**Záródolgozat feladatkiírás**

Tanulók neve: Hay Dávid Lajos, Bánfi András

Képzés: nappali

Szak: 5 0613 12 03 Szoftverfejlesztő és tesztelő technikus

**A záródolgozat címe:**

**„God And Prophet” játék**

Konzulens: Bólya Gábor

Beadási határidő: 2023. 04. 15.

Győr, 2022. 10. 01

**Módos Gábor**igazgató

**Konzultációs lap**

|  |  |  |  |
| --- | --- | --- | --- |
|  | A konzultáció | | Konzulens aláírása |
| ideje | témája |
| 1. | 2023.02.15. | Témaválasztás és specifikáció |  |
| 2. | 2023.03.14. | Záródolgozat készültségi fokának értékelése |  |
| 3. | 2023.04.17. | 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 minket és szakmai vizsgát csak új záródolgozat készítése után tehetünk.

Győr, 2023. április 15

tanuló aláírása

tanuló aláírása

Tartalom

1. [Játékleírás 4](#_Toc164075509)

[**A játék célja:** 4](#_Toc164075510)

[Térkép / Utazás: 4](#_Toc164075511)

[Mérőszámok / fontos alapok: 7](#_Toc164075512)

[Istenek 8](#_Toc164075513)

[Próféta 12](#_Toc164075514)

1. [Adatbázis 13](#_Toc164075515)

[Felépítés: 13](#_Toc164075516)

[Táblák: 15](#_Toc164075517)

1. [Backend 20](#_Toc164075518)

[ActionsController: 20](#_Toc164075519)

[GodAbilityController: 21](#_Toc164075520)

[talkAbility method: 21](#_Toc164075521)

[GodInfoController: 24](#_Toc164075522)

[GodSelectionController: 25](#_Toc164075523)

[NpcController: 26](#_Toc164075524)

[SessionUpdateController: 27](#_Toc164075525)

[TravelController: 28](#_Toc164075526)

[UserDataController: 29](#_Toc164075527)

[Unit testek 30](#_Toc164075528)

[API hívások 30](#_Toc164075529)

1. [frontend 31](#_Toc164075530)

[Bejelentkezés: 35](#_Toc164075531)

[Kijelentkezés: 35](#_Toc164075532)

1. Játékleírás

# A játék célja:

A játék célja a játéktéren elhelyezett összes szereplő kijelölt hit szintjét (Faith level) elérni. Ezen szám minden szereplőnél megegyezik és jelölve van a játékos számára. Miután elérte egy szereplő a kijelölt szintet a szereplő megtérté (converted) válik. A játék célja, hogy a játékos megtérítse az összes szereplőt, a választott isten és a próféta képességeivel. Természetesen minden szerepl másképp reagál a próféta megjelenésére illetve a választott istenre, ezen tényezők nagyban befolyásolják a játékmenetet. A cél elérésére összesen 30 nap áll rendelkezésre a próféta számára.

Térkép / Utazás:

A térképen összesen öt különálló lokáció található ezek mindegyike külön feladatot és funkciót lát el. A területek között az utazás korlátozott. Nem minden helyszínt köt össze útvonal, ezért a játékosnak meg kell terveznie hogyan osztja be az idejét. Minden útvonal megjárása időbe telik, amelyet a próféta elveszít egy napból. Az utazási időt befolyásolja az isten, aki segíti a prófétát és a célhelyszín távolsága.

**Templom (Temple) :** A templom egy speciális lokáció. Ezen a helyen a próféta tud pihenni ezzel az energiáját újra tölteni és innen tud új útra indulni. Csak a templomból érhető el az összes másik helyszíne a játéknak, így a játékosnak mindig vissza kell térnie erre a helyre. Az utazás és az alvás is természetesen időt vesz igénybe, ezért a játékosnak figyelnie kell, nehogy kifusson az időből. Amennyiben a játékos energiaszintje (energi level) eléri a nullát A próféta végelgyengülésben meghal és a játék véget ér.

**Szántóföldek (Fields):** A szántóföldeken két szereplőt talál a játékos. Az első a földesúr (lord) a második pedig a paraszt (peasant). A játékos feladata, hogy mind a két szereplőt egyenként megtérítse ezzel is segítve istenét és haladva a győzelem felé. Ezen helyszínt nagyban befolyásolja a választott isten kiléte.

**Város (Town):** A városban két szereplő található. A bankár (banker) és a kovács (blacksmith). Ezt a két szereplőt kell megtérítenie a játékosnak.

**Erdő (Forest):** Az erdőben két szereplő található, a vadász (hunter) illetve vándor (wanderer). A játékosnak ezen két szereplőt kell megtéríteni e ezen a helyen.ű

**Hadsereg (Military):** A hadseregben található a katona (soldier) és a parancsnok (warlord). Ezen szereplőket kell megtéríteni ezen a helyen.



Mérőszámok / fontos alapok:

A játékban vannak már az előbbiekben is említett merőszámok melyeket folyamatosan figyelnie kell a játékosnak és menedzselnie ahhoz, hogy a lehető leghatékonyabban és leggyorsabban fejezze a játékot.

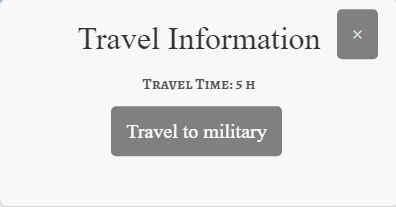
**Általános hit (faith point):** ezen merőszám az átlagos hitet jelenti a játékban. Ezen szám alapvetően nem nulla, hiszen a próféta hite is számításba kerül. Ez egyfajta alapanyag az isten képességei használatához. Az istennek ugyanis csak annyi ereje van amennyien hisznek benne, ezért a játékosnak növelnie kell ezen mérőszámot ahhoz, hogy több és erőteljes3ebb segítséget kapjon a választott istenétől. Ezt a mérőszámot a játékos mindig nyomon követheti a weboldal fejlécében.

**Órák (hours):** Ezen mérőszám folyamatosan számolja az adott napon felhasznált órák számát. Ez természetesen nullázóik egy új nap kezdeténél, de a próféta alvás nélkül csak 3 teljes napot tud végig dolgozni. Az órákat használja a próféta az összes cselekvéshez. Órát használ az utazás, a segítség, az imádság és minden interakció, amelyet a próféta végez. Ezt a mérőszámot folyamatosan nyomon követheti a játékos a weboldal fejlécében.

**Aktuális nap (current day):** Ezen mérőszám megmutatja mennyi napot töltött el a próféta eddig. Amennyiben ez a szám eléri a 30-at a játék véget ér, a próféta veszít. Ezen mérőszám is megjelenik a fejlécben.

**Aktuális energia (current energy):** A fentiekben említett kifáradás mérőszáma. Amennyiben ez a szám eléri a 0-át a próféta meghal. Minden nap végén 25 energiát veszít a próféta. Két módon lehet feltölteni ezen számot. Az első a templomban való alvás. Ez a tevékenység 8 órát vesz igénybe, illetve csak a templomban elérhető, máshol nem. A másik mód egy speciális isteni képesség. Hermész speciális képességével a térképen feltölthető ezen mérőszám. Ezen szám megjelenik a fejlécen.

**Hit szint (faith level):** Ezen mérőszám minden szereplőnél külön merődig. Ezen szám adja meg az aktuális elért szintet az adott szereplőnél, amennyiben ez a szám eléri a megadott szintet a szereplő megtér. Minden megtért szereplő után kap az isten általános hitet mellyel tud gazdálkodni. Miután egy szereplő hit szintje elérte a megfelelő mennyiséget és a szereplő megtért a próféta már nem tud hatni rá sem isteni sem emberi erejével. Ezen mérőszám megjelenik minden szereplő beszéd ablakának alján így a játékos nyoman tudja követi változását.

**Utazási idő (travel time):** Ezen mérőszám megmutatja, melyik elérhető útvonal mennyi időbe fog kerülni, minden alkalommal amikor a próféta megteszi azt a távolságot. Ez a mérőszám megjelenik minden utazás előtt.

**Költség (cost):** Minden isteni képességnek van ilyen mérőszáma, ez jelzi mennyi álltalános hit szükséges a végrehajtásához. Ezen mérőszám megjelenik minden isteni képesség leírása alatt.

Istenek

A játékos a játék kezdése előtt választhat 3 isten közül. Minden istennek saját egyéni képességei vannak illetve egyéni passzívan aktiválódó bonuszai. Ezen képességeket csak a kiválasztott helyen lehet használni. Az elérhető képességeket a játékos látja a weboldal jobb oldalán a választott isten képével és leírásával egyetemben. Minden istennek van egyedi képessége amelyet a térképen tud alkalmazni, amelyet egy beszélgetés során tud alkalmazni, illetve egy passzív képessége amely a játék elejétől kezdve aktív.

**Hermész** (Hermes): Ezen isten képességei a momentumra és a gyorsaságra fókuszál. Minden képessége ezt tükrözi.

Passzív: Minden elérhető útvonal megtételéhez egy órával kevesebb időre van szüksége a prófétának. Ezzel nagyban megnövelve az egy nap megtehető utak számát a játékos számára.

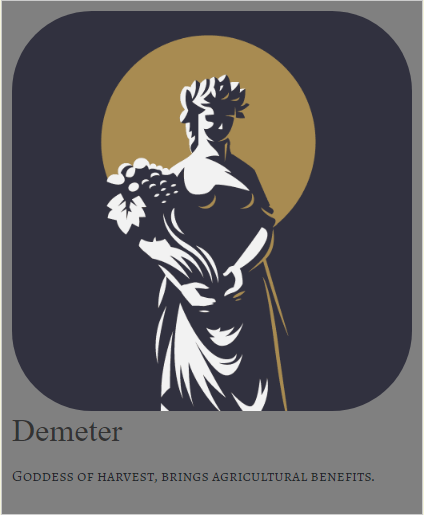
Térképen aktiválható képesség: Hermész képes feltölteni a próféta energiáját ezzel megkímélve a prófétát az alvástól. Ezen képesség nem kerül időbe, így a játékos több dolgot tud egy nap elvégezni alvás helyett.

Beszélgetés közben aktiválható képesség: Két képessége van egy beszélgetés során Hermésznek. Az első egy álltalános képesség mellyel meg lehet duplázni a hit szintjét egy szereplőnek ezzel akár fele annyi munkával megtéríteni azt. A második egy helyszín specifikus képesség. Ezen képességét csak a városban lévő szereplőkre tudja alkalmazni. Az itt elérhető szereplőket egyből meg tudja téríteni nem számít mennyire magas a híd szintje az adott személynek. Ezzel a képességgel a bankárt, illetve a kovácsot egyből át tudja téríteni a játékos.

**Démétér** (Demeter): Ezen isten a szántóföldeken kap bónusz képességeket. Démétér a bőség istene ezért nagy segítséget nyújt a szereplők hít szintjének a növeléséhez.

Passzív: Amennyiben a játékos Démétért választotta, a szántóföldeken élő szereplők számára elég, ha kevesebbet segít a próféta. Ezen passzív képességgel a játékos fele annyi idő alatt képes megtéríteni a szántóföldeken élő szereplőket, ezzel gyorsabban elérve célját.

Térképen aktiválható képesség: Démétér segítségével a próféta a térkép bármely pontjáról azonnal eljuthat a szántóföldekre. Ez nagyban megkönnyíti az utazást a térképen.

Beszélgetés közben aktiválható képesség: Mint minden másik választható isten esetében, Démétér választása esetén is hozzáfér a játékos az adott szereplő hit pontjai megduplázásához. Az egyedi képessége Démétérnek viszont lehetővé teszi a játékosnak, hogy ha a mezőn lévő szereplővel beszél a próféta Démétér azonnal meg tudja téríteni megfelelő mennyiségű általános hitért cserébe.

**Árész (Ares):** Árész a háború istene, így előnyöket tud biztosítani a hadsereg helyszínen. Képességei ezen helyszín szereplői köré épülnek.

Passzív: A hadsereg helyszínen lévő szereplőknek fele annyi hit pontra van szükségük a megtéréshez. Ez nagyban megkönnyíti a játékos dolgát ezen helyszínen.

Térképen aktiválható képesség: Árész lehetővé teszi a próféta számára, hogy a térkép bármely pontjáról rögtön a hadsereg helyszínre mennyen ezzel felgyorsítva ezen terület megtérítését.

Beszélgetés közben aktiválható képesség: Természetesen Árész is hozzáfér a minden helyszínen aktiválható hit pont duplázó képességhez akár csak a többi isten. Ezen képesség melett Árész képes azonnal megtéríteni a hadsereg helyszínen levő szereplőket.



Próféta

A prófétát irányítja a játékos, a próféta személyében a játékos nyilvánul meg. A próféta utazik a térképen és növeli a hit pontjait az egyes szereplőknek. Erre több lehetősége is adódik.

**Segítség (Help):** A próféta lehetőséget kap, hogy segítsen a szereplőknek ezért cserébe pedig a szereplő hit pontjai növekednek. Ez az egyik lehetősége a prófétának a térítésre. Minden segítséggel időt veszít a próféta, így a játékosnak fel kell mérnie mikor mit használ. Egy szereplővel egy nap során csak egyszer lehet interaktálni, így segíteni is csak naponta egyszer van lehetősége a prófétának. Minden segítség után kettő hit pontot kap az adott szereplő.

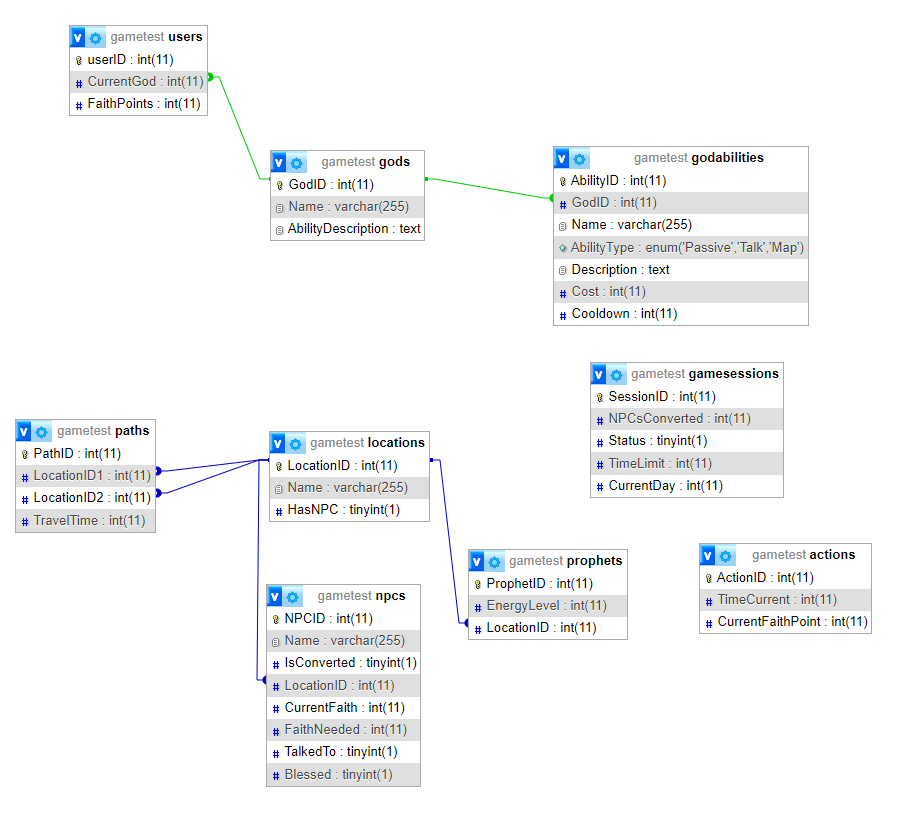
**Imádság (Prey):** A másik lehetőség amellyel a próféta megnövelheti a hit pontjait a szereplőknek az imádság. A prófétának lehetősége van imádkozni a szereplőkkel, imádság közben tanítani őket az istenükről. Ez természetesen több időt vesz igénybe mint a segítség, viszont három hit ponttal megnöveli az adott szereplő hitét. Az imádság is interakciónak számít így naponta egy szereplőnél csak egyszer lehet használni.

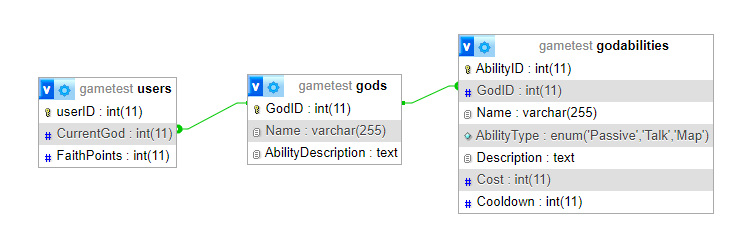
1. Adatbázis

Felépítés:

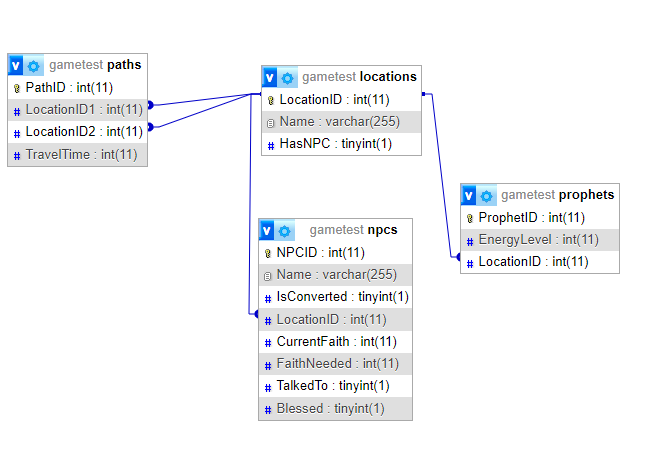
Az adatbázis megtervezésénél a fő szempont az átláthatóság és az egyszerűség volt. Arra törekedtem a megtervezésnél és a megvalósításnál is, hogy első ránézésre átlátható és megérthető legyen akár egy laikus számára. Az adatbázis egyszerű felépítéssel készült a könnyebb kezelés érdekében. Ebből kifolyólag több tábla is csak egy rekordot tartalmaz. Ezen táblák változnak és inkább pillanatnyi adattárolásra használjuk a weboldal futása során.

A másik legfőbb törekvés az volt, hogy minden rekordnak legyen egy olyan sora, ami alapján be lehet sorolni bizonyos kategóriákba. Például az istenek és azok képességei egy különálló azonosító rendszerrel bírnak önmagukban is. Az adatbázis elkészítésekor ugyanis az volt az elképzelés, hogy ezen külön azonosítók alapján egyszerűen és gyorsan le lehet kérdezni a megfelelő adatokat. Ezen módszerrel a hozzáférési jogokat is könnyen kezelhettük, hiszen a játékunkban az isten kiválasztása után a lehetőségek és képességek majdnem harmadára csökkennek.

Az egész adatbázist két külön részre lehet bontani. Az első tartalmazza az adatokat az istenekről, a második pedig a lokációkat és a próféta állapotát. Ezen két csoporton kívül esik három saját tábla melyek nem kapcsolódnak semmivel.

**Istenek táblarendszer:** Ezen táblarendszer tartalmazza az istenek (gods), a felhasználók (users) illetve az istenek képességei (godabilities). A táblarendszer közepén az istenek helyezkedik el. Ezen tábla istenazonosítójához (GodID) kapcsolódik a másik két tábla. A felhasználók a jelenlegi isten (CurrentGod) mezővel kapcsolódik míg az isteni képességek (godabilities) az isten azonosítóval (GodID) kapcsolódik. Ez a felépítés lehetővé teszi, hogy könnyedén elérjen minden szükséges információt a felhasználó.

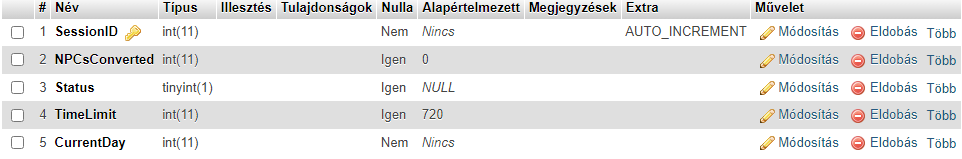
**Helyszínek táblarendszer:** A helyszínek (Locations) tábla áll a középpontjában ennek a rendszernek. Feladata meghatározni a tartózkodási helyét a játék szereplőinek. Elsősorban a prófétának a helyét kell meghatározni, hiszen ez az egyik legfontosabb aspektusa a játéknak. A második meghatározás a szereplőkre vonatkozik, így a rendszer tudja, hogy mely helyszínen milyen szereplőkkel tud interaktálni a játékos. Nem utolsó sorban ez a táblarendszer szolgál arra, hogy a program könnyedén meg tudja határozni a jelenlegi pozícióját a játékosnak illetve a számára elérhető helyszíneket is. Folyamatosan nyomon követhető, milyen helyszínen tartózkodunk és milyen lehetőségeink vannak mind szereplők mind utazás szempontjából. Mindezt ezen táblarendszer határozza meg. A helyszínek (Locations) tábla helyszínazonosítójával kötünk össze minden táblát ami a rendszerben van. A próféták (prophets) Tábla a saját helyszín azonosítójával (LocationID) kapcsolódik, a szereplők (npcs) tábla szintén a saját helyszín azonosítójával (LocationID) kapcsolódik. Az útvonalak (paths) tábla különleges helyet foglal el a rendszerben, mivel két kapcsolattal is rendelkezik a helyszínek tábla helyszínazonosítójához. Mivel ezen tábla az elérhető útvonalak kiindulását és végpontját is tartalmazza ezért mind a két mezőjének kapcsolódnia kell a helyszínek táblához. Így pedig az útvonalak tábla a helyszínazonosító 1 (LocationID1) illetve a helyszínazonosító 2 (LocationID2) mezőjivel kapcsolódik a helyszínek helyszínazonosítójához.



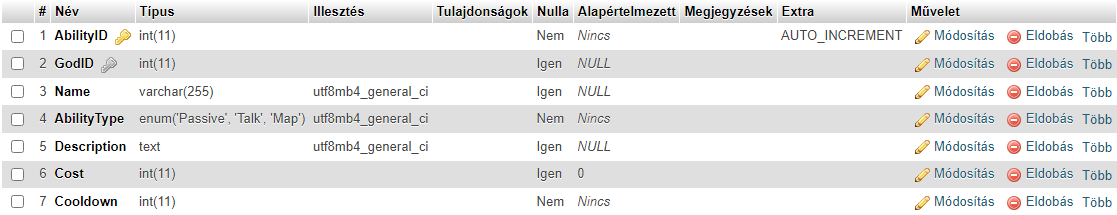
Táblák:

**Akciók (actions)**: Ez egy nagyon egyszerű, de nagyon sokat változó tábla. Ez a tábla összesen egy rekordot tartalmaz és ezen rekord kiválasztott sorait változtatjuk a játék menetén. Ezene tábla két fontos adat folyamatos tárolására szolgál: jelenlegi idő (CurrentTime), jelenlegi hit szint (CurrentFaithPoint). A jelenlegi idő folyamatosan növekszik, amikor játékos csinál bármilyen olyan akciót, ami időt használ, majd nullázóik amikor eléri az egy napot. A hit szint az istenek álltalános „pontjait” tartalmazza. Ez arra szolgál, hogy lekorlátozza a játékos lehetőségeit az isteni képességek terén. Nincsenek kapcsolatai a többi táblával.

**Játék szekciók (gamesession):** Ez a tábla hasonlóan egyszerű, mint az akciók tábla. Lényege, hogy könnyen nyomon követhetően tárolja az adatokat. Ezen tábla adatai határozzák meg a játék egyes pontjait és azok tulajdonságait. Mint minden tábla ez is rendelkezik azonosítóval (SessionID), tárolja a játékban már megtérített szereplők számát (NPCsConverted), a státuszát a prófétának (Status), az idő korlátot órákban meghatározva (TimeLimit), illetve tartalmazza még a jelenlegi napot (CurrentDay).

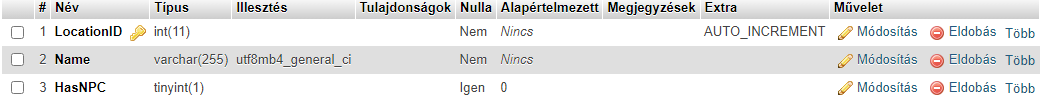


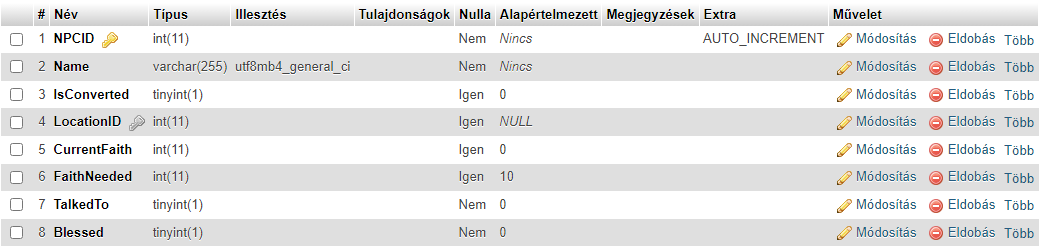
**Isteni képességek (godablities):** Ez a tábla teljese statiku, az egyetlen feladata a képességek és azok tulajdonságainak a tárolása. Természetesen ez is rendelkezik egy azonosítóval (AbilityID), egy a képesség istenét azonosító mezővel (GodID), egy névvel, amit meg tudunk jeleníteni a játékos számára (Name), minden képességnek van egy típusa ami segít meghatározni, hogy a játék mely pontján elérhetőek (AbilityType), Ezen két attribútum szolgál a képességek gyors megkülönböztetésére és besorolására. Minden képesség rendelkezik egy rövid leírással a felhasználó számára (Description), illetve minden képességnek van egy különálló ára (Cost). Ez a tábla egyszerűen bővíthető és kezelhető.

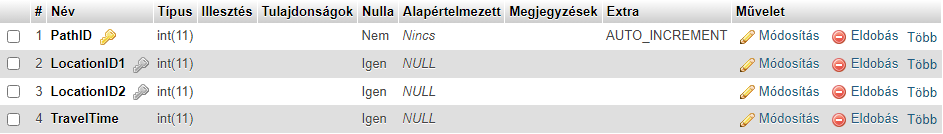


**Istenek (gods):** Ez is egy statikus nem változó tábla. Egyetlen feladata, hogy eltárolja a választható isteneket. Mint a program egészében ebben is szempont volt a könnyű bővíthetőség. Ennek a táblának összesen 3 mezője van: az isten azonosító (GodID) mint azt a fentiekben említettem a kapcsolódáshoz használatos, az isten neve (Name), képesség leírás egy kis összefoglaló az isten stílusáról (AbilityDescription).

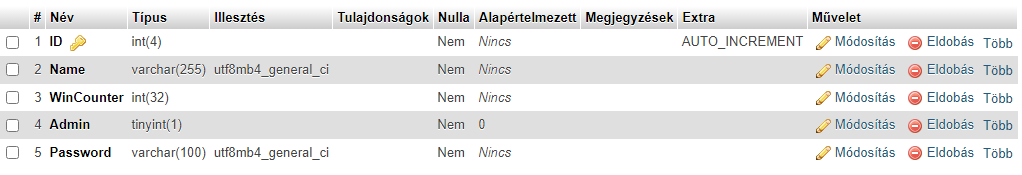


**Helyszínek (locations):** Ez a tábla is statikus és kifejezetten a helyszínek tulajdonságait tárolja. Ebben a táblában minden olyan adat tárolódik ami a térkép könnyű kezelésére szolgál, a felépítését adja meg, mennyi helyszín van ezek, hogyan viszonyulnak egymáshoz és szamon tartja, hogy van e benne szereplő vagy nincs. Tartalmaz egy helyszín azonosítót a rendszer félépítésénél említett okok miatt (LocationID), egy nevet a könnyű megjelenítéshez (Name), illetve egy tartalmaz szereplőket mezőt (HasNPC).

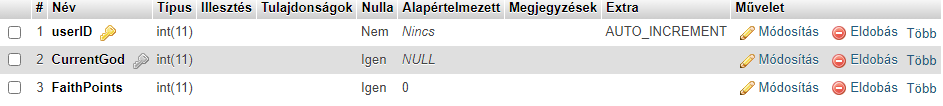
**Szereplők (npcs):** Ez a tábla tartja számon az összes szereplő tulajdonságát. Minden szereplő egy rekord ezzel megkönnyítve a kezelését. Ezen táblával az volt a cél, hogy minden szereplőt könnyen meg lehessen különböztetni és ebből a táblából le lehessen kérni minden adatot. Mint minden táblában ebben a táblában is van egy álltalános azonosító (NPCID), egy név (Name), a konvertált egy olyan boolean mező amely alapján egyből el lehet dönteni, hogy már megtért vagy még téríteni kell (IsConverted), minden szereplőnek van egy helyszín azonosítója amely megadja, hogy melyik helyen van ezen szereplő (LocationID), a pillanatnyi hit a hit pontokat tartja számon minden szereplőnkként (CurrentFaith), szükséges hit mező számon tartja, hogy mennyi elért hit pont után tér meg a szereplő, ezen mező megkönnyíti az istenek passzív képességük végrehajtását. A beszélt már mező egy segéd mező, lehetővé teszi, hogy könnyen számon tudja tartani a program, hogy aznap beszélt e már a próféta az adott szereplővel (TalkedTo). Az áldott mező szintén egy segédmező, amely az isteni képesség használatát korlátozza hasonló módon, mint a beszélt már mező.

**Utak (paths**): Ezen tábla számontartja, hogy milyen útszakaszok vannak a játékba, ezek mennyi időt veszne igénybe, illetve melyik út mit köt össze mivel. Ezek miatt az adatok miatt könnyen kezelhető az utazás. Egy álltalános azonosító (PathID), helyszín azonosító 1 (LocationID1) amely az útszakasz egyik végén lévő helyszínt azonosítja, helyszín azonosító 2 (LocationID2) amely az útvonal másik végét. Az utazási idő tartja számon, hogy mely út megtételéhez mennyi időre van szükség. Ezen mező lehetővé teszik, hogy könnyen változtatni lehet ezen adatokat.

**Próféták (prophets):** Ezen tábla tartja számon a öpróféta sokat változó adatait. Van egy álltalános próféta azonosítója (ProphertID), ezen tábla tartja számon az energia szintjét a prófétának (EnergyLevel) illetve azon helyszín azonosítóját ahol éppen tartózkodik a próféta (LocationID).

**Felhasználói adat (userdata):** Ezen tábla tárolja a felhasználókat és azok adatait. Segít számontartani és befolyásolni minden olyan felhasználót, aki regisztrál. Van egy azonosító mezője amely segít beazonosítani a felhasználót (ID), egy név mezője amely a felhasználó választott nevét tartja számon (Name), tartalmaz még egy győzelem számlálót mely az eddigi sikeres játékait tartja számon a felhasználónak. A tábla még tartalmaz egy admin mezőt amely azonosítja, hogy az adott felhasználó admin jogokkal rendelkezik e (Admin). Az utolsó mezője tartalmazza a jelszót titkosított formában (Password).

**Felhasználók (users):** Ez a tábla is egy rekordot tartalmaz és ez is sokat változik. Lényege, hogy a kiválasztott istent menti el illetve, az elkölthető hit pontokat számolja. Mezői: azonosító (userID), jelenlegi isten (CurrentGod) és a hit pontok (FaithPoints).



1. Backend

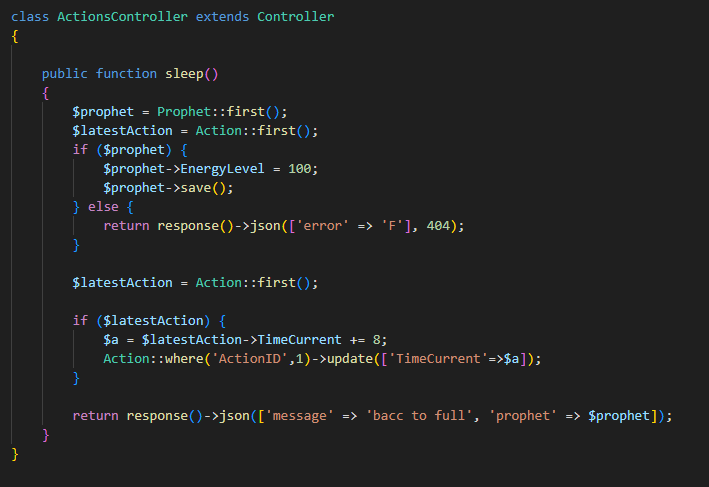
ActionsController:

**sleep method:** Az első változó ebben a metódusban a prophet, ami lekéri az adatait a first() metódussal az első elemnek a próféta táblában. A második változó a latestAction, ami az action táblából kér le adatot szintén a first() metódus segítségével.

A first() metódus a Laravel egyik alapvető adatbázis-lekérdezési módszere, amely lehetővé teszi az alkalmazás számára, hogy az első találatot kérje le egy adott adathalmazból. Ez a metódus rendkívül hasznos abban az esetben, amikor csak az első találatra van szükségünk egy adott lekérdezés eredményéből.

Amikor a first() metódust használjuk, a Laravel az adatbázisban lévő első rekordot fogja visszaadni a megadott feltételek alapján, vagy az adatbázisban található első rekordot, ha nincsenek megadva kifejezések. Ez a metódus lehetővé teszi a gyors és hatékony lekérdezéseket, különösen akkor, ha csak az első találatra vagyunk kíváncsiak.

A first() metódus jellemzően használatos az Eloquent ORM (Object-Relational Mapping) rétegben, amely lehetővé teszi az adatbázis táblákkal való kommunikációt az objektumok formájában. Például egy adott modellhez kapcsolódó első rekordot lehet lekérni a first() metódussal a modell neve előtt.

Ez a metódus különösen hasznos az alkalmazásokban, amelyeknek csak az első találatra van szükségük egy adott lekérdezés eredményéből, például a felhasználók listájából vagy más adathalmazból. Ez lehetővé teszi az alkalmazás számára, hogy gyorsan és hatékonyan hozzáférjen az adatokhoz, és reagáljon a felhasználói kérésekre vagy más eseményekre.

Összességében a first() metódus egy egyszerű, de hatékony eszköz a Laravel alkalmazásokban, amely lehetővé teszi az alkalmazás számára, hogy az első találatot kérje le egy adott adathalmazból, javítva ezzel az alkalmazás teljesítményét és hatékonyságát.

 A metódus először a prophet energyLevel értékét egyenlővé teszi százzal, majd elmenti a változásokat a táblában. Ezután pedig a latestAction TimeCurrent érékét növeli meg nyolcal az where() és az update() metódusok használatával.

GodAbilityController:

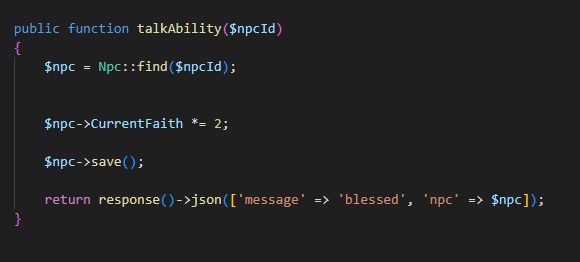
talkAbility method: ez a metódus négy változót használ, az npct aminek adatai a find() metódussal vannak lekérve az adatbázisból.

A find() metódus a Laravel egy másik alapvető adatbázis-lekérdezési módszere, amely lehetővé teszi az alkalmazás számára, hogy egy adott azonosító alapján keresse meg egy adathalmazban található rekordot. Ez a metódus különösen hasznos akkor, amikor egy konkrét rekordra van szükségünk az adatbázisból az azonosítója alapján.

Amikor a find() metódust használjuk, a Laravel az adatbázisban keresni fogja az adott táblában azonosító alapján a megadott azonosítót, és vissza fogja adni a megfelelő rekordot, ha megtalálja. Ha nem találja meg az adott azonosítóval rendelkező rekordot az adatbázisban, akkor null értéket fog visszaadni.

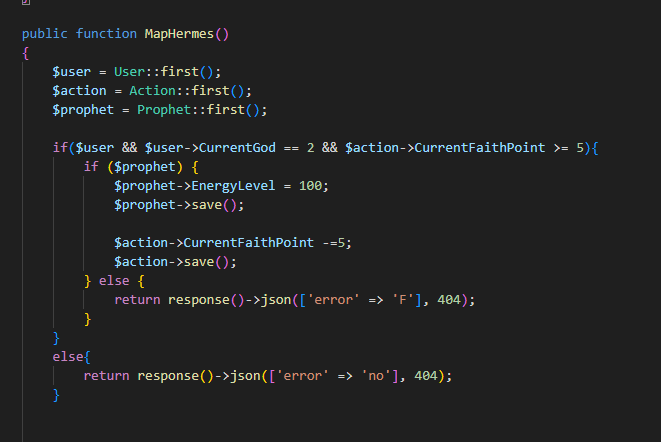
A find() metódus szintén gyakran használatos az Eloquent ORM (Object-Relational Mapping) rétegben, amely lehetővé teszi az adatbázis táblákkal való kommunikációt az objektumok formájában. Például egy adott modell egyedi azonosítója alapján lehet lekérdezni egy rekordot a find() metódussal.

Ez a metódus nagyon hasznos az alkalmazásokban, amelyeknek konkrét rekordokra van szükségük az adatbázisból egyedi azonosító alapján. Ez lehetővé teszi az alkalmazás számára, hogy gyorsan és hatékonyan hozzáférjen az adott rekordhoz, és felhasználja azt az alkalmazás működése során.



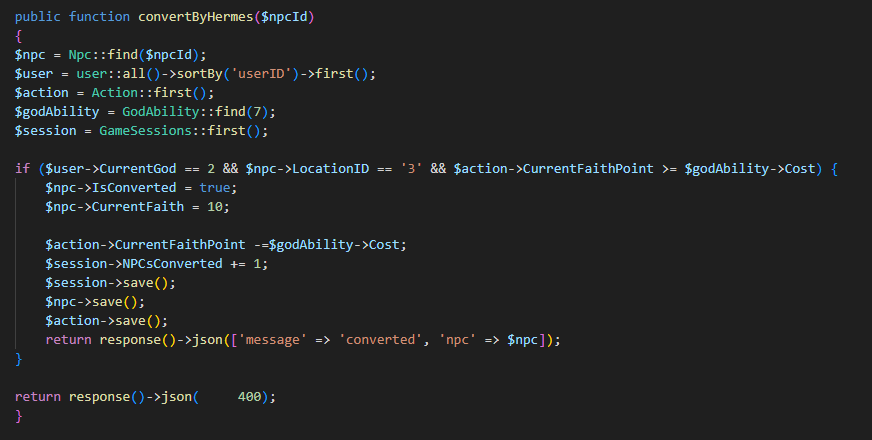
Összességében a find() metódus egy egyszerű és hatékony eszköz a Laravel alkalmazásokban, amely lehetővé teszi az alkalmazás számára, hogy egy adott azonosító alapján keresse meg egy adathalmazban található rekordot, javítva ezzel az alkalmazás teljesítményét és hatékonyságát.

Az action, user és prophet változók a first() metódus használják. Abban az esteben ha a játékos rendelkezik a megfeleő mennyiségű faith pointal akkor az addiott npc currentFaith értéke megkétszereződik, és a levon a prophet currentFaithPoints értékéből kettőt.

**MapHermes method:** ebben a metódusabn három változó található: user, action és prophet, ezek a változók a first() metódus segítségével vesznek fel értéket az adatbázisból. A metódus ellenőrzi hogy a játékos hermest választotta ki a játék elején és hogy megfelelő mennyiségű faithPoitnal rendelkezik, ha ezek a feltételek teljesülnek, akkor a prophet energyLevel értékét egyenlővé teszi százzal majd elmenti a változásokat a  táblában, majd levon a currentFaithPoints értékéből ötöt. Abban az esetben ha valamelyik feltétel nem teljesül a metódus hiba üzenettel tér vissza ami jelzi hogy melyik kondíció miatt nem tudott lefutni.

**teleportAres method:** A metódus három változót használ: user, action és prophet, ezek a változók a first() metódus segítségével kérnek le adatokat az adatbázisból. A metódus ellenőrzi hogy a játékos arest választotta ki a játék elején és hogy megfelelő mennyiségű faithPoitnal rendelkezik, ha ezek a feltételek teljesülnek, akkor próféta location értékét egyenlővé teszi öttel (ami a military location) majd a CurrentFaith point értékéből levon ötöt végül pedig elmenti a változásokat az adatbázisban. Abban az esteben ha a játékos locationja öt akkor a metódus le se fut.

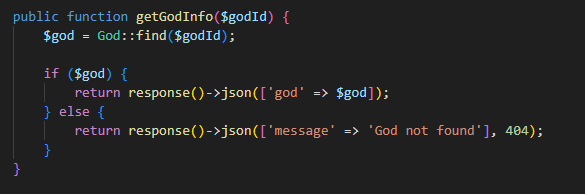
**teleportdemeter method:** A metódus három változót használ: user, action és prophet, ezek a változók a first() metódus segítségével kérnek le adatokat az adatbázisból. A metódus ellenőrzi hogy a játékos demetert választotta ki a játék elején és hogy megfelelő mennyiségű faithPoitnal rendelkezik, ha ezek a feltételek teljesülnek, akkor próféta location értékét egyenlővé teszi kettővel (ami a fields location) majd a CurrentFaith point értékéből levon ötöt végül pedig elmenti a változásokat az adatbázisban. Abban az esteben ha a játékos locationja kettő akkor a metódus le se fut.

**convertByHermes method:** Ez a metódus öt változóval rendelkezik: npc ami a find() metódussal keresi meg az adott npct. user lekéri az összes adatot az all() metódussal, majd sorba rendezi user Id alapján, végül kiválasztja az első elemet. action és a session változók a first() metódust használják. A godability pedig a find() metódust alkalmazza. Első lépésként ellenőrizve van, hogy a játékos hermest választotta, megfelelő npcvel van interakcioban , és rendelkezik-e elegendő faithPointal ha ezek a kondíciók teljesülnek akkor az adott npc azonnal „megtérül” vagyis az adatbázisan az értékeit ennek megfelelően változtatja a metódus, majd elmenti a változásokat.

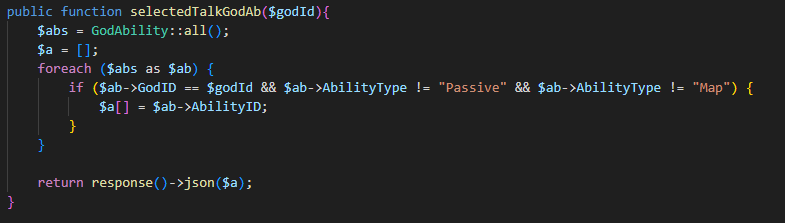
**convertBydemeter method:** Ez a metódus öt változóval rendelkezik: npc ami a find() metódussal keresi meg az adott npct. user lekéri az összes adatot az all() metódussal, majd sorba rendezi user Id alapján, végül kiválasztja az első elemet. action és a session változók a first() metódust használják. A godability pedig a find() metódust alkalmazza. Első lépésként ellenőrizve van, hogy a játékos demetert választotta, megfelelő npcvel van interakcioban , és rendelkezik-e elegendő faithPointal ha ezek a kondíciók teljesülnek akkor az adott npc azonnal „megtérül” vagyis az adatbázisan az értékeit ennek megfelelően változtatja a metódus, majd elmenti a változásokat.

**convertByAres method:** Ez a metódus öt változóval rendelkezik: npc ami a find() metódussal keresi meg az adott npct. user lekéri az összes adatot az all() metódussal, majd sorba rendezi user Id alapján, végül kiválasztja az első elemet. action és a session változók a first() metódust használják. A godability pedig a find() metódust alkalmazza. Első lépésként ellenőrizve van, hogy a játékos arest választotta, megfelelő npcvel van interakcioban , és rendelkezik-e elegendő faithPointal ha ezek a kondíciók teljesülnek akkor az adott npc azonnal „megtérül” vagyis az adatbázisan az értékeit ennek megfelelően változtatja a metódus, majd elmenti a változásokat.

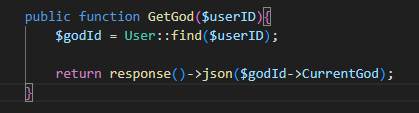
GodInfoController:

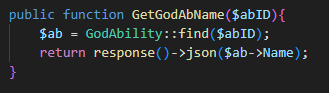
 **getgodinfo method:** egyetlen változó van ebben a metódusban ami megkeresi a find() metódus segítségével az adott istent, majd visszaadja az adatbázisból az adatokat az istentől.

**selectedtalkgodab method:** az abs változó kéri le az adatokat a godability táblából az all() metódus segítségével, majd az a[] tömbváltozóba tárolja el az összes talk abilityt amivel az adott isten rendelkezik.

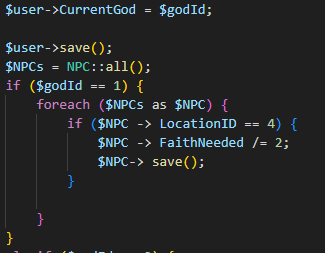


**getgod method:** a godid változó megkeresi a find() metódus segítségével az adott felhasználó által kiválasztott isten azonosítóját, majd az azonosító alapján visszatér az adott isten nevével.

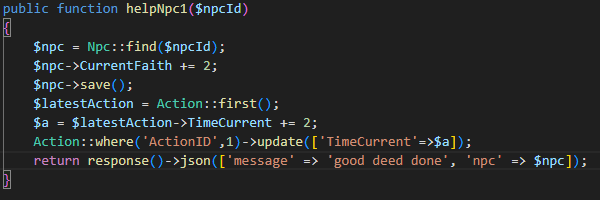


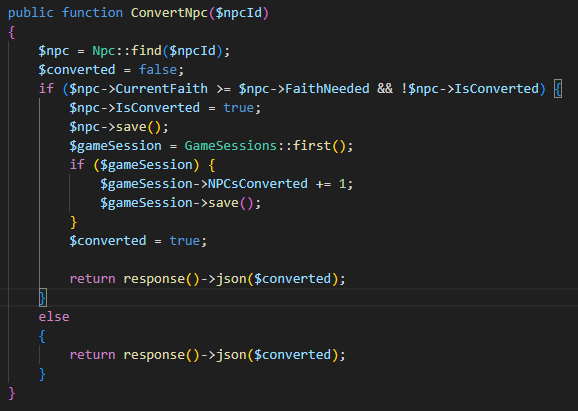
**getgodabname method:** az ab változóba a find() metódus segítségével az adott felhasználó által kiválasztott isten képességének azonosítóját tároljuk el, majd a metódus visszaadja az adott isten adott képességének nevét.

GodSelectionController:

**selectgod method:** ez a metódus felelős a játékos által kiválasztott isten eltárolásáért, és a passive képességek működéséért. Első lépéként a user változóba elmenti a kiválasztott isten azonosítóját, majd az alapján hogy melyik isten lett kiválasztva végig megy azonosító alapján a passzív képességeken, és a megfelelőt hozza érvénybe.

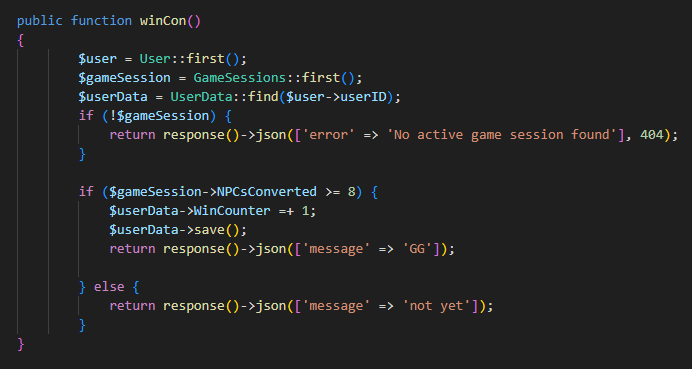
NpcController:

**helpnpc1 method:** az npc változóba a find() metódus segítségével használja az adott npc adatait, hozzáad az npc currentfaith értékéhez és a currenttime értékézhez kettőt majd menti a változsásokat az adatbázisban.

**convertnpc method:** megkeresi az adott npct a find() metósdus segítségével, majd az adatait az npc változóba tárolja, a gamesession változó a first() metódust alkalmazza. Ez a metódus megvizsgálja hogy az adott npc elétre-e a megfelelő mennyiségű faithpointot és abban az esteben ha igen az npc státuszát módosítja „converted” állapotba.

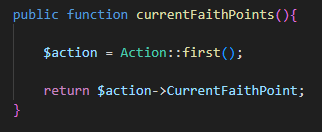
**faith method:** megkeresi az adott npct a find() metósdus segítségével, majd az adatait az npc változóba tárolja, ezután visszatér az adott npc faithpoint értékével.

SessionUpdateController:

**wincon method:**  ez a metódus a user és gamesession változók értéküket a first() metódus segítségével kapják meg. A userdata változó pedig a find() metódust alkalmazza ami a felhasználó azonosítója alapján keres. Abban az esetben ha a játékos megnyert egy játékot akkor az adott játékos megnyert játékainak értéke megnő egyel.

**endofday method:** három változó található itt, a prophet, gamesession és az action, mind a három változó a first() metódus segítségével kap értéket. Abban az esetben ha az eltelt órák száma meghaladja a huszonnégyet vége egy napnak, ez azt jelenti, hogy a napok száma növekszik eggyel, a timeLimit értéke csökkent huszonnéggyel, a próféta enegrylevel értéke csökken huszonöttel, és frissíti az npck státuszát, hogy a játékos ismét tudjon velük interakcióba lépni, abban az esetben ah a timelimit vagy az energylevel eléri a nullát a játéknak veszteséggel van vége.

**resetdatabase method:** ez a metódus négy változót használ: prophet, gamesession és action ezek a változók a first() metódus segítségével kapnak adatot, a path változó pedig a find() metódust alkalmazza. A metódus alaphelyzetbe állítja az energylevelt a locationt, a megtérített npck számát, a timelimitet, az adott napot, az adott időt, és a az adott faith  pointokat. ezen felül az összesn npct is alaphelyzetbe állítja.

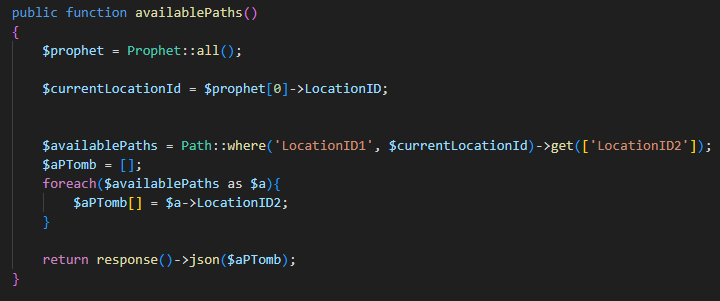
**current faith method**: az action változóba a first() metódus segítségével adatot tárol , majd visszaadja az adott játékos faithpoint értékét.

**current time method**: az action változóba a first() metódus segítségével adatot tárol , majd visszaadja az adott játék currenttime értékét.

**current day method:** az session változóba a first() metódus segítségével adatot tárol , majd visszaadja az adott játék currentday értékét.

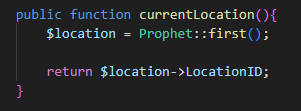
TravelController:

**availablepaths method:** ez a metódus a prophet változóba használ adatokat az all() metódus segítségével. A metódus megnézi a próféta lokációját, majd ez alapján megkeresi és visszaadja az azonosítóját az elérhető utaknak.



**tarvel method:** kettő változót használ ez a metódus: prophet és lastestaction, ezek a változók a firt() metódust alkalmazzák. Első lépésként a metódus feltérképezi hogy a játékos hova akarja a prófétát mozgatni, ha ez nem lehetséges akkor metódus nem fut le. Abban az esetben ha az utazás lehetséges a metódus megváltoztatja a próféta helyének azonosítóját, majd hozzáadja az eltelt időhöz az út idejének értéket végül pedig menti a változásokat.

**current location method:** a location változó a first() metódus segítségével kap adatot, majd a metódus visszatér az adott location azonosítójával.

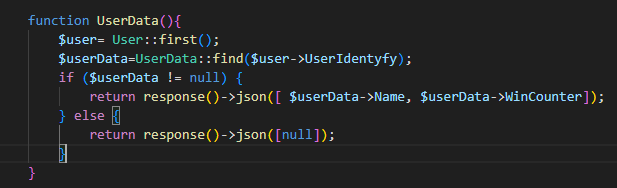


**pathinfo method:** kettő változó található ebben a metódusban: a prophet és a path, melyek a first() és az all() metódusok segítségével kapnak adatot. A metódus végigmegy az összes útvonal adatain majd azonosító segítségével megkeresi az adott útvonalat, végül ennek az útvonalnak az adataival tér vissza

UserDataController:

**register method:** a useridentify változót a first() metódus segítségével alkalmazza, majd létrehoz egy validatort ami meghatározza hogy a regisztrácio közben milyen adatokat kell megadni, ha sikeres a regisztrácio akkor a regisztrált felhasználó adatait lementi az adatbázisba.

**login method:** ez a metódus kéri a felhasználó nevét és jelszavát, amikor megkapja megkeresi az adatbázisban a megadott adatokat és ha az adatok megfelelnek az adatbázisban eltároltaknak akkor sikeres a bejelentkezés.

**userdata method:** megkeresi az adott felhasználó azonosító alapján majd visszaadja a felhasználó adatai az adatbázsból.

**delete user method:** megkeres egy adott felhasználót azonosító alapján és ha sikeresen megtalálta kitörli a felhasználót.

Unit testek

A unit testek fontosak a kóban való hibák felderítésére mielőtt az még élesben lefutna, így sok időt meg lehet spórolni és meg lehet előzni potenciális károkat amiket hibás kód lefutattása okozhat. Ezen indokból készült pár unit test ebben a projectben is, ezekből a testekből néhány példa:

A **testdemeterteleport** unit testet rendkívül fontosnak találtuk, mivel a teleport demeter metódus az egyik legkomplikáltabb metódus a programunkban ezért nagy jelentőséget nyilvánítottunk annak, hogy ez a metódust minden probléma nélkül le tudjon futni élesben elsőre, hogy el tudjuk kerülni a potenciális károkat amiket egy bug okozhat. Maga a test minden lehetséges kimenetelét lekezeli a metódusnak.

A **testaresteleport** metódus rendkívül hasonlít a testdemeterteleport metódushoz ezért ennek is hasonló szintű fontosságot nyilvánítottunk, így ehhez a metódushoz is készült egy unit test ami tovább biztosította hogy a programunk sikeresen lefusson minden probléma nélkül.

A **testhermesmap** **ability** metódus eltér az előző két metódustol (testdemeterteleport, testaresteleport) viszont hasonlóan komplikált, ennek a unit testnek az elkészítése is rendkívül fontos, és hasonló mértékű kihívást jelentett mint az eddigi unit testek elkészítése, szerencsére sikeresen vettük az akadályt és miután a testek sikeresen lefutottak nagyobb nyugalommal tudtuk folytatni a program futását élesben is.

API hívások

Követkető fontos eleme a backendnek az api hívások, mivel ezek segítségével tud kommunikálni a backend a frontendel. Az api hívásokat tartalmazó file fel van osztva hat nagyobb részre az átláthatóság kedvéért:

-**God**

-**NPC**

-**Session**

-**Action**

-**Travel**

-**UserData**

1. frontend

A frontend részén egy állandó elem a navigációs sáv, amely a játékos adatait tartalmazza, mint például a játékos neve, játszott órák és napok száma. Ez a navigációs sáv a Vue.js alkalmazásban található, és a frontend kódot JavaScript nyelven írták a Java Script segítségével. Két fő módszer felel az adatok folyamatos frissítéséért: a watch() metódusok.

Az első watch() metódus figyeli és frissíti a napokat, órákat és egyéb játékon belüli adatokat, míg a második watch() metódus a játékos adatait tartja számon.

A watch() metódus a Vue.js keretrendszer egyik kulcsfontosságú eszköze, amely lehetővé teszi az alkalmazás számára, hogy figyelje és reagáljon a megfigyelt adatok változásaira. Ez a metódus rendkívül hasznos abban az esetben, amikor az alkalmazásnak dinamikusan kell frissítenie a felhasználói felületet vagy végrehajtania más műveleteket az adatok állapotának változása alapján.

A watch() metódus egy objektumot vagy egy függvényt fogad paraméterként, amely figyeli az egyik vagy több adatot. Amikor a megfigyelt adatok értéke megváltozik, a watch() metódus automatikusan elindul, és meghatározott műveleteket hajt végre annak érdekében, hogy reagáljon a változásokra.

Ez a metódus különösen hasznos az alkalmazásokban, amelyek dinamikusan változnak a felhasználói interakciók vagy más események hatására. Például a felhasználói felület automatikusan frissülhet, amikor egy űrlapmező értéke megváltozik, vagy egy másik alkalmazási állapot változik.

A watch() metódus lehetőséget biztosít arra is, hogy összetettebb logikát adjunk a változások kezeléséhez. Például lehetőség van ellenőrizni, hogy az adatok új értéke megfelel-e bizonyos feltételeknek, és csak akkor hajtani végre bizonyos műveleteket, ha ezek a feltételek teljesülnek.

Összességében a watch() metódus egy erőteljes eszköz a Vue.js alkalmazások fejlesztéséhez, amely lehetővé teszi az alkalmazás számára, hogy reagáljon dinamikusan az adatok állapotának változásaira, és ezáltal jobb felhasználói élményt nyújtson.

Ezenkívül fontos szerepe van az onMounted() metódusnak is.

Az onMounted() metódus egy fontos része a Vue.js alkalmazások életciklusának, és akkor fut le, amikor egy komponens fel van szerelve és betöltődik a DOM-ba. Ez az esemény lehetőséget ad az alkalmazásnak arra, hogy inicializálja az adatokat, regisztrálja eseménykezelőket vagy végezzen egyéb műveleteket az alkalmazás indulásakor.

Amikor egy komponens onMounted() metódusa lefut, az alkalmazás már teljes mértékben betöltődött a böngészőbe, és készen áll a felhasználóval való interakcióra. Ez az időpont ideális az inicializálási műveletek elvégzésére, például adatok letöltésére a háttérből vagy más beállítások végrehajtására az alkalmazás indításakor.

Az onMounted() metódus lehetővé teszi a komponensek számára, hogy felkészüljenek az alkalmazás működésére, és hogy azonnal reagáljanak a felhasználói interakciókra, anélkül, hogy várniuk kellene az alkalmazás betöltésére vagy más eseményekre.

Ez a metódus különösen hasznos olyan alkalmazásokban, amelyek dinamikusan változnak az alkalmazás betöltésekor vagy a felhasználói interakciók hatására. Például az onMounted() metódus segítségével az alkalmazás letöltheti az alapértelmezett adatokat vagy inicializálhatja a felhasználói felületet azonnal az alkalmazás betöltésekor.

Összességében az onMounted() metódus lehetővé teszi a Vue.js alkalmazások számára, hogy hatékonyan inicializálják magukat és felkészüljenek a működésre az alkalmazás betöltésekor, ezáltal javítva a felhasználói élményt és a teljesítményt.

A frontend alkalmazásban használt globális változókat a Pinia Store-ban tárolják. A Pinia Store egy állapotkezelő megoldás, amelyet a Vue.js keretrendszerhez fejlesztettek ki. Alapvetően arra szolgál, hogy hatékonyan kezelje az alkalmazás állapotát, lehetővé téve a központosított állapotkezelést a frontend alkalmazásokban. Ezáltal az adatokat könnyen lehet kezelni és elérni a frontend különböző részein keresztül, ami nagyban segíti az alkalmazás fejlesztését és karbantartását.

A Pinia az egyik népszerű alternatíva a Vuex mellett a Vue.js alkalmazásokban. A Vuex szintaxisának egyszerűsítése és a TypeScript támogatása mellett a Pinia előnyei közé tartozik a teljesítmény optimalizálása és a könnyű integrálhatóság más Vue.js alapú projektbe.

A többi komponenset az npcs és a components mappába tároltuk el az átláthatóság kedvéért. Maga a játék a start vue oldalon kezdődik, ahol a játékos kiválasztja az istent amivel játszani szeretne, és megkapja a játék háttértörténetét.  Amikor a játékos kiválaszt egy istent akkor a hozzá tartozó profil adatai közé ez elmentődik az adatbázisba az adott isten azonosítója, aminek segítségével jeleníti meg az oldal az istenek adatait, például a képeket róluk, a képességeiket, stb. Ezen az oldalon vannak alkalmazva a database reset és a godselect metódusok.

Az isten kiválasztása után a játékos átkerül a térképre, ahol a kezdő pont a templom, ami a pálya közepén helyezkedik el.  A frontenden belül ez a legösszetettebb oldal és ennek az elkészítése volt a legnehezebb is. Első lépésként a lokációk közötti utazást oldottuk meg, amiben a kihívást az jelentette hogy a játékos ne tudjon bizonyos helyekre utazni. Az utaknak az adatbázisban megadott adatok alapján kell időbe telnie, a megfelelő adatokat kell kiírni a játékosnak, az utazás után ismét fel kell mérni a szabad utakat. Ezeket sok különböző backendben tudtuk megvalósítani sok különböző metódus segítségével.  Gyakran ütköztünk olyan problémákba hogy nem megfelelő formátumba küldte az adatokat a backend, ezeket a nehézségeket végül szerencsére sikeresen le tudtuk küzdeni.

Maguk a lokációk, (town, temple, military, fields, forest) mind külön komponensben helyezkednek el. Ezeknek a felépítése rendkívül hasonló így csak az egyiket emeltük ki részletesebben, a town components.  Ennek az oldalnak a felépítése rendkívül egyszerű mindössze két npct tartalmaz amelyekkel interakcióba lépve tovább tudunk lépni az npc menübe, ezen felül van egy gomb amivel a játékos vissza tud térni a fő térképre.

A játékban nyolc darab npc van, ezeknek a componenseknek a felépítése szinte identikális, ennek ellenére is ez volt a második legnagyobb kihívás a project frontend részén belül. Itt példaként a banker oldalát prezentáljuk. Az npc menüben a játékos tud segíteni az npcnek, vagy az általa választott isten egyik képességét tudja használni faith point vagy időért cserébe. Ennek az oldalnak a megtervezése és programozás közben nagy kihívást jelentett, hogy mikor az oldal elkészül később tudjuk az oldal egész tartalmát másolni és minimális változtatásokkal felhasználni a többi npc oldalán is.

A Vue útválasztása lényegében arról szól, hogy hogyan navigál az alkalmazás különböző részei között, amikor a felhasználó interakcióba lép velük. Ez azt jelenti, hogy meghatározzuk, hogy az egyes URL-ek mely komponenseket mutassák be az alkalmazásban. A Vue Router segítségével beállíthatjuk ezeket az útvonalakat, és amikor a felhasználó például kattint egy linkre vagy megad egy URL-t a böngészőben, a Vue Router eldönti, hogy melyik komponenst kell megjeleníteni a megfelelő helyen az alkalmazásban. Ezáltal lehetővé teszi az alkalmazásunk számára, hogy válaszoljon a felhasználói interakciókra és az URL-ek változásaira anélkül, hogy az egész oldalt újra kellene betölteni.

A Vue Router használatának dokumentációjában a komponensek közötti navigációhoz a "router" mappában található index fájl szolgál alapul. Ez a fájl tárolja az összes elérhető útvonalat, amelyek segítségével a felhasználók az alkalmazás különböző részei között mozoghatnak. A script rendkívül egyszerű, és kizárólag ezeket az útvonalakat tartalmazza. Fontos megjegyezni, hogy a "name" attribútum használatával könnyíthetjük meg jelentősen a komponensek közötti navigációt a projekt későbbi fázisaiban.

A Vue.js és a Laravel együttes használata egy nagyon hatékony és népszerű módja a webalkalmazások fejlesztésének. A Vue.js egy progresszív JavaScript keretrendszer, amelyet elsősorban felhasználói felületek fejlesztésére terveztek. Másrészt a Laravel egy PHP alapú, nyílt forráskódú webes keretrendszer, amely különféle funkciókat biztosít a háttérlogika és az adatbázis-kezelés terén. Ezek a technológiák különböző rétegekben dolgoznak, és hatékony kommunikációt biztosítanak egymással a frontend és a backend között.

A kommunikáció ezek között a rétegek között elsősorban HTTP kéréseken keresztül történik. A Vue.js által generált kliensoldali alkalmazások a böngészőből HTTP kéréseket küldhetnek a Laravel által kezelt szerveroldali alkalmazásnak. Ezek a kérések lehetnek például AJAX kérések, amelyek segítségével dinamikusan frissíthetők az adatok a felhasználói felületen anélkül, hogy az egész oldalt újratöltenénk. A Laravel felelős az ilyen kérések feldolgozásáért és válaszok küldéséért.

A kommunikáció folyamata a következő lépéseket foglalja magában:

Vue.js alkalmazás felhasználói interakciókat generál a felületen: A Vue.js segítségével a felhasználók interakcióba léphetnek az alkalmazás felületével, például űrlapokat tölthetnek ki, gombokat nyomhatnak meg stb.

HTTP kérések küldése a Laravel backend felé: Amikor a felhasználó interakciót hajt végre, például egy űrlapot elküld, a Vue.js alkalmazás HTTP kérést küld a Laravel backend számára. Ez a kérés általában tartalmazhat adatokat, például űrlapmezőkben vagy más felhasználói felületi elemekben megadott információkat.

Laravel backend feldolgozza a kérést: A Laravel fogadja az HTTP kérést, és az általa biztosított útválasztás és vezérlők révén meghatározza, hogy mit kell tenni a kapott adatokkal. Ez lehet például adatok feldolgozása, adatbázis-lekérdezések végrehajtása vagy más műveletek végrehajtása.

Válasz küldése a Vue.js felé: Miután a Laravel backend feldolgozta a kérést, választ küld vissza a Vue.js kliensalkalmazásnak. Ez a válasz lehet adatok, HTML tartalom, hibaüzenetek vagy más információk.

Vue.js alkalmazás frissíti a felületet: Amikor a Vue.js alkalmazás megkapja a választ a backendtől, frissítheti a felhasználói felületet az új adatokkal vagy más változásokkal. Ez lehet például új adatok megjelenítése egy listában vagy egy űrlapmező értékének frissítése.

Ez a kommunikációs folyamat lehetővé teszi a frontend és a backend közötti hatékony együttműködést a Vue.js és a Laravel kombinált használatával, ami lehetővé teszi dinamikus és felhasználóbarát webalkalmazások fejlesztését.

A bejelentkezés (login) és kijelentkezés (logout) folyamatának működése egy Vue.js és Laravel alkalmazás környezetében hasonló alapelveken alapszik, mint a többi kommunikációs folyamat.

Bejelentkezés:

Felhasználói interakció Vue.js alkalmazásban: A felhasználó megadja a bejelentkezési adatait (felhasználónév és jelszó) a Vue.js által renderelt felhasználói felületen, és elküldi azokat a backend felé.

HTTP kérés küldése a Laravel backend felé: A Vue.js alkalmazás egy HTTP POST kérést küld a Laravel backend felé, amely tartalmazza a felhasználó bejelentkezési adatait.

Laravel backend feldolgozza a kérést: A Laravel fogadja az HTTP kérést, ellenőrzi a kapott adatokat (például az adatbázisban tárolt felhasználói adatokkal összehasonlítva), és ha a bejelentkezés sikeres, létrehoz egy munkamenetet vagy token-t a felhasználó számára.

Válasz küldése a Vue.js felé: A Laravel visszaküld egy választ a Vue.js alkalmazásnak, amely általában tartalmazza a bejelentkezés sikerességét vagy kudarcát, valamint egy esetleges hibaüzenetet vagy egyéb információkat.

Felület frissítése: Ha a bejelentkezés sikeres volt, a Vue.js alkalmazás frissítheti a felhasználói felületet, például megjelenítheti a felhasználó nevét vagy más személyre szabott információkat.

Kijelentkezés:

Felhasználói interakció Vue.js alkalmazásban: A felhasználó valamilyen módon jelezheti a kijelentkezés szándékát a Vue.js által renderelt felhasználói felületen, például egy "Kijelentkezés" gomb megnyomásával.

HTTP kérés küldése a Laravel backend felé: Amikor a felhasználó jelzi a kijelentkezés szándékát, a Vue.js alkalmazás egy újabb HTTP kérést küld a Laravel backend felé, jelezve a kijelentkezést.

Munkamenet vagy token törlése: A Laravel backend fogadja a kijelentkezési kérést, és törli a felhasználóhoz kapcsolódó munkamenetet vagy token-t, ezáltal megszakítva a felhasználói azonosítást.

Válasz küldése a Vue.js felé: A Laravel visszaküld egy választ a Vue.js alkalmazásnak, például megerősítve a kijelentkezést.

Felület frissítése: Miután a Vue.js alkalmazás megkapja a kijelentkezés visszaigazolását, frissítheti a felhasználói felületet, például elrejtheti a bejelentkezett felhasználóhoz kapcsolódó funkciókat vagy megjelenítheti a bejelentkezési űrlapot.

Összességében a bejelentkezés és kijelentkezés folyamata a Vue.js és Laravel között hasonló módon zajlik, mint a többi kommunikációs folyamat. A Vue.js alkalmazás kezdeményezi a kéréseket, míg a Laravel backend végzi el az azokhoz kapcsolódó műveleteket, és visszajelzést ad a Vue.js felé a folyamat eredményéről. Ez a kölcsönhatás lehetővé teszi a hatékony és biztonságos bejelentkezési és kijelentkezési folyamatokat a webalkalmazásokban.