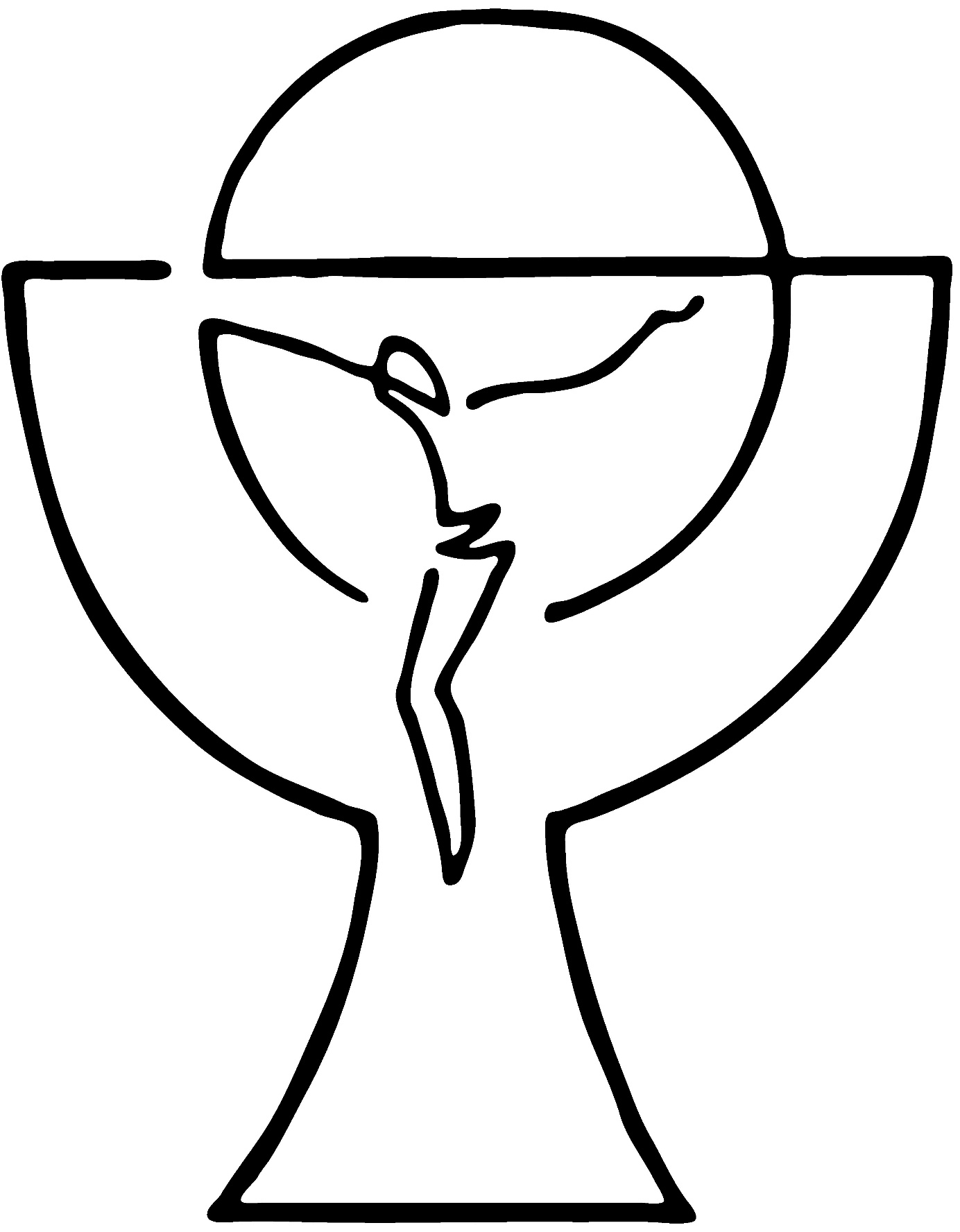
****

Papp Bence

Nagy-Pál Olivér

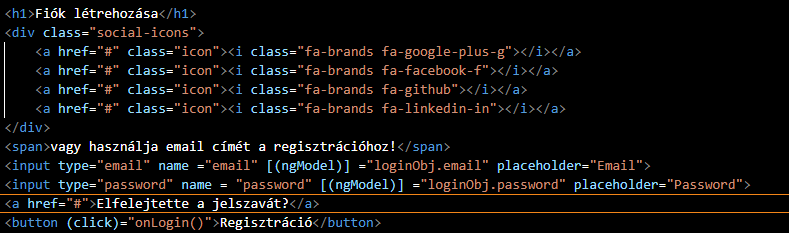
Gabóczi Ádám

## Frontend Regisztráció, bejelentkezés, kijelentkezés Papp Bence:

Bevezetés:

Verzió: 1.0

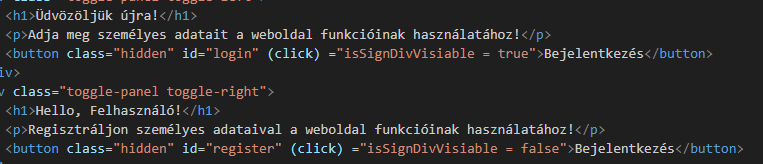
A regisztráció elkészítése során figyelembe került egy olyan szempont, hogy a felület az orvosok számára lett kialakítva, amelyben van lehetőség a személyes profil létrehozására! A regisztráció során ha az orvos a bejelentkezésnél elfelejtette a jelszavát, akkor lehetőség van egy új jelszó kérésére! Egy, a már meglévő profil esetén egy “Bejelentkezés” gomb került kialakításra, ahol a fogorvosok megadhatják a személyes adataikat a betegnyilvántartó használata előtt! Ez a dokumentáció leírja a bejelentkezési felületet, amelyet Angular keretrendszerrel lett létrehozva! A felület kialakításához az app.component.html, és a hozzátartozó stílusokat tartalmazó app.component.css fájlokat használtuk. A dokumentáció a 2024.04.03-i verzióra vonatkozik.

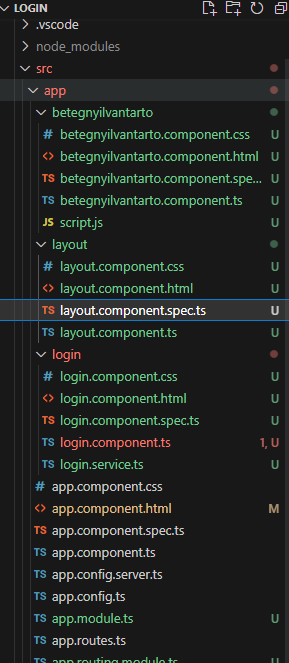
**

*(Az alábbi kódrészlet a bejelentkezés folyamatát, valamint az új jelszó kérést mutatja be!)*

A frontend és a backend összekapcsolása során, két alkalommal is futásidejű hiba mutatkozott meg, így a kapcsolat nem tudott létrejönni a betegnyilvántartó és a frontend között!

A belépésnél az orvosnak szükséges megadni a személyes adatait, hogy használni tudja a számára kialakított weboldalt, és annak funkcióit!

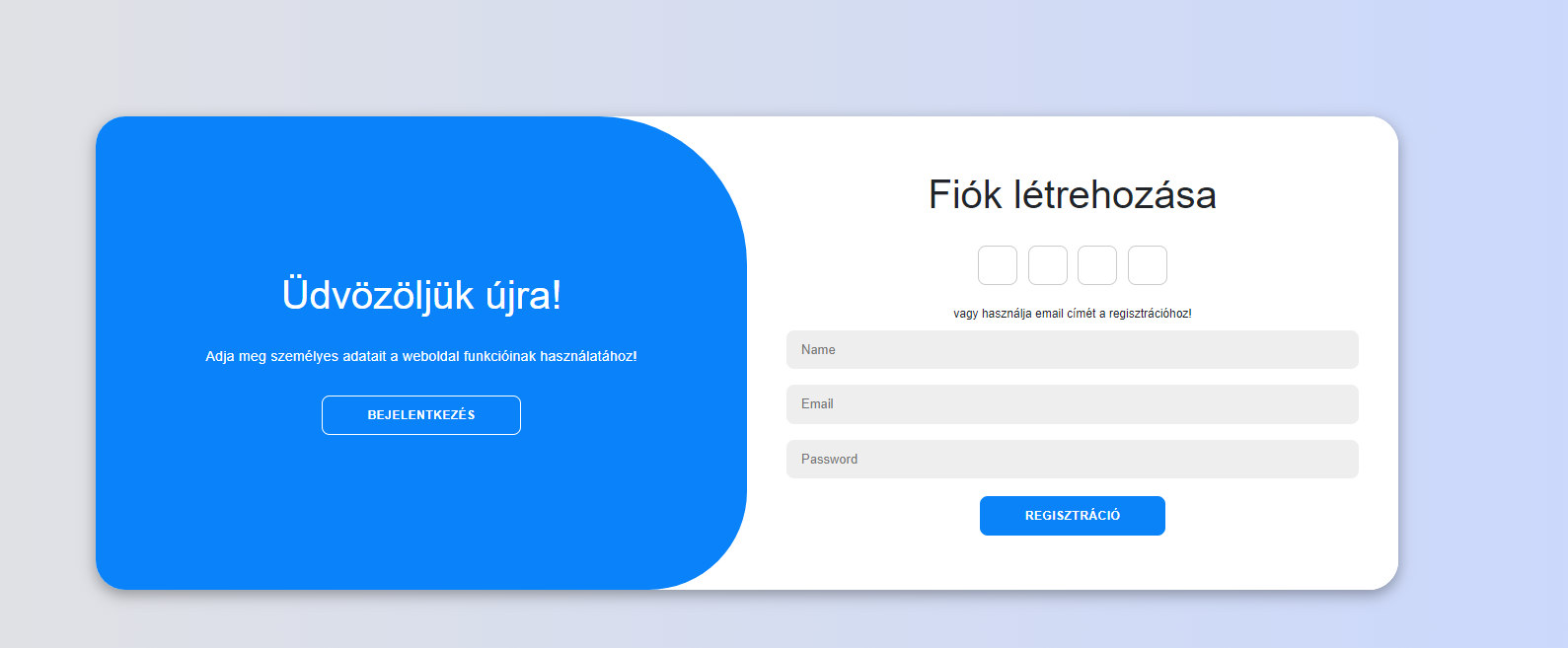


Az alábbi állományokban a kódok elnevezését, és kialakítását láthatjuk, amelyek bemutatják a regisztráció, és a bejelentkezés, valamint a betegnyilvántartó formai ábrázolását mutatja be!  

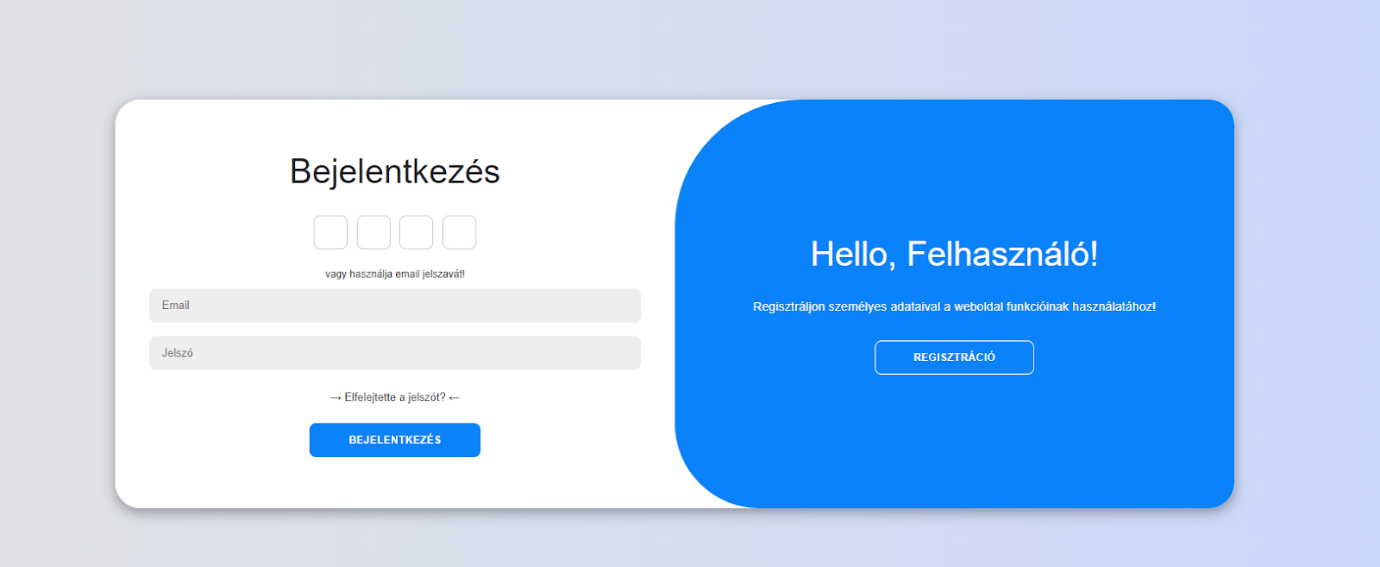
A *login.component.ts* hibás állapotban látható, melynek elkészítésekor egy *export class SignUpModel* nevű osztály ismeretlen token-ként mutatkozott a kódban! A kialakítás során a frontend szerver működőképes állapotban volt használható, még a *login.component.ts* hibájának elhárítása előtt is!

A szerver és a betegnyilvántartó közötti kapcsolat későbbiekben került kialakításra, előtte a futásidejű hiba került még elhárításra!

A frontend működőképes állapotban rendelkezésre állt a betegnyilvántartóval való összekapcsolás előtt!



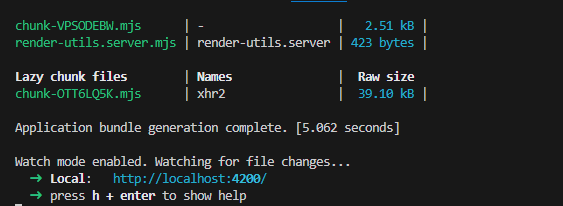
Az alábbi kép a szerver működését tartalmazza a regisztrációs felületnél, ahol a fiókot létrehozhatjuk a személyes adataink megadását megelőzően! A felhasználó itt megadhatja a nevét, email címét, valamint a jelszót a regisztráció után!



Ezt követően a bejelentkezésnél ha már a felhasználó rendelkezik egy létrehozott profillal, akkor elég megadni az email címét, és jelszavát a belépéshez!

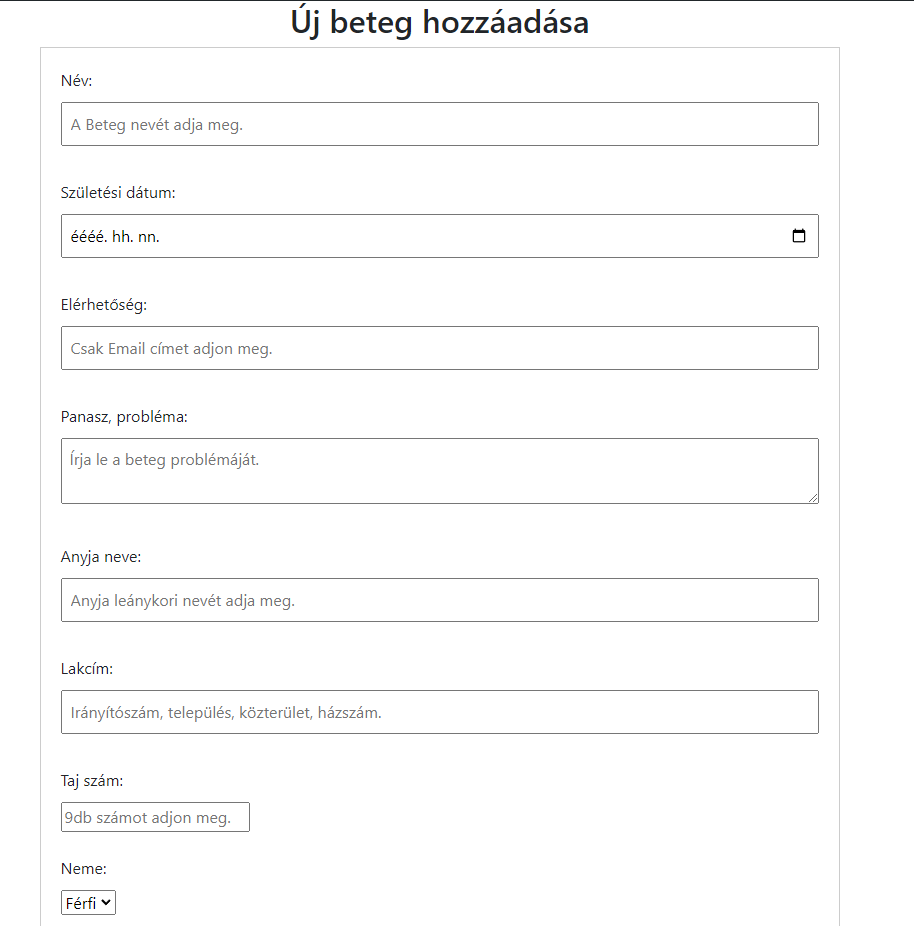
Az “Elfelejtette a jelszót?” kérdésnél ha elveszett a jelszó, akkor lehetőség van egy új kérésére,  ha ez megtörtént, akkor a felhasználó már ezzel visszaállíthatja a belépését!

A szerver egy parancs megadása után elindul, megkapjuk az esetleges localhost számot, hogy a kiszolgáló milyen linken válik elérhetővé. Működése az alábbi módon történik: megadjuk az indítási parancsot, majd azt követően kapjuk meg a helyi kiszolgáló számát!



Az indítási parancs: ng serve  - -open

Az indítás megadása után betölti a szükséges fájlokat, majd ezt követően adja meg a kiszolgáló a host számot, és ezt követően a szerver működésbe lép egy 4200-as localhost-on!

****Frontend Betegnyilvántartó Nagy-Pál Olivér

Betegnyilvántartó Frontend Dokumentáció

A betegnyilvántartó frontend alkalmazás egy Angular keretrendszerben készült alkalmazás, amely lehetővé teszi új betegek felvételét, a már regisztrált betegek szerkesztését és törlését. Az alkalmazás továbbá lehetővé teszi a betegek adatainak helyi tárolását a böngésző localStorage-jében.

Betegnyilvántartó Funkciók

Betegfelvétel:

Az alkalmazásban elérhető funkciók közül az egyik legfontosabb a betegfelvétel. Az űrlap kitöltésével könnyen és gyorsan lehet új betegeket felvenni az alkalmazásba. A betegfelvétel során a felhasználóknak meg kell adniuk bizonyos alapvető információkat a betegről, hogy azokat később könnyen kezelni és nyomon követni lehessen.

A következő adatokat kell megadni a beteg felvételéhez

Név: A beteg teljes nevének rögzítése az egészségügyi nyilvántartásokban és az ezzel kapcsolatos szoftverekben alapvető fontosságú. A beteg teljes nevének megadása lehetővé teszi az egészségügyi szakemberek számára, hogy egyértelműen azonosítsák a beteget és biztosítsák az egészségügyi információk pontos nyilvántartását és kezelését.

A beteg teljes nevének részletezése magában foglalja a keresztnevet, a középső nevet (ha van), valamint a vezetéknevet vagy családnevet. A keresztnév az egyén személyazonosságát jelöli, míg a vezetéknév a családi kapcsolatokat tükrözi. A középső név esetleges felvétele további azonosítást és differenciálást tesz lehetővé több személy esetén ugyanolyan kereszt- és vezetéknévvel.

A beteg teljes nevének pontos rögzítése segíti az egészségügyi személyzetet az esetleges azonosítási hibák elkerülésében, valamint a betegdokumentációk és egészségügyi rekordok pontos nyilvántartásában. Az egészségügyi rendszerekben azonosító adatokként a beteg teljes nevét gyakran azonosítóként használják, amelyek alapján azonosítható és hozzáférhetővé válnak az adott beteg egészségügyi adatai és előzményei.

A beteg teljes nevének pontos és részletes rögzítése lehetővé teszi az egészségügyi intézmények számára, hogy hatékonyan kezeljék a betegellátást és biztosítsák a betegek számára a megfelelő és személyre szabott egészségügyi ellátást.

Születési dátum: A beteg születési ideje, amely fontos információ a beteg kezeléséhez és nyomon követéséhez. A születési dátum alapján az orvosok megállapíthatják a beteg életkorát, ami kulcsfontosságú lehet a diagnózisban és a kezelési terv kidolgozásában. Például bizonyos betegségek kockázata vagy a kezelés hatékonysága függhet az életkortól. Emellett a születési dátum alapján lehetőség van az évfordulók, rendszeres vizsgálatok vagy oltások emlékeztetőinek beállítására az egészségügyi rendszerben.

Elérhetőség: Az elérhetőségi adatok rögzítése fontos lépés a betegnyilvántartásban, mivel lehetővé teszi az orvosok számára, hogy könnyen kapcsolatba lépjenek a beteggel. Az elérhetőségi adatok lehetnek telefonos számok, e-mail címek vagy akár postai címek is, amelyek révén az egészségügyi szakemberek tájékoztatást kaphatnak a beteg állapotáról, további vizsgálatokról vagy kezelésekről.

A telefonszámok és e-mail címek gyors és hatékony kommunikációt tesznek lehetővé az orvosok és a betegek között. Például egy sürgős esetben az orvosok telefonon keresztül értesíthetik a beteget azonnali intézkedésekről vagy sürgős konzultációkról. Az e-mail címek pedig lehetővé teszik az írásos kommunikációt, például vizsgálati eredmények vagy kezelési útmutatók küldését.

Fontos, hogy az elérhetőségi adatokat bizalmasan és biztonságosan kezeljük, hogy megfeleljenek az adatvédelmi előírásoknak és a betegek biztonságának. Az alkalmazásnak megfelelő biztonsági intézkedéseket kell bevezetnie az ilyen érzékeny adatok védelme érdekében, például titkosítási technológiák vagy hozzáférési korlátozások segítségével.

Az elérhetőségi adatok rögzítése és rendszerezése segít az egészségügyi szolgáltatóknak abban, hogy hatékonyan kommunikáljanak a betegekkel, gondoskodjanak róluk és biztosítsák, hogy a betegek számára mindig elérhető legyen az egészségügyi ellátás és támogatás.

Panasz, Probléma: A beteg panaszainak és problémáinak rögzítése az egyik legfontosabb lépés a betegnyilvántartásban, mivel ezek az információk alapvetőek a beteg diagnózisának és kezelésének megértéséhez. A beteg panaszai és problémái lehetnek fizikai, érzelmi vagy mentális jellegűek, és segítenek az orvosoknak azonosítani az egészségügyi problémák forrását és természetét.

Az alkalmazásban a beteg panaszainak és problémáinak rögzítése általában egy szabad szöveges mező segítségével történik, amely lehetővé teszi a betegek számára, hogy részletesen leírják azokat. A beteg panaszainak és problémáinak rögzítése segíti az orvosokat abban, hogy teljes képet kapjanak a beteg állapotáról, és hatékonyan tudjanak diagnosztizálni és kezelni.Az orvosok a beteg panaszai és problémái alapján készíthetnek diagnózist és kezelési tervet, amely segít a beteg gyógyulásában és egészségének helyreállításában. Például egy részletesen leírt fájdalom vagy tünet segíthet az orvosoknak azonosítani azokat az alapvető problémákat, amelyekre a kezelést ki kell dolgozni.

Anyja neve: A beteg anyjának neve, ami egy fontos azonosító adat.

Lakcím: A beteg lakcíme az egyik legfontosabb személyes adat az egészségügyi nyilvántartásokban, amely lehetővé teszi az egészségügyi szakemberek és intézmények számára, hogy kapcsolatba lépjenek a beteggel, postai levelezést küldjenek neki, vagy akár otthoni látogatásokat is végezzenek szükség esetén.

A lakcím pontos rögzítése biztosítja, hogy az egészségügyi intézmények és szakemberek tudják, hol tartózkodik a beteg, és könnyen elérhetik őt fontos információk továbbításához vagy sürgősségi beavatkozásokhoz. Emellett segíti az egészségügyi rendszereket abban is, hogy megfelelően irányítsák az egészségügyi szolgáltatásokat és erőforrásokat az adott területen élő betegek felé.

A beteg lakcímének rögzítése továbbá lehetővé teszi az egészségügyi intézmények számára, hogy nyomon kövessék a beteg lakóhelyét és ennek megfelelően kezeljék az esetleges járványok vagy közegészségügyi problémák kockázatait.

Fontos megjegyezni, hogy a lakcím adatvédelmi szempontból is érzékeny információ, ezért kiemelt figyelmet kell fordítani annak biztonságos tárolására és kezelésére az egészségügyi rendszerekben. Az adatvédelem és a biztonságos adatkezelés elengedhetetlen az egészségügyi adatok védelme és a betegek bizalmának megőrzése érdekében.

TAJ szám: A TAJ szám egyedi azonosítószámot jelent, amelyet minden magyar állampolgár kap, és az egészségügyi ellátás során azonosítja a beteget. Ez az azonosítószám alapvető fontosságú az egészségügyi rendszerben, mivel segítségével az orvosok és az egészségügyi szolgáltatók könnyen azonosítani tudják a beteget, hozzáférhetnek az egészségügyi előzményeikhez és nyomon követhetik az egészségügyi ellátásukat.A TAJ szám általában egy 9 vagy 10 számjegyből álló kódsorozat, amely egyedileg azonosítja a beteget az egészségügyi rendszerben. Ez a szám tartalmazza a beteg születési évét, a neme kódját, valamint egyedi azonosító számokat, amelyek együttesen egyedileg azonosítják a beteget.Az alkalmazásban a TAJ szám rögzítése biztosítja, hogy az egészségügyi szakemberek hozzáférjenek az egészségügyi előzményekhez és az aktuális egészségügyi információkhoz a beteg számára. Ez segít az orvosoknak és a kezelést végző egészségügyi szakembereknek abban, hogy hatékonyan kezeljék a beteget és biztosítsák a megfelelő egészségügyi ellátást.

Nem: A beteg nemének rögzítése az egyik alapvető információ a betegnyilvántartásban, amely fontos szerepet játszik a beteg kezelése és gondozása során. A beteg nemének megadása lehetővé teszi az egészségügyi szakemberek számára, hogy figyelembe vegyék a nem-specifikus kockázati tényezőket, valamint az esetleges biológiai különbségeket a kezelés során.

Az egyes nemeknek sajátos egészségügyi kockázataik és igényeik vannak, ezért fontos, hogy az egészségügyi rendszer lehetőséget biztosítson a betegek nemének nyilvántartására. Például bizonyos betegségek vagy egészségügyi állapotok előfordulási gyakorisága különbözhet a nemek között, így az orvosoknak és a kezelési tervet kidolgozó szakembereknek figyelembe kell venniük ezeket az eltéréseket a megfelelő diagnózis és kezelés érdekében.

Az alkalmazásban a beteg nemének rögzítése lehetővé teszi az egészségügyi szakemberek számára, hogy személyre szabott és hatékony kezelési tervet dolgozzanak ki a betegek számára. Például bizonyos gyógyszerek hatékonysága vagy mellékhatásai eltérőek lehetnek a nemek között, ezért fontos, hogy az orvosok ezeket az információkat figyelembe vegyék a kezelés során.

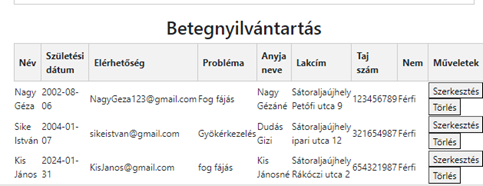
Az alkalmazásban a beteg nemének rögzítése általában egy egyszerű legördülő menü vagy választógombok segítségével történik, amely lehetővé teszi a felhasználók számára, hogy könnyen kiválasszák a megfelelő lehetőséget. Fontos, hogy az alkalmazás biztosítsa a nem-bináris vagy egyéb identitásokat is lehetőséget a megadásra, hogy minden beteg számára inkluzív és tiszteletteljes legyen az egészségügyi ellátás.

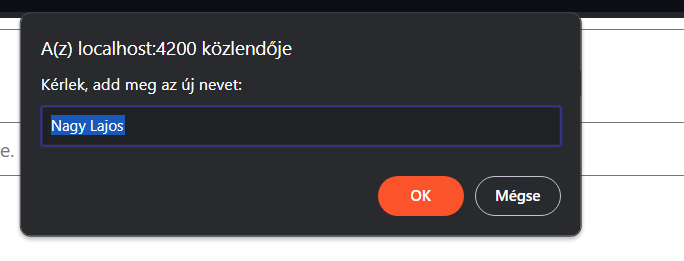
Az egészségügyi rendszerben a betegek nemének rögzítése fontos lépés a betegek egészségének és jólétének megőrzése érdekében, és segíti az egészségügyi szakembereket abban, hogy hatékonyan kezeljék és támogassák a betegeket az egészségügyi problémáik megoldásában.

Ezek az adatok lehetővé teszik az orvosok számára, hogy hatékonyan kezeljék és nyomon kövessék a betegeket az alkalmazásban. A betegfelvétel során rögzített adatok segíthetnek a pontos diagnózisban és a hatékony kezelési terv kidolgozásában, elősegítve ezzel a betegek gyorsabb felépülését és egészségük javítását. Az űrlap egyszerű és felhasználóbarát felépítése segíti a felhasználókat a gyors és pontos adatbevitelben, ezáltal minimalizálva a hibalehetőségeket és növelve az alkalmazás hatékonyságát és felhasználói elégedettségét.

 Felhasznált Technológiák

Az Angular egy alkalmazástervező keretrendszer és fejlesztői platform hatékony és kifinomult egyoldalas alkalmazások létrehozásához. Ezek az Angular dokumentumok segítenek az Angular keretrendszer és fejlesztői platform elsajátításában és használatában, az első alkalmazástól a bonyolult egyoldalas alkalmazások optimalizálásáig a vállalatok számára. Az oktatóanyagok és útmutatók letölthető példákat tartalmaznak a projektek elindításához.

Beteg adatainak szerkesztése

A már regisztrált betegek adatainak szerkesztése fontos funkció egy betegnyilvántartó rendszerben, mivel lehetővé teszi az egészségügyi szakemberek számára, hogy frissítsék és pontosítsák a betegek adatait idővel bekövetkező változások vagy új információk alapján. Az adatok szerkesztése általában a beteg adatainak megjelenítése után, a megfelelő sorban található "Szerkesztés" gombra kattintva történik.

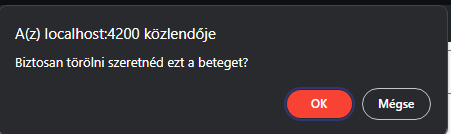
Amikor a felhasználó kiválasztja a szerkesztés lehetőségét, az alkalmazásnak meg kell jelenítenie egy űrlapot vagy ablakot, amely tartalmazza a beteg jelenlegi adatait előre kitöltött mezőkkel. Ez lehetővé teszi a felhasználó számára, hogy könnyen módosítsa vagy frissítse a beteg adatait, például a nevet, a születési dátumot, az elérhetőségi információkat vagy akár a lakcímet.

Fontos, hogy az adatok szerkesztése során az alkalmazás ellenőrizze és validálja a felhasználó által megadott adatokat, hogy biztosítsa azok pontosságát és megfelelőségét. Például a születési dátumnak helyes formátumban kell lennie, az elérhetőségi információknak pedig érvényesnek és könnyen kapcsolatba léphetőnek kell lenniük.

Miután a felhasználó befejezte az adatok szerkesztését, az alkalmazásnak lehetőséget kell biztosítania a változtatások mentésére, és frissítenie kell az adatokat a betegnyilvántartásban. Ezután általában visszatér a felhasználó a betegnyilvántartás főoldalára vagy listájára, ahol láthatja a frissített adatokat.

Az adatok szerkesztésének lehetősége lehetővé teszi az egészségügyi szakemberek számára, hogy naprakészen tartsák a beteginformációkat, ami hozzájárul az egészségügyi ellátás minőségének és hatékonyságának növeléséhez, valamint a betegek elégedettségének és bizalmának megőrzéséhez.

Beteg adatainak törlése

A betegtörlés funkció célja az alkalmazásból már regisztrált betegek eltávolítása. A funkció előtt a felhasználót megerősítő kérelem jelenik meg, hogy biztosan törölni szeretné-e a beteget. Ez segít megelőzni véletlen törléseket, és lehetővé teszi a felhasználó számára, hogy megerősítse a döntését. A betegtörlés gondosan megvalósított mechanizmusának köszönhetően biztosítjuk a betegek adatainak védelmét és a felhasználói élményt.

CSS

Az alkalmazásban a betegek adatait egy JavaScript tömbben tároljuk, amely lehetővé teszi az adatok dinamikus kezelését és manipulálását. Az adatokat a böngésző localStorage-jében is tároljuk, így biztosítva, hogy a felhasználók adatai ne vesszenek el a böngészési  munkamenetek között. Ezáltal a felhasználók számára kényelmes és zökkenőmentes élményt nyújtunk.

Az alkalmazás frontend részét egy űrlap segítségével valósítjuk meg. Az űrlap beküldésekor a JavaScript kód validálja az adatokat, hogy biztosítsa az adatok megfelelőségét és érvényességét. Ez fontos lépés az adatok megbízhatóságának és konzisztenciájának biztosításához. A megfelelő adatokkal frissítjük a betegek táblázatát és a localStorage-t, hogy az alkalmazás mindig naprakész legyen és a legfrissebb adatokat jelenítse meg a felhasználóknak.

A betegek táblázatát dinamikusan frissítjük azáltal, hogy figyelemmel kísérjük az űrlap beküldését, új betegek hozzáadását, meglévő betegek szerkesztését vagy törlését. Ez a megközelítés lehetővé teszi az alkalmazás számára, hogy valós időben reagáljon a felhasználói interakciókra, és azonnal frissítse az adatokat a felhasználók számára.

Az alkalmazás tesztelésénél fontos lépés volt a saját magam általi tesztelés különböző böngészőkben. Ez lehetővé tette számomra, hogy meggyőződjek arról, hogy az alkalmazás megfelelően működik-e az összes támogatott böngészőben. Különféle böngészőkben teszteltem az alkalmazást, beleértve a Brave, Google Chrome és Microsoft Edge és Opera GX böngészőket is. A tesztelés során megfigyeltem, hogy az alkalmazás minden böngészőben megfelelően működik-e, és hogy az esetleges különbségeket hogyan lehet kezelni vagy javítani.

Emellett terveztem egy sötét módot is a betegnyilvántartóba, de végül úgy döntöttem, hogy kiveszem a kódját. Ennek oka az volt, hogy úgy éreztem, hogy az alkalmazás jelenlegi verziójában a sötét mód nem illeszkedik teljesen a designhoz, és ez esztétikailag zavaró lehetne a felhasználók számára. A sötét mód kikommentelt kódját megtalálhatják a CSS fájlban, de jelenleg nem aktív, és nem befolyásolja az alkalmazás működését.

Backend Regisztráció Gabóczi Ádám

A signUp függvény egy olyan függvény amely aszinkron működésű. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await művelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Egy konstans típusú email nevű változót ami a kérés törzséből kapja az értékét ez lesz a regisztráló email-címe amit ebben fogunk tárolni. Létrehozunk egy globális var típusú változót salt néven ami a só lesz és ebben tároljuk azt az értéket amit a bcrypt könytárnak a getSync(10) metódusa generál a zárójelben lévő számérték a költségfaktort jelenti minél nagyobb ez a szám annál időigényesebb lesz a generálása de annál biztonságosabb is. Egy generált só véletlenszerű karakterekből áll például ez: 23$Asa$sz$k$KUGk$. Ezután létrehozunk egy hash nevű globális változót amiben a bcrypt könytár hashSync() metódusával aminek az egyik paramétere a kérés törzséből érkezett jelszó amit a felhasználó beírt és a másik paramétere a generált só lesz. Aztán ez a metódus a jelszót a só val össze hash-eli. Ezáltal a hash-elt jelszót tudjuk tárolni az adatbázisban és nem magát a jelszót titkosítatlanul és lekérni az adatbázisból egy összehasonlító metódussal fogjuk. Aztán a bcrypt hash-et eltároljuk egy konstans típusú password nevű változóba. Generálunk egy tokent ez egy rövid élettartamú token lesz aminek a feladata a felhasználó azonosítása és a felhasználó fiókjának megerősítése. A tokent a crypto modul randomBytes(16) függvényével tesszük meg a zárójelben megadható érték a bájtok hossza. Ezután ezt a 16 karakterhosszúságú bájtot sztringgé alakítjuk .toString(’hexa’) metódussal ami hexadecimális értékké fogja alakítani és mindegyik bájthoz 2 karakter fog párosodni ezáltal a token 32 karakterhosszúságú lesz. Aztán ha megtörténtek a regisztrációhoz szükséges adatok bevitele és a token generálása akkor egy új felhasználót hozunk létre a User táblában ezt a User.create() metódussal hozzuk létre amiben egy objektum kulcs értékei azonosítják az adatbázis mezőit és egy értéknek a kulcs párjai az új felhasználó értékeit. Ezután a létrehozott új felhasználó adatai eltároljuk egy record nevű konstans típusú változóban. Ezután létrehozunk egy globális változót transporter néven. A nodemailer csomag createTransport({}) objektumában megadjuk az smtp protokollon keresztüli email küldésének a konfigurációját. Ez a MAIL\_FROM kulcs az üzenetküldő email-címe a feladó címe. MAIL\_HOST: Az az érték ami az SMTP szerver címe lesz ezt a címet fogja használni az alkalmazás az üzenetküldésre. MAIL\_PORT: Ez az a port amin keresztül az alkalmazás kapcsolódik az SMTP szerverhez. A GMAIL a fiókra 465-ös portot használja SSL/TLS kapcsolaton keresztül. MAIL\_AUTH\_PASS: Ez az a jelszó amit felhasznál az SMTP szerver hitelesítéséhez. Aztán a verificationLink nevű változóban egyesítjük az alkalmazás ügyféloldali címét és egy végpontot írunk amire végén ottvan a felhasználó tokenje ennek a végpontnak a célja a felhasználó fiókjának a megerősítése ezt a linket fogja elküldeni a regisztráció megerősítéséhez. Aztán egy mailOptions nevű globális változóban konfiguráljuk az email feladóját, címzettjét, email tárgyát és az üzenet tartalmát ami tartalmazni fogja azt a linket ahol megerősítjük a regisztrált fiókot. Aztán await kulcsszóval megállítjuk az async függvényt és végrehajtja transporter.sendMail(mailOptions) utasítást. A transporter változóban benne van az üzenetküldő konfigurációjának beállítása. Erre a változóra amiben az objektum van meghívunk egy sendMail() metódust aminek a paraméterében benne van az az objektum ami az email feladóját, címzettjét, tárgyát, tartalmát definiálja. Ha ez az utasítás befejeződött akkor az aszinkron függvény tovább fut. Ha nincs hiba akkor visszaad egy objektumot ami státusz üzenetet tartalmaz és üzenetet és visszaadja a felhasználónak azt az objektumot amiben az új felhasználó adatait tartalmazzák. Ha bármilyen hiba van akkor a catch ág error objektuma megjelenik.

Regisztráció megerősítése

Az emailben érkezett megerősítő link fogja tartalmazni a regisztrált felhasználó tokenjét az URL paraméterében. A linkre kattintva átnavigálja a felhasználót egy másik oldalra ahol megkell erősíteni a regisztrált fiókot. Ezen a végponton a signUpVerify függvény működik. A signUpVerify függvény egy olyan függvény amely aszinkron működésű. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await művelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van egy token nevű konstans típusú változó a kérés paraméteréből veszi a tokent ami regisztrált felhasználó tokenje. Van egy user nevű konstans típusú változó ami tárolja egy await művelet eredményét. A program további futása megáll amíg az await művelet nem fejeződik be azután tovább fut. A művelet a User táblából pontosan egy rekordot fog kereseni azt a rekordot amit megadtunk a where objektumban. Az objektum kulcsa lesz a tábla mezője ahol egyezést keresünk. Az objektum érték párja az az érték aminek lennie kell a mező értékének. A token mezőnek annak a tokennek kell lenni amellyel a felhasználó regisztrált. Az isVerified mezőnek 0-s értéket kell tárolnia ami azt jelzi hogy nincs megerősítve. Ha ezt a rekordot megtalálta a keresés akkor ezt a user változóban tárolja. Egy feltételes elágazás megvizsglálja hogy a user változó nem üres e ha nem üres abban az esetben létrehoz egy record nevű konstans típusú változót aminek az értéke egy await művelet eredménye. Az await művelet a User táblában frissíti a tokent üres sztringre mert már regisztráció után nincs szükség a megerősítő tokenre és az isVerified mező értékét 1 re ami azt jelzi hogy meg van erősítve. Ezeket a műveleteket ott végzi el ahol a User tábla azonosítója megegyezik azzal az azonosítóval amivel a regisztrált felhasználó rendelkezik. Itt a sima kulcs érték pár helyett az Operators csomag equals metódusát használjuk fel mert kifejezettebb. Minden regisztrált felhasználónak egyedi azonosítója van. Ha ezeket elvégezte akkor megvizsgála a frissítés sikerességét. A frissített objektum nulladik eleme mindig a frissítés sikerességét tartalmazza ha 0 az értéke akkor nem sikerült frissíteni akkor egy error objektumot ad vissza ami tartalmaz egy üzenetet és egy másik kulcs érték párt és egy 403-as hibakóddal ami azt jelzi hogy a kérést nem lehet teljesíteni. Ha sikerült a frissítés akkor az értéke 1 és egy json objektumot küldd vissza egy sikeres státusszal üzenettel és a frissített objektummal. Ha a feltétel nem talál ilyen felhasználót aminek a mezői megfelelnek a feltételnek akkor egy error objektumot ad vissza egy üzenettel ami azt tartalmazza hogy a felhasználó már meg lett erősítve vagy a token érvénytelen. Egy 403-as hibakóddal.

Bejelentkezés

A Login függvény egy olyan függvény amely aszinkron működésű. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await művelet be nem fejeződik majd utána fut tovább. Van egy email nevű konstans típusú változó ami a kérés törzséből kapja az adatot ami a felhasználó által beírt email-cím. Van egy password nevű konstans típusú változó ami a kérés törzséből kapja az adatot ez a felhasználó által beírt jelszó. Aztán egy konstans típusú változóban a checkUserVerification nevű változóban van egy érték. Ez az érték annak a felhasználónak az adatai amit megtalált a User.findOne() metódus ott ahol a táblában a mező értéke ez az objektum kulcsa az az érték amit az objektumban értékként megadtunk. Ennek a keresésnek az eredménye egy objektum ami tartalmazza a megadott email cím alapján azt a rekordot aminek az email mezője az az érték amit a felhasználó beírt. Aztán egy feltétel megvizsgálja hogy ez a rekord ami tartalmazza ezt az email címet hogy nem e üres. Ha üres tehét a keresés siketelen akkor visszaad egy json objektumot ami tartalmaz egy üzenetet hogy Nincs ilyen felhasználó regisztrálva ezzel az email-címmel és a hibának a helyét és egy 404-es fájl nem található hibakódot. Ha e keresés eredményes tehát megtalálta az email-cím alapján a rekordot a táblában akkor megvizsgálja ennek a rekordnak a isVerified mezőjét ami azt jelenti hogy meg lett-e erősítve ha nem lett megerősítve akkor a user változó üres lesz mert nincs ilyen felhasználó a táblában és visszaad egy error objektumot ami tartalmaz egy üzenetet és egy hiba helyét hogy melyik mezőnél van hiba és ezután egy 401-es hibakódot visszaad ami azt jelzi hogy hitelesítés szükséges és kilép. Ha a user változó értéke egy olyan objektum ami meg lett erősítve és a email cím megegyezik akkor a user objektum ezt a rekordot fogja tartalmazni. Aztán van egy egy konstans típusú isMatched nevű változó aminek az értéke egy await utasítás lesz igaz vagy hamis. Az await utasítás a bcrypt modul compare metódusát valósítja meg ami összehasonlítja a rekordban lévő hashelt jelszót és a felhasználó által megadott jelszót. A felhasználó által megadott jelszót hasheli azzal a sóval amivel a táblában tárolt hashelt jelszó tartalmaz. Aztán egy feltétel megvizsgálja hogy az egyezés igaz vagy hamis. Ha nem egyezik akkor visszaadunk egy error objektumot ami tartalmaz egy üzenetet azt ami a hiba oka hogy hibás email-cím vagy jelszó és egy 401-es hibakód hogy a felhasználó által megadott hitelesítési adatok érvénytelenek. Ha a jelszó egyezés igaz akkor egy objektumban eltároljuk ennek a rekordnak azon értékeit amit a tokennek tartalmaznia kell ez rekord neve, rekord email-címe, rekord teljes név, és a jogosultság. Visszaküldjük egy json objektumot ami tartalmazza ezeket az adatokat és ebből hozzuk létre a tokent. jwt.sign() metódussal létrehozzuk a tokent a felhasználó adataiból és egy titkos kulccsal aláírjuk ezt. A titkos kulcsot egy környezeti változóban tároljuk. A jwt.sign() első paramétere tartalmazza azt az objektumot amiben megtalálhatók a felhasználó adatai a másik paramétere a titkos kulcs amivel aláírjuk és 7napig érvényes egy token. Minden bejelentkezéskor új token generálódik. A token nem tárolódik az adatbázisban a biztonság érdekében.

ÉrvényesE

A getLoggedInUser függvény egy olyan függvény amely aszinkron működésű. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await művelet be nem fejeződik majd utána fut tovább. Van egy globális típusú token nevű változónk aminek az értéke a kérés fejlécének az Authorization értéke. Az authorization értéke az a felhasználó tokenje. Aztán van egy feltételes elágazás ahl a feltétel azt vizsgálja meg hogy a token nem e üres. Ha a token változó üres akkor visszaad egy error nevű objektumot ami tartalmaz egy üzenetet amiben az van hogy Jogosulatlan felhasználó mert ha nincs token akkor ezt funkciót nem lehet használni tehát nincs jogosultsága. Visszaküld egy 401-es hibakódot ami azt jelenti hogy nem jogosult. De ha van token tehát a token változó nem üres akkor a jwt modul verify() metódusa fut le. Ez a metódus megvizsgálja a token érvényességét és a token tartalmát. A jwt.verify() metódus első paramétere azt teszi hogy a token elejéről kicseréli a Bearer szöveget és üres sztringre állítja tehát kitörli és így lesz belőle maga a token. Ezután a környezeti változóban lévő titkos kulcsot használjuk fel a token érvényesítésének az ellenőrzésére. Aztán van egy callback függvény aminek az első paramétere err. Ezután megvizsgljuk hogy történt e hiba. Ha történt hiba a token érvényesítése közben akkor egy error objektumot ad vissza ami tartalmaz egy üzenetet ami ez Jogosulatlan felhasználó ez a hiba oka és egy mezőt hogy hol történt a hiba. Majd vissza küldi egy 401-es hibakóddal ami azt jelenti nem jogosult. Van az elágazásnak egy else ága ami akkor fut ha a feltétel kiértékelése hamis tehát ha nincs hiba. Ha nincs hiba akkor visszaküld egy json objektumot ami tartalmaz egy üzenetet ami az lesz hogy sikeres és visszaküldi a dekódolt tokent. A dekódolt token ebben az esetben a felhasználó azonosítója, a felhasználó email-címe és a felhasználó teljes neve, és a felhasználó jogosultsága.

Elfelejtett jelszó

A forgotPassword függvény egy olyan függvény amely aszinkron működésű. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await mávelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van egy globális email nevű változó ami a kérés törzséből fogja kapni az adatot . Ez az adat a felhasználó által beírt email-cím lesz amire az az üzenet fog menni ami tartalmazza azt a linket amelyik megnyitásakor a felhasználó egy új jelszót tud beállítani. Van egy token nevű globális változó. Ez a token nevű változó fogja tárolni a tokent. Tokent a crypto modul randomBytes(16) metódusával fogunk készíteni a zárójelben lévő szám a bájtok karakterhosszúságát mutatja amit be lehet állítani tehát ez generált egy 16 karakterhosszúságú bájtot amit egy .toString(’hexa’) metódussal hexadecimálissá alakítunk. Minden bájthoz 2 hexadecimális karakter párosul ezért lesz a token 32 karakterhosszúságú. Van egy konstans típusú result nevű változó aminek az értéke egy await művelet eredménye lesz. Az await művelet megállítja függvény futását addig amíg az await művelet be nem fejeződik azután tovább fut. Ez az await művelet a User táblában frissíti az objektumban megadott kulcs értékét ami a táblában a mező. Token mező értékét arra frissíti amit generált. Ezt annál rekordnál teszi meg ahol az email mező megegyezik azzal az email címmel ami benne van a táblában. Ezután létrehozunk egy globális változót transporter néven. A nodemailer csomag createTransport({}) objektumában megadjuk az smtp protokollon keresztüli email küldésének a konfigurációját. Ez a MAIL\_FROM kulcs az üzenetküldő email-címe a feladó címe. MAIL\_HOST: Az az érték ami az SMTP szerver címe lesz ezt a címet fogja használni az alkalmazás az üzenetküldésre. MAIL\_PORT: Ez az a port amin keresztül az alkalmazás kapcsolódik az SMTP szerverhez. A GMAIL a fiókra 465-ös portot használja SSL/TLS kapcsolaton keresztül. MAIL\_AUTH\_PASS: Ez az a jelszó amit felhasznál az SMTP szerver hitelesítéséhez. Aztán a verificationLink nevű változóban egyesítjük az alkalmazás ügyféloldali címét és egy végpontot írunk amire végén ottvan a felhasználó tokenje. Aztán egy mailOptions nevű globális változóban konfiguráljuk az email feladóját, címzettjét, email tárgyát és az üzenet tartalmát ami tartalmazni fogja azt a linket ahol tok erre a linkre kattintáskor fog átnavigálni egy oldalra ahol a felhasználó megadja az új jelszavát. Aztán await kulcsszóval megállítjuk az async függvényt és végrehajtja transporter.sendMail(mailOptions) utasítást. A transporter változóban benne van az üzenetküldő konfigurációjának beállítása. Erre a változóra amiben az objektum van meghívunk egy sendMail() metódust aminek a paraméterében benne van az az objektum ami az email feladóját, címzettjét, tárgyát, tartalmát definiálja. Ha ez az utasítás befejeződött akkor az aszinkron függvény tovább fut. Ha nincs hiba akkor visszaad egy objektumot.

Elfelejtett jelszó megerősítése

A forgotPasswordVerify függvény egy olyan függvény amely aszinkron működésű. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await mávelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. A megerősítő linkre kattintáskor egy másik oldalra fog navigálni amin lesz egy űrlap amin meg lehet adni az új jelszót. Van egy token nevű globális változó aminek az értéke a kérés paraméteréből jövő token. Tehát ez lesz annak a felhasználónak tokenje amelyik új jelszót akar beállítani. Majd van egy user nevű változó ami egy await művelet eredményét fogja tárolni. Ez az await művelet addig leállítja a függvény folyamatát addig amíg az await művelet be nem fejeződik. A User táblából fog keresni pontosan egy rekordot ott ahol a rekord mezője megegyezik azzal az értékkel ami a felhasználó tokenje. Token alapján megtalálta a felhasználó rekordját a táblában. Van egy feltétel ami megvizsgálja hogy a user változó üres e vagy sem. Ha üres van egy error objektum amiben egy üzenet ami kiírja a hiba okát az lesz hogy Érvénytelen token és a mezőt ahol van a hiba és egy hibakódot ad vissza 401-est ami azt jelenti hogy nincs jogosultsága. Ha nem üres akkor egy üzenet ad vissza hogy Érvényes token és sikeres azonosítás.

Új jelszó beállítása

A sresetPassword függvény egy olyan függvény amely aszinkron működésű. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await mávelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van egy globális token nevű változó ami az értékét úgy fogja megkapni amelyik végponton van az új jelszó beállításának a lehetősége arról a végpontról egy változóban lekérjük a kérés paraméterét mert az tárolja a tokent. Az egy token nevű változóban van és aztán az űrlapon keresztül rejtve elküldjük azt a változót amelyik tárolja a tokent formban egy bemeneti mezőnek a name értéke a változó neve erre a POST végpontra. Aztán itt req.body.token ként hivatkozunk az értékére. Van egy salt nevű változó amelynek az értéke egy bcrypt modul genSaltSync(10) metódusával generált só a 10-es értéke a biztonságot jelzi minél nagyob ez a szám annál biztonságosabb lesz a só. Van egy hash nevű globális változó amely tartalmazza a bcrypt modul hashSync()metódusának hashelését. Az első paramétere a beírt új jelszó. A második paramétere a generált só. Van egy konstans típusú result nevű változó amiben egy rekord frissítés eredménye fog tárolódni. Egy új megadott jelszónak a hashelt értéke fog tárolódni. Az isVerified mező értéke 1 tehát megerősített. A token üres lesz mert a jelszó megváltoztatása után nincs szükség tokenre.Ezeket a módosításokat annál a rekordnál végzi el ahol a token mező értéke a táblában megegyezik azzal a tokennel amelyik felhasználó tokenje használja ezt a funkciót. Aztán a frissített rekord a result változóban van. Visszaküld egy json objektumot ami tartalmazza a függvény sikerességét egy üzenetet hogy megtörtént e az új jelszó beállítása és a frissített rekord objektumát.

Jelszó megváltoztatása bejelentkezve

A changePassword függvény egy olyan függvény amely aszinkron működésű. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await művelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van egy globális típusú változó id névvel. Ez a változó fogja tartalmazni a bejelentkezett felhasználó id-ját. Van egy globális típusú változó salt néven ez fogja tartalmazni a bcrypt modul getSaltSync() metódusának felhasználásával generált sót aminek a összetettségét a zárójelben adjuk meg minél nagyobb ez a szám annál bonyolultabb sót generál. Van egy globális hash nevű változó ami a bcrypt modul hashSync() metódusa felhasználásával hasheli a felhasználó által megadott jelszót a sóval. A hashSync() metódusnak az első paramétere a felhasználó által megadott új jelszó. A második paramétere a generált só. Ez a két paraméter kell a jelszó hasheléséhez. Van egy konstans típusú result nevű változó aminek az értéke egy frissített rekordnak az adatai. A frissítés a User táblában frissít ezt egy objektumban adtuk mega az objektum kulcs reprezentálja a tábla mezőjét és az objektum érték reprezentálja hogy annak a rekordnak minek kell lennie. Ezt a módosítást ott végzi el annál a felhasználónál ahol az id mező értéke megegyezik az éppen most bejelentkezett felhasználó id-jával. Ennek a frissítésnek az objektumát eltároljuk a result nevű változóban. Majd visszaadunk egy json objektumot státusz üzenettel és üzenettel és magát a változtatást végrehajtó felhasználó adatait.

Név és email-cím megváltoztatása bejelentkezés után

Az updateProfile függvény egy olyan függvény amely aszinkron működésű ami azt jelenti hogy a függvény nem blokkolja a további kódfuttatást amíg egy adott művelet be nem fejeződik. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await művelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van egy globális változó aminek a neve id és ennek a változónak a neve req.user.id-ból érkezik tehát annak a felhasználónak az id-ját kapja meg amelyik fiókból ezt a kérést végrehajtják. Ez az id-ra azért lesz szükség mert ez az id-fogja azonosítani a nevet változtató felhasználót az adatbázisban. Aztán van egy fullname nevű globális változó ami a kérés testéből érkezett adatot tárolja ez lesz az a név amelyet a felhasználó adott meg erre a névre történik a teljes névnek a megváltoztatása. Van egy konstans típusú result nevű változó ami egy frissített rekordot fogja tárolni egy objektumban. A frissítést a User táblában hajtódik végre egy objektumban adjuk meg az objektum kulcsa a fullname reprezentálja a tábla fullname mezőjét és az érték reprezentálja azt amilyen értlkre kell frissíteni ebben az esetben a fullname változóban lévő adat amit a felhasználó megadott. Záradékban ez a művelet ott fog végrehajtódni ami a where záradékban meg van adva tehát hogy a táblában lévő rekordnak az id-jának meg kell felelnie annak az id-nak amelyik felhasználó be van jelentkezve tehát amelyik felhasználó ezt használja. Van egy feltételes vizsgálat ami azt vizsgálja hogy a result üres e vagy nem ha nem üres akkor visszaad egy json objektumot a egy státusszal egy üzeneteel és a kérés testével. Ha üres akkor visszaad egy error objektumotami tartalmaz egy 401-es hibakódot ami azt jelenti hogy nincs jogosultsága és egy üzenetet.

Összes beteg lekérése

A getAllPatients függvény egy olyan függvény amely aszinkron működésű. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await mávelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van egy globális userId nevű változó aminek az értéke a kérés paraméterének a Userid-ja lesz. A kérés paramétere azt a UserId-t tartalmazza amelyik bejelentkezett fiókból használják ezt a végpontot. Van egy feltételes elágazás ami azt vizsgálja hogy a userId-ban tárolt érték nem e üres. Abban az esetben lehet üres ha hiányzik a Userid az URL-ből vagy a Userid helytelenül van megadva vagy egyéb esetlegesen fellépő hiba. Abban az esetben ha a userId változója üres akkor visszaküldünk egy json objektumot egy hiba státusszal és egy üzenettel hogy nincs megadva felhasználói azonosító és egy 400-as hibakóddal ami azt jelenti hogy hibás kérés. Ha a userId változó nem üres tehát a kérés paraméteréből megkapja annak a felhasználónak a UserId-ját aki használja ezt a végpontot. Abban az esetben készítünk egy konstans típusú patients nevű változót amk tartalmazza egy await művelet keresésének az eredményét. Az await művelet megállítja a függvény további futását ameddig await művelet be nem fejeződik majd utána tovább fut. A Patient.finAll() metódussal megkeressük az összes olyan rekordot a táblában amelyek a feltételnek megfelelnek. A feltételt a where záradékban írjuk. Az objektum kulcsa reprezentálja a tábla mezőnevét. A kulcsnak és a tábla mező nevének meg kell egyeznie. Az objektum értéke reprezentálja azt hogy milyen értéknek kell megfelelnie ennek a mezőnek hogy kitudja választani a táblából az összes olyan rekordot amelyek a feltételnek megfelelnek. Ha megtalálta az összes olyan rekordot amelyek a feltételnek megfelelnek eltárolja a patients nevű változóban. Majd visszaadja a felhasználónak egy json objektumban egy státusz üzenettel együtt. Ennek a végpontnak a visszaadott adatai egy gombra kattintással megjelennek a felhasználónak egy táblázatban. Tehát egy felhasználónak az összes betegét adja vissza.

Egy beteg lekérése

A getOnePatient függvény egy olyan függvény amely aszinkron működésű ami azt jelenti hogy a függvény nem blokkolja a további kódfuttatást amíg egy adott művelet be nem fejeződik. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await mávelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van egy id nevű id nevű konstams típusú változó. Ez tartalmazza a kérés paraméteréből érkezett id-t. A kérés paraméteréből érkezett id annak a betegnek az id-ja amelynek az összes adatát lekérjük. A végpont paramétere úgy tartalmazza a beteg id-ját hogy a beteg adatai egy táblázatban vannak és a táblázatban vannak és minden egyes új beteg létrehozásánál egy új beteg kap egy id-t és azt elküldi a kérés paramétereként. Van egy konstans típusú patient nevű változó amelynek az értéke egy await keresés eredménye lesz. Az await művelet megállítja a függvény további futását amíg be nem fejeződik az await művelet aztán tovább fut. Ez a Patinet nevű táblában egy rekordot fog keresni a feltétel a where záradékban van megadva aminek a kulcsa a tábla mezője az objektum értéke az az érték aminek a táblában lévő rekordoknak meg kell felelniük hogy egy rekord megfeleljen a feltételnek. Megvizsgáljuk a patient változó értékét aminek ha az értéke üres akkor nincs ilyen beteg. Ha nincs betegakkor visszaküld egy json obejktumot hiba státusszal és egy üzenettel ami azt tartalmazza hogy nem talált a keresés id szerint ilyen beteget. Ha patient változó értéke nem üres tehát van ilyen beteg akkor egy json objektumban visszaküldi egy sikeres státusszal és ezzel a feltételeknek megfelelő beteg összes adatával.

Új beteg létrehozása

A createPatient függvény egy olyan függvény amely aszinkron működésű ami azt jelenti hogy a függvény nem blokkolja a további kódfuttatást amíg egy adott művelet be nem fejeződik. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await mávelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van több konstns típusú változónk firstName,lastName,motherName,idCardNumb email, identifierTyp tajNumber,nationalitplaceOfBirth,dateOfBirth,sex,country,city,postalCode,address,UserId, drugAllergy,allergy, chronicIllness, constantMedication ezek az értékeit a kéré törzséből kapja tehát abból amit a felhasználó beírt. Van egy konstans típusú record nevű változó aminek az értéke egy await művelet eredménye lesz. Az await művelet a Patient táblában egy új beteg rekordot hoz létre azokkal az adatokkal amit letrehoztunk a felhasználó beírt adataival maj visszaküldünk egy státusz és egy üzenetet hogy a beteg sikeresen hozzá lett adva a táblához. Majd visszaküldjük az objektumban objektum formátumként az új beteg adatai egy objektumban.

Beteg frissítése

A updatePatient függvény egy olyan függvény amely aszinkron működésű ami azt jelenti hogy a függvény nem blokkolja a további kódfuttatást amíg egy adott művelet be nem fejeződik. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await művelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van több konstans típusú változónk amelynek az értékei a kérés törzséből érkeznek ezek azok az adatok amit a felhasználó a beteg adatainak a frissítésekor megadhat. Aztán van e konstans típusú record nevű változó aminek az értéke egy await művelet a Patient táblában frissített rekord adatai lesznek. A frissítéskor azok a vélktozónevek vannak beírva amelyek tárolják azokat az értékeket amit a felhasználó beírt frissítésre. Ezt a frissítést abban a rekordban végzi el ez a where záradékban van megadva ahol az id mező értéke megyegyezik a beteg azonosítójának az értékével. Tehát annak a felhasználónak az adatait frissíti ahol az id megegyezik a beteg azonosítójával. Van egy feltétel a frissítés eredményét tárolja hogy történt e frissítés. Ha nem történt frissítés akkor visszaad egy json objektumot ami tartalmaz egy státuszt ami error lesz és egy üzenetet ami a hiba oka hogy nincs mit frissíteni mert nincs ilyen felhasználó és egy nem található 404-es hibakódot ami azt mutatja nincs ilyen azonosítóval rekord amit lehetne frissíteni. Ha a frissített sorok száma nem 0 tehát történt frissített rekord akkor visszaad egy státuszt hogy sikeres és egy üzenetet hogy a beteg sikeresen frissítve és egy objektumban a kérés törzsét ami tartalmazza az összes olyan adatot amire a tábla mezőit frissítette.

Beteg törlése

A deletePatient függvény egy olyan függvény amely aszinkron működésű ami azt jelenti hogy a függvény nem blokkolja a további kódfuttatást amíg egy adott művelet be nem fejeződik. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await művelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van egy id nevű konatns típusú változó amiben tárolódik a kérés paraméterében lévő patientId tehát a beteg id-ja. Majd van deleted nevű konstans típusú változó aminek az értéke egy await művelet eredménye lesz ebben az esetben egy törölt rekord. Az await művelet megállítja függvény futását addig amíg az await művelet be nem fejeződik majd befejeződése után továbbb fut. Ez az await művelet a Patient.destroy() táblában fog törölni egy rekordot. A where záradékban van megadva hogy a rekordnak milyen feltételnek kell megfelelnie. A rekord id mezőjének meg kell egyeznie azzal az id-val amit a paraméterből kértünk és egy id változóban tároltuk. Id alapján töröl egy rekordot és ezt a törölt rekordot eltárolja változóban. Egy feltételes elágazás megvizsgálja hogy a deleted véltozó értéke nem 0 tehát nincs törölt rekord. Ebben az esteben egy json objektumot küld vissza egy státusszal és egy üzenettel és egy 404-es nem található hibaüzenettel. Ha a törlés értéke nem 0 tehát 1 akkor sikeres és egy json objektum megy vissza amiben van státusz hibaüzenet és a kitörölt sor objektuma egy objektumon belül.

Diagnózis hozzáadása

A addDiagnosis függvény egy olyan függvény amely aszinkron működésű. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await mávelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van egy konstans típusú id nevű változó ami tartalmazza a kérés testében lévő id-t. Van egy konstans típusú diagnosis nevű változó ami tartalmazza kérés törzséből érkezett dignózist amit a felhasználó beírt ez fogja tartalmazni a diagnózist. Van egy konstans típusú record nevű változó ami egy await művelet eredményét fogja tartalmazni. Ebbben az esetben egy frissített rekordot fog tartalmazni. Az await művelet megállítja a program futását addig amíg az await művelet be nem fejeződik aztán tovább fut. Ebben az esetben a Patient nevű táblában fog frissíteni egy rekordot. Ennek a rekordnak a diagnosis mezőjének az értékét a diagnosis változóban lévő értékre de ezt csak annál a rekordnál ahol a rekord id-ja megegyezik azzal az id-val amelyik betegnek a diagnüzisát akarjuk müdosítani. Egy feltételes elágazással megvizsgáljuk hogy a record váltózónak az értéke nem e üres vagy benne van e frissített sor tehát az lenne az 1. Ha nincs a frissített rekord benne akkor visszaad egy státusz egy üeznetet hogy nincs ilyen id-val beteg. Ha viszont a sikerült a diagnózis frissítése akkor egy státuszt adunk vissza és egy üzenetet és a kérés törzsét ami ebben az esetben a diagnózis és a frissített beteg id-ja.

Kezelés frissítése

A updateTreatment egy olyan függvény amely aszinkron működésű ami azt jelenti hogy a függvény nem blokkolja a további kódfuttatást amíg egy adott művelet be nem fejeződik. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await művelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van több konstans típusú változó amik az értékeiket a kérés törzséből kapják. Van egy record nevű konstans változó ami tartalmazza a frissített rekordot. A Treatment táblában frissítjük a Treatment tábla mezőit azokra az változókra amikben eltároltuk a felhasználó által beírt adatokat ezt annál a rekordoknál végzi el ahol a Treatment tábla id mezőjének az értéke megegyezik azzal az id-val amelyiket kezelést módositjuk. Megvizsgáljuk egy feltételes elágazásban hogy sikerült e a frissítés tehát hogy 0 vagy 1 a frissített sorok hossza. Ha nem sikerült az azért nem sikerült mert előtte nem létezett kezelés amit módpsítnai lehet. Visszatér egy 404-es hibakóddal hogy nem található egy státusszal és egy üzenettel. Ha viszont sikerült a módosítás akkor egy státusszal üzenettel és a frissített rekorddal tér vissza.

Kezelés hozzáadása

A addTreatment függvény egy olyan függvény amely aszinkron működésű ami azt jelenti hogy a függvény nem blokkolja a további kódfuttatást amíg egy adott művelet be nem fejeződik. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await művelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van több konstans típusú változó amik az értékeiket a kérés törzséből kapják. Van egy record nevű konstans változó ami tartalmazza a frissített rekordot. Van egy record nevű konstans típusú változó ami az értékét az await művelet végeredményét kapja. A Treatment.create() metódus létrehoz egy kezelést azokkal az adatokkal amiket eltároltunk a változóban és eltárolja a rekord nevű változóban a rekordot majd megvizsgálja egy feltételes elágazásban hogy sikerült e a létrehozás. Ha a record nevű valtozó üres akkor egy 404-es hibakódot ad vissza és egy error státuszt mert nem sikerült létrehozni a kezelést. Ha viszon a record változó nem üres teháét sikerült új kezelést létrehozni akkor visszaküldi a sikeres státuszt és egy üzenetet hogy a kezelés sikeresen hozzá lett adva és az új kezelés tömbjlét vissazküldjük.

Beteg kártyájának megtekintése

A getPatientCard függvény egy olyan függvény amely aszinkron működésű. Ez a funkció a http kérés kezelésére szolgál mivel a paraméterei a req a request és a res a response objektumát tartalmazza. A next funkció lehetővé teszi a middleware futtatását vagy az aktuális middleware után lévő következő middleware meghívását a sorban. Az async kulcsszó jelzi hogy a függvény aszinkron módon működik ez lehetővé teszi az await kulcsszó használatát a funkcióban ami lehetővé teszi más műveletek végrehajtását a kódban úgy hogy a függvény futása megvárja amíg az await mávelet be nem fejeződik majd utána fut tovább. A függvény működését egy try catch szerkezetbe tesszük ami azt jelenti hogy a try részben részben lesznek az esetek amelyek közül hiba történhet és a catch részben pedig a lehetséges hibára adott választ. A try részben hozzuk létre a következőket. Van egy konstans típusú patientId nevű változó aminek az értéke a kérés paraméterének patientId-ja. Van egy konstans típusú patientCard nevű változó amiben egy await művelet eredményét tároljuk. A Patinet.findOne() metódussal pontosan egy rekordot tárolunk a patientCard változóban. A Patient betegek táblában és a hozzá kapcsolódó Treatments táblában és a hozzá kapcsolódó User felhasználó táblában azokat a rekordokat keressük ahol a Patient táblának az id értéke megegyezik a Treatments táblában lévő patient\_id val tehát hogy már van hozzáadva kezelés és ha mér van hozzáadva kezelés akkor a Treatment tábla kapcsolódik a User táblához amiben benne van a kezelő orvos adatai. Tehát a betegre kattintáskor ez a folyamat fog futni és a beteg kártyája fog megjelenni ami tartalmazni fogja az egy rekordnak az összekapcsolt táblában lévő adatait. Az include-azt írjuk le hogy a Patient tábla összekapcsolódik a Treatments táblával ami magába foglalja a User tábla adatai tehát ezek tábla összekapcsolások hogy azonosít alapján hozzá tudunk férni egy rekord egy mezőjének értékéhez ami egy másik táblában tárolódik. Megvizsgáljuk a patientCard változót és ha üres akkor visszaküld egy objektumot egy hiba üzenettel. Ha a patientCard változó nem üres tehát tehát van ilyen azonosítójú beteg akkor visszaküld egy sikeres státuszt és a patientCard változó értékét ami egy objektum és tartalmazza az összes mezője értékét azokat amik az összekapcsolt táblában vannak.

MIDDLEWARE

schemaSignUp

Ez egy validációs séma azt állítja be hogy a fullName mezőhöz hozzáadta a kötelező kitöltés szabályát a require() metódussal. Az email mezőhöz hozzáadta a kötelező kitöltés szabályát a required() metódussal és mivel az email cím hozzáadta az email-cím formátum validálását email() metódussal.A password mezőhöz hozzáadta a required() metódussal a kötelező kitöltés szabályát és azt hogy a jelszónak minimum 6 karaternek kell lennie a min() metódussal.

validationSignUp

A schemaSignup .validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re.

isUserExistsSignUp

Ez egy felhasználót keres meg email cím alapján azt vizsgálja meg hogy ezzel az email címmel regisztráltak e már ha igen akkor visszaaadja hogy igen regisztráltak ha nem akkor a hiba objektumát.

**schemaLogin**

Ez egy validációs séma azt állítja be hogy email mezőhöz hozzáadja a kötelező kitöltést ha ez nem történi meg akkor Email cím megadása kötelező jelenik meg és hozzádjamég azt hogy email formátumnak megfeleljen ha nem történik meg akkor kérlek érvényes email-címet adj megd zöveg jelenik meg. A password mezőhöz hozzáadja a kötelező kitöltést ha ez nem történik meg akkor jelszó megadása kötelező szöveggel válaszol. password mezőhöz beállítja a jelszó minimum hosszát hogy minimum 6-nak kell lennie mert ha nem akkor Kérlek jelszó minimum 6 karakter.

validateLogin

A loginSignUp .validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re. Ha van hiba akkor a catch ág teljesül ami visszaadja a hibát.

authenticateToken

Ez a kérés headernek az authorization fejlécéből van a token. Ez dekódolja a tokent és ha a dekódolás közben hiba van akkor a token érvénytelen. Ha nincs hiba akkor a token érvényes. Token dekódolása előtt a Bearer szót le kell törölni az nem tartozik magához a tokenhez.

**schemaUpdateProfile**

Ez egy validációs séma azt állítja be a fullName mezőre hogy kötelező legyen kitölteni ha nem történik meg akkor Kérlek írdd be a teljes neved üzenet jelenik meg. Az email mezőre beállítja hogy kötelező legyen kitölteni ha ez nem történik meg akkor Kérlek írd be az email-címedet jelenik meg. Majd beállítja hogy email-cím formátumban legyen megadva ha ez nem történik meg akkor Kérlek érvényes email címet adj meg.

validationUpdateProfile

A validationUpdateProfile .validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re. Ha van hiba akkor a catch ág teljesül ami visszaadja a hibát.

isUserExistsUpdate

Ez azt vizsgálja meg hogy a User táblában egy felhasználót keres amelyiknek az id-ja megegyezik a fiók tulajdonosának id-jával és az email-cím értéke ennek a rekordnak az az email-cím amire a felhasználó megakarja változtatni. Ez az eset ha megtörténik akkor az sikertelen lesz mert az email címet csak olyan email címre lehet frissíteni amelyik még nem létezik.

**schemaForgotPassword**

Ez egy validációs séma. A .required() azt állítja be hogy az email-címet kötelezően ki kell tölteni mert ha nem történik meg akkor az jelenik hogy Kérlek írd be a regisztrált email-címedet. .email() metódussal azt állítjuk be hogy az email-címnek meg kell felelnie az email-cím formátumnak ha ez nem történik meg akkor Kérlek írdd be az érvényes email-címedet.

validationForgotPassword

A validationForgotPassword .validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re. Ha van hiba akkor a catch ág teljesül ami visszaadja a hibát.

**schemaChangePassword**

Ez egy validációs séma. A new Password mezőre a required() metódussal beállíthjuk hogy kötelezően ki kell töleteni ha ez nem történik meg Kérlek írdd be az új jelszót jelenik meg. .min(6) metódussal beállítjuk hogy a jelszónak minimum 6 karakternek kell lennie. Ha ez nem történik meg akkor Kérlek a jelszó minimum 6 karakterből kell állnia. A confirmPassword mezőre a .required() metódussal beállítjuk hogy kötelezően ki kell töltenie ha ez nem történik meg akkor Kérlek erősítsd meg az új email-címedet. A .min metódussal azt állítjuk be hogy minimu 6 karakternek kell lennie ha ez nem történik meg Kérlek a jelszó minimum 6 karakternek kell lennie. A .oneOf() metódussal beállítjuk hogy a newPassword mezőnek és a confirmPassword mezőnek meg kell egyeznie ha nem történik meg akkor az Új jelszó és megerősítése nem egyezik.

validationChangePassword

validationChangePassword.validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re. Ha van hiba akkor a catch ág teljesül ami visszaadja a hibát.

schemaResetPassword

Ez egy validációs séma. A newPassword mezőre a required() metódussal beállítjuk hogy kötelezően ki kell töleteni ha ez nem történik meg Kérlek írj be egy új jelszót jelenik meg. .min(6) metódussal beállítjuk hogy a jelszónak minimum 6 karakternek kell lennie. Ha ez nem történik meg akkor Kérlek a jelszó minimum 6 karakterből kell állnia. A confirmPassword mezőre a .required() metódussal beállítjuk hogy kötelezően ki kell töltenie ha ez nem történik meg akkor Kérlek ismételd meg az új jelszavadat. A .min metódussal azt állítjuk be hogy minimum 6 karakternek kell lennie ha ez nem történik meg Kérlek a jelszó minimum 6 karakternek kell lennie. . A .oneOf() metódussal beállítjuk hogy a newPassword mezőnek és a confirmPassword mezőnek meg kell egyeznie ha nem történik meg akkor az Új jelszó és megerősítése nem egyezik. A token mezőre beállítjuk hogy kötelezően nem lehet üres ha üres marad akkor kiírja hogy a jelszó visszaállító token nem létezik.

validationresetPassword

validatiopnResetPassword .validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re. Ha van hiba akkor a catch ág teljesül ami visszaadja a hibát

isResetTokenValid

????????????

schemaPatientDelete

Ez egy validációs séma itt azt adjuk meg hogy törlésnél ha ki akarunk valamit választani törlésre akkor az id-nak kötelezően tartalmaznia kell azt az id-t.

validateDeletePatient

.validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re. Ha van hiba akkor a catch ág teljesül ami visszaadja a hibát

schemaAddDiagnosis

Ez egy validációs séma itt adjuk meg hogy diagnózis megadásakor kötelezően megadottnak kell lennie az id-nak nem lehet üre ha üres marad akkor a Beteg azonosítójának megadása kötelező jelneik meg. A diagnózis mezőre azt állítjuk be hogy kötelezően kitöltött legyen ha ez nem történik meg akkor a Diagnózis megadása kötelező jelenik meg.

validateAddDiagnosis

.validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re. Ha van hiba akkor a catch ág teljesül ami visszaadja a hibát

**schemaAddTreatment**

Ez egy validációs séma itt azt tudjuk megadni itt megadjuk hogy a description PatientId UserId dateOfTreatment mezőknek kötelezően kitöltöttnek kell lennie. Ha ez nem történik meg akkor a zárójelben megadott szöveg jelenik meg.

validateAddTreatment

.validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re. Ha van hiba akkor a catch ág teljesül ami visszaadja a hibát

**schemaUpdateTreatment**

Ez egy validációs séma itt azt tudjuk megadni a description PatientId UserId dateOfTreatment mezőknek kötelezően kitöltöttnek kell lennie. Ha ez nem történik meg akkor a zárójelben megadott szöveg jelenik meg

validateUpdateTreatment

.validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re. Ha van hiba akkor a catch ág teljesül ami visszaadja a hibát

**schemaAddPatient**

Ez egy validációs séma. Ami egy beteg hozzáadásához kell azt minegyiket kötelező megadni kivéve drug allergy,allergy, chronicIlness, constantMedication, diagnosis.

validateAddPatient

.validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re. Ha van hiba akkor a catch ág teljesül ami visszaadja a hibát

isPatientsexistCreate

Ez azt vizsgálja meg hogy a beteg amit hozzá akarunk adni az már létezik olyan email címmel. Ha létezik akkor hiba történt ha nem létezik akkor a frissítés sikeres.

**schemaUpdatePatient**

Ez egy validációs séma. Ami egy beteg hozzáadásához kell azt minegyiket kötelező megadni kivéve drug allergy,allergy, chronicIlness, constantMedication, diagnosis.

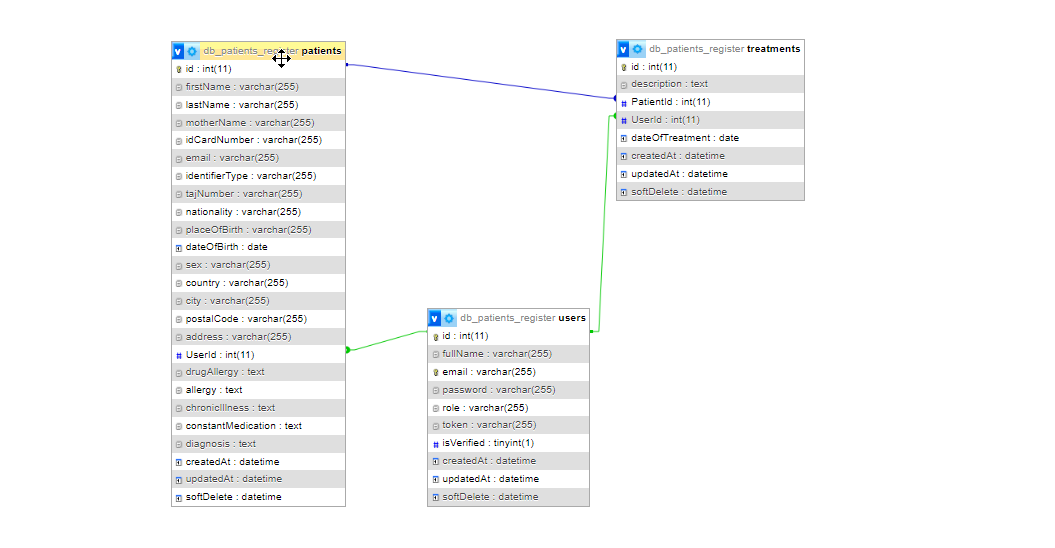
validateUpdatePatient

.validate() függvényét hívjuk meg ami ellenőrzi az űrlap által beküldött adatokat. A validate() függvénynek két paramétere van az első objektum tartalmazza az ellenőrizendő adatokat a második objektum az opcionális. A második objektum az jelzi az abortEarly: false hogy a validáció ne álljon meg az első hibánál hanem ellenőrizze az összes mezőt majd az összes hibát adja vissza. A then ág fut le ha minden validációs szabály teljesül ezután next() függvény hívódik meg ami tovább ugrik a következő middleware-re. Ha van hiba akkor a catch ág teljesül ami visszaadja a hibát

isPatientsExistsUpdate

Ez azt vizsgálja meg hogy a betge amire akarjuk frissíteni az már létezik olyan email címmel. Ha létezik akkor hiba történt ha nem létezik akkor a frissítés sikeres.

ADATBÁZIS Gabóczi Ádám



**Táblák közötti összekapcsolások**

USERS🡪PATIENTS

Egy **users táblában** lévő rekordot az id mező azonosít. A **patients táblában** lévő rekord idegen kulcsának az értéke megegyezik a users tábla id mezőjének az értékével ami azt jelenti hogy a users tábla egy rekordjához tartoznak azon rekord mezőinek az értékei amelyik rekordnak az idegen kulcsa megegyezik a users tábla id mezőjének az értékével. **users.id=patients.UserId.** A users tábla elsődleges kulcsa kapcsolódik a patients tábla UserId idegen kulcsához. Megkapjuk azoknak a rekordok mezőinek az értékeit

USERS🡪TREATMENTS

Egy **users táblában** lévő rekordot az id mező azonosít. A **treatments táblában** lévő rekord idegen kulcsának az értéke megegyezik a users tábla id mezőjének az értékével ami azt jelenti hogy a users tábla egy rekordjához tartoznak azon rekord mezőinek az értékei amelyik rekordnak az idegen kulcsa megegyezik a users tábla id mezőjének az értékével. **users.id=tretaments.UserId**

PATIENTS🡪TREATMENTS

Egy **patients** **táblában** lévő rekordot az id mező azonosít. A **treatments táblában** lévő rekord idegen kulcsának az értéke megegyezik a patients tábla id mezőjének az értékével ami azt jelenti hogy a patients tábla egy rekordjának mezőihez tartoznak azon rekord mezőinek az értékei amelyik rekordnak az idegen kulcsa megegyezik a treatments tábla id mezőjének az értékével. **patients.id=treatments.PatientId**

**patients tábla**

firstName, lastName: A beteg vezeték- és keresztnevét tárolják. Ezek STRING szöveg típusú mezők, amelyek nem lehetnek nullák. A vezetéknév és a keresztnév hosszának 2 és 20 karakter között kell lennie és csak az ABC betűit tartalmazhatják.

fullName: Ez egy virtuális mező, ami a vezetéknév és a keresztnév kombinációját adja vissza. Ez a mező nem lesz fizikailag jelen az adatbázisban, de használható lekérdezéseknél vagy a modell objektumokkal való munkavégzés során.

motherName: Egy beteg anyja nevét tartalmazza. Ennek a mezőnek a típusa string tehát szöveg és nem lehet az értéke üres.

idCardNumber: Egy beteg személyigazolványszámát tartalmazza. Ennek a mezőnek a típusa string tehát szöveg és nem lehet az értéke üres.

email: Egy beteg email-címét tartalmazza. Ennek a mezőnek a típusa string tehát szöveg és nem lehet az értéke üres. Van egy egy érvényesítés hogy megfelel e az email-cím formátumnak ha nem felel meg akkor adatbázisból való lekéréskor jelenik meg a hiba és az üzenet hogy Érvénytelen email formátum.

identifierType: Egy azonosító okmányának típusát tartalmazza. Ennek a mezőnek a típusa string tehát szöveg és nem lehet az értéke üres.

tajNumber: Egy beteg taj számát tartalmazza. Ennek a mezőnek a típusa string tehát szöveg és nem lehet az értéke üres.

nationality: Egy beteg nemzetiségét tartalmazza. Ennek a mezőnek a típusa string tehát szöveg és nem lehet az értéke üres.

placeOfBirth: Egy születési helyét tartalmazza. Ennek a mezőnek a típusa string tehát szöveg és nem lehet az értéke üres

dateOfBirth: Egy beteg születési idejét tartalmazza. Ennek a mezőnek a típusa dátum tehát szöveg és nem lehet az értéke üres.

sex: Egy beteg nemét tartalmazza. Ennek a mezőnek a típusa string tehát szöveg és nem lehet az értéke üres.

country: Egy beteg lakhelyét tartalmazza. Ennek a mezőnek a típusa string tehát szöveg és nem lehet az értéke üres.

city: Egy beteg lakhelyét várost tartalmazza. Ennek a mezőnek a típusa dátum tehát szöveg és nem lehet az értéke üres.

postalCode: Egy beteg irányítószámát tartalmazza. Ennek a mezőnek a típusa dátum tehát szöveg és nem lehet az értéke üres.

address: Egy beteg címét tartalmazza. Ennek a mezőnek a típusa string tehát szöveg és nem lehet az értéke üres.

UserId: Ez a beteg egyéni azonosítóját tartalmazza típusa string tehát szöveg. Nem lehet üres.

drugAllergy: Ez egy beteg gyógyszerallergiáját tartalmazza az adattípusa text tehát szöveg nem rögzített mint a string. Lehet az értéke nulla.

allergy: Ez egy betegallergiáját tartalmazza az adattípusa text tehát szöveg nem rögzített mint a string. Lehet az értéke nulla.

chronicIllness: Ez egy beteg betegségeit tartalmazza az adattípusa text tehát szöveg nem rögzített mint a string. Lehet az értéke nulla.

constantMedication: Ez egy beteg szedett gyógszereinek a nevét tartalmazza az adattípusa text tehát szöveg nem rögzített mint a string. Lehet az értéke nulla.

diagnosis: Ez egy beteg diagnózisát tartalmazza az adattípusa text tehát szöveg nem rögzített mint a

string. Lehet az értéke nulla.

**treatments tábla**

description: Egy kezelés leírását tartalmazza az értéke lehet üres. Az adattípusa text tehát nem rögzített hosszúságú.

PatientId: A beteg id-ját tartalmazza amelyikhez tartotik a kezelés. Az adattípusa Integer tehát szám. Az értéke nem lehet üres.

UserId: A felhasználó id-ját tartalmazza amelyikhez tartozik a kezelés. Az adattípusa Integer tehát szám. Az értéke nem lehet üres.

dateOfTreatment: A kezelés dátumát tartalmazza a típusa dátum és az értéke lehet üres.

**users tábla**

fullName: Ez a felhasználó teljes nevét tartalmazza. Az adattípusa string a string rögzített hosszúságú.

Az értéke nem lehet üres.

email: Ez a felhasználó email-címét tartalmazza. Az adattípusa string a string rögzített hosszúságú.

Az értéke nem lehet üres. Ennek egyedi értéknek kell lennie.

password: Ez tárolja a felhasználó hashelt jelszavát. Lehet az értéke üres.

role: Ez tárolja a felhasználó jogosultságát. . Az adattípusa string a string rögzített hosszúságú.

Az értéke nem lehet üres. Alapértelmezetten minden felhasználónak a jogosultsága doctor.

token: Ezt tárolja a felhasználó tokenjét. Stringben tárolja. Lehet az értéke üres.

isVerified: Ez a mező tárolja hogy egy felhasználó meg van e erősítve az alapértelmezett értéke 0

az adattípusa logikai érték.

Ajánlott Rendszerkövetelmény:

1. **Operációs Rendszer:** Windows 10 64 bit vagy annál újabb
2. **Processzor:** AMD A12-9720p RADEON R7, 12 COMPUTE CORES 4C+4G (4 CPUs) ~2.7GHZ
3. **RAM:**  8 GB RAM
4. **Videokártya:** AMD Radeon R7 Graphics
5. **Kijelző:** 1920X1080
6. Tesztelve a következő böngészőkben: Brave, Google Chrome, Microsoft Edge, Opera GX