

FELADATKIÍRÁS

A feladatkiírást a tanszéki adminisztrációban lehet átvenni, és a leadott munkába eredeti, tanszéki pecséttel ellátott és a tanszékvezető által aláírt lapot kell belefűzni (ezen oldal *helyett*, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell beleszerkeszteni ezt a feladatkiírást.



Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Leltározási feladatokat segítő, grafikus felülettel rendelkező adatbázis tervezése és fejlesztése

SZAKDOLGOZAT

Készítette
Király Bálint Martin

Konzulens
Schulcz Róbert

2020. december 8.

Tartalomjegyzék

Kivonat	i
Abstract	ii
1. Bevezetés	1
2. Feladat specifikáció	2
2.1. Funkcionális követelmények	2
2.1.1. Regisztráció	2
2.1.2. Bejelentkezés	2
2.1.3. Kijelentkezés	2
2.1.4. Raktár épületek kezelése	2
2.1.5. Tárolók kezelése	2
2.1.6. Eszközök kezelése	3
2.1.7. Raktár térképes nézettel	3
2.1.8. Tároló térképes nézettel	3
2.1.9. Kereshetőség	3
2.1.10. Naplózás	3
2.2. Nem funkcionális követelmények	3
2.2.1. Webes párhuzamos működés	3
2.2.2. Üzemeltetéssel szemben támasztott követelmények	4
3. Wireframe-ek	5
3.1. Bejelentkezés és regisztráció	5
3.2. Keresés	5
3.3. Raktár nézet	5
3.4. Tároló nézet	6
4. Választott technológiák	8
4.1. Frontend	8
4.1.1. Angular	8
4.1.2. React	8
4.1.3. Vue	8
4.1.4. Trendek	9
4.1.5. Konklúzió	9
4.2. Backend	10
4.2.1. NodeJS	10
4.3. Kommunikációs megoldások	10
4.3.1. REST API	11
4.3.2. GraphQL	11
4.3.2.1. Query	11

4.3.2.2.	Mutation	11
4.3.2.3.	Subscription	11
4.3.3.	Konkluzió	11
4.4.	Adatbázis	12
4.4.1.	Sequelize	12
4.4.2.	TypeORM	12
4.4.3.	Mongoose	12
4.4.4.	Prisma	12
4.4.5.	Konkluzió	12
4.5.	Közös technológiák	13
4.5.1.	TypeScript	13
5.	Tesztelés	14
5.1.	Frontend tesztelés	14
5.2.	Backend tesztelés	15
6.	Üzemeltetés	16
6.1.	Docker	16
7.	Összefoglalás	18
7.1.	Tovább fejlesztési lehetőségek	18
7.1.1.	Keresés	18
7.1.2.	Egyedi tulajdonságok kezelése	18
7.1.3.	3D-s térkép nézet	19
	Ábrák és táblázatok jegyzéke	20
	Irodalomjegyzék	20
	Függelék	22

HALLGATÓI NYILATKOZAT

Alulírott *Király Bálint Martin*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2020. december 8.

Király Bálint Martin
hallgató

Kivonat

Rengeteg raktárkezelő alkalmazás érhető el a piacon, azonban a térképes nézet egy ritka funkciónak számít ezekben a rendszerekben. Az erre alkalmas szoftverek rendszerint előredefiniált térképpel dolgoznak. A szakdolgozatom célja egy olyan raktár kezelő rendszer tervezése és fejlesztése, amely lehetővé teszi az eszközök pozíciójának pontos meghatározását a raktáron belül mindezt dinamikusan. A rendszer segítségével a felhasználók képesek a raktár méretét és elrendezését is meghatározni, így a pontos igényeknek megfelelő alkalmazást kaphatnak.

Abstract

There is a great deal of warehouse management applications available in the market, however, the map view is a rare feature in these systems. The warehouse management software usually wordks with a predefined maps. The aim of my dissertation is to design and develop a warehouse management system that allows to determine the exact position of items within the warehouse. With the help of the system, users can also set the size and layout of the warehouse, so they can get the application that suits their exact needs.

1. fejezet

Bevezetés

A félév során a feladatom egy olyan grafikus felülettel ellátott leltár rendszer tervezése és fejlesztése volt, melynek segítségével az alapvető leltári funkciókon felül könnyedén behatárolhatjuk a leltárba vett eszközök pontos helyzetét a raktárunkon belül egy grafikus “térkép” segítségével.

A dolgozatom során első körben egy részletes specifikációt készítettem, melyben taglaltam az alkalmazással szemben támasztott funkcionális és nem-funkcionális követelményeket. Ezután megterveztem a webalkalmazás felhasználói felületét, egyszerű wireframe-ek segítségével.

A specifikáció és a wireframe-ek elkészítése után kiválasztottam a használni kívánt technológiákat és megterveztem az alkalmazás architekturális felépítését.

Az alkalmazás két fő részből áll, frontend és backend. Utóbbi tartalmazza az üzleti logikát és az adatbázis kommunikációt, míg a frontend az adatok lekéréséért és megjelenítéséért felel, valamint a felhasználói interakciók által eljuttatja a módosításokat a backend részére. Ezek fejlesztését párhuzamosan végeztem. Minden egyes funkciónak először elkészítettem a backend oldali implementációját, majd hozzáálltam a frontend oldali kód fejlesztésének. Sok esetben szükséges volt a backend módosításra a frontend fejlesztése közben is.

Az alkalmazás tervezésén és fejlesztésén kívül az üzemeltetés előkészítését is elvégeztem, valamint bevezettem olyan megoldásokat, melyek a fejlesztés minőségét segítették elő.

Végül a tesztelésre fektettem a hangsúlyt, amely közben a felmerülő hibák javítását eszközöltem az alkalmazásban.

A dolgozat fejezeteivel próbáltam ezt a sorrendet tartani, hogy az olvasó számára is könnyedén követhető legyen a tervezés és fejlesztés folyamata valamint a köztük lévő összefüggés.

2. fejezet

Feladat specifikáció

A fejezet kitér az alkalmazással szemben támasztott funkcionális és nem funkcionális követelményekre.

2.1. Funkcionális követelmények

2.1.1. Regisztráció

Az elkészítendő alkalmazásban legyen lehetőség felhasználót létrehozni egy regisztrációs oldal segítségével. A regisztráció során a felhasználó nevét, email címét, jelszavát valamint a jelszavának megerősítését kérjük. További fontos követelmény, hogy egy email címhez csak egy felhasználó tartozhat, így az alkalmazásnak szükséges elvégezni ezt az ellenőrzést is.

2.1.2. Bejelentkezés

A rendszer csak autentikált felhasználók számára legyen elérhető. A regisztráció során megadott email cím és jelszó segítségével a látogatónak képesnek kell lennie autentikálnia magát, ezáltal hozzáférni az alkalmazásban tárolt eszközökhöz. Nem autentikált felhasználóknak csak a bejelentkezés és a regisztráció opciókat kínáljuk fel.

2.1.3. Kijelentkezés

A bejelentkezett felhasználónak biztosítsunk lehetőséget a munkamenet megszüntetésére, annak folytatásához csak újbóli bejelentkezéssel legyen lehetősége.

2.1.4. Raktár épületek kezelése

Az alkalmazással szemben követelmény, hogy képesnek kell lennie több raktár (raktár épület) kezelésére. Ez alatt értjük a raktár létrehozását, szerkesztését valamint az ezekhez tartozó jogosultságok menedzselését. A raktárról tároljuk a méreteit, a nevét és természetesen a szerkesztésre jogosult felhasználók listáját.

2.1.5. Tárolók kezelése

Minden raktárba tárolók helyezhetőek. A tárolókat a nevükkel, méretükkel és a raktáron belüli pozíciójukkal együtt rögzíthetjük. Amennyiben a felhasználó rendelkezik a megfelelő jogosultsággal az adott raktáron belül, legyen lehetősége a tárolók szerkesztésre, létrehozására és törlésére.

2.1.6. Eszközök kezelése

A hierarchia harmadik szintjén helyezkednek el az eszközök. Minden eszköz rendelkezzen az alábbi tulajdonságokkal. Név, amely az egyszerű azonosítást és a kereshetőséget biztosítja. Érték, ami az eszköz raktárba vétele kori értékét tartalmazza. Mérete és a tárolón belüli pozíciója. Minden egyes leltárba vett eszközhöz legyen lehetőségünk a kiadások rögzítésére. Minden kiadáshoz egy összeg és egy leírás tartozik, amely a kiadás okának és mértékének magyarázatára szolgál.

2.1.7. Raktár térképes nézettel

A raktárakban a tárolók elhelyezkedését jelenítsük meg egy felülnézeti, térképes nézet formájában is. A tárolók mozgását nem szükséges megvalósítani ezen a térképen. Ennek oka, hogy míg az eszközök pozíciója gyakran változik a tárolók fixen telepítve vannak. Ennek a funkciónak a nem implementálása felesleges félreértések elkerülését is szolgálja.

2.1.8. Tároló térképes nézettel

Minden tároló oldalán jelenítsünk meg egy képet a tároló tartalmával. A tárolt eszközöket egyszerű téglalappal reprezentáljuk. Fontos követelmény, hogy a ezen a nézeten legyen lehetőség az eszközök mozgatására is. A mozgatást a felhasználó a mozgatni kívánt elemre kattintva, majd az egér bal billentyűjét lenyomva tartva mozgathatja a tárolón belül. A fent leírt módszer segítségével legyen lehetőség egy ideiglenes tárolóba rakni. Ezt az ideiglenes tárolót jelenítsük meg minden (az adott raktárban lévő) tároló oldalán, az eszközök tárolók közötti mozgatását megvalósítva.

2.1.9. Kereshetőség

Az alkalmazásban legyen lehetőség a leltárba vett eszközök közötti keresésre. Fontos, hogy a keresés segítségével ne csak az adott elemet kapjuk meg hanem annak az elhelyezkedését is könnyedén le tudja kérdezni a felhasználó.

2.1.10. Naplózás

Egy alkalmazásnál általában fontos követelmény, hogy képesek legyünk nyomon követni az adatbázisba történt változások okait, különösen igaz ez egy raktár rendszer esetén. A rendszer naplózzon minden olyan eszközökkel kapcsolatos felhasználói interakciót, amely adatbázis művelethez vezet. Tehát az eszköz létrehozását, szerkesztését és törlését, a megleltetések naplózása nem szükséges.

2.2. Nem funkcionális követelmények

Az alkalmazással szemben természetesen nem csak funkcionális követelményeket támasztunk. A nem funkcionális követelmények figyelembevétele is legalább annyira fontosak egy alkalmazás tervezésénél és fejlesztésénél, mint a funkcionális követelmények.

2.2.1. Webes párhuzamos működés

Az elkészülő programmal szemben támasztott követelmények közé tartozik az egyidejűleg több felhasználó kiszolgálása webböngésző segítségével. Erre legyen lehetőség az összes modern böngészőből, azonban elegendő a böngészők legfrissebb változatának a támogatása.

2.2.2. Üzemeltetéssel szemben támasztott követelmények

Továbbá fontos követelmény, hogy az alkalmazás üzemeltetéséhez ne legyen szükség speciális hardware-re vagy speciális operációs rendszerre. Elindításához elegendőnek kell lennie egy átlagos személyi számítógép hardware készlete. A fejlesztést és üzemeltetést tegyük lehetővé Linux, MacOS és Windows operációs rendszereken is.

3. fejezet

Wireframe-ek

A wireframe-ek egy alkalmazás tervezésénél nagyon gyakran elmaradnak, pedig igenis fontos szerepük van. Rengeteg olyan dologra világíthatnak rá, amire egyébként nem gondolnánk és segíti a kommunikációt a fejlesztő(k) és a megrendelő között.

A dolgozatba csak a főbb képernyők wireframe-ét helyeztem el. Ezeknek az elkészítéséhez a Figma ¹. névre keresztelt webes alkalmazást használtam. Ennek segítségével az egyszerű wireframe-ektől kezdve komplex prototipizált design terveket is készíthetünk.

3.1. Bejelentkezés és regisztráció

A bejelentkezés és regisztráció oldalak (3.1. ábra) felépítése azonos, egyedül a beviteli mezőkben térnek el. A navigációs sávban csak a bejelentkezés és a regisztráció opciók közül választhatunk. A képernyő közepén mind a két esetben egy formot jelenítünk meg a szükséges adatokat bekérő beviteli mezőkkel.

3.2. Keresés

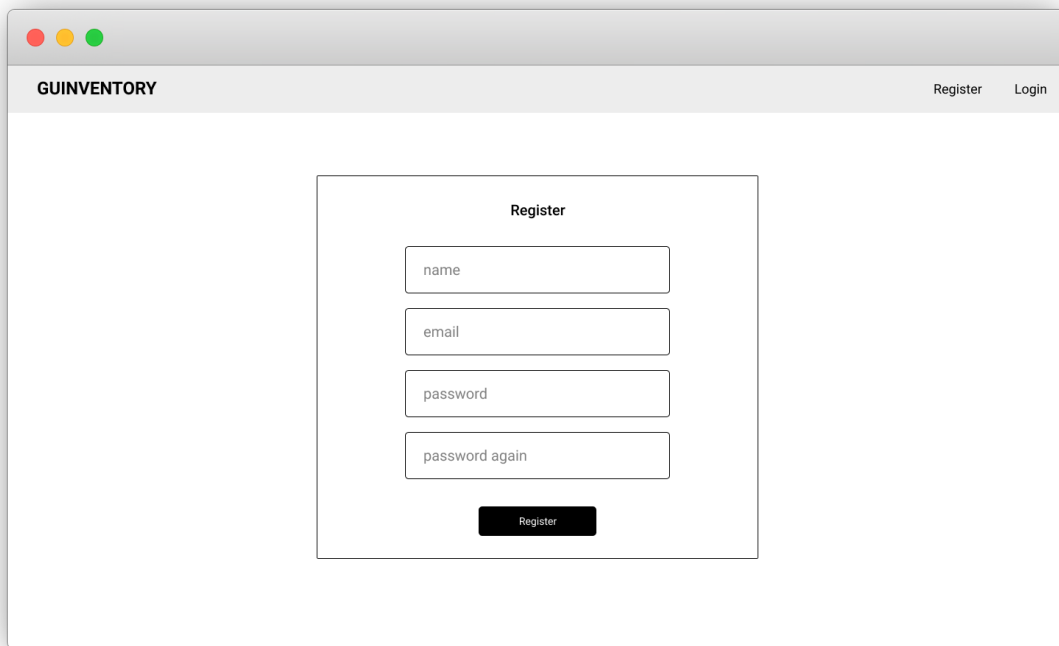
A keresést, annak érdekében, hogy az alkalmazás bármely részéről könnyedén elérhető legyen a felső navigációs sávba helyeztem el. A wireframe (3.2. ábra) alapján látszik, hogy kereséskor egy legördülő listában jelennek meg az eredmények, így az aktuális oldal elhagyása nélkül láthatjuk a keresett eszköz helyét.

3.3. Raktár nézet

A raktár oldalán (3.3. ábra) láthatunk egy térképes nézetet és egy listát is a tárolókról. A térképes nézeten a kurzort a tároló fülé mozgatva megjelenítjük annak nevét a könnyebb azonosítás érdekében.

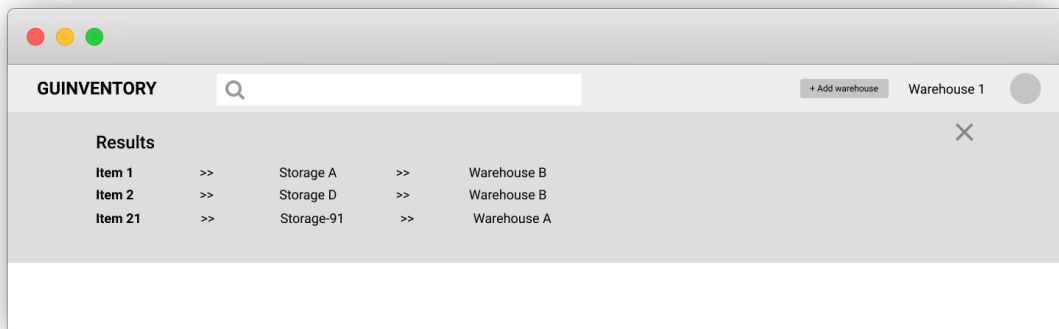
Ezen felül a navigációs sávban láthatjuk az éppen kiválasztott raktárat, ahol egy legördülő menü segítségével azonnal választhatunk másik raktárat is, amennyiben több raktárhoz is van hozzáférésünk. A raktár választó mellett megjelenítünk egy gombot, amellyel új tárolót hozhatunk létre.

¹A Figma hivatalos weboldala: <https://figma.com/>



The wireframe shows a web browser window titled "GUINVENTORY". In the top right corner, there are links for "Register" and "Login". The main content area features a centered "Register" form. This form contains four input fields stacked vertically, labeled "name", "email", "password", and "password again". Below these fields is a black "Register" button.

3.1. ábra. Regisztráció wireframe



The wireframe shows a web browser window titled "GUINVENTORY". It includes a search bar with a magnifying glass icon. To the right of the search bar are buttons for "+ Add warehouse" and "Warehouse 1", followed by a close button (X). Below the search bar, a "Results" section is displayed. It contains a table with three rows of data:

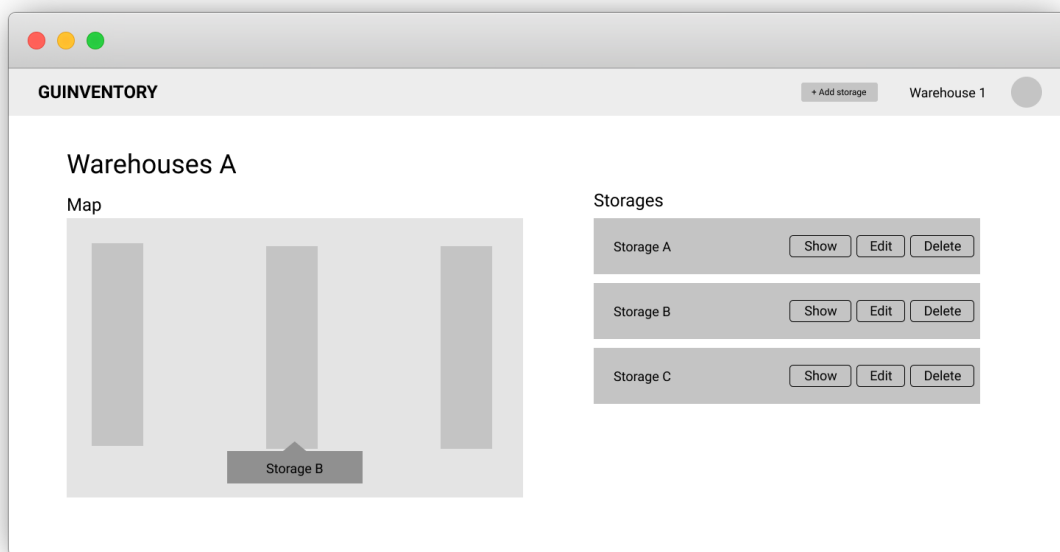
Results			
Item 1	>>	Storage A	>> Warehouse B
Item 2	>>	Storage D	>> Warehouse B
Item 21	>>	Storage-91	>> Warehouse A

3.2. ábra. Keresés wireframe

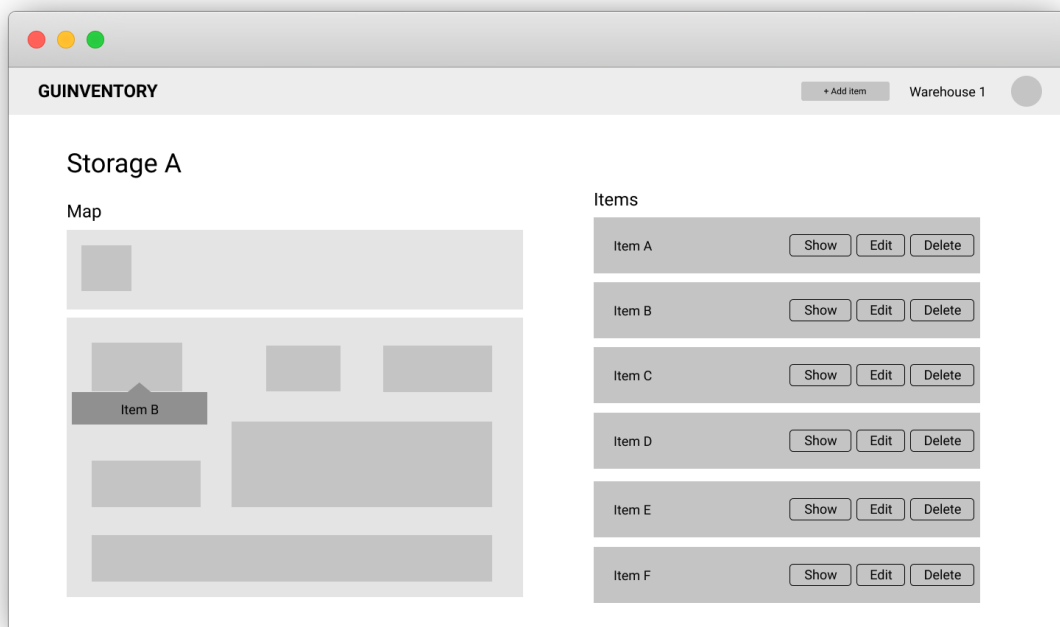
3.4. Tároló nézet

A tároló nézete (3.4. ábra) nagyon hasonlít a raktáréhoz, azonban itt kiegészítésként a térkép felett megjelenítünk egy másik tárolót is. Ez a feladatspecifikációban megkövetelt tárolók közötti eszköz mozgatását teszi lehetővé.

A navigációs sávban itt is megjelenítjük a raktár választó gombot, azonban a tároló létrehozása helyett, itt az eszköz felvétele gombot találhatjuk.



3.3. ábra. Raktár wireframe



3.4. ábra. Tároló wireframe

4. fejezet

Választott technológiák

Ennek a fejezetnek a keretein belül a felhasznált technológiák kiválasztásának szempontjait és folyamatát mutatom be. Először a frontend technológiát választottam ki ugyanis az alkalmazásnak ez a része a bonyolultabb a raktárak térképes nézetének kezelése miatt.

4.1. Frontend

A piacon jelenleg 3 meghatározó keretrendszer/könyvtár érhető el, melyek segítségével webes alkalmazások felhasználói felületét készíthetjük el.

Ezek az Angular, a React és a Vue. Bár mind a három keretrendszer célja ugyanaz mégis nagy eltéréseket tapasztalhatunk a kódbázisban, felépítésben és a fejlesztők filozófiájában.

4.1.1. Angular

Az Angular a Google által 2010-ben elindított keretrendszer, fejlesztésbe és karbantartásába azóta rengeteg a másik nagy cég is csatlakozott[?]. Filozófiája, hogy egy általános felhasználásra felkészített alapot ad a fejlesztők kezébe. Tartalmazza a formok validációját, az állapot kezelést, a routing-ot, a felhasználói input-ok kezelését és ezeken felül még rengeteg más olyan dolgot is, ami hasznos lehet egy webalkalmazás fejlesztéséhez.

4.1.2. React

Az Angular-hoz hasonlóan a React mögött is egy nagy cég áll. A Facebook 2013 óta fejleszti és tartja karban a keretrendszert[?]. Az Angular-ral szemben a React, filozófiája szerint csak egy könnyű súlyú keretet ad. Emiatt talán nem is nevezhetjük keretrendszernek, sokkal inkább csak egy könyvtár. Azonban ennek és a fejlesztők által implementált virtuális DOM-nak köszönhetően sokkal jobb sebesség érhető el vele. Természetesen az, hogy csak egy keretet ad nem okoz semmilyen hátrányt. Ugyanis rengeteg hivatalos csomag érhető el hozzá, melyek megvalósítják az Angular által is nyújtott megoldásokat.

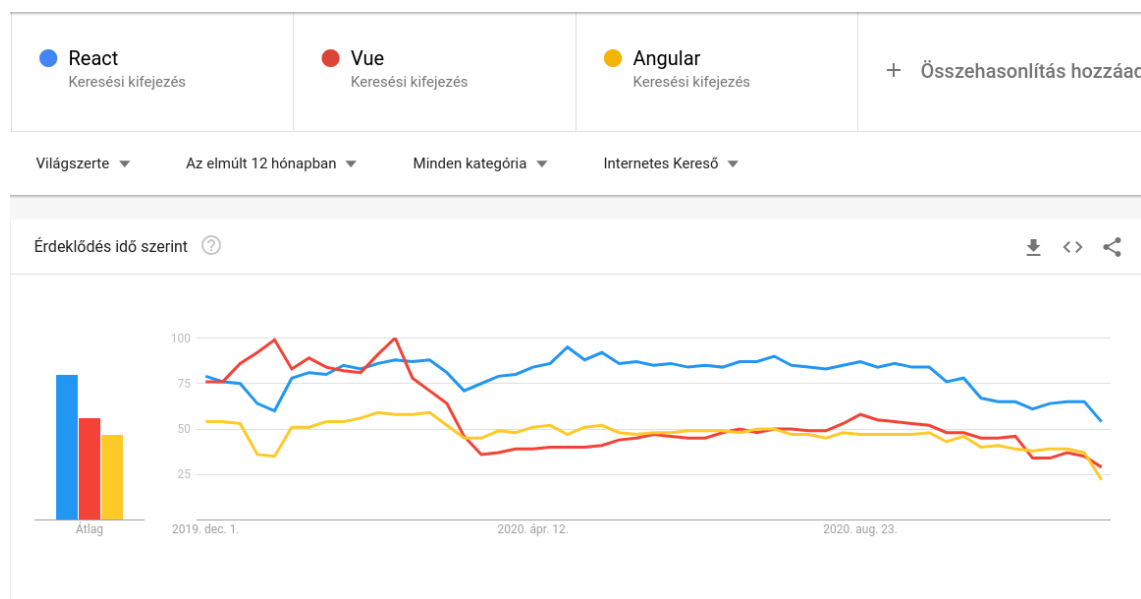
4.1.3. Vue

A három keretrendszer közül a legújabb és legkevesebb fejlesztői erőforrással rendelkező opció. A fejlesztését 2014-ben kezdte a Google egyik korábbi mérnöke, aki a mai napig szerves részét képezi a projektnek[?]. Filozófiáját tekintve a korábban tárgyalt két rendszer között helyezhető el. Többet tartalmaz, mint a React de korán sem annyit, mint az Angular. Itt is találkozhatunk a virtuális DOM-mal melynek köszönhetően nagyon gyors

a működése. Az összehasonlított keretrendszerek közül ezzel a legkönnyebb elkezdni a fejlesztést, egy egyszerű alkalmazás elkészítése nem igényel sok ismeretet a HTML, CSS és JavaScripten kívül. Természetesen komplex alkalmazásokat is készíthetünk vele és ipari környezetben is remekül megállja a helyét, azonban ilyen esetben komolyabb tervezést igényel ugyanis teljesen szabad-kezet kapunk az alkalmazás felépítését illetően.

4.1.4. Trendek

A döntés meghozása előtt végeztem egy kisebb kutatást a napjainkban tapasztalható trendekről. Ehhez a Google Trends¹ és az NPM Trends² szolgáltatásait vettem igénybe. Előbbi segítségével a Google keresések számát tudjuk összehasonlítani, míg utóbbival az NPM³ csomagkezelő oldalról történő letöltések számát.



4.1. ábra. Google Trends – keresések összehasonlítása.

A Google keresések alapján a korábbi években az Angular egyértelműen uralta a piacot, azonban a vezető szerepet mára már átvette tőle a React, ahogy az a grafikonon (4.1. ábra) is látszik.

A Google keresések nem mutattak szignifikáns különbséget, azonban az NPM letöltések számában már jelentős eltéréseket tapasztalhatunk. A grafikonról (4.2. ábra) könnyedén leolvasható, hogy több, mint 4-szer annyi a letöltések száma a React esetén, mint a másik 2 keretrendszerénél.

4.1.5. Konklúzió

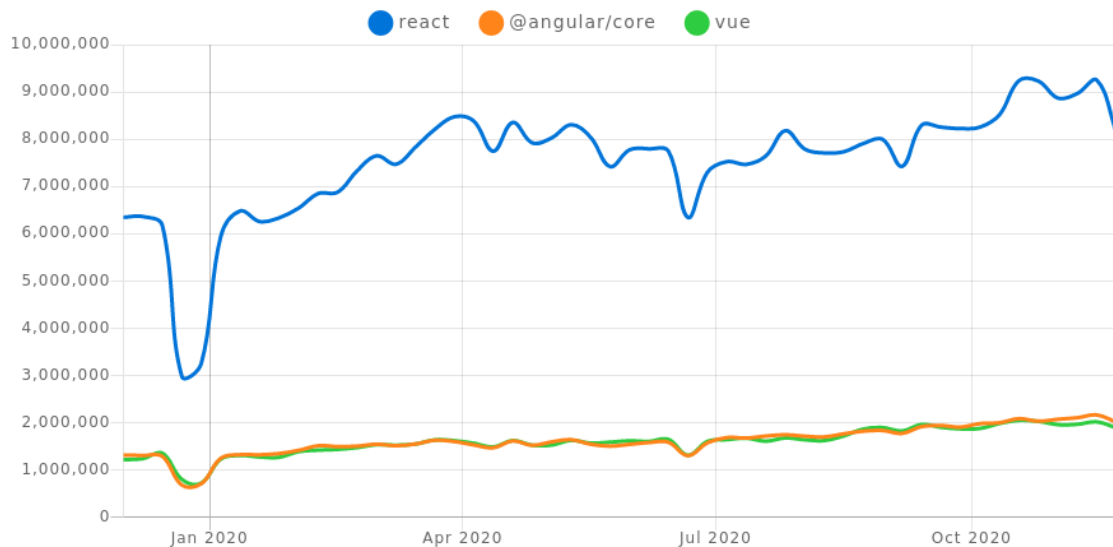
A fent felsoroltakat alapul véve végül a React-re esett a választásom. A virtuális DOM és az aktív fejlesztői közösség hatalmas előnyt jelent a fejlesztés és az üzemeltetése során is. A döntést továbbá segítette, hogy ezzel a keretrendszerrel rendelkezem tapasztalattal, korábbi fejlesztéseim során minden problémát remekül meg tudtam oldani a segítségével.

¹A Google Trends hivatalos weboldala <https://trends.google.com/>

²Az NPM Trends hivatalos weboldala <https://www.npmtrends.com/>

³Node Package Manager

Downloads in past 1 Year ▾



4.2. ábra. NPM Trends – letöltések összehasonlítása.

4.2. Backend

A kliens oldali technológia kiválasztása után a szerveroldali keretrendszer kiválasztására került sor. Itt sokkal több népszerű és az iparban is használt rendszer érhető el. Ilyen például a DotNet, a Spring, a Laravel, a Ruby on Rails, a Django és még sok más. A mennyiségük miatt ezeknek összehasonlítása nagyon nehézkes, így megpróbáltam egyszerűsíteni a folyamatot. Az elsődleges szempont a kiválasztás során az, hogy a választott frontend technológiával a legegyszerűbben és a legjobban tudjon együtt működni. Természetesen a felsorolt és a nem felsorolt keretrendszerek is működnek React-tel probléma nélkül. Azonban egy kiemelkedik közülük azáltal, hogy a programozási nyelve azonos a React-ével. A NodeJS segítségével frontend és backend oldalon is használhatunk JavaScriptet. Az azonos programozási nyelv felveti a kódmegosztás lehetőségét is a két komponens között.

4.2.1. NodeJS

A NodeJS története több, mint 11 éve indult Ryan Dahl keze által. A projekt a Google által fejlesztet V8 JavaScript motor segítségével teszi lehetővé JavaScript futtatását web böngészőn kívül, így lehetővé téve a nyelv felhasználását backend oldalon is. A NodeJS gyors ütemben fejlődött, 2011-ben már a Microsoft is kivette a részét a fejlesztésből napjainkra pedig az egyik legnépszerűbb technológia webes környezetben.[?]

4.3. Kommunikációs megoldások

A frontend és a backend technológiák kiválasztása után a következő lépés a köztük történő kommunikáció mikéntjének eldöntése volt. Itt szerencsémre sokkal kevesebb opció közül kellett választanom. Napjainkban két fő irányvonal figyelhető meg. Ezek a REST API és a GraphQL.

4.3.1. REST API

A REST feloldása REpresentational State Transfer, ami magyarra fordítva Reprezentatív Állapot Átvitelt jelent. Ez – ahogy a nevéből is következtetni lehet – próbálja kifejező módon átvinni az adatot a kliens és a server alkalmazások között. Ezt úgy valósítja meg, hogy ajánlást tesz a végpontok nevére és típusára rendeltetésük szerint.

Művelet angolul	Művelet magyarul	HTTP üzenet típusa	Végpont
Create	Létrehozás	POST	/users
Read	Megtekintés	GET	/users/:id
Update	Módosítás	POST	/users/:id
Delete	Törlés	DELETE	/users/:id
List	Listázás	GET	/users

4.3. táblázat. Példa egy entitáson végezhető műveletekre a REST API elvei szerint

4.3.2. GraphQL

A GraphQL egy lekérdező nyelv, amely a jelenleg elterjedt REST API-s megoldásokat próbálja leváltani/kiegészíteni. A megszokott REST API-val ellentétben GraphQL-nél csak egyetlen egy végpont létezik, valamint csak POST típusú HTTP kéréseket használunk.

Az összes kérést erre a végpontra küldjük a megfelelő tartalommal, melyet a POST kérés törzsében (body) helyezünk el.

A bevett REST API-s megoldással szembeni hatalmas előnye, hogy mindig azt kapjuk amit kérünk. A POST kérés törzsében elhelyezett GraphQL operation pontosan meghatározza, hogy milyen entitások milyen tulajdonságait szeretnénk visszakapni. Ez a GraphQL operation nagyon hasonlít a JSON formátumra, azonban egy-két dologban eltér attól. Lehetőségünk van több entitásból is adatot lekérni egyetlen kéréssel, így csökkentve a HTTP üzenetek számát.

A kéréseket minden esetben egy (vagy több) úgynevezett resolver szolgálja ki nekünk. A resolver-ekből 3 fő típust különböztetünk meg Query, Mutation és Subscription. A REST API elveihez hasonlóan a GraphQL esetén is kapunk ajánlásokat arra, hogy milyen adatbázis művelethez milyen típusu resolvert használjunk.

4.3.2.1. Query

Adatok lekérésére szolgál.

4.3.2.2. Mutation

Ahogy a nevéből is következtethetünk rá főként adatok módosítására és létrehozására szolgál.

4.3.2.3. Subscription

A standard GraphQL implementáció tartalmazza a websocket kommunikációt is. A subscription-ök segítségével lehetősége van a kliensnek feliratkozni bizonyos eseményekre, melyek bekövetkeztéről azonnal értesül socket kapcsolaton keresztül.

4.3.3. Konklúzió

A fent leírt szempontokat figyelembe véve a végső választásomat a GraphQL mellett tettem le. Tanulmányaim során rengetegszer találkoztam REST API-t használó vagy

annak megvalósítását követelő feladattal, így ezen opció választása esetén nem mélyítettem volna el a tudásomat egy kevésbé ismert, azonban mégis remek technológiában.

4.4. Adatbázis

Az alkalmazás nem rendelkezik olyan követelménnyel, amely komoly adatbázis műveletet igényel. Ezért a választás szempont elsődlegesen az volt, hogy a már kiválasztott technológiákkal együtt a lehető legkényelmesebb és legjobb fejlesztési élményt nyújtsa. Ennek következménye az volt, hogy adatbázisok helyett ORM⁴ rendszerek összehasonlítását végeztem el. Az ORM egy absztrakciós réteget helyez az adatbázis és az alkalmazás közé, így elfedve a lekérdező nyelvet, ez nagyobb biztonságot és gyorsabb fejlesztést eredményez.

Az ORM-ek többsége rengeteg dologban hasonlít, így az összehasonlítás során leginkább a különbségekre fókuszáltam.

4.4.1. Sequelize

Több, mint 7 éve elérhető ORM rendszer, az ötös verziótól beépített TypeScript támogatással érkezik. A sequelize-cli segítségével generálhatunk modelleket és migrációkat is.[?]

4.4.2. TypeORM

Fejlesztése 5 éve kezdődött, azonban a mai napig sem érte el az 1.0-ás verziót. Beépített TypeScript támogatással rendelkezik és a Sequelize-hoz hasonlóan generálhatunk modelleket és migrációkat.[?]

4.4.3. Mongoose

A Mongoose ORM a mongoDB kezeléséhez több, mint 10 évvel ezelőtt létrehozott csomag. Mivel a mongoDB egy document alapú NoSQL adatbázis, így még fontosabb az adatok validációja.[?] A mongoose séma természetesen kezeli ezt, azonban a migrációk meglévő adatok esetén problémás és sok hibalehetőséget rejt. A TypeScript támogatás megoldható de szintén körülményes.

4.4.4. Prisma

2020 júniusában jelentették be a 2.0-ás verziót, amely az 1.0 alapoktól újraírt változata. Teljes körű TypeScript támogatás érkezik és rengeteg extra funkcióval. A migrációs generálása automatikus, így a séma módosítása után automatikusan generálható bármilyen fejlesztői beavatkozás nélkül. Természetesen ha adatmigráció is szükséges azt nekünk kell kezelniük. A Prisma egy beépített Studio nevű grafikus interface-szel érkezik, így teljes grafikus hozzáférést kapunk az adatbázishoz bármilyen külső megoldás nélkül.

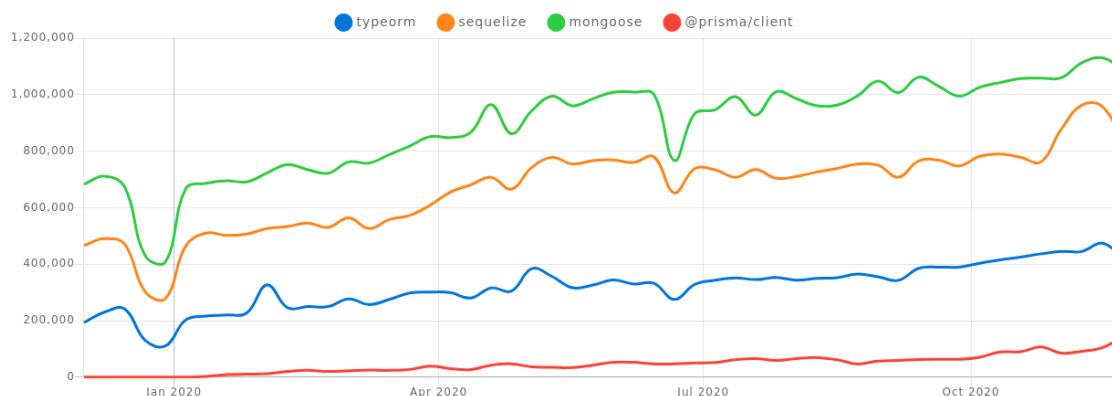
4.4.5. Konklúzió

Habár a trendek (4.4. ábra) alapján a Prisma jócskán el van maradva népszerűségben az összehasonlításban részt vett többi ORM-től mégis erre esett a választásom. A korábban felsorolt előnyök és a növekvő népszerűség miatt úgy érzem, hogy a lemaradása egyedül az újdonságának köszönhető.

⁴Object-Relational mapping.

Tekintve, hogy fejlesztése még mindig egy korai stádiumban van az adatbázis kezelők listája szűkös. A fejlesztés kezdetekor csak PostgreSQL, MySQL és SQLite volt támogatott, utóbbi kisebb hiányosságokkal. A dolgozat írása közben bejelentették a MSSQL támogatást is. A választáskor a MySQL és PostgreSQL között kellett döntenem. Az adatbázisom bonyolultsága nem lesz túl nagy, így személyes preferencia alapján a PostgreSQL mellett döntöttem.

Downloads in past 1 Year ▾



4.4. ábra. NPM Trends – ORM-ek letöltésének összehasonlítása.

4.5. Közös technológiák

Ahogy azt a korábbi fejezetben is említettem a NodeJS-nek hála a backend és a frontend oldali alkalmazás azonos nyelvet használ, a JavaScriptet. A JavaScript egyik nagy hátránya az, hogy gyengén típusos nyelv (természetesen ezt bizonyos esetekben tekinthetjük előnynek is). Ennek a megoldására JavaScript helyett TypeScriptet használtam az alkalmazás megvalósításához.

4.5.1. TypeScript

A TypeScript egy – a Microsoft által fejlesztett – nyílt forráskódú nyelv, amely JavaScriptet egészíti ki statikus típus definíciókkal. Mondhatjuk, hogy a JavaScript egy superset-je.

A típusok segítségével hamarabb észrevehetjük a hibákat az alkalmazásunkban. Azonban fontos megjegyezni, hogy a típusok definiálása opcionális, ezért TypeScript mellett érdemes valamilyen linter-t használni, amely figyelmezteti a programozót ha elmulasztja a típusdefiníciók használatát. Minden érvényes JavaScript kód egy érvényes TypeScript kód is, ez részben az elhagyható típusdefiníciók miatt igaz.

Annak érdekében, hogy probléma nélkül futtathassuk a TypeScript kódunkat a böngészőkben minden kódot JavaScript-re transzformálunk. Erre több megoldás is létezik, ilyen például a Babel vagy a TypeScript compiler.[?]

A NodeJS-nek köszönhetően használhatjuk backend oldali nyelvként is, így a frontend és a backend közös nyelvet használhat, amely akár a kódmegosztás lehetőségét is felveti.

5. fejezet

Tesztelés

Az alkalmazás működésének validációjában elengedhetetlen lépés a tesztelés. A tesztelés ezen felül segíti a fejlesztő munkáját is, bármilyen aprónak tűnő változtatás olykor hatással lehet az alkalmazás más részeire is. Előfordulhat, hogy már meglévő és működő funkciók válnak használhatatlanná új funkciók bevezetése közben. Ennek a kockázatát megfelelő tesztlefedettséggel minimálisra csökkenthetjük.

5.1. Frontend tesztelés

A frontend teszteléséhez úgy nevezett end-to-end tesztek készítését. Az E2E¹ tesztek esetén a teszt a felhasználó viselkedését szimulálja. Egy szimulált böngészőben végzi el a tesztet és tényleges kattintás eseményt vált ki majd figyeli a renderelt képet a tesztnek megfelelő egyezést keresve.

Ezt a Jest² és a Playwright³ package-ek segítségével valósítottam meg. A tesztelés során szükség van bizonyos adatbázis adatokra, ezeket környezeti változókba helyeztem. Jelenleg a tesztek csak lokális környezetben futnak, de a későbbi fejlesztések során így biztosítva lesz, hogy adatbázisból adatok ne kerüljenek harmadik fél kezébe.

Az alábbi példában (5.1. szakasz) a bejelentkezés E2E tesztje látható. A bejelentkezési oldalra navigálunk, ahol kitöltjük az email és jelszó mezőket majd elküldjük a formot. A teszt rövid várakozás után ellenőrzi, hogy megtörtént-e az átirányítás és a renderelt oldal tartalmazza-e a Warehouse szöveget egy h2 HTML tag-ben.

```
1 it('should allow users to sign in', async () => {
2   await page.goto('http://localhost:3000/auth/login')
3   await page.waitForTimeout(1000)
4   await page.fill('[name="email"]', process.env.TEST_USERNAME)
5   await page.fill('[name="password"]', process.env.TEST_PASSWORD)
6   await page.click('[type=submit]')
7
8   await page.waitForTimeout(1000)
9   await expect(page.url()).toBe('http://localhost:3000/')
10  await expect(page).toHaveText('h2', 'Warehouses')
11 })
```

5.1. kódrészlet. Bejelentkezés E2E teszt

¹end-to-end

²Jest NPM oldala <https://www.npmjs.com/package/jest>

³Playwright NPM oldala <https://www.npmjs.com/package/playwright>

5.2. Backend tesztelés

A Prisma fejlesztő csapata csupán pár hónappal ezelőtt jelentette be a 2.0-ás verziót. A folyamatos fejlesztés ellenére is még hiányos az eszközkészlete. Ennek köszönhetően teszteléshez sem kínál semmilyen megoldást.

A tesztelési környezet kialakításához egy SQLite adatbázist szerettem volna használni, hogy ne a fejlesztés közben használt adatbázist használjam és ne teljen túl sok időbe a tesztek futtatása. Azonban a Prisma jelenlegi verziója nem támogatja az enumerációt SQLite adatbázis esetén. Emiatt itt is egy PostgreSQL adatbázissal kellett dolgoznom. A tesztelés előtt egy scripttel automatikusan létrehozom a szükséges adatbázist, majd a tesztek végeztével törölöm azt, így biztosítva, hogy véletlenül se maradjon semmilyen adat az adatbázisba, ami esetleg fals eredményt váltana ki a tesztekben.

A tesztek során a GraphQL végpontot hívtam meg különböző query-vel és mutation-ökkel, majd a választ vizsgálva megbizonyosodtam a helyes működésről.

```
1 test('successfully register a user', async () => {
2   const data = {
3     name: 'User Name',
4     email: 'test.user@example.org',
5     password: 'password',
6   }
7   const req: any = await request(config.url, register, data)
8
9   expect(req).toHaveProperty('register')
10  expect(req.register.user.email).toEqual(data.email)
11 })
```

5.2. kódrészlet. Regisztráció első teszt eset

6. fejezet

Üzemeltetés

Korábbi fejezetben írtam arról, hogy az üzemeltetést Heroku és Vercel segítségével oldottam meg. Azonban ez az alkalmazás tényleges üzembe helyezése után ez nem feltétlenül a legköltséghatékonyabb módja az üzemeltetésének. Ezért igyekeztem alternatívákat kínálni az esetleges üzembehelyezőknek.

6.1. Docker

Az üzemeltetés megkönnyítése érdekében minden komponenshez készítettem egy-egy dockerfile-t és docker-compose file-t.

A frontendhez tartozó Docker felépítése három lépésből áll. Első lépésben a szükséges függőségeket telepítjük a yarn csomagkezelő segítségével. Második lépésben a TypeScript kódot fordítjuk JavaScript kódra, a harmadik lépésben pedig egyszerűen elindítjuk az alkalmazást.

A három lépéses konténer készítés oka a docker cachelésének maximális kihasználása. Ez különösen nagy segítség lehet, ha ugyan azt a verziót több környezetbe is szeretnénk élesíteni.

```
# 1 - Install dependencies
FROM node:12-alpine AS dependencies

WORKDIR /app
COPY package.json yarn.lock ./
RUN yarn install

# 2 - Build
FROM node:12-alpine AS build

ENV NODE_ENV=production
WORKDIR /app
COPY . .
COPY --from=dependencies /app/node_modules ./node_modules
RUN yarn build

# 3 - Run
FROM node:12-alpine AS run

WORKDIR /app
ENV NODE_ENV=production
COPY --from=dependencies /app/node_modules ./node_modules
COPY --from=build /app/public ./public
COPY --from=build /app/next.config.js ./
COPY --from=build /app/.next ./next
CMD ["node_modules/.bin/next", "start"]
```

6.1. kódrészlet. Frontend Dockerfile

A backendhez tartozó container sokkalta egyszerűbb. A hozzá tartozó docker-compose file tartalmazza a backend és az adatbázis elindítását, valamint a közöttük lévő kapcsolat kiépítését és a Prisma Studio indítását is. A Prisma Studio és az alkalmazás konkurensen futnak egy containeren belül. Fontos megjegyezni, hogy a Studiot csak feltétlenül szükséges esetben használjuk, érdemes tiltani a hozzáférést ugyanis nem tartalmaz semmilyen beépített jelszavas védelmet.

```
version: '3'
services:
  server:
    build: .
    links:
      - postgres
    depends_on:
      - postgres
    ports:
      - '${SERVER_PORT}:4000'
      - '${STUDIO_PORT}:5555'
    environment:
      NODE_ENV: ${NODE_ENV}
      DATABASE_URL: ${DATABASE_URL}
      APP_SECRET: ${APP_SECRET}
    networks:
      - server-network
  postgres:
    image: postgres:12.5
    restart: always
    environment:
      POSTGRES_USER: prisma
      POSTGRES_PASSWORD: prisma
      POSTGRES_DB: prisma
    networks:
      server-network:
        aliases:
          - postgresql.db
    volumes:
      - db_data:/var/lib/postgresql/data
volumes:
  db_data:
networks:
  server-network:
```

6.2. kódrészlet. Backend docker compose

7. fejezet

Összefoglalás

A fejlesztés során rengeteg olyan problémát kellett megoldanom, melyekkel egyébként nem találkoztam volna. Elmélyültem olyan technológiákban, melyek napjainkban divatosak és piaci környezetben is megállják a helyüket. Ezeket a későbbiekben alkalmazhatom, mint tanulmányaimban, mint munkám során.

7.1. Tovább fejlesztési lehetőségek

A félév során temérdek új fejlesztési lehetőség jutott eszembe, melyeket megvalósítva egy még jobb és még inkább a piaci igényeknek megfelelő alkalmazást kaphatunk. A szakdolgozat keretein belül igyekeztem az elengedhetetlen funkciókat a lehető legjobban megvalósítani és inkább a fejlesztői eszköztár felépítését tartottam fontosnak, mint a rengeteg funkció belezsúfolását egy alkalmazásba. Ennek köszönhetően remélhetőleg egy hosszú távon is fejleszthető és fenntartható alkalmazás született. A félév végeztével folytatnám a munkát, hogy a lehető legtöbb piaci igényt legyen képes kiszolgálni az alkalmazásom.

7.1.1. Keresés

A jelenlegi keresés egy nagyon egyszerű string összehasonlításra alapul, már egyetlen karakter elgépelése esetén sem talál egyezést. Ennek a problémának a megoldására több lehetőség is kínálkozik, ilyen például az Elastic Search vagy a PostgreSQL-be épített full-text search. Ezeknek az alapvető működési elve, hogy nem magában az adatbázisban keres hanem létrehoz egy index-szelt szótárat és ezt hasonlítja a keresési kifejezéshez. Ennek köszönhetően nagy adatbázisok esetén is gyors keresés érhető el.

7.1.2. Egyedi tulajdonságok kezelése

Már a félév elején felvetődött a raktárban tárolt eszközök egyedi tulajdonságainak kezelése, mint ötlet. A tervezés során kiderült, hogy ez jóval több időt venne igénybe, mint azt az elején gondoltam, így a megvalósítását kihagytam a szakdolgozatból.

A koncepció lényege, hogy kategóriákat hozhattunk létre minden raktáron belül. A kategóriákhoz egy név megadása után felvehetőek a tulajdonságok nevei és típusai. Ezután az eszköz felvételekor kiválasztjuk, hogy milyen kategóriába, kategóriákba tartozik. Így a kategóriák révén már tudjuk azt, hogy milyen egyedi tulajdonságai lehetnek. Természetesen ehhez szükséges előredefiniált attribútum típusok létrehozására is.

7.1.3. 3D-s térkép nézet

Az alkalmazás már a jelenleg verziójában is tárolja a 3 dimenziós adatokat, azonban ez a megjelenítésben figyelmen kívül lett hagyva. A későbbi fejlesztések során a jelenlegi térkép helyett egy 3 dimenziós nézet segítségével lehetőség nyílna az eszközök a valósághoz még közelebb eső állapotának tárolására egy 3 dimenziós nézet segítségével. Ennek megvalósítása azonban nagyon sok problémát vet fel, így az implementációja rengeteg időt emésztethet fel.

Ábrák és táblázatok jegyzéke

3.1. Regisztráció wireframe	6
3.2. Keresés wireframe	6
3.3. Raktár wireframe	7
3.4. Tároló wireframe	7
4.1. Google Trends – keresések összehasonlítása.	9
4.2. NPM Trends – letöltések összehasonlítása.	10
4.3. Példa egy entitáson végezhető műveletekre a REST API elvei szerint	11
4.4. NPM Trends – ORM-ek letöltésének összehasonlítása.	13

Irodalomjegyzék

Függelék