



győri szakképzési centrum

Jedlik Ányos
Gépipari és Informatikai
Technikum és Kollégium



9021 Győr, Szent István út 7.

+36 (96) 529-480

+36 (96) 529-448

OM: 203037/003

jedlik@jedlik.eu

www.jedlik.eu

Záródolgozat feladatkiírás

Tanuló(k) neve¹: Kiss Ádám, Hóbor Martin, Filip Dávid
Képzés: nappali / felnőttoktatás - esti munkarend
Szak: 5 0613 12 03 Szoftverfejlesztő és tesztelő technikus

A záródolgozat címe:

„Balatoni borozók” webshop

Konzulens: Horváth Norbert

Beadási határidő: 2022. 04. 29.

Győr, 2022. 04. 28

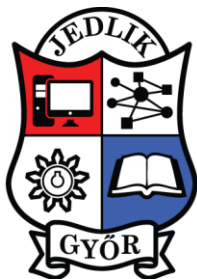
Módos Gábor
igazgató

¹ Szakmajegyzékes záródolgozat esetében több szerzője is lehet a dokumentumnak, OKJ-s záródolgozatnál egyetlen személy ad le záródolgozatot.



győri szakképzési centrum

Jedlik Ányos
Gépipari és Informatikai
Technikum és Kollégium



9021 Győr, Szent István út 7.

+36 (96) 529-480

+36 (96) 529-448

OM: 203037/003

jedlik@jedlik.eu

www.jedlik.eu

Konzultációs lap²

	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 28.

tanuló aláírása

tanuló aláírása

tanuló aláírása

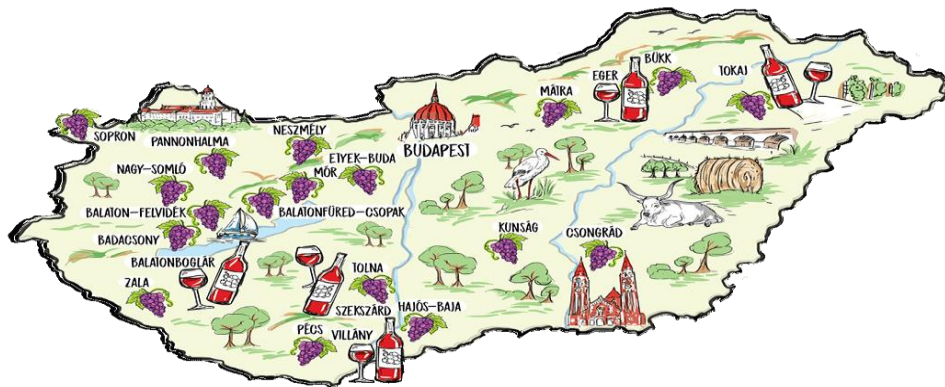
² Szakmajegyzékes, csoportos konzultációs lap

Tartalomjegyzék

1.	Bevezetés	4
2.	Specifikáció.....	4
2.1.	Szoftver alapvető jellemzői.....	5
2.2.	Szoftver funkciói.....	5
3.	Alkalmazott technológiák	6
3.1.	Felépítése	7
3.2.	Weboldal felépítése	8
3.3.	Node JS	10
3.4.	Fejlesztői környezet	12
3.	Felhasználói dokumentáció.....	12
3.1.	Előkészítése.....	12
3.2.	Főbb funkciói	13
3.3.	Vásárlói funkciók.....	14
5.	Fejlesztői dokumentáció	17
5.1.	Kezdet	18
5.1.	Menük	21
5.2.	Belső funkciók felépítése.....	23
5.3.	GET, POST kérések.....	31
5.4.	Fejlesztői konferencia	32
5.5.	Tesztelés.....	32
6.	Összefoglalás	35
6.1.	Jövőbeli tervek	36
7.	Felhasznált irodalmak	37

1. Bevezetés

Modern napjainkban a magyarországi túrizmus érezhetően megnövekedett a Balaton és Magyarország más tájain is, de mi most azoknak a turistáknak segítünk akik igazából érdeklődnek a magyarországi borkultúra iránt amely nem csak a Balatoni tájat foglalja magában hanem egyesíti az alföld egyes régió ízeit. Magyarországon borászati oktatásáról az első írásos emlékeink az 1700-as évek végétől vannak, így például Tóth Prónay Prónay Pál nógrádi és honti táblabíró 1780-as vagy - a többek között a bécsi Teréziánumban főnemesi házak gyermekeinek birtokgazdálkodást oktató - Mittelpacher Lajos 1815-ös kiadású műve. Az oktatás és kutatás szoros kölcsönhatásban működött. 18.-19. század folyamán neves szakemberek segítették a szakoktatást, és hagyták ránk a tudásukat, mint például Wartha Vince, a Műegyetem neves professzora, Kosutány Tamás, a magyaróvári gazdasági akadémia kémia tanára, majd 1894-től az új Országos Kémiai Intézet igazgatója, aki a bor erjedése, a borélesztő, a borbetegségek kérdéseivel foglalkozott, Schams Ferenc szőlőfajtakutató, Entz Ferenc pomológus, az első nem magán szakmai oktatási intézmény, a Vincellér- és Kertészképző Gyakorlati Tanintézet megalapítója és Rudinai Molnár István, a kertészeti szakoktatás és termesztés korszerűsítője.



2. Specifikáció

A Szakdolgozatunk egy weboldalt ami a Node JS szoftverrendszer segítségével lett megvalósítva. A fejlesztés első lépéseként meg kellett határozunk, hogy mit tudjon a programom, miben végzem a fejlesztést. A program több részből is áll elsők egy igényes weboldal backend részének az elkészítése után a frontend rész segítségével az ember szépség ingerküszöbét meghaladó díszítés kialakítása.

2.1. Szoftver alapvető jellemzői

- Szoftver elnevezése, címe: „Balatoni borozók”
- Szoftver kategóriája: Weboldal, Mobil Alkalmazás
- Szoftver alapvető funkciója: A weboldal alap funkciója, hogy a felhasználó tájékozódni tudjon a Balatoni borozási szokásokról illetve a pincészetek meglétéről.
- Fejlesztői környezet: Visual Studio termékcsalád
- Programozási nyelvek: Java
- Platform: Böngésző illetve Android telefonok

2.2. Szoftver funkciói

Ez egy olyan weboldal amelyben lehet nézni a Balatoni körüli térképet ahol megtalálhatók a különböző pincészetek amelyekből a borokat meg lehet nézni milyenek elérhetők és igazából a megfelelő bor kiválasztása után lehet is rendelni a megadott borok közül, de ennek van olyan része is, hogy az admin aki kezeli a weboldalt olyan részből a termékek törlése, szerkesztése és újaknak a felvétele. Alapvető funkciók amik működnek a bejelentkezés esetleg elfelejtett jelszó esetén lehet megváltoztatni, regisztrációnál jelszó titkosítás a visszaélések elkerülése végett. Kötelező mezők bevonása, hogy az állítólagos Robotokat elkerüljük. Az alapvető adatbázis a MongoDB amelyet azért használtunk mert ez volt számunkra a legkönnyebb illetve a legszimpatikusabb és kezelhető. A rendelés menete: kiválasztja az adott felhasználó, hogy melyik terméket szeretné választani bedobja a kosárba ott tudja ellenőrizni, hogy valóban azt akarja-e megvenni ha igen akkor rákattint a megrendelésre és átdobja a rendelés földre amelyben leírja, hogy miket rendelt meg és milyen értékben. Ezt az admin tudja majd kezelni és befogadni és ellenőrizni, hogy minden adat rendben van. A többi dolog meg úgymond a „Developer” dolga, hogy az alapvető funkciók működjenek.

Funkció szerinti csoportosítás

- **Felhasználói szoftver:**
olyan programok, amelyek egyfajta felhasználói igényt elégítenek ki
 - Szövegszerkesztők ([Word](#), [Jegyzettömb](#))
 - Táblázatkezelők ([Excel](#))
 - Adatbáziskezelők ([Access](#))
 - Grafikai programok ([Paint](#), [Photoshop](#), [GIMP](#))
 - Prezentációs programok ([PowerPoint](#), [Prezi](#))
 - Böngészők ([Explorer](#), [Firefox](#), [Google Chrome](#), [Opera](#))
 - Multimédiás programok
 - Játékok

3. Alkalmazott technológiák

A Visual Studio a Microsoft több programozási nyelvet tartalmazó fejlesztőkörnyezete, amely az évek során egyre több új programnyelvvvel bővült. Jelenleg a *F#*, *C++*, *C#* (ejtsd: Szí-sárp) és Visual Basic programozási nyelveket, valamint az XML-t támogatja. A csomag része még a MASM (Microsoft Macro Assembler) is, ami részleges assembly támogatást biztosít.

A Visual Studio Code (rövidítve: VSCode vagy VS Code) egy ingyenes, nyílt forráskódú kódszerkesztő, melyet a Microsoft fejleszt Windows, Linux és OS X operációs rendszerekhez. Támogatja a hibakeresőket, valamint beépített Git támogatással rendelkezik, továbbá képes az intelligens kódkezelésre (intelligent code completion) az IntelliSense segítségével. Ezen felül testre szabható, így a felhasználók megváltoztathatják a kinézetet (témát), megváltoztathatják a szerkesztő gyorsbillentyű-kiosztását, az alapértelmezett beállításokat és még sok egyebet.

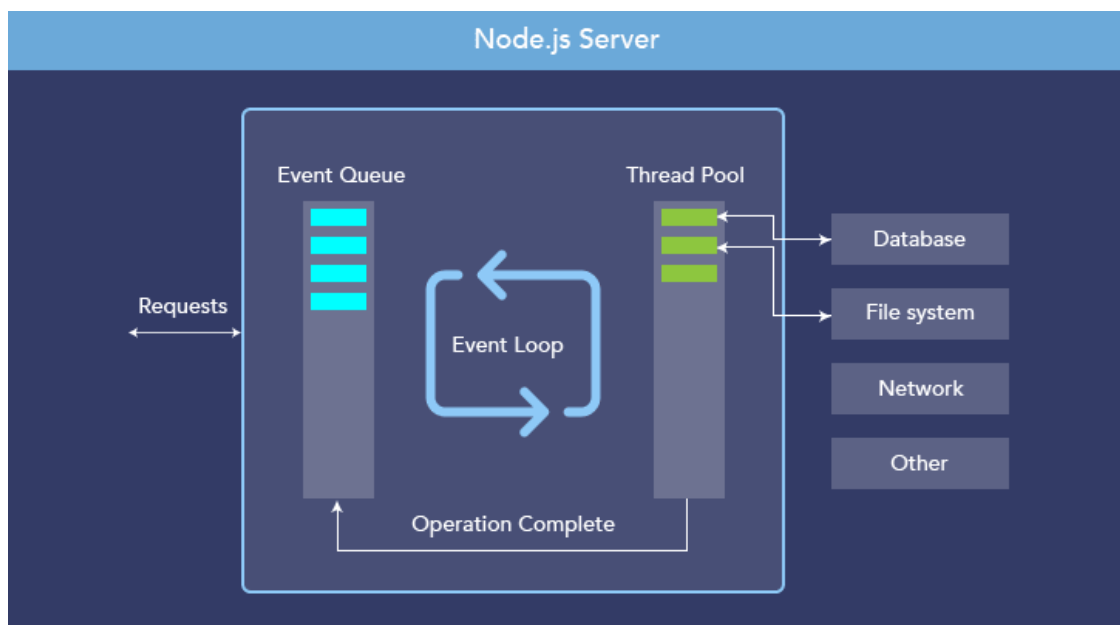
3.1. Felépítése

A webes programozás olyan számítógépes programozás, amely lehetővé teszi a weboldalak szerkesztését . Lehetővé teszi olyan alkalmazások létrehozását, amelyeket az Interneten vagy az Intraneten kívánnak telepíteni . Ezek a webalkalmazások különböző formájú weboldalakból állnak, például: „ Statikus ” oldalak : annak tartalmát nem befolyásolja az internet-felhasználó, aki azt kéri, és fejlődik a kézi beavatkozás annak forrását kódot .

" dinamikus " oldalak : tartalma úgy fejlődik, hogy a webhely kódját nem módosítják manuálisan, hanem a felhasználó interakciója , automatikus oldalgenerálás vagy számítás útján. A 2000-es évek eleje óta ez a webhelyek túlnyomó többségével történt.

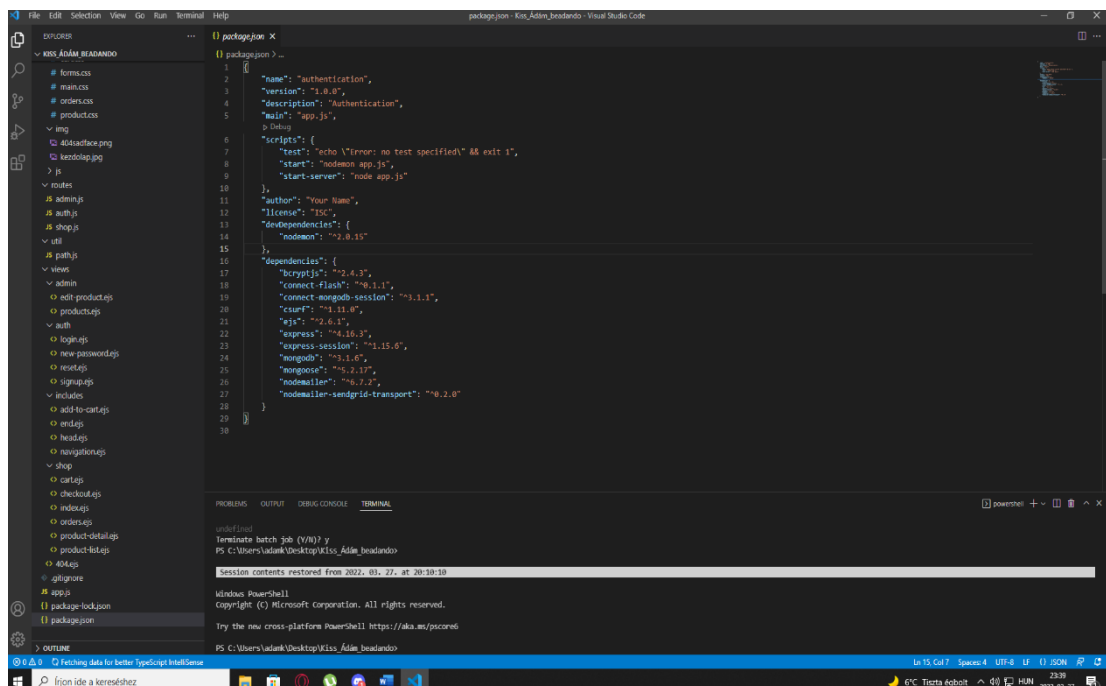
Az URL-t használó internethasználó számára a megfelelő weboldal, akár statikus, akár dinamikus, "kliensoldali" kódból (HTML , CSS , Javascript stb.) Áll, amelyet böngészője (Google Chrome , Firefox , stb.) Értelmez . ..) felhasználói felület előállítására . Ezt a webkiszolgáló által visszaküldött kódot a szerveroldali számítás hozhatja létre, vagy nem, adott programozási nyelveken keresztül, amelyek adatbázist , API-t stb. Hívhatnak meg .

Különböző technológiák és különféle nyelvek teszik lehetővé a webszervereken keresztüli weboldalak felépítését , amelyek ezért két elv szerint oszthatók: a programozási nyelvek a kliens oldalon vagy a szerver oldalon . Ezt a megkülönböztetést megkülönböztetjük a "kiszolgálóoldali" nyelvektől, amelyek kódját a webkiszolgálón futtatják, mielőtt a felhasználó böngészőjébe érkeznek, és az "ügyféloldali" nyelvektől, amelyek végrehajtása nem igényel számítást a weben. de csak az oldal letöltése után a felhasználó böngészőjének értelmezése.



3.2. Weboldal felépítése

A weboldal felépítése a következő kell egy `app.js` fájl amelyben az alap dolgokat mint például a szerver futását írjuk bele, hogy hányas porton induljon el és mit írjunk, hogyha elindul a használandó csomagok beimportálása után meg kell határoznunk az útvonalakat amelyeket a routes mappa létrehozása után meg is tehetünk fontos, hogy ejs fájlokba tároljuk az útvonalak által kimutatható weboldal képét. Ha megvan az `app.js` fájl beállítása akkor a következő dolgunk a függőségeknek a telepítése amely egy új terminál megnyitása után az `npm init` ezzel generáljuk le a `json` fájlt majd az `npm i` nevű parancs segítségével tudjuk telepíteni a fontos függőségeket. Ezek közé tartozik a `nodemon` ez arra való, hogy folyamatosan a szerver újraindítása nélkül tudjunk menteni és az eredményt látni és a weboldalunkon.



```
package.json
{
  "name": "authentication",
  "version": "1.0.0",
  "description": "Authentication",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "nodemon app.js",
    "start-server": "node app.js"
  },
  "author": "Your Name",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "connect-flash": "^0.1.1",
    "connect-mongodb-session": "^3.1.1",
    "ejs": "^3.1.6",
    "express": "^4.16.3",
    "express-session": "^1.15.6",
    "mongoose": "^5.11.6",
    "morgan": "^1.10.0",
    "nodemailer": "^6.2.7",
    "nodemailer-sendgrid-transport": "^0.2.0"
  }
}
```

Az `express` a `mongoose` amely a `mongoDB` szerverhez való kapcsolódást és igazából a háttérben dolgozó adatbázis kezelést tudja futtatni. Ezek az alap függőségek utána pedig a weboldal rendezéshez tartozik a fájlok elhelyezése a kinézet fájlok amelyek `ejs` formában vannak kellenek még a `css` állományok amelyek külön mappában vannak az egyszerűség miatt.


```
# forms.css
# main.css
# orders.css
# product.css
└─ img
   └─ 404sadface.png
   └─ kezdoalap.jpg
└─ js
└─ routes
   └─ JS admin.js
   └─ JS auth.js
   └─ JS shop.js
└─ util
   └─ JS path.js
└─ views
   └─ admin
      └─ <> edit-product.ejs
      └─ <> products.ejs
   └─ auth
      └─ <> login.ejs
      └─ <> new-password.ejs
      └─ <> reset.ejs
      └─ <> signup.ejs
   └─ includes
      └─ <> add-to-cart.ejs
      └─ <> end.ejs
      └─ <> head.ejs
      └─ <> navigation.ejs
   └─ shop
      └─ <> cart.ejs
      └─ <> checkout.ejs
      └─ <> index.ejs
      └─ <> orders.ejs
      └─ <> product-detail.ejs
      └─ <> product-list.ejs
      └─ <> 404.ejs
└─ .gitignore
└─ JS app.js
└─ {} package-lock.json
└─ {} package.json
```

Ez az elrendezés több szempontból is fontos az egyik részből aki csinálta a weboldalt az ne tévedjen és es egyszerűbben tudja megtalálni a fájlokat, routes-on belül vannak a navigációs részek az adminon belül a adminisztrációs felület, az auth-on belül vannak a loginnal kapcsolatos részek, shop-on belül pedig a vásárlással kosárba rendezéssel kapcsolat dolgok, utána van a view felület amely ahogy előbb említettem a megjelenést fogja tartalmazni. Meg kell egy 404-es státusz állapotú fájl amely egy rosszul beírt vagy elírt oldalt hibaként fogja jelezni amelynek a jelentése, hogy nem találta azt.

3.3. Node JS

A Node.js lehetővé teszi webszerverek és hálózati eszközök létrehozását JavaScript és különféle alapvető funkciókat kezelő „modulok” használatával. Modulok állnak rendelkezésre a fájlrendszer I/O-hoz, hálózatkezeléshez (DNS, HTTP, TCP, TLS/SSL vagy UDP), bináris adatokhoz (pufferekhez), kriptográfiai funkciókhoz, adatfolyamokhoz és egyéb alapvető funkciókhoz.

A Node.js moduljai olyan API-t használnak, amely csökkenti a szerveralkalmazások írásának bonyolultságát. A JavaScript az egyetlen nyelv, amelyet a Node.js natívan támogat, de számos JS-be fordítható nyelv is elérhető. Ennek eredményeként a Node.js alkalmazások CoffeeScript, Dart, TypeScript, ClojureScript és mások nyelven írhatók.

A Node.js-t elsősorban hálózati programok, például webszerverek készítésére használják. A legjelentősebb különbség a Node.js és a PHP között az, hogy a PHP-ben a legtöbb függvény a befejezésig blokkol (a parancsok csak az előző parancsok befejeződése után futnak le), míg a Node.js függvények nem blokkolnak (a parancsok egyidejűleg vagy akár párhuzamosan futnak, és használjon visszahívásokat a befejezés vagy a hiba jelzésére).

A Node.js hivatalosan támogatott Linuxon, macOS-en és Microsoft Windows 8.1-en, valamint Server 2012-n (és újabban), a SmartOS és az IBM AIX 2. szintű támogatásával, valamint a FreeBSD kísérleti támogatásával. Az OpenBSD is működik, és LTS verziók is elérhetők IBM i-hez (AS/400).

A megadott forráskód a hivatalosan támogatott operációs rendszerekhez hasonló operációs rendszerekre épülhet, vagy harmadik felek módosíthatják, hogy támogassanak másokat, például a NonStop OS-t és a Unix szervereket.

A Node.js eseményvezérelt programozást visz a webszerverekre, lehetővé téve gyors webszerverek fejlesztését JavaScriptben. A fejlesztők szálfűzés nélkül is létrehozhatnak méretezhető szervereket az eseményvezérelt programozás egyszerűsített modelljének használatával, amely visszahívásokat használ a feladat befejezésének jelzésére. Csomópont.

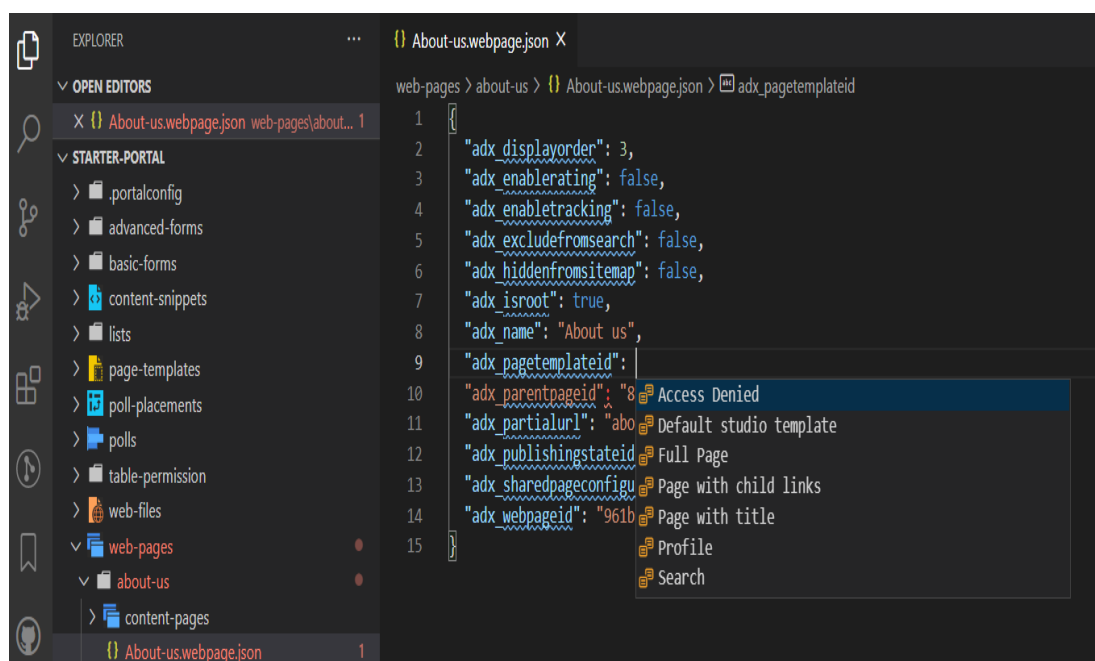
js összekapcsolja a szkriptnyelv (JavaScript) egyszerűségét a Unix hálózati programozás erejével.

A Node.js a Google V8 JavaScript motorjára épült, mivel nyílt forráskódú a BSD licenc alatt. Jártas az internet alapjaiban, mint például a HTTP, a DNS és a TCP. A JavaScript szintén jól ismert nyelv volt, így a Node.js elérhető a webfejlesztő közösség számára.

A modulok betölthető JavaScript-csomagok, amelyek meghatározott funkciókat biztosítanak az alkalmazás számára. A modulokat általában az npm parancssori eszköz használatával telepítik, azonban egyes modulok (például a HTTP-modul) a csomag alapsomagja Node.js részét.

A modulok telepítésekor a rendszer a node_modules könyvtárban, az alkalmazás könyvtárszerkezetének gyökerében tárolja őket. A node_modules-címtár minden modulja saját könyvtárat tart fenn, amely tartalmazza az összes olyan modult, amelytől függ, és ez a viselkedés minden modul esetében megismétlődik a függőségi láncon belül. Ez a környezet lehetővé teszi, hogy minden telepített modul saját verziókövetelményekkel rendelkezzen azokkal a modulokkal kapcsolatban, amelyektől függ, azonban jelentős könyvtárstruktúrát eredményezhet.

A node_modules könyvtár telepítése az alkalmazás részeként növeli a telepítés méretét a package.json vagy npm-shrinkwrap.json fájlhoz képest; Ugyanakkor garantálja, hogy az éles környezetben használt modulok verziói ugyanazok, mint a fejlesztésben használt modulok.



3.4. Fejlesztői környezet

A fejlesztés lépéseihez a környezet kialakításához kell egy Visual Studio termékcsalád amelyben benne foglalja a bővítményeket amely a weboldal készítés esetén a különböző függőségek telepítése amely a terminálon keresztül lesz telepítve. A mobilos applikáció felépítéséhez a különböző moduloknak a beilleszkedése kell a programba. Tesztekhez a serenity alkalmazást használjuk amely egy bővítmény bármelyik webhelyen elérhető.

A Selenium egy nyílt forráskódú ernyőprojekt számos eszköz és könyvtár számára, amelyek célja a böngésző automatizálásának támogatása. Lejátszási eszközt biztosít funkcionális tesztek készítéséhez anélkül, hogy meg kellene tanulnia egy teszt szkriptnyelvet (Selenium IDE). Ezenkívül egy tesztterület-specifikus nyelvet (Selenese) biztosít tesztek írásához számos népszerű programozási nyelven, beleértve a JavaScriptet (Node.js), a C#-t, a Groovy-t, a Java-t, a Perl-t, a PHP-t, a Python-t, a Ruby-t és a Scalát. A tesztek ezután futhatnak a legtöbb modern webböngészőn. A Selenium Windows, Linux és macOS rendszeren fut. Ez egy nyílt forráskódú szoftver, amelyet az Apache License 2.0 alatt adtak ki.

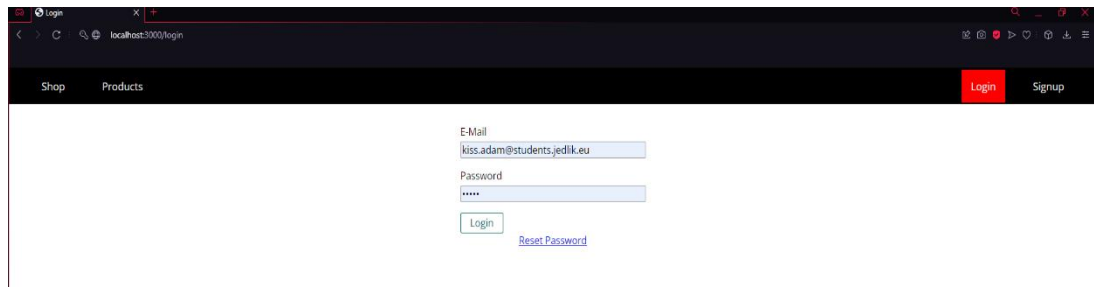
3. Felhasználói dokumentáció

3.1. Előkészítése

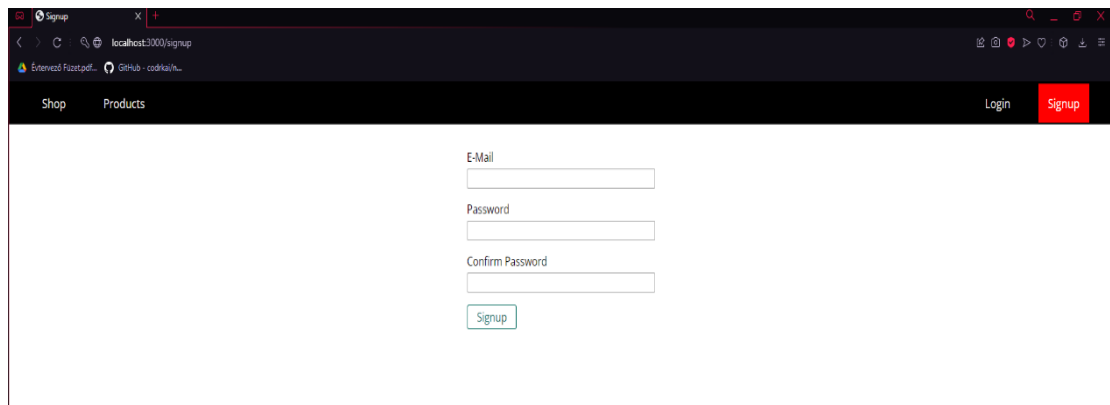
Mivel ennek a weboldalnak több része is van először a Visual Studio Code-s részét kell megemlítenünk mivel található benne egy package.json ezért a terminálon keresztül amit meg lehet nyitni PowerShell vagy esetleg CMD formátumban be kell írni, hogy npm i ilyenkor feltelepíti a függőségeket és ezáltal már a npm start paranccsal el is indul a háttérben futó szerverünkön a weboldal. A másik része a mobilos applikáció amelyben Visual Studio-s .sln végződésű fájl segítségével megtudjuk nyitni utána a CTRL + F5 segítségével tudjuk futtatni majd követjük a meglévő utasításokat és ott tudunk terméket hozzáadni ez egy adminisztrációs felület amelyben a meglévő termékeket tudjuk szerkeszteni és követni hogyha esetleg nem elérhető az adatbázis a számítógépünkön.

3.2. Főbb funkciói

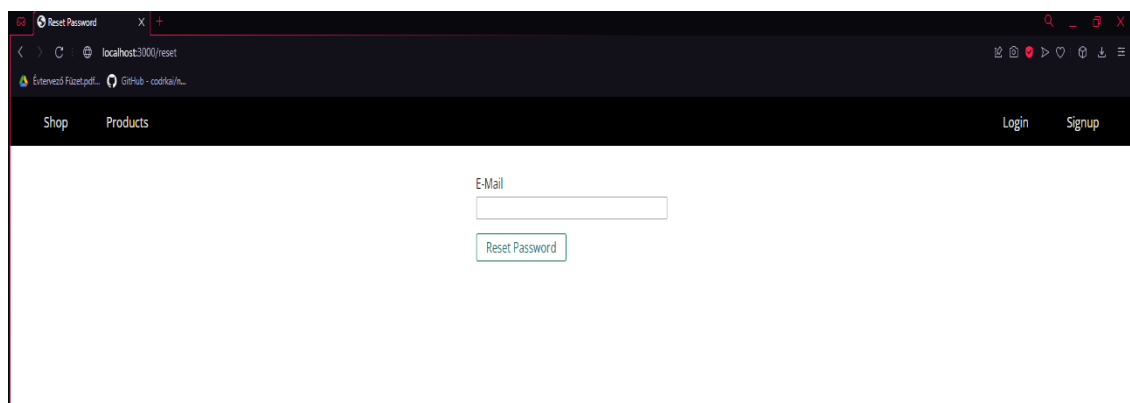
Weboldal részének a jellemzője, hogy amíg nem loginolunk be addig csak a menüsorban csak a regisztrációs fül illetve a főoldal elérhető ezzel meggátolva azt, hogy bejelentkezés nélkül tudjanak rendelni amelyet a következő képpen láthatunk:



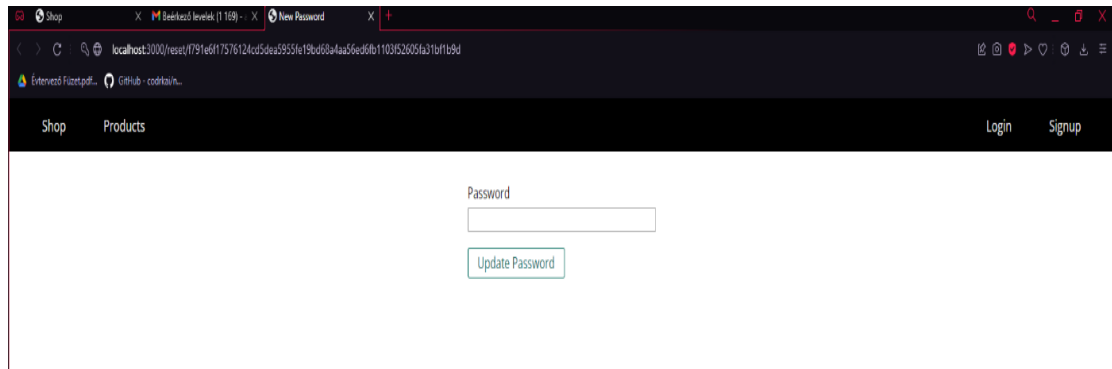
Következő funkció a regisztráció amely a következő képpen működik meg kell adni egy valós email címet majd két jelszót amellyel megerősítjük, hogy nem vagyunk robotok, ha készen vagyunk ezzel akkor be is jelentkezhetünk a login panelon keresztül, ha esetleg a email cím és a jelszó párosítás nem működik akkor igazából hiba üzenetet kapunk és korrigálhatjuk azt:



A végső funkció amely elengedhetetlen az oldalon ha esetleg a jelszó és az email bejelentkezés nem sikerül akkor igazából van egy jelszó visszaállítás:

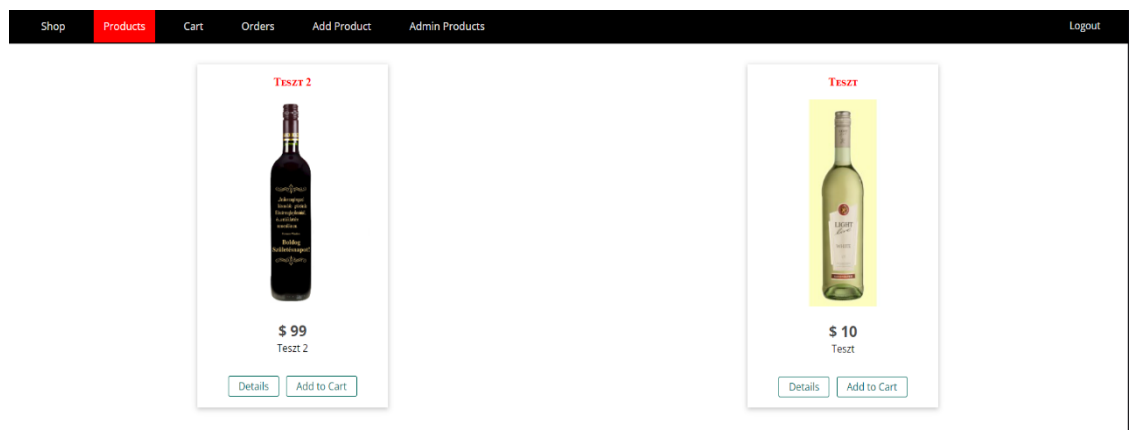


Ez a funkció úgy működik, hogy az emailben kapott linket megkapjuk amely átirányít minket ide és itt az új jelszó beírásával változtatjuk meg azt és ezentúl már az lesz nekünk a jelszavunk:

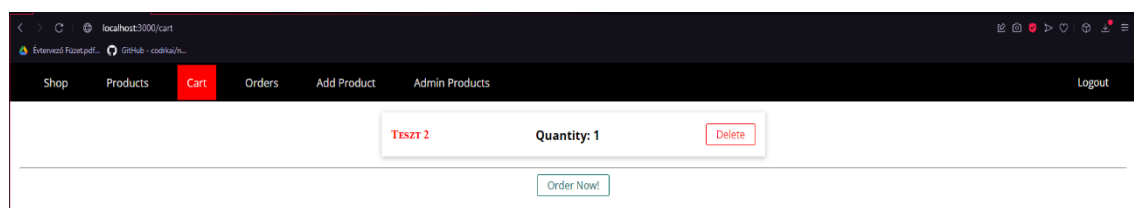


3.3. Vásárlói funkciók

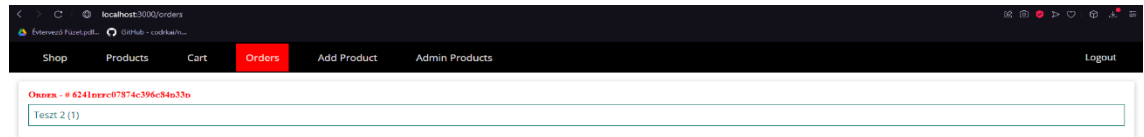
Az első az a funkció amellyel szembe találjuk magunkat azok a termékek fül ahol a különböző borok közül választhatunk és amelyben a nevét a leírását az árát is megtekinthetjük itt két funkció közül is választhatunk a terméken belül a részletek illetve a kosárba tétel:



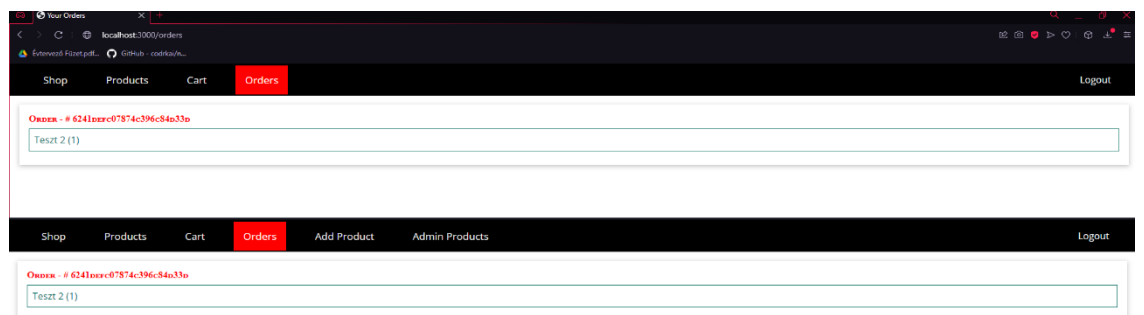
Ha megvan a kosárba tétel akkor a következő funkciók lesznek elérhetők a kosár fül alatt. A kosárba látható annak a terméknek a neve amelyet bele tettünk és a darabszám amennyiszer bele tettünk ezt ki is törölhetjük ha nem megfelelő terméket tettünk bele a kosárba, de ha a megfelelő termék van a kosárban az a megrendelésekbe fog kerülni:



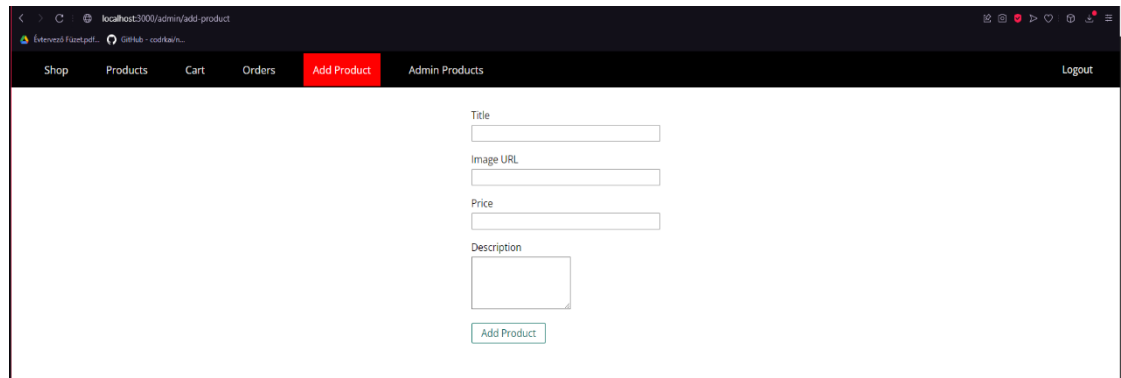
A megrendelés fülön a következőt kapjuk: megrendelés számát amely beazonosítja, hogy tényleg a mienk a rendelés minden különböző felhasználó más számot fog kapni így elkerülve az adatbázis hibát. A termék neve illetve a mennyiség amely a következő képen látható:



Ezzel a felhasználói felület vásárlási része kimerült most pedig jönnek a következő funkciók amelyek igazából az adminisztrációs részét fogják alkotni amelybe beletartozik, hogy egy új terméknek a felvétel és meglévő termékek amiket felvettünk annak a szerkesztése illetve törlése ezeket csak azok a felhasználók tudják használni akik nem sima user joggal rendelkeznek hanem az adatbázison keresztül admin joggal fel lett ruházva. Az alap termékeket amelyek az adatbázisban benne vannak azokat nem lehet törölni a visszaélések miatt ha esetleg egy olyan user kap admin jogot aki úgymond „rossz akaró” ne tudja a meglévő adatokat törölni csakis azokat amiket ő hozott létre. Más admin nem tudja törölni a másik terméket amit egy másik admin hozott létre ezzel is csökkentve ez a funkciót. Az első az admin jog nélkül utána az admin joggal a képen:



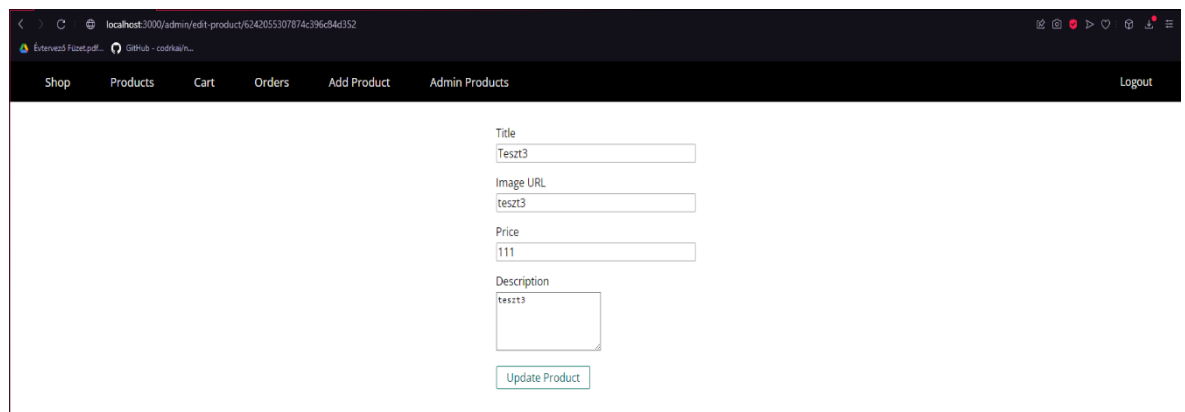
Az utolsó két menüpont amely a legfontosabb szinte az egész weboldalon, hogy az termék hozzáadása amellyel az admin joggal való felhasználó tud hozzáadni már a meglévő termékekhez ott meg kell adni a termék nevét az árát illetve a leírását majd egy kép url amely a fényképét fogja jelenteni:



The screenshot shows a web browser at localhost:3000/admin/add-product. The navigation bar includes 'Shop', 'Products', 'Cart', 'Orders', 'Add Product' (highlighted in red), 'Admin Products', and 'Logout'. The form contains the following fields:

- Title:
- Image URL:
- Price:
- Description:
- Submit button: Add Product

Az utolsó menüpont amely az a termékek amelyeket az admin vett fel itt lehetőség szerint tudjuk törölni a terméket illetve tudjuk szerkeszteni azt ilyenkor az adatbázisban. Fontos, hogy ezt az admin jog beállítást csak az adatbázisban tudjuk megadni másképpen minden felhasználó admin lesz és ezáltal tudnak új terméket felvenni.



The screenshot shows a web browser at localhost:3000/admin/edit-product/6242055307874c396c84d352. The navigation bar includes 'Shop', 'Products', 'Cart', 'Orders', 'Add Product', 'Admin Products', and 'Logout'. The form contains the following fields:

- Title:
- Image URL:
- Price:
- Description:
- Submit button: Update Product

Fontos még, hogy minden apró változtatás a terminálon belüli log rendszer segítségével lehet ellenőrizni és ezáltal pontosan megnézni, hogy ténylegesen mit változtattunk ha esetleg valami sikertelen lesz akkor a terminál mindenben a segítségünk lesz.

5. Fejlesztői dokumentáció

Amiközben dolgoztunk a szakdolgozaton több tényezőt is figyelembe kellett venni ami a fontos volt, hogy szikronban dolgozzunk ha valamelyikünk végzett az adatbázis és a weboldal általi kapcsolat létesítésére akkor a meglévő munkában már lehetett dolgozni a dizájn elemeken amely elfogadhatóvá teszi ezt a leegyszerűsített nyers weboldalt, ennek az eredménye majd láthatóvá lesz, hogy kis okoskodással, hogyan lehetett megcsinálni ezt az alap követelményt. Ha ezzel végeztünk akkor a dokumentációt a nulláról olyan formában építettük fel ahogy készült a projekt első sorban azokat az alapokat kellett megteremteni amely az alap weboldal felépítette az nem más mint a fejlesztői környezet kialakítása Visual Studio Code segítségével itt a package.json fájlba csakis azokat a fájlokat tartalmazza amelyek a weboldalt felépítik vagy esetleg az adatbázishoz való csatlakozáshoz elősegítik a többi felesleges dolgot kitöröltük. Második fontos szikron folyamat a mobilos applikáció amely Visual Studioban lett elkészítve ez annak a részét valósítja meg a weboldalnak, hogy telefonon is el lehessen érni az admin funkciót amellyel lehet törölni illetve felvenni új termékeket a weboldalra ez elősegíti a gyors elérést bármely eszközről ami Androidos. A legutolsó lépés pedig az amikor az egészet leteszteljük amelyhez a fent említett Seleniumot használjuk amelyben a következőket ellenőriztük: belépés, regisztráció, jelszó visszaállítás, kosárba való hozzáadás és annak törlése, rendelés funkció, új termék felvétele annak módosítása. Ennek az összehangolt szinte már szikronban való munkáját nehéz volt picit megoldani az iskola után még némelyikünknek dolgozni kellett így az esti munkákban tudtunk temetkezni ezt a Discord nevű applikáción tudtunk lebonyolítani amelyet hetente tudtunk összeülni.

5.1. Kezdet

Elsőnek az volt a cél, hogy a megfelelő package.json kialakítása után az első dolgunk egy app.js létrehozása amely a következőket tartalmazza:

```
const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const session = require('express-session');
const MongoDBStore = require('connect-mongodb-session')(session);
const csrf = require('csurf')
const flash = require('connect-flash')
```

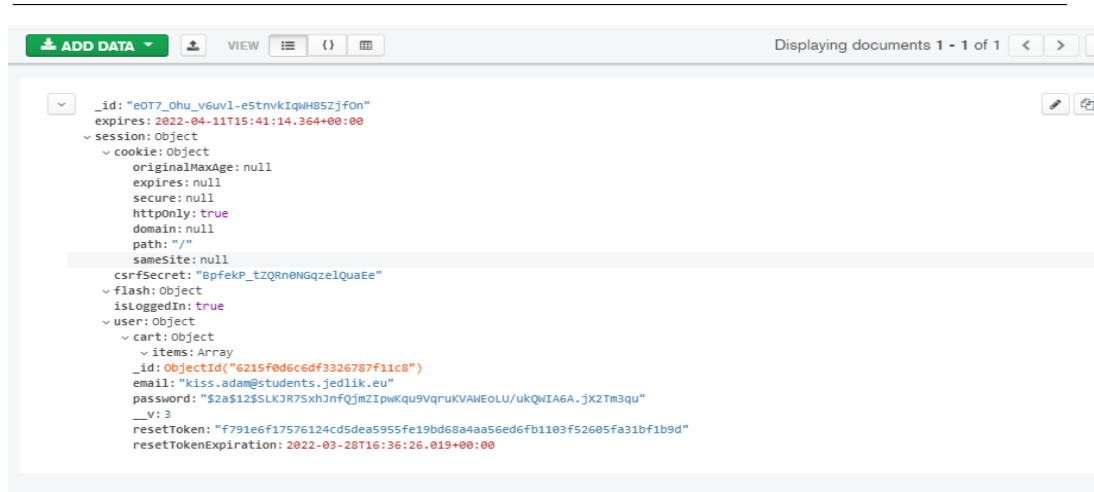
Ezek a következőket jelentik:

A path az az útválasztást segíti elő, az express amelyben különböző funkciók vannak DELETE, POST, GET, body-parser az előhívást segíti az adatbázisnak például az id-jét egy adott terméknek, a mongoose felel az adatbázis feladatért, session ez arra való, hogy maga a usert rögzítse(email, jelszó, token, bejelentkezési idő), a mongodb pedig a csatlakozást biztosít az adatbázis részéhez, csrf pedig a tokenért felelős amely elfelejtett jelszó esetén generálódik és lehetővé teszi az emailban való értesítést, a flash egy üzenet közvetítés amely a csrf függőséggel együtt az email token küldést biztosítja a felhasználók számára.

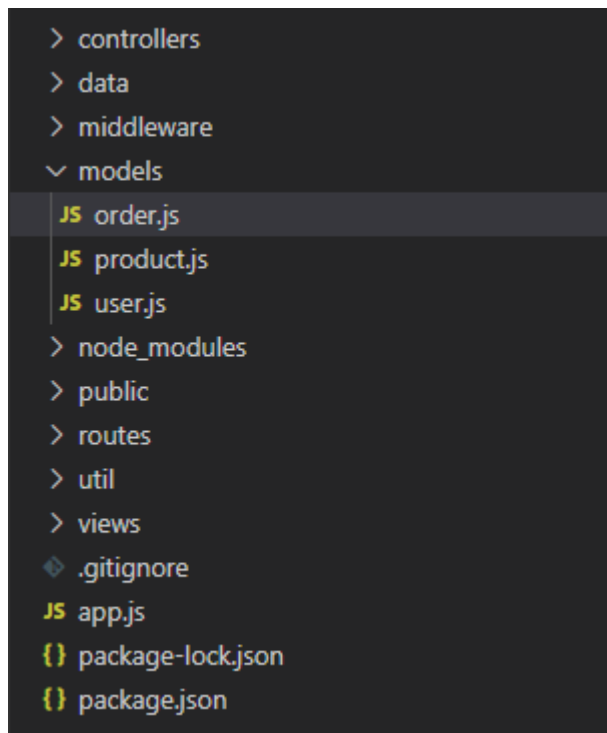
A következő lépés az adatbázishoz való csatlakozás, amely a következőt jelenti:

```
14 const MONGODB_URI =
15   'mongodb+srv://codeine:Codeine@cluster0.wq3bg.mongodb.net/shop?retryWrites=true&w=majority';
16
17 const app = express();
18 const store = new MongoDBStore({
19   uri: MONGODB_URI,
20   collection: 'sessions'
21 });
22 const csrfProtection = csrf()
```

Elsősorban itt a csatlakozás függvényét láthatjuk amelyben a MongoDB csatlakozási parancsát láthatjuk elsősorban a név illetve a jelszó amelyet bele kell írni, hogy az adatbázisba való belépés rendben legyen ez jelen esetben: codeine és Codeine majd a shop nevű kollekcióhoz csatlakozik ha esetleg nem létezne ez a kollekció akkor létrehozza az adott adatbázis. Ha ezzel megvagyunk akkor az express funkciójába tartozik a kapcsolati pár, amely el van látva csrfProtectionnal amely azt eredményezi, hogy a jelszót letitkosítja amely a következő képen látható:



Az adatbázishoz az adatokat a következő módon tudjuk megadni, hogy miket mentsen el és mi szükséges az adatbázishoz a kommunikációhoz ezek a models mappába találhatóak meg:



Az orders mappában található fájlok értelmezése a következő: az orderben találhatóak a bejelentkezési adatok illetve a felhasználó által felvett termékek idje, hogy megtudja különböztetni a többi felhasználótól. A productban találhatóak a termék ismertető jegyei amelyet felvételkor kerül rögzítésre. A userben pedig a felhasználó adatait tartalmazza amely az email a jelszó illetve a tokenet amely jelszó változtatás

után kap újat és még ami fontos a kosár tartalma. Amikor egy ilyen fájl hozunk létre akkor egy Schema-t hozunk létre és ott megnevezzük a táblát és az azon belüli attribútumokat ott megtudjuk adni, hogy mi legyen a kötelező és melyik a nem. Itt megkell adni, hogy milyen a type-ja itt jelen esetben a productnak Object mivel ez egy termék és ahogy látszik, hogy kötelező mező ezt is meg kell adni, hogy ne legyenek üres mezők az adatbázisban.

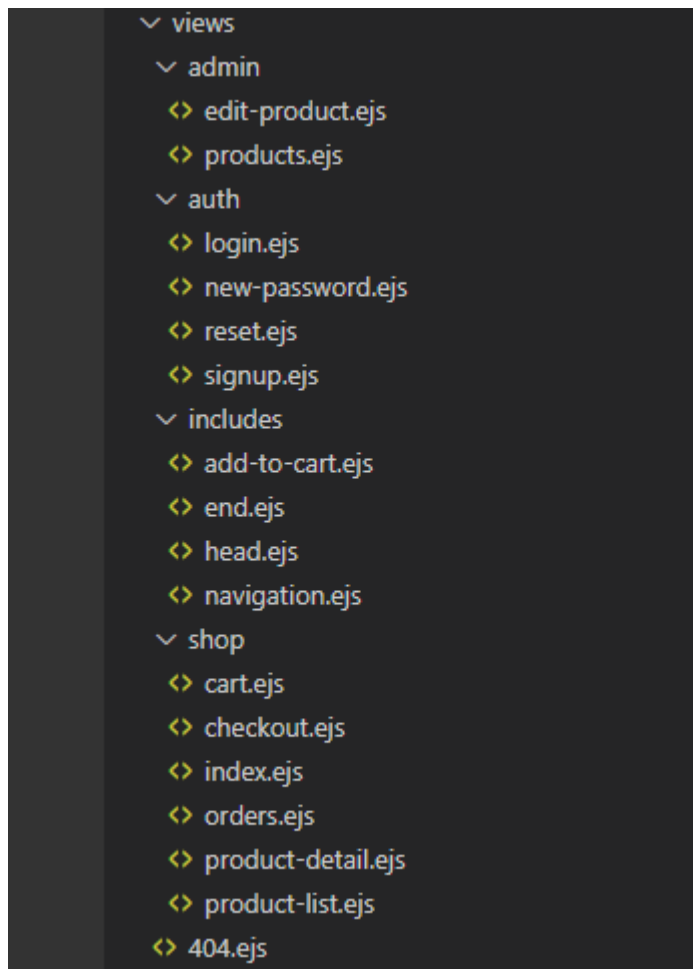
```
const orderSchema = new Schema({
  products: [
    {
      product: { type: Object, required: true },
      quantity: { type: Number, required: true }
    }
  ],
  user: {
    email: {
      type: String,
      required: true
    },
    userId: {
      type: Schema.Types.ObjectId,
      required: true,
      ref: 'User'
    }
  }
});
```

Ha az adatbázis csatlakozás jó akkor kezdhethetjük az útvonalakat kialakítani itt jelen esetben ami kell az az admin az auth és a shop útvonal. Az adminnak az a feladata, hogy az add-product a products-ot az edit-productot is a delete-productot tudja kezelni ezeknek fontos, hogy másik fájlba legyenek az egyszerűség kedvéért és ekkor nem fognak eltévedni esetleg azok az emberek akik kíváncsiak a program belső felépítésére. Az auth útvonal pedig teljes mértékben levezeti a bejelentkezést a regisztrációt, a kijelentkezést illetve a jelszavak kezelését is szóval ha ez nem lenne akkor nem tudnánk felhasználót létrehozni. A shop útvonal felel azokért a részekért hogyha már be vagyunk jelentkezve akkor tudjunk terméket hozzáadni a kosárhoz a kosár tartalmát tudjuk megnézni ki tudjunk törölni a kosárból illetve a rendelés földre kattintva előállít egy megrendelést is amellyel véglegesíteni tudjuk a vásárlásunkat.

<ul style="list-style-type: none"> > controllers > data > middleware > models > node_modules ▼ public > css > img > js ▼ routes JS admin.js JS auth.js JS shop.js > util > views ◆ .gitignore JS app.js { } package-lock.json { } package.json 	<pre>1 const path = require('path'); 2 3 const express = require('express'); 4 5 const adminController = require('../controllers/admin'); 6 const isAuthenticated = require('../middleware/is-auth'); 7 8 const router = express.Router(); 9 10 // /admin/add-product => GET 11 router.get('/add-product', isAuthenticated, adminController.getAddProduct); 12 13 // /admin/products => GET 14 router.get('/products', adminController.getProducts); 15 16 // /admin/add-product => POST 17 router.post('/add-product', adminController.postAddProduct); 18 19 router.get('/edit-product/:productId', adminController.getEditProduct); 20 21 router.post('/edit-product', adminController.postEditProduct); 22 23 router.post('/delete-product', adminController.postDeleteProduct); 24 25 module.exports = router;</pre>
---	---

5.1. Menük

A második nagy feladat a menük kialakítása ezeken belül meg kellett tervezni, hogy milyen útvonal hova vezessen ezért létrehoztunk egy views mappát amelyet felbontottunk: admin, auth, includes és shop részre és még a views mappán belül létre kellett hoznunk egy 404-es error státusz kóddal ellátott ejs fájlt amely egy hibás útvonal beírása vagy esetleg átirányítás esetén dobja ki a hibakódot és az üzenetet.



Az admin mappában a következők találhatók az edit-product és a product amelyek azért felelnek, hogy amikor az adminisztrátor felvett egy terméket és esetleg szerkeszteni akarja vagy esetleg törölni itt tudja vizuálisan megtenni mert ezek az oldalak felelnek a kinézetért. Az auth mappában a bejelentkezésért felelős kinézeti elemek találhatók meg. Az includes mappában pedig a navigációs menüsor illetve

az oldalnak az eleje illetve a vége található amelyben lehet tenni footert. A shop mappában pedig a menüsor kinézet elemei találhatók. A 404-es fájl pedig önmagáért beszél.

```

<div class="topnav">
  <header class="main-header">
    <button id="side-menu-toggle">Menu</button>
    <nav class="main-header__nav">
      <ul class="main-header__item-list">
        <li class="main-header__item">
          <a class="<%= path === '/' ? 'active' : '' %>" href="/">Shop</a>
        </li>
        <li class="main-header__item">
          <a class="<%= path === '/products' ? 'active' : '' %>" href="/products">Products</a>
        </li>
        <% if (isAuthenticated) { %>
          <li class="main-header__item">
            <a class="<%= path === '/cart' ? 'active' : '' %>" href="/cart">Cart</a>
          </li>
          <li class="main-header__item">
            <a class="<%= path === '/orders' ? 'active' : '' %>" href="/orders">Orders</a>
          </li>
          <li class="main-header__item">
            <a class="<%= path === '/admin/add-product' ? 'active' : '' %>" href="/admin/add-product">Add Product
            </a>
          </li>
          <li class="main-header__item">
            <a class="<%= path === '/admin/products' ? 'active' : '' %>" href="/admin/products">Admin Products
            </a>
          </li>
        <% } %>
      </ul>
    </nav>
  </header>
</div>

```

Elsősorban ami fontos az itt látható kódrészletben az nem más minthogy úgy épül fel a menü szerkezete, hogy kell egy navbar amibe amiben bele kell helyoznunk ez esetben a Shop a Products a Cart az Orders és az Add-Product mellett az Admin Productot a Path jellemzője az az útvonal ahonnan átirányít minket a weboldal az active rész azt jelenti, hogy amikor rákattintunk az a css stílusnak megfelelő jelöléssel fogja jelezni, hogy jelenleg ezen a menü ponton vagyunk a href része pedig amely a webcíme mellett azt jelzi, hogy hova navigált minket az oldal. Az isAuthenticated dolog pedig azt jelenti, hogy a sikeres bejelentkezés után csak azok a menü pontok lesznek elérhetőek amelyek bent vannak a megfelelő felsorolásban a többi csak azoknak lesz megjelenítve akik nem jelentkeztek be.



Itt is van a két mód közötti különbség amely látható a bejelentkezés előtt majd a bejelentkezés utáni állapotot mutatja

5.2. Belső funkciók felépítése

Első sorban kell egy külön mappa kialakítása amely a controllers nevet kapta itt különböző js fájlokat hoztunk létre amely a következő módon épülnek fel. Kell egy admin fájl amelyben a product fájlból szedi ki az adatokat amely a különböző termékek létrehozása, szerkesztése illetve a törlése érdekében van.

```
exports.postAddProduct = (req, res, next) => {  
  const title = req.body.title;  
  const imageUrl = req.body.imageUrl;  
  const price = req.body.price;  
  const description = req.body.description;  
  const product = new Product({  
    title: title,  
    price: price,  
    description: description,  
    imageUrl: imageUrl,  
    userId: req.user  
  });  
  product  
    .save()  
    .then(result => {  
      // console.log(result);  
      console.log('Created Product');  
      res.redirect('/admin/products');  
    })  
    .catch(err => {  
      console.log(err);  
    });  
};
```

Itt a következőket láthatjuk a termékek létrehozásában meg kell adni az alapvető adatokat amely a név a kép url az érték a leírás majd ezt egy új termékként hozza létre az adatbázisunkban. A save() metódussal mentjük el az adatbázisba az elkészült terméket amely ezzel feltölti

a táblát adatokkal kell egy .then ág amely azért felelős átirányítson ha megtörtént ennek a létrehozása az admin/products célpontra, a .catch ág azért kell mert a hibák elkerülése végett a console-ba ki kell íratnunk esetleges hiba esetén, hogy melyik része történt meg és melyik része nem történt meg a függvénynek.

A másik funkció az a szerkesztés amely a következő módon működik belépünk a termékek menüpontba és ha rámegyünk egy általunk létrehozott termék szerkesztéséhez ez belép egy edit mode-ba amely a fentiek alapján a megadott adatokat tudjuk módosítani ha ezzel megvagyunk akkor visszairányít a termékek nevű menüpontban amit még fontos tudni, hogy ez az adatbázisban úgy keres rá, hogy prodID alapján amely pontosítja, hogy csak ezt módosítjuk ezt mind a beépített függvények segítségével tudjuk használni.

```
exports.getEditProduct = (req, res, next) => {
  const editMode = req.query.edit;
  if (!editMode) {
    return res.redirect('/');
  }
  const prodId = req.params.productId;
  Product.findById(prodId)
    .then(product => {
      if (!product) {
        return res.redirect('/');
      }
      res.render('admin/edit-product', {
        pageTitle: 'Edit Product',
        path: '/admin/edit-product',
        editing: editMode,
        product: product
      });
    })
    .catch(err => console.log(err));
};

exports.postEditProduct = (req, res, next) => {
  const prodId = req.body.productId;
  const updatedTitle = req.body.title;
  const updatedPrice = req.body.price;
  const updatedImageUrl = req.body.imageUrl;
  const updatedDesc = req.body.description;

  Product.findById(prodId)
    .then(product => {
      if (product.userId.toString() !== req.user._id.toString()) {
        return res.redirect('/');
      }
      product.title = updatedTitle;
      product.price = updatedPrice;
      product.description = updatedDesc;
      product.imageUrl = updatedImageUrl;
      return product.save()
        .then(result => {
          console.log('UPDATED PRODUCT!');
          res.redirect('/admin/products');
        })
    })
    .catch(err => console.log(err));
};
```


Következő egyben utolsó admin lehetőség az nem más mint ezeket a termékeket amelyeket felvettünk annak megjelenítése az admin menüben és ezeknek a törlése a következő módon történik. Ha a saját termékünket akarjuk megkeresni azt a termék menüben találjuk itt látható, hogy userID vagyis mindenki a sajátját látja aki egyáltalán hozott létre sajátot.

A másik dolog pedig a saját termék törlése itt a létrehozott termék prodID és a felhasználó userID összevetése után törölhető a megadott termék itt fontos, hogy deleteOne funkciót használjuk a beépített függőségek mert erre ez a legcélszerűbb.

```
exports.getProducts = (req, res, next) => {
  Product.find({userId: req.user._id})
    // .select('title price -_id')
    // .populate('userId', 'name')
    .then(products => {
      console.log(products);
      res.render('admin/products', {
        prods: products,
        pageTitle: 'Admin Products',
        path: '/admin/products'
      });
    })
    .catch(err => console.log(err));
};

exports.postDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  Product.deleteOne({ _id: prodId, userId: req.user._id })
    .then(() => {
      console.log('DESTROYED PRODUCT');
      res.redirect('/admin/products');
    })
    .catch(err => console.log(err));
};
```

Következő rész a bejelentkezés a regisztráció illetve az a visszaállított jelszó rész található amely szinte a legfontosabb dolog a weboldal életében amely azt kezeli, hogy egyáltalán emberek érdeklődését tudják kifejezni a regisztráció segítségével itt is a beépített funkciókat használtuk amely a modelsekből a user-re volt szükségünk, hogy a felhasználói adatokat tudjuk tárolni illetve ezeket a tárolt adatokat az visszaállított jelszó funkcióval tudjuk alkalmazni itt is sessionra van szükségünk amely megjegyzi az adott felhasználó adatait a tokenjét a felhasználó nevét illet jelszavát titkosítva így a titkosítást visszafejtve tud bejelentkezni az adott illető és itt még az admin jog adására is van lehetőségünk.

```
const bcrypt = require('bcryptjs');
const nodemailer = require('nodemailer')
const sendgridTransport = require('nodemailer-sendgrid-transport')
const crypto = require('crypto')
const User = require('../models/user');
const res = require('express/lib/response');

const transporter = nodemailer.createTransport(sendgridTransport({
  auth: {
    api_key: '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'
  }
}))

exports.getLogin = (req, res, next) => {
  res.render('auth/login', {
    path: '/login',
    pageTitle: 'Login',
    errorMessage: req.flash('error')
  });
};

exports.getSignup = (req, res, next) => {
  res.render('auth/signup', {
    path: '/signup',
    pageTitle: 'Signup'
  });
};
```

Itt ahogy láthatjuk a megfelelő függőségek exportálása után tudjuk használni az üzenet küldő rendszer amely a megfelelő Api Key segítségével küldd üzenetet az adott email címre amelynek elfelejtettük a jelszavát illetve a visszaállítási token is erre kapjuk meg.

Itt még a bejelentkezés és a regisztráció látható amely hibás bejelentkezés vagy esetleg a jelszó megerősítéskor hibát dob fel ha ezt a flash nevű függőség biztosítja. Ezen belül a SendGrid nevű oldalt használtuk amely megfelelő ehhez a funkció betöltéséhez. A SendGrid (más néven Twilio SendGrid) egy denveri, Colorado állambeli ügyfélkommunikációs platform tranzakciós és marketinges e-mailekhez. A SendGrid felhőalapú szolgáltatást nyújt, amely segíti a vállalkozásokat az e-mailek kézbesítésében. A szolgáltatás különféle típusú e-maileket kezel, beleértve a szállítási értesítéseket, baráti kéréseket, regisztrációs visszaigazolásokat és e-mailes hírleveleket.

```
exports.postReset = (req, res, next) => {
  crypto.randomBytes(32, (err, buffer) => {
    if (err) {
      console.log(err);
      return res.redirect('/reset');
    }
    const token = buffer.toString('hex');
    User.findOne({email: req.body.email})
      .then (user => {
        if (!user) {
          req.flash('error', 'No account with that email found.')
          return res.redirect('/reset')
        }
        user.resetToken = token;
        user.resetTokenExpiration = Date.now() + 3600000;
        return user.save();
      })
      .then(result => {
        res.redirect('/')
        transporter.sendMail({
          to: req.body.email,
          from: 'kiss.adam@students.jedlik.eu',
          subject: 'Password reset',
          html: `
            <p>You requested a password reset </p>
            <p>Click this <a href="http://localhost:3000/reset/${token}">link</a> to set a new password.</p>
          `
        })
      })
      .catch(err => {
        console.log(err)
      })
  })
}
```

Itt látható a jelszó visszaállítás ugye a jelszót `crypto.randomBytes()` metódus segítségével kriptográfiailag jól felépített mesterséges véletlen adatot és az írott kódban generálandó bájtok számát állítja elő. Itt a `User.findOne()` metódust használtuk, hogy az email cím szerint nézze a bejelentkezést ha esetleg nem talált a megadott fiókhoz kapcsolódó emailt találni akkor visszadobta az előző oldalra természetes a hibával együtt. Ha esetleg sikerült akkor a kódot emailban megkaptuk amelyre ha beírtuk akkor új jelszót változtathattunk és következőnek már ezzel a segítségével tudunk belépni az oldalra. Az email tartalma, hogy egy jelszó visszaállítást kezdeményeztünk amely a következő tokenra való kattintás segítségével tudunk eljutni a kívánt oldalra itt a `transporter.sendMail()` metódust használtuk ennek a segítsége érdekében, ide is kell egy `.catch` ág amely meggátolja, hogy a hibáink mellett esetleg elmenjünk. Sokszor szokott hiba lenni a `SendGrid` nevű oldalnak hogyha túl sokszor kérünk emailt akkor letilthatja a fiókunkat ezt a fejlesztés során vettük észre és egy picit hátráltatta ennek a funkciónak a tesztelését illetve az élesben való működését.

```
exports.postNewPassword = (req, res, next) => {
  const newPassword = req.body.password;
  const userId = req.body.userId;
  const passwordToken = req.body.passwordToken;
  let resetUser;

  User.findOne({resetToken: passwordToken, resetTokenExpiration: {$gt: Date.now()}, _id: userId})
    .then(user =>{
      resetUser = user;
      return bcrypt.hash(newPassword, 12);
    })
    .then(hashPassword => {
      resetUser.password = hashPassword;
      resetUser.resetToken= undefined;
      resetUser.resetTokenExpiration = undefined;
      return resetUser.save();
    })
    .then(result => {
      res.redirect('login');
    })
    .catch(err => {
      console.log(err)
    })
}
```

Itt látható az új jelszónak a véglegesítése erről azt kell tudni, hogy létrehoz egy newPassword attribútumot amelyet userID és passwordTokenhez köt és ezt neveztük el resetUser funkciónak itt a User.findOne() metódust használjuk amelyben még el is tárol egy dátumot, hogy esetleg vissza lehessen nézni, hogy mikor volt az utolsó módosítás. Utána érkezik az új jelszó úgymond hashelése ami annyit takar, hogy a bcrypt az egy adaptív funkció amely idővel az iterációk számát növeli, így ellenálló marad a nyers támadásokkal szemben, kényszeríti a keresési támadásokat még növekvő számítási teljesítmény mellett is. Így megkapjuk az új jelszót amely ugyanúgy le van korlátozva a hozzáférése mint a régiek is.

```
1 exports.get404 = (req, res, next) => {
2   res.status(404).render('404', {
3     pageTitle: 'Page Not Found',
4     path: '/404'
5   });
6 };
7
```

Ami még megemlíthető, hogy 404-es error.js is bekerült ide amely kapott egy státusz kódot és egy szöveget ezt pedig ejs fájlban a kinézetét saját formánkra dizájnolhattuk.

```
exports.getProducts = (req, res, next) => {
  Product.find()
    .then(products => {
      console.log(products);
      res.render('shop/product-list', {
        prods: products,
        pageTitle: 'All Products',
        path: '/products'
      });
    })
    .catch(err => {
      console.log(err);
    });
};
```

```
exports.getProduct = (req, res, next) => {
  const prodId = req.params.productId;
  Product.findById(prodId)
    .then(product => {
      res.render('shop/product-detail', {
        product: product,
        pageTitle: product.title,
        path: '/products'
      });
    })
    .catch(err => console.log(err));
};
```

```
exports.getCart = (req, res, next) => {
  req.user
    .populate('cart.items.productId')
    .execPopulate()
    .then(user => {
      const products = user.cart.items;
      res.render('shop/cart', {
        path: '/cart',
        pageTitle: 'Your Cart',
        products: products
      });
    })
    .catch(err => console.log(err));
};
```

```
exports.postCart = (req, res, next) => {
  const prodId = req.body.productId;
  Product.findById(prodId)
    .then(product => {
      return req.user.addToCart(product);
    })
    .then(result => {
      console.log(result);
      res.redirect('/cart');
    });
};
```

```
exports.postCartDeleteProduct = (req, res, next) => {
  const prodId = req.body.productId;
  req.user
    .removeFromCart(prodId)
    .then(result => {
      res.redirect('/cart');
    })
    .catch(err => console.log(err));
};
```

```
exports.getIndex = (req, res, next) => {
  Product.find()
    .then(products => {
      res.render('shop/index', {
        prods: products,
        pageTitle: 'Shop',
        path: '/'
      });
    })
    .catch(err => {
      console.log(err);
    });
};
```

```
const Product = require('../models/product');
const Order = require('../models/order');
```

Itt a következő látható amely már a shop.js fájlban található ami már a weboldal legesleg végső lépése amely a models mappából exportálható termékeknek a megjelenítését amely az összeset jeleníti meg és tartalmazza a részletes leírását is ez a kód részlet. Ugyanúgy vannak itt is .catch ágak amelyek megakadályozzák a hibáknak a lehetőségét és a megelőzhető eljárását. Ebbe még beletartozik a következő fontos rész a rendelésnek amely a kosár funkció itt a user egy meghatározott prodId-t fog felvenni amelyet tud törölni is arra is gondoltunk ha esetleg az egyik felhasználó felvesz valamit és betesz a kosarában akkor bejelentkeztünk egy másik profilba ezzel meggátoltuk a duplikálódását a terméknek.

```
exports.postOrder = (req, res, next) => {
  req.user
    .populate('cart.items.productId')
    .execPopulate()
    .then(user => {
      const products = user.cart.items.map(i => {
        return { quantity: i.quantity, product: { ...i.productId._doc } };
      });
      const order = new Order({
        user: {
          email: req.user.email,
          userId: req.user
        },
        products: products
      });
      return order.save();
    })
    .then(result => {
      return req.user.clearCart();
    })
    .then(() => {
      res.redirect('/orders');
    })
    .catch(err => console.log(err));
};

exports.getOrders = (req, res, next) => {
  Order.find({ 'user.userId': req.user._id })
    .then(orders => {
      res.render('shop/orders', {
        path: '/orders',
        pageTitle: 'Your Orders',
        orders: orders
      });
    })
    .catch(err => console.log(err));
};
```

Ez már a rendelés véglegesítése erről a részről azt kell tudni, hogy a user aki a kosár tartalmát elküldi a rendelésre annak az adatbázis az orders táblájában a következő attribútumokat fogja elmenteni: mennyiséget és a termék nevét ezek után létrehozza arra a user és email alapján a táblát és a kosár tartalmát pedig kifogja törölni és elfogja menteni. A rendelésünk pedig az orders menüben fog látszódni. Ez is tesztelve lett, hogy a felhasználók ne lássák egymás megrendelését csakis mindig a sajátjukat lássák így elkerülve számos hibát illetve hibához vezető dolgokat.

5.3. GET, POST kérések

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. Ellentétben, 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, tetszőleges típusú adat küldhető a szerverre a kérési üzenet törzsében. A POST-kérés fejlécmezője általában az üzenettörzs internetes médiatípusát jelzi. A világháló és a HTTP számos kérési módszeren vagy „igén” alapul, beleértve a POST-ot és a GET-et, valamint a PUT-t, a DELETE-t és még sok mást. A webböngészők általában csak a GET-et és a POST-ot használják, de a RESTful online alkalmazások sok mást is.

A POST helye a HTTP metódusok tartományában az, hogy egy új adatentitás reprezentációját küldje el a szervernek, így az az URI által azonosított erőforrás új alárendeltjeként kerül tárolásra. A legtöbb webböngésző természetes korlátai csak a GET vagy POST használatára vonatkoznak, a tervezők szükségét érezték, hogy a POST-ot sok más adatküldési és adatkezelési feladat elvégzésére használják fel, beleértve a meglévő rekordok megváltoztatását és törlését. Számos űrlapot használnak arra, hogy pontosabban meghatározzák az információk kiszolgálóról történő lekérését, anélkül, hogy a fő adatbázist megváltoztatnák. A keresési űrlapok például ideálisak a `method="get"` megadására. Ez nem jelenti azt, hogy minden webes űrlapnak meg kell adnia a `method="post"` elemet a nyitó címkéjében. Amikor egy webböngésző POST-kérést küld egy webes űrlapelemtől, az alapértelmezett internetes médiatípus az „application/x-www-form-urlencoded”. Ez a formátum kulcs-érték párok kódolására, esetlegesen ismétlődő kulcsokkal. Minden kulcs-érték pár egy „&” karakterrel van elválasztva, és minden kulcsot egy „=” karakter választ el az értékétől. A kulcsokat és az értékeket a szóközők „+” karakterrel való helyettesítésével, majd az összes többi nem alfanumerikus karakteren százalékos kódolással kell megszabadítani

5.4. Fejlesztői konferencia

Először is azt kellett a csoporttal megtárgyalni, hogy kinek mi a feladata és mihez ért a legjobban. A csapat különböző tagjainak az erőssége: Kiss Ádám az adatbázis felállítása illetve a weboldallal való összeköttetésért felelt amelyet maximális tudással és odafigyeléssel hajtott végre. Filip Dávid a weboldal felépítésért felelt amelyben belesegítettem a különböző tesztek segítségével illetve az azonnali dizájn elemek miatt is. Hóbor Martin pedig a telefonos applikációért felelt ha esetleg végeztünk akkor egymásnak segítettünk. Úgy álltunk hozzá, hogy januárig az alapvető dolgokat megtudtuk tanulni és ezt felhasználni a hónapok alatti gyakorlás során és szikronban dolgoztunk ha esetleg lett volna valami fennakadás akkor igazából megoldottuk többen. A kommunikáció a következő módon történt Gmail-en egyeztettünk az adott fájlokkal majd ha ez megvolt akkor a Facebook segítségével tudtunk egymással időpontot egyeztetni és a heti 2-3 óra programozás mellett 2 hetente összeültünk, hogy legyen mit összevessünk illetve a haladási kis mérföldköveinket is összetudtuk vetni. 1 hónap után már felépítettük a weboldal azt az alap részét amelyből lehet majd táplálkozni a későbbiekben utána haladtunk a mobil applikációval amelyben az admin részét szikronba csináltuk a webes résszel. amikor már 2 vagy 3 hónap telt el már körvonalazódott, hogy még mi megvalósítható.

5.5. Tesztelés

Tesztelésnek különböző fajtái vannak az egyik része amellyel a weboldalt lehet tesztelni ez nem más mint a Selenium a másik pedig a mobil applikációnak a tesztelése amely a beépített Xamarinos telefon illetve még a sima Androidos telefon is lehet. A Seleniumra jellemző, hogy egy teljes integrált fejlesztői környezet (IDE) a Selenium tesztekhez. Firefox-bővítményként és Chrome-bővítményként valósul meg. Lehetővé teszi a funkcionális tesztek rögzítését, szerkesztését és hibakeresését. A szkriptek automatikusan rögzíthetők és manuálisan szerkeszthetők az automatikus kiegészítés támogatásával és a parancsok gyors mozgatásának lehetőségével. A szelenes nyelvű tesztírás alternatívájaként a tesztek különféle programozási nyelveken is írhatók. Ezek a tesztek ezután kommunikálnak a Seleniummal a Selenium Client API metódusainak meghívásával. A Selenium jelenleg Java, C#, Ruby, JavaScript, R és Python kliens API-kat biztosít.

Először is a regisztráció részét teszteltük Seleniummal és a Login részét a weboldalnak amelyet a következő képen lehet látni.

Project: weboldal

Tests +

Search tests

http://localhost:3000

Alapvető funkciók	Command	Target	Value
1	open	/	
2	set window size	945x1020	
3	click	linkText=Signup	
4	click	id=email	test@test.com
5	type	id=password	admin
6	type	id=confirmPassword	admin
7	sendKeys	id=confirmPassword	ENTER
8	click	id=email	test@test.com
9	type	id=password	admin
10	type	id=password	ENTER
11	sendKeys	id=password	ENTER
12			

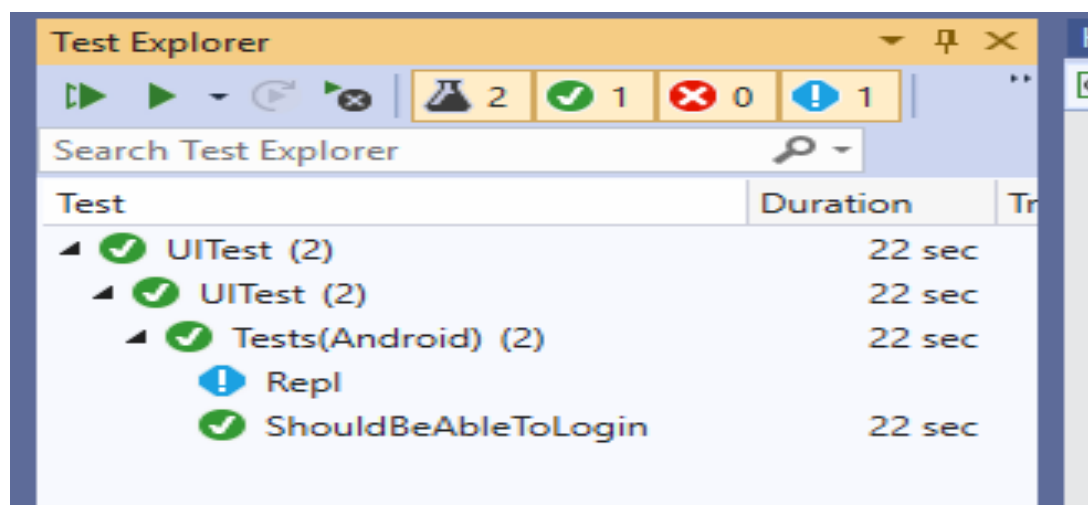
A következő képen pedig az elfelejtett jelszó funkció látható amelyet pedig először az email kiküldése előtt teszteltük a rendszerben, hogy esetleg van ott-e található email vagy sem, ha van akkor igazából kiküldte az emailt és a következő oldalon le is frissítettük majd bejelentkeztünk.

Alapvető funkciók	Command	Target	Value
Elfelejtett jelszó*	1	open	/
	2	set window size	945x1020
	3	click	linkText=Login
	4	click	linkText=Forgot Password?
	5	click	id=email
	6	type	id=email
	7	click	name=confirm

Ezen a képen láthatjuk a végső lépést amely az új jelszóval ellátott fiókba jelentkeztünk és ezáltal már megvan az új jelszavunk.

Alapvető funkciók	Command	Target	Value
Elfelejtett jelszó*	1	open	/reset/153b2d1d07b1c1eeb0c1b0c950b363a14c852c77562dc28673f1ee3d7
Elfelejtett jelszó 2*	2	set window size	945x1020
	3	click	id=password
	4	type	id=password
	5	click	css=btn
	6	click	css=card-body
	7	click	id=email
	8	type	id=email
	9	click	id=password
	10	type	id=password
	11	sendKeys	id=password

A mobilos applikáció tesztelése Xamarinban a következő módon lehet elérni. Ehhez nem mást mint egy tesztet kell írni ehhez ezt nevezzük UITestnek amelyet a Xamarin.Formsnak nevezzük. Az UITest a Xamarin.Forms-szal használható felhasználói felület tesztek írásához, amelyek a felhőben futhatnak több száz eszközön. A fejlesztők automatizált felhasználói felületteszteket írhatnak az iOS és Android alkalmazásokhoz. Kisebb módosításokkal a Xamarin.Forms alkalmazásokat a Xamarin.UITest segítségével lehet tesztelni, beleértve ugyanazon tesztкод megosztását. Az UITest automatizálja a felhasználói felületet azáltal, hogy aktiválja a vezérlőket a képernyőn, és mindenhol megadja a bemenetet, ahol a felhasználó általában kapcsolatba lép az alkalmazással. Az olyan tesztek engedélyezéséhez, amelyek megnyomhatnak egy gombot vagy szöveget írhatnak be egy mezőbe, a tesztкодnak módot kell adnia a vezérlőelemek azonosítására a képernyőn. Ha engedélyezni szeretné, hogy az UITest kód hivatkozzon a vezérlőkre, minden vezérlőnek egyedi azonosítóra van szüksége. A Xamarin.Forms alkalmazásban az azonosító beállításának javasolt módja az AutomationId tulajdonság használata. Következően pedig egy Xamarinos tesztet fogunk látni amelynek a lényege, hogy igazából rákeresünk a weboldalra itt jelen esetben <http://localhost:3000> itt pedig a következő tesztet hajtjuk vége amelyet a rendelésben megkapott kód segítségével vagy esetleg a felhasználói bejelentkezésünkkel tudunk megtenni bejelentkezünk és itt is feltudunk venni terméket ez egy admin felületként szolgál amely egy kis kicsit könnyebben elérhető más eszközről is nem csak a telefonnak a böngészőjéből.



6. Összefoglalás

A szakdolgozatunk célja, hogy felnyissuk másik szemét, hogy a kreativitáshoz és a piacé réshez csak olyan gondolkodás menet kell amely a jelenlegi világnak a fontos és jövedelmező részét amely a turizmusban és a mostani kriptopiacokban van ezeket kell felhasználni arra, hogy jövedelmet illetve úgymond „hírnevet” kapjon az ember ezért is alkottunk meg ez a projectet illetve szakdolgozatot.

A másik része esetleg a leendő munkahelyen ezt feltudjuk mutatni ilyen kis beugró specifikációnk amelyet előnyünkre is tudjuk fordítani vagy esetleg üzleti megkereséssel céljából bármelyik vállalkozásban feltudjuk mutatni, hogy ezt csináltuk weboldal és mobil applikáció összességével.

A harmadik lehetőség pedig igazából, hogy külföldre el lehet adni ha esetleg lenne rá igény mert ha ilyet jelenleg Magyarországon elsőnek hozunk létre akkor a külföldi publikum is felfog figyelni rá és a mai világban mindenki minden programban az üzletet keresi. Ennek elősegítése érdekében itt hirdetjük és hátha eljut a hírünk oda is.



6.1. Jövőbeli tervek

A legnagyobb lehetőség ebben van igazából, hogy a belföld és külföldiek körében igazából honosítható is bármely részben mert még nem nagyon láttunk olyan oldalt amely hasonló dolgokkal foglalkozna mint egy ország főbb turizmusának a hirdetése és összegzése, jövőben még a tervek közé tartozik, hogy útválasztás lesz elérhető benne amely tud ilyen „túrát” csinálni ahol maga a felhasználónak lehet megtervezni, hogy melyik borászat vagy pincészetbe megy és ezáltal egy picit személyre szabhatja az utazását az íz világában. Még esetleg hozzá tartozik, hogy egy ügyfélszolgálat gomb is elérhető lesz, hogy esetleg mindig a legfrissebb tartalmak legyenek elérhetők és észrevétel és ötlet láda is lehet amely az oldal fejlődését illetve a magasra emelkedését fogja elősegíteni. Mobilos applikációban még lehet hozzátenni, hogy a felhasználó által megvett termékek ellenőrzése és igazából a nagyobb adminisztrációs ellenőrző felület kialakítása amely igazából lehet egy külön applikáció amely szétválasztja a felhasználó és az admin részleget.

New User

← Back Reset **Save User**

USER INFORMATION

User Info

User Role

Account Information

User Name *

First Name *

Last Name *

Email *

Password *

Password Confirmation *

Interface Locale

This account is

7. Felhasznált irodalmak

- <https://docs.microsoft.com/hu-hu/azure/developer/javascript/>
- <https://docs.microsoft.com/en-us/cpp/cross-platform/?view=msvc-170>
- <https://www.javascript.com/>
- <https://www.w3schools.com/nodejs/>
- <https://nodejs.org/api/errors.html>
- <https://docs.microsoft.com/en-us/appcenter/errors/>