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) neve[1]: Scheuer Patrik, Ábrahám Domonkos

Képzés: nappali / felnőttoktatás - esti munkarend

Szak: 5 0613 12 03 Szoftverfejlesztő és tesztelő technikus

A záródolgozat címe:

„SPACEY” webalkalmazás

Konzulens: Horváth Norbert
Beadási határidő: 2022. 04. 29.

Győr, 2022.04.29

Módos Gábor
igazgató

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

9021 Győr, Szent István út 7.
+36 (96) 529-480
+36 (96) 529-448

OM: 203037/003
jedlik@jedlik.eu
www.jedlik.eu

Konzultációs lap[2]

	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 29.

Scheuer Patrik

Ábrahám Domonkos

[2] Szakmajegyzékes, csoportos konzultációs lap

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

Jedlik Ányos Gépipari és Informatika
Technikum és Kollégium

SPACEY

Készítette: Scheuer Patrik, Ábrahám Domonkos

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 6
3.2. Webalkalmazás felépítése 7
3.3. Fejlesztői környezet 8
4. Felhasználói dokumentáció 11
4.1. Telepítés 11
4.2. Fő funkciói 11
4.3. Vásárlás jellemzői 12
5. Fejlesztői dokumentáció 15
5.1. Kezdet 15
5.2. Menük 16
5.3. Belső funkciók felépítése 16
5.4. GET, POST kérések 19
5.5. Tesztelés 22
6. Összefoglalás 32
6.1. Jövőbeli tervek 32
7. Felhasznált irodalmak 33

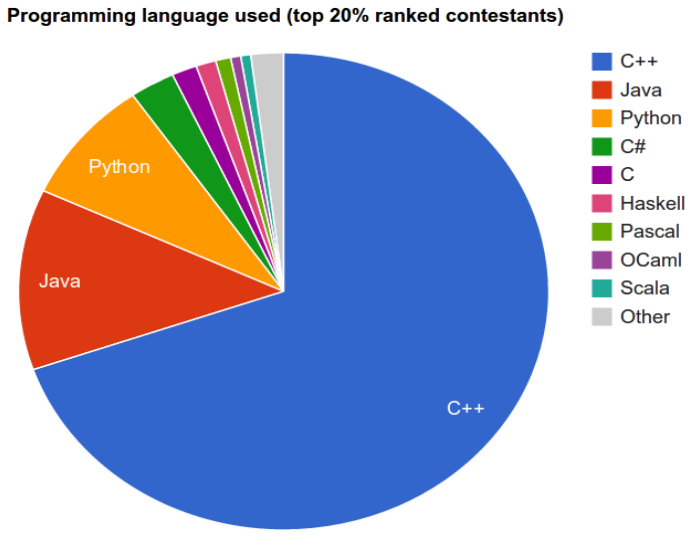
1. Bevezetés

Az elmúlt években az űrhajózás annyira közel került ahhoz, hogy civilek is részt vehesenek űrutazásban, hogy mára már közel sem elkpézelhetetlen az eféle űrutazás. Napjaink meghatározó űrkutatási cége a SpaceX áll legközelebb annak megvalósításához, hogy akkár nagyobb mennyiségben civileket küldjön az űrbe. Ennek apropóján jött az ötlet a szakdolgozat témájaként, hogy egy olyan komplex webes alkalmazást készítsünk amely különböző űrutazásokat kínál civileknek.

Az űrutazásra jelentkezők sok információt kaphatnak az adott utazással kapcsolatban. Különböző ismertetőkön keresztül az utazó megismerheti magát az űrhajót és az adott bolygót részletesen. Az utazással kapcsolatos alap információkat is megtalálhatja az oldalon. A mobil applikációban jegyzeteket rögzíthetünk az út során.

2. Specifikáció

A Szakdolgozatunk egy űrutazással foglalkozó webes alkalmazás. A fejlesztés első lépéseként meg kellett határoznunk, hogy mit tudjon a programunk, miben végezük a backend, frontend, az adatbázis fejlesztését. Mielőtt kiválasztottuk az adott fejlesztői környezeteket, az előbb említetekre lebontva felmértük, hogy az ötleteinket, elképzeléseinket melyikekben tudnánk tudásunk szerint a lehető legjobb módon megvalósítani. A mobil applikáció fejlesztői környezete már eleve adott volt számunkra. A választások a következő program nyelvekre estek: Backend: NodeJs Frontend: VueJs Adatbázis: MongoDb Mobil applikáció: Xamarin.

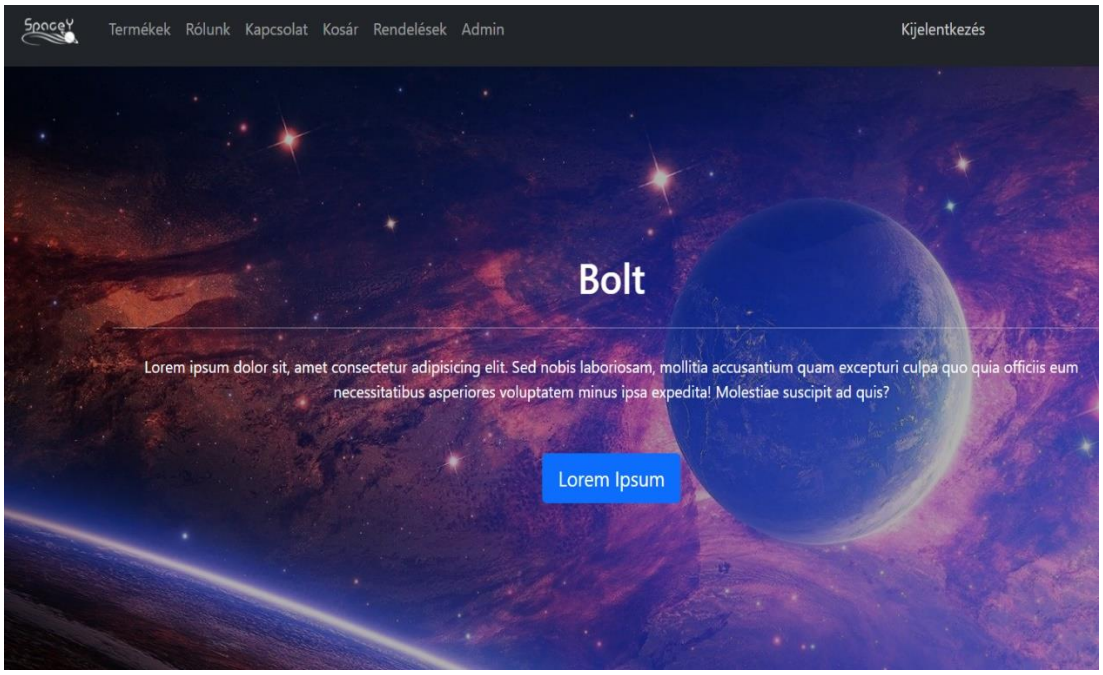


2.1. Szoftver alapvető jellemzői

- Szoftver elnevezése, címe: SPACEY
- Szoftver kategóriája: Webes alkalmazás
- Szoftver alapvető funkciója: Űrutazások vásárlása, kezelése, ismertetése
- Fejlesztői környezet: Backend: NodeJs Frontend: VueJs Adatbázis: MongoDb Mobil applikáció: Xamarin
- Programozási nyelvek: C#, JavaScript
- Platform: Windows

2.2. Szoftver funkciói

A dolgozatomban szeretnék megvalósítani egy űrutazással foglalkozó honlapot, amely lehetőséget add ,hogy magánszemélyként űrutazásokban vehesen részt bárki. A megadott információk alapján a honlapon magunk választhatjuk ki a számunkra legmegfelelőbb utat. Az oldalra navigálva rögtön az éppen meghirdetet utakat láthatjuk. A termékek menüpont alatt az adott utakról találhatunk részletes információkat. A rólunk menüpontban pedig magáról az oldalról és a cégről láthatóak információk.



3. Alkalmazott technológiák

NodeJs:

A Ryan Dahl által 2009-ben létrehozott, Windowson, Linuxon és Mac OS-en is futó, nyelvként JavaScriptet használó Node.js egy olyan, ingyenes szerverkörnyezet, amely a Google V8-as motorjára épül. Platformfüggetlen megoldás, amely tökéletes skálázható online alkalmazások megvalósítására.

Lehetővé teszi a gyors és hatékony JavaScript fejlesztést és a könnyű prototypingot ezzel a leghatékonyabban használható szerveroldali programnyelvek egyike. Egyre népszerűbb, számos nagy cég (Linkedin, Microsoft, Yahoo!, Walmart, PayPal, Netflix…stb.) használja.

VueJs:

A Vue.js egy JavaScript könyvtár. Sok esetben keretrendszernek szokták hívni, pedig valójában nem az: az Angular, React, Vue trióból egyedül az Angular minősül frameworknek. 2014-ben jelent meg először, a Google egyik volt fejlesztője készítette eredetileg. Azóta hihetetlenül sok újításon ment keresztül, és a használati tendenciája egyre inkább felfelé ívelőnek látszik.

MongoDb:

A MongoDB egy nyílt forrású adatbáziskezelő rendszer (DBMS), amely dokumentum-orientált adatbázis-modellt használ. A MongoDB C ++ nyelven íródott. A MongoDB támogatja az adatok különféle formáit. A MongoDB az adatokat sima fájlokban tárolja, saját bináris tárolóobjektumaik segítségével. Ez azt jelenti, hogy az adattárolás nagyon kompakt és hatékony, kiválóan alkalmas nagy adatmennyiségek tárolására. A MongoDB az adatokat JSON-szerű dokumentumokban tárolja, ami nagyon rugalmasvá és méretezhetővé teszi az adatbázist.

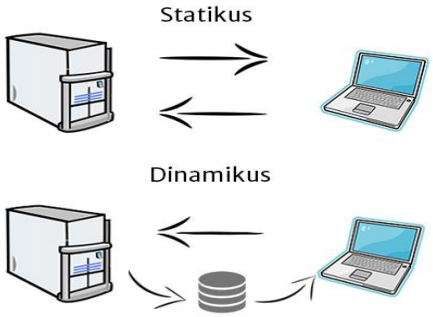
A MongoDB dokumentum-orientált adatbázis-modell. Minden MongoDB adatbázis gyűjteményeket tartalmaz, amelyek viszont dokumentumokat tartalmaznak. Az egyes dokumentumok különbözhetnek, és a különböző mezők számától függhetnek. Az egyes dokumentumok mérete és tartalma különbözik egymástól. Az adatmodell funkciói lehetővé teszik tömbök és komplexek hierarchikus kapcsolatban strukturált tárolását.

Xamarin:

A Xamarin egy nyílt forráskódú platform, amellyel a C# vagy F# használatával számos különböző eszközre hozhat létre mobilalkalmazásokat.

3.1. Felépítése

A honlapok lehetnek statikusak és dinamikusak. A statikus weblapok tartalma állandó, nem változik (például bemutatkozó oldalak) csak manuális átszerkesztéssel. A dinamikus weblapok alapeleme az ún. motor, amelynek segítségével ismételten végezhetőek azonos műveletek egyszerűen (például hírportálok) és nem kell a forráskódot manuálisan szerkeszteni, mert azt a honlapmotor állítja elő sablonszerűen. A statikus weblapok létrehozására alkalmas a HTML és JavaScript és minden kliens oldalon futó webes nyelv. A dinamikus weblapokhoz szükség van egy-egy olyan leíró nyelvre (például PHP, Java), amely szerveroldali és a szerveren képes a felvitt információkat/adatokat rögzíteni akár szöveges (fizikai fájlok), akár adatbázisba betárolás formájában (például mySQL, postgreSQL).



A legtipikusabb ilyen nyelvek:

- kliens-oldali, a böngészőben végrehajtott szkriptek:
    - JavaScript
- szerver-oldali szkriptek (a böngésző statikus HTML/XHTML lapot és esetleg kliens-oldali szkripteket kap):
    - PHP
    - ASP
    - JSP
    - Perl

A szkripteken túl lehetnek egy lapba ágyazva további speciális objektumok is, például Flash (SWF) vagy ún. Java kisalkalmazások.

3.2. Webalkalmazás felépítése

A programozási nyelvek fejlõdését figyelemmel kisérõ programozók fülében ismerõsen cseng a Java szó. A Java napjaink talán legígéretesebb programozási nyelve, melyet a Sun Microsystems és a Netscape Communications Corporation együtt fejlesztettek ki. A Java nyelv nem más, mint egy hatalmas, tisztán objektum-orientált programozási nyelv, melynek nagy ereje, hogy segítségével platformfüggetlen alkalmazások készíthetõk. A Java nyelven megírt programokat egy ún. köztes kódra (byte kód) kell lefordítani, melyek különbözõ platformokon, különbözõ operációs rendszerek alatt futtathatóak - egy értelmezõ program segítségével. A Java szóval napjainkban - a WWW világában - három területen találkozunk.

- Egyrészt beszélünk Java programokról, mely programokat le kell fordítani és önállóan futnak. (Nem támaszkodnak más programokra.)
- A JavaScript a Java nyelv "kis testvérkéje"- mondják sokan. A JavaScript-ek a HTML lap szövegébe épülnek be, csak a böngészõben képesek futni.
- A Java appletek (programocskák) - kis Java programok -, melyek a Java-t támogató böngészõkbõl futtathatóak. Ezek tehát felhasználják a böngészõk által nyújtott szolgáltatásokat. Az appleteket is le kell fordítani és - ahogyan azt már korábban bemutattam - ezek HTML dokumentumokból hívhatók.

A JavaScript, mint neve is mutatja, egy script nyelv, amit legelõször a Netscape Navigator 2.0 támogatott. A JavaScript-eket HTML lapokba lehet beágyazni, a lappal együtt töltõdnek le, majd a böngészõ értelmezi és futtatja azokat. Hasonlít a Java-ra, de ugyanakkor sokkal kisebb, egyszerûbb, korlátozottabb mûködésû is annál.

Nézzünk meg egy összehasonlító táblázatot arra vonatkozóan, hogy mik a fõbb hasonlóságok és különbségek a JavaScript és a Java appletek között:

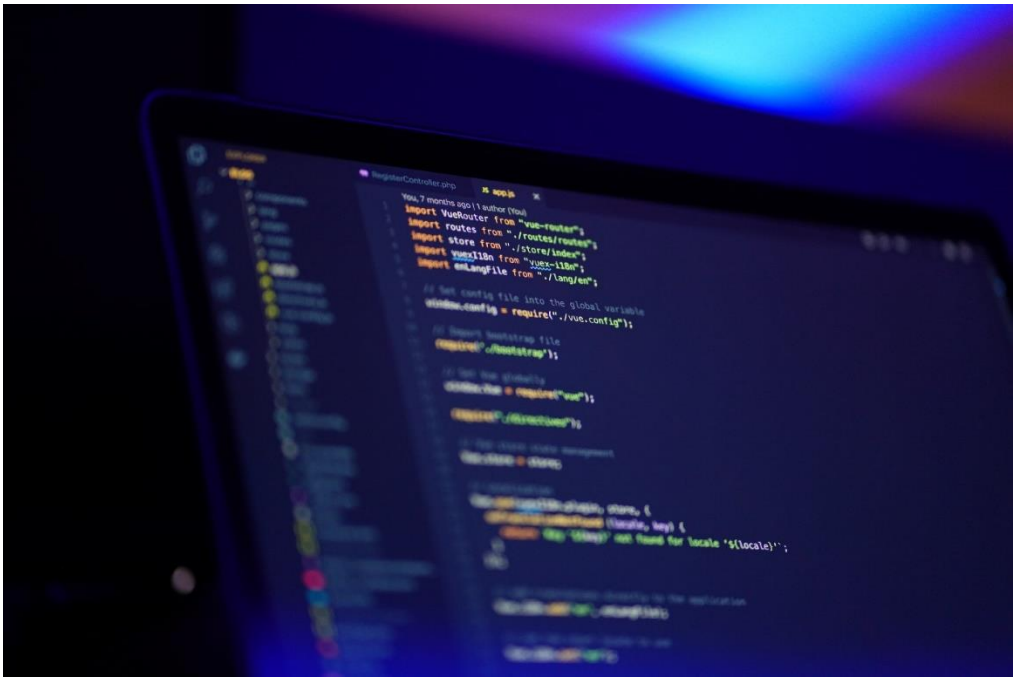
JavaScript	Java applet
A kliens értelmezi (interpretálja).	A szerveren fordított - letöltés után - kliensen futó.
Objektum-alapú. Beépített, kiterjeszthetõ objektumokat használ, de nincsennek osztályok és öröklõdés.	Objektum-orientált. Az appletekben vannak osztályok és öröklõdés.

A program beágyazódik a HTML dokumentumba.	Az appletek a HTML lapból hívhatók meg, de azoktól elkülönülnek.
A változók típusát nem kell deklarálni. (Nem típusos nyelv.)	A változókat deklarálni kell. (Erõsen típusos nyelv.)
Dinamikus kötés. Az objektum-hivatkozások ellenõrzése futásidõben történik.	Statikus kötés. Az objektum-hivatkozásoknak már fordításkor létezniük kell.
Nem tud önállóan a merevlemezre írni.	Nem tud önállóan a merevlemezre írni.

A JavaScript-et azért fejlesztették ki, hogy az egyébként statikus lapokba dinamizmust vigyenek. Egyrészt látványosabbá tehetjük vele azokat (dinamikusan, futásidõben generálhatunk HTML lapokat), másrészt számos praktikus dologra használhatjuk fel (pl. jelszót kérhetünk a lapunk megtekintéséhez, vagy pl. ellenõrizhetjük az elküldendõ adatok helyességét még a kliens oldalon - hálózati erõforrásokat spórolva meg, stb).

3.3. Fejlesztői környezet

Bármely program / szoftver, amelyet látunk vagy használunk, a háttérben futó kóddal működik. Hagyományosan a kódolást a hagyományos szerkesztőknél, vagy akár az alapvető szerkesztőknél, mint például a Jegyzettömb használják! Ezek a szerkesztők alapvető támogatást nyújtottak a kódolók számára. Néhányuk annyira alapvető volt, hogy nagyon nehéz volt benne írni az alapvető angol szintű programokat. Az idő múlásával néhány programozási nyelvnek speciális keretre és támogatásra volt szüksége a további kódoláshoz és fejlesztéshez, ami ezen szerkesztőkkel nem volt lehetséges. A VI szerkesztő, a Sublime szövegszerkesztő és a Visual Studio kód a sokféle szerkesztő közül, amelyek léteznek. A legszembetűnőbb, és szinte minden kódoló nyelvet támogat, a VISUAL STUDIO CODE. A Visual Studio Code szolgáltatásai lehetővé teszik a felhasználó számára, hogy a felhasználásnak megfelelően módosítsa a szerkesztőt, azaz a felhasználó letöltheti a könyvtárakat az internetről, és integrálhatja a kódhoz az igényei szerint.



A Visual Studio Code egy laikus szempontból kódszerkesztő. Meghatározásaként a Visual Studio Code „ingyenes szerkesztő, amely segíti a programozót a kódírásban, segít a hibakeresésben és a kód javítását az intelli-sense módszerrel”. Normál esetben megkönnyíti a felhasználót, hogy egyszerűen írja a kódot. Sokan azt mondják, hogy az IDE fele és szerkesztője, de a döntés a kódolóktól függ.

A Visual Studio kódot támogató leggyakoribb nyelvek:

- C #
- Visual Basic
- Java-Script
- R
- XML
- Piton
- CSS
- MEGY
- PERL

Egy másik szolgáltatás, amelyet a naiv felhasználók vagy bárki azonnal láthat, és különbözik a többi szerkesztőtől, a Visual Studio kód felhasználóbarát jellege. A

Visual Studio Code használhatóságát nagyon könnyű kezelni. A fájl hierarchikusan van elrendezve, és rendszeres szoftverekkel rendelkezik, mint például eszköztár, állapotsor és oldalsáv. Ezenkívül van egy lebegő Windows explorer ablaka is, amelyet a kényelem alapján egy helyen rögzíthetünk, amely a fájlok könyvtárstruktúrájából áll. Ezek a fájlok (kódfájlok, képmappák stb.) Innen nyithatók vagy átnevezhetők, és a változások automatikusan tükröződnek a tárolóban.

3. Felhasználói dokumentáció

3.1. Telepítés

A felhasználói dokumentáció úgy készült, hogy minden olyan információt tartalmazzon, amire szüksége van a felhasználónak a rendszer használatához. Ezért a felhasználóknak azt tanácsoljuk, hogy a szoftver használata előtt tekintsék át ezt a dokumentumot és jobban megismerkedni vele.

Mivel böngészőben futó alkalmazásról szól a szakdolgozatom így minimális hardverigénye van csupán egy internet böngészőre van szükségünk, és egy internet kapcsolatra.

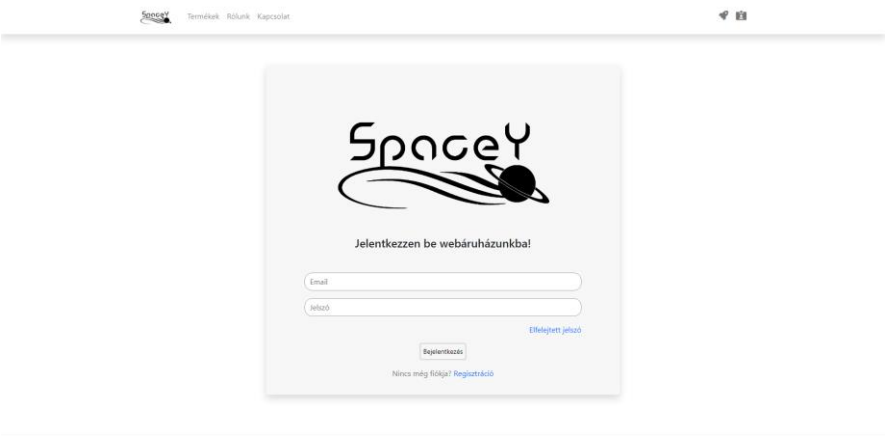
- Javasolt böngésző: Google Chrome

- Ajánlott felbontás: 1024 * 786

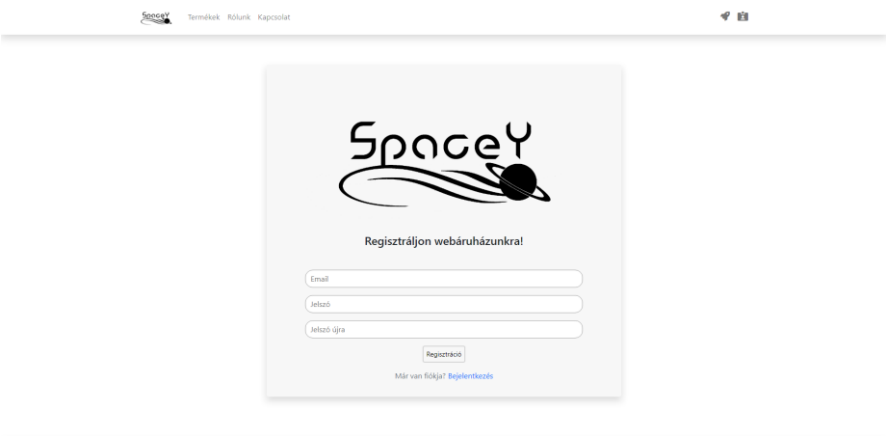
Mivel ennek a weboldalnak több szekciója van, ezért először a Visual Studio Code részt említjük meg, mivel ez egy package.json fájlt tartalmaz, így az npm start parancs elindítja a weboldalt a háttérben futó szerverünkön. A másik része a mobilalkalmazás, ahol egy Visual Studio-ra végződő fájl segítségével nyithatjuk meg, majd CTRL+F5-tel futtathatjuk, és a meglévő utasításokat követve termékeket adhatunk hozzá, ez egy admin felület, ahol szükség esetén meglévő termékeket tudunk építeni és követni, ha az adatbázis a számítógépünkön nem érhető el.

3.2. Fő funkciói

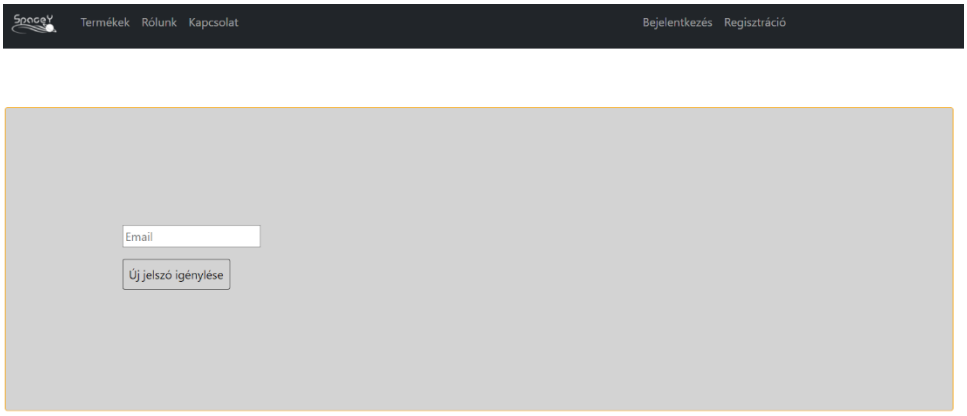
A weboldal szekciót az jellemzi, hogy a regisztráció fül és a főoldal a menüsorban csak bejelentkezésig érhető el, megakadályozva ezzel, hogy bejelentkezés nélkül adjunk le rendelést, ami az alábbiak szerint látható:



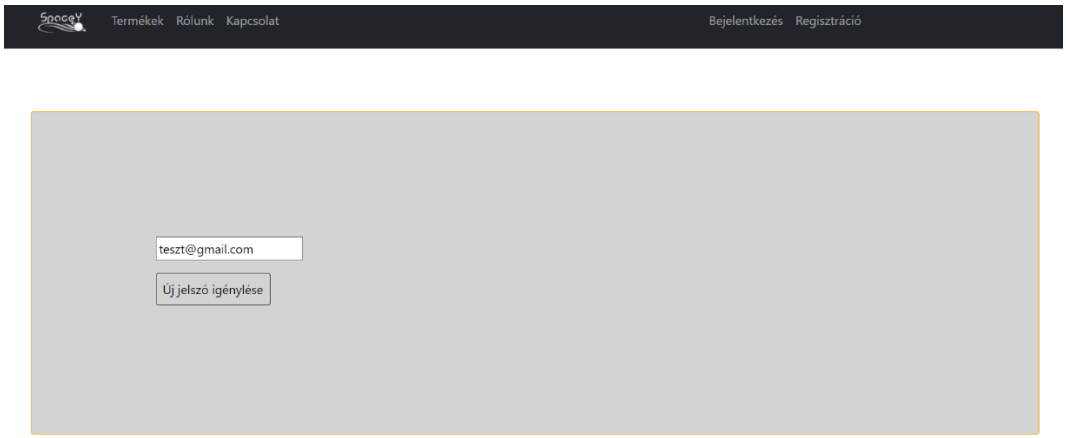
A következő funkció a regisztráció, amely a következőképpen működik: meg kell adnia egy érvényes e-mail címet és két jelszót, hogy bizonyítsa, hogy nem robot, majd a bejelentkezési panelen keresztül bejelentkezhet; ha az e-mail cím és a jelszó párosítása nem működik, hibaüzenetet kap, amelyet kijavíthat:



A jelszó-visszaállítás az utolsó funkció a webhelyen, amelyre akkor van szükség, ha a jelszó és az e-mail bejelentkezés sikertelen:

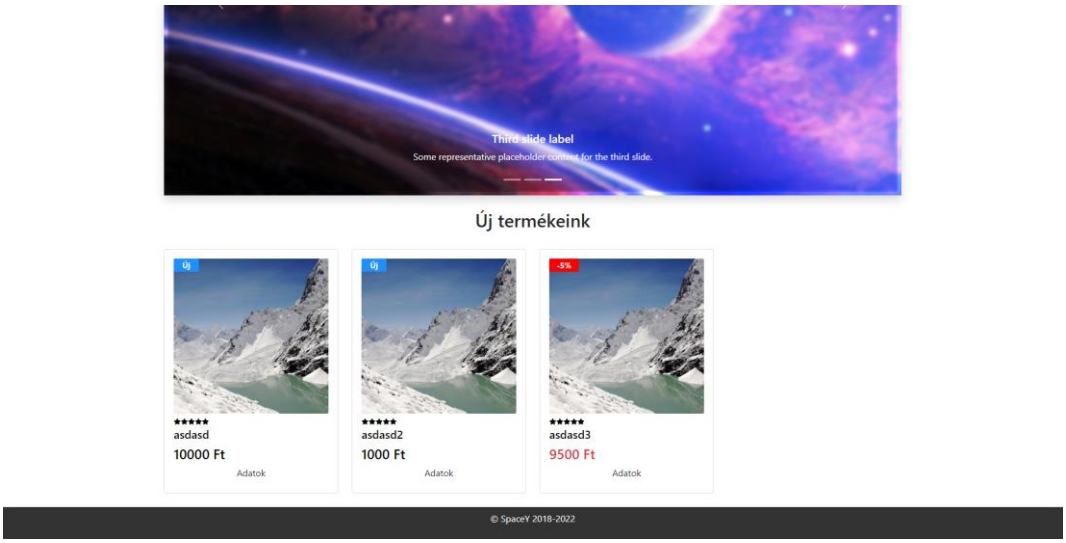


Ez úgy működik, hogy kapsz egy e-mailt egy linkkel, amely egy oldalra vezet, ahol megváltoztathatod a jelszavadat az új jelszó megadásával, amely a jelszavad lesz:

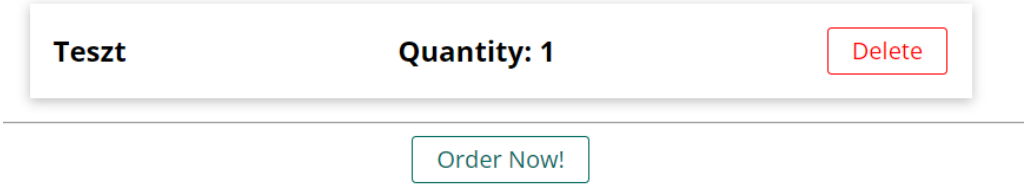


3.3. Vásárlás jellemzői

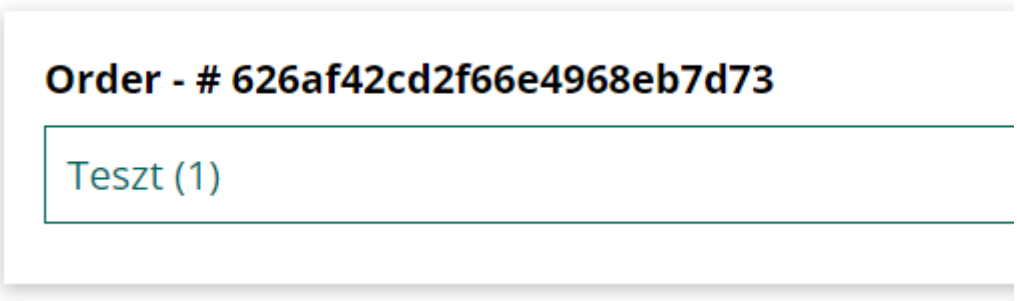
Az első az a funkció, amely a termékek fülre visz minket, ahol különböző utazások közül választhatunk, és megtekinthetjük mindegyiknek a nevét, leírását és árát. A terméken belül két funkció közül választhatunk: információ és kosárba helyezés:



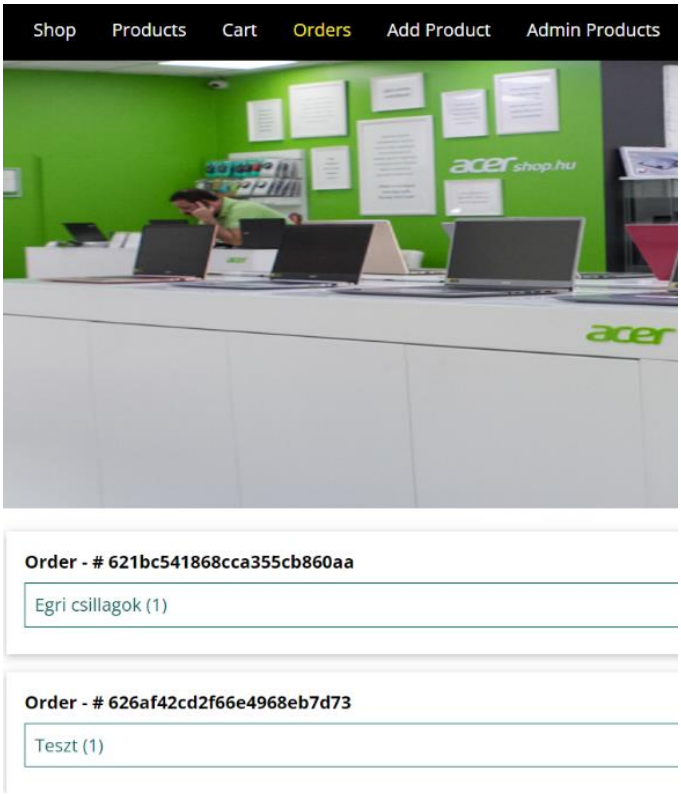
Miután hozzáadta a terméket a vásárláshoz, a következő funkciók jelennek meg a kosár fülén. Látni fogja a kosárba helyezett termék nevét, valamint a kosárba helyezett tételek számát, és törölheti, ha nem megfelelő termék a kosárban, de ha igen, akkor megjelenik és az lesz hozzáadva a megrendeléshez:



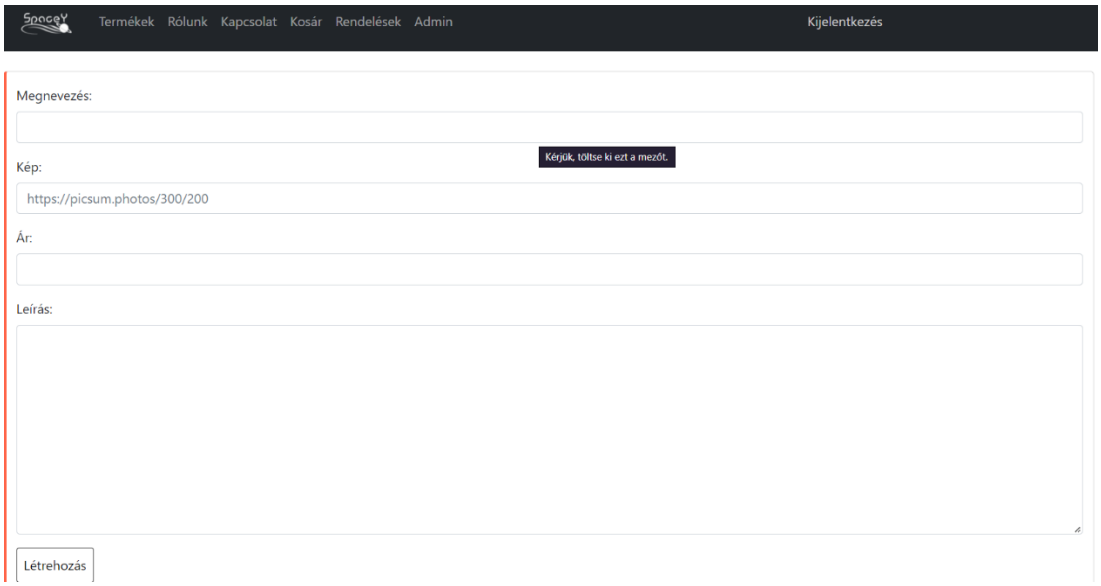
Az alábbiakat a megrendelés lapon találja: Minden felhasználóhoz egyedi számot rendelünk, így biztosítva az adatbázis-problémák elkerülését. A következő kép a termék nevét vagy mennyiségét ábrázolja:



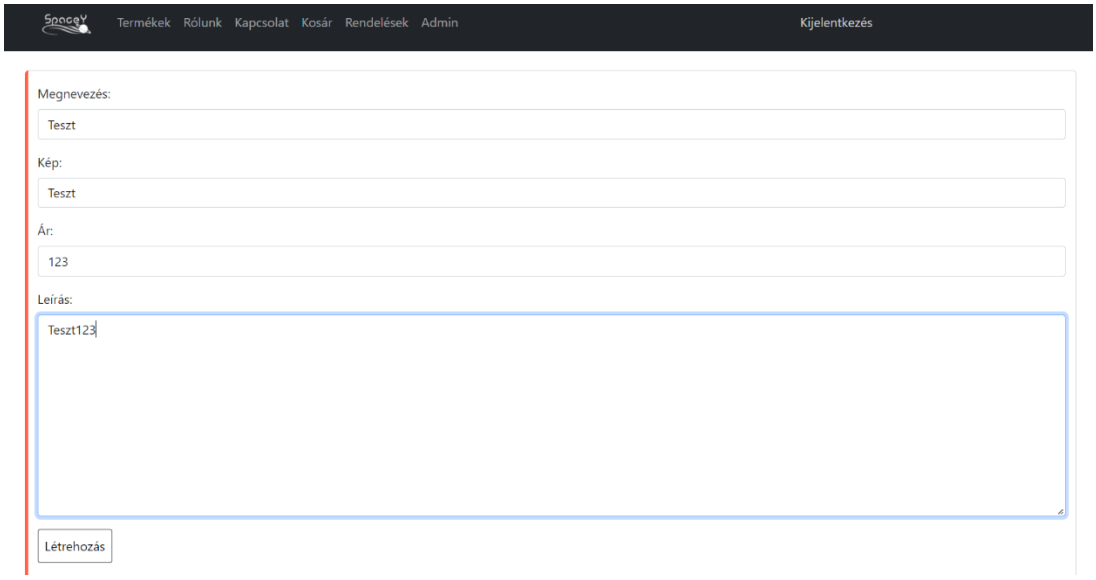
A felület vásárlási része mostanra elkészült, és a következő funkciók, amelyek közé tartozik az új termék hozzáadása, valamint a már hozzáadott termékek szerkesztése és törlése, csak olyan felhasználók számára érhető el, akik nem rendelkeznek normál felhasználói jogokkal, hanem az adatbázisban adminisztrátori jogokat kaptak. Az adatbázis alaptermékei nem törölhetők visszaélés miatt; hasonlóképpen, ha egy felhasználó admin jogokat kap, és "rossz akaratú", akkor csak az általa létrehozott adatokat törölheti Egyetlen admin sem képes törölni egy másik admin termékét, ami korlátozza a funkcionalitást. A képen először az adminisztrátori jogokkal nem rendelkező admin, majd az adminisztrátori jogokkal rendelkező admin:



Az utolsó két menüpont, amelyek gyakorlatilag az egész weboldalon a legjelentősebbek, a termék hozzáadása és a termék szerkesztése opciók, amelyek lehetővé teszik az admin felhasználó számára, hogy a termék nevét, árát és leírását, valamint egy kép url-jét, amely a fényképed lesz, megadva termékeket adjon hozzá a meglévő termékekhez:



Az utolsó menüpont az admin által hozzáadott termékek, ahol törölheti vagy módosíthatja a terméket az adatbázisban. Nagyon fontos, hogy ez az admin jogosultság csak az adatbázisban legyen elérhető, különben minden felhasználó admin jogokkal rendelkezik és új árukat hozhat létre.



Az is kritikus, hogy minden apró módosítást ellenőrizzen a terminál naplórendszerének segítségével, így pontosan láthatja, hogy mit változtatott meg, és ha valami rosszul megy, a terminál képes lesz mindent ellenőrizni és segíteni önnek.

5. Fejlesztői dokumentáció

A szakdolgozat készítése során számos olyan tényezőt kellett figyelembe vennünk, amelyek fontosak voltak a szinkronban történő munkavégzéshez. Miután egyikünk befejezte az adatbázis és a weboldal összekapcsolását, a meglévő munkában dolgozhattunk azokon a tervezési elemeken, amelyek ezt az egyszerűsített nyers weboldalt elfogadhatóvá teszik, és amelyeknek az eredménye látható lesz, hogyan lehet ezt az alapkövetelményt egy kis okossággal megvalósítani. Ha ez sikerül, akkor a dokumentációt az alapoktól kezdve a projektnek megfelelő formában hozta létre. A projekt első lépése az volt, hogy a csomagban található Visual Studio Code segítségével megteremtette az alap weblap alapjait. A json fájlban csak a weboldal létrehozásához szükséges fájlok, illetve potenciálisan az adatbázis eléréséhez szükséges fájlok kerültek bele, a többit eltávolítottuk. A Visual Studio-ban tervezett mobilalkalmazás a második nagy szinkronfolyamat. Ez a weboldalnak az a része, amely lehetővé teszi a felhasználók számára, hogy a telefonjukon hozzáférjenek az adminisztrációs funkcióhoz, hogy töröljék vagy új termékeket adjanak hozzá a weboldalhoz. Ez lehetővé teszi a könnyű hozzáférést bármely Androidos eszközről. Az utolsó lépés az egész rendszer tesztelése Seleniummal, ellenőrizve az olyan dolgokat, mint a bejelentkezés, regisztráció, jelszó visszaállítása, kosárba helyezés és törlés, rendelési funkció, új termék hozzáadása új rendelés, új termék hozzáadása, új rendelés módosítása, és így tovább. Mivel minden olyan gyorsan összejött, néhányunknak még iskola után is nehéz volt dolgozni, így el tudtunk merülni az esti munkában a Discordon, amit heti rendszerességgel tudtunk végezni.

5.1. Kezdet

A kezdeti célunk az volt, hogy a megfelelő package.json fájl elkészítése után létrehozzunk egy app.js fájlt, amely a következőket tartalmazta:

```
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');

const errorController = require('./controllers/error');
const User = require('./models/user');
```

Ezek a következők:

Az útvonal az útvonal, express az útvonal, amelyben különböző funkciók DELETE, POST, GET, body-parser az adatbázis hívására például egy adott termék azonosítóját, mongoose felelős az adatbázis feladatáért, session a felhasználó rögzítésére maga (e-mail, jelszó, token, bejelentkezési idő), és mongodb biztosítja az adatbázis-kapcsolatot, csurf felelős a token, amely elfelejtett jelszó esetén generálódik, és lehetővé teszi az e-mail értesítést.

A következő lépés az adatbázis-kapcsolat létrehozása, ami a következőket jelenti:

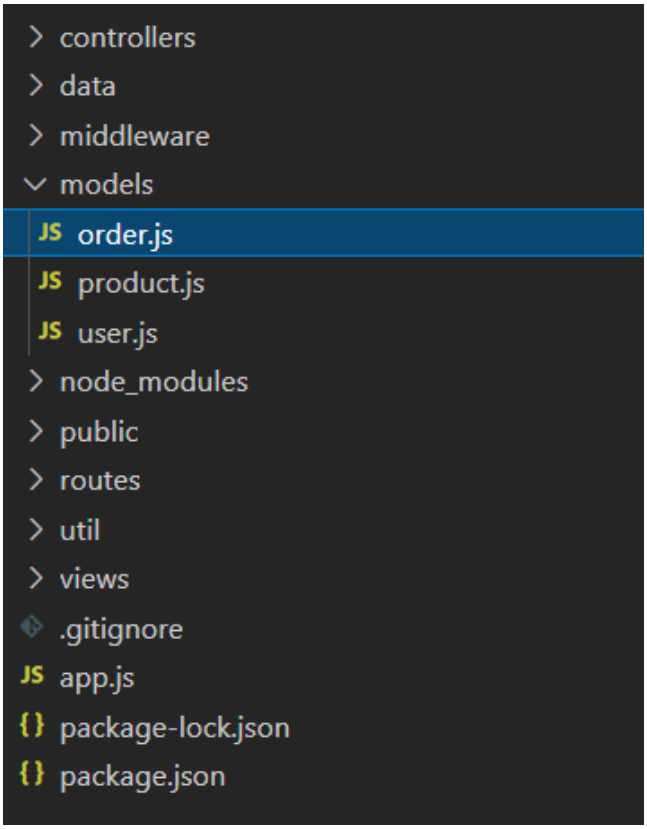
```
const MONGODB_URI =
    'mongodb+srv://scheuer_patrik:asdasd123@cluster0.icldh.mongodb.net/shop?authSource=admin&replicaSet=atlas-22dxd0-shard-0&readP

const app = express();
const store = new MongoDBStore({
    uri: MONGODB_URI,
    collection: 'sessions'
});
const csrfProtection = csrf();
```

Mindenekelőtt a connection funkciót látjuk, amely megjeleníti a MongoDB csatlakozási parancsot, elsősorban azt a nevet vagy jelszót, amelyet a meg kell adni ahhoz, hogy az adatbázisba való bejelentkezés helyes legyen, jelen esetben: codeine, majd a Codeine csatlakozik a shop nevű gyűjteményhez, és ha ez a gyűjtemény nem létezik, akkor létrehozza azt. Miután ez megtörtént, az express függvény tartalmazza a csrfProtectionnel ellátott kapcsolati párt, aminek eredményeképpen a jelszó titkosítva lesz, ahogy az a következő képen látható:

```
_id: ObjectId("6249d927ae2c5339e4e766a9")
title: "Űrutazás #1"
price: 10000
description: "Lorem Ipsum"
imageUrl: "https://picsum.photos/650"
userId: ObjectId("6245e4bdc1a5d4179886be48")
createdAt: 2022-04-03T17:28:07.300+00:00
__v: 0
```

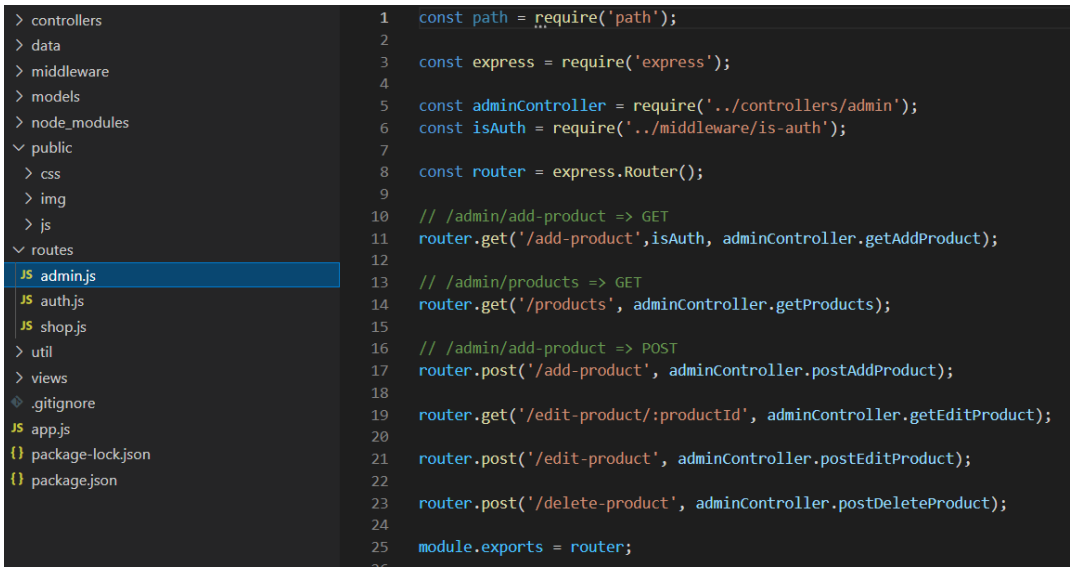
Az adatbázis lehetővé teszi az adatbázis tárolását és a vele való kommunikációt a következő módszerekkel:



A fájlok a sorrendben a bejelentkezési információk és a termékek által hozzáadott felhasználói idő, hogy azonosítsa azt a többi felhasználó. Ezek a megrendelések mappában találhatók. A termékleírás a termékhez tartozik, és a rögzítéskor kerül hozzáadásra. A felhasználói adatok közé tartozik az e-mail cím, a jelszó és a token (amely megváltoztatható), valamint a kosár tartalma. Amikor egy ilyen fájlt hozunk létre, felépítünk egy Sémát, amelyben megnevezzük a táblázatot és annak tulajdonságait, valamint leírjuk, hogy mi legyen kötelező és mi nem kötelező. Mivel ez egy termék, és mint látható, szükséges mező, itt kell megadni a termékobjektum típusát is, hogy az adatbázis ne tartalmazzon üres mezőket.

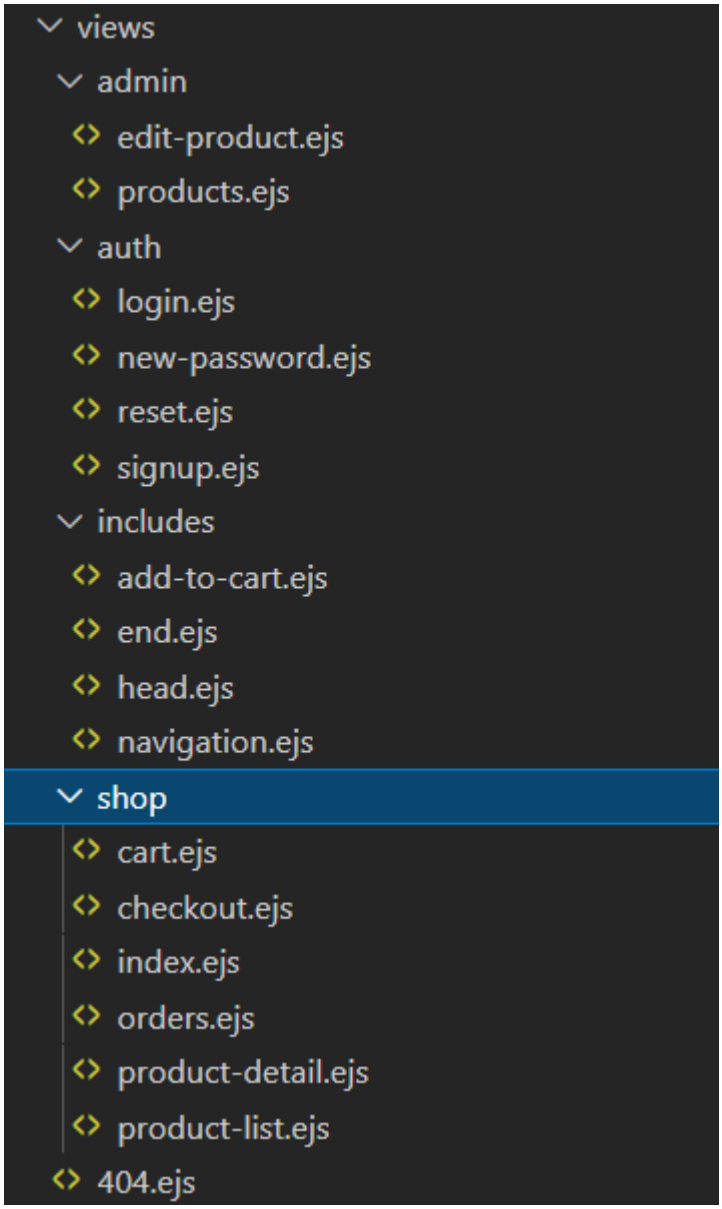
```
const orderSchema = new Schema({
  products: [
    {
      product: { type: Object, required: true },
      quantity: { type: Number, required: true }
    }
  ],
  user: {
    email: {
      type: String,
      required: true
    },
    userId: {
      type: Schema.Types.ObjectId,
      required: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

Ha az adatbázis-kapcsolat stabil, elkezdhetjük az útvonalak építését; ebben az esetben az admin, a hitelesítési és a bolti útvonalakra van szükség. Az adminisztrátor a termékek kiegészítéseiért felelős. a végtermék a szerkesztés a törölt termék Az egyszerűség kedvéért külön fájlban legyenek azoktól, akiket érdekel a program alapstruktúrája. És mivel az auth path kezeli a bejelentkezés, regisztráció, kijelentkezés és jelszókezelés minden aspektusát, nélküle nem tudnánk felhasználót létrehozni. A shop path az alkatrészekért felel, és ha már bejelentkeztünk, akkor a kosárba termékeket adhatunk hozzá, és megtekinthetjük a tartalmát. A vásárlás befejezéséhez vagy törölje az elemeket a kosárból, vagy kattintson a rendelés fülre a megrendeléshez.



5.2. Menük

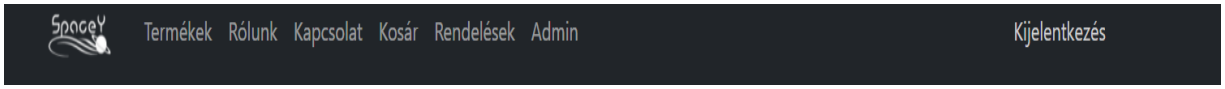
A második fő feladat az volt, hogy tervezze meg a menük ezeken belül kellett tervezni, melyik útvonal vezetne hová, így létrehoztunk egy nézetek mappát, amely osztott admin, auth, magában foglalja, és a bolt, és még a nézetek mappán belül kellett létrehozni egy ejs fájlt egy 404 hiba állapotkód, amely lenne egy, amely dob egy hibakódot, ha egy helytelen útvonal van megadva, vagy esetleg átirányítva, és az üzenet.



Az admin mappában található edit-product és product oldalak felelősek a termék megjelenéséért és érzéséért. Amikor egy rendszergazda hozzáad egy terméket, és meg akarja változtatni vagy törölni azt, akkor azt vizuálisan itt teheti meg, mert ezek az oldalak felelősek a megjelenésért. A bejelentkezésért felelős vizuális elemek az auth mappában találhatók. A navigációs menüsor, valamint az oldal eleje és vége, ahol láblécet adhat hozzá, mind az includes mappában található. A menüsor megjelenéséért felelős részeket a shop mappa tartalmazza. A 404-es mappa magától értetődő.

```
<div class="backdrop"></div>
<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>
```

Az itt látható kódrészlet jelentősége mindenekelőtt az, hogy a menüszerkezet úgy van felépítve, hogy szükségünk van egy navbarra, amelybe ezt az esetet be kell illesztenünk. Add-Product az Admin Product és a Shop Products, Carts, and Orders mellett Az aktív rész azt jelenti, hogy ha rákattintasz, akkor egy css stílusú jelölés jelzi, hogy most az adott menüpontnál vagy, a href rész pedig, ami a webcím mellett van, azt mondja meg, hogy hova irányított az oldal. Az isAuthenticated tulajdonság pedig azt jelzi, hogy a menüpontokat csak sikeres bejelentkezés után érheti el, azok számára lesz elérhető, akik be vannak jelentkezve, a többi pedig csak azok számára lesz látható, akik nem.



A két üzemmód közötti különbség, amelyet a bejelentkezés előtt, majd a bejelentkezés után láthat, szintén az alábbiakban látható.

5.3. Belső funkciók felépítése

Kezdetnek hozzon létre egy vezérlők mappát a projekt mappájában. Számos különböző js fájlt készítettünk, amelyek a következőképpen szerveződnek. Szükségünk van egy admin fájlra, amely beolvassa az adatokat a termékfájlból, amely a különböző termékeket tartalmazza, és lehetővé teszi számunkra, hogy létrehozzuk, módosítsuk és töröljük őket.

```
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);
    });
```

Az alábbiakban egy példa arra, hogy mit találhat itt: A termékek létrehozásához először meg kell adnia az alapvető információkat, például a nevet, a kép url-jét, az értéket és a leírást. Ezáltal egy új termék jön létre az adatbázisunkban. A save() metódus elmenti a létrehozott terméket az adatbázisba, ami feltölti a táblázatot adatokkal. Ezután következik az ág, amelynek feladata, hogy az admin/termék célsorra terelje a forgalmat, miután elkészült. Hiba esetén a hibák elkerülése érdekében ki kell írnunk a konzolra, így láthatjuk, hogy a folyamat mely részei készültek el, és mely részek nem.

A másik funkció a szerkesztés, ami a következőképpen működik: a termékek menüpontba lépünk, és ha rákattintunk egy általunk létrehozott termékre, akkor az a fentiek alapján szerkesztési módba lép, és ha elégedettek vagyunk vele, akkor visszairányít a termékekhez. Azt is fontos megjegyezni, hogy a prodID alapján keresi meg az adatbázisban, ami meghatározza, hogy csak a beépített funkciókat változtatjuk meg.

```
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));
};
```

A következő és egyben utolsó adminisztrációs lehetőség, hogy töröljük ezeket a termékeket, amelyeket az admin menübe helyeztünk.

a következő módon történik. Ha meg szeretné keresni a saját termék termék menüjét, látni fogja, hogy az userID alapú, így bárki, aki készített egyet, látni fogja.

A másik lehetőség a saját termék törlése itt, miután a létrehozott termék prodID-jét összevetette a felhasználó userID-jével. Kritikus, hogy a beépített függőségekkel együtt használja a deleteOne függvényt, mert csak így törölheti saját termékét.

```
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));
};
```

Ezután jön a bejelentkezési szakasz és a jelszó visszaállítása szakasz. Nagyjából ez a legfontosabb dolog egy weboldal kezelésének életében Egyszerűen feliratkozhatnak, hogy kifejezzék az emberek érdeklődését. Itt is a beépített funkciót használjuk modellről felhasználóra Képesnek kell lennünk a felhasználói adatok tárolására. Az adatok itt is alkalmazhatók a munkamenetre a jelszó visszaállítása funkcióval. Konkrét felhasználói információkat kell feltüntetnünk a felhasználói tokenben.

A nevének vagy jelszavának titkosításával, hogy visszafejtéssel tudjon bejelentkezni. Jogunk van rendszergazdai jogokat adni.

```
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'
  });
```

Mint látható, a megfelelő függőségek exportálása után használhatjuk az üzenetküldő rendszert, amely a megfelelő Api-kulcs használatával üzenetet küld arra az e-mail címre, amelynek elvesztettük a jelszavát, valamint a visszaállítási tokent.

Láthatjuk a bejelentkezési és regisztrációs oldalakat is, amelyek hibát jeleznek, ha a bejelentkezés sikertelen, és esetleg jelszó megerősítést, ha egy flash nevű függőséget használunk. Ehhez egy SendGrid nevű weboldalt használtunk, amely tökéletesen alkalmas ennek a funkciónak a betöltésére. A SendGrid (más néven Twilio SendGrid) egy ügyfélkommunikációs platform tranzakciós és marketing e-mailekhez, amelyet Denverben, Colorado államban alapítottak. A SendGrid egy felhőalapú megoldás, amely segíti a szervezeteket az e-mail kézbesítésben. A kézbesítési értesítések, a baráti kérések, a regisztrációs visszaigazolások és az e-

mail hírlevelek kezelését mind a szolgáltatás végzi.

```
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: 'abraham.domonkos@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)
      })
  })
}
```

A jelszó a crypto.randomBytes() metódus segítségével állítható vissza, amely kriptográfiailag jól felépített mesterséges véletlenszerű adatokat és a megírt kódban létrehozandó bájtok számát használja. Ebben a forgatókönyvben a User.findOne() metódust használtuk a bejelentkezés megkeresésére az e-mail cím alapján. Ha nem talált a megadott fiókhoz kapcsolódó e-mailt, hibaüzenettel tért vissza az előző oldalra. Ha működött, akkor a kódot e-mailben küldte el nekünk, és ha beírtuk, akkor meg tudtuk változtatni a jelszavunkat és be tudtunk jelentkezni az oldalra. A jelszó visszaállítását a következő tokenre kattintva kezdtük el, hogy az email szerint a kívánt weboldalra lépjünk, ehhez a transporter.sendMail() metódust használtuk segítségül, és szükségünk van egy.catch ágra is, hogy megakadályozzuk, hogy a hibáinkon túl folytassuk. Ha túl sokszor kérünk e-mailt a SendGrid nevű oldalon, az blokkolhatja a funkciót. Erre a fejlesztés során jöttünk rá, és ez egy kicsit megterhelte a funkció tesztelését és éles 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(hashedPassword => {
      resetUser.password = hashedPassword;
      resetUser.resetToken= undefined;
      resetUser.resetTokenExpiration = undefined;
      return resetUser.save();
    })
    .then(result => {
      res.redirect('login');
    })
    .catch(err => {
      console.log(err)
    })
}
```

Itt látható az új jelszó véglegesítése. Ehhez tudnia kell, hogyan hozzon létre egy newPassword attribútumot, amely a userID-hez és a passwordTokenhez van kötve. Ezt resetUser függvénynek hívjuk, és a User.findOne() metódust használjuk, amely még egy dátumot is tárol, hogy láthassuk, mikor történt az utolsó módosítás. Az új jelszó ezután hashelésre kerül, ami azt jelenti, hogy a bcrypt egy adaptív függvény, amely idővel növeli a körök számát, így a számítógépek teljesítményének növekedésével is ellenállóvá válik a brute force támadásokkal és a kényszerített keresési támadásokkal szemben. Ennek eredményeképpen egy új jelszóval rendelkezünk, amely ugyanolyan hozzáférési korlátozásokkal rendelkezik, mint a régi.

```
exports.get404 = (req, res, next) => {
  res.status(404).render('404', {
    pageTitle: 'Page Not Found',
    path: '/404'
  });
};
```

Érdemes megemlíteni a 404-es hibát is. Ez js-t tartalmazott, amely kapott egy státuszkódot és egy szöveget, és ejs-filezett, hogy úgy tűnjön, mintha a saját űrlaptervezésünk lenne.

```
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.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 van a következő, ami a shop.js fájl, ami már a weboldal legvégső fázisában van, és a modellek mappából exportálható, ami az összes terméket megjeleníti, és alapos leírást ad ez a kód részletesen. Vannak.catch ágak is, amelyek megakadályozzák a hibák és a megelőzhető művelet bekövetkezését. Ide tartozik a megrendelés következő fontos része, a kosár funkció is, ahol a felhasználónak van egy egyedi prodId azonosítója, amelyet törölhet. Azt is figyelembe vettük, hogy mi történne, ha az egyik felhasználó felvesz valamit, és beteszi a kosarába, de mi egy másik profilba vagyunk bejelentkezve, megakadályozva a termék duplikálását.

```
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.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));
};
```

A felhasználó, aki a kosár tartalmát elküldi a megrendeléshez, a következő attribútumokat rögzíti a megrendelések táblázatban: mennyiség és terméknév, majd a felhasználó és az e-mail alapján felépíti a táblázatot, törli a kosár tartalmát, és elmenti azt. A megrendelések menüben megjelenik a megrendelés. Ezt is ellenőriztük, hogy a felhasználók ne egymás rendeléseit lássák, hanem mindig a sajátjukat, így számos probléma és hiba kiküszöbölhető.

5.4. GET, POST kérések

A POST a World Wide Web által a számítástechnikában engedélyezett HTTP-kérelmezési módszer. A POST kérési módszer eleve arra kéri a webkiszolgálót,

hogy fogadja el a kérési üzenet testében lévő adatokat, esetleg tárolásra. [1] Gyakran használják egy kitöltött webes űrlap elküldésekor vagy egy fájl feltöltésekor. A HTTP GET kérési módszer ezzel szemben adatokat fogad a kiszolgálótól. A GET-kérés részeként az URL-lekérdezés kérdőjeles karakterláncához megadhatók bizonyos információk, amelyek (például) keresőkifejezéseket, dátumtartományokat vagy más, a lekérdezést meghatározó információkat tartalmaznak.

A kérési üzenet testében bármilyen mennyiségű, bármilyen típusú adatot meg lehet adni a kiszolgálónak küldött POST-kérés részeként. Az üzenettörzs internetes médiatípusa általában a POST-kérelem fejlécmezője. A POST és a GET, valamint a PUT-DELETE és számos más kérési módszer vagy "ige" áll rendelkezésre a világhálón és a HTTP-ben. Míg a legtöbb webböngésző csak a GET és a POST módszereket használja, a RESTful internetes alkalmazások számos különböző módszert használnak.

A POST egy olyan HTTP-módszer, amely egy új adategység reprezentációját küldi a kiszolgálónak, amely az URI által meghatározott erőforrás új gyermekeként kerül tárolásra. Mivel a legtöbb webböngésző csak a GET vagy a POST funkciót tudja használni, a tervezők kénytelenek voltak a POST-ot használni a különböző adatszolgáltatási és adatkezelési funkciók kezelésére, például a meglévő információk módosítására és eltávolítására. Számos űrlapot használnak a kiszolgálóra vonatkozó információk egyértelműbb meghatározására anélkül, hogy az alapadatbázist módosítani kellene.

A keresési űrlapok például kiválóan használhatók a method="get" használatával. Ez nem jelenti azt, hogy minden webes űrlap kezdő elemének tartalmaznia kell a method="post" attribútumot. Az alapértelmezett webes médiatípus az "application/x-www-form-urlencoded", amikor a webböngésző POST-kérést küld egy webes űrlapelemről. Ez a formátum olyan kulcs-érték párok tárolására szolgál, amelyeknek több kulcsuk is lehet. Az egyes kulcs-érték párokat egy "&" karakter választja el egymástól, míg az egyes kulcsokat egy "=" karakter választja el az értéküktől. A szóközök "+" karakterrel való helyettesítésével a kulcsok és az értékek elválasztásra kerülnek, és minden más nem alfanumerikus karaktert százalékos kódolással szabadítanak fel.

5.5. Summary

We've reached the conclusion of the application's demonstration. The characteristics of the application's features and capabilities. We covered how to design the algorithms needed to implement different functions during the presentation of the program, as well as the specifications for storing the necessary data and how to save it. The need for dynamic websites has risen considerably in recent years, and it is projected to continue to rise in the future. That is why I chose to build a dynamic website. All dynamic websites, in general, rely on data and are linked to a database; in my work, I used the MongoDb database management system. My thesis provided me with a wider understanding of the approaches used, allowing me to expand my experience and knowledge of the subject, as well as the software. As I aim to work in this subject, I hope that my thesis will serve as an excellent reference for my future work.

5.6. Tesztelés

A tesztelésnek két módja van: az egyik a weboldal tesztelése a Selenium segítségével, a másik pedig a mobilalkalmazás tesztelése a beépített Xamarin telefon vagy egy hagyományos Android telefon segítségével. A Selenium tesztek teljes integrált fejlesztőkörnyezete (IDE) különbözteti meg a Seleniumot. Firefox és Chrome bővítményként is elérhető. Bővítmény a Chrome számára. Lehetővé teszi a funkcionális tesztek rögzítését, módosítását és hibakeresését. A szkriptek automatikusan gyűjthetők és kézzel szerkeszthetők, automatikus kitöltéssel és a parancsok gyors mozgatásának lehetőségével. A tesztek különböző programozási nyelveken is létrehozhatók a shell nyelven való írás helyett. A Selenium Client API-t ezután a tesztek interfészének használatára használják jelenleg a Selenium támogatja a Java, C#, Ruby, JavaScript és R nyelveket és Python kliens API-kat biztosít.

Tests

Search tests...

Alapvető funkciók

Elfelejtett jelszó*

Elfelejtett jelszó 2*

http://localhost:3000

	Command	Target	Value
1	open	/reset/b153fb2d3dc01bc1eefcc1fbc95b86363da14cb92ac7f6a2c8c28d73f1a0ec07	
2	set window size	945x1020	
3	click	id=password	
4	type	id=password	adminadmin
5	click	css=.btn	
6	click	css=.card-body	
7	click	id=email	
8	type	id=email	kiss.adam@students.jedlik.eu
9	click	id=password	
10	type	id=password	adminadmin
11	send keys	id=password	${KEY_ENTER}

Lehetőség van egy mobilalkalmazás tesztelésére a Xamarin. Ehhez csak egy tesztet kell kifejleszteni, amit UITestnek hívunk, és amit Xamarin. Forms. Az UITest kompatibilis a Xamarin. Forms segítségével olyan felhasználói felületi teszteket írhatunk, amelyek több száz eszközön futtathatók a felhőben. Az iOS és Android alkalmazásokhoz a fejlesztők automatizált felhasználói felületi teszteket készíthetnek. Xamarin, kisebb módosításokkal. A Xamarin használható a Forms alkalmazások tesztelésére. UITest, amely magában foglalja a tesztkód megosztását. Az UITest automatizálja a felhasználói felületet azáltal, hogy a képernyőn megjelenő vezérlőelemeket aktiválja, és mindenhol, ahol a felhasználó rendszeresen kapcsolatba lép a programmal, bemenetet ad. A tesztkódnak biztosítania kell egy mechanizmust a képernyőn lévő vezérlőelemek leírására, hogy lehetővé tegye a teszteket, amelyek megnyomhatnak egy gombot vagy beírhatnak szöveget egy mezőbe. Minden egyes vezérlőelemnek saját egyedi azonosítóval kell rendelkeznie ahhoz, hogy az UITest kód képes legyen hivatkozni rá. Egy Xamarin alkalmazás. AutomationId tulajdonsága ajánlott az azonosító beállításához az űrlapokban.

Tests

Search tests...

Alapvető funkciók

Elfelejtett jelszó*

http://localhost:3000

	Command	Target	Value
1	open	/	
2	set window size	945x1020	
3	click	linkText=Login	
4	click	linkText=Forget Password?	
5	click	id=email	
6	type	id=email	kiss.adam@students.jedlik.eu
7	click	name=_csrf	

6. Összefoglalás

Dolgozatunk célja, hogy mások szemét is felnyissuk arra a kreatív és piaci résre, amely egyszerűen olyan gondolkodásmódot igényel, amely összhangban van a jelenlegi földgolyó legjelentősebb és legjövedelmezőbb részével, amely a turizmus, valamint a jelenlegi kriptopiacok. Ezért hoztuk létre ezt a projektet és szakdolgozatot, hogy foglalkozzunk a piacon és a kriptovalutákkal kapcsolatos kérdésekkel. A másik része az, hogy megmutathatjuk bármely cégnek egy üzleti megkeresés céljából, amelyet egy weboldallal és egy mobilalkalmazással végeztünk a jövőbeli munkahelyen, mint egy kis specifikáció, amelyet a saját előnyünkre használhatunk, vagy talán az üzleti életben, megmutathatjuk bármely cégnek egy üzleti megkeresés céljából, amelyet egy weboldallal és egy mobilalkalmazással végeztünk. A harmadik lehetőség, hogy külföldön adjuk el, ha van rá piac, hiszen ha mi vagyunk az elsők Magyarországon, akik ilyesmit csinálnak, akkor a külföldi közönség felfigyel rá, és a mai világban minden programban mindenki a pénzt keresi. Hogy könnyebb legyen a dolgunk, feltesszük ide, és talán eljut az üzenetünk.

6.1 Jövőbeli tervek

A jövőben szeretnék megvalóstani egy jobb és felhasználó barátab kezelői felületett. Az adatbázis továbbfejlesztését mint strukturálisan, és funkciónálisan bővteni szeretnénk. Nagyobb adminfelület hozzáadása a könnyeb kezelhetőség szempontjából.

7. Felhasznált irodalmak

- https://www.w3schools.com/nodejs/
- https://www.stackoverflow.com/
- https://www.nodejs.org/en/
- https://hu.wikipedia.org/wiki/Node.js
- https://docs.microsoft.com/hu-hu/azure/developer/javascript/
- https://docs.microsoft.com/en-us/cpp/cross-platform/?view=msvc-170
- https://docs.microsoft.com/en-us/appcenter/errors/
- https://gobertpartners.com/what-mongodb-used-for
- https://visualstudio.microsoft.com
- https://hu.wikipedia.org/wiki/Weblap
- http://meniszkusz.hu/pag/structure.html