ETDK-dolgozat

Bíró Dániel

Nagy Kristóf-Bálint

XXVI. reál- és humántudományi Erdélyi Tudományos Diákköri Konferencia (ETDK)

Informatika II.: innovatív számítástechnikai termékek, alkalmazások szekció

Kolozsvár, 2023. május 18–21.

ThermoSmart

Otthoni termosztát és az őt vezérlő szoftverrendszer

Szerzők:

Bíró Dániel

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

Nagy Kristóf-Bálint

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

Témavezetők:

dr. Sulyok Csaba, egyetemi adjunktus

Babeş–Bolyai Tudományegyetem, Matematika és Informatika Kar

Gagyi Mátyás, szoftverfejlesztő,

Codespring

Nagy-Seres Norbert, szoftverfejlesztő,

Codespring

Zsebe Ákos, szoftverfejlesztő,

Codespring

CODESPRING

OUR CODE YOUR SUCCESS

Kivonat

Az átlagember számára problémát jelenthet, hogy egy hideg téli nap után egy kihűlt otthon fogadja. A hagyományos termosztátok csak részleges megoldást jelentenek a problémára, hiszen ha nem az előre beállított időben érünk haza, akkor így is hideg fogadhat otthon, vagy akár kárba mehet a melegítéshez szükséges energia.

A ThermoSmart projekt célja, hogy lehetővé tegye a felhasználók számára, hogy bárhol, bármikor befolyásolhassák otthonuk hőmérsékletét. A rendszer egy szerveren alapszik, amely adatokat fogad a hőmérséklettel kapcsolatban egy Raspberry Pi 4 alapú termosztáttól, amelyet egy mobil applikációval irányíthatunk.

A rendszer az egyszerűség elvét követi. Továbbá kényelmi funkciókat biztosít, például az időjárás-előrejelzést, amely hozzásegíti a felhasználót ahhoz, hogy optimális energiafelhasználást valósíthasson meg. A termosztát két darab hőmérővel operál, amelyekből az egyiket a házon belülre, a másikat pedig a házon kívülre kell elhelyezni. Az általuk generált adatokból statisztikák jönnek létre, amelyek célja szintén az, hogy támpontot nyújtsanak a házát optimálisan fűteni akaró felhasználónak.

A dolgozat bemutatja a rendszer részletes működését, a használt technológiákat és részletezi a funkcionalitásokat.

Tartalomjegyzék

Bevezető

Rendkívül kellemetlen, ha valaki a lakóhelyére érkezve a beltéri környezetet hidegnek találja. Egy opcionális megoldás az, hogy ha beállítjuk az otthoni termosztátunkon, hogy délután hat órakor kapcsoljon be a fűtés, és melegítsen addig, ameddig 21-22 Celsius-fok nem lesz. Ez egy megfelelő módszer, amennyiben biztosan tudjuk, hogy körülbelül hat órakor érünk haza. Viszont, ez nem mindig fog így történni.

A hagyományos termosztátok, amelyek képesek eltárolni, hogy a hét melyik napján hány órakor milyen hőfokot szeretnénk, majd ezt fent is tartani a kazán beindításával, 1950 óta vannak otthoni használatban [31]. Mindig is problémát jelentett az, hogy ha elutazunk, és elfelejtjük kikapcsolni a fűtést, akkor a termosztát a megadott időben fenntartja a beállított hőmérsékletet, de mivel nem vagyunk otthon az energia kárba vész.

Továbbá, ezek az egyszerű termosztátok nem veszik figyelembe a házon kívüli hőmérsékletet. A kazánok általában előre megadott hőmérsékletre melegítik fel, és keringetik a vizet a fűtőtestekben, viszont abban az esetben, ha például odakint 18 fok van, és azt szeretnénk, hogy bent 21 legyen, nem szükséges felmelegíteni a vizet egészen 60 fokig, elég lenne csak feléig. Továbbfejlesztési lehetőségként felhasználhatjuk a külső hőmérőt, hogy automatikusan szabályozzuk a kazán hőfokát.

A *US. Energy Information Administration* az írja, hogy az amerikai otthonok 85 százalékában van központi fűtés, amelyből kevesebb mint felében használnak programozható termosztátot [16]. Másodsorban, *Chris Mooney* az *Americans could save a fortune this winter — if they only understood their thermostats* [39] című cikkében arról ír, hogy a termosztátok használata nehézkes és összezavaró, referenciaként megemlíti *Peffer et al.* cikkét [6] ezzel kapcsolatban. *Mooney* arról is beszél, hogy az otthonainkban lévő termosztátok régiek, és nem mi választottuk, hanem valószínűleg örököltük őket, így nem nyerik el a tetszésünket sem a felhasználói felület, sem funkcionalitás szempontjából.

A piacon található pár megoldás, viszont mindegyiknek van valamilyen hiányossága. A *POER Smart Controls* cég termosztátja [40] például nem rendelkezik külső hőmérővel, így értékes információt veszít, továbbá nem nyújt időjárás-előrejelzést sem. A *Biticino* cég terszmosztátja [52] pedig az eddig felsorolt problémák tetejébe nem nyújt megfelelő felahsználói élményt. A ThermoSmart projekt célja ezeknek és a fennebb említett problémáknak a megoldása egy letisztult, egyszerű és funkcionális mobilapplikáció által.

A szoftverrendszer három részből áll: egy mobilalkalmazásból, amely kezelőfelületet

biztosít a felhasználónak, egy szerverből és egy Raspberry Pi 4-ből [44], amely betölti a termosztát szerepét, monitorizálja a kinti-benti hőmérsékletet, és ki-be kapcsolja a fűtést.

A projekt 2022 júniusában indult el, mint nyári gyakorlatos feladat a Codespring[1] kolozsvári irodájában. Ezúton szeretnénk kifejezni köszönetünket Dr. Sulyok Csabának a koordinálásért és a rengeteg technikai támogatásért, Gagyi Mátyásnak a csapat irányításáért, legfőképpen pedig Nagy-Seres Norbertnek és Zsebe Ákosnak, akik egy teljes nyáron keresztül vezettek és tanítottak minket. Továbbá, köszönet illeti diáktársainkat, akik a csoportos projekt nevű tantárgy alatt segítettek a projekt fejlesztésében: Borok Szilvia, Fodor Tímea, Szőke András-Loránd és Vitus Balázs.

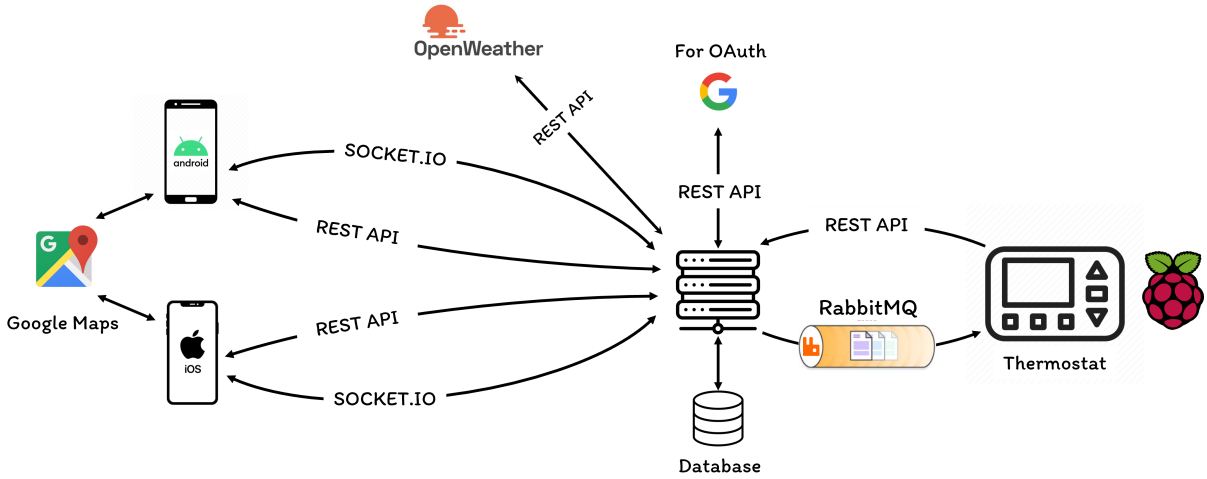
[1] https://edu.codespring.ro

1. Architektúra

A projekt célja, hogy a lehető legtöbb pénzt spóroljunk meg, amikor fűtésről van szó. Ezt és a kényelmes felhasználói élményt három architekturális elem bevezetésével találtuk megfelelőnek: mobilalkalmazás, szerver és Raspberry Pi mint termosztát. A központi elem a szerver, amely összeköti a felhasználó által használt mobilalkalmazást az otthoni termosztáttal.

A felhasználók új fiókot egyedül a Google-fiókjuk segítségével hozhatnak létre, ezzel megkönnyítve a regisztráció folyamatát, ezt követően legalább egy termosztátot hozzá kell rendeljenek önmagukhoz. Egy termosztát összekötése egy felhasználóval (vagy felhasználókkal) QR-kód[2] segítségével történik. Az otthoni egység el van látva egy QR-kóddal, amely egyéni azonosítóként szerepel a rendszerben. A felhasználó a mobilalkalmazáson keresztül ennek a QR-kódnak a beolvasásával kap hozzáférést a termosztáthoz, és annak minden funkcionalitásához. A felhasználók és a termosztátok több-a-többhöz kapcsolatban állnak, azaz egy felhasználó több termosztátot is kezelhet, és egy termosztát is kezelhető több felhasználó által. Ezt a több-a-többhöz kapcsolatot a szerver tartja karban.

Az 1. ábrán bal oldalán látható a két céleszköz, az Android és iOS operációs rendszert futtató telefonok, amelyek REST [28] és WebSocket [55] alapú csatornákon kommunikálnak a szerverrel. A két különböző csatorna jelenléte annak köszönhető, hogy célul tűztük ki, hogy valós idejű adatokat mutassunk a felhasználóknak. A REST kapcsolat megfelelő a mobil által a szerver felé küldött parancsok továbbítására, viszont ellenkező irányban nem, mert a telefonnak



1. ábra. A rendszer architektúrája - Felülnézet.

[2]Quick Response kód, két dimenziós pontkód. A gyors visszafejthetősége miatt alkalmaztuk.

nincsen publikus IP címe[3]. Ez azért nem optimális, mert REST kapcsolatot használva, a mobilapplikáció csak akkor értesülhet bármilyen hőmérsékletbeli változásról, ha erre explicit rákérdez, amely folyamatos és nagyrészt felesleges HTTP [30] kéréseket jelent. Ennek a problémának a megoldására WebSocketeket használtunk, amelyek megvalósítják a kétirányú és késlekedés nélküli kommunikációt (a felhasznált könyvtár a Socket.IO [53]).

A mobilapplikáció időjárás-előrejelzéssel is el van látva, hogy a felhasználó könnyebben dönthessen arról, hogy mikor érdemes bekapcsolnia a fűtést. Például, ha az éjszaka folyamán 15 fok lesz, és így lehűl a ház, de reggelre 22 lesz, akkor nem érdemes melegíteni. Ha a felhasználó ezt tudja, akkor kikapcsolhatja a fűtést az adott reggelre. Hogy pontos legyen az előrejelzés, elengedhetetlen, hogy tudjuk, hogy hol található a termosztát, így került integrálásra a Google Maps [26] is, amely lehetőséget nyújt arra, hogy meghatározzuk a termosztátunk helyzetét.

Középen a szerver található, mely egy PostgreSQL [41] adatbázist kezel. A szerver kapcsolatban áll az OpenWeatherMap API [58]-val, az előbb említett okokból, és a Google OAuth [27] szolgáltatásával, amelyet azért használtunk, hogy Google-féle bejelentkezést valósíthassunk meg.

Jobb oldalon a Raspberry Pi 4 van, amely REST és AMQP [7] protokollok segítségével kommunikál a szerverrel. Két csatorna van, hasonló okból, mint a szerver és a mobil között. Funkció szempontjából a különbség, hogy a termosztát nem a jelenlegi hőmérséklet-információt fogadja, hanem a célhőmérsékletet. A második kapcsolat ismét egy oda-vissza kommunikálásra képes protokollra épül (AMQP), viszont a mobilokkal ellentétben a Raspberry Pi-on egy teljes Linux operációs rendszer fut [45], amely lehetővé teszi a bonyolultabb technológiák használatát is. Mivel azonnali kommunikációt biztosító és nagy teljesítőképességgel bíró technológiát kerestünk, a választás a *message broker*-ekre esett, ezen belül az *AMQP* protokollra. Ezek két alkalmazás között biztosítanak kommunikációt: a küldő fél (esetünkben a szerver) elküldi az üzenetet a *message broker*nek, a fogadó fél (termosztát) pedig feliratkozik a küldő fél által megadott témára, így megkapva az üzenetet. A *CloudAMQP* [2] szolgáltatón keresztül, a *RabbitMQ* [42] protokollal oldottuk meg a két elem közti kommunikációt.

Úgy a szerver, mint a mobilalkalmazás és a termosztát programozásához a TypeScript [57] nyelvet használtuk. A TypeScript a Microsoft által fejlesztett programozási nyelv, a JavaScript [32] egy szuperszetje, ahol statikus típusokkal láthatjuk el az alapból dinamikusan típusos kódot. Objektumorientált kódot is lehet benne írni, ezáltal lehetőséget biztosít osztályok,

[3]Olyan azonosító, amellyel az eszköz megtalálható az interneten.

interfészek és típusok használatára. Nagyméretű alkalmazásoknál ajánlott a használata, mivel a kötöttebb stílusával kevésbé engedi meg az esetleges hibás kód írását. JavaScript-re transzpilálódik, hogy majd a böngészőben futtatható legyen.

1.1. Szerver

A projekt központi eleme a szerver, amely mobilalkalmazás és a termosztát közötti kommunikáció kialakításáért és fenntartásáért felelős. A szerver képes a beérkező kérések fogadására és feldolgozására, majd válasz visszaküldésére a kérő félnek. Ez a komponens közvetítő szerepet tölt be a mobilalkalmazás és a termosztát között mindkét irányban, emellett felel az adatbázis-hozzáférésért és ennek kezeléséért is.

1.1.1. Technológiák

A szerver *TypeScript* [57] nyelven készült, az *Express.js* [11] keretrendszert használva. A kód futtatásához a *Node.js* [3] futtatási környezetet használjuk.

A Node.js egy JavaScript futtatási környezet, amely a Google V8 motorján alapszik, és lehetővé teszi, hogy a kódot egy web böngészőn kívül futtassuk. Egy szálon fut, viszont aszinkron és szinkron programozásra is alkalmas. Híres a skálázhatóságáról, illetve képes nagy méretű adatok feldolgozására is. Népszerű webalkalmazások, webszerverek és parancssorbeli eszközök létrehozására. A beépített csomagkezelőjével, az *npm*el könnyen hozzáadhatunk külső csomagokat a projektünkhöz.

Külső könyvtárakat is használunk a projektben, a fontosabb elemek közé tartozik az *amqplib*, az *AMQP* protokollt behozó könyvtár, a *Winston* és a *Morgan* loggerek, a *Passport.js* bejelentkezést segítő middleware, amely előre megírt stratégiákat ajánl fel annak érdekében, hogy könnyebb legyen a felhasználó bejelentkezése, az *Axios* [9] HTTP kliens, továbbá a *nodemon* kódelemző, amely minden kódváltoztatásnál újraindítja a szervert, elősegítve a gyorsabb fejlesztést. A kód helyességének ellenőrzésére az *ESLint* [10] statikus kódelemzőt alkalmazzuk.

A termosztáttal való kommunikálásra egy, a CloudAMQP szolgáltató által biztosított RabbitMQ klasztert használtunk. A RabbitMQ egy nyílt forráskódú message broker, amely eredetileg az AMQP (Advanced Message Queuing Protocol) protokollt implementálta, majd később kiterjeszkedett több más, ismert protokollra is. Ez egy szoftver, amihez alkalmazások kapcsolódhatnak az üzenetküldés érdekében; a közvetítő szerepét játssza a két fél között. A

küldő fél feladja az üzenetét egy előre megbeszélt csatornára, a fogadó fél pedig feliratkozik ugyanoda, és konstans figyeli a változásokat. Az üzenetek egy sorban (queue) érkeznek meg, ezek típusa bármilyen lehet. Ha a fogadó fél megkapta az üzenetet, egy igazoló jelet küld vissza a szervernek. Ez értesíti a küldő felet arról, hogy az üzenete sikeresen célbaért.

A struktúra a többrétegű architektúrára [22] épül. Az objektumok *DTO*-ként [14] közlekednek a szerver és a mobil között mindkét irányban. A DTO a Data Transfer Object rövidítése; egy olyan objektum, amelyet adatok enkapszulására használunk.

A projekt során egy PostgreSQL [41] adatbázisba mentettük el a perzisztens adatainkat. Ez egy relációs adatbázis kezelő-rendszer, amelyik robosztus és biztonságos adatrendszerezést és mentést biztosít. Igény szerint skálázható, és nagy adathalmazokat is képes kezelni. Gyakori választás az enterprise applikációk fejlesztésekor.

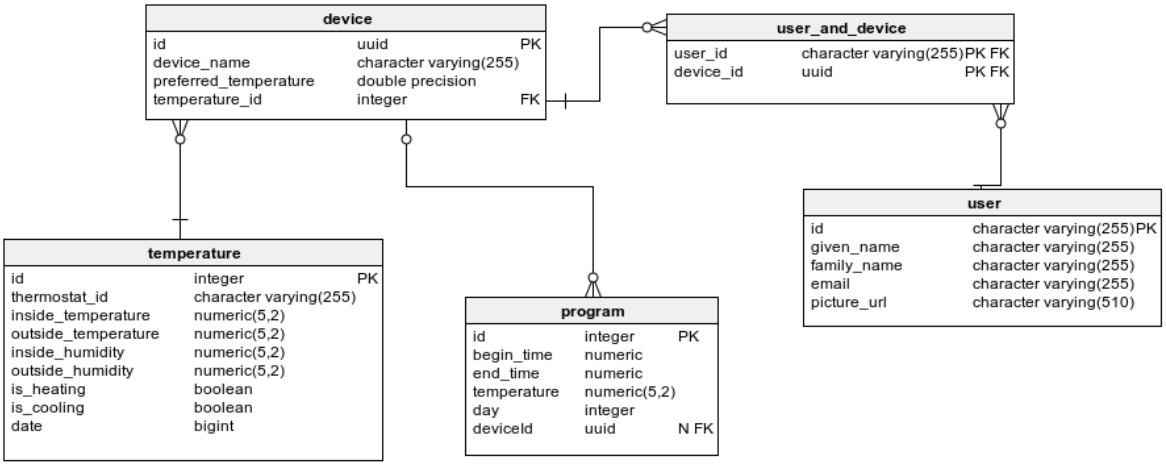
Az adatbázis és a szerver közti kapcsolatot a TypeORM-el [56] oldjuk meg, amely egy TypeScript implementációja az ORM rendszereknek. Az ORM [5] (Object Relational Mapping) technológia az objektumorientált programozási nyelvek és a relációs adatbázisok között képes kapcsolatot teremteni olyan módon, hogy az egyiket képes a másikra leképezni: modellosztályokból automatikusan generál táblákat a nekik megfelelő kapcsolatokkal együtt, továbbá optimálisan megírt függvényeket biztosít annak érdekében, hogy a négy *CRUD* (Create, Read, Update, Delete) alapművelet könnyen elvégezhető legyen. Ezenfelül kínál egy *Query Builder* nevű eszközt, amellyel *SQL* lekérdezéseket tudunk építeni *TypeScript*-ben. Az SQL injection [54] típusú támadások ellen is hatékonyan védekezik, mivel levédi a speciális karaktereket.

1.1.2. Komponensek

A szerver fejlesztéséhez használt csomagokat és könyvtárakat az *npm* [12] csomagkezelővel adjuk hozzá a projektünkhöz. A kód három részre van osztva:

- *api* csomag: Az *api* csomag 3 részből áll. A *dto* csomag a bejövő és kimenő DTO-kat definiálja, a *middleware* csomag a felhasználó bejelentkezettségét ellenőrző, illetve a testreszabott hibakezelést kínáló *middleware*-eket [38] (olyan függvények, amelyek lefutnak, mielőtt a szerver egy bizonyos végpontot elérne) tartalmazza, a *router* csomag pedig az előre definiált végpontok elérését lekezelő függvényeket foglalja magába.

- *domain* csomag: Az *domain* csomag ugyancsak 3 alcsomagot foglal egybe. A mapper csomagban található mapper átkonvertál egy osztályt egy másik típusú osztályba, annak



2. ábra. Az adatbázis diagramja

elemeit egyenként megfeleltetve a célosztály elemeivel; a *model* csomagban találhatóak a célosztályok, amikbe a mapper konvertál, ezek lesznek továbbküldve a legfelső rétegbe, az *api* csomagba. A *service* csomagba érkeznek a kérések az *api* csomagtól és itt található a kód, amely a megfelelő választ generálja ki.

- *persistence* csomag: A *db* csomagban található a kód, amelyik létesíti a kapcsolatot az adatbázissal, illetve a *cache*-t [15] inicializálja. Az *entity* csomagban vannak definiálva az *ORM* által használt entitások, továbbá a *repository* csomag tartalmazza az adatbázissal kommunikáló függvényeket.

1.1.3. Adatmodell

A ThermoSmart projekt adatbázisának felépítése a következő:

- A *device* tábla a regisztrált termosztátok adatait tartalmazza: a gyártáskor kapott azonosítójukat, az eszköz nevét, illetve a felhasználó által beállított hőmérsékletet.
- A *temperature* tábla a termosztát által mért hőmérsékleteket tárolja. A termosztát egyedi azonosítója, valamint szenzorok által mért külső és belső hőmérséklet és páratartalom, mellett megtaláljuk azt is, hogy épp fűt, vagy épp hűt-e az eszközünk, majd az időbélyeget is.
- A *user* tábla a felhasználóról tárol el adatokat a Google profiluk alapján. Ebből a család-, illetve keresztnevet, az email-t, továbbá a felhasználó profilképének URL-jét nyeri ki.

- A *program* tábla a felhasználó által beállított, intervallumokra lebontott hőmérséklet-preferenciát tartalmazza a hét minden napjára. Tartalmazza a kezdési- és végződési időpontot, a beállított hőmérsékletet, valamint a hét napjainak egyikét.
- Az *user_and_device* tábla azt tárolja, hogy melyik termosztát melyik felhasználóhoz van hozzákapcsolva, ezt a két entitás azonosítójának elmentésével valósítja meg.

1.1.4. API

Az *API*-t [8] (Application Programming Interface) két külön applikáció közti kommunikációra használatos, egy könnyen elérhető módszer adatok küldésére és fogadására. A szerver egy ilyen, a *REpresentational State Transfer* - *REST*-en alapuló *API*-t biztosít számunkra, amely *HTTP* kérésekkel érhető el. Ezek a kérések *JSON* [34] formátumú adatokat tartalmaznak.

A *router* csomagban található összes függvény egy adott típusú *HTTP* metódust tud fogadni, ezek a következőek lehetnek: *GET*, *POST*, *PUT*, *DELETE*.

A felsorolt metódusokon kívül kötelezően meg kell határozni egy útvonalat minden függvénynek, illetve opcionálisan ki lehet őket egészíteni *query* paraméterekkel is; az eszközspecifikus függvényeknek mind szükségük van egy *deviceId* nevű query paraméterre, mely alapján azonosítják az eszözt.

A ThermoSmart több fő végpontot is támogat. Ezek közé tartozik a `/temperature` végpont, ahol lehetőségünk adódik a termosztáttól kapott hőmérsékleteket elmentetni cache-ba vagy az adatbázisunkba. A `/device` végpontban az eszközökkel kapcsolatos műveleteket hajthatunk végre, mint például új termosztát regisztrálása, nevének frissítése, valamint egy eszköz adott felhasználóhoz való hozzárendelése. A `/weather` végpont az OpenWeatherMap[4] API-ját felhasználva lekéri a következő nap hőmérséklet-előrejelzését, amelyet majd továbbküld a mobilalkalmazásnak.

A kérések fogadása után következik a kéréssel érkező adatoknak az ellenőrzése, ezeket mind *DTO*-vá alakítjuk. Ha hibás paramétert kaptunk (nem megfelelő típusú, nem létező stb.), a megfelelő státuszkódot visszaküldjük a kérő félnek. Ellenkező esetben a kérést továbbítjuk a service rétegnek.

[4] `https://openweathermap.org/api`

```
const strategyOptions: StrategyOptionsWithRequest = {
  clientID: process.env.GOOGLE_CLIENT_ID as string,
  clientSecret: process.env.GOOGLE_CLIENT_SECRET as string,
  callbackURL: `${process.env.BASE_URL}/auth/google/callback`,
  passReqToCallback: true,
};
```

1. kódrészlet. A Google által generált hitelesítő adatok, amit a Passport.js fel fog használni

1.1.5. Bejelentkezés

Mivel a bejelentkezés csak egy Google profillal történhet meg, ezért a *Passport.js* [4]-et (ld. 1.1.1. fejezet) használtuk fel. A Google az OAuth2.0 protokollt használja, emiatt a passport-google-oauth20 stratégiát használjuk. A middleware használatához szükséges regisztrálni az alkalmazást a Google Developers Console-ban. Ezt követően generálódik egy Client ID és egy Client Secret, amit a stratégia beállításakor meg kell adni.

Használat előtt egy Express.js session létrehozására, a Passport inicializálásra, illetve a Passport-specifikus *session* függvény meghívására van szükség. Ez utóbbi bejelentkezteti az Express session-t a Passport session stratégiájával.

A fentieknek köszönhetően minden alkalommal, mikor bejelentkezési kísérlet történik meg, lefut a *GoogleStrategy* nevű stratégia, ahol három eset történhet meg: sikeres a bejelentkezés és már létezik a felhasználó, sikeres a bejelentkezés, de még nem létezik a ThermoSmart rendszerben ilyen felhasználó, ezért az adatbázisban létrehozzuk, vagy sikertelen a bejelentkezés. Amennyiben sikeres a bejelentkezés, a done függvényt hívjuk meg, ami jelzi a programnak, hogy ugorhat a következő végponthoz.

Két végpont szükséges a bejelentkezéshez, egy, amelyiknél elindul a bejelentkezés, illetve egy, amelyikre vissza lesz irányítva a felhasználó sikeres bejelentkezés esetén. Ekkor újra át

```
app.use(
  session({
    secret,
    resave: false,
    saveUninitialized: true,
  })
);
app.use(passport.initialize());
app.use(passport.session());
```

2. kódrészlet. A Passport.js inicializálása

```
passport.use(new GoogleStrategy(strategyOptions, async (
    {...}
    profile: passportGoogle.Profile,
    done: passportGoogle.VerifyCallback
) => {
  const user: User = await getUserByEmail(profile.emails[0].value);
  {...}
  if (user === null) {
    const newUser: User = new User({...});
    await createOrUpdateUser(newUser);
    logger.info(`[${filePath}] New user created`);
    done(null, newUser);
  } else {
    await createOrUpdateUser(user);
    logger.info(`[${filePath}] User found`);
    done(null, user);
  }
});
```

3. kódrészlet. Bejelentkezés.

```
router.get("/",
  passport.authenticate("google", {
    scope: ["email", "profile"],
    prompt: "select_account",
  })
);
router.get(
  "/callback",
  passport.authenticate("google", { failureRedirect: "/auth/google" }),
  (req, res) => {
    logger.info(`[${filePath}] Successfully authenticated user`);
    res.redirect(`thermosmart://app/login?token=${req.cookies.token}`);
  }
);
```

4. kódrészlet. A Passport.js által használt végpontok

lesz irányítva a mobilalkalmazásra.

1.1.6. Kommunikáció a mobilapplikációval

A pillanatnyi hőmérséklet-információ késlekedés nélküli továbbítása céljából a szerver és a mobil között fent áll egy WebSocket alapú kapcsolat. Egy olyan mechanizmust implementálunk, amely az observer mintára épül, és felhasználja ezt a kapcsolatot.

A mobil nyitja a kommunikációt, és közli, hogy érdekelt egy adott termosztát adataiban. A szerver egy saját fejlesztésű adatszerkezetet használ a feliratkozások tárolására és elérésére. A hasítótáblába [29] ágyazott láncolt listák [37] (minden lista egy termosztát figyelőit tartalmazza) segítségével konstans időben érhetjük el az egy adott termosztáthoz tartozó mobilokat, akiket értesíteni kell a változásokról.

A szerver azon metódusai (ld. 1.1.2. fejezet), amelyek felelősek a termosztátok információinak mentéséért kiváltják a figyelő mobilok kiértesítését.

1.2. Mobilalkalmazás

A mobilalkalmazás a felhasználói felület, ahonnan szabályozhatjuk otthonunk hőmérsékletét, statisztikát kapunk, és beállíthatjuk termosztátunk helyzetét a térképen, ami elengedhetetlen a pontos időjárás-előrejelzéshez, amely szintén egy funkciója az alkalmazásnak. Továbbá itt állíthatunk be hagyományos programokat, például hogy minden reggel hétkor legyen 21 fok. A mobilalkalmazáson keresztül tudjuk csatlakoztatni magunkat egy adott termosztáttal.

Minden funkció egy adott termosztátra érvényes. A rendszer általánosítva van, hogy több termosztát kezelésére is alkalmas legyen. Lehetőség van választani a regisztrált termosztátok között. Mikor áttérünk egy másik termosztátra, a régi adatok helyét átveszik az új termosztáthoz kapcsolódóak.

A program a React Native [48] könyvtár segítségével lett megírva TypeScript nyelven. Ennek ki is használtuk a lehetőségeit mind design, mind működés szempontjából. Például adatelosztásra a Context API [17]-t, design-ra pedig egy külső könyvtárat (React Native Elements [47]).

A React Native egy Reactre [46] épülő könyvtár. A React stílusát, és stratégiáit használja, gyakorlatilag React kódot írunk, apróbb különbségekkel, viszont míg a React weboldalak tervezésére alkalmas, addig a React Native mobilapplikációk fejlesztésére képes, Androidra és iOS[5]-re egyszerre.

Olyan JavaScript függvényeket írunk benne, amelyek képesek a DOM-ot élőben és részlegesen manipulálni vagy generálni. Ezeket programatikusan változtathatjuk tetszés szerint. A HTML fastruktúráját követi, függvények hívnak más függvényeket szélességi bejárás-szerűen, hogy összeálljon egy teljes weboldal.

[5]Jelen esetben ez még nem áll rendelkezésre.

React Native-ben a React szintaxisú kód natív Java/Kotlin és Swift nyelvekre transzpilálódik, így két platformra fejleszthetünk egy kódbázissal. A különböző külső könyvtárak is úgy működnek, hogy az általuk adott React komponenseknek biztosítanak egy natív megfelelőt a két platformra.

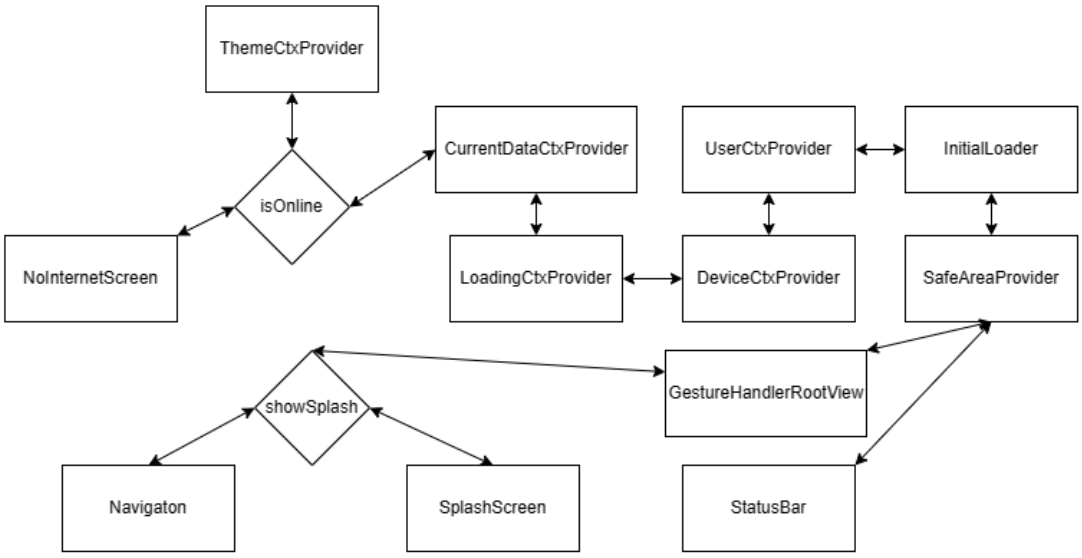
1.2.1. Felépítés

Az applikáció kilenc fajta fontosabb elemből áll össze:

- *modellek*: Adatok reprezentációi
- *komponensek*: Elemi kinézeti elemek (előre design-olt gombok, szövegmezők, third party elemek)
- *képernyők*: Komponensek gyűjteményei. Elfoglalják a teljes képernyőt, egy bizonyos funkcionalitáshoz tartoznak.
- *kontextusok*: A futó program adatait, és a hozzuk társuló műveleteket tárolják.
- *kontextus szolgáltatók*: Globálissá teszik a kontextusokat.
- *WebAPI*: Függvények gyűjteménye a szerverrel való kommunikációhoz.
- *SocketAPI*: WebSocketeken érkező adatok figyelői.
- *stílusok*: Egy komponens vagy képernyő stílusjegyei.
- *navigáció*: Stacknavigátor [50] a képernyők közti váltásokhoz.
- *helyi adattároló*: Könyvtár a telefon háttértárolójába való mentéshez.

A következő felsorolás tárgyalja a fő elemek hierachiáját (ld. 3. ábra).

A gyökerében a `ThemeCtxProvider` található, amely globális beállításokat(design) ad a komponenseknek. Ez után egy teszt jön, amelyik ellenőrzi, hogy van-e internethozzáférés(`isOnline` változó értékének vizsgálata). Ha nincsen, akkor a hierarchia megszakad, és csupán egy darab képernyőt kapunk `NoInternetScreen` néven. Ha van internetelérés, akkor jön a `CurrentDataCtxProvider`, amely a termosztát pillanatnyi státuszáról tárul infomációt. A `LoadingCtxProvider` azt tartja számon, hogy várunk-e éppen választ a szervertől vagy nem. A `DeviceCtxProvider` a termosztát általános információit tárolja. A `UserCtxProvider` nem meglepő módon a felhasználó adatait tartja számon. Az `InitialLoader` az alkalmazás indulásánál történő adatbetöltésért felel.



3. ábra. Gyökérkomponensek sémája

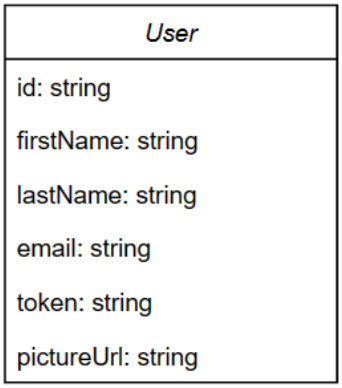
A `SafeAreaProvider` egy külső könyvtár eleme. Biztosítja, hogy csak olyan felületre rajzolhassunk a telefon képernyőjén, amely semmiképpen sem okozhat gondot (pl. nem enged elhelyezni valamit a status bar-ra). A `StatusBar` szintén third party elem, stílust ad hozzá a natív status bar-hoz. A `GestureHandlerRootView` segédeleme az érinthető felületeknek. Ismét teszt következik, amely azt hivatott eldönteni, hogy az úgyn. `SplashScreen`-t vagy a `Navigation`-t jelenítsük meg. A `SplashScreen` egy olyan képernyő, amely csak induláskor látszik, amíg a kezdeti műveletek végrehajtódnak, az alkalmazás logóját tartalmazza. A `Navigation` tartja karban a képernyők közötti navigációt. Verem alapú navigációt használtunk, vagyis a képernyők egymás fölött jelennek meg.

1.2.2. Adattárolás

Az alkalmazás, hogy bizonyos adatokat ne kelljen minden indításkor lekérnie a szervertől, használja a telefon háttértárát. Menti a felhasználó adatait (ld. 4. ábra) és az utolsó ismert célhőmérsékletet. A tárolt adatok kulcs-érték párok formájában érhetőek el. Ennek megvalósítására a react-native-mmkv [49] könyvtárat használtuk.

1.2.3. Bejelentkezés

Induláskor az alkalmazás vizsgálja, hogy vannak-e már lementett felhasználói adatok a telefonon. Ha vannak, akkor lekéri az utoljára használt termosztát adatait, és megjeleníti a főoldalt. Ha nicsenek, akkor kéri a felhasználót, hogy jelentkezzen be.



4. ábra. A dekódolt JWT token [35] felépítése és a tárolt felhasználói adatok sémája

A bejelentkezés a Google hitelesítő szolgáltatásán keresztül történik. Bejelentkezéskor megnyílik egy natív böngésző a telefonon, amelyben a standard Google login form jelenik meg, amelyet már a fent említett szolgáltató biztosít. A hitelesítő adatok beírása után a szerver, miután egyeztet a Google-el, visszadob egy JWT tokent, amely tartalmazza az adott felhasználó minden infomációját (ld. 1.1.5 fejezet). A folyamat *deep linking* [18]-el van megoldva, amely annyit tesz, hogy van egy *event listener*[6], amely a natív böngésző kérésére érkező válaszra vár.

Mikor megkapjuk a token-t, dekódolnunk kell azt, így segítségül hívjuk a szerver *.../auth/google/introspect* végpontját (ld. 1.1.4 fejezet), amely JSON formájában visszaküldi a token tartalmát.

1.2.4. Kommunikáció

Minden alkalmazásban van egy réteg, amely a kommunikációért felel. Jelen esetben két különböző technológiát is felhasználtunk erre: egyszerű HTTP hívásokat és WebSocketeket (ennek oka az 1. fejezetben található).

A HTTP hívások egyszerűsítésére az Axios könyvtárat használtuk, amelyben lehtőség volt globálisan beállítani a szerver URL-jét a következőképpen:

```
const axiosAPI = axios.create({
    baseURL: BASE_URL,
});
```

A `WebAPI` minden kérésénél használja DTO tervezési mintát. Szemléltetésképpen az 5. kódrészlet tekinthető meg. A kérés egy POST igével van megvalósítva. A DTO jelen esetben a `PostPreferredTemperatureDTO` képében van jelen, az adott hőfokot és a

[6] Az Observer minta egyik eleme, egy esemény bekövetkezésekor csinál valamit.

```
static postPreferredTemperature(
   temperature: number,
   deviceId: string,
 ): Promise<void> {
   const data = new PostPreferredTemperatureDTO(temperature, deviceId);
   return axiosAPI
     .post('/device/set', data, getConfigWithTokenFromStorage())
     .then(() => {
       console.log(`[WebAPI] Set preferred temp to ${temperature}C`);
       return Promise.resolve();
     })
     .catch(err => {
       console.error('[WebAPI] Error at setting preferred temp: ', err);
       ToastAndroid.show(
         'Error at setting preferred temperature',
         ToastAndroid.LONG,
       );
     });
}
```

5. kódrészlet. A függvény a célhőmérsékletet közli a szerverrel.

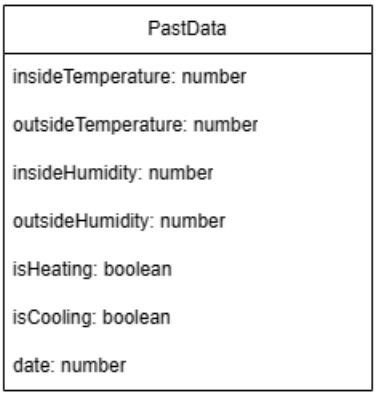
termosztát azonosítóját tartalmazza. A kérés a */device/set* útvonalra megy egy JWT token kíséretében, amelyet a `getConfigWithTokenFromStorage` nevű függvény varázsol elő a telefon háttértárából, majd az *Authorization* header-be illeszti, mint *Bearer* [1].

Másodsorban, a mobilalkalmazásnak valós időben kell mutatnia az otthoni hőmérsékletet. Ez megoldható lenne folyamatos HTTP lekérésekkel, ahogy ez az 1. fejezetben már említve volt, de ennél van jobb megoldás. A WebSocket-ek képesek observer mintára alapuló, valós idejű oda-vissza kommunikációt megvalósítani, így ezt a technológiát használtuk.

A Socket.IO könyvtárat hívtuk segítségül, hogy létrehozzunk egy csatornát, amely az alkalmazás indulásával jön létre, és képes adatokat küldeni és fogadni. A könyvtár biztosítja, hogy bármilyen megszakadás automatikusan orvoslódik az első adódó lehetőségnél, és közben nem történik adatvesztés. Ezt a csatornát használtuk arra, hogy fogadjuk a pillanatnyi adatokat a termosztáttól (pl. külső, belső hőmérséklet, a fűtés ki/be kapcsolásának sikeressége). További információ a 1.1.6 fejezetben található.

1.2.5. Grafikon

A ThermoSmart projekt egyik célja az, hogy segítséget nyújtson a felhasználóknak arra, hogy optimálisan fűthessék az otthonukat. Az egyik módszerünk erre az, hogy statisztikát



5. ábra. A szervertől kapott múltbéli hőmérséklet-páratartalom-információ sémája.

biztosítunk azt illetően, hogy milyen összefüggésben áll a házon belüli, a házon kívüli hőmérséklet és maga a fűtés.

Ezt az ötletet grafikon formájában valósítjuk meg, amely magába foglalja az előbb említetteket. Az információt a szervertől kérjük le a `WebAPI`-n keresztül a következő módon.

```
WebAPI.getPastDataWithin(1,'day',device.id)
```

A függvény általánosítva van, hogy a jelen pillanattól kezdve x darab 'day'/'hour'/'week'/'month'-val visszamenőleg szolgáltassa az összes `PastData`-t (ld. 5. ábra). A dátum Linux stílusú Epoch formátumban van, vagyis az 1970. január 1. 00:00-tól eltelt milliszekundumok számát jelenti. A páratartalom felhasználása egy lehetőség a projekt további fejlesztésére.

A szerver nagy mennyiségű adathalmazt biztosít, mivel kis időközönként (12 adat/óra) mentjük a hőinformációt. A grafikon csak véges mennyiségű pontot rajzolhat ki, hogy esztétikusan nézzen ki, így az adatokat kiátlagoltuk.

1.2.6. Design

Design szempontjából a React Native Elements könyvtárra esett a választás, amely olyan komponenseket biztosított, melyeknek közös stílusjegyeket adhattunk (pl. közös betűtípus, betűméret stb.). Továbbá, a könyvtár alkalmazásával egyszerűen meg tudtuk oldani a világos és sötét (vagy nappali és éjszakai) témák közti váltást. Ahogy a 6. ábrán ez látható, két színpaletta van definiálva. Egy a világos, egy pedig a sötét témához. A komponensek ugyan azt a változót használják színezésre világos és sötét módnál egyaránt. A köynvtár képes kicserélni ezeknek a változóknak az értékeit egy általunk hívott függvény hatására (`setMode(mode)`, ahol a `mode` `'light'` vagy `'dark'` értékeket vehet fel).

```
export const GeneralTheme = createTheme({
  lightColors: {
    backgroundColor: '#fafafa',
    cardColor: '#ffffff',
    shadowColor: 'rgba(0,0,0,0.20)',
        ...
  },
  darkColors: {
    backgroundColor: '#363838',
    cardColor: '#545454',
    shadowColor: '#000000',
        ...
  },
  components: {
    Text: {
      style: {
        fontFamily: 'Itim-Regular',
        fontSize: 18,
        color: 'black',
      },
    },
  },
});
```

6. kódrészlet. A két színpaletta és a szövegmezők általános beállításai.

1.3. Raspberry Pi 4 mint termosztát

A termosztát szimulálását egy Raspberry Pi 4-es miniszámítógéppel oldottuk meg. Azért esett erre az eszközre a választásunk, mivel támogatja az általunk használt szenzorokat, ezért könnyebb és olcsóbb megoldást nyújt a kimondottan erre kitalált termosztátokkal szemben. Erre rákötöttünk két DHT22-es [19] hőmérséklet- és páratartalom-szenzort, amelyek közül az egyik a lakáson kívüli, a másik a lakáson belüli hőmérsékletet hivatott figyelni.

A Raspberry Pi egy tenyérnyi számítógép. Kis termete ellenére alig különbözik asztali géptől. Saját processzora, RAM-ja és videókártyája van. Háttértárolója viszont nincs, ezt egy microSD kártya behelyezésével orvosolhatjuk. Egy teljes értékű operációs rendszert futtathatunk rajta. A forgalmazó a saját fejlesztésű, Linux alapú Raspberry Pi OS-t ajánlja, amit erre a célra találtak ki, és amit mi is használtunk.

A Raspberry Pi abban tér el igazán a klasszikus számítógépektől, hogy negyven darab kiálló pin-je van, amelyekből egy pár csak elektromos áramot szolgáltat, viszont a legtöbb programozható. Ezt legfőkébb úgy tudjuk kihasználni, hogy direkt erre a célra

```
async setEventLoop() {
  logger.info('[App/setEventLoop]: Setting event loop');
  setInterval(async () => {
    const temperature: Temperature = await this.getAndSaveTemperature();
    await this.addTemperatures(temperature);
    await this.checkTemperature(temperature);
  }, parseInt(process.env.INTERVAL_PER_REQUESTS));
}
```

7. kódrészlet. A Raspberry Pi 4-en futó fő ciklus.

gyártott külső modulokat csatlakoztatunk az eszközhöz, mint például a hőmérséklet- és páratartalom-érzékelők, valamint a relé, amiket felhasználtunk a projektünkben.

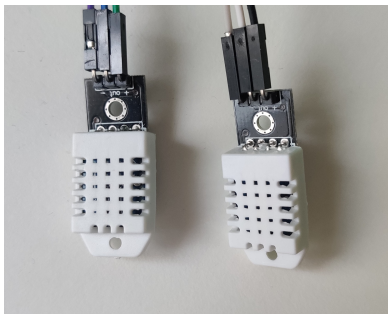
1.3.1. A kód felépítése

A Raspberry Pi-on futó kód TypeScriptben van írva és a projekt többi részéhez hasonlóan objektumorientált elveket követ. A szerverrel a *REST API*-n keresztül kommunikál, míg a szerver neki a *CloudAMQP* szolgáltató RabbitMQ klasztere segítségével sugároz neki vissza adatokat (ld. 1.1.1. fejezet). A magja egy cikluson alapszik, amely előre megadott időnként újra és újra lefut. Ez a ciklus három függvényt tartalmaz:

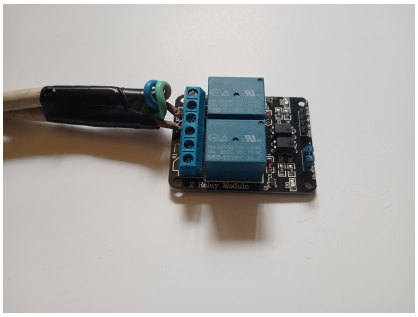
A `getAndSaveTemperature` függvény lekéri a szenzoroktól a jelenlegi hőmérsékletet illetve a páratartalmat, majd elküldi ezeket a szervernek a fent említett *REST API* segítségével. A szerver elmenti az adatbázisba is a mért adatokat, viszont mivel rövid intervallumonként küld adatokat a termosztát, túl hamar megtelne adatokkal. Ennek érdekében a `addTemperatures` függvény az előző öt mérésből átlagot von, majd ezt az értéket elküldi a szervernek, hogy az elmentse az adatbázisba, így nem lesz túlterhelve az adatbázis. A `checkTemperature` függvény ki vagy bekapcsolja a fűtő és hűtő berendezéseket a hőmérséklet függvényében, majd ezek állapotát elküldi a szervernek.

1.3.2. Fűtés-hűtés

A kazánunkat és vagy légkondinkat úgy tudjuk irányítani, hogy hol adunk nekik áramot, hol pedig nem, amit egy SL-C relé [51] (ld. 7. ábra) segítségével oldottunk meg. Két egység van egybeépítve. Mind a kettőnek három csatlakozója van. A középső bemeneten szolgáltatott áram vagy a jobb vagy a bal oldali kimeneten fog távozni, ezt tudjuk a Raspberry-n keresztül befolyásolni, és így értük el az energia továbbítását vagy megszakítását egy kábellel, amibe be



6. ábra. DHT-22 hő és páratartalom-érzékelő.



7. ábra. SL-C relé.

8. ábra. Raspberry Pi 4.

van ékelve a relé.

2. Technológiák

A cikk ezen részében a szerver, a mobilalkalmazás és a termosztát fejlesztéséhez használt technológiákról lesz szó, amelyek rengeteg időt megspóroltak nekünk. Olyan eszközök ezek, amelyek nem kizárólag a rendszer egyik rétegéhez tartoznak.

2.1. Git

A Git [23] egy verziókövető rendszer, amely segít a fejlesztőknek a kódjukban bekövetkezett változások kezelésében, a fejlesztés több ágának kezelésében és a másokkal való együttműködésben. Beépített eszközöket biztosít a konfliktusok feloldásához és összevonásához. Világszerte népszerű eszköz a szoftverfejlesztő csapatok számára [13].

2.2. GitLab

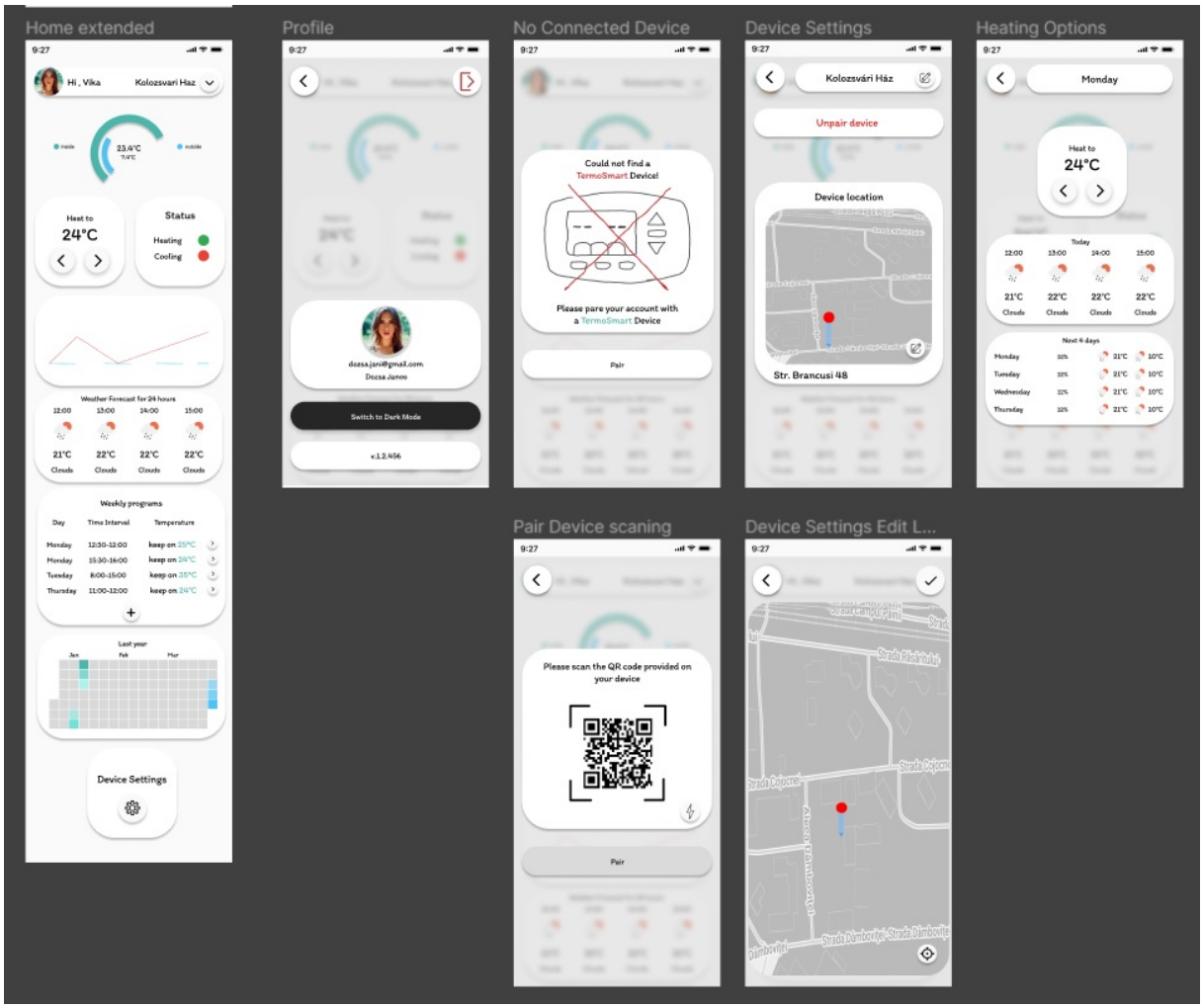
A GitLab [24] egy Gitre épített weboldal, ahová feltölthetjük a kódunkat. Ezen kívül lehetőséget nyújt a programozói csapat feladatainak rendszerezésére. Felvehetünk taskokat, leírást adhatunk nekik, hozzárendelhetünk egy programozót, majd ha elkészült vele, megoldható, hogy egy másik ember átnézze a munkáját, és kommenteket hagyjon rajta, a lehető legjobb kódminőség elérésének érdekében.

Továbbá, bevezeti a pipeline [25] fogalmát, amely arra szolgál, hogy miután elkészültünk egy program egyik verziójával, legyen az bármilyen nagy vagy kicsit, különböző automatikus műveleteket hajtsunk végre rajta. Pl: verfikáció & validáció, kitelepítés.

2.3. Docker, K8s és Rancher

A Docker [20] egy eszköz, amely számtalan problémánkat oldja meg. Ezek közül az egyik a verziószámok problémája. Bizonyos programok x verzióját követelik meg egy segédeszközüknek, más programok pedig talán y-t, így nem futhatnak egymás mellett.

Másod sorban, amikor elkészülünk egy programmal, és átadjuk a kliensnek, akkor ő csak azután tudja futtatni az adott kódot, miután bizonyos előkészületek meg nem történnek az adott számítógépen. Nem tudunk JavaScript kódot futtatni Node.js nélkül, Java kódot JRE [33] nélkül. Továbbá, szükségünk lehet környezeti változókra, amelyeket Docker nélkül kézzel kéne beállítani. Ezek mind olyan dolgok, amelyek telepítésre és vagy kalibrálásra várnak futtatás előtt.



9. ábra. Figmában készült design.

A Docker ezeket, és a számtalan fel nem sorolt problémát úgy oldja meg, hogy "konténerizálja" a programunkat. Ez annyit tesz, hogy létrehoz egy pehelysúlyú futási környezetet, amire rátelepíteni programunk minden függőségét és elvárását, majd elindítja. Így egy elszigetelt környezetben futhat, amelyben csak az van, ami számára releváns.

A Rancher [43] egy platform, amely megkönnyíti és egységesíti a konténerizált applikációink kitelepítését és karbantartását különböző infrastruktúrákban. Lehetőséget nyújt automatikus kitelepítésre, skálázásra, monitorizálásra és logolásra. Őt választottuk arra, hogy egy helyi infrastruktúrán futó Rancher által menedzselt Kubernetes klaszterre telepítsük a szerverünket, amely bármikor, bárhonnan elérhető.

A K8s, a Kubernetes [36] rövidítése, egy nyílt forráskódú konténer-orchestrációs platform, amelyet a konténeres alkalmazások telepítésének, skálázásának és kezelésének automatizálására terveztek. A K8s a konténerizált munkaterhelések széles skáláját képes kezelni, beleértve az állapot nélküli és állapotú alkalmazásokat, a kötegelt feladatokat és

még sok mást. Olyan funkciókat is biztosít, mint a szolgáltatásfelfedezés, a terheléselosztás, a tárolás-orchestrálás és az öngyógyítás, amelyek segítenek a csapatoknak megbízható és skálázható alkalmazások létrehozásában és futtatásában.

Docker-t használtunk, hogy kitelepítsük a szerverünket Kubernetes segítségével Rancher-re.

2.4. Figma

A Figma [21] egy felhő alapú design eszköz. Segítségével a tervezők nagy hűségű maketteket, prototípusokat és tervezési rendszereket hozhatnak létre, és valós időben együttműködhetnek a csapat többi tagjával. Tartalmaz vektorgrafikát, ikonokat és minden tipikus design eszközt. A mobilapplikáció kinézetének megtervezésére és karbantartására használtuk (ld. 9. ábra).

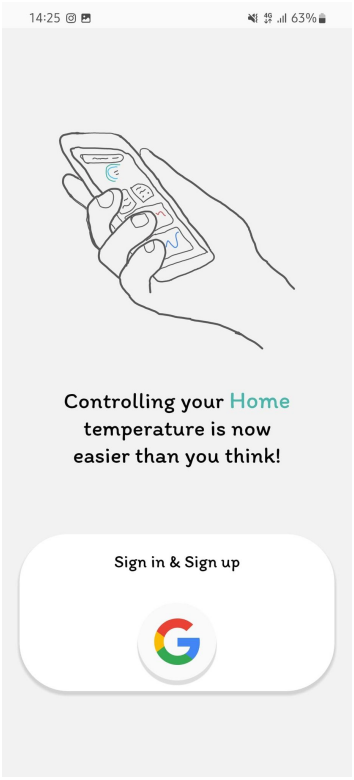
3. Használat

A cikk ezen részében a termosztát felszerelését, a mobilalkalmazás működését és kialakítását vizsgáljuk meg, amely lehetővé teszi a felhasználók számára, hogy távolról vezéreljék termosztátjukat. Az okosotthon-technológia térhódításával egyre népszerűbbé vált az a lehetőség, hogy okostelefonról állítsuk be otthonunk hőmérsékletét. Ezt a mobilalkalmazást úgy terveztük, hogy a felhasználóknak teljes körű ellenőrzést biztosítson a termosztátjuk felett, függetlenül attól, hogy hol tartózkodnak. Megbeszéljük az alkalmazás jellemzőit, beleértve az intuitív kezelőfelületet, a könnyű használatot és a felhasználók számára elérhető különböző beállításokat.

3.1. Hőmérők felszerelése

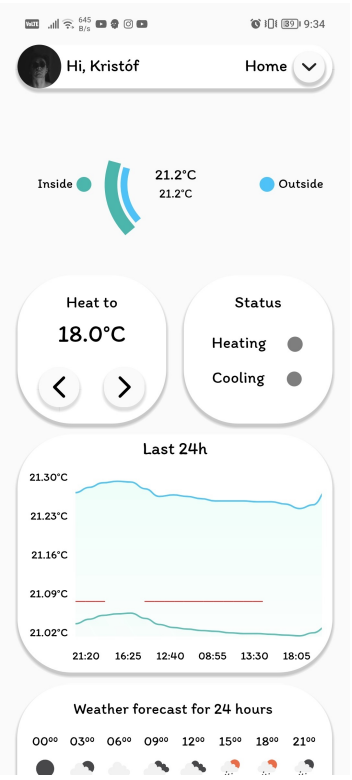
Annak érdekében, hogy a hőmérők valós információt tudjanak biztosítani, mind a kettőt naptól védett helyre kell elhelyezni, az egyiket a házon belülre, a második pedig kívülre.

A külsőt védeni kell az időjárás viszontagságaitól. A legfontosabb, hogy ne érje víz, és ne lepje be a hó. Ha ez mégis megtörténik, az eszköz tönkremehet.

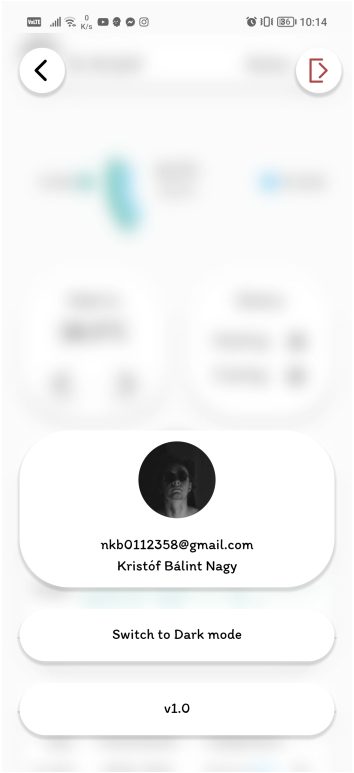


10. ábra. Bejelentkezési oldal

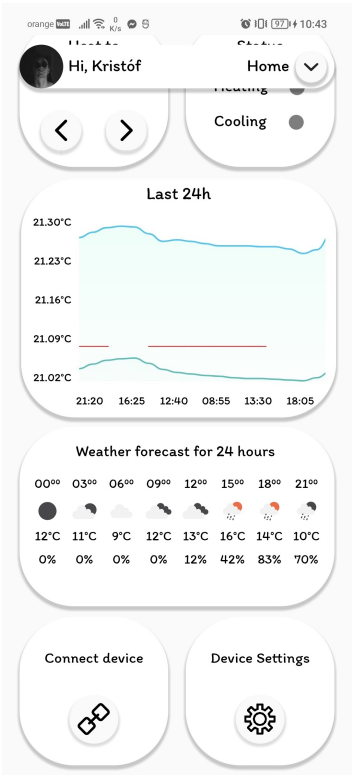
11. ábra. QR-kód olvasó



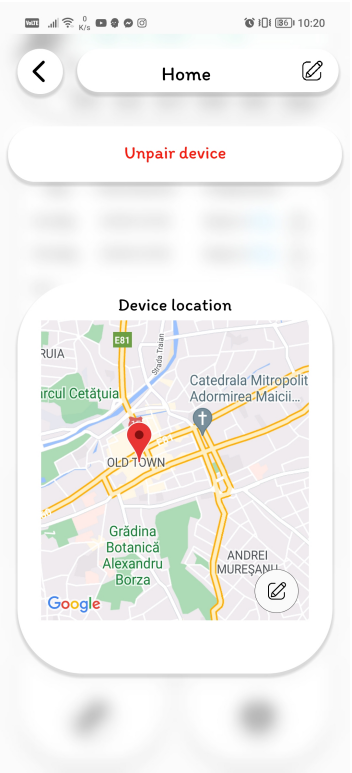
12. ábra. A főoldal teteje



13. ábra. A profil képernyője–Itt válthatunk sötét módba



14. ábra. A főoldal alja



15. ábra. A beállítások képernyője

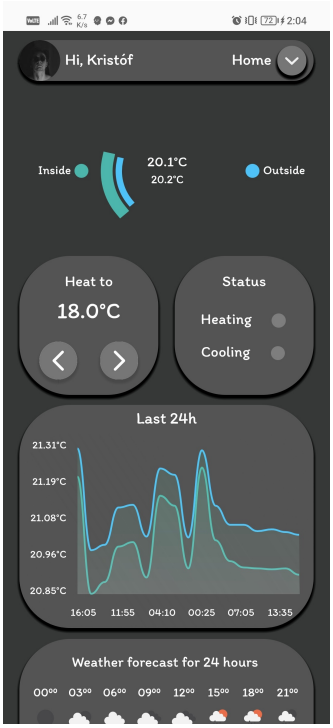
3.2. Bejelentkezés és a termosztát csatlakoztatása

A bejelentkezés Google-ön keresztül történik. A felhasználónak szüksége van egy Google fiókra, hogy használatba vegye a mobilapplikációt. Indulás után a 10. ábrán látható oldal fogadja a felhasználót. A Google gombot megérintve megnyílik a klasszikus Google féle bejelenkező oldal, ahol szükség van egy felhasználónévre és egy jelszóra. Bejelentkezés után egy QR-kód olvasóval találjuk szembe magunkat (ld. 11. ábra). A termosztát el van látva egy QR-kóddal, amelyet itt beszkennelhetünk a csatlakoztatáshoz. Ha sikeres a beolvasás, akkor megjelenik a termosztát egyedi azonosítója, és aktívvá válik a `Pair` gomb. Ha túl sötét van a QR-kód beolvasásához, akkor a jobb alsó sarokban a villám jelre nyomva bekapcsolhatjuk a telefon vakuját.

3.3. Főoldal

A felső sávban, bal oldalon a Google profilunk fényképét láthatjuk (ld. 12. ábra). Az avatárra nyomva felugrik a profil képernyő, amely további információkat nyújt a felhasználóról (ld. 13. ábra).

16. ábra. Nagyobb térkép



17. ábra. Sötét mód

Jobb oldalon a jelen pillanatban kezelt termosztát neve van. A mellette lévő nyíllal le tudjuk nyitni az összes, a felhasználóhoz csatlakoztatott termosztát listáját. Itt, ha rányomunk az egyikre, akkor az alkalmazás innentől kezdve ezt a termosztátot irányítja, és a tőle származó információkat jeleníti meg.

Középen található a "*monitor*", amely élő információt mutat a hőmérsékletről. A külső gyűrű a benti(felső, nagyobb betűvel írt szám), a belső gyűrű a kinti(alsó, kisebb betűvel írt szám) hőmérsékletet reprezentálja.

A "*monitor*" alatt balra van a legfontosabb "*kártya*", amellyel a hőmérsékletet szabályozhatjuk. A gombokat hosszan tapintva gyorsan, nagy változást érhetünk el, ha csak egyszer nyomjuk meg őket egy fél fokkal tudjuk fennebb vagy lennebb vinni a célhőmérsékletet.

Tőle jobbra a státuszkártya van. Azt jelzi, hogy a fűtő és hűtő berendezéseink be vannak-e kapcsolva. Ha igen, akkor egy színes kör jelenik meg a szürke helyén.

Alatta található a grafikon, amely statisztikát biztosít az elmúlt egy napról. A színek jelentését nem tüntettük fel ismét, felül a "*monitor*"-nál láthatóak. A vörös vonal azt jelzi, hogy mikor volt bekapcsolva a fűtés.

A státuszkártya alatt a következő 24 óra időjárás-előrejelzése van (ld. 14. ábra), amely élőben jön az internetről, és a termosztát helyéhez van kötve (ld. 15. ábra). Ennek működése az 1.1.4. fejezetben van részletezve.

Az utolsó két kártya egyike arra szolgál, hogy új termosztátot rendelhessünk az aktív felhasználóhoz. Ezt megnyomva egy QR-kód szkennelő fogad minket, ugyan úgy, mint a 3.2. fejezetnél.

Az utolsó kártya pedig a beállítások képernyőjére irányít (ld. 15. ábra). A jobb felső sarokban lévő "*edit*" gombra nyomva megváltoztathatjuk a termosztát nevét, amely globális a felhasználók között. Az `Unpair device` gombbal lecsatlakoztathatjuk a termosztátot a felhasználóról. Ha elfogyott az összes termosztát, akkor ismét a QR-kód szkennelővel találjuk szembe magunkat. Az alatt lévő térkép a termosztát jelenlegi helyzetét jeleníti meg, amely az időjárás-előrejelzésben játszik szerepet.

A 16. ábrán egy nagyobb térkép látható, amelyet úgy érünk el, hogy megnyomjuk a kisebb térkéb jobb alsó sarkában lévő gombot. Itt foghatjuk meg a pin-t, és tehetjük arrébb.

Végső soron, a 17. ábrán a sötét mód látható éjszakai használathoz.

Következtetések és továbbfejlesztési lehetőségek

A jelen dolgozat bemutatta a ThermoSmart projektet, annak főbb komponenseit és elemeit. A kezdetekkor felvázolt célokat sikeresen teljesítettük, létrehoztuk a mobil alkalmazást és a szervert, sikeresen konfiguráltuk a Raspberry Pi eszközt és a szenzorokat, illetve a reléket működésbe hoztuk. Mindezek ellenére viszont sok funkcionalitást felfedeztünk, úgy a munka során, mint a kódírás befejezte után, amelyek kisegíthetik az alkalmazást, azt jobbá tehetik. Több szálon is továbbfejleszthető a projekt, amiket a későbbiekben tervezünk bevezetni.

A legfontosabb opció egy mesterséges intelligencia bevezetése, amely az elmúlt (adatbázis) és a jövendőbeli (időjárás-előrejelzés) hő-információk alapján optimális fűtési programot tervezne. Így a felhasználónak gyakorlatilag csak annyi lenne a dolga, hogy megmondja mikor hány fokot szeretne, majd minden mást elintézne a mesterséges intelligencia. Ezzel a módszerrel olyan programok születnének, amelyekre ember nem gondolna.

Egy másik opció egy adminisztrációs felület elkészítése, amely egy weboldalt jelentene, ahol a termosztátokat vezethetjük be a rendszerbe, felhasználókat törölhetünk, meghibásodás esetén pedig akár újra is indíthatnánk a rendszert.

Végső soron, a biztonság növelése fontos továbbfejlesztés, ugyanis jelen pillanatban bárki, aki rendelkezik egy termosztát QR-kódjával, irányíthatja a termosztátot. Ez egy jelszó bevezetésével megoldható.

Hivatkozások

[1] *A bearer token dokumentációja a Swagger weboldal részéről.* URL: https://swagger.io/docs/specification/authentication/bearer-authentication/ (elérés dátuma: 2022. aug. 12.)

[2] *A CloudAMQP weboldala*. URL: https://www.cloudamqp.com/ (elérés dátuma: 2022. júl. 23.)

[3] *A Node.js futtatási környezet hivatalos dokumentációja.* URL: https://nodejs.org/en (elérés dátuma: 2022. júl. 23.)

[4] *A Passport.js hivatalos dokumentációja.* URL: https://www.passportjs.org/ (elérés dátuma: 2022. aug. 2.)

[5] Ihechikara Vincent Abba. „What is an ORM – The Meaning of Object Relational Mapping Database Tools”. (2022).

[6] Peffer et al. „Facilitating energy savings with programmable thermostats: evaluation and guidelines for the thermostat user interface”. (2013). URL: https://pubmed.ncbi.nlm.nih.gov/23005033/.

[7] *AMQP hivatalos dokumentáció.* URL: https://www.amqp.org/ (elérés dátuma: 2022. júl. 20.)

[8] *API meghatározás az AWS révén.* URL: https://aws.amazon.com/what-is/api/ (elérés dátuma: 2022. júl. 18.)

[9] *Axios hivatalos dokumentáció.* URL: https://axios-http.com/docs/intro (elérés dátuma: 2022. júl. 1.)

[10] *Az ESLint hivatalos dokumentációja.* URL: https://eslint.org/ (elérés dátuma: 2022. júl. 1.)

[11] *Az Express.js hivatalos dokumentációja.* URL: https://expressjs.com/ (elérés dátuma: 2022. júl. 14.)

[12] *Az NPM hivatalos dokumentációja.* URL: https://www.npmjs.com/ (elérés dátuma: 2022. júl. 12.)

[13] *Beyond Git: The other version control systems developers use*. URL: https://stackoverflow.blog/2023/01/09/beyond-git-the-other-version-control-systems-developers-use/ (elérés dátuma: 2022. aug. 16.)

[14] Colin But. „Dissecting the DTO pattern”. (2020). URL: https://blog.devgenius.io/dissecting-the-dto-pattern-ac3e54d0e4c8.

[15] *Cache meghatározás az AWS révén.* URL: https://aws.amazon.com/caching/ (elérés dátuma: 2022. júl. 19.)

[16] Chip Berry Chrishelle Lawrence Maggie Woodward. „Most homes have central thermostats on heating and cooling equipment". (2017). URL: https://www.eia.gov/todayinenergy/detail.php?id=14771.

[17] *Context API hivatalos dokumentáció.* URL: https://react.dev/reference/react/useContext (elérés dátuma: 2022. jún. 21.)

[18] *Deep Linking hivatalos dokumentáció.* URL: https://reactnavigation.org/docs/deep-linking/ (elérés dátuma: 2022. aug. 20.)

[19] *DHT22 szenzor hivatalos dokumentáció.* URL: https://reference.arduino.cc/reference/en/libraries/dht22/ (elérés dátuma: 2022. aug. 24.)

[20] *Docker hivatalos dokumentáció.* URL: https://www.docker.com/ (elérés dátuma: 2022. júl. 3.)

[21] *Figma hivatalos dokumentáció.* URL: https://www.figma.com/ (elérés dátuma: 2022. júl. 9.)

[22] M. Fowler. *Patterns of Enterprise Application Architecture: Pattern Enterpr Applica Arch*. Addison-Wesley Signature Series (Fowler). Pearson Education, 2012. ISBN: 9780133065213. URL: https://books.google.ro/books?id=vqTfNFDzzdIC.

[23] *Git hivatalos dokumentáció.* URL: https://git-scm.com/ (elérés dátuma: 2022. júl. 18.)

[24] *GitLab hivatalos oldal.* URL: https://gitlab.com/ (elérés dátuma: 2022. júl. 9.)

[25] *GitLab pipeline hivatalos dokumentáció.* URL: https://docs.gitlab.com/ee/ci/pipelines/ (elérés dátuma: 2022. júl. 23.)

[26] *Google Maps hivatalos dokumentáció.* URL: https://developers.google.com/maps (elérés dátuma: 2022. júl. 10.)

[27] *Google OAuth 2.0 hivatalos dokumentáció.* URL: https://developers.google.com/identity/protocols/oauth2 (elérés dátuma: 2022. jún. 26.)

[28] Lokesh Gupta. *What is REST*. URL: https://restfulapi.net/ (elérés dátuma: 2022. jún. 20.)

[29] *Hash Table leírása a programiz weboldal jóvoltából.* URL: https://www.programiz.com/dsa/hash-table (elérés dátuma: 2022. júl. 19.)

[30] *HTTP dokumentáció - Mozilla*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP (elérés dátuma: 2022. júl. 16.)

[31] *Illustrated history of the thermostat*. URL: https://www.endesa.com/en/blogs/endesa-s-blog/air-conditioning/illustrated-history-of-the-thermostat (elérés dátuma: 2022. júl. 16.)

[32] *JavaScript hivatalos dokumentáció.* URL: https://www.javascript.com/ (elérés dátuma: 2022. jún. 27.)

[33] *JRE - Oracle*. URL: https://docs.oracle.com/goldengate/1212/gg-winux/GDRAD/java.htm#BGBFJHAB (elérés dátuma: 2022. jún. 20.)

[34] *JSON hivatalos dokumentáció.* URL: https://www.json.org/json-en.html (elérés dátuma: 2022. jún. 26.)

[35] *JWT token hivatalos dokumentáció.* URL: https://jwt.io/ (elérés dátuma: 2022. jún. 15.)

[36] *Kubernetes meghatározása - Red Hat*. URL: https://www.redhat.com/en/topics/containers/what-is-kubernetes (elérés dátuma: 2022. júl. 16.)

[37] *Linked List leírása a programiz weboldal jóvoltából.* URL: https://www.programiz.com/dsa/linked-list (elérés dátuma: 2022. júl. 19.)

[38] *Middleware meghatározás az IBM által.* URL: https://www.ibm.com/topics/middleware (elérés dátuma: 2022. jún. 25.)

[39] Chris Mooney. „Americans could save a fortune this winter — if they only understood their thermostats”. (2014). URL: https://www.washingtonpost.com/news/wonk/wp/2014/11/21/americans-could-save-a-fortune-this-winter-if-they-only-understood-their-thermostats/.

[40] *POER Smart Controls weboldal*. URL: https://www.poersmart.ro/ (elérés dátuma: 2022. aug. 25.)

[41] *PostgreSQL hivatalos dokumentáció.* URL: https://www.postgresql.org/ (elérés dátuma: 2022. jún. 27.)

[42] *RabbitMQ meghatározás - CloudAMQP*. URL: https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html (elérés dátuma: 2022. júl. 23.)

[43] *Rancher hivatalos dokumentáció.* URL: https://www.rancher.com/ (elérés dátuma: 2022. júl. 10.)

[44] *Raspberry Pi hivatalos dokumentáció.* URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (elérés dátuma: 2022. jún. 27.)

[45] *Raspberry Pi OS hivatalos dokumentáció.* URL: https://www.raspberrypi.com/software/ (elérés dátuma: 2022. júl. 1.)

[46] *React hivatalos dokumentáció.* URL: https://react.dev/ (elérés dátuma: 2022. jún. 30.)

[47] *React Native Elements hivatalos dokumentáció.* URL: https://reactnativeelements.com/ (elérés dátuma: 2022. júl. 29.)

[48] *React Native hivatalos dokumentáció.* URL: https://reactnative.dev/ (elérés dátuma: 2022. jún. 15.)

[49] *React Native MMKV hivatalos dokumentáció.* URL: https://github.com/mrousavy/react-native-mmkv (elérés dátuma: 2022. aug. 13.)

[50] *React Native Stack Navigator hivatalos dokumentáció.* URL: https://reactnavigation.org/docs/stack-navigator/ (elérés dátuma: 2022. júl. 11.)

[51] *SL-C relé részletes leírása a circuitbasict weboldaltól.* URL: https://www.circuitbasics.com/wp-content/uploads/2015/11/SRD-05VDC-SL-C-Datasheet.pdf (elérés dátuma: 2022. aug. 3.)

[52] *SMARTHER with Netatmo*. URL: https://www.poersmart.ro/ (elérés dátuma: 2022. aug. 25.)

[53] *Soket.IO hivatalos dokumentáció.* URL: https://socket.io/ (elérés dátuma: 2022. aug. 2.)

[54] *SQL injection leírása az imperva weboldal jóvoltából.* URL: https://www.imperva.com/learn/application-security/sql-injection-sqli/ (elérés dátuma: 2022. aug. 1.)

[55] *The WebSocket Protocol.* URL: https://www.rfc-editor.org/rfc/rfc6455 (elérés dátuma: 2022. jún. 23.)

[56] *TypeORM hivatalos dokumentáció.* URL: https://typeorm.io/ (elérés dátuma: 2022. aug. 5.)

[57] *TypeScript hivatalos dokumentáció.* URL: https://www.typescriptlang.org/ (elérés dátuma: 2022. jún. 18.)

[58] *Weather API hivatalos dokumentáció.* URL: https://openweathermap.org/api (elérés dátuma: 2022. júl. 12.)