XXI. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)
Kolozsvár, 2018. május 24-27.

Magic Dashboard

Szoftverrendszer informatikai projektek állapotának valós idejű nyomon követésére

Szerzők:
Daragus Botond
Babeş-Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év
Fodor Lóránt
Babeş-Bolyai Tudományegyetem, Matematika és Informatika Kar, Informatika szak, III. év

Témavezetők:
Sulyok Csaba, doktorandusz,
Babeş-Bolyai Tudományegyetem,
Matematika és Informatika Kar
Kelemen-Fehér Dénes Bálint,
szoftverfejlesztő, Codespring
Mátis Szilárd-Gábor,
szoftverfejlesztő, Codespring

Kivonat

Napjainkban a szoftverfejlesztők egyre ritkábban kötik le figyelmüket egyetlen alkalmazás fejlesztésével. A projektek mögött pedig legtöbbször a projektmenedzserek állnak, akik még inkább fel kell osszák idejüket, figyelmüket és koncentrációjukat a rájuk bízott projektek között.

A Magic Dashboard elsődleges célja, hogy ezen szoftverfejlesztők és projektvezetők munkáját könnyítse meg úgy, hogy naprakészek maradnak a projektjeikben történő változásokkal. A projekt jól látható vizuális elemre fekteti a hangsúlyt, a képernyőre - ez lehet egy falba épített kijelző is -, ahol a felhasználó által beállított projektmenedzsment rendszerekből származó, fontos adatok összegyűjtésére és megjelenítésére kerül sor. Ez a lehetőség felgyorsíthatja a projektek folyamatos megfigyelését, ellenőrzését és karbantartását.

A dolgozat bemutatja az alkalmazás részletes architektúráját, kitér a felhasznált technológiákra és eszközökre, majd a projekt működését szemlélteti valós környezetben.

Tartalomjegyzék

Bevezető

Napjainkban az embereknek egyre kevésbé marad szabadidejük, azt érzik, hogy lassan elhalad mellettük a világ, pedig minden egyes nap lépést próbálnak tartani a felgyorsult fejlődéssel, rohanással. Ennek hatására olyan megoldásokat próbálnak keresni, amik felgyorsíthatják és könnyebbé tehetik a mindennapi munkavégzést. Erre az egyik legjobb módszer, hogy állapotok valós idejű visszajelzést szolgáló rendszereket hoznak létre.

Ilyen komplex rendszereket fejleszt és biztosít például a Valarm[1], a műszerfalakon megjeleníthető aktuális állapotok követésére, amelyek használhatóak a nagy tüzek megelőzésében, az időjárás helyzetének megfigyelésében, ipari gépek állapotainak valós követésében stb. Ezen rendszerek egyik megfelelője az átlagos emberi életre, a mindennapokban használható személyes asszisztens MagicMirror^2 projekt [2]. Ez a rendszer volt a Magic Dashboard projekt ötletének egyik fő alappillére és mivel a szoftverfejlesztés területén is nagyon szükséges a produktív időbeosztás, így az alkalmazás erre az iparágra lett kitalálva és fejlesztve.

A Magic Dashboard szoftverrendszer elsődleges célközönsége tehát a szoftverfejlesztők vagy még inkább a projektmenedzserek, akiknek legtöbbször egyidejűleg több projektet is kell koordináljanak és néha nagyon sok energiájukba kerül feleleveníteni, hogy milyen fázisban látták utoljára a projektet. A szoftver lehetőséget add a nap 24 órájában, az adott projekteken végbemenő tevékenységek folytonos megfigyelésére, követésére. Ezt úgy valósítja meg, hogy egy közös felületre összegyűjti a fontosabb, megjelenítésre kívánt adatokat. A különböző értékek nem statikusak, hanem valós időben frissülnek a kijelzőn. A jobb átláthatóság érdekében előnyösebb egy minél nagyobb megjelenítésre alkalmas készüléket használni, legyen az hagyományos vagy okos TV, képernyő, vetítőgép vagy akár egy falba épített okos tükör, a látni kívánt adatok mennyiségétől függően. A szoftver az adatokat egy kiválasztott projektmenedzsment rendszertől kapja, jelenleg a GitLab[3] rendszertől. Innen nyeri a szoftverfejlesztő csapat projektjeihez tartozó szükséges adatokat. A rendszer tervezése során fontos szempont volt, hogy a cégek belső szolgáltatásaival is együtt tudjon müködni, ezért a rendszer egy sor konfigurációs lehetőséget ad a rendszergazdáknak. A felhasználók a saját jogosultságainak megfelelően tudnak különböző adatokat megjeleníteni a kijelzőkön.

A Magic Dashboard projekt tehát, elsődleges célként azt tűzte ki, hogy az alkalmazás használatával időt spóroljon a szoftveriparban dolgozóknak. A projekt fejlesztői igyekeztek a tervezéskor és fejlesztéskor olyan felületet megvalósítani, amely a felhasználók szempontjából hasznos, átlátható és logikus működésű.

A dolgozat bemutatja a Magic Dashboard projekt főbb funkcionalitásait, architekturális felépítését, majd részletezi a felhasznált technológiákat és eszközöket, továbbá taglalja az alkalmazott módszerek fontosabb alkotóelemeit is. A dolgozat végéhez érve betekintést ad az alkalmazás működés közbeni használatára, végül megemlít néhány kiemelt továbbfejlesztési lehetőséget. Az alkalmazás fejlesztésének kezdete a „Csoportos projektek” tantárgy keretein belül kezdődött, 2017 októberétől és a Codespring Mentorprogram részét képezte. Így a projekt létrejöttéért köszönet illeti dr. Simon Károly és Sulyok Csaba koordinátorokat, Kelemen-Fehér Dénes Bálint és Mátis Szilárd-Gábor mentorokat, valamint a tantárgy keretein belül csatlakozó diáktársakat: Beiland Arnold, Borsos Barna és Kovács Péter-Róbert.

1. Magic Dashboard projekt

A Magic Dashboard egy szoftverrendszer, amely a felhasználó által beállított különböző projektmenedzsment rendszerekből származó adatok megjelenítését teszi lehetővé. A rendszer használatával a felhasználók egy átfogó képet kapnak az általuk használt eszközökből összegyűjtött adatokról, az általuk beállított virtuális műszerfalon keresztül.

A projekt motivációja abból adódott, hogy fejlesztőként egy projektre vonatkozó különböző információkat sok esetben több projektmenedzsment rendszerből lehet csak elérni. A cél egy olyan rendszer létrehozása, amelyet használva a fejlesztők, vagy akár a projektek vezetői, ezeket az információkat egy helyen tudják megjeleníteni és a változásokat valós időben követni.

1.1. A projekt funkcionalitásai

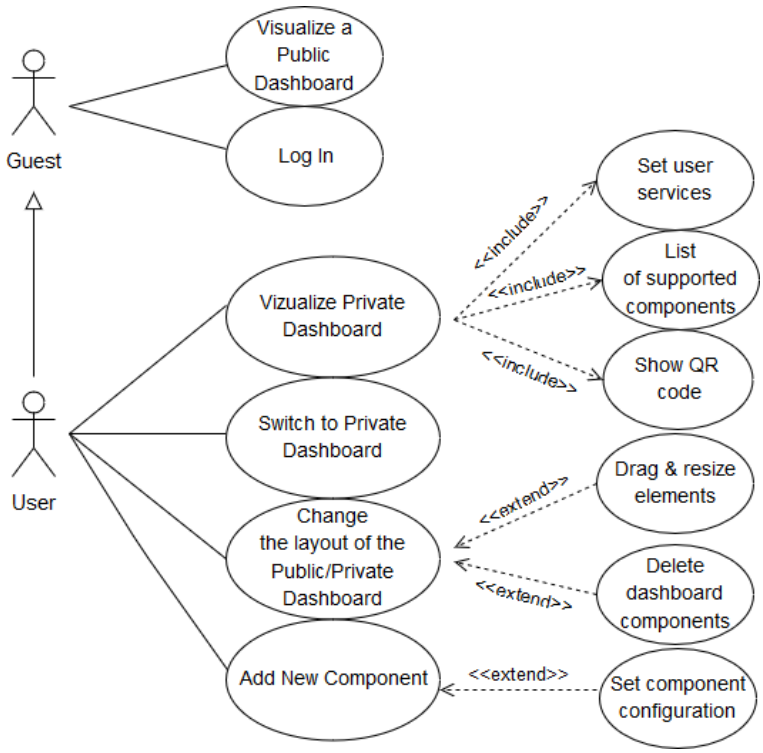
A fejezet a Magic Dashboard szoftverrendszer fontosabb funkcionalitásait részletezi. Az első részben a webes kliens, majd a mobil kliens funkcionalitásait mutatja be.

1.1.1. A webes felület funkcionalitásai

A felhasználók a webes felületen keresztül lépnek interakcióba a szoftverrendszerrel (1. ábra). A webes felület megtekinthető vendégként vagy bejelentkezett felhasználóként. A felhasználónak lehetősége van egy a GitLab által érvényesített [4] bejelentkezésre, amely olyan jogosultsággal ruházza fel, hogy a saját (Private Dashboard) vagy akár a „közös műszerfalat” (Public Dashboard) is tudja módosítani. Amennyiben nem jelentkezik be a rendszerbe, csak a publikus műszerfal nézet (Public Dashboard View) érhertő el számára, ahol a fejlesztőcsapat közösen összerakott műszerfalát tekintheti meg.

Amennyiben a bejelentkezés mellett dönt, akkor beléphet a szerkesztő nézetbe (Edit Dashboards View) is. Itt alapértelmezetten a felhasználó közös műszerfala lesz látható, amin kezdetben egy segítséget nyújtó komponens (Help text component[1]) jelenik meg, tartalmazva a privát műszerfala azonosító nevét és QR kódját [5] , amit a mobil kliens segítségével tud felhasználni. Az első

[1] egy olyan kisebb mozgatható felület, ami fontos információt jelenít meg a felhasználó számára



1. ábra. A webes felület használati esetei a vendég, valamint a hitelesített felhasználók szempontjából.

lépés ahhoz, hogy a felhasználó módosításokat tudjon végezni a műszerfalon az az, hogy a szolgáltatások (Services) fül alatt be kell állítson a GitLab szolgáltató által biztosított egyéni azonosító kódot (personal access token). Az azonosító beállítasa után szabadon helyezhet el, méretezhet át, mozgathat és törölhet a komponensek közül.

A komponensek műszerfalra helyezése után az adott komponenshez kapcsolódó beállításokat lehet elvégezni. Például, az utolsó hozzájárulók (Latest Contributors) komponens személyre szabásakor megadható, hogy melyik projektekhez tartozó hozzájárulók profilképe legyen megjelenítve és beállítható az is, hogy az utolsó hány ilyen aktivitás legyen látható.

A fent említett funkcionalitások ugyanúgy kivitelezhetőek a privát műszerfalon is. A felhasználónak több lehetősége van arra, hogy átváltson a saját műszerfalára: mobil alkalmazás segítségével, navigációs ikon segítségével vagy akár a Slack parancs[6] végrehajtásával. Ekkor a publikus műszerfal egy rövid időre a fejlesztő privát műszerfalát mutatja.

Átváltva a publikus nézetre, az előzőleg beállított komponensek lesznek elérhetőek külső interakció lehetősége nélkül. Az említett nézet egy olyan tükörre vagy képernyőre van tervezve, ahol a

felhasználóknak nincs lehetőségük a komponensek módosítására, mozgatására.

A felhasználóknak olyan adatokat jelenít meg a webes felület, amelyekhez egyébként is lenne hozzáférése, de a testreszabhatósága miatt azonnal és egységes felületen férhetnek hozzá különböző metrikákhoz. A megfelelő jogosultságok kezelése az adatokat szolgáltató rendszerek felelősége. A megjelenített adatok áteshetnek bizonyos előfeldolgozáson, vagy összegzésen, hogy a rendszert használók számára a lényeges adatokat jelenítsék meg.

1.1.2. A mobil alkalmazás funkcionalitásai

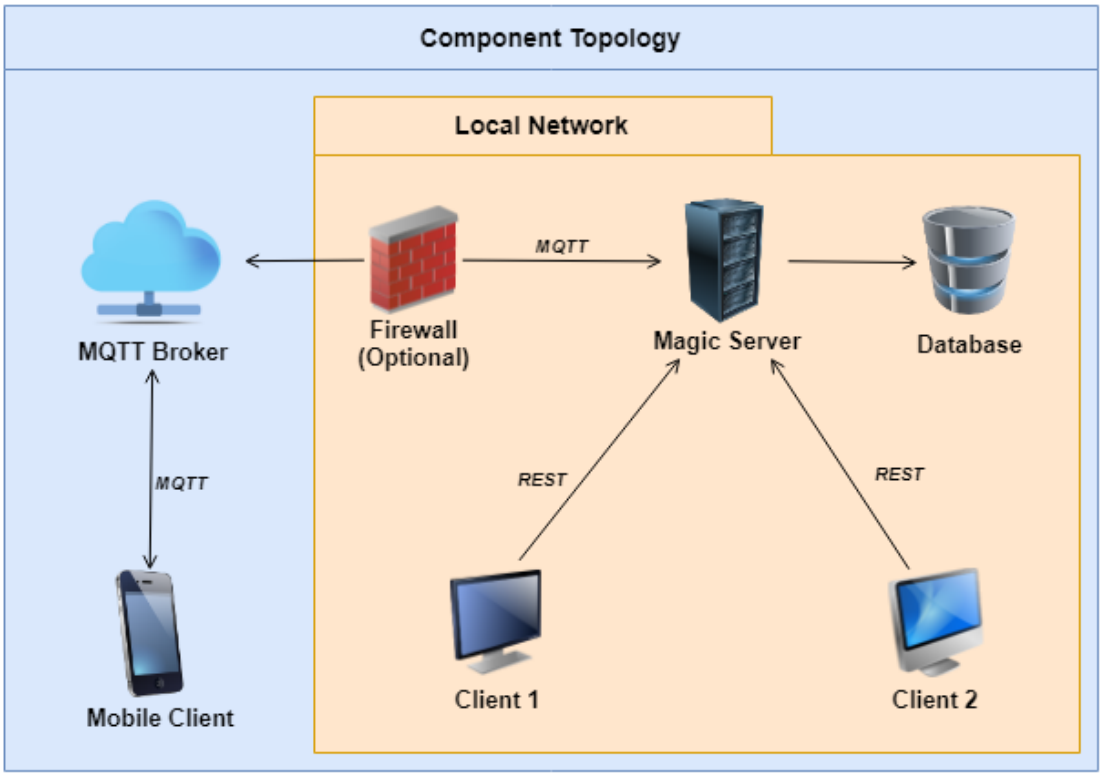
A szoftverrendszer lehetőséget biztosít a webes felületen megjelenített műszerfalak közötti váltásra. Ezt a felhasználók Android vagy iOS rendszeren futtatható mobilalkalmazás segítségével tehetik meg. Az alkalmazás szoros kapcsolatban áll a webes felülettel, hiszen funkcionalitásainak eredményei ott lesznek nyilvánosan láthatóak. Ezáltal lehetőséget nyújt a kijelzőkön megjelenített publikus műszerfal leváltására. A leváltás olyan privát műszerfalra lehetséges, amely a mobil alkalmazás birtokában lévő felhasználó jogos tulajdona, személyes műszerfala.

A felhasználó a műszerfalváltást kétféleképpen kivitelezheti az alkalmazásban: megadhatja a műszerfala nevét vagy beolvashatja az integrált QR kód olvasóval a webes felületen megjelenített QR kódot. Az alkalmazásban használt műszerfalak nevei tárolásra kerülnek egy listában, így nem kell a felhasználó minden váltás előtt begépelje, vagy beolvassa azt. A felhasználó a publikus műszerfal nézetre való visszaváltást is kivitelezheti, abban az esetben ha épp a privát műszerfal van megjelenítésen.

1.2. Architektúra

A Magic Dashboard rendszer architektúráját négy komponens alkotja: egy webszerver, egy webes kliens, egy mobil kliensalkalmazás, illetve egy MQTT[7] (Message Queuing Telemetry Transport) Broker. Ezen komponensek topológiáját az 2. ábra szemlélteti.

Az architektúra mivoltát nagyban befolyásolja a kitelepítési környezet, mivel a szerver és a webes kliensek többnyire lokális (pl. céges) hálózatokon belül lesznek futtatva. Ezek a lokális hálózatok biztonsági okokból tűzfal által védve vannak a hálózaton kívüli eszközökről érkező kérésektől,

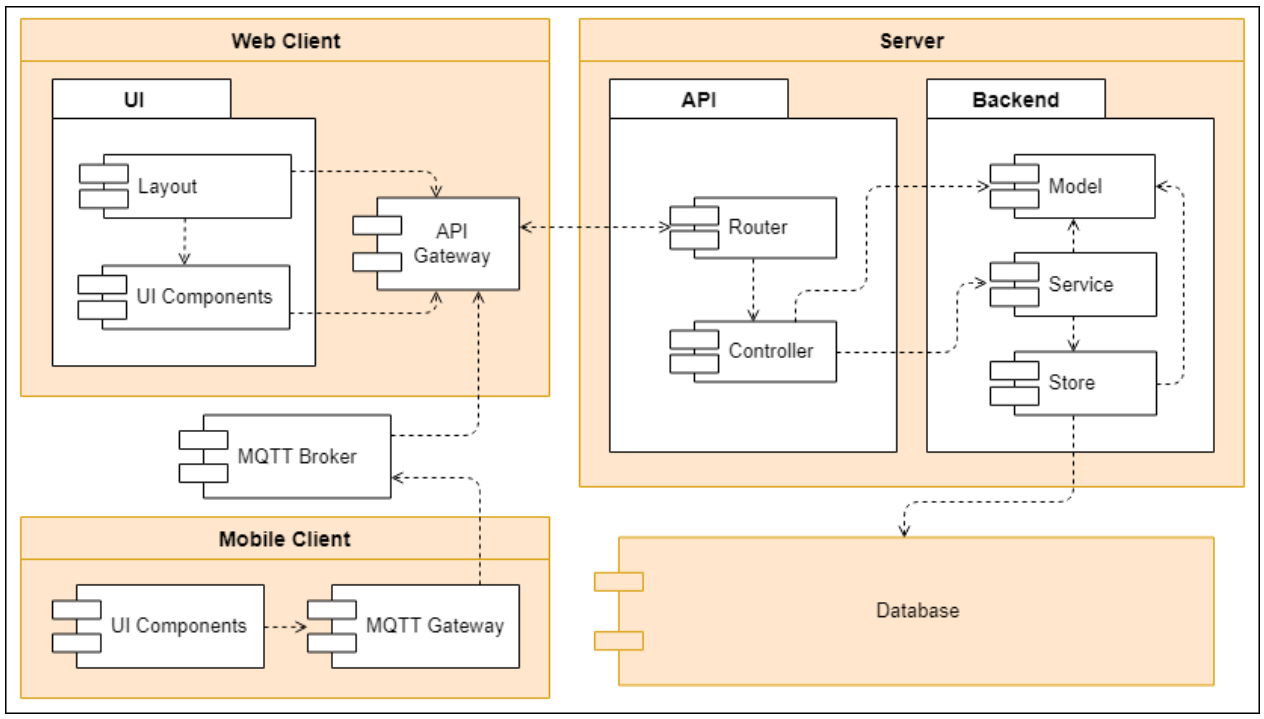


2. ábra. A Magic Dashboard szoftverrendszert képező komponensek, a szerver, a mobil és a webes kliensalkalmazások topológiája

emiatt a felhasználók személyes okostelefonjai és a szerver közötti kommunikáció nem tudna megvalósulni. Erre a problémára egy üzeneteket továbbító technológia, az MQTT Broker bevezetése nyújt megoldást. Segítségével a hálózaton kívüli kliensek a kéréseiket először ennek a komponensnek küldik, majd ez továbbküldi a lokális hálózaton belüli kliensnek, így a kérés a szerverre már egy megbízható eszközről érkezik. Ezen kommunikáció megvalósításához nincs szükség extra tűzfal szabályokra, így a rendszer valamint a hálózat biztonsága sem csorbul.

A Magic Dashboard rendszerét felépítő komponensek további alkomponensekre oszthatóak. Ezen alkomponensek és a köztük levő kapcsolatok a 3. ábrán láthatóak.

A szerver oldal fő feladatai közé az adatok kezelése valamint a kliensek kéréseinek kiszolgálása tartozik, ugyanakkor a külső rendszerekkel való kommunikációért is felelős. A Model csomagban levő struktúrák tartalmazzák az alkalmazás központi entitásainak reprezentációját, amelyeket az adathozzáférési (Store), szolgáltatási (Service) és vezérlő (Controller) rétegek közvetlenül használnak. Az adatok tárolása egy dokumentumorientált NoSQL adatbázis, a RethinkDB[8] segítségével



3. ábra. A Magic Dashboard rendszer szerverét, mobil és webes kliensalkalmazását alkotó komponensek diagramja

történik.

Az adatbázis-kezelő rendszerrel az adathozzáférési (Store) réteg kommunikál, itt történik az adatok adatbázisba való beszúrása illetve az abból való lekérése. A szolgáltatási (Service) rétegen belül vannak implementálva az alkalmazás logikájához tartozó műveletek, amelyek a vezérlő (Controller) réteg számára interfészeken keresztül érhetőek el. A kliensektől érkező REST (Representational State Transfer) kéréseket az API réteg (Router) fogadja, majd különböző vezérlők dolgozzák fel és a szolgáltatási rétegekel együttműködve adatokat térítenek vissza a kliensek számára.

A webes kliensalkalmazás (Web Client) felelős a felhasználói felület megjelenítéséért. A szerverhez hasonlóan itt is megtalálhatóak struktúrák formájában a megjelenítéshez szükséges entitások reprezentációi. A szerverrel való kommunikációt az „API Gateway” nevű réteg biztosítja, ennek metódusait a felhasználói felület legtöbb komponense közvetlenül használja.

A mobil kliens feladata a műszerfalak távolról való publikálásáért szükséges felület megjelenítése, illetve az MQTT Brokerrel (ezáltal pedig egy lokális hálózaton levő webes klienssel) való kommunikáció. A kérések küldését és fogadását az MQTT Gateway réteg valósítja meg.

2. Felhasznált technológiák

Az alábbiakban a Magic Dashboard rendszer fejlesztése során felhasznált fontosabb technológiák vannak ismertetve.

2.1. Szerver oldali technológiák

A szerver Go programozási nyelvben írodott, RethinkDB adatbázisba menti az adatokat, Viper könyvtárat használja konfigurációk kezelésére valamint Gin keretrendszerrel publikálja a REST API interfészt. A továbbiákban ezek a technologiák kerülnek bemutatásra.

2.1.1. Go programozási nyelv

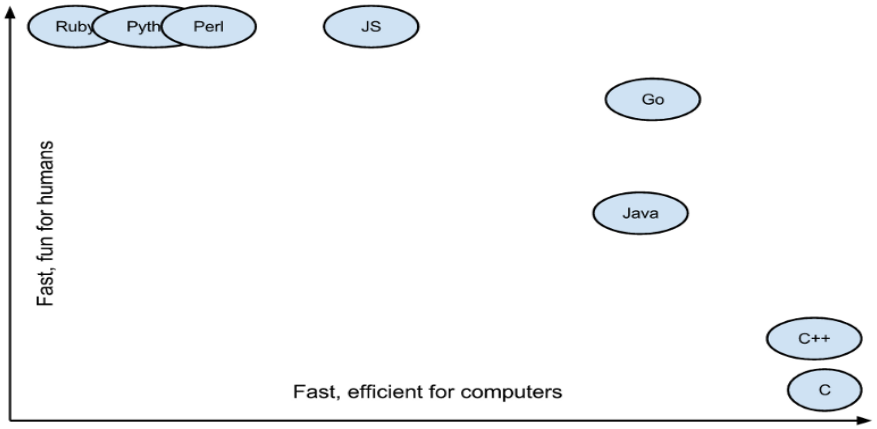
A Magic Dashboard szerver alkalmazás Golang nyelvet felhasználva skálázható és nagyobb terhelés alatt is megállja a helyét.

A Go[9] vagy más néven Golang egy a Google által fejlesztett nyílt forráskódú programozási nyelv. A nyelv megalkotásakor Robert Griesemer, Rob Pike és Ken Thompson figyelt arra, hogy más nyelvek hiányosságait, nehézségeit lehetőség szerint kiküszöböljék. Ezért nem rendelkezik kivételkezeléssel, osztályokkal, dinamikus típusokkal, de biztosít például pehelysúlyú szálkezelést (goroutines), memória menedzsmentet. A pehelysúlyú szálak lehetőséget adnak a több szálon futó konkurens és párhuzamosítható feladatok futtatására, valamint, a C++-al szemben, a memória menedzsment nyújt automatikus szemét kitakarítást is (garbage collection). A szintaxisa hasonlít a C nyelvre, kompilált, ezáltal teljesítmény orientált programkód is írható benne (4. ábra[2]).

2.1.2. RethinkDB és GoRethink

A RethinkDB egy JSON dokumentum alapú, nyílt forráskódú, osztott NoSQL adatbázis, amely skálázható és valós idejű alkalmazásokra van optimalizálva. Lehetőséget biztosít a lekérdezések eredményeinek változásának a valós idejű követésére. Kiemelendő a magas rendelkezésre állás (availability), amit egy automatikus hibakivédő mechanizmussal (failover) és toleráns hibakeze-

[2] `https://medium.com/@kevalpatel2106/why-should-you-learn-go-f607681fad65` [Utolsó megtekintési dátum: 2018.04.08.]



4. ábra. A Go összehasonlítása más magasszintű programozási nyelvekkel számítási gyorsaság (vízszintes tengely) illetve használhatóság (függőleges tengely) szempontjából"

léssel ér el. ReQL lekérdező nyelvvel rendelkezik, amelynek sajátossága, hogy a lekérdezések jól egymáshoz láncolhatóak. Golang nyelvben az adatbázis műveletek elvégzésére a GoRethink[10] nyílt forráskódú könyvtár alkalmas, amelyet a Go közösség fejleszt és tart karban. Segítségével az előre definiált metódusok következetes és biztonságos lekérdezéseket eredményeznek.

2.1.3. Viper

A Magic Dashboard alkalmazás működéséhez elengedhetetlen több konfigurációs érték megadása, mivel a rendszer szoros kapcsolatban áll külső szolgáltatásokkal. Erre a problémára biztosít megoldást a Go-ban fejlesztett Viper [11] könyvtár. A különböző beállítások segítségével az alkalmazás betöltheti a konfigurációs információkat környezeti változokból, állományokból (JSON, TOML, YAML, HCL és Java tulajdonságfájlokból) vagy parancssori argumentumokból. A szoftverfejlesztők ezeknek megadhatnak alapértelmezett értekeket, akár beállíthatnak betöltési, elsőbbségi sorrendet.

2.1.4. Gin

A Gin egy Golangban fejlesztett web keretrendszer, amely a Go teljesítmény mérések [12] alapján jelenleg az egyik leghatékonyabb és leggyorsabb kiszolgálással bíró keretrendszer a vetélytársai közt. Jelentős gyorsaságbeli különbségét a HttpRouter[13]-nek köszönheti ami a Radix tree [19] adatszerkezetet használja. A fejlesztőknek a Gin egy beépített loggolási mechanizmussal teszi

átláthatóbbá a kérések visszakövetését.

A projekt szervere a Gin segítségével egy előre meghatározott „`api/v1/`" elérési útvonal alá csoportosítja a kérés-válaszokat, ami tartalmazza az API verziószámát, így rendszer további fejlesztése során is megtartható a visszafele való kompatibilítás. A beépített köztes ellenörző mechanizmus (middleware) pedig lehetővé teszi a bejövő kérések hitelesítését, biztosítva, hogy az adott parancsok csak akkor hajtódnak végre, ha a kérő fél rendelkezik a megfelelő jogosultságokkal.

2.2. Kliens oldali technológiák

A Magic Dashboard rendszer két kliens alkalmazással rendelkezik: egy webes felhasználói felülettel, valamint egy mobil alkalmazással. A fejezet ezek fejlesztése során felhasznált fontosabb technológiákat ismerteti.

2.2.1. ReactJS

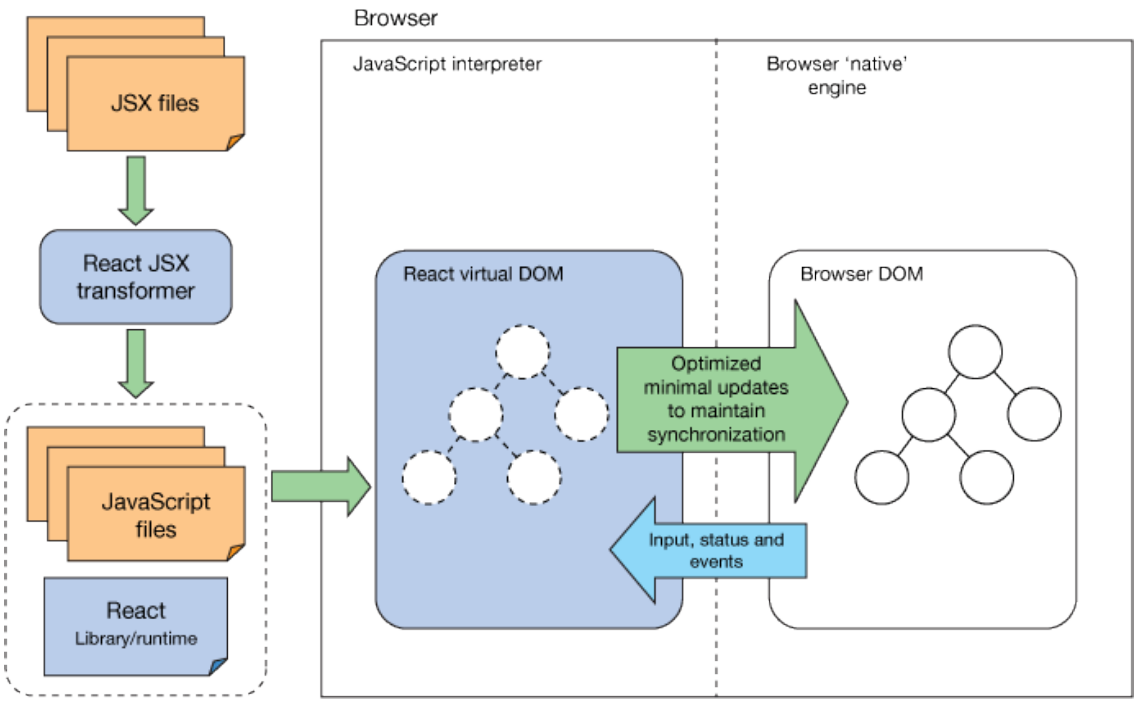
A ReactJS [14] vagy egyszerűen React egy JavaScript alapú könytár, amely SPA-k (Single Page Application) felhasználói felületének elkészítését teszi lehetővé. Egy viszonylag új keretrendszer, amelyet a Facebook fejlesztett ki és használt legelőször, majd nyílt forráskódúvá tett 2013-ban.

A keretrendszer alapját a komponensek képezik, amelyek egy egyszerű szövegmezőtől kezdve bármilyen komplex felhasználói felületi elemek lehetnek. Ezen komponensek egymásba ágyazhatóak, egymás között pedig property-nek nevezett adatokkal kommunikálnak. A komponensek egymásba ágyazhatósága miatt a felhasználói felületek komponensekként megfelelnek az egységbezárás elvének, miszerint egy adott komponens csak saját magáért felelős, ugyanakkor a jól definiált komponensek többször is felhasználhatóak.

Egy React alkalmazás fejlesztése szintaktikai szempontból kétféle módon történhet: JavaScript vagy JSX használatával. A JSX [15] a JavaScript szintaxisának egy kibővítése, segítségével a JavaScript kódban a HTML-hez hasonló struktúrákat is használhatunk, ezzel nagyban megkönnyítve a különböző React komponensek létrehozását valamint az ezeknek egymásba való ágyazását.

A React egy másik erőssége más könyvtárakhoz képest a virtuális DOM használata (5. ábra[3]).

[3] `https://www.ibm.com/developerworks/library/wa-react-intro/` [Utolsó megtekintési dátum: 2018.04.19.]



5. ábra. A ReactJS alkalmazások alkotóelemei és a virtuális DOM

Ez a mechanizmus lehetőséget ad arra, hogy a teljes weboldal helyett csak az a komponens frissüljön, amelynek az állapota használat közben megváltozott.

2.2.2. Golden Layout

A Golden Layout [16] egy JavaScriptben megírt könyvtár, amellyel webes applikációk felhasználói felületének felosztását és elrendezését lehet módosítani. A könytár segítségével a felhasználónak lehetősége nyílik felosztani a weboldal felületét több kisebb ablakra, illetve az ablakokhoz hozzáadhat több fület (tab). Ezek az ablakok egy oldalsó menüből fogd meg és húzd (drag-and-drop) módszerrel bármikor hozzáadhatóak a felülethez vagy törölhetőek onnan, emellett a méretük tetszőlegesen állítható, ezzel testreszabhatóvá téve a felhasználói felületet.

A Magic Dashboard esetében a több ablakos felosztás ad lehetőséget arra, hogy a különböző komponensek adatait különböző ablakok jeleníthessék meg és ezeket a felhasználó tetszés szerint pozícionálhassa az oldalon.

2.2.3. TypeScript

A TypeScript [17] egy Microsoft által fejlesztett nyílt forráskódú programozási nyelv, amely a JavaScriptnek egy szintaktikai kibővítése. A TypeScript bevezetésével lehetőség nyílik osztályok, interfészek, primitív, generikus és egyéb típusok használatára. Míg a JavaScript egy nem típusos és interpretált nyelv, ezért futtatás nélkül sok programozói vagy szintaktikai hiba nem szűrhető ki. A TypeScript nyújt statikus kódelemzés funkciót, amely már fejlesztés közben is megjeleníti az esetleges hibákat.

2.3. Mobil technológiák

2.3.1. React Native

A mobil kliens fejlesztése React Native nyelvben történt, aminek felépítése, koncepciója alapvetően a ReactJS-en alapszik, így fejlesztéséhez szükséges a React gondolkodásmód elsajátítása. Mivel a React egy JavaScript alapú könyvtár, az ECMAScript [20] 2015-ös újításai is gond nélkül használhatóak (arrow functions, import, extends, class stb.), egy összetett, elegáns és hatékonyabb kód érdekében. Az alkalmazás fordításakor natív kód generálódik a különböző mobil platformoknak megfelően, legyen szó Android vagy iOS platformról és ehhez a fejlesztő néhány esetben kell platform specifikus beállításokat végezzen, vagy kódot írjon. Lényeges különbsége a ReactJS-el szemben, hogy új komponens típusokat lehet használni, amelyek kizárólag a mobil alkalmazásokra lettek kifejlesztve. A Magic Dashboard projekt törekszik arra, hogy a mobil alkalmazást minél több platformon lehessen elérni. A natív kódnak köszönhetően az alkalmazás, futtatásakor a platformnak megfelelő felhasználói élményt nyújtja.

2.3.2. Expo

Az Expo[21] egy olyan eszköztár, amit a React Native köré építettek, hogy a szoftverfejlesztők egyszerre tudjanak Android és iOS alkalmazásokat készíteni. Előnye, hogy úgy integrálták a natív kódot az alapkönyvtárakkal, hogy a szoftverfejlesztők nem kell külön Java/Objective-C natív modulokat fejlesszenek. Az Expo alkalmazás pedig egy olyan lehetőséget nyújt a fejlesztőknek, hogy

a környezettől függetlenül elérhetővé teszi a fejlesztés alatt álló alkalmazást, így megkönnyítve állapotának követését és a tesztelés kivitelezését, több valós eszközön is.

A Magic Dashboard mobil alkalmazás az Expo segítségével lett létrehozva, használva voltak az Expo által kínált beépített könyvtárak és használata elősegítette a fejlesztési folyamatokat.

2.4. További technológiák

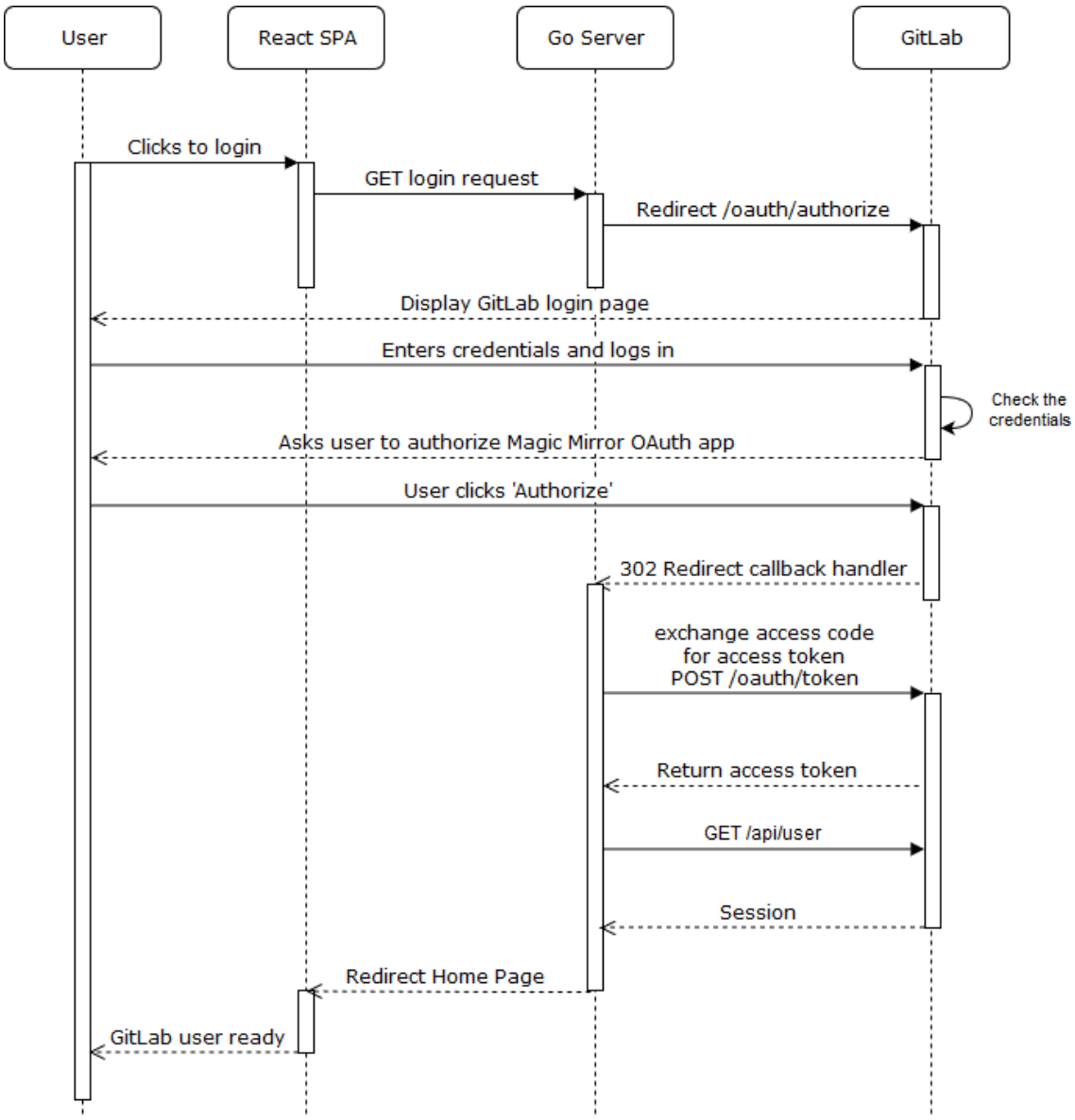
Ezen technológiák fontos szerepet játszottak a Magic Mirror projekt megvalósításához, azonban szorosan nem kapcsolódnak sem a szerver oldali, sem pedig a kliens oldali technológiák köréhez vagy éppen mindkét oldalon ugyanolyan fontossággal bírnak.

2.4.1. OAuth2

Az OAuth2 [22] egy protokoll, amely lehetőséget ad egy HTTP szolgáltatás, jelen esetben a Magic Dashboard alkalmazás felhasználói hitelesítésére, egy megbízható külső szolgáltató segítségével, mint például a GitLab rendszer. Az OAuth2 egyaránt lehetőséget ad webes, asztali és mobil alkalmazások hitelesítésére. A GitLab mint OAuth2-es szolgáltató három féle meghatalamazási folyamatot különböztet meg, úgynevezett "web alkalmazás folyamatok", "implicit folyamatok" és "erőforrás tulajdonos jelszó hitelesítő folyamatok". A Magic Mirror projekt a Web alkalmazás érvényesítési típusú folyamatokat használja, mint legmegbízhatóbb és legelterjedtebb protokollt. A 6. ábra ennek könnyebb megértésére és a megvalósított lépések átláthatóságában nyújt segítséget.

2.4.2. MQTT

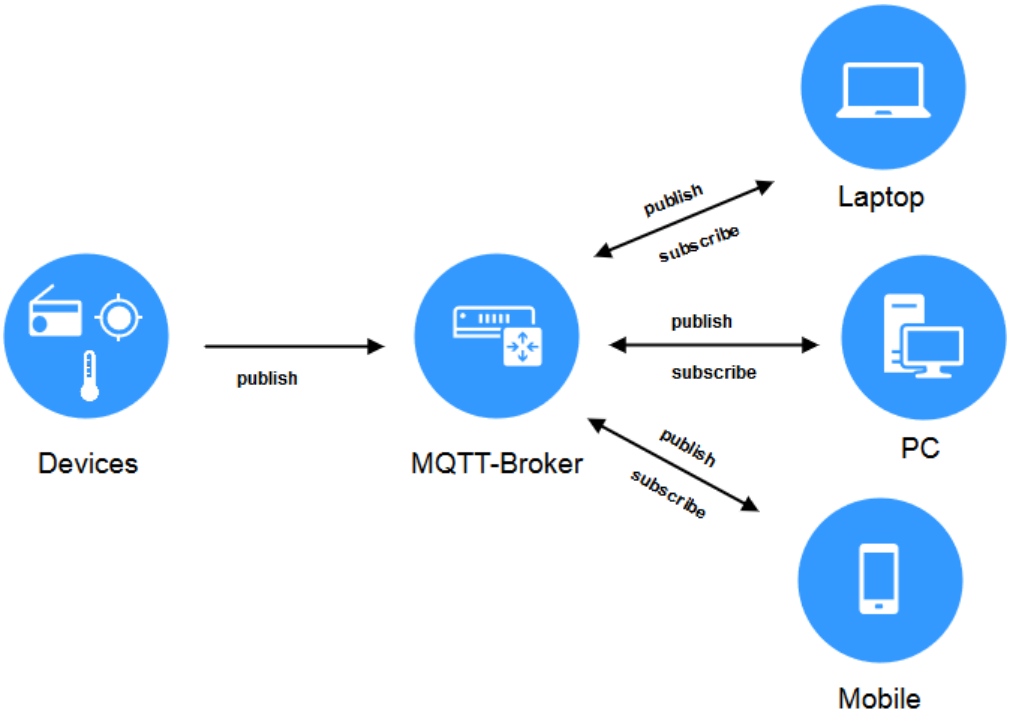
Az MQTT egy olyan pehelysúlyú üzenet szállítási protokoll (7. ábra), amit elsősorban az IoT eszközöknek terveztek, de ugyanakkor előszeretettel használják a mobil alkalmazások világában is. Mivel az IoT világában a kevés erőforrás használata egy nagyon fontos szempont, az MQTT protokoll egyik legfontosabb előnye a kis sávszélességen való gyors adatszállítás, kevés energiafelhasználással. Emellett fontos célja megalkotása óta a könnyed megvalósítás és az aszinkron működés elve is. Az MQTT protokoll és sajátosságai lehetőséget adnak arra, hogy megvédjék a belső hálózatok sértetlenségét, hisz egy külső eszköz úgy tud kapcsolatba lépni egy belső eszközzel, hogy



6. ábra. A bejelentkezéshez szükséges folyamatok, a felhasználó kattintásától a GitLab szolgáltató hitelesítéséig .

annak nem tud semmit kilétéről. Szükséges kiemelni a broker szerverek fontosságát, amelyek mint egy köztes pont (továbbítók) két kliens vagy akár kliens szerver között állnak.

A kérések az úgynevezett publish/subscribe mechanizmussal működnek, miszerint az üzenet-küldő közzé teszi az üzenetét a broker irányába egy adott téma (topic) néven és az üzenetfogadók ha fel vannak iratkozva rá, akkor gond nélkül megkapják ezt az üzenetet. Mivel az üzenet nem egy kiválasztott fogadónak van küldve, hanem egy adott topic névre, így bárki megkaphatja azt, aki előzőleg feliratkozott erre a névre és ez fordítva is igaz.



7. ábra. MQTT kommunikációs diagram

A Magic Dashboard projekt esetén a mobil alkalmazás MQTT protokollt használ a műszerfalváltás kivitelezésére. Használatának fő oka, hogy a mobil alkalmazás nem kell direkt kommunikációt folytasson a belső hálózattal, hanem ezt egy megbízható külső szerveren keresztül teheti meg. Az alkalmazás elküldi az adott műszerfal nevét a meghatározott témára, egy mindkét fél számára elérhető broker szerverhez, amelyre a webes kliens feliratkozott és így azonnal megkapja az üzenetet. A sikeres vagy éppen sikertelen műveletek végrehajtásának visszajelzésére, a fordított irányban történő kommunikáció is az MQTT protokoll segítségével kivitelezhető.

2.4.3. Slack App Commands

A fejlesztői csapat kommunikációs felületnek a Slack alkalmazást[23] választotta. Az alkalmazás különböző platformokon elérhető, több szolgáltatás integrálható a Slack rendszerbe, ezáltal újabb funkciókkal ruházható fel a rendszer, növelve a csapat hatékonyságát. A Slack platform lehetőséget ad saját alkalmazások integrálására is. Az integrációk két irányúak lehetnek: a saját rendszer küldhet üzeneteket a csatornákba, valamint a felhasználók is kiadhatnak parancsokat (Slack App Command). A parancsok a „/” karakterrel kezdődnek és a parancs készítők feladata a parancs defi-

niálása.

Utóbbi lehetőséget kihasználva, a Magic Dashboard saját parancsot integrált a csatornájába `/dashboard <dashboard name>` néven. Ha egy felhasználó az előre meghatározott parancsot kiadja, akkor a Slack rendszer indít egy HTTP POST hívást a beállított URL-re és extra információkat szolgáltat a parancs kiadásának körülményéről, például a felhasználó neve vagy a parancs után beírt üzenet. Ezután már a fogadó fél, jelen esetben a Magic Dashboard szerver alkalmazás, feladata az üzenet feldolgozása és válasz küldése a parancsot kiadott felhasználó számára a Slack rendszeren keresztül. Ilyen típusú Slack alkalmazások közzétehetőek más Slack felhasználók számára is, így ők is integrálhatják a saját csatornáikba.

3. Felhasznált eszközök és alkalmazott módszerek

A következő fejezetekben a projekt fejlesztése közben használt fontosabb felhasznált eszközök és alkalmazott módszerek kerülnek bemutatásra.

3.1. Scrum

A fejlesztés során a csapat egy elterjedt és egyben megbízható agilis módszert használt, a Scrum [25] néven ismert, inkrementális és egyben iteratív szoftverfejlesztési módszert. A szoftver fejlesztése során kettő-három hetes fejlesztési ciklusok (Sprint) voltak beütemezve, amelyet mindig megelőzött egy tervezési fázis (Sprint Planning), ahol mindig megbeszélésre kerültek a következő hetek elvárásai és kitűzött feladatai. Majd a feladatok relatív nehézségi szint alapján Story pontok jellemeztek. Az elvégzésre kitűzött feladatok bekerülnek a Sprint teendő listájába (Sprint Backlog) ahonnan a megfelelő fázisok szerint (Doing, Review, Close) elérnek a kívánt státuszba, így mindig egy újabb funkcionalitás segítette a projekt fejlődését. A Scrum ceremóniák közül minden Sprint végén bemutatóra (Sprint demo) került sor és mindemellett alkalmazva volt a visszatekintés (retrospective) ceremóniája is, amely az elmúlt hetek ellemzésével, nyílt véleménynyilvánítással és az új javaslatokkal elősegítette a fejlesztés lépéseit.

3.2. Folyamatos integráció

Fejlesztési folyamatok közül fontos szerepet kapott a folyamatos integráció (Continuous Integration - CI), amely nem csak a közös, csapatban történő fejlesztést tette hatékonyabbá, hanem maga a projekt komponensei is használják a hozzá fűződő információkat. A fejlesztés során a GitLab CI [26] használatával valósult meg a folyamatos integráció, ahol minden új módosítás feltöltése (push) esetén, automatikus fordítás (build) történt mind a szerver, mind a kliens alkalmazások esetén. A kliens oldalon emellett a CI része a statikus kódelemző (TSLint), forráskód fordítása is, amelyek ha hibát térítnek vissza eredményül, akkor jelzik a fejlesztőknek, hogy az adott verzió nem felel meg minden követelménynek. Ebben az esetben a hátramaradt automatikus feladatok nem is kerülnek végrehajtásra, tehát elbukik a pipeline. A pipeline ezen feladatok összessége, amiket job-

oknak nevezünk. Az automatikus kódelemzések, tesztek, fordítások, minden verziókövető alatt lévő fájl módosulása esetén végrehajtásra kerülnek. Célja, hogy az alkalmazás forráskódja folyamatosan ellenőrizve legyen minden új változtatás esetén, így törekedve arra, hogy az alkalmazás bármikor elérhető és használatra kész állapotban legyen a legutóbbi módosítás után. A GitLab CI lehetőséget add az automatikus konténerizáláshoz szükséges lépések elvégzésére is, a Magic Dashboard projekt esetén minden megcímkézett (tag) állapot esetén létrejön egy Docker [27] image, mind kliens, mind szerver oldalon és ez feltöltésre kerül a központi Docker regiszterbe (registry), amely szintén a GitLab szerver egyik szolgáltatása. Utolsó lépésként a rendszer automatikus módon kitelepítésre kerül egy teszt szerverre.

3.3. Git

A Git verziókövető használatával a projekt változásai követhetően nyomon, amely a csapattagok és a mentorok közös munkáját is segítette. Az újabb funkcionalitások mindig egy új ágra (branch) kerültek, betartva a „Git-flow[28]" pontos szabályait. Minden elkészült új funkcionalítás a mentorok jóváhagyása után bekerült a fő fejlesztési ágba (develop), majd egy Sprint lezárása után a fő (master) ágba.

3.4. Webpack

A kliensoldalon levő forráskód és a szükséges statikus fájlok becsomagolására a Webpack [18] tűnt a megfelelő opciónak. Ez egy modul összecsomagoló (bundler) rendszer, amelyet JavaScript alkalmazások esetében használnak. Segítségével a projektben levő fájlok közti függőségeket (legyen az kép, stílusosztály, JavaScript vagy HTML fájl) egy végső csomagba tömöríthetjük, ezzel megoldva a függőségek esetleges duplikálását. A Webpack által generált végső statikus fájlok a böngészők számára ugyanúgy értelmezhetőek lesznek, azonban a generált kód tipikusan kisebb lesz.

Napjainkban egyre inkább elterjedtek a mikroszolgáltatások, amelyek célja a nagy alkalmazások egyre kisebb részekre bontása, így megkönnyítve a piaci elvárásokhoz való rugalmas változtatásokat. A mikroszolgáltatások lehetőséget ad a hatékony skálázhatóságra és nem utolsó sorban a

platform-függetlenség előnyeit is könnyebben elérhetővé teszi.

3.5. Docker

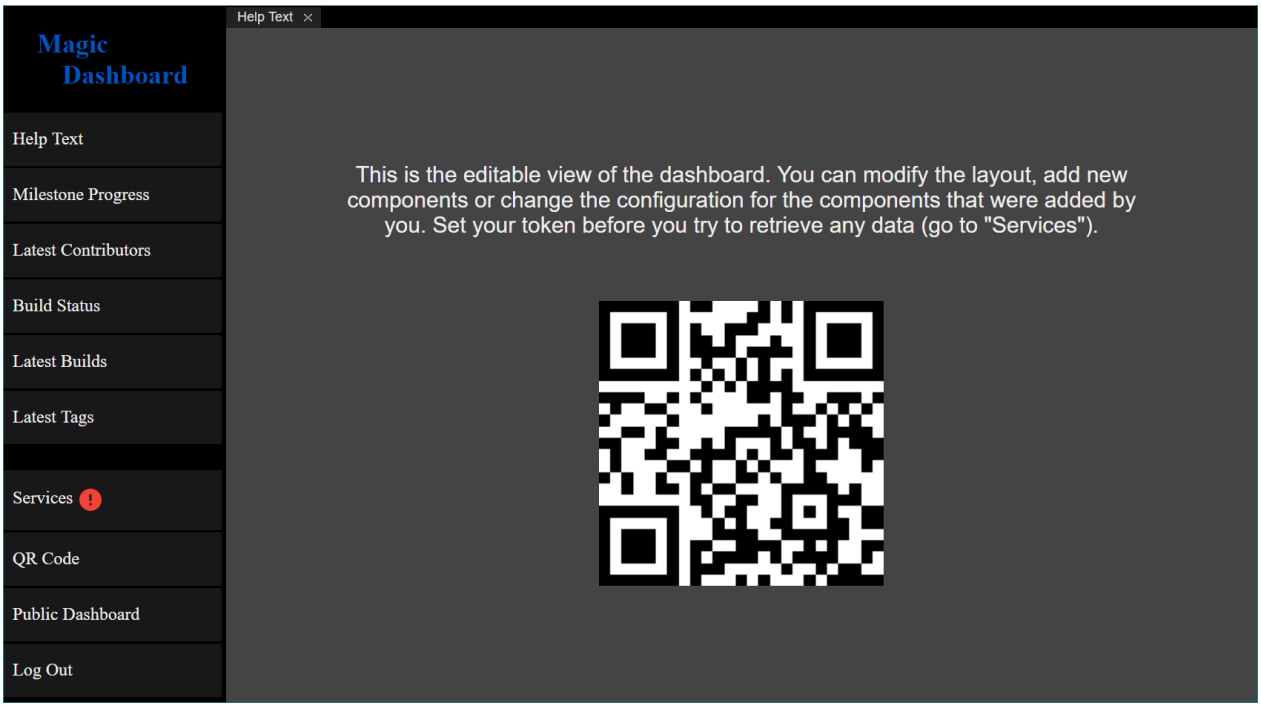
A Magic Dashboard projekt esetében három mikroszolgáltatást különböztethetünk meg: a Golang szervert, a React statikus webszervert és a RethinkDB adatbázist. Ezen szolgáltatások konténerizálására a Docker[27] platform adott lehetőséget, ahol a kisebb image-ek instanciájának futtatása során, elkülönített konténerek jönnek létre, amelyek platformfüggetlen futtatásra kész állapotba helyezik a mikroszolgáltatásokat. Annak érdekében, hogy a rendszer kevés parancs futtatásval használható állapotba kerüljön és megfelelően bekonfigurált konténerek jöjjenek létre, a Docker Compose[29] eszköz használható. A Docker-nek és a Docker Compose-nak köszönhetően egyetlen parancs végrehajtásával a teljes Magic Mirror alkalmazás elindul egy platform-független környezetben.

4. A Magic Dashboard működése

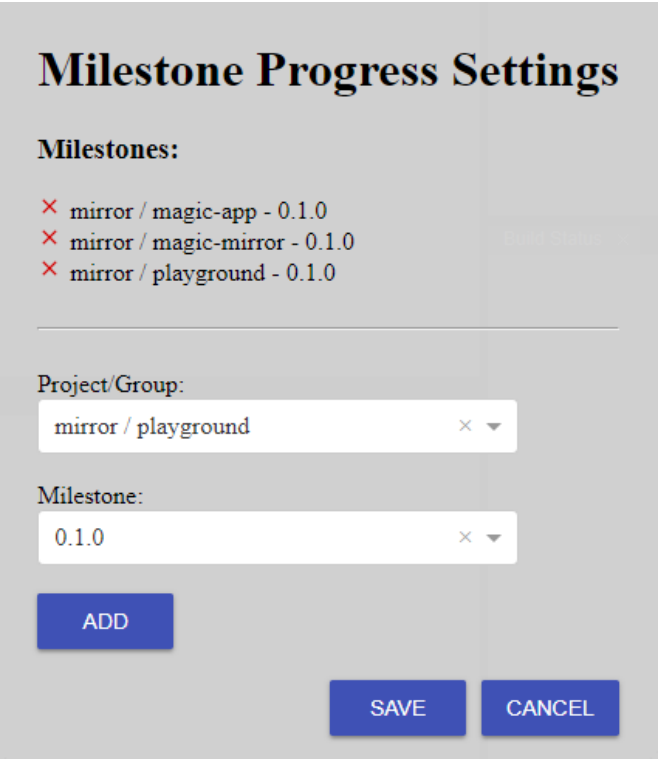
A következő fejezetek a Magic Dashboard szoftverrendszer webes felületének és mobil alkalmazásának működését mutatják be a felhasználók szempontjából.

4.1. A webes felhasználói felület használata

A rendszer webes felületének első alkalommal való megnyitásakor a felhasználó elé két lehetőség tárul: a "Public Dashboard" gombra kattintva megtekintheti a mindenki számára elérhető publikus műszerfalat, illetve bejelentkezhet az alkalmazásba a "Login" gombra kattintva. Bejelentkezés során a böngésző átirányul a GitLab weboldalára. Itt a felhasználónak jóvá kell hagynia, hogy az alkalmazás a felhasználó jogaival használhatja a Gitlab API-t. Ha ez megtörtént, a Magic Dashboard rendszerbe való bejelentkezés sikeres lesz, és a böngésző visszairányul az előbb említett felületre, ahol ezúttal a "Login" gomb helyett az "Edit Dashboard" gomb jelenik meg. Ezt az opciót választva a felhasználó egy másik felületen kiválaszthatja, hogy a mindenki számára publikus vagy a privát műszerfalát akarja szerkeszteni. Innen a kívánt opciót választva beléphet az adott műszerfal szerkesztői felületére.



8. ábra. A privát műszerfal szerkesztői felülete első bejelentkezés után

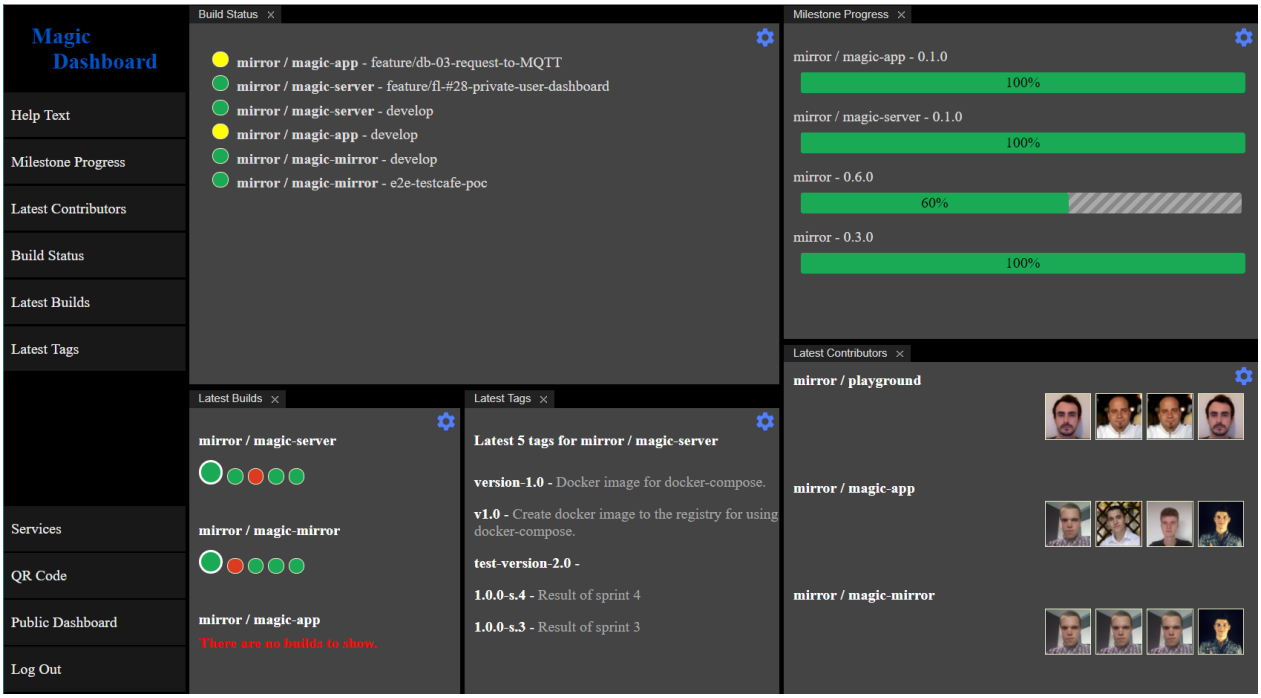


9. ábra. A Milestone Progress komponens beállítási nézete

Első alkalommal a szerkesztői felületre navigálva a 8. ábrán látható nézet fogadja a felhasználót, egyetlen komponenssel, amely rövid szöveg formájában útbaigazítást ad és tartalmazza a műszerfala azonosítójának QR kódját.

A szerkesztői felület bal oldala a menünek ad helyet. A menü felső részén találhatóak a műszerfalra húzható komponensek listája, ezek alatt pedig rendre a „Services”, „QR Code”, „Public Dashboard” és „Logout” gombok. A Service gomb segítségével a felhasználó a GitLab felhasználójához tartozó hozzáférési azonosítót (Access Token) állíthatja be, enélkül a komponensek nem tudnak adatokat lekérni a GitLab rendszertől. A felület értesít egy piros felkiáltójellel a Services felirat után, ha ezt az azonosítót a felhasználó még nem állította be. A QR Code gombra kattintva egy ablakban a felhasználó privát műszerfala azonosítójának QR kódja jelenik meg. A Public Dashboard gomb használatával a felhasználó átnavigálhat a publikus műszerfal nézetre, illetve a Logout opciót választva kijelentkezhet az alkalmazásból.

A menüben levő komponensek műszerfalra helyezése a fogd meg és húzd (drag-and-drop) módszerrel történik. A műszerfalon levő komponenseket a felhasználó tetszése szerint méretezheti és helyezheti át, ugyanakkor bármikor bezárhatja ezeket. Egy komponens jobb felső sarkában találha-



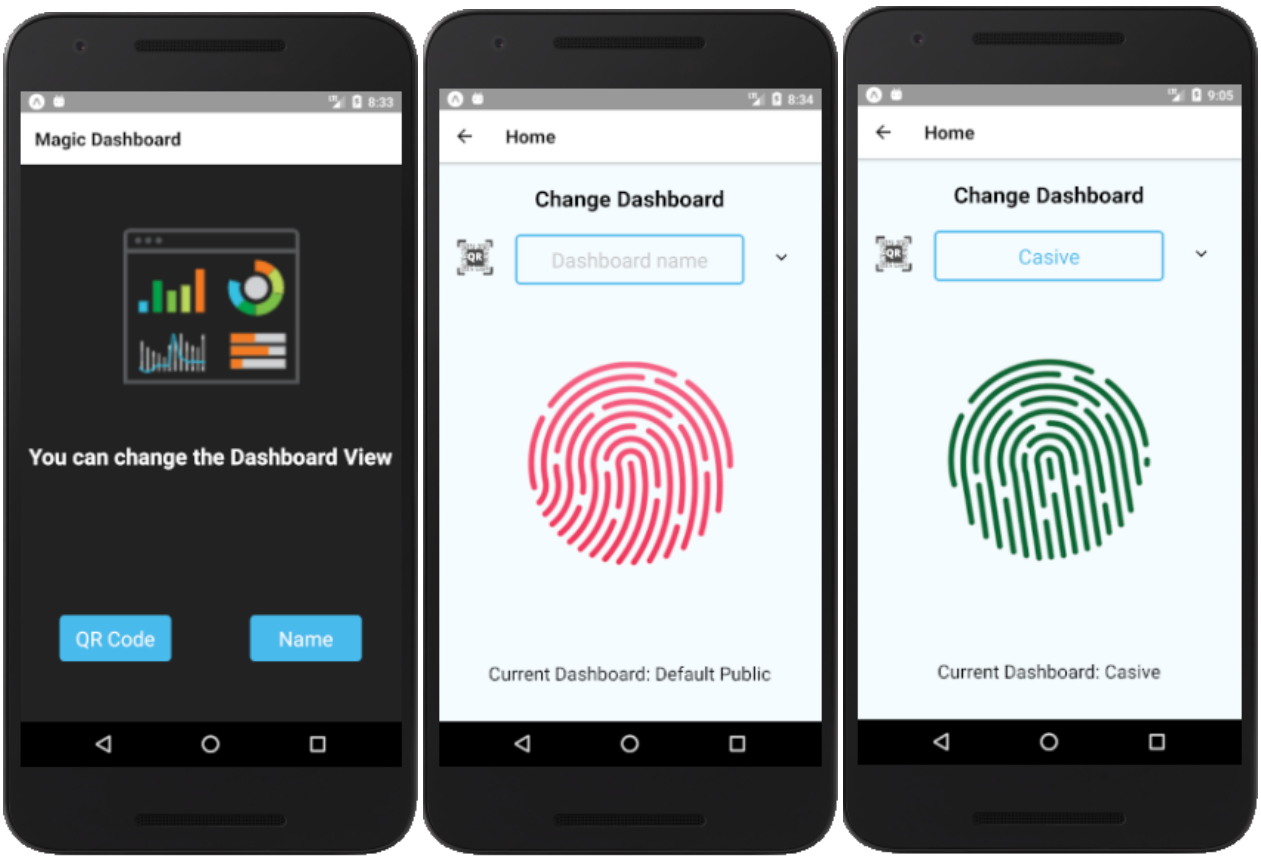
10. ábra. Példa egy teljesen konfigurált műszerfalra

tó fogaskerék ikonra kattintva egy felugró ablakban megjelennek az adott komponensre vonatkozó beállítások (9. ábra). Ezeken beállítható, hogy az adott típusú komponens mely adatokat jelenítse meg. Miután a felhasználó behúzta a műszerfalra a számára fontos komponenseket és ezeket sikeresen konfigurálta, a 10. ábrához hasonló műszerfal állhat a rendelkezésére.

Ha a felhasználó úgy dönt, hogy meg szeretné tekinteni a jelenleg aktuális publikus műszerfalat, megteheti a menüben elhelyezett Public Dashboard gomb megnyomásával. Ezen a felületen nem érhető el a menürendszer, valamint semmilyen interakció nem lehetséges a műszerfallal.

4.2. A mobil kliens használata

A Magic Dashboard rendszer mobil alkalmazásának (12. ábra) fő feladata, hogy a felhasználó a privát műszerfalát távolról, a webes felülettel való interakció nélkül tudja publikussá tenni. Hogy ezt megtehesse, az alkalmazás először tudomást kell szerezzen a műszerfal azonosítójáról. Erre először az alkalmazás kezdő oldalán van lehetőség: a felhasználó eldöntheti, hogy a privát műszerfalának webes felületén található QR kódot szkenneli be a „QR Code” gombra kattintva, vagy a "Name" gombot választva beírja a műszerfala nevét. Miután az alkalmazásba bekerült a kívánt azo-



12. ábra. A mobil applikáció felhasználói felülete

nosító, a publikáló felületen lehetőség van a műszerfal 2 percig való publikussá tételére a kijelző közepén elhelyezkedő "Publish" gomb megnyomásával. Amennyiben a felhasználó nem szeretné kivárni a 2 perc elteltét, a "Publish" gomb ismételt megnyomásával visszavonhatja műszerfalát a publikus nézetről.

Az alkalmazás lehetőséget ad több műszerfal azonosítójának tárolására is. Ezeket az azonosítókat a publikáló nézeten lehet beolvasni, ahol ugyancsak lehetőség van QR kód szkennelésre vagy szöveges bevitelre. Hogy a "Publish" gomb melyik műszerfalat fogja publikussá tenni, az egy lenyíló listából kiválasztható. Az aktuálisan beállított műszerfal neve a publikáló gomb alatt látható, a "Current Dashboard" felirat után.

Következtetések és továbbfejlesztési lehetőségek

A Magic Dashboard projekt keretén belül sikerült egy olyan szoftverrendszer elkészítése, amelyet használva a szoftverfejlesztők és projekt menedzserek egy helyen, valós időben tudják a projektreikre vonatkozó fontosabb információkat nyomon követni. A webes felület mellett sikerült egy mobil kliensalkalmazást is létrehozni, amely segítségével a felhasználók távolról is publikussá tehetik a saját műszerfalukat.

A fejlesztési folyamat során több ötlet is felmerült, amivel a jövőben ki lehet bővíteni az alkalmazás funkcionalitásait, ezek a következők:

- más projektmenedzsment rendszerek és fejlesztői eszközök (JIRA, GitHub, stb.) integrálása
- felhasználónként egynél több privát dashboard támagotása
- új komponensek létrehozása a támogatott fejlesztői eszközökből érkező adatok megjelenítésére
- a már meglévő komponensek továbbfejlesztése (például egy GitLab-os projekt esetében a legutóbbi hozzájárulókat megjelenítő komponens a hozzájárulás típusát is mutassa)
- jelenleg a webes kliens a long polling mechanizmust használva tölti be az adatokat, de továbbfejlesztésként lehetne esemény orientált megoldással frissíteni az adatokat
- IoT (Internet of Things) eszközök integrálása az alkalmazásba, mint például LED-sorok (amelyek villognak, ha egy komponensen valamilyen változás történt) vagy fizikai „doButton” (egy fizikai gomb, aminek megnyomására a publikus dashboard egy adott időre átvált a felhasználó privát dashboardjára)
- virtuális személy asszintensen (Amazon Alexa, Google Home, Microsoft Cortana, Samsung Bixby) keresztül történő vezérlés implementálása

Hivatkozások

[1] Valarm hivatalos weboldal. Monitor anything, anywhere `http://www.valarm.net` [Utolsó megtekintési dátum: 2018.04.20.]

[2] MagicMirror[2] hivatalos weboldal. The open source modular smart mirror platform `https://magicmirror.builders/` [Utolsó megtekintési dátum: 2018.04.20.]

[3] GitLab hivatalos weboldal. A single application for the complete DevOps lifecycle `https://about.gitlab.com/` [Utolsó megtekintési dátum: 2018.02.19.]

[4] GitLab hivatalos dokumentációs weboldal. *GitLab as an OAuth2 provider.* `https://docs.gitlab.com/ce/api/oauth2.html` [Utolsó megtekintési dátum: 2018.02.19.]

[5] Google QR kód hivatalos weboldal. *What is a QR Code?* `https://www.the-qrcode-generator.com/whats-a-qr-code` [Utolsó megtekintési dátum: 2018.03.27.]

[6] Slack Command hivatalos weboldal. *Slash Commands* `https://api.slack.com/slash-commands` [Utolsó megtekintési dátum: 2018.04.07.]

[7] MQTT protokoll. *MQTT Essentials Wrap-Up* `https://www.hivemq.com/blog/mqtt-essentials-wrap-up` [Utolsó megtekintési dátum: 2018.04.05.]

[8] RethinkDB hivatalos weboldal. *The open-source database for the realtime web* `https://www.rethinkdb.com` [Utolsó megtekintési dátum: 2018.04.18.]

[9] Golang hivatalos dokumentációs weboldal. *The Go Programming Language* `https://golang.org/doc` [Utolsó megtekintési dátum: 2018.04.08.]

[10] GoRethink hivatalos dokumentációs weboldal. *GoRethink* `https://godoc.org/github.com/GoRethink/gorethink` [Utolsó megtekintési dátum: 2018.04.23.]

[11] Tit Petric. *Managing configuration with Viper* https://scene-si.org/2017/04/20/managing-configuration-with-viper/ [Utolsó megtekintési dátum: 2018.04.18.]

[12] Golang web keretrendszerek teljesítmény mérése. *Package httprouter* https://github.com/gin-gonic/gin/blob/master/BENCHMARKS.md [Utolsó megtekintési dátum: 2018.04.10.]

[13] Golang hivatalos dokumentációs weboldal. *Benchmark system* https://godoc.org/github.com/julienschmidt/httprouter [Utolsó megtekintési dátum: 2018.04.10.]

[14] ReactJS hivatalos weboldal. *ReactJS - A JavaScript library for building user interfaces* https://reactjs.org [Utolsó megtekintési dátum: 2018.04.12.]

[15] JSX hivatalos dokumentációs weboldal. *JSX Documentation* https://jsx.github.io/doc.html/ [Utolsó megtekintési dátum: 2018.04.12.]

[16] Golden Layout hivatalos dokumentációs weboldal. *Golden Layout Documentation* http://golden-layout.com/docs/ [Utolsó megtekintési dátum: 2018.04.12.]

[17] TypeScript hivatalos dokumentációs weboldal. *TypeScript Documentation* https://www.typescriptlang.org/docs/home.html/ [Utolsó megtekintési dátum: 2018.04.12.]

[18] Wabpack hivatalos dokumentációs weboldal. *Webpack Documentation* https://webpack.js.org/concepts/ [Utolsó megtekintési dátum: 2018.04.12.]

[19] Nikolai Ershov. *Radix Trees* https://kukuruku.co/post/radix-trees/ [Utolsó megtekintési dátum: 2018.04.10.]

[20] ECMA hivatalos dokumentácós weboldal. *ECMAScript® 2015 Language Specification* http://www.ecma-international.org/ecma-262/6.0/ [Utolsó megtekintési dátum: 2018.04.11.]

[21] Expo hivatalos dokumentácós weboldal. *Introduction* https://docs.expo.io/versions/latest/ [Utolsó megtekintési dátum: 2018.04.12.]

[22] Mitchell Anicas. *An Introduction to OAuth 2* https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2 [Utolsó megtekintési dátum: 2018.03.19.]

[23] Slack hivatalos weboldal. *It's the foundation for teamwork* https://slack.com/features [Utolsó megtekíntési dátum: 2018.04.18.]

[24] Slack App hivatalos weboldal. *Building Slack apps* https://api.slack.com/slack-apps [Utolsó megtekíntési dátum: 2018.04.06.]

[25] Agile Coach. *Scrum* https://www.atlassian.com/agile/scrum [Utolsó megtekintési dátum: 2018.04.07.]

[26] GitLab CI/CD hivatalos weboldal. GitLab Continuous Integration and Deployment https://about.gitlab.com/features/gitlab-ci-cd/ [Utolsó megtekintési dátum: 2018.04.18.]

[27] Docker hivatalos dokumentációs weboldal. *Docker concepts* https://docs.docker.com/get-started/ [Utolsó megtekintési dátum: 2018.03.01.]

[28] Daniel Kummer. *Git-flow cheatsheet* https://danielkummer.github.io/git-flow-cheatsheet/ [Utolsó megtekintési dátum: 2018.04.18.]

[29] Docker Compose hivatalos dokumentációs weboldal. *Docker Compose* https://docs.docker.com/compose/ [Utolsó megtekintési dátum: 2018.04.18.]