**Záródolgozat feladatkiírás**

Tanulók neve: Torma Erik, Puskás Máté, Takács Adrián

Képzés: nappali

Szak: 5 0613 12 03 Szoftverfejlesztő és tesztelő technikus

**A záródolgozat címe:**

**„Nagyteveli Tó” weboldala**

Konzulens: Horváth Norbert

Beadási határidő: 2022.04.29.

Győr, 2022. 04. 15

**Módos Gábor**igazgató

**Konzultációs lap**

|  |  |  |  |
| --- | --- | --- | --- |
|  | A konzultáció | | Konzulens aláírása |
| ideje | témája |
| 1. | 2022.02.15. | Témaválasztás és specifikáció |  |
| 2. | 2022.03.14. | Záródolgozat készültségi fokának értékelése |  |
| 3. | 2022.04.17. | Dokumentáció véglegesítése |  |

**Tulajdonosi nyilatkozat**

Ez a dolgozat a saját munkánk eredménye. Dolgozatunk azon részeit, melyeket más szerzők munkájából vettünk át, egyértelműen megjelöltük.

Ha kiderülne, hogy ez a nyilatkozat valótlan, tudomásul vesszük, hogy a szakmai vizsgabizottság a szakmai vizsgáról kizár minket és szakmai vizsgát csak új záródolgozat készítése után tehetünk.

Győr, 2022. április 15.

Tanuló aláírása

Tanuló aláírása

Tanuló aláírása

Tartalomjegyzék

[1. Bevezetés 3](#_Toc102055281)

[2. Specifikáció 4](#_Toc102055282)

[2.1. Szoftver alapvető jellemzői 4](#_Toc102055283)

[2.2. Szoftver funkciói 5](#_Toc102055284)

[3. Alkalmazott technológiák 5](#_Toc102055285)

[3.1 Fejlesztői környezet 8](#_Toc102055286)

[3.2. A node js felépítése 9](#_Toc102055287)

[3.2. A vue js felépítése 14](#_Toc102055288)

[4. Felhasználói dokumentáció 18](#_Toc102055289)

[4.1. Telepítés 18](#_Toc102055290)

[4.2 A weboldal fő jellemzői 19](#_Toc102055291)

[5. Fejlesztői dokumentáció 22](#_Toc102055292)

[5.1. Első lépések 22](#_Toc102055293)

[5.2. Views 23](#_Toc102055294)

[5.3. Routes 24](#_Toc102055295)

[5.4. controllerek és a modellek 26](#_Toc102055296)

[5.5. Auth (login, sign up) 26](#_Toc102055297)

[5.6. Admin 28](#_Toc102055298)

[5.7. Adatbázis 29](#_Toc102055299)

[5.8. Munkamegosztás 30](#_Toc102055300)

[6. Tesztelés 34](#_Toc102055301)

[7. Összefoglalás 35](#_Toc102055302)

[7.1. jövőbeli tervek 35](#_Toc102055303)

[8. Felhasznált források 36](#_Toc102055304)

# 1. Bevezetés

Egy fizikai üzlet megnyitásához legyen akár egy kis bolt, vagy nagyobb kereskedelmi egység számos feltételnek kell megfelelni. Ahhoz, hogy az üzlet megkapja a működési engedélyt, számos adatot szükséges megadni, és jó néhány előírásnak megfelelni. Ha mindez megvan, több hatóság engedélye szükséges, melyek ellenőrzik a helyet, feltételeket, adatokat, nem csak induláskor, hanem később is. Számtalan szabály vonatkozik a nyitva tartásra, a dolgozókkal kapcsolatos előírások betartására, a környezettel, technikai feltételekkel, és számos más külső-belső tényező meglétével. Egy online webáruház esetén ezek nagy része nem játszik szerepet, így ég és föld mind a megfelelés, mind a fenntartás nehézségeit tekintve, ezért választottuk a web shop készítést.

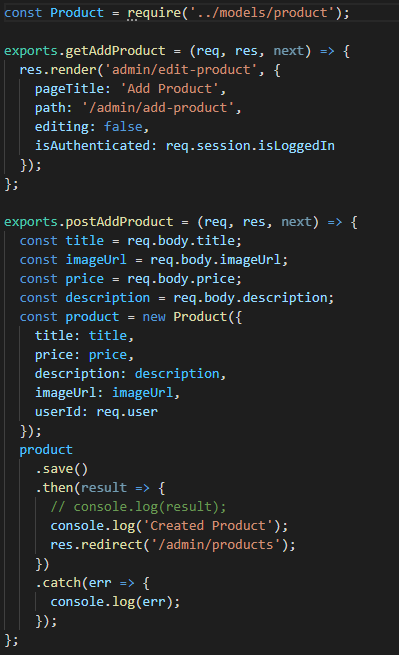
# 2. Specifikáció

A Szakdolgozatunk egy webshopból áll. A kezdéshez először megkellett beszélnünk pontosan milyen weboldalt szeretnénk csinálni. A weboldalt milyen frontend illetve backendből alakítsuk ki. Végül a választásunk egy hozzánk közel eső falunak tavára esett Nagytevelre, amit mi is rendszeresen látogatunk, de a jegy vásárlásra, és bérlésre csak a helyszínen van lehetőség. A webshopot a VUE JS illetve a MONGODB-ben írjuk meg. Ezen programok futtatásához a Visual Studio Code szükséges.

## 2.1. Szoftver alapvető jellemzői

* Szoftver elnevezése, címe „Nagyteveli Tó”
* Szoftver kategóriája: weboldal
* Szoftver alapvető funkciója: Lehetőség online termék vásárlás, bérlésre, és jegy vásárlásra
* Fejlesztői környezet: VUE JS és NODE JS
* Programozási nyelvek: Java script

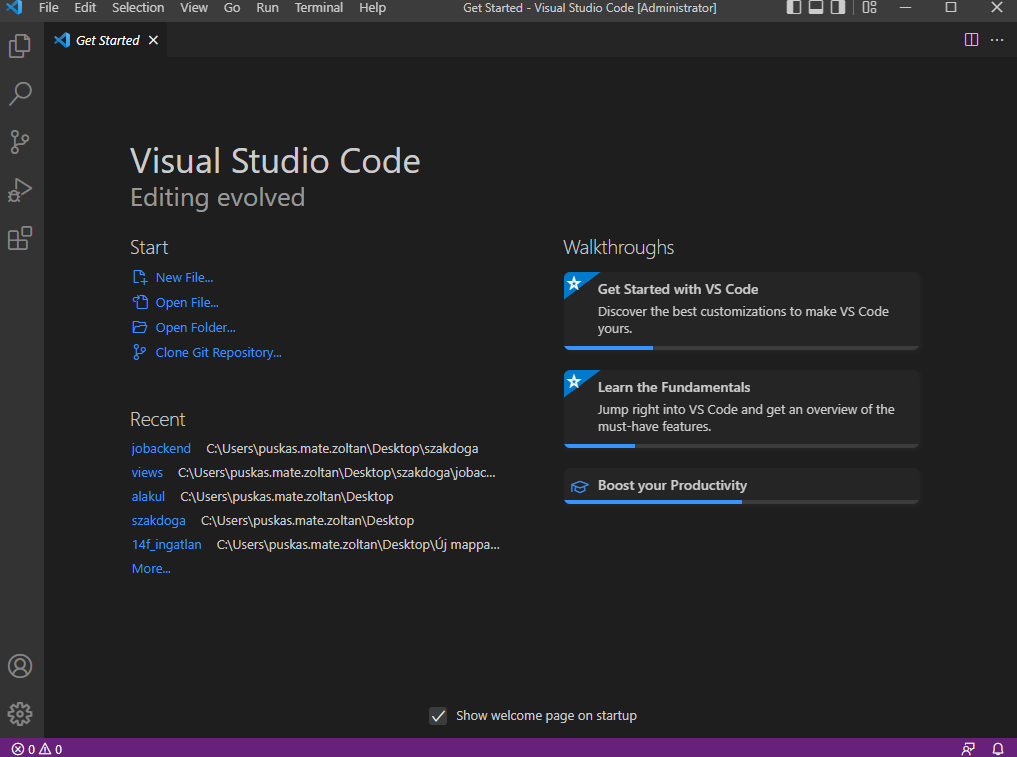
## 2.2. Szoftver funkciói

A weboldalon a Nagyteveli tóról olvashat információkat a felhasználó, térképpel lesz ellátva, hogy ez is segítése a felhasználót a tó könnyebb megtalálásához. Majd a felhasználónak regisztrálni kell a webshopra, hogy lehetősége legyen bármi féle terméket vásárolni. Regisztráció után kapunk emailben egy értesítést a felhasználóról, neki pedig lehetősége van bejelentkezni, hogy hozzá férjen a termékéhez. A termékeknél pontos leírást lát mi mennyibe kerül, majd a kosár fülön belül egy összeget az általa vásárolt termékekre.

Ezeken kívül vagy egy admin felület is az oldalon, ide kizárólag az admin jogú felhasználó tud bejelentkezni. Bejelentkezés után látja az admin oldalakat is, itt lehetősége van az összes terméken árat, termék nevet, képet módosítani tudja, valamint a új termék hozzá adásra és régi törlésére is van lehetősége

# 3. Alkalmazott technológiák

VISUAL STUDIO CODE: A Visual Studio Code (VS Code) egyszerű, de hatékony forráskódszerkesztő, amely az asztalon fut, és Windows-, macOS- és Linux-rendszereken érhető el. Beépített a JavaScript-, a TypeScript- és a Node.js-támogatással rendelkezik, és gazdag ökoszisztémával rendelkezik más nyelvek (például C++, C#, Java, Python, PHP és Go) és futtatókörnyezet (például .NET és Unity) bővítményei számára. További információkért lásd: Első lépések a VS Code szerkesztővel. A VS Code lehetővé teszi, hogy bővítményekkel terjessze ki a képességeit. A VS Code-bővítmények további funkciókat adhatnak hozzá a teljes élményhez. A funkció kiadásával mostantól a VS Code bővítményt is használhatja a Power Apps-portálokkal történő munka során.



VUE JS: Manapság minden egyes weboldalnak, webáruháznak szüksége van a megfelelő böngészőoldali élmény, valamint funkcionalitás megteremtésére. Ez alapvető fontosságú: ugyanis ezek nélkül az adott online megjelenés nem fogja azokat az eredményeket hozni, amiket szeretnénk, és elvárnánk. A Vue.js ebben tud nekünk segíteni. Aki dolgozott már JavaScripttel, az tisztában van vele, hogy egy viszonylag könnyen használható scriptnyelv, viszont hosszú távon natív JavaScriptet használni-na, ez sokak szerint nem igazán jó ötlet. Gondolhatunk itt számos különféle funkcionalitásra: egy felugró ablak megnyitása, bizonyos elemek elrejtése adott feltételek mellett: mindez szimpla JavaScripttel megoldva több időbe telhet. Akkor pedig, ha komplexebb applikációról lenne szó, egészen biztosan rosszul járunk. A Vue.js a háttérben egy virtuális DOM-ot használ. Amennyiben az applikációnk változói frissülnek, az igazi DOM is módosulni fog, ez egy sokkal gyorsabb performanszt, teljesítményt nyújt, mint például a jQuery.Ez a könyvtár egy MVC-hez hasonló architektúrát követ. Az adataink, a különféle metódusok, valamint az oldalunk HTML része könnyedén szétválasztható a segítségével, ebből adódóan egy jól áttekinthető, átlátható kódot tudhatunk a magunkénak. Az adatok „egy helyre gyűjtésében” lehet segítségünkre a Vuex, ami teljesen engedi különválasztani az applikációnktól azokat, ezáltal a különféle állapotokat meghatározó adatok egy helyre kerülnek. Á Vue.js komponensekből épül fel. Ennek kapcsán könnyedén hozhatunk létre „újrahasznosítható” elemeket az alkalmazásunkon belül: ha például egy képgalériát szeretnénk létrehozni, akkor az egyetlen képet megtestesítő elem lehet egy ilyen komponens. Vagy akár egy űrlap, melyet a weboldalunkon máshol is használni szeretnénk. Á Vue segítségével képesek lehetünk akár többoldalas appokat is építeni, ebből adódóan az úgynevezett routing is jól meg van benne oldva. Az Vue.js mellé rengeteg kiváló packaget, és third party plugint készítettek. Ezek közé tartozik például a már említett Vuex, vagy a Vuelidate, aminek a segítségével űrlap-érvényesítést valósíthatunk meg könnyedén. Kifejezetten szeretem, és egyik kedvenc packagem a Bootstrap Vue. A mindenki által ismert, és szeretett Bootstrapet veszi alapul, azzal a különbséggel, hogy tökéletesen illeszkedik a Vue.js-hez.

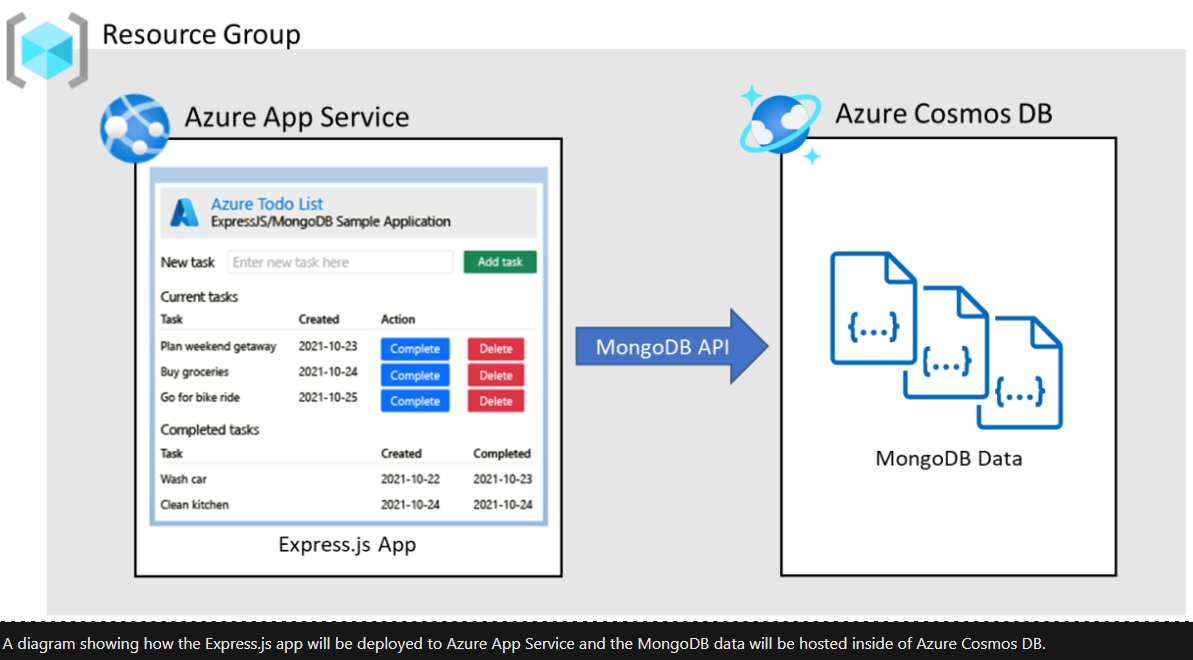
A képen szöveg látható

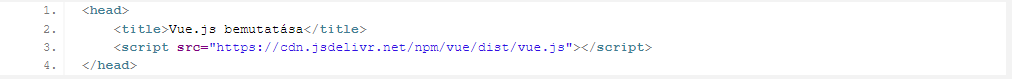
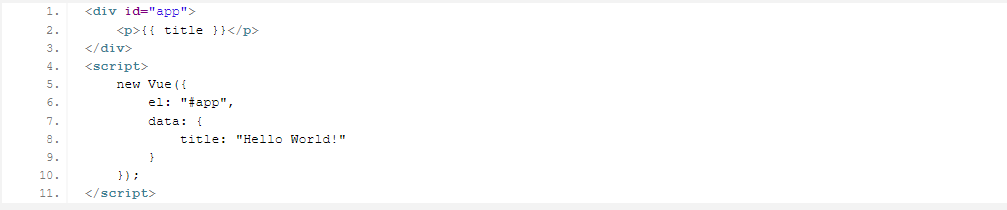
Automatikusan generált leírás

NODE JS: Egy MongoDB-alapú adatbázis rendszer legfelső rétege az ún. klaszter, ebbe szervezzük a szervereket. Mi klaszterekkel ebben a tárgyban nem foglalkozunk, azok a skálázás eszközei. A második szint a szerver szintje (a mongod processz), ami alatt az adatbázis foglal helyet. Egy szerver/klaszter több adatbázist tárolhat. Az adatbázisok pedig gyűjteményekből (collection) épülnek fel. A dokumentum a MongoDB alap tárolási egysége. Egy dokumentum egy JSON (jellegű) fájl, tehát kulcs-érték párokat tartalmaz. Maga a MongoDB BSON-ként, bináris reprezentációként tárolja mindezt. Kulcsoknak többnyire bármilyen szabad szöveget választhatunk, de a neveknek egyedinek kell lenniük és nem kezdődhetnek a $ karakterrel. A nevek case sensitive-ek. Az érték lehet szöveg, szám, dátum, bináris, beágyazott elem, null, vagy akár a fenti példában a groups kulcsnál láthatóan tömb is - relációs adatbázisban ezt így nem lehet reprezentálni. Az objektum orientált világban egy dokumentum felel meg egy objektumnak. Fontos megkötés, hogy a dokumentumok maximális mérete 16MB lehet, és ez nem konfigurálható érték.



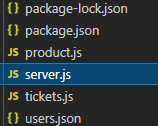
## 3.1 Fejlesztői környezet

A képen egy mintaalkalmazást fog üzembe helyezni Express.jsMongoDB-adatbázis használatával az Azure-ban. A Express.js alkalmazás Azure App Service lesz üzemeltetve, amely támogatja Node.js alkalmazások üzemeltetését Linux (Node 12, 14 és 16) és Windows (12- és 14-os verzió) kiszolgálói környezetekben. A MongoDB-adatbázis az Azure Cosmos DB-ben lesz üzemeltetve, amely egy 100%-ban MongoDB-kompatibilis API-t kínáló natív felhőbeli adatbázis.  Annak érdekében, hogy a Vue.js működése kicsit érthetőbb legyen, érdemes lehet néhány alapvető példát áttekinteni.A Vue.js installálásaA Vue.js-t nagyon egyszerűen installálhatjuk CDN segítségével. Abban az esetben, ha compilert szeretnénk használni, az NPM-hez kell majd nyúlnunk, viszont ha néhány egyszerű példát szeretnénk ismertetni, bőven elég lesz a CDN is. Adjuk az oldalunk tetejére az alábbi kódrészletet a headingen belül.

. A Vue komponensünket a new Vue példánnyal hozhatjuk létre. Itt az „el” rész lesz mindig a kijelölő (lényegében CSS selector), a komponensünk ezen belül fog „létezni”. A data objektumban kulcs-érték párokat megadva különféle értékeket hozhatunk létre, melyekre könnyedén hivatkozhatunk az applikációnkon belül a {{}} interpoláció szintaxis segítségével. 

## 3.2. A node js felépítése

A mongodb json fajlokból áll, ezek azok a fájlok amik szükségesek a weboldal futtatásáhóz.



Elindítás elött egy üres mappát hozunk létre egy nekünk tetsző helyen, ezt a mappát megnyitjuk a visual studio code segítségével, ez után telepítenünk kell az alap induló fájlokat, ami elöszőr az „npm init” parancs segítségével jön létre, ez a parancs legenerál nekünk egy „package.json” fájlt benne pár alap programmal. (ezen a képen már több program is telepítve van).



npm init <initializer>új vagy meglévő npm csomag beállítására használható. initializerebben az esetben egy npm nevű csomag create-<initializer>, amelyet telepíteni fog npm-exec, majd végrehajtja a fő binjét – feltehetően létrehozva vagy frissítve package.jsonés futtatva bármilyen más inicializálással kapcsolatos műveletet.

Ha a csomag közzétételét tervezi, a package.json fájl legfontosabb elemei a név és a verzió mezői, mivel ezekre szükség lesz *.*A név és a verzió együtt egy azonosítót alkot, amelyről feltételezzük, hogy teljesen egyedi. A csomag változásainak együtt kell járniuk a verzióval. Ha nem tervezi a csomag közzétételét, a név és a verzió mezők megadása nem kötelező.

A név az, amit a dolgodnak hívnak.

Néhány szabály:

* A név legfeljebb 214 karakter lehet. Ez magában foglalja a hatályos csomagok hatókörét.
* A hatályos csomagok neve ponttal vagy aláhúzással kezdődhet.Ez hatótávolság nélkül nem megengedett.
* Az új csomagok nevében nem lehetnek nagybetűk.
* A név végül egy URL része, egy argumentum a parancssorban és egy mappanév. Ezért a név nem tartalmazhat olyan karaktert, amely nem biztonságos az URL-címen.

Néhány tipp:

* Ne használja ugyanazt a nevet, mint egy központi csomópont modul.
* Ne tegyen „js” vagy „node” szót a névbe. Feltételezzük, hogy ez js, mivel egy package.json fájlt ír, és az "motorok" mezőben megadhatja a motort. (Lásd lentebb.)
* A név valószínűleg argumentumként kerül átadásra a request()-hez, tehát valami rövidnek, de ésszerűen leírónak kell lennie.
* Érdemes ellenőrizni az npm registry-t, hogy van-e már valami ezen a néven, mielőtt túlságosan kötődne hozzá.

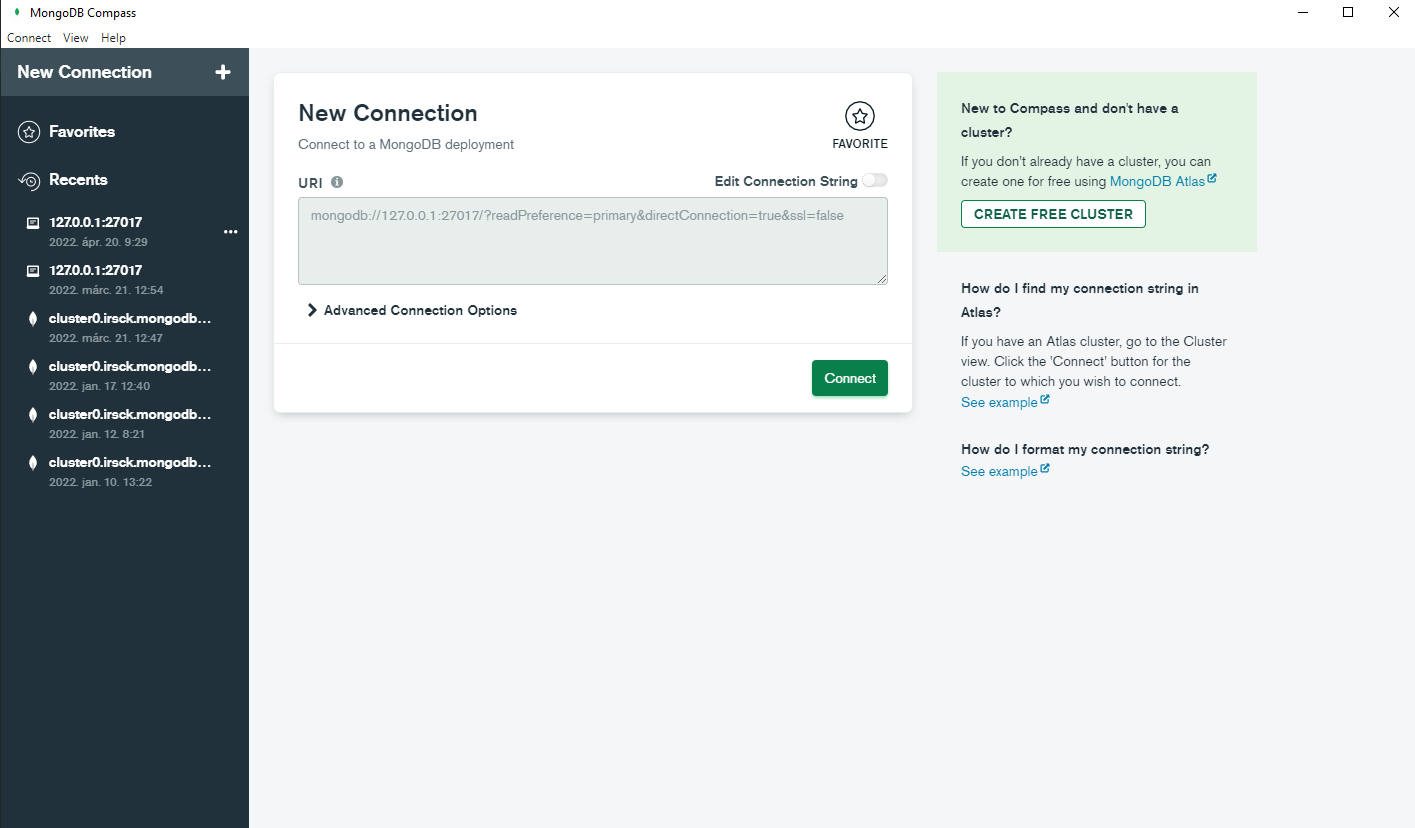
Ezen kívül mi módosítottuk a start scriptet is, hogy a program könnyebben indulhasson az „npm start” paranccsal.

"scripts": {

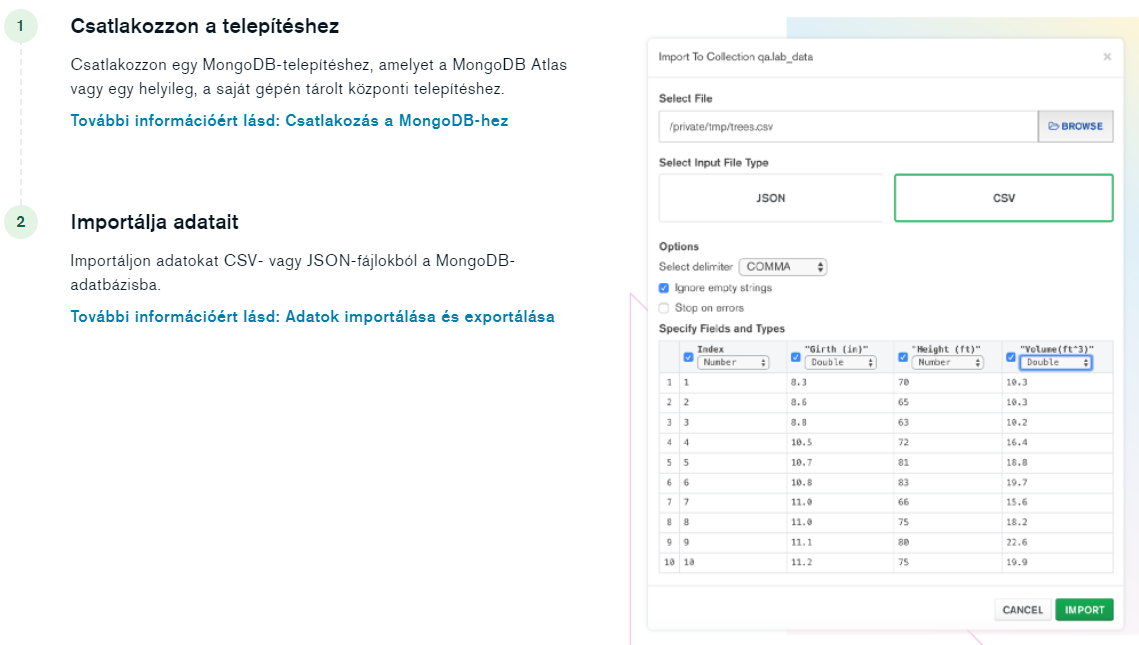
    "start": "nodemon ./bin/www"

  },

A mi programunk még kettő fontosabb programmal működik ami a mongodb és a mongoose. A mongodb egy adatbázis amit a compass segítségével érhet el a felhasználó.



Itt a felhasználó a mongodb által létrehozott ip-n keresztül tud felcsatlakozni a saját adatbázis szerverére, ahol collectionokat (táblákat) tud létre hozni, amiben az adatokat tudja letárolni, és ezt vissza kérni. Létrehozhat adatbázisokat és gyűjteményeket a MongoDB-ben a MongoDB Shell, mongosh használatával. Ezt külön eszközként töltheti le és telepítheti. Alternatív megoldásként elérheti a MongoDB asztali alkalmazásával, a MongoDB Compass-szal. Ez egy sokoldalú grafikus felhasználói felület, amely lehetővé teszi az adatbázisok és a hozzájuk tartozó gyűjtemények megtekintését. Opcionális grafikus felületet is kínál a MongoDB szolgáltatásainak használatához. Tehát ez egy jó lehetőség, ha nem szeretne szöveges parancsokat begépelni adatbázisok és gyűjtemények létrehozásához.





A másik program a mongoose, amit a „ npm i –save –mongoose” paranccsal kell telepíteni. A MongoDB szerver (általában) egy TCP foglalatot hallgat, és a Node.js folyamat TCP kapcsolaton keresztül csatlakozhat hozzá. De a TCP tetején a MongoDB saját protokollal is rendelkezik annak megértéséhez, hogy az ügyfél (a Node.js folyamatunk) pontosan mit akar az adatbázisnak. Ehhez a kommunikációhoz ahelyett, hogy megtanulnánk azokat az üzeneteket, amelyeket a TCP rétegre kell küldenünk, elvonatkoztatunk egy "driver" szoftver segítségével, amelyet ebben az esetben MongoDB drivernek hívunk. A MongoDB illesztőprogram itt érhető el npm csomagként. Ne feledje, hogy a MongoDB illesztőprogram felelős az alacsony szintű kommunikációs kérelem / válaszok összekapcsolásáért és kivonásáért - de ez csak fejlesztőként vezet el. Mivel a MongoDB egy sémátlan adatbázis, így sokkal több energiát ad, mint amennyire kezdőként szüksége van. A nagyobb teljesítmény nagyobb felületet jelent a dolgok rosszra fordításához. Csökkentenie kell a hibáinak és csavarjainak felületét, amelyeket a kódjában megtehet. Kell még valami. Találkozz Mongoose-val. A Mongoose egy absztrakció a natív MongoDB illesztőprogram felett (a fent említett npm csomag).

## 3.2. A vue js felépítése

Első sorban telepíteni kell a Visual Studio Codeban a Vue JS-t, és a vetur-t

A képen szöveg látható

Automatikusan generált leírás

Ezt követően egy parancssort kell índitanunk, ahol a következők szerint kell haladni:

**vue create <projekt neve>**

> **Manually select features**

? Please pick a preset: Manually select features

? Check the features needed for your project:

 (\*) Choose Vue version

 (\*) Babel

 ( ) TypeScript

 ( ) Progressive Web App (PWA) Support

**>(\*) Router**

 ( ) Vuex

 ( ) CSS Pre-processors

 (\*) Linter / Formatter

 ( ) Unit Testing

 ( ) E2E Testing

**> 3.x**

 Use history mode for router? (Requires proper server setup for index fallback in production) (**Y**/n)

**> ESLint with error prevention only**

**>(\*) Lint on save**

**> In dedicated config files**

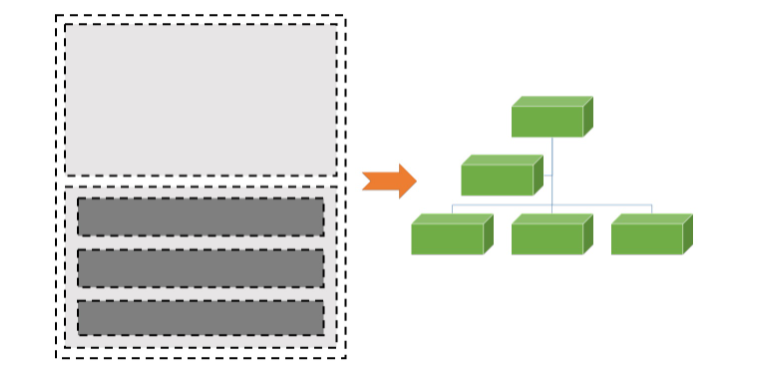
? Save this as a preset for future projects? (y/**N**)

Ez után a következő alap porgram telepítödik fel, amit visual studio code-al kell megnyitni

A képen szöveg látható

Automatikusan generált leírás

Itt több dolgot is láthatunk, és létre hozghatunk az első a components. A komponens a Vue.js keretrendszer egyik kulcsfontosságú koncepciója ami megadja azt a lehetőséget, hogy a kódunkat egyrészt funkcionalitás szerint csoportosíthatjuk, másrészt újrafelhasználhatjuk (rengeteg sornyi kódot megspórolva) Nem szükséges magunknak újraírni teljes komponenseket ha találunk az interneten való böngészés során funkcionalitásban hasonló komponenseket, elegendő azokat módosítani. Úgy képzelhetjül el őket, mint a LEGO kockák: egymással összeilleszthetjük őket és bármit építhetünk belőle. Az alkalmazásban egy-egy nézet eltérő helyet foglal el a képernyőn. Amikor egy alkalmazás sok, de kis helyet elfoglaló nézetet tartalmaz, szokás egymásba ágyazni őket. A Vue.js keretrendszerben egy úgy valósítható meg, hogy a komponenseket egyszerűen egymásban meghivatkozzuk. Kétféle megközelítés ismert: az összes komponens deklarálása a main.js fájlban, továbbá minden egyes komponens implementálása külön fájlokként. Mind a két megközelítést tárgyalni fogjuk, egyenlőre fókuszáljunk az elsőre. Tegyük fel, hogy egy-egy funkcionalitást egy-egy komponens segítségével szeretnénk megvalósítani. A funkciók eléréséhez az egyes komponenst regisztrálni és végrehajtani kell a main.js fájlban. A komponenseket regisztrálhatjuk globálisan és lokálisan. Ebben az esetben tegyük globálisan elérhetővé a komponenseket a Vue.component(nev,beallitasok) metódus meghívásával. Globálisan regisztrált komponenst bármely másik komponensben meghívható, ugyanazon Vue példányán belül. Az első paraméter a komponens neve, amilyen név szerint szeretnénk regisztrálni. A második paraméter egy objektumot vár, amiben a komponenst definiáljuk. Minden egyes komponens tartalmaz egy <template></template> szekciót, amiben a nézetünk felépítését definiálhatjuk. Ebben az esetben használjuk a sztring literál deklarálási formát. A <template></template> egészen pontosan egyetlen gyökér elemet tartalmazhat. Ha több HTML elemből építettük fel a nézetet, akkor szúrjuk be őket egy <div></div>. A nézeten belül a következőképpen hívhatunk meg egy tetszőleges komponenst: <komponens\_neve ></komponens\_neve >, ahol komponens\_neve a komponens regisztrált neve. A Vue példányunkhoz hasonlóan itt is értelmezve van a data tulajdonság. Azonban a data itt egy függvény, ami egy objektummal tér vissza. Ha a data tulajdonság egy objektumot tartalmazna, akkor minden egyes komponens ugyanazon az adatokon fog osztozni, tehát az újrafelhasználhatóság koncepcióját dobhatnánk a kukába, nem használhatjuk sablonként. Ehelyett visszatérünk egy objektummal a data függvény segítségével, így minden adat egyedi lesz, ahányszor hívjuk meg a kérdéses komponenst. Definíció szerint azt a komponenst, amibe ágyazunk egy másikat szülő komponensnek nevezzük, amit beleágyaztunk pedig gyerek komponensnek hívjuk.



A Vue Router használata: A modern egyoldalas webalkalmazások (SPA) egyik leghatékonyabb funkciója az útválasztás. A modern egyoldalas alkalmazások, például a Vue-alkalmazások oldalról oldalra válthatnak a kliens oldalon (a szerver kérése nélkül). A Vue Router a Vue alkalmazások oldalnavigációjának hivatalos könyvtára. A Vue Router használata egyszerű, mégis nagy teljesítményű. Ebben a cikkben a Vue Router 4-et (a Vue 3-mal együtt használva) elmélyülünk. Mindent áttekintünk, amit tudnia kell a Vue Router kényelmes használatához.

* A Vue Router alapjai
* Dinamikus útválasztás
* A router paramétereinek átadása

**A Vue Router alapjai:** Hozzunk létre egy új projektet, és vágjunk bele a kódolásba. Létrehozunk egy új projektet a vue-cli segítségével. Ha nincs telepítve, telepítheti az „npm install -g @vue/cli” parancs futtatásával.

A képen szöveg látható

Automatikusan generált leírás

**Dinamikus útválasztás:** Tegyük fel, hogy az alkalmazásunkban ez az elérési út /posts. Amikor a felhasználó ezekre a cikkekre kattint, a felhasználó az adott cikk oldalára kerül. Minden cikknél más lesz az elérési út (/posts/article-1, /posts/article-2 stb.). Hogyan érjük el ezt? Minden bejegyzésünkhöz hozzáadhatunk egy új útvonalat, de ezután minden új bejegyzés megjelenésekor módosítanunk kell a kódunkat. Ezért dinamikus útvonalat fogunk használni. Itt jönnek jól a router paraméterei. Alkalmazásunkban megadhatunk egy általános útvonalat, mint például a /posts, majd opcionális útvonalparamétereket is megadhatunk ugyanahhoz az útvonalhoz, mint ez a /posts/:id. Az: id bármilyen dinamikus karakterláncot felvesz paraméterként. Ez azt jelenti, hogy mostantól olyan elérési utakra tudunk irányítani, mint a /posts/article-1, /posts/article-2 stb., és az alkalmazásban lévő Vue útválasztó tudni fogja, hogy melyik útvonalra próbálunk menni.

A képen szöveg látható

Automatikusan generált leírás

**A router paramétereinek átadása**: Amikor hozzáadjuk a Vue routert a Vue alkalmazásunkhoz, hozzáférést kapunk egy speciális $route objektumhoz az alkalmazásunk globális hatókörén belül. Ez az objektum tartalmazza az alkalmazásunk összes útválasztó információját. A $route objektumot a this. $route egyszerű meghívásával érhetjük el. A paraméterek információi ezen a $route objektumon belül is elérhetők. A Profile komponensen belüli params objektumot a $route objektumon keresztül érhetjük el. Vessen egy pillantást az alábbi kódrészletre.

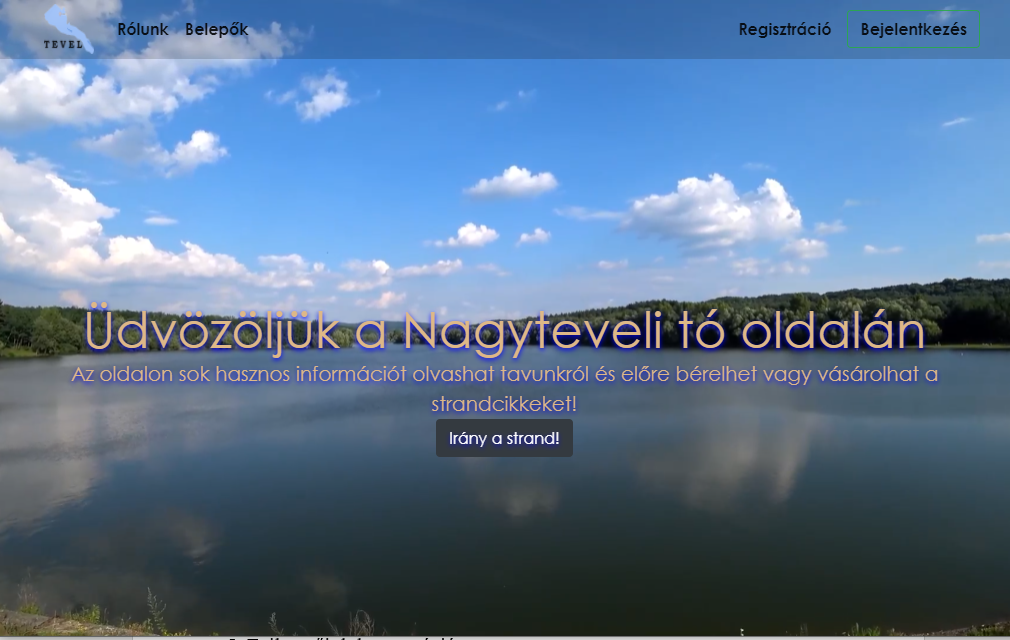
A képen szöveg látható

Automatikusan generált leírás

# 4. Felhasználói dokumentáció

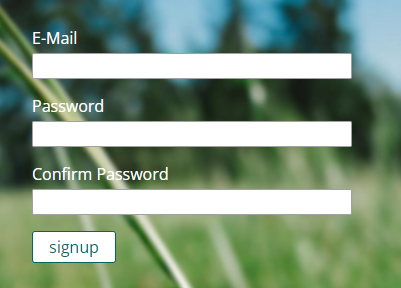
## 4.1. Telepítés

A fájl letöltése után ki kell csomagolni, és visual studio codeal megnyitni. A programot elindítás előtt telepíteni kell a terminálon belül az ,,npm i” parancsala többi már telepítve van a fájlokban, ez azért nincs, hogy a fájl kisebb méretű legyen, és könnyebben lehessen másolni. Ha mindkettő terminálban elvégeztük a node modul telepítését el kell indítani a servert. A backendnél az „npm start” parancsal indul el a server, a frontendél az „npm run serve” parancsal, ezután megkell nyitni a böngészőt és a LOCALHOST:8080 porton meg is találjuk a szervert.



## 4.2 A weboldal fő jellemzői

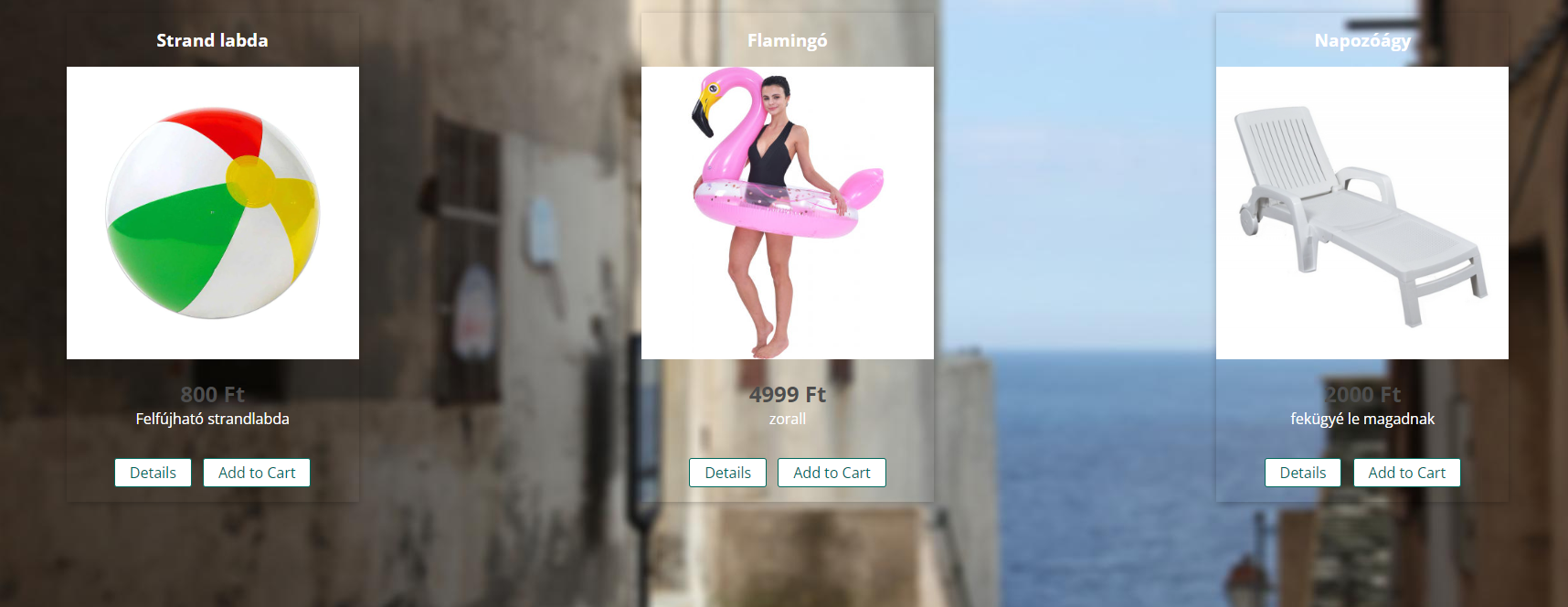
A weboldalunk egy saját egyedi logóval rendelkezik, amit mi készítettünk, az oldalon Nagytavelről olvashat a felhasználó pár információt, ezt követően a „regisztráció” gombra kattintva megjelenik a regisztrációs oldal ahol a felhasználónak bekell regisztrálni, hogy terméké, és bérleteink között tudjon vásárolni.



Ez követően be kell jelentkezni a felhasználónak a megadott adatokkal.



Igy már a bejelentkezett felhasználónak lehetősége van az általunk kinált termékek, és bérletek között vásárolni.



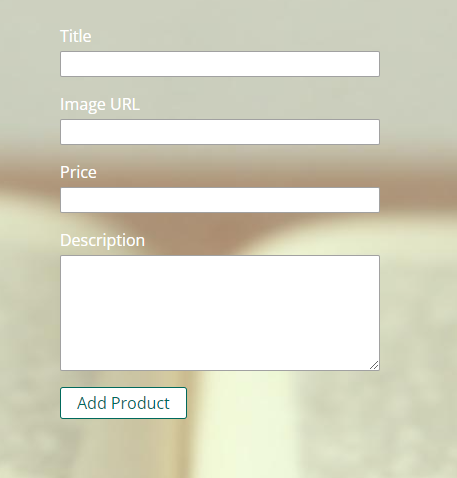
Ezen felül az oldalon van admin jogú felhasználó is, ezen jogú felhasználók egyedi engedélyel bírnak, míg az alap felhsználók csak az alap füleket látják:

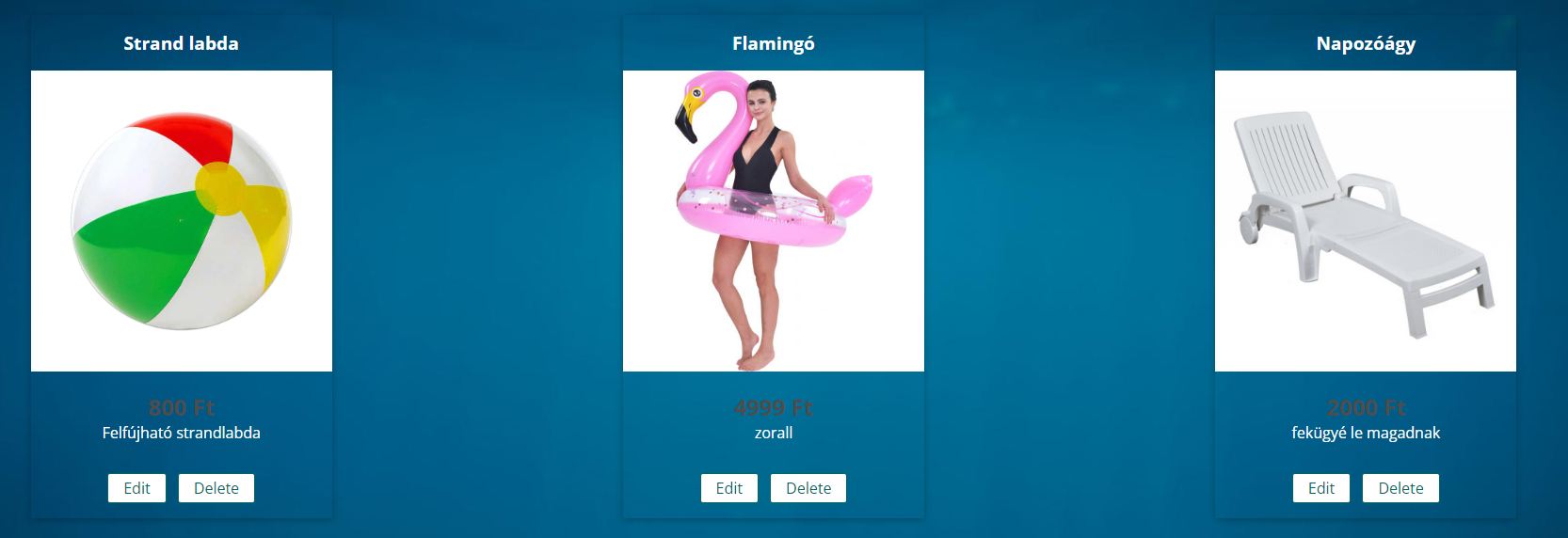


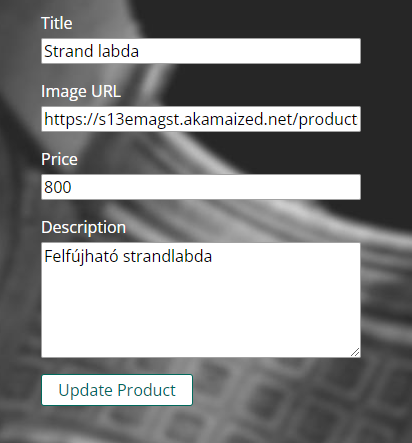
addig az admin jogú felhasználó az admin felülelet is:



Itt kettő extra funkcióra van lehetősége az admin-nak az egyik az „ Add Product, ahol új terméket tud hozzá adni a felhasználó, ahol megkell adni az általa létrehozott termék nevét, a termék egy képét, az árát és egy rövid leírást.



Az admin felhasználó másik lehetősége a már megadott termékek modosítása, itt ugyan úgy csak már a kész termék nevét, képét, árát és a leírásán tud változtatni, és lehetősége van arra, hogy a már kész terméket ki törölje a weboldalról (a mongodb collectionból). 



# 5. Fejlesztői dokumentáció

A fejlesztés során több dolgot is szerettünk volna, ezek küzöl az egyik egy qr kód olvasó lett volna, ami a felhasználóknak segít akar a bejelentkezésben akar a termékek elérésében, de sajnos ezt nem sikerült megvalósítani. A webshopnuk fő célja az volt hogy külső adatbázisból jelenjenek meg a termékeink, amit bármikor könnyen lehet modosítani, és ez sikerült is. a mongodb compass segítségével. A termékek külön kollekciókban vannak eltárolva ez segít az adatok könnyü modosításában, hozzádásában esetleg törlésében, valamint az admin jogú és sima felhasznáók adati kollekciókban van tárolva.

## 5.1. Első lépések

A weboldal megtervezése és alap kinézete volt az első dolgunk, ez csak egy kezdetleges elinduló vázlat volt, megterveztük a view-t ami a weboldal kinézetéért volt felelős, ezt követően routes mappat hoztunk létre, ahol linkelnki kellett az adatokat, ahhoz hogy képesek legyünk össze kapcsolni a weboldalt, ezt követően jöttek a controller mappak, hogy müködhessenek a routerek. Itt exportalni kellett a kérésket, hogy vissza adja a weblap az adatokat. Majd elkeszítettünk egy auth pontosabban bejelentkezés, és regisztráció lehetőséget, ahol a weblapunkra tudnak, regisztrálni a felhasználók, hiszen igy lesz lehetőségük, arra hogy a térmék és bérletek közül vásárolni tudjanak. Majd létre hozuk az admin jogú felhasználót, akinek lehetősége van az adott termékek modosítására, hozzáadására és akar törlésére is.

## 5.2. Views

A képen szöveg látható

Automatikusan generált leírásA views mappában kellett a legtöbb fájlt létre hoznunk, hiszen ez az, ami visszaadja a weboldalunk kinézetét. A shop volt az első mappa, amit létre hoztunk itt több ejs fájlt hoztunk létre, az ejs ebben a szegmensben az Embedded Javascript-et fogjuk érteni, közismertebb nevén EJS-t. Javascript kód HTML kódba való beágyazására szolgál. Bevezető definíciónk részeként említettük, hogy az EJS-t elsősorban a node.js-ben lévő Expresszel használják. Sablonmotorként működik, amely alapvetően segít a JavaScript-kód megjelenítésében az ügyféloldalon. Értsük meg a Javascript kód HTML-be ágyazásának fogalmát egy példa segítségével. Ez a kód még nem fog működni, mert a HTML kódba be van ágyazva a Javascript. Ebben a konkrét példában meg kell mondanunk a HTML-nek, hogy a WebDev egy JavaScript kód, és úgy kell kezelni. Ha nem mondjuk ki kifejezetten, hogy ez a változó a Javascript része, a kimenet sima HTML lesz. Itt jön a képbe az EJS. Egy tipikus EJS kód így néz ki: <%=%> Szóval ez egy kisebb, mint jel, amelyet a százalékjel követ, van egy hely, ahová a JavaScript kód beírható, a Javascript kód után ismét egy százalékjel ezt követi a nagyobb, mint jel. Majd miután létre hoztuk az ejs fájlokat incdule-ni kellett a products-t, hogy a termékeink meg jelenhessenek.

A képen szöveg látható

Automatikusan generált leírás

## 5.3. Routes

A views mappa után a routes következett, itt útvonalakat kellett megadnunk, hogy a webshopunk visszaadhassa az adatokat. Valamint a különböző függőségeket is létre kellett hozni, valamint be exportálni, ez azért kellett, hogy get kéréseket tudjunk a későbbiekben, majd létrehozni. Itt egyesével megkellett írni az összes get kérést, hogy a weboldalon megtudjuk őket, majd nyitni, hiszen ezek irányítják át a másik oldalra a felhasználót. A kód beállítja és inicializálja az Express szervert, és két útvonalat határoz meg: a kezdőlapot a „/” -nél és egy naplóoldalt a „/log” -nál. Ezután felpörgethetjük a szerverünket a node server.js futtatásával. Ha ezután a localhost:3000-re lépünk, akkor a kezdőlapra kell jutnunk (ahol jelenleg csak a „Home Page” jelenik meg). Ha a localhost:3000/log oldalra lépünk, akkor a „Log Page” -ot kell látnunk.

A képen szöveg látható

Automatikusan generált leírás

## A képen szöveg látható Automatikusan generált leírás5.4. controllerek és a modellek

Létre hoztuk a controllers mappát, és ezzel együtt a models mappát is, hiszen először megkellett írnunk a models mappán belül az új schema-t, ami a termékek nevéhez árához, leírásához és képéhez kellettek, itt külön megkellett adni az adatok típusát, és azt hogy ezek kötelező mezők. Ezt külön megkellett csinálnunk a felhasználók (user) megrendelések (orders)és a jegyek (ticket) között A képen szöveg látható

Automatikusan generált leírás

Amint megírtuk a modelseket jöttek a controllerek itt post kéréskét kellett írnunk. A számítástechnikában a POST a World Wide Web által használt HTTP által támogatott kérési módszer. Tervezés szerint a POST kérési metódus azt kéri, hogy egy webszerver fogadja el a kérésüzenet törzsébe foglalt adatokat, valószínűleg azok tárolására.[1] Gyakran használják fájl feltöltésekor vagy kitöltött webes űrlap beküldésekor. Ezzel szemben a HTTP GET kérési módszer információkat kér le a szerverről. A GET-kérés részeként bizonyos adatok átadhatók az URL lekérdezési karakterláncán belül, megadva (például) keresési kifejezéseket, dátumtartományokat vagy a lekérdezést meghatározó egyéb információkat POST kérés részeként tetszőleges mennyiségű adat küldhető a szervernek a kérés üzenet szövegében. A POST-kérés fejlécmezője általában az üzenettörzs internetes médiatípusát jelzi.

## 5.5. Auth (login, sign up)

Miután a weblapunkon már megjelentek a termékeink az adatbázisból, és minden más fül is működött, auth azaz bejelentkezés illetve regisztrációt készítettünk az oldalunkhoz. Itt minden már megírt mappánkhoz új js-ejs fájlokat kellett létre hoznunk, de a legfontosabb a controllers mappa volt, hiszen itt kellett megírnunk a bejelentkezéshez szükséges parancsokat és funkciókat.

* bcrypt
* nodemailer
* sendgridTransport

Ezeket a dependencies-t használtok az email küldéshöz, illetve a regisztrációhoz

A képen szöveg látható

Automatikusan generált leírás

Mint ahogy az ábrán is látható ismét a post kéréskét kellett alkalmaznunk, ezen kívül szükségünk volt a csurf dependencies -ra is hiszen ennek segítségével tudjuk a felhasználók jelszavát titkosítani. A **Bcrypt** egy népszerű és megbízható módszer a jelszavak kivonására és kivonására. Megtanulta, hogyan kell használni a bcrypt NodeJS könyvtárát a jelszó tárolására és kivonatolására, mielőtt azt adatbázisban tárolná. Azt is megtanulta, hogyan lehet a bcrypt összehasonlítás funkciót használni a jelszó és a hitelesítéshez szükséges hash összehasonlítására. A **Nodemailer** egy csomópont. js modul, amellyel könnyedén küldhet e-maileket a szerveréről. Függetlenül attól, hogy kommunikálni szeretne a felhasználókkal, vagy csak értesíteni szeretné magát, ha valami elromlott, az egyik lehetőség a levélben.

## 5.6. Admin

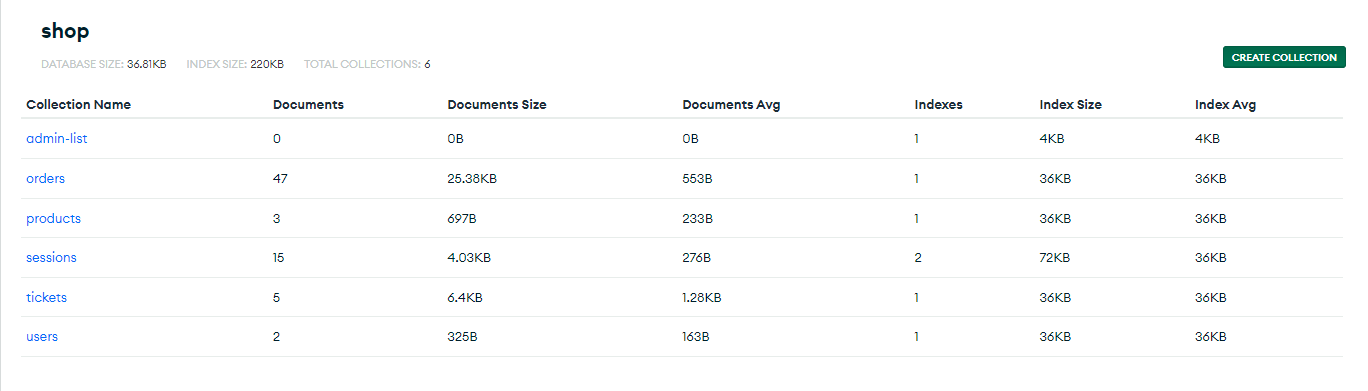
Miután a felhasználók már betudnak jelentkezni, és vásárolni is tudnak az oldalon úgy döntöttünk létre hozunk egy admin felületet is, amit kizárólag olyan felhasználó érhet el, akinek erre külön engedélyt adunk. Itt az admin-nak lehetőséget adtunk termékeink módosítására, hozzáadására és akar eltávolítására is, fontos, hogy ha egy terméket töröl az admin akkor az a termék az adatbázisból is törlődni fog, ezzel csökkentve az adatbázis méretét. Itt megterveztük a views mappában az admin oldal kinézetét majd controllers-nél megírtuk a postEditProduct, postAddProduct és a postDeleteProduct, illetve get függvényeket. Ahhoz, hogy csak az admin felhasználók láthassak csak ezeket az oldalakat el kellett rejtenünk, a sima felhasználó elől ezt a weblapon, ezt az authentication dependencies segítségével oldottuk meg. Az alapfelhasználókat is hitelesíteni kell, mielőtt hozzáférést adnánk nekik a felhasználói útvonalhoz. Létre hozunk egy másik köztes szoftvert az auth.js fájlban, amely hitelesíti az alapvető felhasználókat: az exportálást. userAuth = (req, res, next) => {const token = req. sütiket. jwt if (token) {jwt

. A képen szöveg látható

Automatikusan generált leírás

## 5.7. Adatbázis

Az adatbázisunk mongodb-t használ, ez egy NoSQL. A NoSQL adatbázis-technológia JSON-dokumentumokban tárolja az információkat a relációs adatbázisok által használt oszlopok és sorok helyett. Az egyértelműség kedvéért a NoSQL azt jelenti, hogy „nem csak SQL”, nem pedig „nincs SQL”.



Ezenkívül a csurf dependes segítségével, a felhasználók jelszavát titkosítani is tudtuk.

A képen szöveg látható

Automatikusan generált leírás

A mongodb a többi adatbázistól eltérőbben, nem táblázatokból áll, hanem collectionokból, de a funkciója ugyan az.

## 5.8. Munkamegosztás

Próbáltuk körülbelül egyenlő féleképpen elosztani a feladatokat. Mivel mind hárman Pápáról és távolabb járunk iskolába, így a mindennapos vonatozás közben rengeteg lehetőségünk volt a feladat megbeszélésében, átbeszélésében. Ezenkívül discordon beszélgettünk még. Pár idő pont a beszélgetésekről:

A képen szöveg látható

Automatikusan generált leírás

A képen térkép látható

Automatikusan generált leírásTakács Adrián volt a Frontend felelős (de ő is segített a design és a backend megírásánál), ő írta a VUE-t, ezen kívül még tervezett, egy egyedi logót is az oldalhoz. Összekötötte a Backendet és a Frontendet, qr kód generálást és kiküldése készített.

A képen szöveg látható

Automatikusan generált leírás

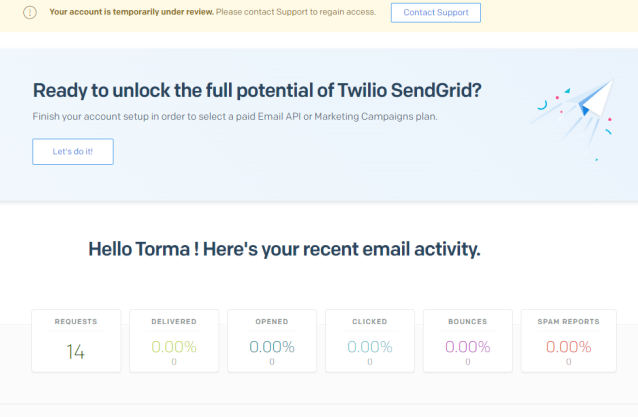
Torma Erik készítette az adatbázist, amit sikeresen össze is kapcsolt a backenddel. A képen szöveg látható

Automatikusan generált leírás

Ezen kívül megírta az email küldés funkciót, amivel nagyon sok nehézség volt, mivel a sendgrid oldalával probléma adódott.

. A képen szöveg látható

Automatikusan generált leírás



A képen szöveg látható

Automatikusan generált leírásPuskás Máté készítette a fő designe-t, valamint a backendhez az útvonalakat készítette el, megoldotta a csurf token problémát, tesztelte a programot, és megírta a dokumentáció nagy részét.

A képen szöveg látható

Automatikusan generált leírásA képen szöveg látható

Automatikusan generált leírás

A backenden belül: Takács Adrián felvitte a termékeket az adatbázisba. Megcsinálta a termékek módosítását, és törlését, hogy ez a weboldalon keresztül működhessen. Torma Erik elkészítette a models mappát, ahol schémákat hozott létra az adatokhoz. Tökéletesítette az email küldés funkciót. Valamint elkezdte az authentication-t. Puskás Máté megírta authentication-t, amivel lehetőség van a regisztrációra, és a bejelentkezésre, ezen kívül megcsinálta az admin felület, ahol titkosította az admin fület, az átlag felhasználótól, valamint megírta a get kéréseket az oldalhoz.

# 6. Tesztelés

A weboldalunk teszteléséhez a Thunder Client-t használtuk, A Thunder Client a könnyű Rest API-kliens-bővítmény, amely lehetővé teszi a fejlesztők számára a HTTP-kérés tesztelését és a válaszüzenet megtekintését közvetlenül a VSCode tiszta és egyszerű grafikus felületén. Localhost:3000 Localhost:3000/users, Localhost:3000/products, Localhost:3000/tickets   
Képek a sikeres tesztelésről:

A képen szöveg látható

Automatikusan generált leírásA képen szöveg, monitor, képernyőkép, képernyő látható

Automatikusan generált leírás

A képen szöveg látható

Automatikusan generált leírás

# 7. Összefoglalás

Szakdolgozatunk célja egy online elérhető weboldal, ahol regisztráció után a felhasználóknak lehetőséget adunk, arra, hogy a sok különböző termékeink közül szabadon válogasson, és akár jegyet, bérletet foglaljon magának. Úgy gondoljuk, hogy végeredményül egy használható weboldalt/webshopot hoztunk létre. Ennek a megvalósításához Javascriptet használtunk. A kivitelezés közben adódtak komolyabb hibák, amiket meg kellet oldanunk. Jelen állás szerint a széles körűen elvégzett tesztelések eredménye ként azt tapasztaltuk, hogy remekül működött a weboldal. A következő pontban azokat a lehetőségeket szeretnénk összegezni, amelyek egy esetleges későbbi weboldal során kelhetnek. Ezek közül vannak, amelyek bár belekerülhettek volna a kész programba, valamilyen okból kifolyólag nem kerültek megvalósításra, és vannak olyan pontok is, amelyek csak a program befejeztével merültek fel, mint új lehetséges fejlődési irányvonalként.

## 7.1. jövőbeli tervek

A legnagyobb lehetősége az online fizetés tényleges lehetősége, ezzel képesek lennénk egy tényleg eladható webshopot létrehozni. Ehhez a megvalósításhoz több banki engedély is kellene, valamint egy komplett online fizetési opció. Ezenkívül több mellék funkcióval kibővítés ilyen például egy gps location, hiba üzenet helytelen email beírásánál, valamint az általunk készített weboldal tényleges eladása a falunak.

# 8. Felhasznált források

* <https://www.w3schools.com>
* <https://vuejs.org>
* <https://nodejs.org/en/>
* <https://www.mongodb.com>