TDK Dolgozat

Blood Notes

Véradással kapcsolatos információs rendszer és mobilalkalmazás a véradók számára

Szerzők:

Hegedüs Hunor

Szász Kata

XXXV. Országos Tudományos Diákköri Konferencia (OTDK)

Budapest, 2021. április 6-9.

Blood Notes

Véradással kapcsolatos információs rendszer és mobilalkalmazás a véradók számára

Szerzők:

Hegedüs Hunor

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Szász Kata

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:

dr. Simon Károly, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Fazakas Tibor, szoftverfejlesztő,

Codespring

Mihály Andor, szoftverfejlesztő,

Codespring

Nagy Katalin, szoftverfejlesztő,

Codespring

Kivonat

A Blood Notes projekt célja a véradás elősegítése és népszerűsítése, illetve egy hatékony és folyamatosan rendelkezésre álló kommunikációs csatorna kialakítsa a projekt két célcsoportja között. Az egyik célcsoportot a vért adni szándékozó személyek alkotják, akik számára egy mobiltelefonos alkalmazást biztosít a rendszer véradásaik dokumentálására és egyéb véradással kapcsolatos szolgáltatások használatára. A másik célcsoportba a véradó központok alkalmazottai tartoznak. Számukra egy olyan webes felületet biztosít a Blood Notes, amelyen kezelni tudják a mobiltelefonokon megjelenő informatív anyagokat, illetve értesítéseket küldhetnek az aktuális vérszükségletről.

A szoftverrendszernek három alkotóeleme van: egy iOS és Android platformon működő alkalmazás a donorok számára, egy webalkalmazás a véradó központok alkalmazottainak, illetve egy központi szerver, amely az adatokat kezeli és szolgáltatja.

A dolgozat szemlélteti a rendszer működését, architektúráját, részletezi a funkcionalitásokat, a felhasznált technológiákat és eszközöket.

Tartalomjegyzék

Bevezető

Bár óriási szükség van véradókra, a véradás még nem örvend elegendő népszerűségnek, és sajnos más régiókkal összehasonlítva ilyen szempontból Romániában még rosszabb a helyzet. Míg Ausztriában a lakosság 70%-a, Franciaországban pedig az 55%-a adományoz vért, addig Romániában mindössze a lakosság 1,7%-a szánja rá magát a véradásra.[1] A kórházak és a különböző szervezetek kampányokkal és rendezvényekkel próbálják buzdítani az embereket, felhívni a figyelmet az óriási hiányra, de nehezen érnek el eredményeket. Ehhez a népszerűsítési folyamathoz kíván hozzájárulni a jelen dolgozatban bemutatott Blood Notes projekt.

A Blood Notes alkalmazás kapcsolatot teremt a véradó központok és az emberek között, ezáltal lehetővé teszi a gyors információcserét, és egy felhasználóbarát, egyszerűen kezelhető felületet szolgáltat a donoroknak. Az alkalmazásnak három alkotóeleme van: egy webalkalmazás a véradó központok alkalmazottainak, egy iOS és Android mobiltelefonokon elérhető alkalmazás a vért adni szándékozó emberek számára, illetve egy központi szerver, amely kapcsolatot teremt az előző két komponens között és adatokkal szolgálja ki azokat.

Véradás előtt minden embernek ki kell töltenie a hivatalos véradói kérdőívet a véradó központban, amelyből kiderül, hogy alkalmas-e véradásra. A mobilalkalmazáson keresztül a felhasználó kitöltheti az említett kérdőívet, így időt takaríthat meg, és előre ellenőrizni tudja alkalmasságát. Továbbá, lehetőség van a korábbi véradások dokumentálására annak érdekében, hogy a felhasználó tisztában legyen azzal, hogy mikor adhat vért legközelebb. Erről az időpontról értesítést is kérhet, illetve arról is értesítést kaphat, ha valakinek sürgősen vérre van szüksége.

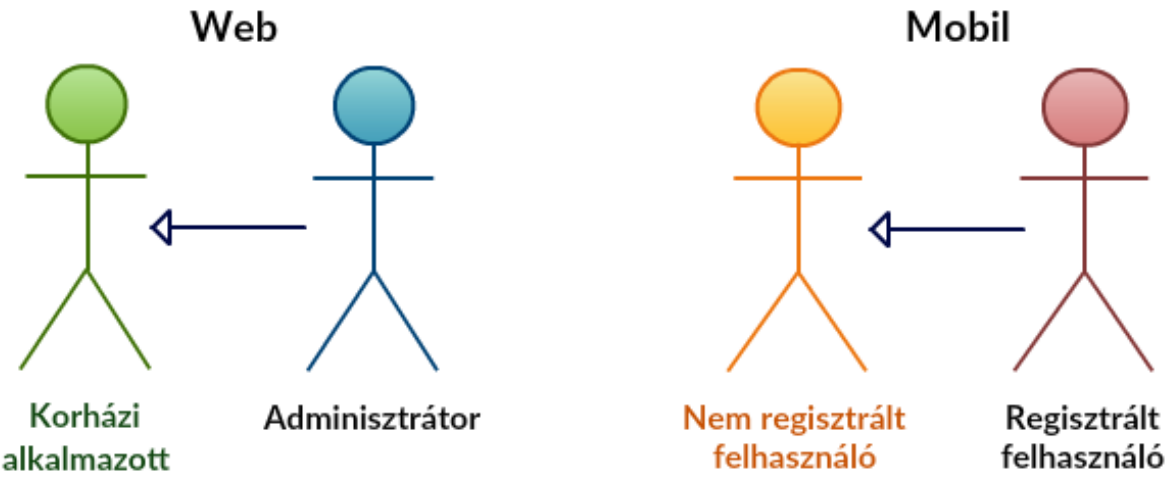
Az említett értesítést a vérszükségletről a véradó központból küldhetik ki a mobiltelefonokra a webalkalmazás használatával. Ha egy betegnek vérre van szüksége, akkor a kórházi alkalmazott értesítheti azokat a felhasználókat, akiknek a vére megfelelő lenne és az adott időszakban adományozhatnak vért. A véradó központok így kialakíthatnak maguk körül egy stabil "donor-tábort". Ezen kívül lehetőségük van arra is, hogy módosítsák a mobilkészülékeken megjelenő informatív adatokat, szerkeszthetik a véradáshoz kapcsolódó tudnivalókat és a véradó központok adatait, illetve megváltoztathatják a kérdőív kérdéseit is.

A dolgozat hátralevő része a következőképpen van strukturálva: az 1. fejezet áttekintés nyújt a szoftverrendszer felhasználói szerepköreiről és az ezekhez tartozó funkcionalitásokról. A 2. fejezet szemlélteti a rendszer architektúráját, lebontva szerver, mobil és web részekre. A 3. fejezetben szó lesz a felhasznált technológiákról és eszközökről. A 4. fejezet az alkalmazás működését mutatja be.

A projekt fejlesztése a U-Hub Mentorprogram keretén belül kezdődött 2018 nyarán, a Co-

[1] Forrás: https://ziarmedical.ro/2018/11/14/romanii-doneaza-sange, utolsó megtekintés dátuma: 2019-03-07

despring koordinálásával. A fejlesztés során segítségünkre voltak mentoraink: Nagy Katalin, Fazakas Tibor és Mihály Andor szoftverfejlesztők a Codespring részéről, valamint témavezető tanárunk Dr. Simon Károly egyetemi adjunktus. A csoportos projekt tantárgy keretén belül csatlakozott a csapathoz Móré Gerda Zsejke, Székely Anna, Balogh Szabolcs és Lukács Róbert-Sándor, akik egy egyetemi félév erejéig segítettek a fejlesztésben.



1. ábra. A web- és a mobilalkalmazás szerepkör-hierarchiája. Az adminisztrátoroknak rendelkezésükre áll minden funkcionalitás, amellyel a kórházi alkalmazottak rendelkeznek, és a regisztrált felhasználóknak rendelkezésükre áll minden funkcionalitás, amely a nem regisztrált felhasználók számára is elérhető.

1. Szerepkörök és funkcionalitások

A Blood Notes rendszernek két felhasználói felülete érhető el a felhasználók számára: a weboldal és a mobilalkalmazás.

A webes felületen a felhasználóknak lehetősége van a mobilon megjelenő adatok kezelésére, értesítések létrehozására és továbbítására. Ehhez a platformhoz két különböző szerepkörből férhetnek hozza: kórházi alkalmazott és adminisztrátor szerepkörökből.

A mobiltelefonos alkalmazás felhasználóinak olyan funkcionalitások állnak rendelkezésre, mint a véradó központokkal kapcsolatos információk listázása, a véradói kérdőív kitöltése, értesítésekre való feliratkozás, illetve a véradásaik naplózása. Ezen a felületen a nem regisztrált és regisztrált felhasználói szerepkörök léteznek.

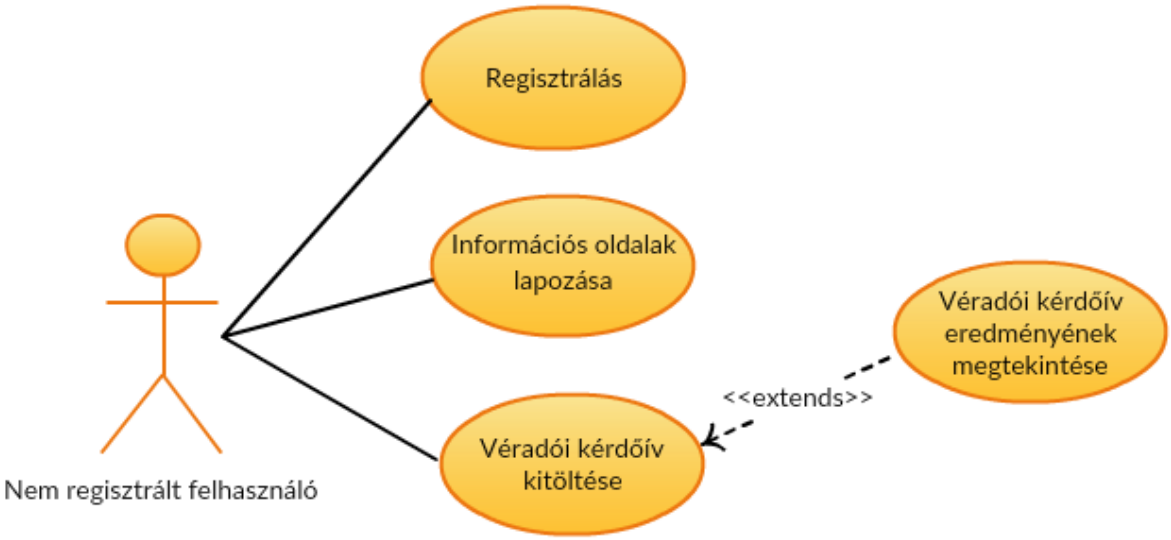
A kórházi alkalmazottak kevesebb joggal rendelkeznek, mint az adminisztrátorok, és a nem regisztrált felhasználóknak kevesebb joguk van, mint a regisztráltaknak. Ezt a hierarchiát mutatja az 1. ábra.

A dolgozat első fejezete részletesen tárgyalja az alkalmazás funkcionalitásait, szerepkörökre lebontva.

1.1. Nem regisztrált felhasználó

A nem regisztrált fehasználók számára a mobiltelefonos alkalmazásnak csak néhány funckionalitása elérhető, melyeket a 2. ábra szemléltet.

Az alkalmazás indításakor megjelenik néhány oldalnyi információ, melyeket ha a felhasz-



2. ábra. Mobiltelefonos szolgáltatások, melyek a nem regisztrált felhasználók számára elérhetőek.

nálók végiglapoznak, olvashatnak a véradással kapcsolatos általános tudnivalókról, a véradás folyamatáról és feltételeiről. A lapozás végén, vagy lapozás közben a "Főoldal" gombra kattintva, a felhasználók az alkalmazás központi képernyőjére érkeznek, ahonnan további lehetőségek nyílnak számukra.

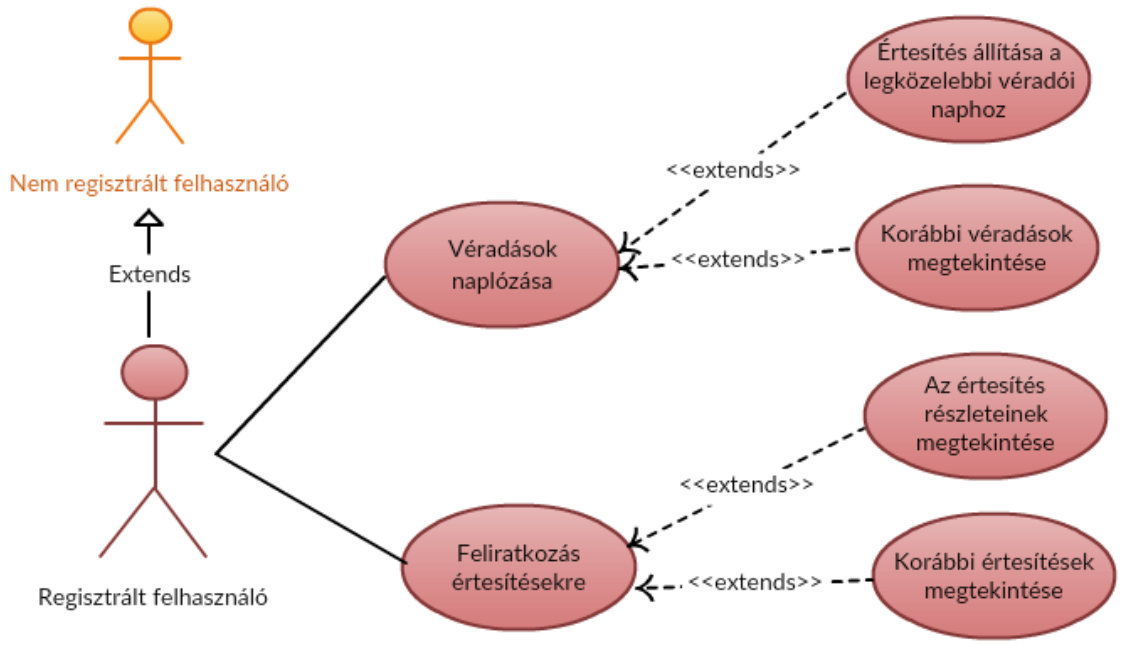
Megtekinthetik a véradó központok listáját, ahol a központok neve mellett megtalálják azok címét, illetve telefonszámát is. Lehetőségük van a véradói teszt kitöltésére. Itt igennel vagy nemmel válaszolhatnak olyan kérdésekre, amelyekből kiderül, hogy alkalmasak-e véradásra, vagy sem. A teszt kitöltése után a felhasználók megkapják az eredményt, és sikertelen teszt esetén azt is megtudják, hogy melyek azok az okok, amelyek miatt nem lehetnek véradók.

1.2. Regisztrált felhasználó

A regisztrált felhasználók rendelkeznek a nem regisztrált felhasználók jogaival, emellett további funkcionalitások is elérhetőek számukra. Ezeket a 3. ábra szemlélteti.

Bejelentkezés után feliratkozhatnak értesítésekre. Ha egy kórháznak bizonyos típusú vérre van szüksége, akkor küldhet egy üzenetet minden olyan regisztrált felhasználónak, aki feliratkozott az ilyen jellegű értesítésekre, illetve alkalmas véradásra és a vére megegyezik a szükséges típussal.

A magyarországi Országos Vérellátó Szolgálat adatai szerint, legkevesebb 56 napnak[1] kell eltelnie két véradás között. A regisztrált felhasználóknak lehetőségük van a véradásaik naplózására, és az alkalmazás jelzi, ha eltelt az említett 56 nap az utolsó véradás óta és a felhasználó újra adhat vért.



3. ábra. Mobiltelefonos szolgáltatások, melyek a regisztrált felhasználók számára elérhetőek.
r

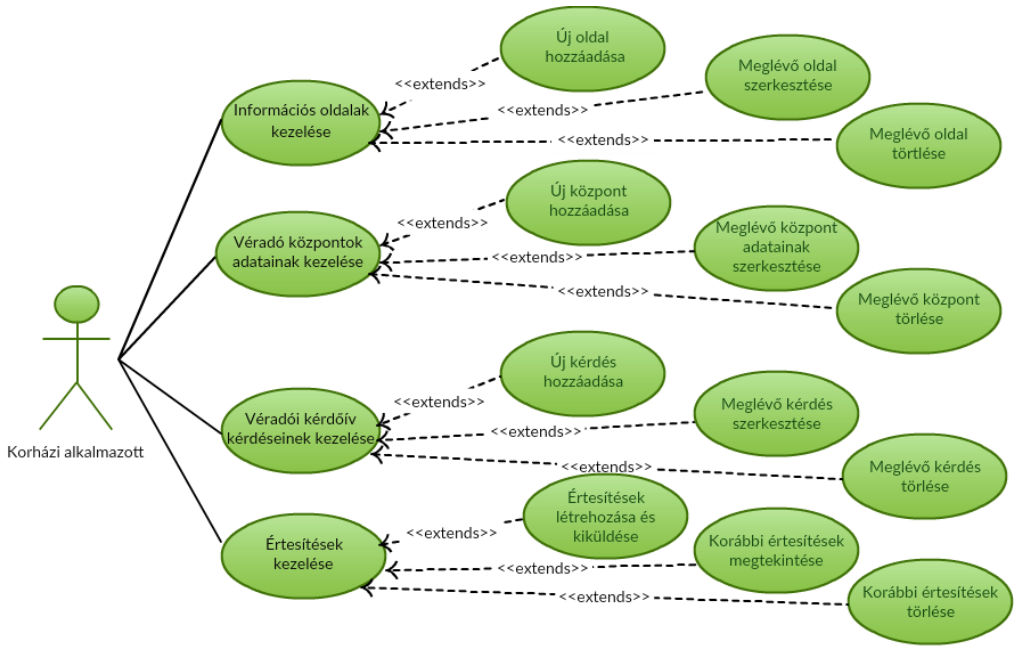
1.3. Kórházi alkalmazott

A kórházi alkalmazottak a kórházak vagy véradó központok által előre kijelölt személyek. Nekik a webes felülethez van hozzáférésük, és ők a felelősek azért, hogy az alkalmazás adatai naprakészek legyenek. A kórházi alkalmazottakhoz tartozó funkcionalitásokat a 4. ábra szemlélteti.

Kezelni tudják a véradó központok listáját, tudnak új központot hozzáadni, egy régit törölni vagy annak adatait módosítani. Kiegészíthetik vagy módosíthatják a véradói kérdőívet, illetve az információs oldalak tartalmának módosítására is lehetőségük van.

A véradással kapcsolatos információkat tartalmazó oldalak szerkesztésekor egy HTML szöveget kell módosítaniuk, amelyhez egy előnézet is tartozik. Ezzel tudják ellenőrizni, hogy a változtatások után hogyan fog kinézni az adott oldal a mobilalkalmazásban.

Lehetőségük van értesítést küldeni az alkalmazás regisztrált felhasználóinak, abban az esetben ha sürgősen vérre van szükség. Ebben az értesítésben informálják a leendő donorokat arról, hogy melyik véradó központ milyen típusú vért keres. Ezeket az értesítéseket utólag is elolvashatják, illetve törölhetik.



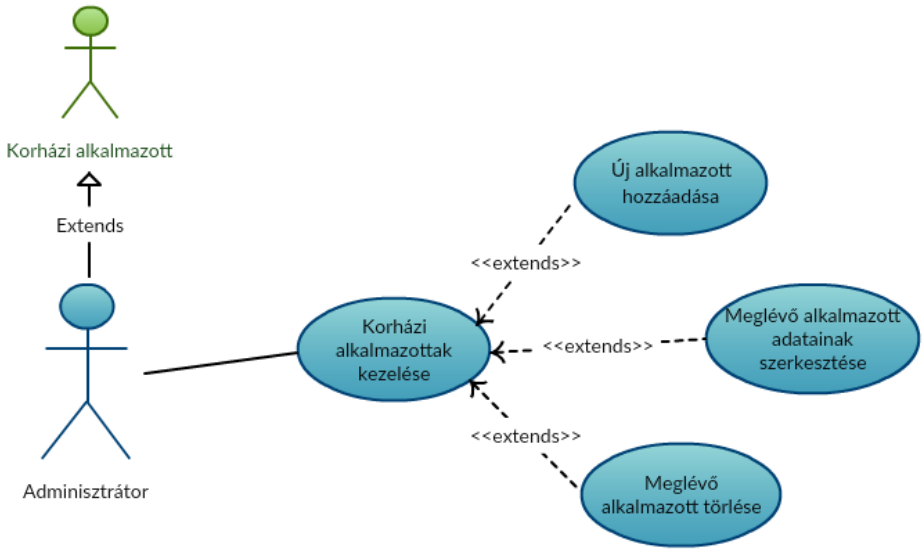
4. ábra. A webalkalmazás szolgáltatásai, melyek a kórházi alkalmazottak számára elérhetőek.

1.4. Adminisztrátor

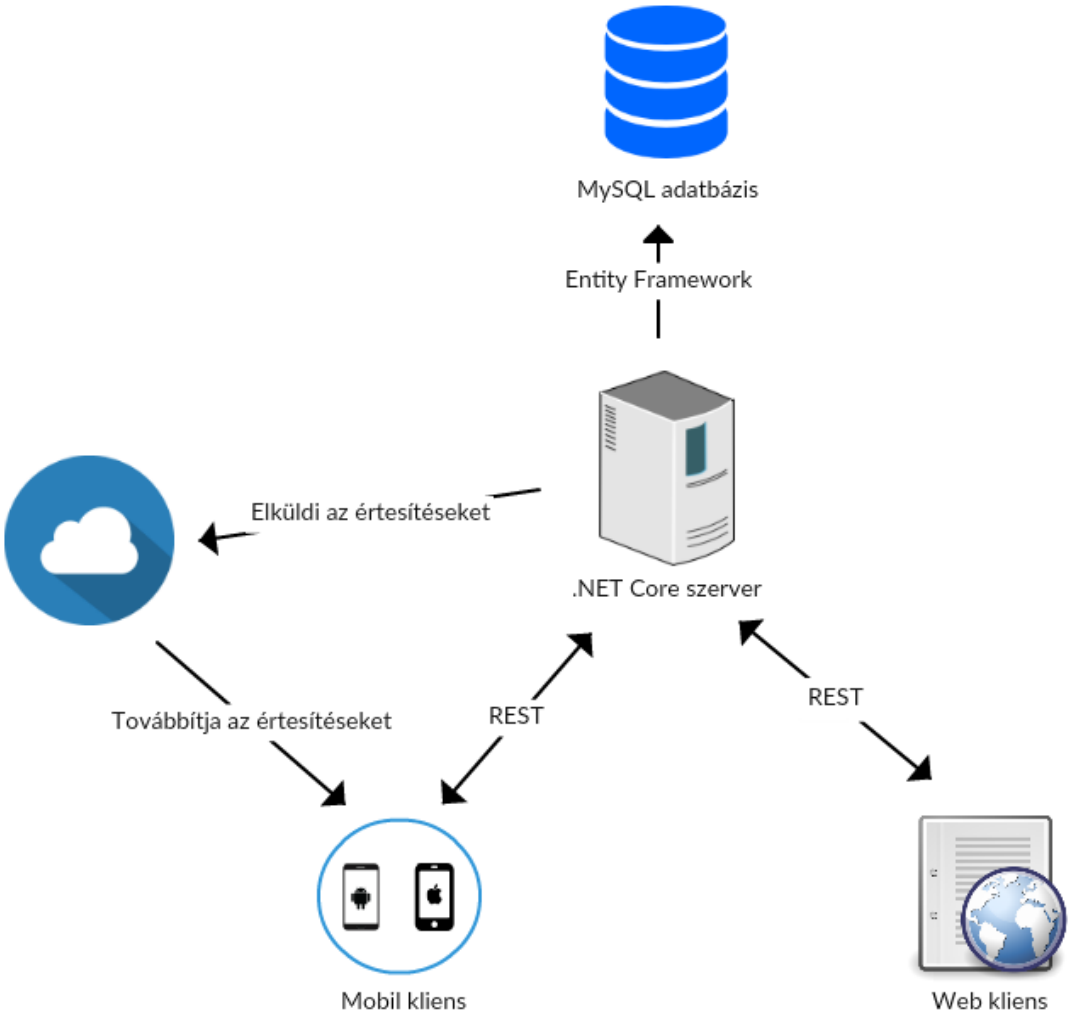
Az adminisztrátoroknak ugyancsak a webes felülethez van hozzáférésük és rendelkezésükre áll minden funkcionalitás, ami a kórházi alkalmazottak számára is elérhető.

Ezen kívül megtekinthetik a kórházi alkalmazottak listáját, hozzáadhatnak új alkalmazottat, vagy kitörölhetnek már létező alkalmazottat a rendszerből.

Az adminisztrátorhoz tartozó funkcionalitásokat a 5. ábra szemlélteti.



5. ábra. A webalkalmazás szolgáltatásai, melyek az adminisztrátorok számára elérhetőek.



6. ábra. A Blood Notes szoftverrendszer vázlatos architektúrája.

2. Architektúra

A szoftverrendszert három fő komponens alkotja: egy iOS és Android platformon működő *mobilalkalmazás* a donorok számára, egy React-ban megvalósított *webalkalmazás* a véradó központok alkalmazottainak, illetve egy központi .NET Core *szerver*, amely az adatokat kezeli és szolgáltatja. A rendszert továbbá kiegészíti egy MySQL adatbázis és a Firebase Cloud Messaging (FCM) platform, amely az értesítések továbbítását valósítja meg a szervertől a mobil kliensek felé.

A szerver és a mobil-, illetve webkliens közti kommunikációt a REST [2] konvenciónak megfelelő HTTP [3] kérések valósítják meg, míg a rendszerhez tartozó MySQL adatbázissal Entity Framework segítségével kommunikál a szerver. Ezeket a kapcsolatokat szemlélteti a 6. ábra.

Mindhárom fő komponensre jellemző a többrétegű architektúra. Ennek egyik előnye, hogy

[2]Representational State Transfer - Kliens és szerver közti kommunikációt megvalósító architektúra típus.
[3]Hypertext Transfer Protocol - Egy internetes információátviteli protokoll, amely kliens-szerver architektúrát alkalmaz.

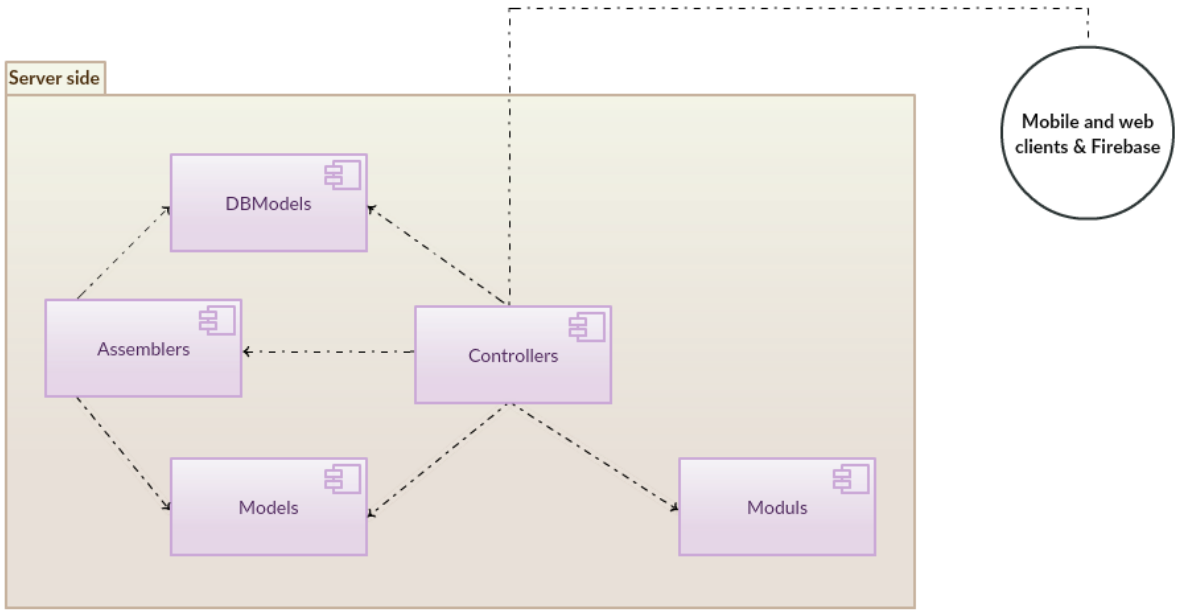
a kódbázis átlátható és érthető, a rendszer fejlesztése és karbantartása könnyebbé válik. Egy másik előnye, hogy minden réteg csak a szomszédos réteggel kommunikál, így a hibakezelés is hatékonyabb, mert módosítások esetén nem kell a teljes kódbázissal dolgozni, hanem elég csak egyes rétegeket változtatni. A kód hatékony szétválasztásának köszönhetően az egyes rétegek újrahasználhatóak.

2.1. Szerver

A szerver feladata, hogy kiszolgálja a rendszerhez tartozó klienseket a megfelelő adatokkal. A kéréseket a *Controllers* réteg fogadja és a kérések hatására a megfelelő Controller az Entity Framework segítégével kapcsolatba lép az adatbázissal és elvégzi a kért műveleteket.

A *DBModels* csomag osztályai írják le az adatbázisban levő adatokat, míg a *Models* csomagban levő osztályok a felhasználók számára látható adatokat tárolják. A két csomag osztályai közötti átalakítás *Assembler*-ek, illetve az AutoMapper[4] plugin segítségével van megvalósítva.

A Moduls csomagban található osztályok a szerver funkcionalitásainak a megvalósításában játszanak szerepet.



7. ábra. A rendszer szerverének komponens diagramja.

2.1.1. Adatmodell

A szoftverrendszer adatait egy MySQL relációs adatbázis tárolja. Négy különböző tábla létezik: *TestQuestion, Information, BloodDonationCenter* és *DbUser*.

[4]AutoMapper - Egy Nuget csomag, amely automatikus átalakítást végez két előre megadott osztály között.

A *TestQuestion* tábla tartalmazza a véradói kérdőív kérdéseit és az ezekhez tartozó egyéb tartalmakat. Olyan mezőkkel rendelkezik, mint *TestQuestionID, Question, Answer, Excluding, QuestionGender, Feedback, TextLanguage*. A *Question* mező tartalmazza az adott kérdés helyes válaszát, ami *Igen* vagy *Nem* lehet. Az *Excluding* mező tájékoztat arról, hogy nem megfelelő válasz esetén a kérdés kizáró jellegű-e, azaz eltiltja-e a válaszadót a véradástól. Ha igen, akkor a *Feedback* mező tartalmazza a magyarázatot erre. A *QuestionGender* megadja, hogy a kérdést nőknek vagy férfiaknak kell feltenni, illetve a *TextLanguage* jelöli a kérdés nyelvét.

Az *Information* táblában a mobilalkalmazás elindításakor megjelenő információs oldalak tartalma és az ezekhez tartozó adatok kerülnek tárolásra. Egy infomációs oldalnak eltárolódik a témája (*About*), maga a tartalma (*InformationData*) és a nyelve is (*TextLanguage*), akárcsak a kérdőív kérdéseinek esetében. Ezen kívül a *PageNr* mező tartalmazza, hogy az adott információ hányadik oldalon jelenik meg, illetve a *ContentType* megadja a fájl típusát, amely jelenleg *html* lehet, ugyanis az *InformationData* egy teljes HTML oldal kódját tartalmazza. Az egyedi kulcsot az *InformationID* mező tárolja.

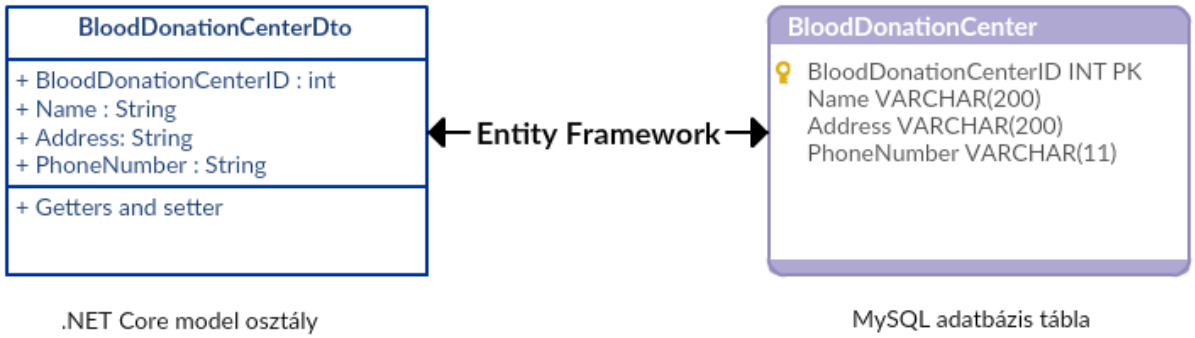
A véradó központokhoz tartozó információkat a *BloodDonationCenter* tábla tartalmazza, amely a következő mezőkkel rendelkezik: *BloodDonationCenterID, Name, Address, PhoneNumber*. A *Name* mezőben a központ teljes neve, az *Address* mezőben annak címe, és a *PhoneNumber* mezőben a telefonszáma található.

A szoftverrendszer felhasználóinak adatai a *DbUser* táblában tárolódnak. A *UserID* azonosító mellett, megtalálható a felhasználó felhasználóneve (*Username*), titkosított jelszava (*Password*), keresztneve (*FirstName*), vezetékneve (*LastName*), email címe (*Email*), szerepköre (*Role*) és a Firebase azonosítója (*FirebaseToken*). A szerepkör lehet *Admin, Operator* vagy *MobilUser*, az 1. fejezetben tárgyalt jogosultsági szinteknek megfelelően. Egy felhasználónak akkor van Firebase azonosítója, ha már feliratkozott értesítésekre.

2.1.2. Adathozzáférési réteg

Az adatbázis és a szerver közti kommunikáció megvalósítása az adathozzéférési réteg feladata. A BloodNotes alkalmazás esetében a központi szerver az adatokat egy MySQL relációs adatbázisból nyeri, mellyel Entity Framework segítségével kommunikál. A keretrendszer lehetővé teszi, hogy az adatbázisban levő táblákat osztályokként lehessen kezelni. A kódbázis egy modell osztálya az adatbázis egy táblájának felel meg, az osztály adattagjai a tábla mezőit jelölik. Ezt a megfeleltetést a 8. ábra szemlélteti a véradó központok példájával.

Amikor az alkalmazás először elindul, a szerver létrehozza az adatbázist a kódbázisban definiált modell osztályoknak megfelelően. Az elkészített táblákat feltötli adatokkal előre megírt SQL scriptek alapján, majd a továbbiakban az így elkészített adatbázis adatait módosítja.



8. ábra. Az Entity Framework segítségével az adatbázis táblái osztályokként kezelhetőek, a tábla mezői az osztály adattagjainak feleltethetőek meg. Az ábrán a véradó központok modellje látható.

Nincs szükség SQL parancsokra a CRUD [5] műveletek elvégzéséhez, a keretrendszer egyszerű metódusokat biztosít a négy alapvető adatmanipulációs művelet végrehajtására.

2.1.3. API

A mobilalkalmazás és webapplikáció által küldött HTTP kéréseket egy API szerver szolgálja ki, amely egy RESTful szolgáltatásként üzemel. A HTTP protokollon alapuló kommunikáció JSON formátumú üzenetekkel valósul meg.

A */blooddonationcenters, /informations* és */testquestions* elérési pontokra küldött GET kérések válaszai a véradó központok, a véradással kapcsolatos információk, illetve a véradói kérdőív kérdéseinek listái. Ugyanezekre az elérési pontokra küldött POST, PUT és DELETE kérésekkel szerkeszthetőek az adatbázis adatai. Ezekhez azonban már csak a webapplikációba bejelentkezett felhasználók férhetnek hozzá, amit a fejlécben csatolt bejelentkezési kulcs segítségével ellenőriz a szerver.

A bejelentkezési adatok ellenőrzése a */signin* címre küldött POST kérés hatására történik.

Az értesítéseket kezelő elérési pont a */notifications*. A mobilalkalmazásba bejelentkezett felhasználó erre e címre küldött POST kéréssel iratkozhat fel az értesítésekre. Az értesítések elküldése a */pushnotifications* címre küldött POST kéréssel történik.

Az adminisztrátor számára még elérhető az */operators* elérési pont, ahol a kórházi alkalmazottak adatait kezelheti.

2.1.4. Biztonság

Az 1. fejezetben bemutatott szerepkörökhöz különböző jogosultságok tartoznak. Ahhoz, hogy minden felhasználó pontosan a neki szánt tartalmakat érhesse el, ellenőrizni kell, hogy a kérés küldője be van-e jelentkezve vagy sem, illetve, hogy melyik szerepkörhöz tartozik.

[5] CRUD = Creat, Read, Update, Delete. Az adatbázis adatainak manipulációjára szolgáló négy alapművelet.

Mobil oldalon bejelentkezett és nem bejelentkezett felhasználókat lehet megkülönböztetni. A bejelentkezés első lépésében a mobilkliens elküldi a szervernek a mobilfelületen megadott felhasználónevet és jelszót. A szerver ellenőrzi, hogy a megadott adatpáros helyes-e, azaz létezik-e a rendszer adatbázisában. Ha igen, akkor sikeres a bejelentkezés, és a felhasználó számára generálódik egy JSON Web Token (JWT [19]). Ez a token tartalmazza a megadott felhasználónevet és a token lejárati idejét kulcs-érték párosok formájában, titkosítva. A szerver létrehoz egy objektumot, amelynek adattagjai a felhasználónév, a jelszó és a token lesznek, és ezt az objektumot visszaküldi a mobilkliensnek. Ezzel befejeződik a bejelentkezés, és ezután minden olyan kérés esetén, melyhez bejelentkezés szükséges, a *header*-ben elküldődik a token, így ellenőrizni lehet, hogy a kérés milyen jogosultságokkal rendelkező felhasználótól származik.

A webalkalmazás esetében a bejelentkezés mellett azt is ellenőrizni kell, hogy az adott felhasználó *adminisztrátor* vagy *kórházi alkalmazott* szerepkörbe tartozik. A bejelentkezés a mobilalkalmazás esetében is alkalmazott módszerrel történik, míg a szerepkört a *sessionStorage*-ban (részletesebben tárgyalva a 2.3.5. fejezetben) beállított adatok alapján lehet megállapítani. Ha a felhasználó nem rendelkezik a megfelelő jogokkal, akkor egy tájékoztatót kap arról, hogy számára nem elérhető szolgáltatást próbált használni, és a tartalom nem fog megjelenni.

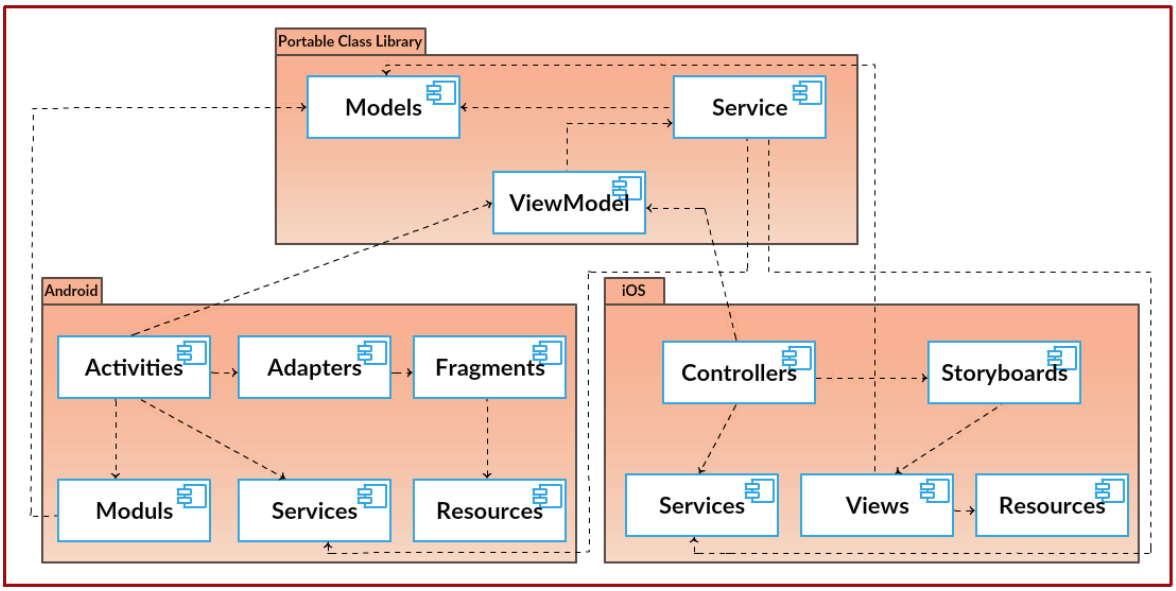
2.2. Mobil

A mobilalkalmazás feladata a kommunikáció a szerverrel, illetve egy Android és iOS felhasználói felület üzemeltetése.

A Xamarin Cross Platform technológia lehetőséget nyújt arra, hogy egyetlen C# [6] kódbázis keretén belül, amely három projektet tartalmaz, párhuzamosan lehessen fejleszteni az Android és iOS felületeket. Az üzleti logika nagy része egy közös, osztott könyvtárban található, ezt a kódbázist mindkét alkalmazás használja. A felhasználói felületet viszont sajátos módon kell implementálni a két különböző platformon, külön könyvtárba kerül az Android felület, illetve az iOS felület kódbázisa. Ezekbe a sajátos könyvtárakba kerülnek azok az üzleti logikához tartozó implementációk is, amelyek valamilyen okból nem lehetnek a közös könyvtár részei, hanem külön-külön kell implementálni ezeket a két platformon. Ezt a felépítést a 9. ábra mutatja be, és a Xamarin Cross Platform jellegzetességeit a 3.2. fejezet részletezi.

Az kód átláthatósága, tesztelhetősége és újrahasználhatósága érdekében a mobilalkalmazás implementációja az MVVM [20] tervezési minta elveit követi.

[6] C# - Objektum-orientált programozási nyelv.



9. ábra. A mobil oldal architektúráját három fő komponens alkotja. Egy Android csomag, melyben az Android készülékeken használható alkalmazás van implementálva Android-specifikus kódbázissal; egy iOS csomag, melyben megvalósul az iOS készülékeken működő alkalmazás iOS-specifikus kódbázissal; egy közös csomag, melyben olyan modellek és metódusok vannak implementálva, melyeket az Android és iOS csomag is felhasznál.

2.2.1. Kommunikáció a szerverrel

A mobilalkalmazás REST kéréseken keresztül kommunikál a központi szerverrel. A kérések az MVVM [20] tervezési mintának megfelelően a közös kódbázisban vannak implementálva. Ezeket az implementációkat használja az Android és az iOS kliens.

Az aszinkron kérések elküldése a *.NET System.Net.Http* [37] névtér *HttpClient* [35] osztályának segítségével történik, pontosabban az osztály *SendAsync metódusával*. A *SendAsync*-nak egyetlen kötelező paramétere van, egy *HttpRequestMessage* [36], ami az elérési útból és a kérés típusából (GET, POST, PUT, DELETE) tevődik össze. A válasz elkészítése a *Newtonsoft.Json*[21] framework segítségével valósul meg, míg az átalakított *JSON* formátumú adatok feldolgozása külön folytatódik a két kliensen.

2.2.2. Üzleti logika

Az alkalmazás üzleti logikájának implementálása a 9. ábrán bemutatott *Portable Class Library*-ben történik. A *Models* alkönyvtárban vannak megírva azok az osztályok, amelyek az adatbázisban tárolt adatok reprezentálását valósítják meg. A *Service* alkönyvtárban találhatóak azok az interface[7]-ek és osztályok, melyek az előbb említett modell osztályokat felhasználva implementálják a beérkező kérések kezeléséhez szükséges metódusokat. Ezeket a metódusokat a

[7]Interface - implementáció nélküli metódusok halmaza, melyet egy konkrét osztály megvalósíthat az által, hogy implementálja az összes metódusát.

ViewModel használja fel és szolgáltatja az Android és az iOS könyvtárak megfelelő komponenseinek.

2.2.3. Megjelenítési réteg

Ez a fejezet külön részletezi az Android és iOS megjelenítési rétegének megvalósítását. Míg az üzleti logika esetében a két platform közös kódbázist használ, addig a megjelenítési réteg implementálásához eltérő megoldásokat alkalmaznak.

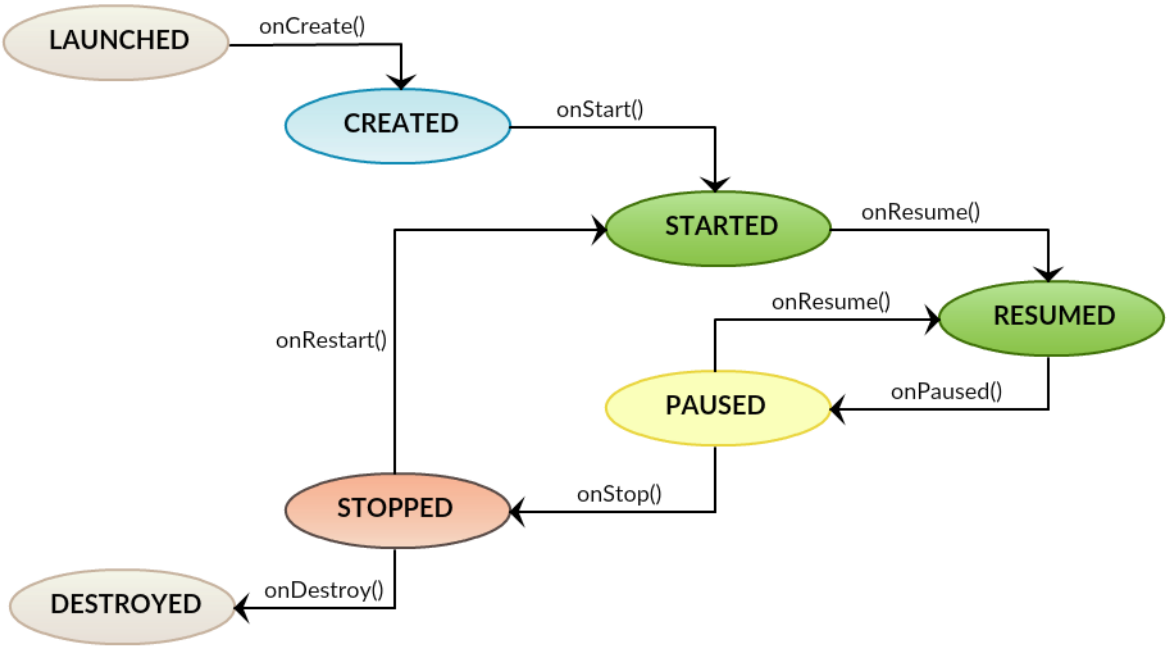
Az **Android** esetében az *Activity* osztályból [7] példányosított *activity*-k felelősek a felhasználói felület megjelenítéséért és a felhasználói interakciók kezeléséért.

Egy *activity* az alkalmazásnak egy képernyője, ahova a felhasználói felület kirajzolható. Van egy *MainActivity*, amely az alkalmazás belépési pontját jelenti, majd az alkalmazás működése során az *activity*-k egymást hívják meg, váltakozásukat a rendszer egy "activity verem" segítségével menedzseli. Amikor egy *activity* aktívvá válik, akkor az a verem tetejére kerül, elfedve az addigi legfelső elemet, amely ezáltal háttérbe kerül. Amikor a verem tetején levő *activity*-t megszünteti a rendszer, az alatta elhelyezkedő kerül ismét előtérbe.

Ezeket az állapotváltásokat a következő, előre meghatározott metódusok teszik lehetővé:

- **onCreate()** - akkor hívódik meg, amikor az *activity* először létrehozódik
- **onStart()** - akkor hívódik meg, amikor az *activity* által létrehozott nézet megjelenítődik a telefon képernyőjén
- **onResume()** - akkor hívódik meg, amikor felhasználói interakció történik
- **onPause()** - akkor hívódik meg, amikor a rendszer arra készül, hogy egy másik *activity*-t hozzon előtérbe
- **onStop()** - akkor hívódik meg, amikor a veremben az aktuális *activity* fölé egy másik kerül, így az aktuális háttérbe kerül
- **onRestart()** - akkor hívódik meg, amikor egy elfedett *activity* újra előtérbe kerül
- **onDestroy()** - a metódus, amelyik meghívódik mielőtt a rendszer törölné az *activity*-t a veremből

Az állapotváltásokat és a hozzájuk tartozó metódusokat szemlélteti a 10. ábra.



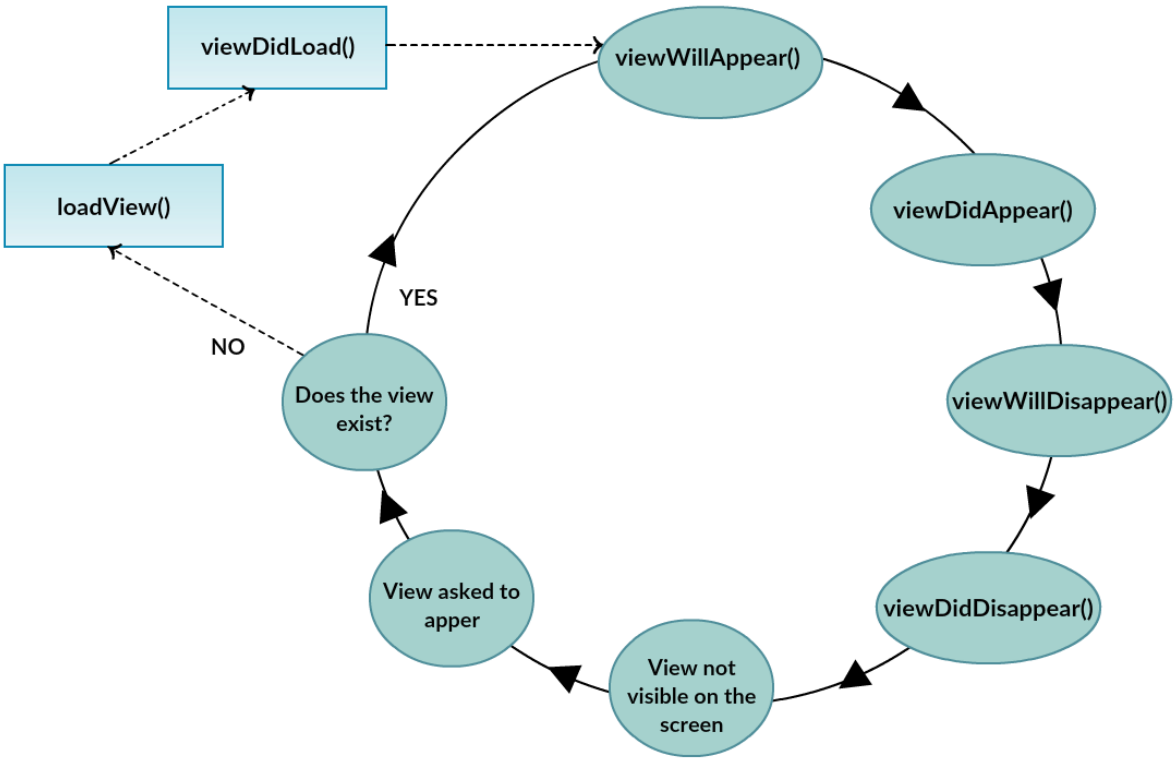
10. ábra. Az ábra egy *Activity* életciklusát szemlélteti, melynek több állapota van és az állapotváltásokat előre definiált metódusok valósítják meg.

Az **iOS** alkalmazás esetében a felhasználói felület kezelését a *ViewController*-ek végzik. Minden sajátos *ViewController*, mint például a *SignInViewController, CentersViewController, LangingPageViewController stb.* a *UIViewController* [6] egy példánya, melynek fő feladatai közé tartozik, hogy frissítse a nézetet, válaszoljon a felhasználó interakcióira, kezelje a nézet méreteit és tagoltságát, illetve kommunikáljon az alkalmazás más objektumaival. Egy *ViewController* egyetlen, a 2.2.4. fejezetben bemutatott, *View*-ért felelős.

A *Navigation Controller* felelős a *View*-k hierarchiájának kezeléséért. Ez a controller tartalmaz egy úgynevezett *navigation stack*-et, amely hasonló elveken alapszik, mint az Android esetében említett "activity verem".

Akárcsak az *Activity*-nek, a *ViewController*-nek is vannak életciklus-metódusai:

- **loadView()** - ez a metódus hívódik meg legelőször és csak egyszer, feladata, hogy létrehozza a nézetet
- **viewDidLoad()** - akkor hívódik meg, amikor a nézet elkészült és betöltődött a memóriába
- **viewWillAppear()** - figyelmezteti a controllert, hogy a nézet meg fog jelenni, illetve beállítja a nézet kereteit
- **viewDidAppear()** - akkor hívódik meg, amikor a nézet megjelent a mobiltelefon képernyőjén



11. ábra. A *UIViewController* életciklusaihoz tartozó metódusok rendszere.

- **viewWillDisappear()** - meghívódik mielőtt az aktuális nézet eltűnne a mobiltelefon képernyőjéről
- **viewDidDisappear()** - akkor hívódik meg, amikor a nézet eltűnt a képernyőről

A *UIViewController* életciklusát meghatározó metódusok rendszerét a 11. ábra szemlélteti.

2.2.4. Felhasználói felület

Az **Android** felhasználói felület [23] esetében a *View*-k hierarchikus elrendezése képezi a felhasználói felületet, egy *View* az alkalmazás egy oldalát valósítja meg. A *View*-kat kétféleképpen, deklaratívan és programatikusan, lehet létrehozni. A BloodNotes projekt esetében a deklaratív módszer dominál, de helyenként keveredik a programatikus megoldásokkal.

A *View*-k deklaratív létrehozását *AXML* [8] állományok használata teszi lehetővé. Ezekben az állományokban *tag*-ek határozzák meg azokat az elemeket, amelyek a *View* részét képezik. Egy *View*-t alkothat több *Fragment*, a *Fragmentek* pedig több elemből állnak. Az ilyen elemek (gombok, címkék, szövegdobozok stb.) elrendezéséért olyan előre megírt *layout*-ok felelősek, mint a *Linear Layout, Relative Layout, Table Layout*, és a dinamikus adatok az *AdapterView* irányításával kerülnek elrendezésre.

[8] AXML - Active Extensibel Markup Language. Android felhasználói felületek implementálásakor használt leíró nyelv.

Az *AXML* állományokban használt elemeknek és attribútumoknak osztályok és metódusok felelnek meg a kódban. Az állományok által definiált kinézeteket egy, a 2.2.3. fejezetben bemutatott, *Activity* könnyen létre tudja hozni, általában az *onCreate()* metódus törzsében a *setContentView()* segítségével.

Az alkalmazásban megjelenő médiafájlokat, illetve a stílusokat, lokalizációt és nemzetköziesítést megvalósító állományokat a *Resources* csomag tartalmazza.

A projekthez tartozó **iOS** felhasználói felületet [24] az *Interface Builder* által támogatott *.xib* és *.storyboard* állományok valósítják meg. Egy *.xib* állomány meghatároz egy *View*-t, a *.storyboard* állomány *View*-k halmazát tartalmazza, a köztük levő folytonos átmenetekkel.

A *View*-k kezeléséért a *ViewController* (lásd 2.2.3. fejezet) felelős. Minden *Controller*-hez tartozik egy fő *View*, a *root View*, amelyhez további *View*-k kapcsolódnak. Ezek a *View*-k egymásra halmozódnak egy veremben, melynek tetején a *root View* helyezkedik el, maga alá vonva a többi *View*-t. Ezt a hierarchiát *Content View Hiearchy*-nak nevezik.

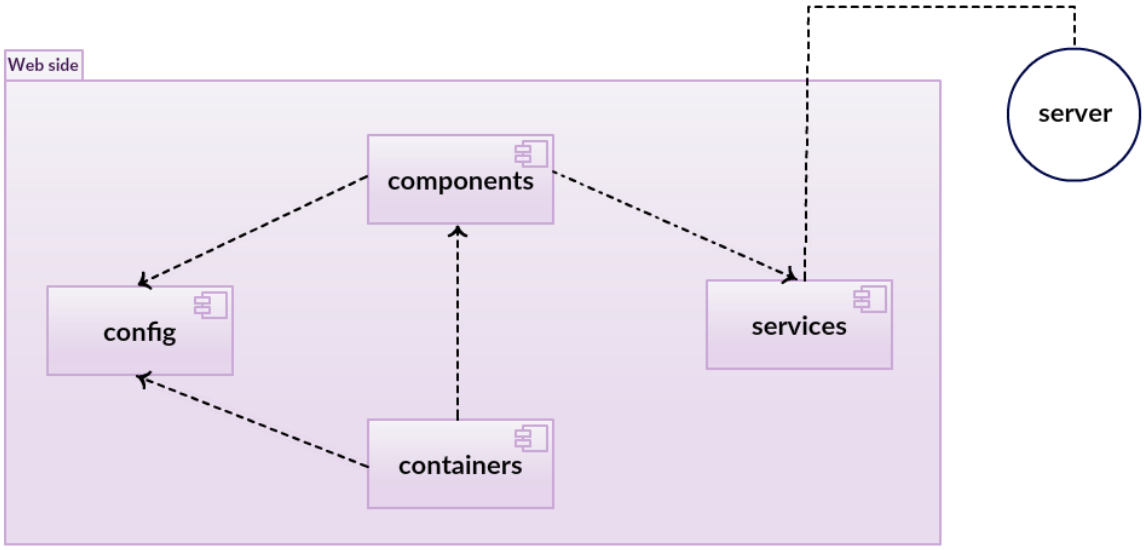
A *.xib* és *.storyboard* fájlok egy vizuális felületet nyújtanak a felhasználói felület megvalósítására. A nézet elemeit (gombokat, címkéket, szövegdobozokat stb.) egyszerűen be lehet húzni arra a mobiltelefon képernyőre, amit az állományok szimulálnak. Az elemek elhelyezkedését, kapcsolatát más elemekkel, stílusát és egyéb tulajdonságait is erről a felületről lehet menedzselni. Az átláhatóság és a modularitás érdekében a projekt keretén belül a *.xib* állományok használatára esett a választás

2.3. Web

A webalkalmazás kommunikál a szerverrel és lehetőséget ad a felhasználóknak a szervertől nyert adatok módosítására, valamint értesítések küldhetőek innen a mobil klienseknek.

A projektnek ez a része *React.js*-ben íródott, komponens alapú architektúrával rendelkezik. Egy komponens a webalkalmazás egy oldalát valósítja meg, és minden komponenst a *components* csomag tárol. Az oldalakra érkező kérések és a szerverrel való kommunikáció kezelését a *services* könyvtárban levő sajátos *service*-ek menedzselik. Az elérési címek és a weboldalak közti megfeleltetést az úgynevezett *routing* során végzi a rendszer, melyért a *config* csomag felelős. Egy *container* feladata, hogy megjelenítse a megfelelő elérési címhez tartozó komponenst.

A webalkalmazás architektúráját a 12. ábra szemlélteti.



12. ábra. A webalkalmazás architektúrája. Egy *komponens* a webalkalmazás egy oldalának felel meg, melyet a megfelelő elérési címmel egy *container* jelenít meg. A kérések kezelését és a szerverrel való kommunikációt a *service*-ek valósítják meg.

2.3.1. Kommunikáció a szerverrel

A webalkalmazás és a szerver kommunikációja REST kérések küldésén és fogadásán alapszik, a mobilalkalmazás és a szerver közti kommunikációhoz hasonlóan. A kérések kezeléséért a *service* csomagban található sajátos *service*-ek felelősek.

A kérések küldését és a válaszok fogadását a *Fetch API* [9] valósítja meg, amely egy globális *fetch()* metódust szolgáltat, valamint egy *Request* [10] és *Response* [11] objektumot. A metódus egyetlen kötelező paramétere az elérési cím, melyre a kérést küldi, illetve ahonnan a választ várja. Emellett megadható a kérés típusaként valamely HTTP metódus, az adat, melyet elküld, valamint testreszabható a *header* és a *credentials* opció. [9]

A válasz *JSON* objektum formájában érkezik, és a *Response.json()* [8] metódus elemzi.

2.3.2. Komponensek

A webalkalmazás esetében egy *komponens* az alkalmazás egy oldalának felel meg, például a véradó központokkal kapcsolatos információkat megjelenítő oldalt a *Components/Centers* komponens valósítja meg. Ezek megírása *.jsx* [25] állományokban történik, ez a kiterjesztés lehetőséget nyújt arra, hogy a *JavaScript* kód *HTML* szerű elemekkel egészítődjön ki (különböző attribútumokkal ellátott *tag*-ekkel). Ezeket az elemeket olyan külső könyvtárak biztosítják, mint a *reactstap, reactjs-popup, react-confirm-alert* stb.

Minden komponens a *React.Component* [38] osztályt terjeszti ki. Ez maga után vonja, hogy a komponenseknek implementálniuk kell a *render()* metódust, amely hozzáadja őket a megfe-

[9]Forrás: `https://github.github.io/fetch/#options`, utolsó megtekintés dátuma: 13.04.2019

lelő *DOM* [10]-hoz.

A *state* objektum tárolja a szervertől érkező adatokat egy komponensen belül. Valahányszor változik a *state* egy esemény hatására (pl. egy elemre való kattintás), a komponens automatikusan újra *renderelődik*, azaz frissül az a része, amely a megváltozott adatot megjeleníti.

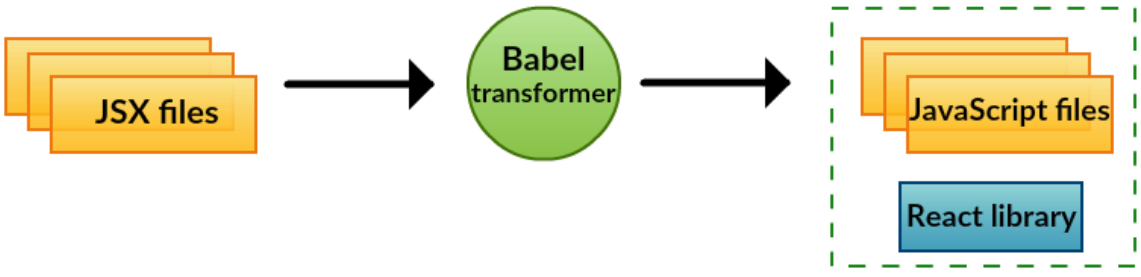
2.3.3. Routing

Routing során az *elérési útvonal*hoz hozzárendelődnek a megfelelő *komponens*ek. Ez azt jelenti, hogy egy állományban meg vannak határozva az *útvonal-komponens* párosok, és ezeket fogják felhasználni a *komponenseket* tartalmazó *containerek*.

Például, ha a véradó központokat szeretné elérni a felhasználó, akkor a */centers* elérési címet fogja használni, melynek hatására a *Centers* komponens jelenítődik meg a böngészőben.

2.3.4. Felhasználói felület

Ahogy már korábban szó volt róla, mindent, amit a böngésző megjelenít, komponensek valósítanak meg, melyek *.jsx* állományokban íródnak, így lehetőség van *HTML* szerű elemek használatára, amelyekkel könnyen lehet fejleszteni a felhasználói felületet. A böngészők nem tudják közvetlen módon értelmezni a *.jsx* állományokat, ezért szükség van arra, hogy ezek a böngészők számára értelmezhető *JavaScript* kóddá fordítódjanak. Ezt a fordítást a *Babel* [26] könyvtár *babel-plugin-transform-react-jsx* pluginja hajtja végre, ahogyan azt a 13. ábra szemlélteti. Az átalakítás során elkészült kódból a React létrehoz egy virtuális DOM-ot, melyből majd a böngésző felépíti a saját DOM-ját, és ennek alapján megjeleníti a tartalmat a felhasználó számára.



13. ábra. A böngésző számára közvetlen módon nem értelmezhető *JSX* állományok a *Babel* könyvtár *babel-plugin-transform-react-jsx* pluginja által alakíthatóak *JavaScript* álományokká.

A felhasználói felületet alkotó komponensek elemeinek stílusát *CSS* és *SCSS* állományok határozzák meg.

[10]DOM - Document Object Model. A weboldal betöltődésekor a böngésző elkészíti a weboldalt alkotó elemek gráfját.

2.3.5. Biztonság

A felhasználói adatok kezelését részletezte már a 2.1.4. fejezet, így a dolgozatnak ez a része csak a szerepkörök ellenőrzését tárgyalja.

A *window.sessionStorage* [12] attribútum lehetőséget kínál arra, hogy adatokat tároljunk a *session* lejártáig . Sikeres bejelentkezés után a felhasználó neve, tokenje és jogosultsága eltárolódik a *sessionStorage*-ban, mindaddig, amíg be nem záródik a böngésző. A *sessionStorage* tartalma megmarad az oldalak újratöltése/helyreállítása után is, de amint a böngésző bezáródik, a *sessionStorage* tartalma törlődik.

Amikor a *komponenseket* és *elérési címeket* tartalmazó *contanier* létrejön, ellenőrzi a *sessionStorage* tartalmát és csak azokhoz az *elérési címekhez* tartozó *komponenseket* jeleníti meg, melyekhez joga van a felhasználónak. Így például egy *kórházi alkalmazott* számára nem fog megjelenni a menüpontok között az *Operators* fül, mivel egy kórházi alkalmazott nem módosíthatja más kórházi alkalmazottak adatait.

3. Technológiák és eszközök

Ebben a fejezetben először a fejlesztés során felhasznált technológiák kerülnek bemutatásra, majd azokról az eszközökről lesz szó, amelyek segítették a hatékony és gyors fejlesztést.

3.1. Szerveroldali technológiák

.NET Core [30] A .NET Core a Microsoft nyílt forráskódú keretrendszere. Platformfüggetlensége nagy előnyt jelent, több operációs rendszeren is futtatható (Windows, Linux és macOS). A fejlesztés C# nyelvben történt, és a Microsoft Nuget [31] csomagjainak a felhasználásával vált egyszerűbbé. A .NET Core segítségével sikerült egy megbízható, stabil API szervert létrehozni, amely aszinkron hívásokkal szolgálja ki a mobilalkalmazás és webalkalmazás kéréseit.

MySQL [4] A MySQL az egyik legnépszerűbb relációs adatbázis-kezelő rendszer. A .NET Core-hoz hasonlóan szintén több platformon is használható.

Entity Framework [28] Az Entity Framework egy objektum-relációs leképező rendszer a .NET keretrendszerhez. A segítségével egyszerűen, SQL lekérdezések megírása nélkül hozzá lehet férni az adatbázisban tárolt adatokhoz. A beépített lekérdezések használatához a System.Linq (Language-Integrated Query) névtér használható. A rendszer két mintát kínál az adatbázis kezeléséhez, egy Code-First és egy DataBase-First lehetőséget. A szerver a Code-First változatot használja, ami azt jelenti, hogy a szerver indulásakor, ha még nem létezik az adatbázis, akkor először létrehozza azt, majd utána feltölti adatokkal.

3.2. Mobil technológiák

Xamarin [34] A Xamarin a Microsoft tulajdonában álló nyílt forráskódú crossplatform keretrendszer, amely C# nyelven teszi lehetővé a fejlesztést iOS, Android és Windows készülékekre. Segítségével natív alkalmazás készíthető különböző eszközökre, úgy, hogy használjuk azok saját grafikus elemeit és az SDK-k minden funkcióját. Ezt úgy teszi lehetővé, hogy az üzleti logikához szükséges kódot közösen használja minden platform, csak a felhasználói felületet kell külön elkészíteni. A háttérben a Xamarin a C# kódot natív kóddá alakítja, így az alkalmazás a felhasználói élmény szempontjából is olyanná válik, mintha teljesen natív módon íródott volna. A felhasználói felület egyedi elemei az iOS és az Android projektekben vannak létrehozva, a közös részek pedig egy Portable Class Library nevű projektben. A Xamarin nagyban megkönnyítette a fejlesztést, hiszen a kód jelentős részét elég volt egyszer megírni, és utána az változtatások nélkül használható volt mindkét platformon.

Model-View-ViewModel (MVVM) [20] Az MVVM egy architekturális tervezési minta, amely elválasztja az alkalmazás üzleti logikáját a felhasználói felület elemeitől. A minta hasz-

14. ábra. Az ábra az MVVM tervezési mintát mutatja be, ami elválasztja az alkalmazás grafikus felületét a közös üzleti logikától

nálatával a kód három jól elkülönített részre lesz felosztva. A View a grafikus felület, a Model az adatok pillanatnyi állapota, a ViewModel pedig az adatok átalakításáért és könnyű megjelenítéséért felelős. Segítségével az alkalmazás üzleti logikáját teljesen szét tudjuk választani a felhasználói felülettől, így a Xamarin sajátosságainak megfelelően az üzleti logika részt fel tudjuk használni mind az Android, mind az iOS platformon. A ViewModel adatkötéssel (Data Binding) kapcsolódik a View-hoz. Emiatt egy állandó kapcsolat áll fenn a felhasználói felület és a mögötte lévő adatszerkezetek között.

A Blood Notes alkalmazásban az **MVVM Light**[29] eszközkészlet használatával valósul meg az MVVM tervezési minta. Az eszközkészlet gördülékenyebbé teszi az MVVM alkalmazások fejlesztését, illetve egyszerűbbé válik a felület elválasztása az üzleti logikától.

Firebase [13] A Firebase egy Google által fejlesztett szolgáltatáscsomag, amely hasznos funkciókat biztosít az alkalmazások számára (bejelentkezés, bizonyos tartalmak kedvencek közé helyezése, értesítések küldése/fogadása stb.). A Blood Notes rendszer a Cloud Messaging, azaz az értesítésekkel kapcsolatos szolgáltatást használja. A Firebase Cloud Messaging segítségével a szerver egy HTTP POST kéréssel képes értesítéseket küldeni az előre feliratkozott felhasználóknak. A mobilalkalmazás felhasználói a beállítások menüpontban iratkozhatnak fel az értesítésekre. Az értesítések elküldése a webalkalmazásból egy szerverre küldött HTTP POST kéréssel történik. A szerver ugyancsak egy kérést küld a Firebase szerverére, ami kiküldi a megfelelő értesítéseket a klienseknek.

3.3. Web technológiák

React.js [16] A React.js egy felhasználói felületek készítésére használt JavaScript könyvtár. A Facebook által fejlesztett könyvtár újrahasznosítható komponensek használatán alapszik, amelyek HTML és JavaScript használatával íródnak. A JavaScriptXML (JSX) szintaxis lehetővé teszi a HTML beékelését a JavaScript kódba.

A React.js alapját a Virtual DOM jelenti, amely egy új réteg a kód és a DOM között. Segítségével az oldal újratöltése nélkül lehet megjeleníteni nagyméretű oldalakat, ami a komponensek állapotainak figyelésével valósul meg. Ahogy egy komponens állapota változik, az oldalon is megtörténik a változás, de nem az egész oldal változik, hanem csak az adott komponens, így

hatékonyabb a frissítés.

A fejlesztést az npm (Node Package Manager) csomagkezelő tette egyszerűbbé, amely segít az előre megírt modulok felhasználásában.

Node Package Manager (npm) [22] Az npm egy nyílt forráskódú csomagkezelő rendszer, amely node programok telepítésére és karbantartására használható. Használata egyszerűbb, strukturáltabb és gyorsabb fejlesztést biztosít. A webalkalmazás az npm 'create react app' parancsa alapján létrehozott alap projektből indult, és különböző npm csomagokat felhasználva bővült.

Bootstrap [27] A Bootstrap egy nyílt forráskódú CSS keretrendszer, amely segít reszponzív, azaz bármilyen eszközön optimális módon megjeleníthető felhasználói felületek létrehozásában. A webalkalmazás a 'bootstrap', 'reactstrap' és 'react-bootstrap' npm csomagok előre megírt komponenseiből áll.

3.4. Eszközök

Visual Studio [32] A Visual Studio a Microsoft hivatalos fejlesztői környezete, amely egyaránt megfelelt a szerver, illetve a mobilalkalmazás fejlesztéséhez és lokális futtatásához. Támogatja a .NET Core és a Xamarin keretrendszerek használatát is.

Egyszerű hozzáférést biztosít a Microsoft Nuget [31] csomagok használatához, mind a szerver, mind a mobilalkalmazás esetében. Emellett rendelkezik megfelelő Android emulátorral és egy *Mac* operációs rendszerű számítógéphez csatlakozva használja annak *Remoted iOS Simulator*-át is, amelyeken tesztelhető az alkalmazás.

Visual Studio Code [33] A Visual Studio Code szintén a Microsoft fejlesztői környezete, amely segítségül szolgált a webalkalmazás fejlesztésekor. Rendelkezik beépített parancssorral, ahonnan elindítható az alkalmazás és kezelhetőek az npm csomagok.

Git [3] A Git az egyik legnépszerűbb osztott verziókövető rendszer. Segítséget nyújt a gördülékeny közös fejlesztésben és a különböző verziók kezelésében. A Git repositoryban vannak tárolva az egyes verziók. A projekt esetében három repository van, egy a szervernek, egy a mobilalkalmazásnak, illetve egy a webalkalmazásnak.

Gitlab [17] A GitLab egy Gitet használó repository-menedzsment rendszer. Segít az alkalmazás lépésenkénti felépítésében, az egyes verziók nyomonkövetésében és ellenőrzésében.

A GitLabon történt emellett a rendszerrel kapcsolatos dokumentációk elkészítése és tárolása, illetve a kitűzött feladatok kezelése.

Gitlab Continous Integration (Gitlab CI) [18] A GitLab CI a GitLab beépített rendszere, amely biztosítja az újonnan elkészült részek helyes integrációját a már meglévő rendszerbe. Amikor új kódrészlet került a repositoryba, a GitLab mindig megpróbálta lefuttatni az aktuális

projektet és jelzett, ha hiba lépett fel.

GitKraken [2] A GitKraken egy grafikus felhasználói felület a Git verziókövető rendszerhez. Segítségével a fejlesztők egyszerűen kezelhetik a rendszer különböző verzióit, összevonhatják azokat, illetve egy áttekintést kapnak arról, hogy hol tartanak az egyes feladatok elvégzésével.

Docker [14] A Docker egy konténer-alapú virtualizációs eszköz, amely lehetőséget ad arra, hogy egy rendszert elszigetelve egy virtuális gépen lehessen futtatni. Használatával a szerver bárhol, bármilyen környezetben futtatható, egyszerűen parancssorból. A Docker container-eket biztosít, amelyekben az alkalmazáshoz szükséges környezet van jelen. Előnye más hasonló rendszerekkel szemben, hogy könnyen használható és a container-ek kisebb mérete miatt lényegesen gyorsabb.

Docker Compose [15] A Docker Compose egy eszköz, amellyel több container-ből álló Docker alkalmazások készíthetőek. Nagyban elősegíti a rendszer skálázhatóságát és modularitását. Docker Compose-al valósul meg, hogy a szerver egy külön docker container-ben fut, míg a MySQL adatbázis egy másikban, és kapcsolódni tudnak egymáshoz. Ezután egyetlen paranccsal futtatható a rendszer, bármilyen környezetben.

MySQL Workbench [5] A MySQL Workbench egy grafikus felhasználói felület a MySQL egyszerűbb kezeléséhez.

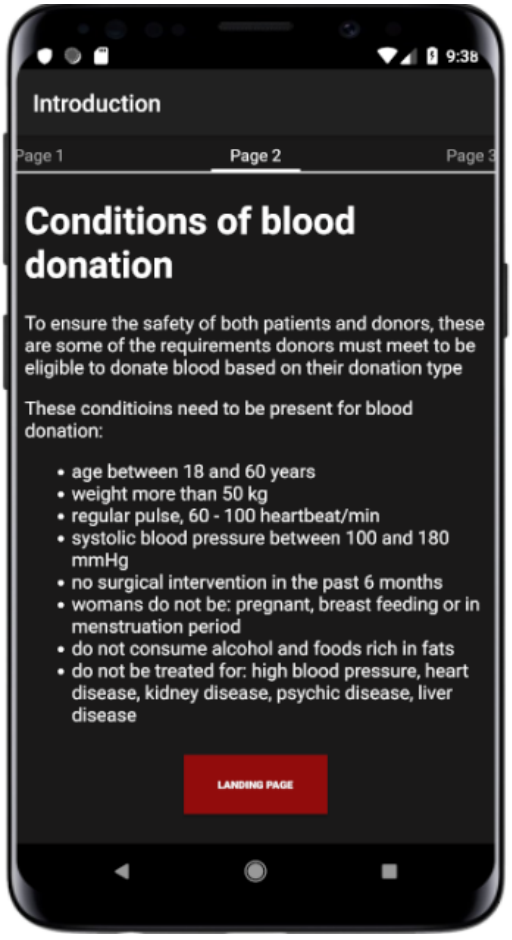
4. A Blood Notes alkalmazás működése

Ez a fejezet rövid leírásokkal és képernyőfotókkal szemlélteti az alkalmazás fontosabb funkcióinak használatát.

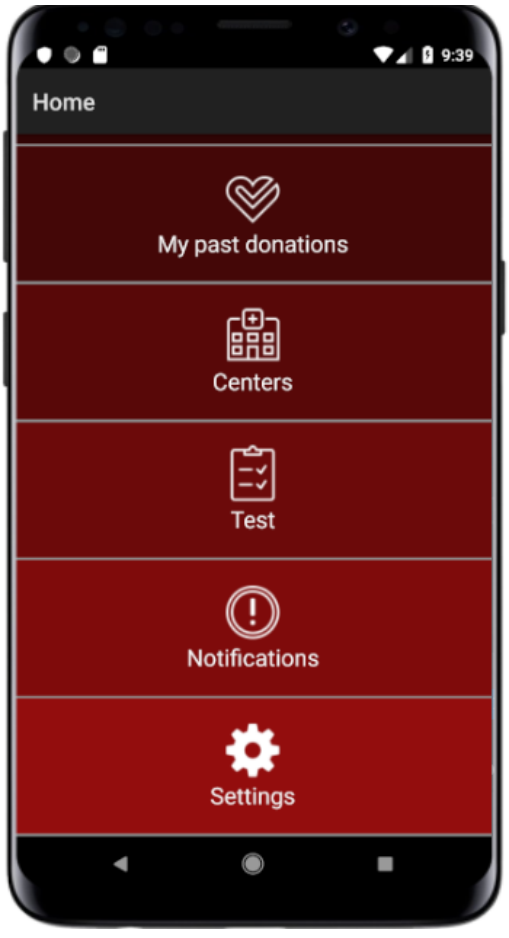
A mobilalkalmazás használatának lépései az Android felületen kerülnek bemutatásra.

Az mobilalkalmazás indításakor néhány oldalnyi bevezető információ fogadja a felhasználókat, amelyeket végiglapozva olvashatnak a véradással kapcsolatos tényekről és szabályokról. Egy ilyen információs oldalt szemléltet a 15. ábra.

Végiglapozva vagy átugorva a bevezető oldalakat, a felhasználó eljut az alkalmazás központi képernyőjére (16. ábra), ahonnan lehetősége van elérni az alkalmazás további funkcionalitásait.



15. ábra. Az alkalmazás indításakor megjelenő informatív oldalsorozat egy oldala.



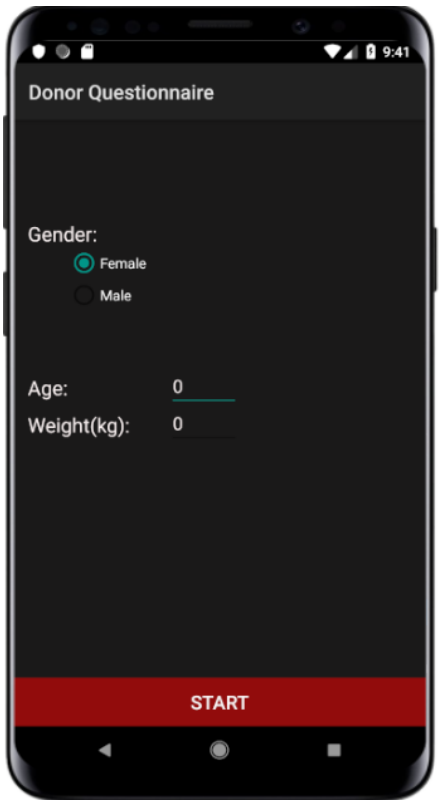
16. ábra. Az alkalmazás központi képernyője.

Egy ilyen lehetőség, hogy megtekintheti a véradó központok listáját (17. ábra), ahol megtalálhatja azok címét és telefonszámát is. Az alkamazásnak ez a funkciója megkönnyíti a felhasználó számára a központokkal való kapcsolatteremtést.

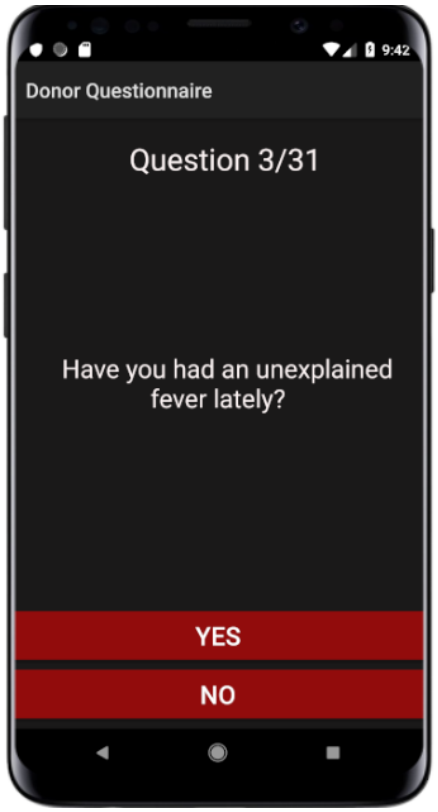
17. ábra. A véradó központok és a hozzájuk tartozó információk listája.

Egy másik funkcionalitás, hogy a felhasználó kitöltheti a véradói kérdőívet. A kérdőív tartalmazza azokat a hivatalos kérdéseket, melyekre a kórházban kellene válaszolni véradás előtt, így a felhasználó időt takaríthat meg azzal, hogy telefonján keresztül tölti ki a kérdőívet. A válaszok alapján megkapja az értékelést, azaz megtudja, hogy adhat-e vért vagy sem. Abban az esetben, ha nem lehet véradó, egy listában megtalálja azokat az okokat, amelyek miatt nem adományozhat vért.

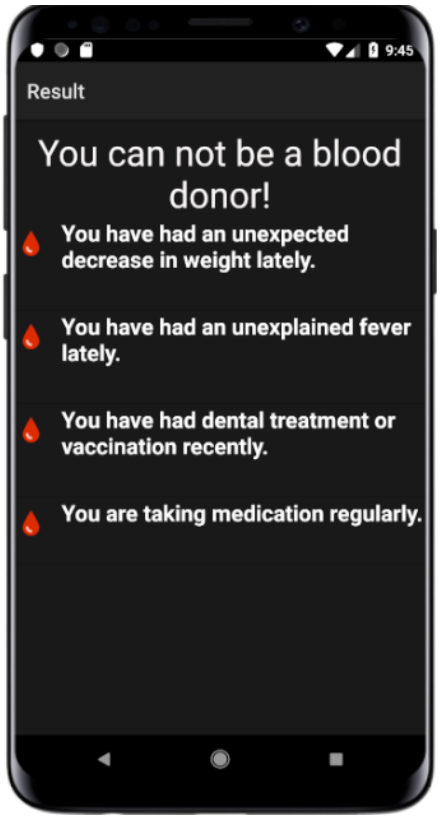
A kérdőív kitöltésének lépéseit a 18., 19., 20. ábrák szemléltetik.



18. ábra. A kérdőív első oldala, melyen a felhasználónak meg kell adnia a nemét, életkorát és testsúlyát.



19. ábra. A kérdőív 31 kérdésének egyike.



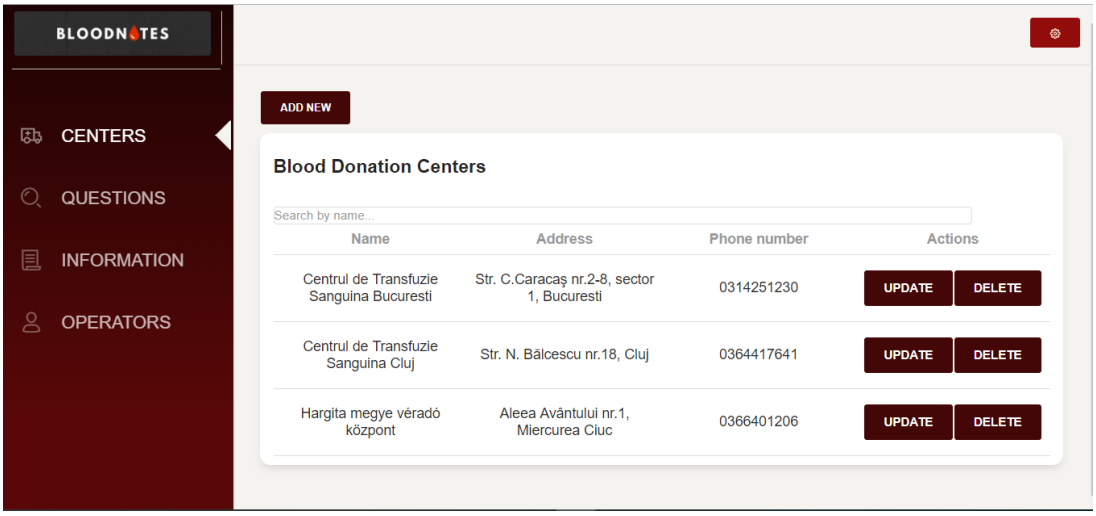
20. ábra. A kérdőív eredményének kimutatása.

A webalkalmazást a kórházi alkalmazottak és az adminisztrátorok használják. Az adminisztrátoroknak eggyel több funkcionalitás áll rendelkezésükre, mint a kórházi alkalmazottaknak: módosítani tudják az alkalmazottak adatait, illetve új alkalmazottakat tudnak hozzáadni a rendszerhez.

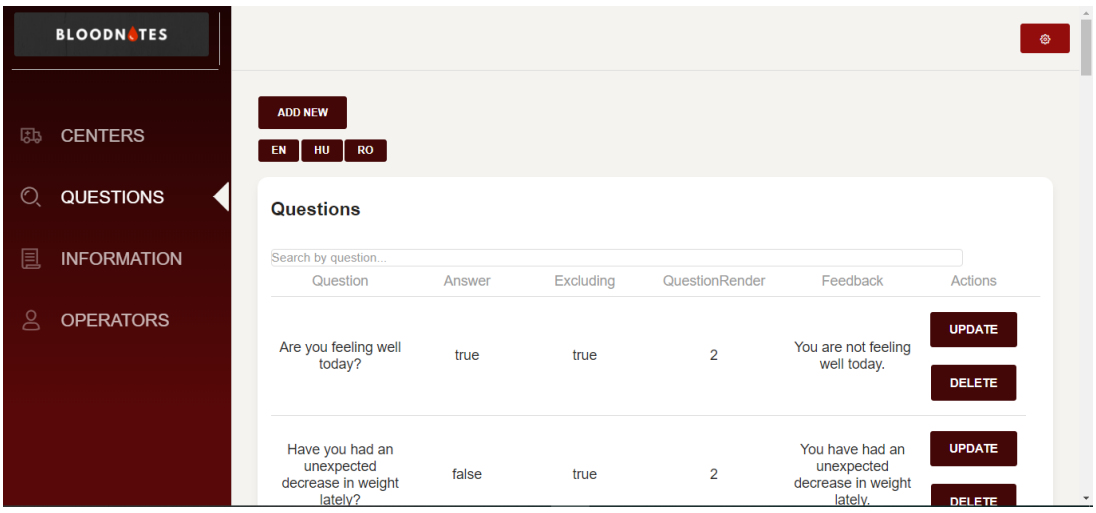
Az alkalmazás használatával lehetőség van a mobilon megjelenő adatok módosítására. Az eltárolt információk módosíthatóak, szerkeszthetőek, törölhetőek, illetve bővíthetőek új adatokkal. Az adatok, melyeket kezelni lehet: központokhoz tartozó adatok, a kérdőívben megjelenő kérdések, az információs oldalak tartalma, illetve a kórházi alkalmazottak adatai.

Mivel a mobilalkalmazást több nyelven is lehet használni (magyarul, románul és angolul), a weboldalon a felhasználó ki tudja választani, hogy az adatoknak melyik nyelven megadott verzióját szeretné módosítani.

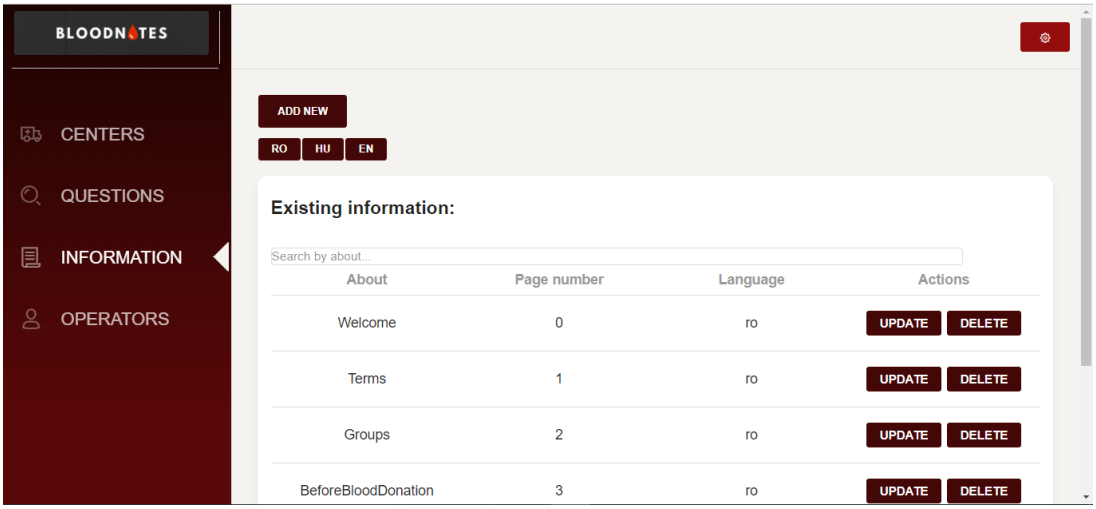
Az adatok kezelésére szolgáló oldalakat a 21., 22., 23., 24. ábrák szemléltetik.



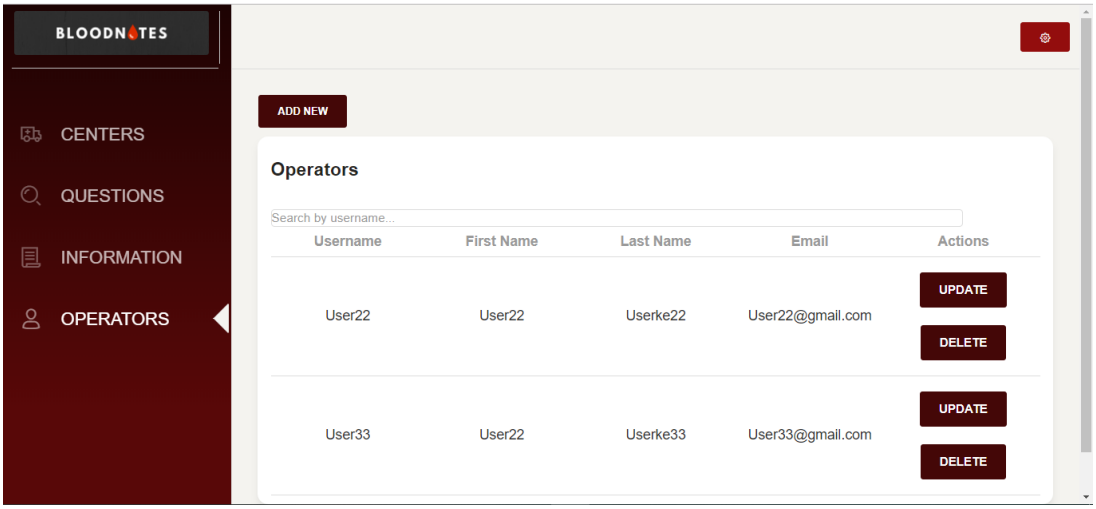
21. ábra. A központokhoz tartozó adatok kezelésére szolgáló oldal.



22. ábra. A véradói kérdőív kérdéseihez tartozó adatok kezelésére szolgáló oldal.



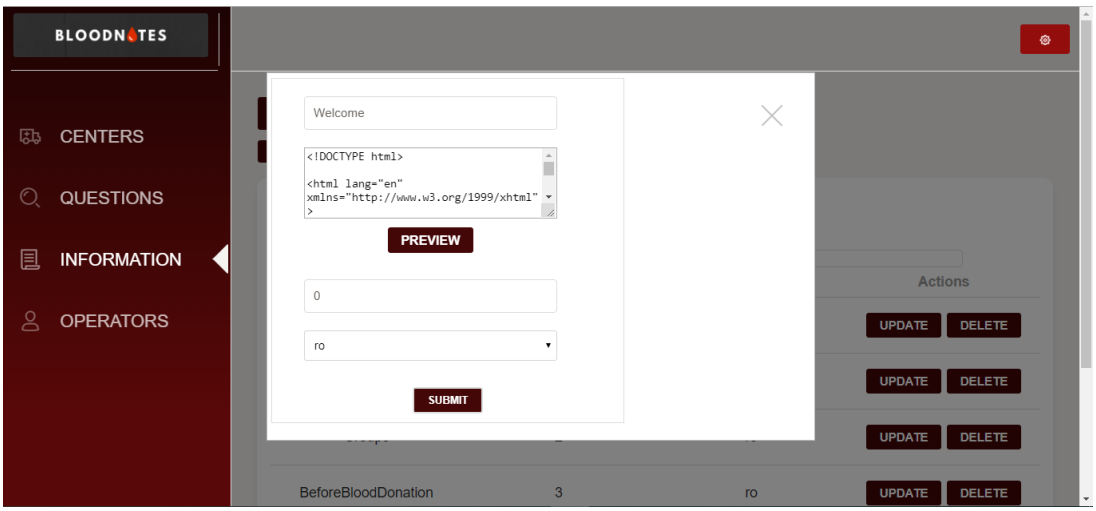
23. ábra. A bevezető informatív oldalak tartalmához tartozó adatok kezelésére szolgáló oldal.



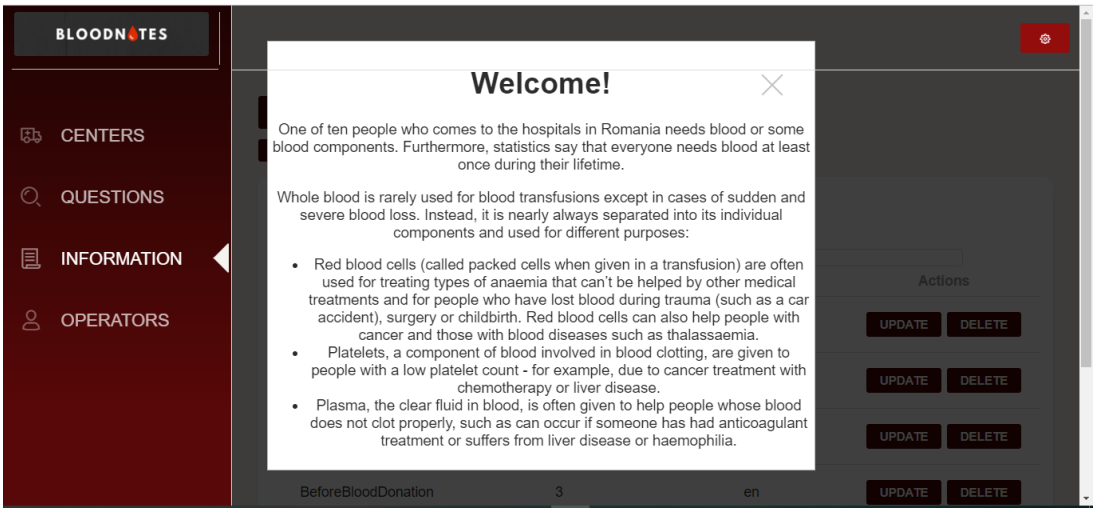
24. ábra. A kórházi alkalmazottakhoz tartozó adatok kezelésére szolgáló oldal. Ezt csak az adminisztrátorok érik el.

Mivel az információs oldalak tartalmai HTML oldalakként vannak eltárolva a rendszer adatbázisában, a felhasználók egy HTML kódot kell szerkesszenek a weboldalon. Mivel ez nehézkes lehet, ezért az oldal egy előnézetet biztosít a felhasználók számára, mellyel folyamatosan ellenőrizni tudják módosításaik eredményét.

Az információs oldalak szerkesztésének folyamatát a 25. és 26. ábrák szemléltetik.



25. ábra. Az információs oldalak szerkesztésére szolgáló oldal.



26. ábra. A HTML oldalak módosítása után az előnézet segítségével ellenőrizhetőek a változások.

Következtetések és továbbfejlesztési lehetőségek

A fejlesztés során megvalósult egy három komponensből álló szoftverrendszer, amely a véradáshoz kapcsolódó folyamatokat hivatott egyszerűsíteni, továbbá célja a véradás népszerűsítése is.

A rendszerhez tartozó webalkalmazás lehetőséget nyújt a véradó központok és mobiltelefonos felhasználók közti kommunikációra. A központban dolgozók azontúl, hogy módosítani tudják a mobiltelefonokon megjelenő információkat, értesítést küldhetnek a felhasználóknak az aktuális vérszükségletről.

Az iOS és Android mobiltelefonos felületeken elérhető alkalmazáson belül a felhasználók olyan funkcionalitásokat érnek el, melyek a saját véradói tevékenységük rendszerezésében nyújtanak segítséget.

A szoftverrendszerrel kapcsolatos továbbfejlesztési lehetőségek terén számos opció merül fel.

Egy gamification jellegű megoldással, egy pontozási rendszerrel lehetne bővíteni a mobiltelefonos alkalmazást, így a felhasználóknak legetőségük lenne az egymás közti versengésre. A véradások dokumentálásával, új felhasználók meghívásával, vagy a véradások közösségi oldalakon való közzétételével szerezhetnének pontokat a felhasználók.

A kórházi alkalmazottnak lehetősége lehetne arra, hogy a webes felületről meghirdessen egy olyan időpontot, melyen a véradó központ tud fogadni egy donort. Erre az időpontra a mobiltelefonos felhasználó telefonján keresztül be tudna jelentkezni véradásra. Így a kórházi felhasználók ki tudnának alakítani egy stabil programot, a donorok pedig tudnák, hogy mikor érkezzenek a véradó központba ahhoz, hogy ne kelljen huzamosabb ideig várakozzanak.

A véradás népszerűsítését szolgálná az a webes funkció is, mellyel egy kórházi alkalmazott kampányt indíthat egy véradáshoz kapcsolódó esemény érdekében. Ez a kampány megjelenik a mobiltelefonokon, informálva a felhasználókat az eseményről.

Technikai szempontból a mobilalkalmazást egy adatbázissal lehetne bővíteni, annak érdekében, hogy a felhasználókra vonatkozó adatok elmentése kliens oldalon történjen. Ennek előnye, hogy ezeknek az adatoknak az eléréséhez nem lenne szükség a szerverrel való kommunikációra, ezek az adatok a rendszer offline állapotában is elérhetőek lennének.

A projekt bemutatásra került a VI. Digitális Székelyföld Konferencián, valamint a XXII. Erdélyi Tudományos Diákköri Konferencián, ahol II. helyezést ért el.

Hivatkozások

[1] Országos Vérellátó szolgálat. *Ki lehet véradó?* 2015. URL: http://www.ovsz.hu/ver/ki-lehet-verado (utolsó elérés dátuma: 2019. ápr. 30.)

[2] Axosoft. *GitKraken*. URL: https://www.gitkraken.com/about (utolsó elérés dátuma: 2019. ápr. 13.)

[3] Scott Chacon. *Git*. URL: https://git-scm.com/ (utolsó elérés dátuma: 2019. ápr. 13.)

[4] Oracle Corporation. *MySQL*. URL: https://dev.mysql.com/doc/ (utolsó elérés dátuma: 2019. ápr. 13.)

[5] Oracle Corporation. *MySQL Workbench*. URL: https://www.mysql.com/products/workbench/ (utolsó elérés dátuma: 2019. ápr. 13.)

[6] Apple Developer. *UIViewController*. URL: https://developer.apple.com/documentation/uikit/uiviewcontroller (utolsó elérés dátuma: 2019. ápr. 12.)

[7] Android Developers. *Activity*. URL: https://developer.android.com/reference/android/app/Activity (utolsó elérés dátuma: 2019. ápr. 12.)

[8] MDN web docs. *Body.json()*. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/API/Body/json (utolsó elérés dátuma: 2019. ápr. 13.)

[9] MDN web docs. *Fetch API*. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (utolsó elérés dátuma: 2019. ápr. 13.)

[10] MDN web docs. *Request*. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/API/Request (utolsó elérés dátuma: 2019. ápr. 13.)

[11] MDN web docs. *Response*. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/API/Response (utolsó elérés dátuma: 2019. ápr. 13.)

[12] MDN web docs. *Window.sessionStorage*. 2019. URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage (utolsó elérés dátuma: 2019. ápr. 15.)

[13] Google Group. *Firebase*. URL: https://firebase.google.com/ (utolsó elérés dátuma: 2019. ápr. 13.)

[14] Docker Inc. *Docker*. URL: https://www.docker.com/ (utolsó elérés dátuma: 2019. ápr. 13.)

[15] Docker Inc. *Docker Compose*. URL: https://docs.docker.com/compose/ (utolsó elérés dátuma: 2019. ápr. 13.)

[16] Facebook inc. *React js*. URL: https://reactjs.org/ (utolsó elérés dátuma: 2019. ápr. 13.)

[17] Gitlab Inc. *Gitlab*. URL: https://about.gitlab.com/ (utolsó elérés dátuma: 2019. ápr. 13.)

[18] Gitlab Inc. *Gitlab CI*. URL: https://about.gitlab.com/product/continuous-integration/ (utolsó elérés dátuma: 2019. ápr. 13.)

[19] N. Sakimura M. Jones J. Bradley. *JSON Web Token (JWT)*. 2015. URL: https://www.rfc-editor.org/rfc/pdfrfc/rfc7519.txt.pdf (utolsó elérés dátuma: 2019. ápr. 7.)

[20] Andrei Moldovan. *MVVM design pattern*. URL: https://www.todaysoftmag.ro/article/525/mvvm-design-pattern (utolsó elérés dátuma: 2019. ápr. 7.)

[21] *Newtonsoft.Json Framework*. URL: https://www.newtonsoft.com/json (utolsó elérés dátuma: 2019. ápr. 11.)

[22] *Node Package Manager*. URL: https://docs.npmjs.com/about-npm/index.html (utolsó elérés dátuma: 2019. ápr. 13.)

[23] Mark Reynolds. *Xamarin Mobile Application Development for Android*. Packt Publishing Ltd., 2014.

[24] Dimitris Tavlikos. *iOS Development with Xamarin Cookbook*. Packt Publishing Ltd., 2011.

[25] TypeScript team. *JSX*. 2019. URL: https://www.typescriptlang.org/docs/handbook/jsx.html (utolsó elérés dátuma: 2019. ápr. 13.)

[26] Babel team. *What is Babel?* 2015. URL: https://babeljs.io/docs/en/ (utolsó elérés dátuma: 2019. ápr. 15.)

[27] Bootstrap Team. *Bootstrap*. URL: https://getbootstrap.com/ (utolsó elérés dátuma: 2019. ápr. 13.)

[28] Microsoft Team. *Entity Framework*. URL: https://docs.microsoft.com/en-us/ef/core/ (utolsó elérés dátuma: 2019. ápr. 13.)

[29] Microsoft Team. *MVVM Light*. URL: https://www.nuget.org/packages/MvvmLight (utolsó elérés dátuma: 2019. ápr. 14.)

[30] Microsoft Team. *.NET Core*. URL: https://docs.microsoft.com/en-us/dotnet/core/ (utolsó elérés dátuma: 2019. ápr. 13.)

[31] Microsoft Team. *Nuget*. URL: https://www.nuget.org/profiles/Microsoft (utolsó elérés dátuma: 2019. ápr. 13.)

[32] Microsoft Team. *Visual Studio*. URL: https://visualstudio.microsoft.com/ (utolsó elérés dátuma: 2019. ápr. 13.)

[33] Microsoft Team. *Visual Studio Code*. URL: https://code.visualstudio.com/ (utolsó elérés dátuma: 2019. ápr. 13.)

[34] Microsoft Team. *Xamarin*. URL: https://university.xamarin.com/ (utolsó elérés dátuma: 2019. ápr. 13.)

[35] Microsoft team. *HttpClient Class*. URL: https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=netframework-4.7.2 (utolsó elérés dátuma: 2019. ápr. 11.)

[36] Microsoft team. *HttpRequestMessage*. URL: https://docs.microsoft.com/en-us/dotnet/api/system.net.http.httprequestmessage?view=netframework-4.7.2 (utolsó elérés dátuma: 2019. ápr. 11.)

[37] Microsoft team. *System.Net.Http Namespace*. URL: https://docs.microsoft.com/en-us/dotnet/api/system.net.http?view=netframework-4.7.2 (utolsó elérés dátuma: 2019. ápr. 11.)

[38] React team. *React.Component*. 2019. URL: https://reactjs.org/docs/react-component.html (utolsó elérés dátuma: 2019. ápr. 15.)