek mentén elhelyezkedő téglalapba helyezünk, és csak akkor alkalmazzuk az ütközés detektálására szolgáló algoritmust, ha a két vizsgált objektum téglalapjának kerete érintkezik egymást, azaz, ha az egyik objektum kerete tartalmazza a másikét. Például, ha a játékban 10 szörny és további 5 vagy 6 akadály van jelen, akkor az összes objektum száma 21, és így rengeteg számítástól kímélhetjük meg a számítógépet. Csak egyszerű logikai feltételt kell ellenőrizni az ütközés meglétének vizsgálatakor, és ha nincs ütközés, azonnal a következő objektumra léphetünk.

A játék objektumait kétfajtára osztottuk: kör alakúakra és egyenes vonal alakúakra. Mielőtt bármilyen számítást végeznénk rajtuk, az AABB (Axis-Aligned Bounding Box) ellenőrzi őket.

A **kör-kör közötti ütközés** detektálás során kiszámítjuk az ütközési sugarak közötti távolságot, és ha ez kisebb, mint a két objektum közötti távolság (a körök középpontjainak távolsága), akkor az ütközés megtörtént.

A **kör és a vonal közötti ütközést** a koordinátageometria segítségével számítjuk ki. Meghatározzuk a játékos pontjának és a vonal közötti távolságot, majd, ha ez kisebb, mint a vonal szélessége, akkor ütközés

történik. A vonal szélessége egy tulajdonság, amely meghatározza, hogy az adott két pont közötti vonal mekkora kiterjedésű.

6.6. Panelek

A játékos főként különböző paneleken keresztül lép interakcióba a játékelemekkel. Ezek magukban foglalják a lakosokkal való párbeszédet, az eszköztár elérését, a blokkok közötti navigációt, az alapanyaggyűjtést. Emellett a paneleken keresztül jelennek meg a lakosok és a szörnyek statisztikái, amikor az egérrel rájuk mutatunk.

Globális panelek azok, melyek akkor jelennek meg, amikor kattintás történik, vagy más DOM esemény bekövetkezik, míg **játékbeli panelek** csak akkor láthatók, ha a játékos egy bizonyos pont közelében tartózkodik.

Mivel a paneleknek különböző HTML struktúrájuk van, és sokféle funkcionalitással és viselkedéssel rendelkezhetnek, ezért nem egyetlen alap Panel osztályt hoztunk létre, amelyből a többi örökölhetne. Ehelyett létrehoztunk egy önálló Panel osztályt, amely kizárólag a játékban használt paneleket kezeli. Ezeket a paneleket tárolnunk kell a megfelelő blokkban valamilyen formában, hogy a játék ciklusában érzékelni tudjuk, hogy a játékos közel van-e hozzájuk vagy sem. A játékban lévő paneleket egy listában tároljuk az adott blokk példányában.

A PanelView osztályon belül létrehoztunk két általunk definiált eseményt, PanelShowed és PanelHide eseményt, amit mindig akkor hívunk meg az adott panel HTML elemen, amikor bármelyik panel hozzáadódik a DOM-hoz. Mindegyik panelnek van egy ID tulajdonsága, amit mindig hozzácsatolunk az esemény argumentumhoz, így az eseményfigyelőben egy if feltétellel meg tudjuk határozni, hogy melyik panel lett éppen megjelenítve a játékos számára, és ha szükséges, egyéb kódot, módosítást is végrehajtunk.

7. Adatbázis

7.1. Bevezetés

12-1 Adatbázis diagram

Az egyik legfontosabb aspektusa egy játékszoftver működésének a megbízható és alaposan megtervezett adatbázis. Ha nincs megfelelő tervezés, akkor a szoftver karbantartása bonyolultabbá válhat, és a bővíthetőség is nehezebben megvalósíthatóvá válik. Ezért kiemelten fontos, hogy a fejlesztés során különös figyelmet fordítsunk az adatbázis tervezésére és megvalósítására, hogy a szoftver hosszú távon is hatékonyan működjön és könnyen karbantartható legyen.

Az adatbázisunk létrehozásához és kezeléséhez a dbForge Studio program által kínált adatbáziskezelő megoldásait választottuk. Ennek oka, hogy úgy véljük, ezáltal rendkívül egyszerűvé válik a fejlesztés bármelyik szakaszában. A dbForge Studio által nyújtott eszközök és funkciók segítségével hatékonyan tudjuk tervezni, létrehozni és karbantartani az adatbázisunkat, ami kulcsfontosságú a szoftverünk stabilitásához és kiterjeszthetőségéhez.

7.2. MySQL

Egy játékban nagyon fontos az adatok integritása és típusa, ezért is választottunk egy relációs adatbázis kezelőt. (MySQL)

Adatbázisunk tervezésénél mindenképpen úgy gondoltuk, hogy szinte minden a játékos táblához köthető, bármilyen kapcsolatról is legyen szó. Az adatintegritás megőrzése végett, a játékos és a hozzá tartozó entitások (szörnyek, eszközök, falusiak, küldetések) között csináltunk egy kapcsolótáblát a küldetések és a falusiak kivételével, amely tartalmazza a játékoshoz tartozó tényleges entitásokat. Ez mindegyike egy több-több kapcsolat, amely áll az előbb említett kapcsolótáblából és a kapcsolat entitás oldalán mindegyik entitás tartamaz egy típus táblát, amely a játékban létező az adott entitáshoz tartozó típust jellemzi. (különböző

szörnyek, eszközök) A küldetéseket kapcsolótábla helyett, kategóriákba soroltuk és ezt a küldetések tábla category mezőjével jellemezzük.

7.3. Kapcsolat a backend-el

A backend oldalon az adatbázishoz történő kapcsolódásra a sequelize keretrendszert választottuk. Minden entitást egy modellel jellemeztünk. Amikor a felhasználó beregisztrál, akkor szükséges, hogy az adatbázisba bekerüljenek olyan kezdőadatok, amelyekkel már a játékos el tud kezdeni játszani. Ez a játékos modell afterCreate életciklushorgában van megvalósítva, amely miután a játékos beregisztrált, létrehozza a megfelelő kapcsolatokat, valamint feltölti a szükséges táblákat a szükséges adatokkal.

8. Tesztelés

8.1. Frontend tesztelés Cypress használatával

12-2 Frontend teszt

A Cypress három fő részt tesztel a frontenden.

Az első blokk az fő oldalt teszteli. Az első teszt azt ellenőrzi, hogy a fő oldal megfelelően megjelenik-e. A második azt vizsgálja, hogy van-e form a főoldalon, ahol a felhasználó bejelentkezhet. A harmadik azt ellenőrzi, hogy a helyes bejelentkezési adatokkal bejelentkezés után a felhasználó átirányítódik-e a játék oldalára. A negyedik pedig azt ellenőrzi, hogy helytelen bejelentkezési adatokkal a felhasználó nem tud-e bejelentkezni, és marad-e a fő oldalon.

A második blokk a regisztrációs oldalt teszteli. Az első teszt ellenőrzi, hogy a regisztrációs oldal megjelenik-e megfelelően. A második azt vizsgálja, hogy van-e form a regisztrációs oldalon, ahol a felhasználó regisztrálhat. A harmadik azt ellenőrzi, hogy a helyes regisztrációs adatok megadása után a felhasználó sikeresen regisztrál-e. A negyedik pedig azt ellenőrzi, hogy helytelen regisztrációs adatokkal a felhasználó nem tud-e regisztrálni.

A harmadik blokk a hibajelentés küldésének oldalát teszteli. Az első teszt ellenőrzi, hogy a jelentés küldésének oldala megfelelően megjelenik-e. A második azt vizsgálja, hogy van-e form az oldalon, ahol a felhasználó hibajelentést küldhet. A harmadik azt ellenőrzi, hogy a helyes adatok megadása után a felhasználó sikeresen elküldheti-e a jelentést.

8.2. Backend tesztelés Cypress használatával

12-3 Backend teszt

Az API-cím és az inicializált token változók előzetesen definiáltak, hogy a tesztek hozzáférjenek az API-hoz és az azonosításhoz.

A Cypress öt fő részt tesztel a backenden.

Az első blokk a játékos regisztrációját ellenőrzi. Ez a teszt egy POST kérést küld a /user/registration végpontra a megadott felhasználónévvel, email

címmel és jelszóval. Az elvárt válasz státuszkódja 201, és a válaszüzenet tartalmazza a "Sikeres regisztráció!" üzenetet.
A következő blokk a játékos bejelentkezését ellenőrzi. Egy újabb POST kérést indít a /user/login végponton a regisztráció során megadott e-mail cím és jelszó alapján. Az elvárt válasz 200-as státuszkódot kell tartalmazzon, valamint a "Sikeres bejelentkezés!" üzenetet.
A harmadik blokk a játékos adatainak frissítését ellenőrzi. Ez egy PUT kérést küld a /player/ végpontra a játékos adataival és az azonosításhoz szükséges tokennel. Az elvárt válasz 200-as státuszkódot és a "Sikeres módosítás!" üzenetet tartalmazza.
A negyedik blokk a játékos adatainak lekérését ellenőrzi. Egy GET kérést indít a /player/ végpontra az azonosításhoz szükséges tokennel. Az elvárt válasz 200-as státuszkódot kell tartalmazzon, és a válasz adatobjektumának meg kell felelnie a teszt során beállított játékos adatainak.
Az ötödik blokk a játékos adatainak törlését ellenőrzi. Ez egy DELETE kérést küld a /player/ végpontra az azonosításhoz szükséges tokennel. Az elvárt válasz 200-as státuszkódot és a "Sikeres törlés!" üzenetet kell tartalmazza.

8.3. Swagger

A Forgotten Kingdom-hoz készült egy Swagger dokumentáció is, amit a swagger-autogen generált. Ez a dokumentum a szolgáltatásokhoz kapcsolódó útvonalakat és azok leírását tartalmazza, melyeket a kliensek használhatnak a kommunikációhoz a szerverrel.
Ide kattinva megtekinthető a Swagger dokumentáció.

9. A játék felépítése

9.1. Az alap ötlet

A játékunk alapját megtervezni nem volt egyszerű feladat. Először is azon tűnődtünk, hogy egy menedzselő játék lenne az ideális választás. A gondolat az volt, hogy könnyen kezelhető legyen, csak egér lenne szükséges, és folyamatosan lehetne épületeket fejleszteni, valamint egyéb dolgokat kezelni.

Az ötlet továbbfejlesztése során azonban úgy döntöttünk, hogy a játék teljes egészében a középkorban játszódjon. Elképzeltük, hogy egy királyság várát és annak épületeit lehet fejleszteni, valamint a lakosok irányítására és utasítására is lehetőség lenne.

Azonban később átgondolva ezt az ötletet, arra a következtetésre jutottunk, hogy már túl sok hasonló játék létezik, és nincs értelme csak egy újabb ugyanolyan alkotást létrehozni. Ezért úgy döntöttünk, hogy bár megtartjuk a középkori környezetet, de elvetjük a menedzselési részt. Ehelyett egy főhőst képzeltünk el egy nyílt világban, aki szörnyekkel vív meg és kalandokat él át.

9.2. Fő küldetés

Az alap ötletünk már megszilárdult, de még mindig voltak területek, amiket át kellett gondolnunk. Egy mozgatható karakter egy királyságban önmagában még nem hangzott túl izgalmasnak.

Így hát arra jutottunk, hogy szükségünk lesz egy érdekfeszítő történetre, amelyet a felhasználó átélhet, miközben irányítja a kis karakterét.

```
Elfeledett Királyság – Első lépések (legelső belépés a játékba)

----------Beszélj az ismeretlen árussal----------

Mesélő: Hősünk, vándorlásai során egy új, ismeretlen vidékre téved. Elhagyatott épületek között járva figyelmesen nézelődik, s próbálja kitalálni, mi történhetett ezen a helyen.

Játékos: *Ez a hely... talán egy királyság lehetett? De most... mégis hol vannak az emberek?*

Mesélő: Nagy bizonytalanságában, végül megpillant valakit a távolban.

Mesélő: Egy idős kereskedő az, ki árucikkekkel a háta mögött reméli, hogy megéli a holnapot.

Mesélő: Harcosunk közelebb kerül hozzá, a kereskedő felé fordul és megkönnyebbülten tapasztalja, hogy végre társaságot talált.
```

Ennek érdekében nekiláttunk egy fő történetszál forgatókönyvének megírásához. Ez a történet adna életet a játéknak, és izgalmas kalandokat kínálna a játékosnak, miközben felfedezi és meghódítja a királyságot.

9.3. Mellékszereplők

Ha már egy főszereplőt irányítunk, akkor neki szüksége lesz más karakterekre, akikkel interakcióba léphet. Kigondoltunk pár fontos szereplőt a történetbe, akikkel a főhősünk beszélgethet, barátkozhat. Egy királyságban mindenképp lennie kell legalább egy kovácsnak és egy kereskedőnek. Szerepet kapott továbbá egy titokzatos mesélő, akit a fő történet megismertetésével bíztunk meg. Hősünk találkozni fog még lovagokkal, szörnyvadásszal, mágussal, favágóval, farmerrel, horgásszal és végül egy igazi boszorkánnyal is.

9.4. Harc a dicsőségért!

A történetünkből, már csak a küzdelem hiányzott. A forgatókönyv megírása közben arra gondoltunk, hogy a karakterünknek lehetnének ellenfelei is, akikkel harcolhat, jutalmakért cserébe.

Végül ezt a játékmenetet is elképzelve, megszületett a végső változata a fő küldetésnek.

A játékos egy elfeledett királyságban találja magát, aminek lakosait

szörnyek rabolták el. Főhősünknek legfontosabb feladata az lesz, hogy a szörnyek leküzdésével felszabadítsa a királyság népét is visszaállítsa annak régi dicső fényét, miközben barátokat, társakat szerez.

9.5. A nyílt világ darabkái

Miközben a forgatókönyvet írtuk, a pálya megtervezésére is gondolnunk kellett. Más játékok megvalósításait nézegetve azt láttuk, hogy sajnos a nyílt világ ötletét el kell vetnünk, mivel túl sok idő lenne a felépítése.
Hosszas ötletelés és tervezgetés után végül rájöttünk a megoldásra. A nyílt világot részben megtartottuk, viszont darabokra szettük a királyságot.
A játékos, csak a világ főbb részeire jut el a karakterével, így például kihagyjuk az olyan nyílt világba tartozó elemeket, mint az adott épülettől épületig vezető utak. Ezzel jóval több elhelyezendő „láthatatlan falat” és pálya részeket megspórolva.

9.6. Izometrikus nézet

A világunk első felül nézetes épületének elkészült változatát, ismét egy újra gondolás követte. Mivel darabokra osztottuk a nyílt világot, így már nem igazán illett bele a játékba a felül nézetes stílus.
Végül, ötletek után kutatva találtunk olyan játékokat, amik úgynevezett izometrikus nézetben készültek el, így mi is ezt a megvalósítást választottuk.

A mi esetünkben is az izometrikus nézet lényegében azt jelenti, hogy az adott játékot még mindig 2D-ben játsszuk, de magát a világot 3D-s formájában látja a játékos.
Így születettek meg a játékunk végleges formái, az „izometrikus blokkok”.

9.7. Pixel Art

Végezetül, a játéknak szüksége van egy művészeti témára is, amiben elkészülhet a világ. A választásunk pedig a pixeles grafikára esett, vagyis a pixel art-ra.

Viszonylag sokkal egyszerűbb elkészíteni egy ilyen játékot annál, mintha valósághű ábrázolással próbálkoznánk, de mégis ugyan úgy megvan ennek a stílusnak is a varázsa.

9.8. Sprite sheet

Ha mozgatni szeretnénk egy karaktert, akkor az nem elég, hogy csak annak képét irányítjuk. Szükség lesz mozgás, vágás (harc) és más egyéb animációkra.

Ennek megvalósításához az úgynevezett sprite sheet elkészítésének megoldását választottuk. Ez a módszer lényegében úgy működik, hogy adott egy olyan kép, amely több kisebb képet (sprite-ot) tartalmaz egyetlen nagyobb képen belül. A sprite-ok minden darabja egy cselekvést hoz létre. Például az elsőn még a karakter lábai egymás mellett vannak, de a harmadik-on már a bal lába elől van, a jobb lába pedig hátul.

Ha ez elkészült, akkor a következő lépés, hogy egy ciklus használatával folyamatosan kiszedjük az egymás melletti sprite-okat, a nagy képből, amik a sétálásért felelnek.

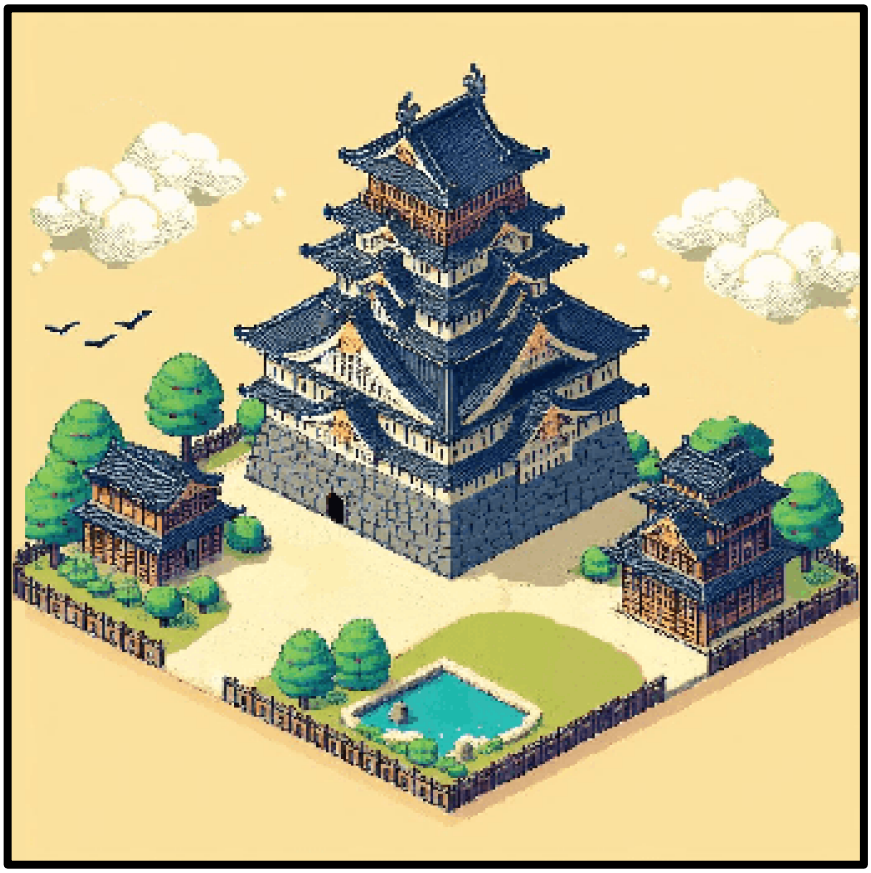
Végül ezt a ciklust hozzá kapcsoljuk egy „keydown-ra” és készen is van a balra sétáló karakterünk, az „A” gomb lenyomásával

10. Összegzés

A projektünk végére érve mindannyian elmondhatjuk, hogy nem volt egyszerű feladat elkészíteni a terveinket. Mind a hárman szorgosan dolgoztunk a végeredményért. A tudásunk rengeteget fejlődött a tanév alatt, mind backend, frontend és egyéb más téren.

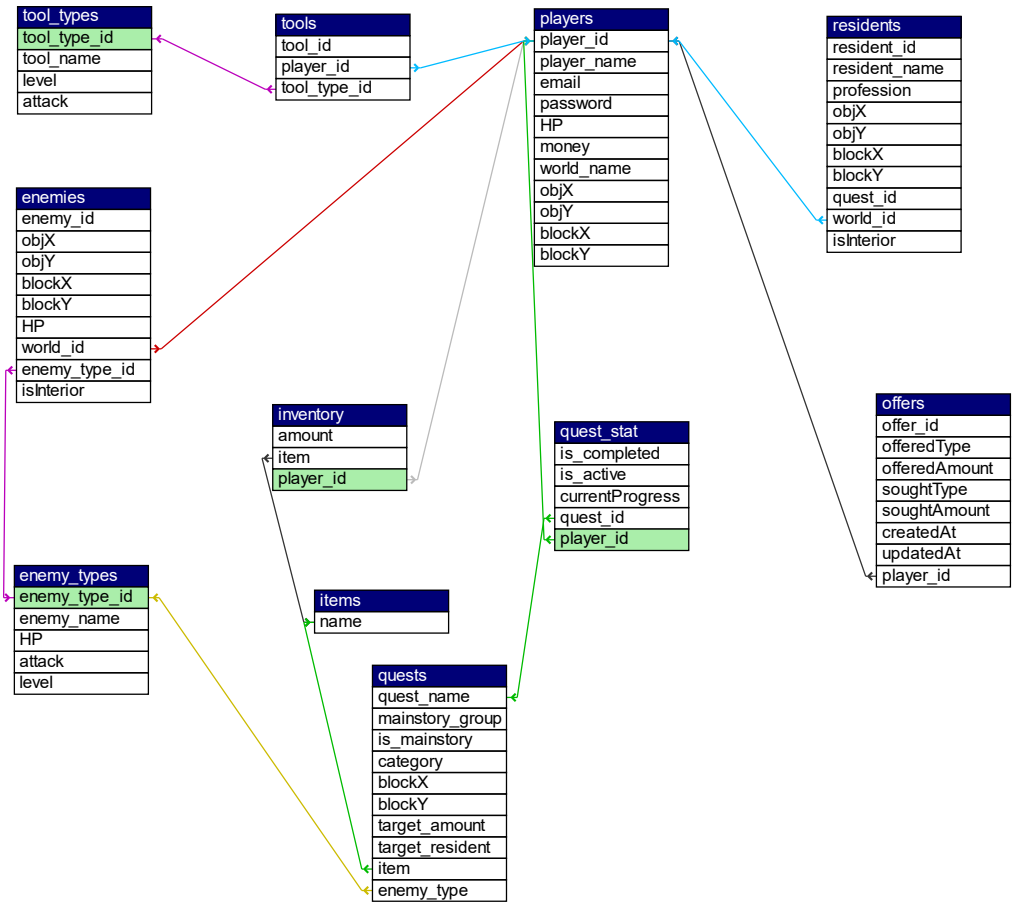
Mivel egy játékot készítettünk, ezért több kedvvel programoztunk, annál mintha más témában készítettünk volna projektet. Jó érzés volt, például egy hosszas kódolás után végre megpillantani azt, ahogy a karakterünk elkezd mozogni, vagy az egyik pályáról átvihetjük őt egy másikra. Számos funkciót sikerült a játékban megvalósítanunk.

Lehet, hogy nem a Forgotten Kingdom lesz az utolsó, de mindig ez lesz az első videójáték, amit elkészítettünk.

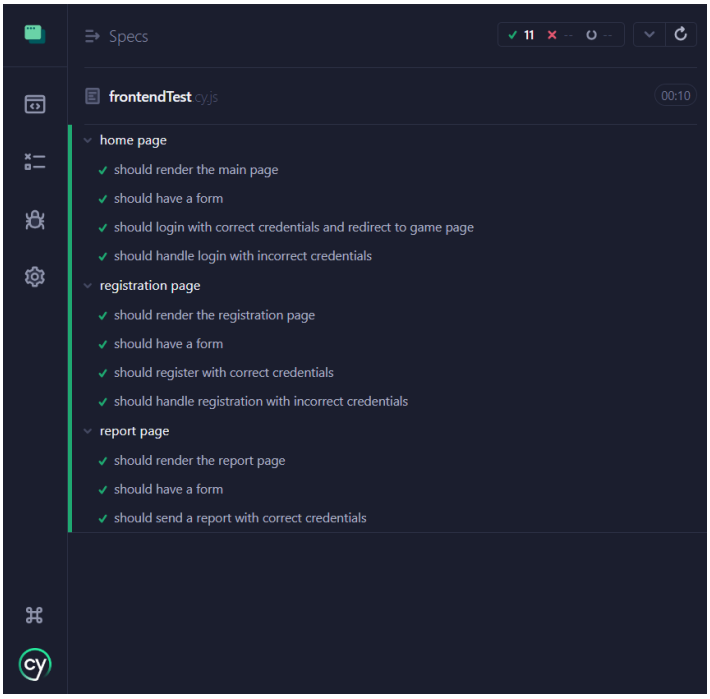


11. Tartalomjegyzék

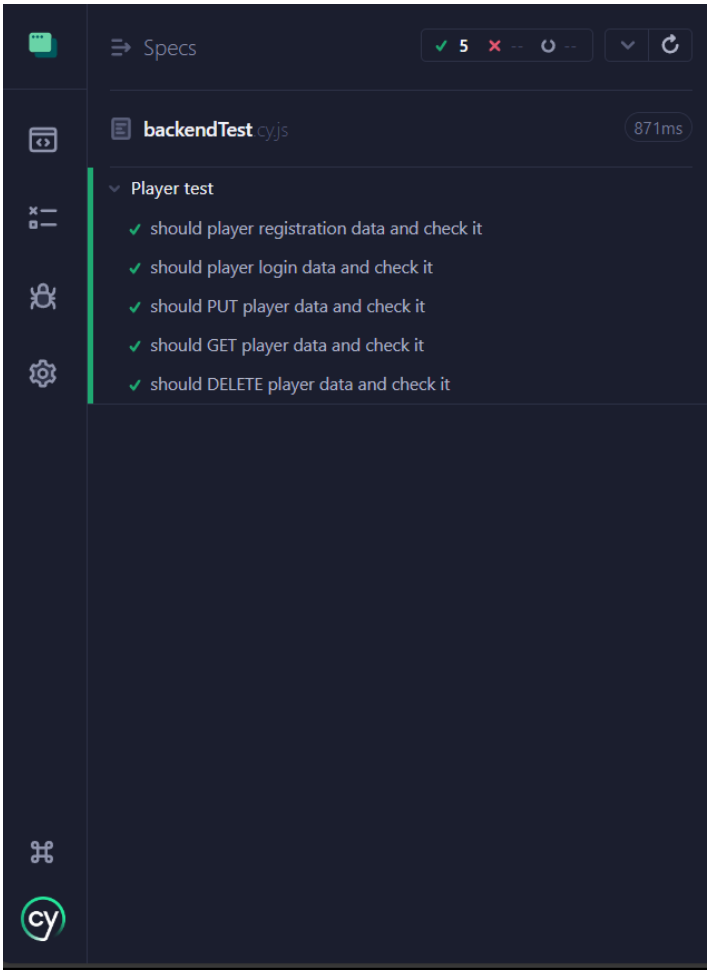
12. Melléklet



12-1 Adatbázis diagram



12-2 Frontend teszt



12-3 Backend teszt

13. Források

13.1. Általános

Node.js — Run JavaScript Everywhere (nodejs.org)

Vue.js - The Progressive JavaScript Framework | Vue.js (vuejs.org)

Testing Frameworks for Javascript | Write, Run, Debug | Cypress

API Documentation & Design Tools for Teams | Swagger

GitHub: Let's build from here · GitHub

Home - MEGA

Medieval Song Village Consort [No Copyright Music] (youtube.com)

13.2. Frontend

ECMAScript® 2023 Language Specification (tc39.es)

JavaScript reference - JavaScript | MDN (mozilla.org)

https://www.youtube.com/watch?v=U34lXz5ynU&ab_channel=freeCodeCamp.org

13.3. Backend

Node.js Tutorial for Beginners: Learn Node in 1 Hour (youtube.com)

Node.js Ultimate Beginner's Guide in 7 Easy Steps (youtube.com)

Node.js and Express.js - Full Course (youtube.com)

13.4. Grafika

How To Pixel Art In 10 Minutes | Pixel Art Tutorial (youtube.com)

How To Pixel Art - Beginner To PRO Tutorial (youtube.com)

Modular Isometric Pixel Art | Tutorial (youtube.com)

Isometric Pixel Art Tutorial - Pixel Art Tips (youtube.com)

Isometric Pixel Art Practice (youtube.com)

10 Minute Pixel Sprite TUTORIAL for COMPLETE BEGINNERS (youtube.com)

Pixel Art Class - Isometric Character Basics (youtube.com)

VWolfdog | OpenGameArt.org

Super Super Rampage Monster @ PixelJoint.com