ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

ÓE-NIK **Papp Bence**

2022 **T/007044/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Papp Bence**
Törzskönyvi száma: T/007044/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Webes kiskereskedési platform online fizetéssel és automatikus számlázással
Web retail platform with online payment and automatic billing

Intézményi konzulens: Kovács András
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Szoftvertechnológia és grafikus felhasználói interfész tervezése
Szoftvertervezés és -fejlesztés specializáció

A feladat

A hallgató feladata egy platform elkészítése, amely kiskereskedők számára nyújt segítséget termékeik népszerűsítésében és eladásában. Az elkészítendő alkalmazást webes platformra kell kialakítani, szükséges a hatályos GPDR törvényeknek megfelelnie, online fizetést és online számlázást biztosítania. Egy kereskedő saját aloldalt hozhat létre az üzletet számára, ahol saját maga képes a termékeket feltölteni, leírást készíteni, beállítani az árazást és különböző kedvezményeket rögzíteni. A rendszerben a vásárló és az eladó mint két független szerepkör jelenjen meg, az utóbbit bővítse a rá jellemző többlet jogosultságokkal. A rendszer megvalósításakor tartsa be a rétegzés elveit, válassza szét a felhasználói felületet és az üzleti logikát. A két rendszer közti kommunikáció felépítésekor vizsgálja meg a REST és a GraphQL közti különbségeket, indokolja mérnöki döntéseit!

A dolgozatnak tartalmaznia kell:

- a szakirodalom részletes áttekintését és értékelését,
- a megoldandó feladat pontos leírását és részletes elemzését,
- a meglévő rendszerek elemzését,
- az alkalmazni kívánt technológiák elemzését, döntések indoklását,
- a rendszertervet,
- a megvalósítás pontos leírását,
- a tesztelési tervet, teszteredményeket, a fejlesztés során felmerült problémák elemzését,
- az elkészült rendszer felhasználási és továbbfejlesztési lehetőségeit.

ÓBUDAI EGYETEM Neumann János Informatikai Kar 1. Budapest Ph

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

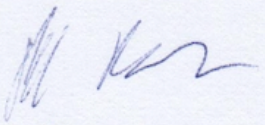
.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022. május 6.



hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun Kód: Tagozat:

Papp Bence [redacted] NAPPALI

Telefon: Levelezési cím (pl.: lakcím):

[redacted]

Szakdolgozat címe magyarul:

Webes kiskereskedési platform online fizetéssel és automatikus számlázással

Szakdolgozat címe angolul:

Webes kiskereskedési platform online fizetéssel és automatikus számlázással

Intézményi konzulens: Külső konzulens:

Kovács András

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 09. 20	Téma tervezése	
2.	2021. 10. 18	Irodalomkutatás ellenőrzése	
3.	2021. 11. 15	Rendszer tervezése	
4.	2021. 12. 10	Végső ellenőrzés, prezentációs próba	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a **Szakdolgozat I. (BSc)** tantárgy követelményét teljesítette, **beszámolóra** bocsátható.

.....

Intézményi konzulens

Budapest, 2021. 12. 10

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun Kód:	Tagozat:
PAPP BENCE	[redacted]	NAPPALI

Telefon: Levelezési cím (pl.: lakcím):

[redacted]

Szakdolgozat címe magyarul:

WEBES KISKERESKEDÉSI PLATFORM ONLINE FIZETÉSSEL ÉS AUTOMATIKUS SZÁMLÁZÁSSAL

Szakdolgozat címe angolul:

WEBES KISKERESKEDÉSI PLATFORM ONLINE FIZETÉSSEL ÉS AUTOMATIKUS SZÁMLÁZÁSSAL

Intézményi konzulens: Külső konzulens:

KOVÁCS ANDRÁS

Kérjük, hogy az adatokat nyomtatott nagy betűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.15.	IMPLEMENTÁCIÓ TERVEZÉS	
2.	2022.03.25.	IMPLEMENTÁCIÓ ÁTTEKINTÉSE	
3.	2022.04.10.	TESZTELÉSEK KIÉRTÉKELÉSE	
4.	2022.05.10.	VÉGSŐ ELLENŐRZÉS, LEZÁRÁS	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat II. tantárgy követelményét teljesítette, védésre bocsátható.

.....

Intézményi konzulens

Budapest, 2022. május 10.

Tartalom

1. BEVEZETÉS

A ma ismert internet felhasználók milliárdjait kapcsolja össze. Az első weboldal megszületése óta eltelt nagyjából 30 év rengeteg változást és technológiai fejlődést hozott a webes alkalmazások világába. Az első HTML kódsoroktól eljutottunk azokig a technológiákig, amik a mai modern webfejlesztést meghatározzák.

Szakdolgozatom célja egy olyan üzleti célra felhasználható webes platform létrehozása, amely lehetővé teszi az élelmiszeripar kis szereplői számára szolgáltatásaik népszerűsítését.

A pandémiás időszak nagyon nagy hatást gyakorolt a társadalomra és a gazdaságra. Nagyon sok nagyvállalat, áruházlánc óriási fejlesztésekbe kezdett, hogy a vásárlóit ki tudja szolgálni. Ezekre általában egy webes áruház lett a megoldás, ahol a vásárlók éppúgy el tudják intézni a hétvégi nagybevásárlást, mint azelőtt, sőt most egyenesen az ajtó elé kapják a termékeket. Azonban az élelmiszeripar nem minden szereplője volt ilyen szerencsés helyzetben. Nagyon sok kiskereskedő/őstermelő már a pandémiát megelőző időszakban rendelkezett egy - nagyjából állandónak mondható - vásárlói körrel. Sokan azonban a hirtelen bekövetkező változásokat nem tudták követni, többen csődbe mentek annak ellenére is, hogy termékeik ugyanolyan jó minőséget képviseltek, de nem tudták őket eladni, hiszen ismeretségük lehet csak néhány faluig terjedt.

Erre a problémára szeretném a platformot létrehozni, ahol az ilyen kis méretű vállalkozások is legalább akkora szerepet kaphatnak, mint az egész üzletláncokat üzemeltető cégek. A platform lehetőséget biztosít az egyszerű reklámozás mellett termékek árusítására, előrendelésére. A teljes fizetés lebonyolításáról (számlázással együtt) az alkalmazás fog gondoskodni. A fizetésre PayPal vagy Barion Pay rendszeren keresztül lesz lehetőség. Továbbá bemutatásra kerülnek a fejlesztés során használt technológiák, a Next JS, Tailwind CSS, GraphQL, ASP.Net és a MySQL.

A modern webes alkalmazások két fő részre bonthatók, kliens- és szerveroldalra. A köztük történő kommunikációért az úgynevezett REST API a felelős. Ezzel van lehetősége a két oldalnak adatok küldésére, illetve fogadására, feldolgozására. A REST HTTP kérések segítségével teszi lehetővé a kommunikációt. Egy alkalmazás futása során akár több ezer ilyen kérés is végbemehet. Sok esetben a kliens számára „felesleges” adatokat is kap a szervertől, amikre ott és akkor nincs is szüksége. Ez biztonság szempontjából sem előnyös, illetve képes nagymértékben leterhelni a szervert, ami az alkalmazás lassulásához vezethet. Ezen problémák kiküszöbölése érdekében választottam a GraphQL-t. A GraphQL egyre nagyobb térhódítása arra enged következtetni, hogy ez egy biztos pont lehet az alkalmazás jövőbeni fejlesztésére és karbantartására nézve.

A mai webes alkalmazásokhoz támasztott igények kielégítése komplex feladat. Ha egy alkalmazás felépítését egy felhasználó szemszögéből vizsgáljuk, akkor világossá válik, hogy ők csak a kliensoldalon megjelenő tartalommal fognak találkozni, a mögöttes megoldásokat vagy egyáltalán nem, vagy csak kismértékben ismerik.

Egy jól használható alkalmazás alapját a megfelelő szerveroldal és egy jól strukturált adatbázis adja. Az ASP.Net nagyon sok megoldást és eszközt biztosít a fejlesztők számára, mindezt egy nagyon áttekinthető és jól karbantartható módon teszi. A MySQL az ASP.Net számára jól támogatott adatbázis óriási felhasználói táborral.

Szakdolgozatom ezen technológiák széleskörű felhasználása mellett keres megoldást a fentebb leírt, jelenleg is érezhető problémára.

2. HASONLÓ RENDSZEREK ELEMZÉSE

A webes világban nem újkeletű dolog a közösségi platformok, illetve a webáruházak használata sem. Véleményem szerint minden felhasználónak, akik ilyen típusú oldalakat látogatnak, megvannak a személyes preferenciái. Ezen preferenciák az ételrendelésre is igazak. Ezért történhetett, hogy külön az ételekre fókuszáló szolgáltatások és platformok ekkora körben elterjedhettek világszerte. Mivel szakdolgozatom témája elsősorban élelmiszeripari termékekre helyezi a hangsúlyt, így kezdetben egy Magyarországon jól ismert és használt alkalmazást mutatok be, ami a Foodpanda. A platform nemzetközi szinten több országban is jelen van, de szakdolgozatomban csak a Magyarországra vonatkozó részével fogok foglalkozni, tekintve, hogy az általam bemutatni kívánt alkalmazás is Magyarországra fókuszál és nem nemzetközi piacra.

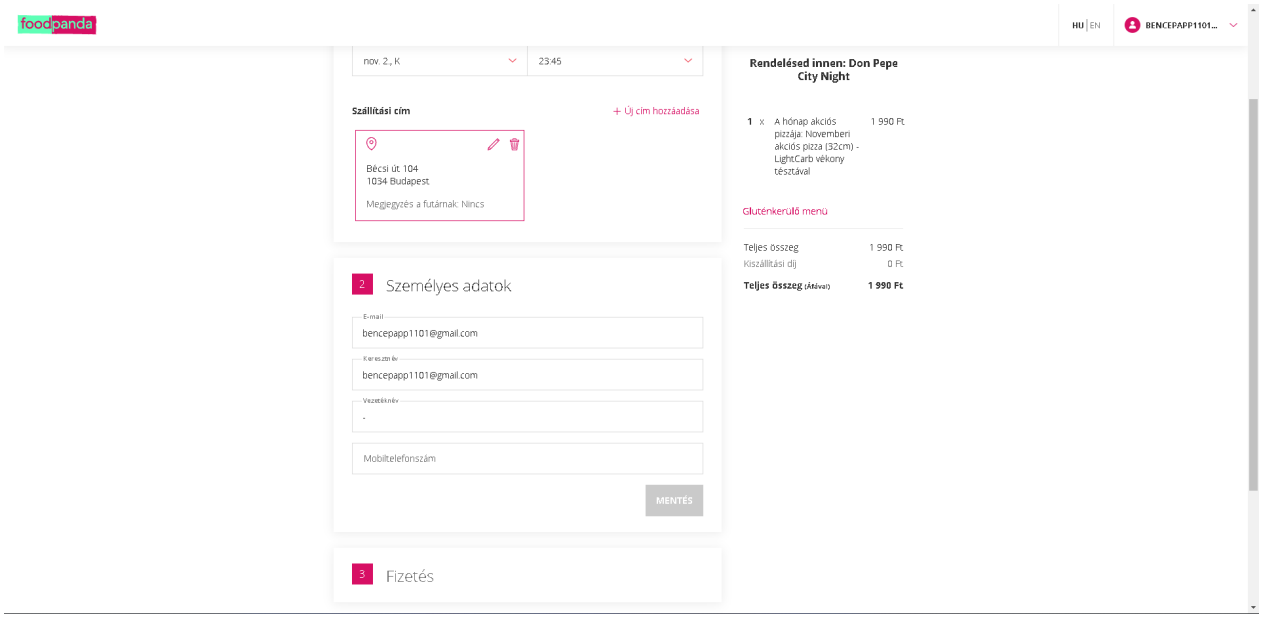
A Foodpanda egy ételrendelésre használt platform, korábbi nevén NetPincér. Az oldalon lehetőségünk van éttermekből előre elkészített ételt rendelni, illetve nagybevásárlásainkat is intézhetjük az oldalon, azon belül a Panda Marketen. Az alkalmazás arra helyezi a hangsúlyt, hogy az ételt minél hamarabb megkapja a vásárló. Erre két lehetősége van: vagy elmegy a kiválasztott étterembe és felveszi az előre leadott rendelést vagy házhoz rendeli a termékeket. Kezdetben a kiszállítás lefedettsége Budapestre volt szűkítve, ez mára több nagyvárosban is elérhető szolgáltatás Magyarországon. Az ételrendelés az oldalon felhasználói fiókhoz kötött. Itt van lehetőségünk fiók létrehozására egyénileg, de akár Facebook fiókunkat is használhatjuk, Facebook autentikáció segítségével.

Az étterem kiválasztása előtt érdemes lokáció alapján szűkíteni a keresést. Egy-egy étteremből való kiszállítás maximális távolsága eltérhet, illetve különböző kiszállítási ár is lehet (akár ingyenes is). Ezután kiválaszthatjuk a rendelni kívánt ételt. Van lehetőségünk egyszerűen csak böngészni az ételek közt, de ha van konkrét elképzelésünk arra, hogy milyen típusú ételt szeretnénk fogyasztani, akkor kategóriák alapján is szűkíthetjük a keresést, hogy már csak az éttermet kelljen megválasztanunk. Az étel kiválasztása után leadhatjuk rendelésünket. Általában az étteremtől függően az oldal felkínál nekünk választható extrákat az étel mellé, például extra szószokat. Ez általában a rendelés véglegesítése előtt történik, így ekkor rendelésünk még csak szerkeszthető állapotban van, amin még lehet változtatni. A kosár megtekintése esetén változtathatunk még rendelésünkön, eltávolíthatunk tételeket vagy megnövelhetjük a rendelni kívánt mennyiséget. Mivel az oldal a bankkártyás fizetésre helyezi a hangsúlyt, itt van lehetőségünk a futárnak borravalót adni, nem pedig a rendelés átvételénél. Ezután a fizetésre ugorhatunk tovább, ahol többféle fizetési mód közül választhatunk.

Dolgozatomban a bankkártyás fizetés kerül középpontba, ezért én ezt a módot szeretném jobban bemutatni. Kiszállítási adataink megadása után választhatunk bankkártyás fizetési módot. Van lehetőségünk PayPallal, Barion Payjel vagy akár Apple Payjel is fizetni.

Az általam elkészíteni kívánt alkalmazás is PayPal és Barion Pay fizetési lehetőségeket fog támogatni. A Foodpanda egyik új funkciója a későbbi időpontra leadható ételrendelés. Ez azt jelenti, hogy akár hetekre előre is leadhatunk rendeléseket. Ezt a funkciót az általam tervezett alkalmazásban is használom. Ez nem újkeletű dolog a vendéglátásban, de az online ételrendelési platformok esetében még annak mondható. A Foodpanda éttermekkel áll kapcsolatban, így számukra a megrendelések későbbi elkészítése nem jelent problémát. A tervezett alkalmazás esetében nem éttermekről, hanem kisvállalkozásokról, kiskereskedésekről beszélhetünk inkább. Az ő számukra ez a megrendelési forma kicsit másként működik. Az ő termékeik nem biztos, hogy egész évben elérhetőek, így ezekre a periódusokra is figyelnie kell majd az alkalmazásnak.

A Foodpanda fejlesztői szemmel nézve egy modern alkalmazás. A felhasználói felület áttekinthető, könnyen kezelhető és modern designt kapott. A felület követi a mai trendeket, és az alkalmazás fő funkciói is folyamatosan alkalmazkodnak az ügyfelek igényeihez. Az oldal fő technológiái közül kiemelkedik a JavaScript, azon belül a React és a Node.js (Express keretrendszer), továbbá megtalálható a php és a bootstrap is az oldalon, valamint a Google Maps Platform API, ami a térképes funkciókért felel.



2.1. ábra: A Foodpanda ételrendelés folyamata

Az elmúlt évtizedben Magyarországon is egyre nagyobb hangsúlyt kapott az egészséges életmód, azon belül is a testmozgás és különösképp a megfelelő táplálkozás. Az ÉtkezzJól! nevű platform erre alapozta meg kínálatát. Az oldalon csak olyan ételek, illetve ételekből álló menük találhatóak, amiket a vásárlók könnyen beilleszthetnek az étrendjükbe. Maga a rendelés nem a hétköznapi ételrendelésre hasonlít, sokkal inkább egy szolgáltatásra emlékeztet. Különböző „bérletek" közül válogathatunk, attól függően, hogy milyen étrendet követünk vagy melyik szakaszában vagyunk a diétánknak. Van lehetőségünk különleges csomagok közül is választani, mint például a két fő részére szóló „home office bérlet". Egy csomag napi háromszori étkezést tartalmaz, amit egyszerre szállítanak ki minden nap. Egy bérlet, az étkezések számától függetlenül, fél éven belül használható fel rugalmasan, amikor a vásárló szeretné. Ez a rugalmasság lehetővé teszi, hogy valóban csak akkor vegyük igénybe a szolgáltatást, amikor tényleg szükség van rá. Egy csomag árát az is meghatározza, hogy pontosan mekkora a testsúlyunk. Ennek megadásával valóban csak annyi makrotápanyagot fognak az ételek tartalmazni, amikre ténylegesen szüksége van a szervezetünknek.

Az ÉtkezzJól! weboldala hordozza azokat a funkciókat, amiket egy ilyen platformtól elvárhatunk. Az oldal reszponzív felépítést kapott, továbbá van lehetőség rendelésre és online bankkártyás fizetésre is. Design szempontjából nem túl szemkímélő a színválasztás, de elfogadható. A weboldalon több helyen videók találhatók, beágyazott videók formájában. A folyamatos GDPR változások miatt az iframe-k használata mára nem számít jó megoldásnak. Az oldalon használt legfontosabb technológiák a jquery, illetve a php.

2.2. ábra: Az ÉtkezzJól! weboldala

3. KAPCSOLÓDÓ TÉMAKÖRÖK

3.1. Ételrendelés kultúrája

„Mi legyen ma a vacsora? Pizza? Sushi? Sajtburger?” Az ételkiszállítás elterjedésével ez a kérdés már-már hétköznapinak mondható manapság. Az első feljegyzett ételkiszállítástól, ami Olaszországban történt 1889-ben egy pizzával, az Indiában a 19. század óta hétköznapinak mondható Dabbawalasokon és a kínai ételfutárokon át, az internet elterjedésével eljuthattunk azon weboldalak „őseihez”, amiket a mai napig használunk ezekhez a rendeléseinkhez. Az első online történt ételrendelésben szintén a pizza játszotta a főszerepet, amit a Pizza Hut-ból rendeltek. Szakdolgozatom szempontjából azonban a leglényegesebb weboldal a waiter.com, ugyanis először ez az oldal volt, ami 60 különböző étterem kínálatát egyesítette kezdetben. [1] Ezzel megszületett az első ételkiszállítással foglalkozó szolgáltatás. Az elkészíteni kívánt platform is hasonló mintát követ. A waiter.com koncepciója mára már jól ismert. A legnagyobb fókuszt a különböző ételek kiszállítása, nem pedig azok elkészítése kapja. A platform, lényegét tekintve, megteremteti a kapcsolatot a kereskedő és a vásárló között.

3.2. GraphQL és a Hagyományos REST

A Representational State Transfer, röviden REST egy szoftverarchitektúra. Először Roy Fielding definiálta doktori disszertációjában. Koncepciójában egy REST típusú architektúra kliensekből, valamint szerverekből áll. A kliensek kéréseket indítanak, a szerverek pedig ezen kéréseket dolgozzák fel. Alapvetően egy kliensnek kétféle állapota lehet egy adott időpillanatban. Lehet állapotok közti átmenetben vagy úgynevezett „nyugalmi állapotban”. A kliensek nyugalmi állapotban képesek interakcióra a felhasználóval, de ilyen esetben nem terheli semmilyen formában a szervert. Abban az esetben, ha a szerver kész az állapotok közti átmenetre, vagyis kész átlépni egy új állapotba, akkor kéréseket küld a szerver irányába. Egyszerre akár több kérés is lehet függőben, de a kliens egészen addig átmeneti állapotban marad, míg legalább egyre nem érkezik válasz.

A REST eredeti rendeltetése a http-hez kapcsolódott, de ez nem jelenti, hogy egyedül erre a protokollra lenne lekorlátozva. Azokat az alkalmazásokat, amelyek megfelelnek a REST-nek, RESTful architektúrának nevezzük. Egy ilyen architektúra más alkalmazási rétegbeli protokollra is épülhet. Egy hagyományos REST típusú architektúrában különböző HTTP kéréseket különböztethetünk meg. A Hyper Text Transfer Protocol (rövidítve HTTP) információátviteli protokoll. A HTTP egy kérés-válasz alapú protokoll kliens és szerver között. A protokollt alapvetően a World Wild Web fejlődésével lehet kapcsolatba hozni, de nem kizáróan webböngészők képesek ezzel a protokollal kommunikálni. A HTTP a TCP/IP réteg felett helyezkedik el. A HTTP implementálható más megbízható szállítási

réteg felett is, akár az interneten, akár más hálózaton. Kizárólagosan TCP protokollt használ, mivel az adatveszteség nem megengedhető. Egy HTTP kérés felépítését tekintve mindig a kérés típusával kezdődik. Ezt követi az a végpont, ahova a kérést el kell küldeni. A végpontot az úgynevezett fejlécérték követi, ez tetszőleges számú sorból állhat. Ebbe a szekcióba kerülhet, többek között, a böngésző típusa vagy a kliens és szerver közti cachelésére vonatkozó részek is.

Metódusokban összesen kilencfélét különböztethetünk meg. A GET kéréssel általában a szerveren lévő adatokból kérünk le. Ez a kéréstípus csak olvasást végez és nem módosít adatot. A böngésző is ilyen kéréseket kezdeményez a szerver felé egy-egy frissen megnyitott oldal betöltésekor. A HEAD kérés hasonlóan működik, mint a GET, azzal a kivétellel, hogy az üzenettestet kihagyja a visszaérkező válaszból. A POST kérések feldolgozásra szánt adatokat küldenek a szervernek, ilyen lehet például egy HTML űrlap tartalma. Ebben az esetben a kérés teste az űrlapban szereplő adatokat fogja tartalmazni. A kéréssel érkező adatokat ezután a szerveroldal dolgozza fel, ez legtöbb esetben új adat létrehozását jelenti, de már meglévő adatokat is módosíthat. A PUT és a PATCH nagyon hasonlóan működnek. Ezekkel a kérésekkel általában adatmódosítást végzünk már létező adatokon, de ilyen kérés általában egy oldalra történő bejelentkezés is. A kérés hasonló felépítésű, mint a POST, csak itt az üzenettestben lévő adatok azt tartalmazzák, hogy a módosítani kívánt adat végeredményében hogyan nézzen ki. A DELETE kérést akkor használjuk, ha adatokat törlünk a szerver által kezelt adatbázis(ok)ból. A kérésnek általában nincs üzenetteste, a végpont címébe kerül bele a törölni kívánt entitás id-je vagy valamilyen egyedi azonosítója, a szerver ez alapján találja meg és törli az adatot.

A TRACE visszaküldi a kapott kérést. Ezt kevés esetben alkalmazzák, de hasznos, ha a kliensoldal arra kíváncsi, hogy a köztes eszközök változtatnak-e, illetve mit változtatnak a kérésen. Előfordul, hogy olyan típusú kérést küldünk a szerver felé, amit az nem támogat. Az OPTIONS kérés ebben nyújt segítséget, ugyanis visszaadja a szerver által támogatott HTTP metódusok listáját. A CONNECT képes a kérést transzparens TCP/IP-csatornává alakítani. Ez is egy keveset használt kéréstípus, jellemzően SSL kommunikáció megvalósításához használják. A kliensek által küldött kéréstípusok mellett a szerver által küldött válaszokat is megkülönböztetjük. Ezek megkülönböztetésére különféle státuszkódokat alkalmazunk. A HTTP válasz első sora tartalmazza a státuszt. Ez a státuszkódok részletes jelentését az RFC 2616 tartalmazza. Az egyes számmal kezdődő státuszkódok informatív jellegű válaszokat tartalmaznak. A kettessel kezdődők sikeres kéréseket jelentenek, például a 200-as kód, az úgynevezett OK. A hárommal kezdődők átirányításra vonatkoznak. A négyes kezdetű kódok kliensoldali hibákra utalnak. Ha ilyen válaszokat látunk, akkor abból következtethetünk arra, hogy a hiba a kliensoldalon keresendő, a kérés üzenetteste szintaktikailag helytelen vagy nem teljesíthető. Ilyen például

a 400-as kód, ami a kérés üzenettestében lévő hibára utal. Az ötös kezdetű státuszkódok pont ellentétesen, szerveroldali hibákra utalnak. Ebben az esetben a szerver nem tudta teljesíteni az egyébként helyesen küldött kérést. [2]

A GraphQL egy nyílt forráskódú adatlekérdező és manipulációs nyelv API. A Meta által fejlesztett rendszer megszületése 2012-ig vezethető vissza, első hivatalos megjelenése 2015-ben volt. Végül a projekt 2018-ban átkerült az újonnan létrehozott GraphQL Foundation berkei alá, amelynek tulajdonosa a Linux Foundation. A technológia megalkotója Lee Byron volt, fő célja a technológiát illetően, hogy mindenhol jelen legyen a webes platformokon. [3] A hagyományos REST-hez képest egy másfajta megközelítést biztosít a fejlesztők számára. Lehetőséget biztosít a kliensek számára, hogy meghatározzák a szükséges adatok struktúráját, szerkezetét. Ezzel lehetővé téve azt, hogy a klienshez valóban csak azok az adatok jussanak el a szervertől, amikre valóban szüksége van.

```
{
  hero {
    name
    height
    mass
  }
}
```

```
{
  "hero": {
      "name": "Luke Skywalker",
      "height": 1.72,
      "mass": 77
  }
}
```

3.1. ábra: GraphQL lekérdezés be- és kimenete

Ezzel a módszerrel a kliens hatást gyakorol a lekérdezések eredményére, illetve a gyorsítótárazásra is. A REST-hez hasonlóan, támogatja az adatlekérdező és -módosító műveleteket, valamint a valós idejű adatmódosításra való feliratkozást is, leggyakrabban websocket segítségével. A GraphQL, hasonlóan a REST-hez, végpontokkal kommunikál. Azonban egy szerverhez elég egyetlen végpont. Ez nagyban könnyíti a kérések elküldését. A GraphQL-lel küldött kérések az esetek túlnyomó részében POST típusú kérések (néhány könyvtár támogat GET kéréseket, de nem jellemző a POST-on kívül mást használni). Ennek oka, hogy a kérésekben lekérdezéseket írunk. A GET típusú kéréseknek nincs üzenetteste,

így a lekérdezés ebben az esetben csak az URL lekérdezési paraméterként küldhető el. Azonban sok szerver biztonsági okokból tiltja a túl hosszú címeket, egy GraphQL lekérdezés pedig könnyen elérheti ezt a hosszúságot. Ebben az esetben a szerverről csak egy 414-es státuszú választ fogunk kapni, tehát egy kliensoldali hibát fog jelezni nekünk. Ezért a legjobb gyakorlat az, hogy POST kéréseket küldünk, amiknek lehet üzenetteste, így ezek a hibák könnyedén elkerülhetőek.

3.3. JavaScript

A JavaScript egy magas szintű, objektumorientált programozási nyelv. Procedurális, imperatív és deklaratív egyben. Protoype-based, vagyis a primitív típusokon kívül minden egy objektumnak számít. A JavaScript prototype tervezési minta alapján valósítja meg az objektumorientáltságot. A legelső példányt kinevezi osztálynak, ezzel definiálva a működést. Később erről készítünk másolatokat származtatással. A tervezési minta jellemzői közül kiemelhető, hogy másolás esetén a memóriahelyet is lemásolja. A JavaScript sajátossága továbbá, hogy a különböző függvények úgynevezett „first-class function” -k. A függvények úgy vannak kezelve, mint a változók, ezért át tudjuk adni őket más függvények paramétereként. Dinamikusan típusos nyelv. Nem adjuk meg az adat típusát a változóknak, csak fordításkor dől el, hogy az adott változó milyen típusú, a változó típus futási időben is változhat. A JavaScript egy szálon dolgozik, tehát egyszerre egy műveletet tud végrehajtani, így a JS motor a háttérbe rakja a sokáig tartó műveleteket, és ha kész vannak, visszateszi őket a fő szálra. [4]

3.4. CSS

A Cascading Style Sheets, röviden CSS, egy stíluslapnyelv. Elsősorban weboldalak vagy natív mobilos alkalmazások formázásához használt leíró nyelv. A CSS segítségével különböző stíluslapokat hozhatunk létre oldalaink formázásához. Lehetővé teszi a tartalom és a megjelenítés elkülönítését, beleértve a betűtípusok, színek és az elrendezést is. Ez nagyban könnyíti a tartalom hozzáférhetőségét. Háromféle módon tudunk egy webalkalmazásban CSS-t alkalmazni. Tudunk belső CSS-t alkalmazni, ebben az esetben a <head> tag nyitó és lezáró tag-ben elhelyezünk egy <style> tag-et, ezen a tag-en belül van lehetőségünk CSS leíró nyelven stílusokat meghatározni.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
        body {
            background-color: blue;
        }
        h1 {
            color: red;
            padding: 60px;
        }
        </style>
    <title>Szakdolgozat</title>
</head>
<body>
    <h1>Szakdolgozat</h1>
</body>
</html>
```

3.2. ábra: Belső CSS használata

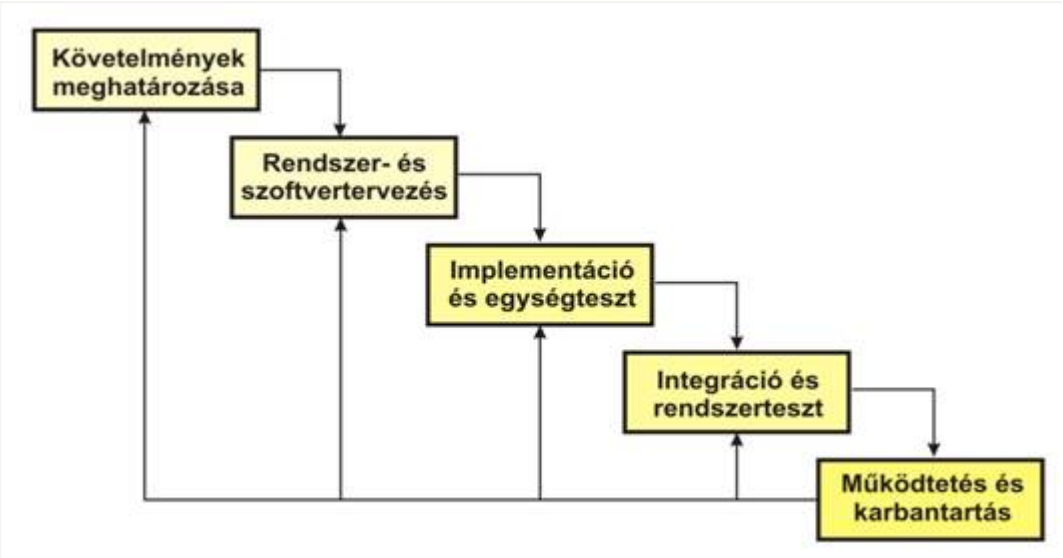
Nagy előnye a beágyazott formázáshoz képest, hogy könnyen alkalmazhatók a CSS-ben megszokott kiválasztók, így kevesebb formázással is ugyanaz az eredmény érhető el. Egy másik módja a stíluslapnyelv alkalmazásának az úgynevezett külső CSS lapok alkalmazása. Ebben az esetben az oldal formázása teljesen külön fájlban történik. A HTML oldalunkban beimportálásra kerül a használni kívánt CSS fájl. A leírónyelv használatának harmadik módja a beágyazott formázás. Ez a módszer mindig egy adott HTML elem stílusának meghatározására szolgál. Ebben az esetben az adott elemhez egy stílus attribútumot kell rendelni és már használható is a CSS. Ilyenkor azonban nem alkalmazhatók szelektorok, mindig csak az adott HTML elem lesz formázható azzal a stílussal, illetve a gyerekelemei.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Szakdolgozat</title>
</head>
<body style="background-color:black;">
    <h1 style="color:white;padding:30px;">Szakdolgozat</h1>
    <p style="color:white ;">Hasznos információk.</p>
</body>
</html>
```

3.3. ábra: Beágyazott CSS használata

Az első webböngészők még maguk döntötték el, hogy az adott weboldal hogyan nézzen ki. Azonban a CSS1 1996-os megjelenését követően egy egészen új világot köszönthettek a webes fejlesztéssel foglalkozó fejlesztők. [5]

3.5. Vízesés modell



3.4. ábra: Royce-féle Vízesés modell

A vízesés modell egy projektmenedzsment módszertan. Egy szoftverprojektre több, egymáshoz szorosan kapcsolódó, lépést határoz meg, amit a fejlesztés során követni kell. Az első lépés a követelmények pontos definiálása. Ez azt jelenti, hogy a fejlesztendő alkalmazástól elvárt minden követelményt meg kell határoznunk a projekt indításakor. Ennek a lépésnek a része a tervezés is, ami egy ideális projektben, akár a teljes projekt felét is kitöltheti. A következő lépcsőfok maga a fejlesztés, amely során elkészítjük az alkalmazást.

Az alkalmazás elkészítése után tesztelési folyamatok következnek. Első körben implementációs és egységteszteket hajtunk végre a projekten. Egységtesztelés során a forráskód egységeit teszteljük különböző tesztesetekkel, figyelve arra, hogy a lehető legátfogóbb teszteseteket hajtsuk végre (például határérték-tesztelés). Az egységtesztek fő célja megbizonyosodni arról, hogy a forráskód tesztelt egysége minden esetben eléri a kitűzött célt. Implementációs tesztek során az új kódrészletet implementáljuk a már meglévő alkalmazásunkba. A folyamat során azt vizsgáljuk, hogy ezután is a kívánt módon működik-e a kibővített modul.

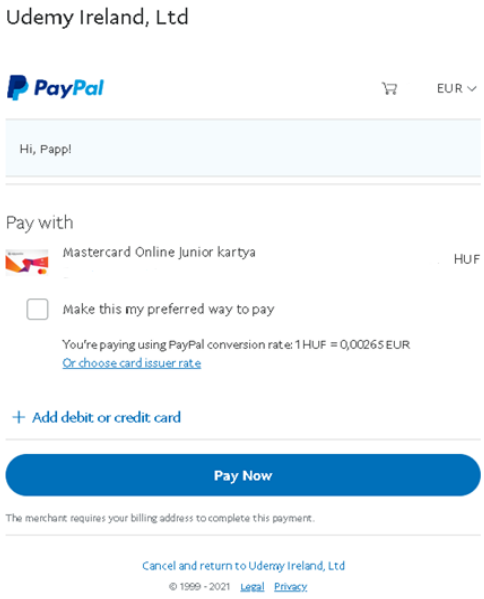
A vízesés modell következő lépése az integrációs és rendszerteszt. A legfontosabb különbség az előző lépéshez képest, hogy míg az implementációs és egységtesztek csak egy-egy kódrész működésére, vagy egy modul működésére fókuszálnak, addig az integrációs és rendszertesztek az egész alkalmazás működését vizsgálják, beleértve minden - az alkalmazáshoz köthető - modul működésével együtt. Úgy is fogalmazhatunk, hogy implementációs és egységteszteket legtöbb esetben az a fejlesztő végzi, aki azt az adott

kódrészt készítette, integrációs és rendszerteszteket viszont ideális esetben egy tesztelő csapat végzi, annak érdekében, hogy a lehető legnagyobb mértékben tudják a forráskódot átvizsgálni. A vízesés modell utolsó lépése a működtetés és karbantartási lépés. Ez lényegében a projekt végső fázisát jelenti. Ebben a lépésben van a projektzárás is. [6]

A modell legnagyobb problémája, hogy rugalmatlan. Egy szoftverprojekt akár évekig is futhat, ez a hátrány ilyenkor érezhető leginkább. Sok esetben a projekt indításakor a megrendelő nincs tisztában azzal, hogy pontosan mit is szeretne az elkészíteni a kívánt alkalmazásban. Az igényei sokszor csak a fejlesztési folyamat közben, vagy a projekt egy még későbbi szakaszában tisztázódnak. Azonban vannak olyan szektorok, ahol a mai napig egy bevett szokás a vízesés modell alkalmazása. Itt gondolhatunk akár egy iskolai projektre vagy féléves feladatra, de ilyen például maga a szakdolgozat is. Az üzleti világban leginkább a bankszektorra jellemző ez a modell, hiszen itt nagyon pontos specifikáció szükséges az elkészíteni kívánt alkalmazásról már a projekt megtervezésekor.

3.6. Online fizetés

Az internet és a webáruházak elterjedésével az emberek fizetési szokási is nagy változáson mentek át. Megjelent az igény a hagyományos készpénzes fizetést helyettesítő fizetési metódusokra. A fizikai készpénz használata egyre csökken a hétköznapjainkban. Ennek egyik meghatározó oka, hogy az online fizetési módszerek sok esetben nagyobb sebességet, biztonságot és kényelmet jelentenek a tranzakcióban résztvevő felek számára. Az első online fizetést követően csupán pár év kellett, hogy megjelenjenek azok a vállalatok, amik kifejezetten az online fizetési rendszerekre szakosodtak. Az egyik első, valóban világszinten elterjedt vállalat volt a PayPal, amit 1998-ban alapítottak. A PayPal lehetőséget nyújt felhasználóinak, hogy pénzügyi tranzakciókat hajtsanak végre. Ehhez mindössze egy ingyenes fiókra van szükség, amiben el kell tárolni azt a bankkártyát, amit a tranzakció során használni szeretnénk. A szolgáltatás nagy biztonságot nyújt felhasználóinak (titkosítja a bank- vagy hitelkártyát) és lehetővé tesz sokféle egyéb biztonsági szolgáltatást is, mint például a kétfaktoros autentikáció. Online fizetés esetén az alkalmazás egy minimális tranzakciós díjat számít fel, a szolgáltatás igénybevételéért. Ha egy oldal támogatja a PayPal fizetés lehetőségét, akkor azt kiválaszthatjuk a fizetési metódus kiválasztásánál. A metódus jóváhagyása után az adott oldal átirányít minket a PayPal saját oldalára, ahol fiókunkba történő belépést követően egyből megjelenik a tranzakció, nekünk már csak a fizetésre kell kattintani. Ezzel a folyamattal úgy tudunk fizetni egy honlapon, hogy nem adjuk meg a bankkártya- és/vagy személyes adatainkat. A fizetés befejeztével emailben értesítést kapunk, amiben a fizetési adatok szerepelnek.



3.5. ábra: PayPal fizetési szolgáltatás

3.7. Összegzés

A mai ételrendeléssel foglalkozó platformok központi szerepet töltenek be sok ember hétköznapjaiban világszerte. Ezek az alkalmazások sok esetben csak egy kisebb területen belül megtalálható éttermeket és boltokat foglalják magukba, esetleg egy külön, csak egy áruházláncnak szólnak. A legtöbb platform nem rendelkezik olyan funkciókkal, amik egy kiskereskedő vagy őstermelő számára előnyösek lennének, mint például a teljesen online számlázás, az előrendelés lehetősége, valamint a az áru reklámozása saját - a platformon belül található - oldalon. Mivel a feltételek az ilyen élelmiszeripari szereplőknek nem kedvezőek, ezért ők részben vagy teljesen kiszorulnak az online értékesítés világából. Sok ételrendelő szolgáltatás weboldala elavult technológiákkal készült. Így sok esetben a felhasználók számára is felmerülhetnek biztonsági kérdések, az esetleges rossz felhasználói élmények mellett. Az általam elkészíteni kívánt rendszer ezeket a hiányosságokat fogja kiküszöbölni, modern technológiák alkalmazása mellett.

4. SPECIFIKÁCIÓ

Szakdolgozatom választott témája, egy kiskereskedőknek és termelőknek szóló platform létrehozása. Az alkalmazás célja, hogy az itt létrehozott fiókkal rendelkező kereskedők értékesíteni és népszerűsíteni tudják saját termékeiket egy sokkal nagyobb körben, mint azt ezelőtt tehették.

Az elkészítendő platformot webes alkalmazásként kell kialakítani. Egy webes alkalmazás mindenki számára elérhető tud lenni, nem igényel mást csak egy eszközt, ami képes egy böngésző futtatására. Ezzel a lépéssel nem zárjuk ki azokat a mobilos felhasználókat sem, akik nem tudnák használni az alkalmazást abban az esetben, ha az csak Androidra, vagy csak IOS-re készülne el.

A terméknek szükséges megfelelnie a hatályos GDPR törvényeknek. Mivel az alkalmazásban a regisztrációhoz személyes adatok is szükségesek, mind a normál felhasználók mind a kereskedők részéről, így fontos, hogy ezeket az adatokat nagy odafigyeléssel és körültekintően tároljuk.

Mivel egy olyan webalkalmazásról van szó, ahol van lehetőség termékeket rendelni, így fontos, hogy az alkalmazás magába foglalja az online fizetés lehetőségét, illetve a számlázás funkcióját is. Egy kereskedő saját oldalt hozhat létre üzlete és termékei számára. Ezen az oldalon csak az ő termékei lesznek megtalálhatók, illetve minden egyéb médiatartalom, amit meg szeretne osztani az idelátogató potenciális vásárlókkal. A termékeket saját maga árazhatja be, nem lesznek előre meghatározott árak bizonyos termékekre. Ezekhez készülhet leírás, amiben megfogalmazhatja, hogy terméke miben különleges, és mit érdemes róla tudni.

Az alkalmazásban a vásárlók és a kereskedők két, teljesen különböző entitásként fognak megjelenni, megfelelően elkülönítve, különösen a jogosultságokat tekintve. A kereskedőknek többletjogosultsága lesz a felhasználókhoz képest, de a rendszerben lesznek adminisztrátor jogosultságú entitások is, ők viszont felhasználóként kerülnek majd be a rendszerbe, azzal a különbséggel, hogy plusz jogosultságok lesznek hozzájuk rendelve. Az alkalmazás megvalósítása során szétválasztásra kerül a felhasználói felület és a szerveroldal. A két fél közötti kommunikáció GraphQL segítségével kerül majd megvalósításra. Az ilyen megvalósításra szánt üzleti logika is implementálásra kerül, hogy az alkalmazás a lehető legnagyobb színvonalat tudja képviselni, ami egy webalkalamzástól elvárható.

4.1. Funkciólista

Az elkészíteni kívánt rendszerünknek az alábbi funkciókkal kell rendelkeznie:

felhasználók számára:

- fiók létrehozása, bejelentkezés
- termékek kosárban való módosítása
- termékek megrendelése
- rendelési előzmények áttekintése
- termékek szűrése

kereskedők számára:

- fiók létrehozása kereskedőként, bejelentkezés
- saját oldal készítése, testre szabása
- árucikkek meghirdetése, feltöltése, szerkesztése
- bejegyzések létrehozása, kezelése
- megrendelések kezelése
- integrált számlázó rendszer
- integrált fizetési rendszer

4.2. Technológiaválasztás

4.2.1. MYSQL

A MySQL az egyik legelterjedtebb relációs adatbázis. Nyílt forráskódú adatbáziskezelő-rendszer. Az adatokat egy vagy több adattáblába rendezi, ezek az adatok összefügghetnek egymással. A MySQL lehetővé teszi a jól skálázható, strukturált adattárolást. Jól áttekinthető adatstruktúrát készíthetünk benne, így a szerveroldali alkalmazás könnyedén tud ezekkel az adatokkal dolgozni. A MySQL támogatja a Language-Integrated Query (LINQ) alkalmazását. A LINQ lehetővé teszi, hogy egységesen kérdezzünk le adatokat az adatbázisból. A LINQ az általam használni kívánt Asp.NET-tel jól alkalmazható, de feltételei vannak. Ahhoz, hogy alkalmazzuk adatforrásokon, szükséges, hogy ezek az adatforrások megvalósítsák az IEnumerable<T> vagy az IQueryable<T> inteface-t. A lekérdezése eredményét később átalakíthatjuk tömbökké vagy épp listákká, ha szükséges.

4.2.2. .NET 6

A .NET 6 a szoftverfejlesztés egy új, a mai trendeknek és technológiáinak megfelelő alkalmazásának lehetősége. A .NET 6 nagy változásokat hozott a rajta alapuló technológiák, így az Asp.NET fejlesztés tekintetében is. Az új verzió egyik legszembetűnőbb újítása, a korábbi verziókhoz képest, a teljesítménynövekedés. A .NET 6 az első olyan kiadás, ami natívan támogatja az Apple Silicon-t(Arm64), valamint továbbfejlesztésre került benne a Windows Arm64 is. Megtalálható benne egy új, dinamikus profil vezérelt optimalizálási rendszer (PGO), ami mélyreható optimalizálhatóságot tesz lehetővé a fejlesztők számára. A .NET 6 magával hozott új API-kat, többek közt http/3-hoz és JSON fájlok feldolgozásához, valamint a memória közvetlen manipulálásához. Az új .NET verzióval megjelent a c# 10 is. A c# 10 egyik legfontosabb ismérve, hogy folytatja azt az utat, amit a c# 9 elkezdett, ez pedig az egyszerűsítés, a kódot tekintve. A korábbi c# verziókban bevett volt, hogy a legegyszerűbb alkalmazásaink is sok sorból álltak. Ez a c# 10-zel megváltozott, akár 1 soros programjaink is lehetnek, amik hatékonyabban képesek lefutni, mint elődei. Ilyen újítások a globális direktívák használata és a fájl hatókörű névterek használata is. Az ilyen direktívákkal például elég egy alkalommal megadni egy direktívát, majd a programunk bármelyik fájljában alkalmazhatók lesznek. [7]

4.2.3. Next.js

A Next.js egy Node.js-re épülő, nyílt forráskódú fejlesztői keretrendszer. React.js alapú webalkalmazások funkcióit teszi elérhetővé, például szerveroldali megjelenítés és statikus webhelyek generálása. A Next nagyon sok funkciót alapból implementál, amiket React-ben csak különböző csomagok alkalmazásával lehetne kivitelezni. A React minden tartalmat a kliensoldalon „épít fel”, a Next viszont lehetővé teszi a szerveroldali rendelést. A keretrendszer támogatja a CSS-t, valamint az előre lefordított Scss-t és Sass-t, valamint a JSX-et. Ezenkívül TypeScript támogatással is rendelkezik. A Next.js fő jellemzője a szerveroldali renderelés használata, a böngészők terheltségének csökkentése és a fokozott biztonság érdekében. Ez akár az egész alkalmazásra használható. A "hot reloading" funkció figyeli a változtatásokat, és amint szükséges újra rendereli a megfelelő oldalakat, így a szervernek elkerülhető az újraindítás. Ez lehetővé teszi, hogy az alkalmazás kódjában végrehajtott módosítások azonnal megjelenjenek az alkalmazásban. [8]

5. TERVEZÉS

5.1. Az adatbázis kialakítása

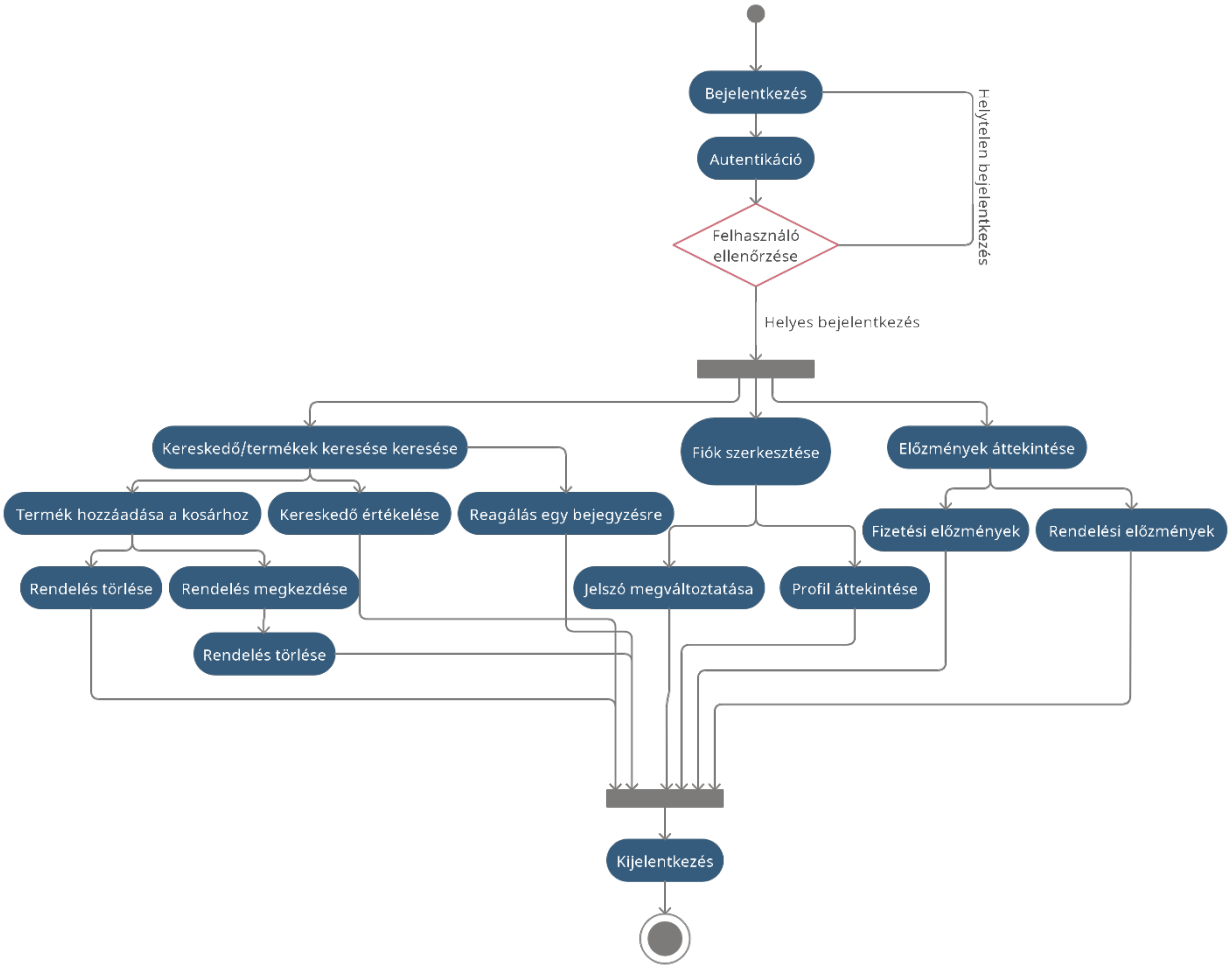
Az elkészíteni kívánt platform egyetlen adatbázist fog használni. Ebben kerülnek majd eltárolásra a termékek, a megrendelések, a kereskedők, a kereskedők oldalon megosztott tartalmai és a felhasználói adatok. A táblák minden esetben először egy „id” mezőt kaptak, ez fogja jelenteni minden tábla elsődleges kulcsát. Ha esetleg egy táblában előfordul idegen kulcs, akkor azok „(főtábla)ID” mezőnévvel kerülnek eltárolásra. Az alkalmazásban kiemelten fontos szempont lesz a biztonság, ezért a jelszavak eltárolására különös figyelmet fordítok. Ezek tárolására több megoldás is szóba jöhet.

A jelszavakat egy az egyben tárolni egy adatbázisban nagyon veszélyes mivel egy apró adatszivárgás vagy kibertámadás alkalmával is jelszavak százai kerülhetnek ki a nagyvilágba. Ezért különböző titkosítási eljárásokat szokás alkalmazni. Ennek egyik formája az úgynevezett Hash táblák használata. Ebben az esetben egy értéket generálunk a szövegből egy matematikai függvény segítségével. Ez azonban csak bizonyos mértékig csökkenti a veszélyfaktorát a jelszavak tárolásának. Ezért alkalmazásom a Hash mellett sózást is alkalmazni fog a jelszavak védelmére. Ezek együttesen HMAC segítségével fogják a jelszót titkosítani. A HMAC egy specifikus üzenet-hitelesítési kód. Ez az eljárás a hash funkciót és egy titkos kriptográfiai kulcsot foglal magában. A kulcs a mi esetünkben a sózás folyamán fog képződni. A HMAC aszimmetrikus titkosítási eljárás. Ez .NET-ben HMACSHA512 osztályként található meg. A HMACSHA512 egy kulcsos hash algoritmus, amely az SHA-512 hash függvényből képződik és hash-alapú üzenethitelesítési kódként használt. A HMAC egy folyamat, amely titkos kulcsot hoz létre az üzenetadatokkal kimenetként. Ezután a hash értékét ismét összekeveri a titkos kulccsal, majd másodszor is létrehoz egy kimenetet. A kimeneti hash végeredményében egy 512 bit hosszú hash.

5.1. ábra: Az elkészítendő alkalmazás adatbázis terve

5.2. Tevékenységdiagram

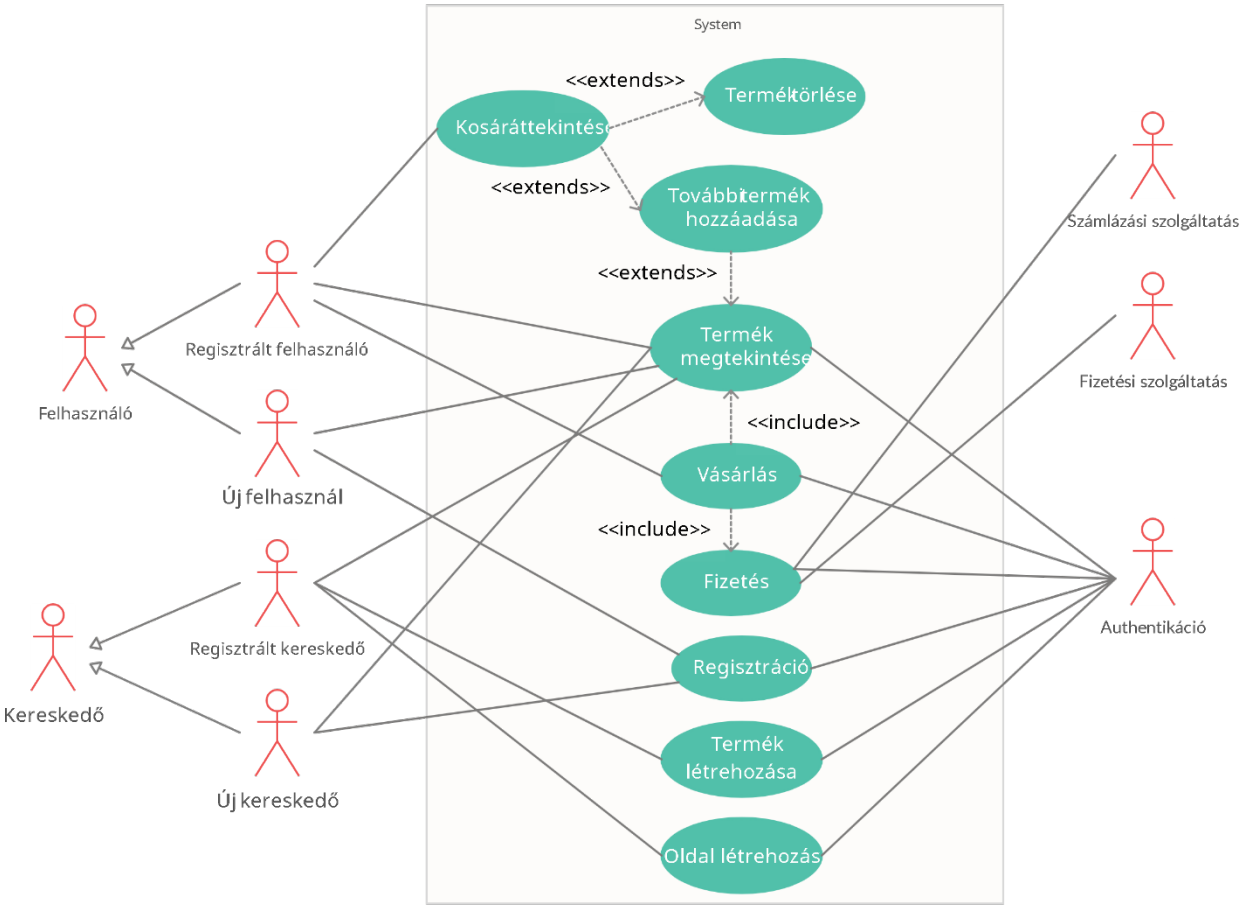
Az elkészíteni kívánt alkalmazásban egy felhasználó számos tevékenységeket végezhet majd. Első lépésként szükséges egy autentikáció, amit bejelentkezéssel tehet majd meg. Ha a felhasználó ellenőrzése sikeres volt, akkor átirányítjuk a főoldalra, ahol választhat, mivel szeretné a tevékenységét folytatni. Van lehetősége termékeket vagy kereskedőt keresni. Ebben az esetben elkezdheti a rendelés folyamatát vagy reagálhat egy - a keresett entitáshoz köthető - bejegyzésre. Ha fiókját szeretné szerkeszteni, akkor van lehetősége jelszót módosítani vagy egyszerűen csak áttekinteni a profilját. Ha előzményeit szeretné megtekinteni, akkor választhat, hogy a fizetési előzményeire vagy a rendelési előzményeire kíváncsi. Tevékenysége azzal fog véget érni, hogy elhagyja a weboldalt. Ekkor történik meg a felhasználó kijelentkeztetése.



5.2. ábra: Az elkészítendő alkalmazás egy tevékenységdiagramja

5.3. Használatieset-diagram (USE CASE DIAGRAM)

Az alábbi használatieset-diagram mutatja be a létrehozni kívánt rendszerrel szemben támasztott követelményeket. A platform használói az egyszerű felhasználók és a kereskedők lesznek. Egy felhasználó lehet olyan, aki most először látogat az oldalra, illetve lehet már beregisztrált felhasználó is. A rendszernek ennek következtében el kell látni az autentikáció funkcióját, amely lehetőséget biztosít a felhasználói azonosításra, azok bejelentkezése és/vagy regisztrációja által. Az oldal felhasználói megtekinthetnek termékeket, amiket meg is rendelhetnek. Itt szintén szükséges lesz autentikáció. Ezen felül fontos, hogy fizetési és számlázási szolgáltatások is rendelkezésre álljanak a rendelések véghezviteléhez. A felhasználók lekérdezhetik a kosaruk tartalmát.



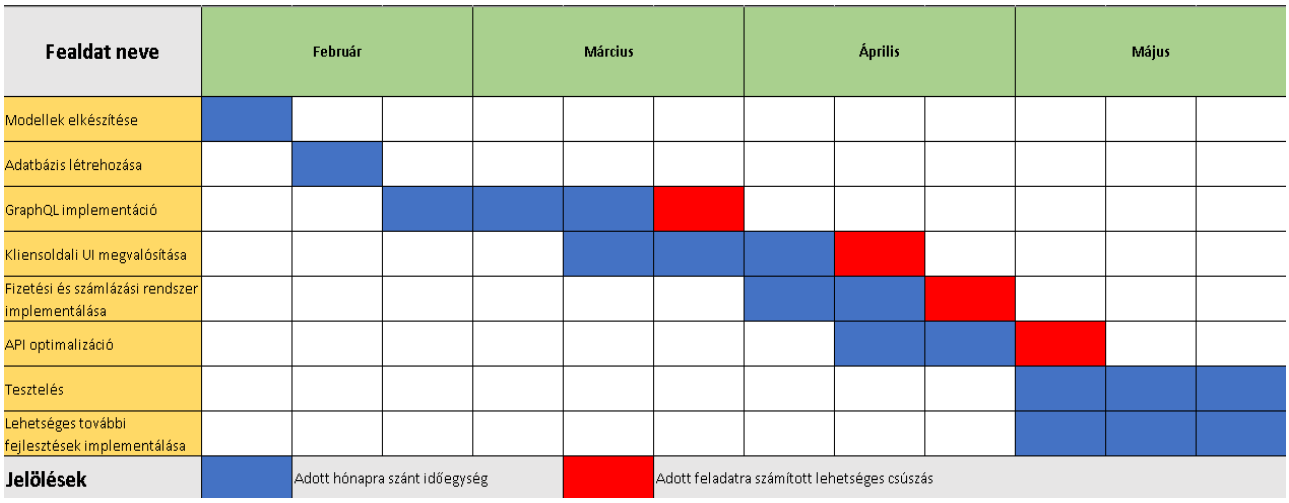
5.3. ábra: Az elkészítendő alkalmazás egy használatieset - diagramja

5.4. A fejlesztés tervezett menete

A fejlesztést, a körültekintő és alapos tervezési folyamatot követően, a szerveroldali alkalmazás elkészítésével fogom megkezdeni. Legelőször a projekt létrehozását követően néhány, a fejlesztés elkezdéséhez nélkülözhetetlen, NuGet csomag hozzáadásával fogom kezdeni. Ezek után kezdődhet a valódi fejlesztési folyamat, első körben a modellréteg elkészítésével. Ezek a modellek fogják reprezentálni az adatbázisom tábláit. Ebből következik, hogy a MySQL adatbázis tábláit nem előre hozom létre, hanem a szerverhez NuGet csomagként hozzáadott Entity keretrendszerre bízom azt. Fontos, hogy a modellek pontosan reprezentálják már a kezdetekben a táblákat, így csökkentve a megismételt migrációs folyamat számát. Természetesen a fejlesztés folyamán felmerülhetnek változások a modellekben. Ebben az esetben szükségesek migrációs lépések, hogy az adatbázis is konzisztens maradjon a szerveroldali alkalmazással. A modellek és az adatbázis létrehozása után következő lépésként implementálni kell a GraphQL-t. A GraphQL támogatás szintén NuGet csomagként kerül majd be a projektbe, HotChocolate keretrendszerrel lesz használva.

Miután a szerver elérte azt a szintet, hogy egy kliens képes legyen tőle adatokat lekérni, elkezdem a kliensoldali alkalmazás elkészítését. A szerver ekkor már képes lesz bejelentkezéseket és regisztrációkat fogadni, valamint termékek létrehozására is lehetőség lesz. A kliensoldali alkalmazást Next.js-ben fogom elkészíteni. Tervek szerint ebben a fejlesztési fázisban implementálni tudom majd a szükséges oldalakat, és már tesztelni is tudom majd a szerveroldali alkalmazással. A kliens ténylegesen csak a következő fázisban fog elkészülni, mikor a számlázási és fizetési rendszer is implementálásra kerül. Ezek a rendszerek elképzelhető, hogy csak kis mértékben vagy egyáltalán nem támogatják a GraphQL-t így ezekkel az API-kal a kliens HTTP kérésekkel fog kommunikálni. Ha szükséges, a következő fázisban optimalizálom a már korábban elkészített szerveroldali alkalmazást.

A fejlesztés egyik utolsó, ámde egyik legfontosabb fázisa a tesztelés. Folyamatosan fognak egységtesztek készülni a fejlesztés során, azonban vannak olyan tesztek, amelyek csak az alkalmazás fejlesztésének végső fázisában végezhetők el. Emellett, plusz funkciókat fogok a projektbe implementálni, amikre való igény esetleg felmerül a fejlesztés folyamán. Egy szoftverprojektben mindig számolni kell azzal, hogy az egyes fázisok nem készülnek el időre, így az időbe belekalkuláltam az esetleges csúszásokat, ezeket piros színnel jelzem.



5.4. ábra: A fejlesztés tervezett menete

6. MEGVALÓSÍTÁS

A fejezetben az alkalmazás tervezését követő implementációs lépéseket ismertetem. Több helyen eszközöltem javításokat, módosításokat. Ezek fő oka, hogy a korábbinál jobb vagy korszerűbb megoldásokat találtam. A fejlesztés során mind a szerveroldal, mind pedig a kliensoldal nagy szerepet kapott. Ezek mellett a fizetési rendszerrel való kommunikációra fektettem a legnagyobb hangsúlyt, hogy az elvártaknak megfelelő végeredmény kaphassak.

6.1. Az elkészült rendszer felépítése

A rendszer fejlesztése során mind a kliensoldali, mind pedig a szerveroldali alkalmazást úgy készítettem el, hogy megfeleljen egy modern alkalmazással szemben támasztott elvárásoknak. Ennek megfelelően nagy szerepet kapott a modularizáltság és a tesztelés egyaránt. Az általam választott kliensoldali technológia nagyban könnyítette ennek kivitelezését. Az alkalmazást itt különböző komponensekre bontottam és azokból állítottam össze az egyes oldalakat. Ez, többek közt, lehetővé tette a komponensek újrafelhasználását. Több helyen volt lehetőségem kihasználni a keretrendszer adta lehetőségeket. Ez nagyban hozzájárult a felhasználói élményhez.

Kliensoldalon TypeScriptet alkalmaztam a hagyományos JavaScripten felül. Ez lehetőséget biztosított számomra arra, hogy sok, csak futásidőben előkerülő, hibát kiküszöböljek, illetve kliensoldalon is létre tudtam hozni az egyes entitások modelljeit, amiket később az egyes komponensekben típusként tudtam felhasználni.

A szerveroldali alkalmazás elkészítése során ügyeltem arra, hogy az egyes rétegek jól elkülöníthetők legyenek egymástól. A GraphQL egyik sajátossága, hogy a kliens egyetlen végponton keresztül kommunikál a szerverrel. Ezért külön végpontokat nem hoztam létre, helyette jól definiált metódusokat, illetve osztályokat alkalmaztam, hogy a kód ennek ellenére is jól áttekinthető legyen. A szerveroldali alkalmazást rétegekre bontottam, így külön réteg gondoskodik a GraphQL függőségeiről, külön választottam a különböző szolgáltatások rétegét, illetve a mapparéteget is, ezek mellett a különböző entitások modelljei is külön réteget kaptak a szerveroldali alkalmazásban.

6.1.1. Adatbázis-módosítások

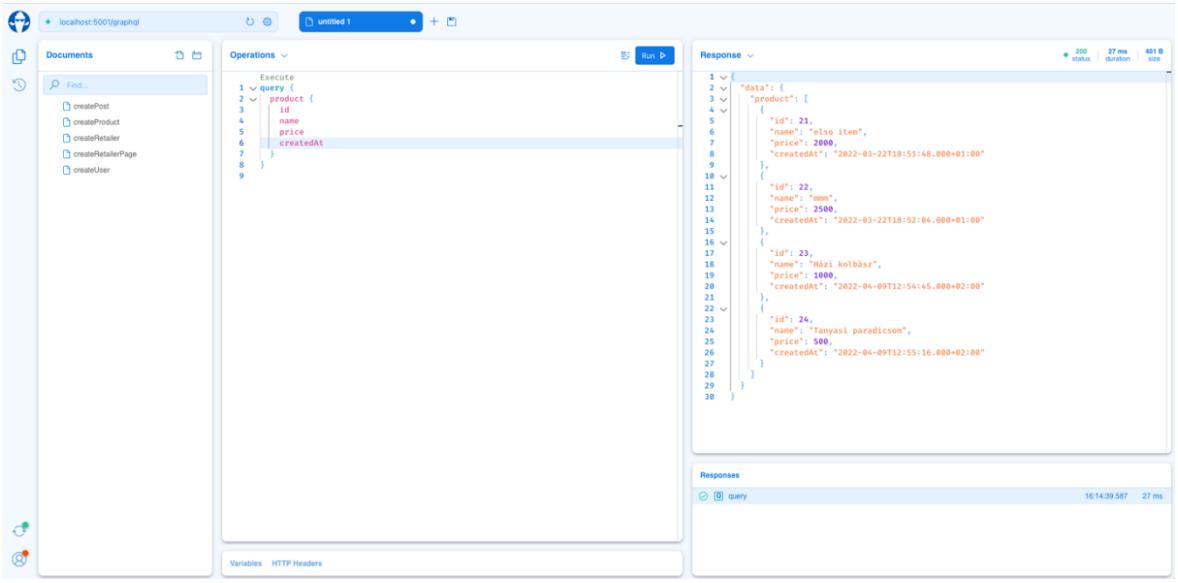
A szakdolgozatomban elkészített alkalmazás egy relációs adatbázist, a MySQL-t használja az adatok tárolására. A fejlesztés során az alapvető adatbázis-kapcsolatokon nem módosítottam, csak néhány kiegészítést hajtottam végre az egyes táblákban, hogy az alkalmazás egyes funkcióinak elkészítését könnyebbé tegyem. Ugyanakkor egy táblával kiegészítettem az adatbázist a fejlesztés során. A tervezés során nem látott okokból az eredeti adatbázis nem lett volna képes a rendelési adatokat megfelelően tárolni. Ez abban volt tetten érhető, hogy nem álltak rendelkezésre a megfelelő adatok. A rendelés végén lehetett volna tudni, hogy ki, kinek és miről adta le a rendelést, ugyanakkor a vevő pontos adatait a regisztrációkor nem kellett, hogy megadja, így a kereskedő nem tudta volna hova szállítsa az adott terméket. Ezért létrehoztam egy külön táblát, amiben a vevő aktuális adatait tárolom a rendelésről.

6.1. ábra: A végleges adatbázis modellje

6.1.2. Szerveroldal

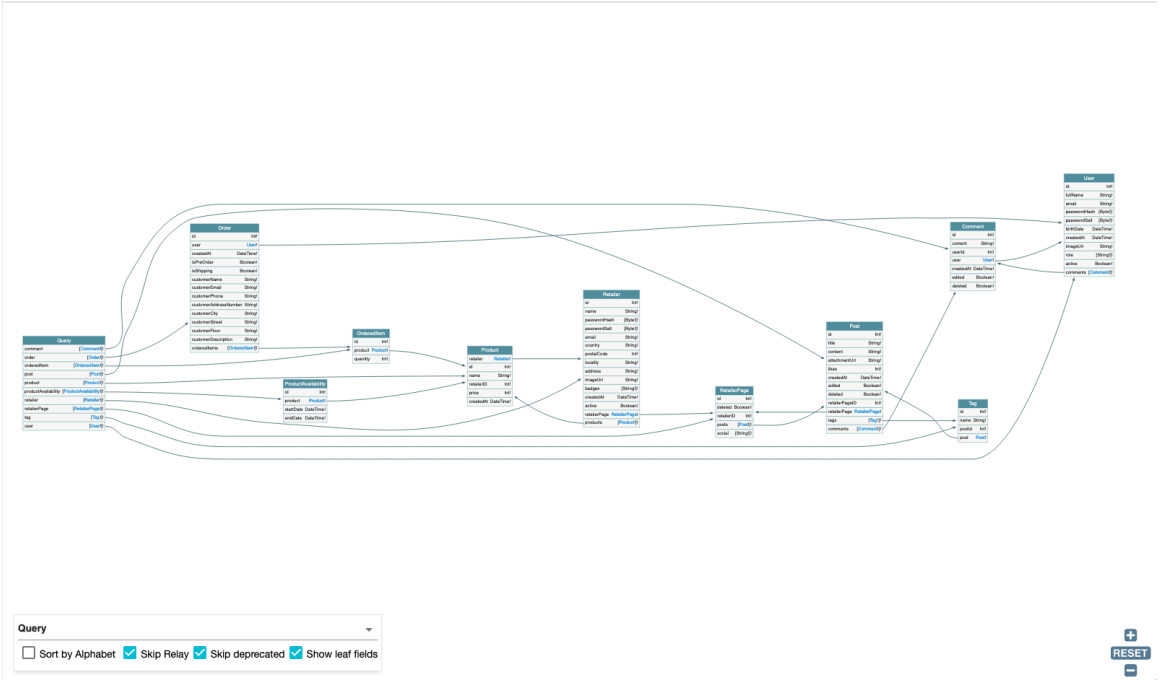
A szerveroldali alkalmazás Microsoft ASP.NET keretrendszerrel készült .NET6 alapokon. A szerveroldali alkalmazás elkészítése alatt a rétegzett felépítésre törekedtem. Ennek fő szempontjai közé tartozott, hogy ezek a rétegek jól elkülöníthetők legyenek, illetve, hogy az egyes rétegek csak a szomszédaikkal kommunikáljanak. A rétegeket sikerült megfelelően kialakítanom, viszont a GraphQL esetében helyenként el kellett ettől térnem. Egy alkalmazást Code first illetve Database first koncepció alapján lehet elkészíteni, én a Code first megvalósítást választottam. Ennek fő oka az volt, hogy a .NET keretrendszer lehetővé teszi az Entity Framework használatát. Ez pedig lehetőséget biztosít az adatbázissal való kommunikációra. A szerveroldalon így definiálhatunk modelleket a modell rétegben, ezek objektumorientált osztályok, amikből az Entity Framework létre tudja hozni a különböző adattáblákat. Ez felelős továbbá a CRUD műveletek elvégzéséért az adatbázisban. A CRUD szó a Hozzáadás-Olvasás-Módosítás-Törlés angol megfelelőinek rövidítése. A mapparéteg felett - az alkalmazás egyik leghangsúlyosabb része - az üzleti logikai réteg található. Ez a réteg felelős, többek közt, minden számítást igénylő művelet elvégzéséért, illetve a kliensoldali alkalmazás is vele kommunikál. [9]

A .NET és a köré szerveződött infrastruktúra több lehetőséget is kínál a GraphQL implementációjára. A szerveroldali fejlesztés elkezdése előtt alaposan megvizsgáltam a legnépszerűbbnek tartott implementáláshoz használt könyvtárakat. Ezek közül a legkézenfekvőbb megoldásnak a HotChocolate könyvtár bizonyult. A Hot Chocolate nem csak különféle függvények gyűjteményét hozza magával, de miután hozzácsatoltuk a projektünkhöz, lehetőséget ad egy saját végpont implementálására, ami a szerverrel egy URL címen érhető el annak egy saját végpontján. Ez egy saját GraphQL platform, ami a könyvtáron belül található, fő felhasználási módja a már elkészült kód manuális tesztelhetőségének segítése. Mivel GraphQL használata mellett elég egyetlen kapcsolódási pont a kliens és a szerver között, így a végpontok cserélgetésével nem kell törődnünk. A felületen három fő szekció található. A középsőbe írhatjuk a különféle GraphQL lekérdezéseinket. A szintaktika betartására folyamatosan figyelmeztet a felület, ha bárhol hibát vétenénk. A harmadik oszlopban pedig a szerver válaszát láthatjuk az elküldött lekérdezésre. Ha szeretnénk, akkor a különféle lekérdezéseinket saját névvel ellátva elmenthetjük. Ez látható az első oszlopban.



6.2. ábra: HotChocolate keretrendszer GraphQL API teszteléshez használható végpontja

A projektben megtalálható még egy kifejezetten fejlesztést segítő csomag is, a GraphQL Voyager. Ez a csomag segít vizualizálni az általunk felépített GraphQL sémát. Segítségével láthatjuk, hogyan épülnek fel az egyes adattáblák és köztük milyen kapcsolat van, továbbá kiváló dokumentációs lehetőségeket nyújt a fejlesztőknek.

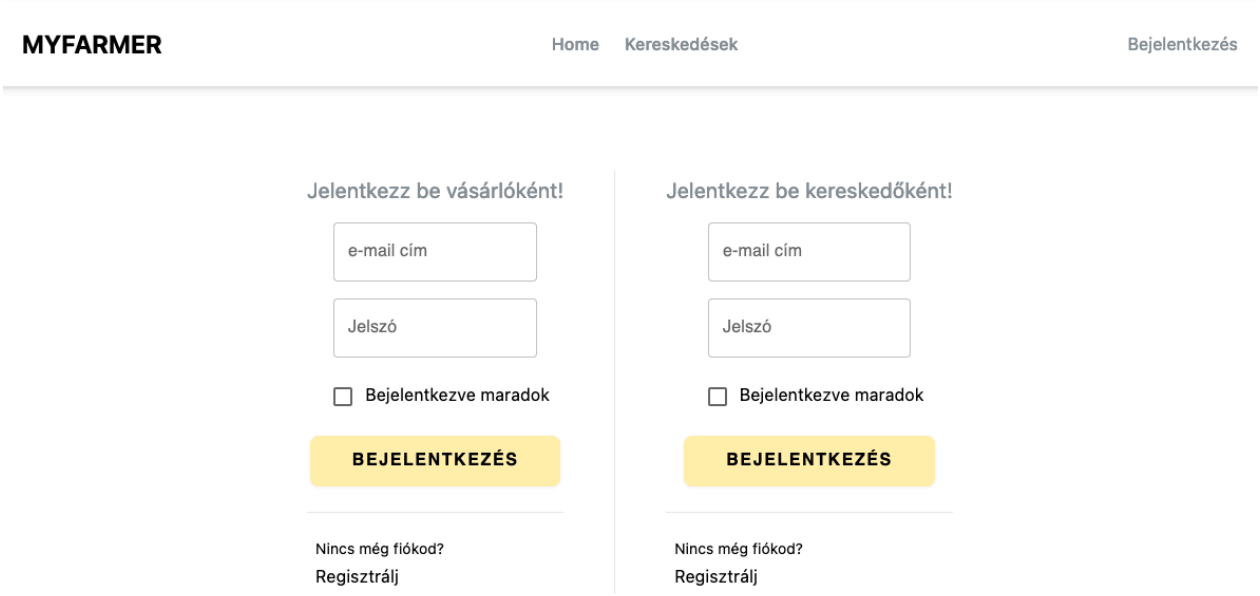


6.3. ábra: GraphQL Voyager könyvtár GraphQL API vizualizációs végpontja

Általánosságban a GraphQL nyelv két fő kommunikációs elemből áll: lekérdezésekből (query) és mutációkból (mutation). A lekérdezések segítségével kérhetünk le az adatbázisból adatokat. Itt emelkedik ki a lekérdező nyelv egyik meghatározó előnye. Pontosan azokat az adatokat fogjuk megkapni a folyamat végén, amelyeket szeretnénk, se többet se kevesebbet. Ezzel segíthetjük azt, hogy a szervernek ne kelljen felesleges adatbázis-műveleteket végrehajtania és a kliensnek se kelljen felesleges adatokkal dolgoznia. A lekérdezések kliensoldalon bármikor kiegészíthetők, viszont a szerveren semmilyen módosítást nem kell végrehajtanunk, ha megfelelő hibakezelést végeztünk. A mutációk segítségével végezhetjük el a különféle adatmódosításokat. Létrehozhatunk, módosíthatunk és törölhetünk is adatot. Véleményem szerint összességében jelentősen kevesebb kódsorból elkészíthető ugyanaz a szerveroldali alkalmazás GraphQL segítségével, mint hagyományos REST architektúrával, továbbá a kliens esetleges változásait is sokkal kevesebb munkával lehet lekövetni.

6.1.3. Kliensoldal

A rendszer elkészítésére a technológiaválasztás indoklásánál leírt indokok alapján webes alkalmazást választottam. Next.js segítségével készítettem el a klienst, amivel a felhasználó egy böngészőn keresztül tud interakcióba lépni. Funkcionalitásában nagyon kiterjedt. A felhasználóknak van lehetőségük termékek vagy kereskedők után böngészni. Termékek esetében van lehetőség vásárlásra, kereskedők megtekintése esetén, pedig azok aktivitásának megtekintésére. Egyszerű böngészésre bejelentkezés vagy regisztráció nélkül is lehetőség van, azonban, ha vásárolni vagy hozzászólást szeretne írni a felhasználó, az már felhasználói fiók nélkül nem lehetséges. Mind kliens-, mind szerveroldalon a vásárlókat, illetve a kereskedőket különállóan kezeltem. Már a bejelentkezés folyamán is külön rész található mindkét entitás autentikációjára. Igaz, mindkét esetben a bejelentkezéshez ugyanazok az adatok szükségesek, de szerveroldalra érve már a logika egy más része foglalkozik velük.



6.3. ábra: Kliensoldali alkalmazás bejelentkezési felülete

A regisztráció folyamán a vásárlóknak és a kereskedőknek különböző adatokat kell megadni. Sikeres regisztráció esetében a vásárlót is és a kereskedőt is a nyitóoldalra irányítja az alkalmazás. Kereskedők ekkor már eljuthatnak a saját oldalukra. Első látogatáskor az alkalmazás létrehoz egy új oldalt a kereskedő számára, ezek után már mindig erre az oldalra fog eljutni a kereskedő az idevaló kattintás után. Itt a kereskedő elkezdheti saját adataival megtölteni a mezőket. Létrehozhat postokat, leírást adhat a boltja számára, valamint feltöltheti termékeit. A vásárló sikeres bejelentkezés vagy regisztráció után elkezdhet

böngészni, van lehetősége rendeléseket leadni, amiket a leadáskor kell kifizetnie, valamint írhat hozzászólásokat a kereskedők által írt postokhoz.

Véleményem szerint ez egy nagyon jó módja a kereskedő értékelésének, a hagyományosnak mondható módszerektől - mint például az öt csillag alapján történő értékelés - eltérő, viszont egy sokkal átfogóbb és őszintébb képet kaphatunk a kereskedőről. Véleményem indoklása az, hogy a hasonló oldalak, amikből többet megvizsgáltam, számontartják az összesített értékelést a korábban említett öt- vagy tízcsillagos rendszerben, és több közülük az értékelések számát is tárolja. Viszont egy fontos információ ekkor nem jelenik meg, mégpedig az, hogy a kereskedő mióta van jelen az oldalon. Ez azért fontos, mert ha ez az idő még nem mérhető csak például hetekben, az egy ok lehet arra, amiért a kereskedő még csak kevés értékelést kapott. Ha ezen értékelések nem is oszlanak meg igazán, akkor is lehet, hogy azok átlaga egy sokkal másabb eredménnyel szolgál, mintha megfelelő számú értékelésünk lenne. Hozzászólások esetében viszont már kevesebb darabszám esetén is egy valósabb képet kaphatunk a kereskedő által nyújtott szolgáltatásokról és kínált termékekről.

6.1.4. Fizetési szolgáltatás integrációja

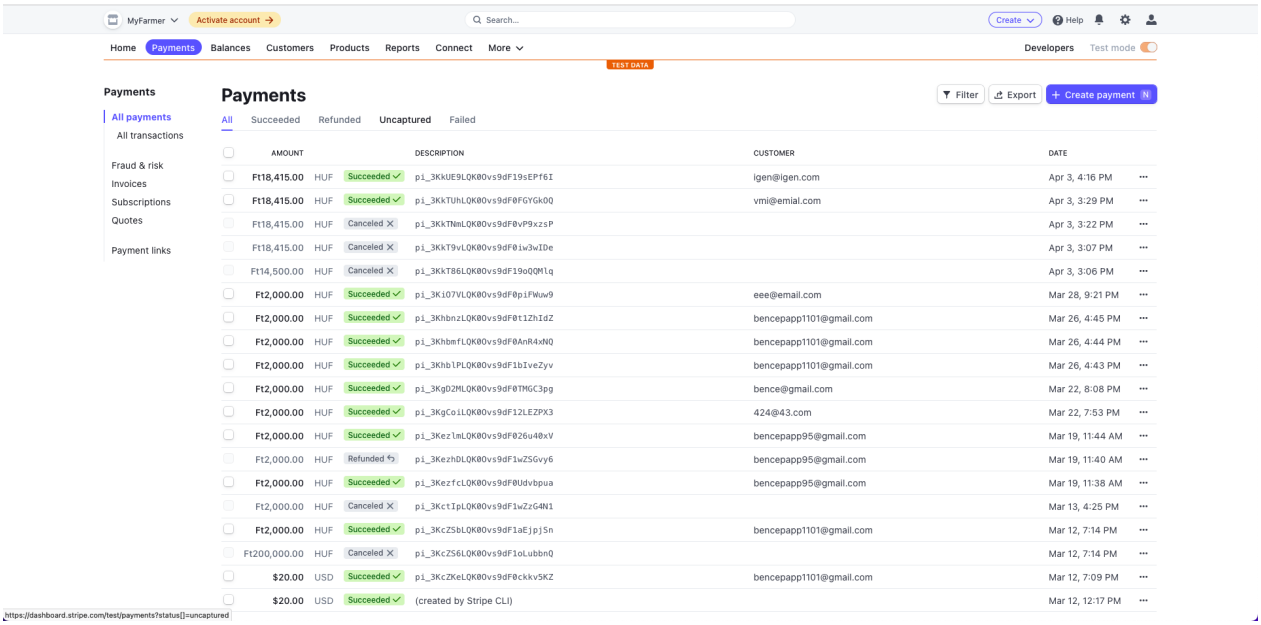
A szakdolgozatomban elkészített webalkalmazás egyik kulcsfunkciója a fizetési szolgáltatás. Általánosságban elmondható, hogy egy modern webalkalmazás, amely valamilyen szolgáltatást kínál, és ezt a szolgáltatást pénzösszeghez köti, abban az esetben megtalálható benne legalább egyféle fizetési szolgáltatás integráció (talán legelterjedtebb megvalósítások a bankkártyás fizetés netbankon keresztül, illetve a PayPal-on keresztül történő fizetés). Ennek fő oka, hogy így a felhasználó számára a fizetést követő néhány pillanat múlva rendelkezésre tud állni a szolgáltatás, illetve a szolgáltató is megkaphatja vele egy időben az összeget.

Az ilyen fizetési platformok általában három fő komponensből épülnek fel. Az első a kereskedői számla. Ez egy bankszámla, amely lehetővé teszi az online fizetések feldolgozását internetes vállalkozások számára. Kereskedelmi számlát fizetésfeldolgozó cégen, független vállalkozón vagy bankon keresztül kaphat. A számla nélkül nem lenne hol tartania az ügyfelek által fizetett pénzt.

Fizetésfeldolgozó. Egy fizetésfeldolgozó cég vagy pénzintézet kezeli az ügyfelek bankja és a cég bankja közötti tranzakciókat. Olyan kérdésekkel foglalkoznak, mint a hitelkártya érvényessége, a rendelkezésre álló források, a kártyalimitek stb. A fizetésfeldolgozó másik lényeges funkciója a biztonság. A fizetésfeldolgozó felelőssége ellenőrizni, hogy a kártyaadatok helyesek-e, illetve képesek legyenek megvédeni a feleket a csaló tevékenységektől. Különféle hibákról, véletlen tranzakciókról és hibás kártyaterhelésekről is gondoskodnak.

Fizetési átjáró. Ez olyan, mint egy online értékesítési pont. A fizetési átjáró egy közvetítő a webhelyen végrehajtott tranzakció és a fizetésfeldolgozó között. Ez kapcsolja össze a kereskedő vagy a cég fiókját olyan hitel- és betéti kártyát kibocsátókkal, mint például a Mastercard vagy a Visa. Fizetési átjáróra minden esetben szükség van, mivel a biztonsági intézkedések tiltják az adatok bankról bankra történő közvetlen átvitelét. A fizetési átjáró, tehát elengedhetetlen egy internetes fizetésfeldolgozó rendszerhez.

Az általam elkészített alkalmazás a Stripe rendszerét használja az online fizetések megvalósításához. A Stripe kiválóan integrálható bármely webes vagy mobilos alkalmazáshoz, amik a támogatott programozási nyelvek valamelyikét használják. Szakdolgozatomban a szolgáltatást teszt módban használtam, ez sok mindenben megkötést jelentett, ugyanakkor a szolgáltatás csak ezen feltételek mellett vehető igénybe ingyenesen.
A fizetési folyamat a rendelés leadásának utolsó fázisa. Ekkor a vásárlót egy Stripe által létrehozott oldalra irányítja az alkalmazás, ahol bankkártyával van lehetősége fizetni. Ehhez a kártyaadatok mellett meg kell adnia egy email címet is. Többek közt, ezzel is lesz később visszakövethető a fizetés. Sikeres fizetés esetén a nyitó oldalra irányítjuk a felhasználót, sikertelen fizetés esetén, pedig egy direkt erre a célra létrehozott oldalra. Fizetést követően a megrendelő emailt kap a leadott rendeléséről, a kereskedő, akinek a termékére a rendelést leadták, pedig a rendelések oldalán láthatja a változásokat. A kereskedő itt részletesen láthatja a rendelés adatait, ami alapján tudja hova és mennyi terméket kell szállítania. Mivel a Stripe-ot csak teszt módban tudtam ingyenesen használni, így sajnos nem volt lehetőségem azt a funkciót elkészíteni, amiben a kereskedő egyből megkapja a pénzét sikeres megrendelés esetén.



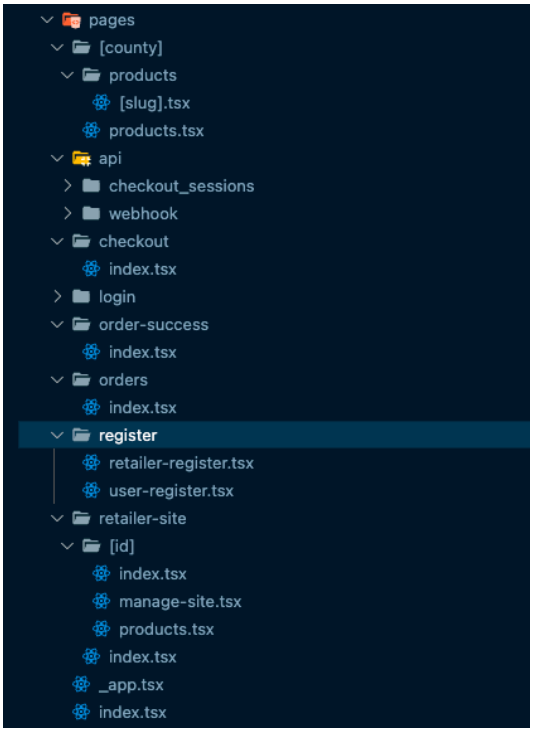
6.5. ábra: Stripe fizetés áttekintő felülete

A fizetési és emailküldő szolgáltatások implementációjánál több módot is megvizsgáltam arra, hogy hogyan lehetne ezeket a rendszereket legjobban a projektemben felhasználni. A Stripe esetében van lehetőség több nyelvben is a szolgáltatás használatára. Alapesetben nem elég csak kliensoldalon implementálni, szükség van a szerveroldalra is. Az általam elkészített alkalmazás szerveroldali része egy .NET6 alkalmazás, a Stripe ezt is támogatja, viszont az implementációhoz sokkal több munkára van szükség, mint a Node JS esetében. Ezért végül a Node-os megvalósítás mellett döntöttem. Az alkalmazásom kliensoldala Next.js keretrendszert használ. Ennek sok előnye közül kiemelkedő, hogy egy Next.js projekt létrehozásakor nem csak egy kliens, de egy szerver is létrejön. Ennek magyarázatára szakdolgozatomban egy külön részt hoztam létre, ahol bemutatom ennek a pontos működését. Jelen esetben viszont ez azt jelenti, hogy a Node JS szerveroldali alkalmazás is adott, csak ki kell használni annak lehetőségeit.

6.1.5. Egyoldalas Alkalmazás (SPA)

Az SPA (Single Page Application) egy webalkalmazás megvalósítás, amely csak egyetlen webdokumentumot tölt be, majd frissíti az adott dokumentum tartalmát különféle JavaScript API-kon keresztül. Így a felhasználók anélkül használhatják a webhelyet, hogy teljesen új oldalakat töltenének be a szerverről, amely többek között teljesítménynövekedést és egy sokkal dinamikusabb felhasználói élményt eredményez, természetesen néhány kompromisszumos hátránnyal. Ilyen például az, hogy az állapot fenntartásához több erőforrás szükséges, többek közt a navigáció megvalósításához. [10]

Mivel alkalmazásomban használt keretrendszer React JS alapú, így én is egy ilyen felépítésű kliensalkalmazást készítettem. A Next.js több, egy átlag SPA-ra jellemző, nehézségre megoldást kínál. A keretrendszerbe beépítésre került a navigáció és az útválasztás is. Egy-egy oldal létrehozásához elegendő az erre kijelölt mappában egy fájlt létrehozni, és a fájl nevével megegyező oldalt kapunk a böngészőben. Ha hosszabb útvonalat szeretnénk egy-egy oldalnak, akkor azt mappák létrehozásával tehetjük meg. Az Next.js támogatja a dinamikus útválasztást, ebben az esetben a fájlokat kapcsos zárójellel ellátott névvel kell létrehozni.



6.6. ábra: Az elkészült alkalmazás oldalainak mappája

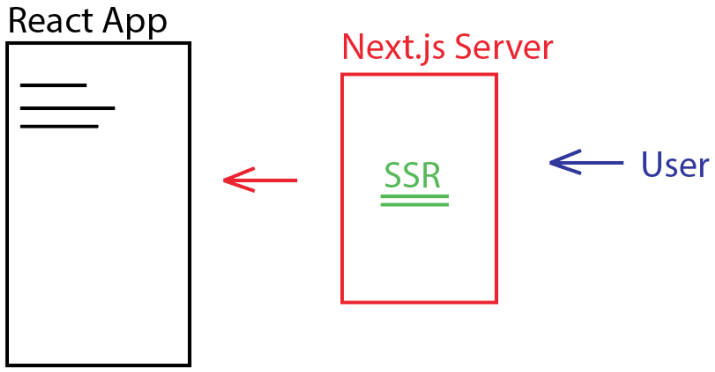
Egy SPA-ra jellemző, hogy a DOM (Document Object Model) manipulálására egy erőforrásbarátabb, összességében jobb megoldást kínál a hagyományos JavaScipttel szemben. A React JS és így a Next.js is virtuális DOM-ot használ. A virtuális DOM (VDOM) egy olyan koncepció, amelyben a felhasználói felület úgynevezett virtuális reprezentációja a memóriában van eltárolva, és a könyvtár - vagy a mi esetünkben keretrendszer - a ReactDOM szinkronizálja a valódi DOM-mal. Ez az úgynevezett egyeztetés folyamata. Ez a koncepció lehetővé teszi számunkra, hogy egy állapotváltozás bekövetkezésekor ne a teljes oldal frissüljön, hanem csak az a komponens, amiben az állapotváltozás bekövetkezett.

6.1.6. Next.js projekt felépítése

A Next.js egy React JS könyvtárra épülő JavaScript alapú keretrendszer. Mivel a Next esetében egy keretrendszert kapunk a React könyvtáron felül, így sok funkció alapból megtalálható benne, ami egy könyvtár esetében különféle integrációkat venne igénybe. A Next.js egy kliens- és szerveroldalt is magába foglaló keretrendszer. A mára hagyományosnak mondható webfejlesztésben egy alkalmazás kliens- és szerveroldala teljesen elkülönül egymástól. Viszont a mára elterjedt legnagyobb webes keretrendszerek többsége egyoldalas alkalmazásokat (SPA) hoznak létre. A Next kiküszöböli a háttérkeretrendszer szükségességét azáltal, hogy olyan React JS alkalmazásokat hoz létre, amelyek szerver által létrehozottak. Ez azt jelenti, hogy a felhasználó böngészője letölt egy teljes HTML oldalt, beleértve az összes szükséges elemet, például JavaScript és CSS fájlokat, valamint képeket.

A Next.js webpack használatával működik azért, hogy a teljes React alkalmazást egyetlen fájlba tömörítse, amely futtatható a szerveren. Amikor az alkalmazás betöltődik a szerverre, a Next.js a ReactDOMServer-t használja az alkalmazás megjelenítéséhez a szerveren. A Next.js ezután elküldi az elkészített HTML-t a kliensnek, ahol a kliensoldali React alkalmazás átveszi az irányítást és megjeleníti az oldalt. [11]

Az ilyen típusú megjelenítés legnagyobb előnyei: Az alkalmazás a szerveren jelenik meg, így azonnal elérhető a felhasználók számára anélkül, hogy meg kellene várnia a kliensoldali React alkalmazás betöltődését. A kliensoldali React alkalmazásnak csak egyszer kell betöltenie, még akkor is, ha a felhasználó több oldalt is felkeres az alkalmazásban. A Next.js számos olyan funkciót kínál, amelyek megkönnyíti a szerveren megjelenített alkalmazások fejlesztését. Alkalmazásomban ezekből többet is kihasználtam.



6.7. ábra: Next.js alkalmazás a böngészőben

6.2. Tesztelés

Az alkalmazás fejlesztésének utolsó fázisában teszteltem az alkalmazást. Tesztelési eljárásokból egységteszteket hajtottam végre az alkalmazáson. Fontos kiemelnem, hogy az ilyen jellegű tesztelések nem egyenlőek a felhasználói tesztelésekkel vagy kipróbálásokkal. Megfelelő tesztesetek megválasztásával lehetőségünk van a lehetséges hibákat még a fejlesztés során kiszűrni és javítani. Alkalmazásomban minden lekérdezéshez és mutációhoz használt függvény tesztelésre került. Mivel alkalmazásomban megfelelően szétválasztásra kerültek a különböző rétegek, így a tesztelési fázis is gördülékenyen tudott folytatódni. A teszteléshez egy külön mappát hoztam létre, ide csak olyan fájlok kerültek, amik ehhez a szakaszhoz szükségesek voltak. A teszteket entitások alapján bontottam szét, így biztosítva, hogy minden entitás megfelelően kerüljön tesztelésre.

A folyamat elvégzéséhez a .NET keretrendszerhez legtöbbet alkalmazott NUnit-ot és Moq-t választottam. Az NUint egy tesztelési keretrendszer, a Moq pedig egy mockolási eljáráshoz használt keretrendszer. Az NUnit segítségével tesztosztályok hozhatók létre melyeket el kell látni a TestFixture attribútummal. Az osztályon belül a Test attribútummal ellátott függvények az egyes tesztesetek. A mockolás során egy utánzatot - más néven álobjektumot - készítünk, ami képes az eredeti objektumot helyettesíteni a tesztelés során. Ez segítséget nyújt az osztály elkülönített tesztelésében. Alkalmazásomban a mapparétegről készítettem minden tesztfájl esetében egy ilyen álobjektumot. Ezt minden tesztesetnél használtam, ahol valamely mapparétegbeli- műveletet kellett végrehajtani.

Összességében a tesztesetek átfogóak voltak, segítségükkel sikerült az alkalmazás esetleges hiányosságait kiszűrni és ezeket még időben javítani. Természetesen az alkalmazásban minden funkció nagy szerepet kapott, viszont vannak kiemelkedően fontos funkciók, melyek

tesztelésére különös figyelmet fordítottam. Az egyik ilyen a két fő entitás létrehozása, illetve a felhasználó saját profiljának szerkesztése. A kereskedők a saját oldalukkal módosíthatnak az arculatukon, nem közvetlenül profiljuk szerkesztésével, így ezen funkció tesztelését nem emeltem ki.

Teszt neve	Elvárt eredmény	Valós eredmény	Sikeres
`CreateUserTest`	új felhasználó kerül létrehozásra	új felhasználó lett létre hozva	✅
`EditUserTest`	A már meglévő felhasználót módosítjuk a bemenetek szerint	A felhasználó a bemeneti adatoknak megfelelően módosításra került	✅
CreateRetailerTest	Új kereskedő kerül létrehpzásra a bemeneti adatoknak megfelelően	Új kereskedő került létrehozásra a bemeneti adatoknak megfelelően	✅

6.8. ábra: Fő entitások létrehozásának és szerkesztésének kiemelt egységtesztjei

Ezen funkciókon felül kiemelt fontosságúnak tartom, hogy a kereskedők megfelelően létre tudják hozni oldalukat, azt az elvártak szerint tudják szerkeszteni, illetve a felhasználók véleményezni tudják őket. A következő tesztesetekben ezen funkciókat teszteltem. Minden esetben az elvárt kimenetet kaptam az adott bemenetekre, így minden teszt sikerrel zárult.

Teszt neve	Elvárt eredmény	Valós eredmény	Sikeres
`CreateRetailerPageTest`	új kereskedői oldal kerül létrehozásra	új kereskedői oldal lett létre hozva	✅
`EditRetailerPageTest`	A már meglévő kereskedői oldalt módosítjuk a bemenetek szerint	A meglévő kereskedői oldalt a bemeneti adatoknak megfelelően módosításra került	✅
CreatePostTest	Új bejegyzés kerül létrehozásra a bemeneti adatoknak megfelelően	Új bejegyzés került létrehozásra a bemeneti adatoknak megfelelően	✅
`EditPostTest`	A már meglévő bejegyzést módosítjuk a bemeneti paramétereknek megfelelően szerint	A meglévő bejegyzés a bemeneti adatoknak megfelelően módosításra került	✅
`DeletePostTest`	A megfelelő bejegyzés eltávolításra kerül az adatbázisból	A bejegyzés az elvártak szerint eltávolításra került	✅
`CreateCommentTest`	Új komment kerül létrehozásra a megfelelő bejegyzéshez	Új komment került létrehozásra a megfelelő bejegyzéshez a megfelelő paraméterekkel	✅

6.9. ábra: Szociális tevékenységek kiemelt egységtesztjei

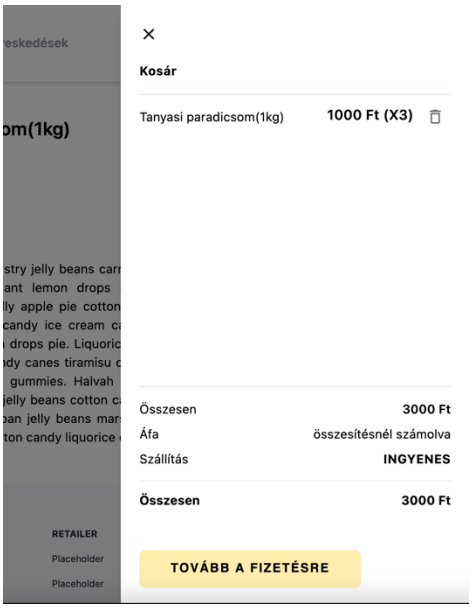
Az alkalmazás további funkcióihoz is hasonló körültekintéssel készítettem egységteszteket. Így minden esetben megbizonyosodhattam róla, hogy az egyes funkciók megfelelőek. További kiemelt szerephez jutottak a terméklétrehozás és a rendelés folyamatához tartozó egységtesztek.

7. AZ ELKÉSZÜLT RENDSZER BEMUTATÁSA

Az általam elkészített alkalmazás minden érdeklődő számára megfelelő felhasználói élményt kínál, különös tekintettel a dolgozat fókuszpontjában álló kiskereskedőkre. Ebben a fejezetben a két fő entitás oldaláról szeretném az alkalmazást bemutatni külön-külön. Ez a bemutatás segítséget nyújt az alkalmazás egyszerű, de mégis körültekintő használatához.

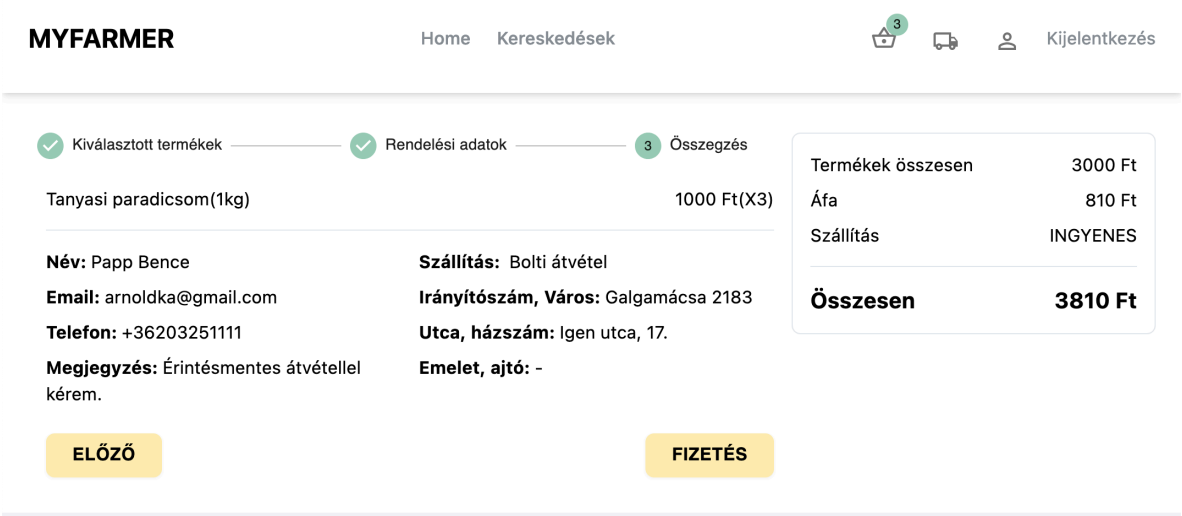
7.1. Az alkalmazás bemutatása egy felhasználó szemszögéből

Dolgozatomban elkészített és ismertetett alkalmazás a felhasználók számára lehetőséget kínál termékek keresésére, megvásárlására, valamint kereskedések felkeresésére és véleményezésére. Alapvetően, ha a teljes funkcionalitást ki szeretnénk használni, ahhoz elengedhetetlen a felhasználói fiók. Az alkalmazás nem korlátozza a böngészést fiók nélkül viszont így egyéb funkciókat a felhasználó nem tud igénybe venni. Ebben az esetben egy olyan folyamatot szeretnék bemutatni, ami a regisztrációval indul. Ha a bejelentkezés oldalon a felhasználó a regisztráció gombra kattint, akkor átirányításra kerül a regisztrációs oldalra, ahol néhány saját adat megadásával van lehetősége az platformra regisztrálni. Ezek után a főoldalra kerül átirányításra, ahol van lehetősége visszamenni a bejelentkezés oldalra, ahol a frissen létrehozott fiókjával képes bejelentkezni. Ezek után már autentikált felhasználóként képes vásárolni, illetve kommenteket írni bejegyzésekhez. Ha néhány terméket a kosarába tesz, akkor a kosarában ezek a termékek egyből megjelennek, ugyanezen a fülön el is távolíthat egységeket abból.



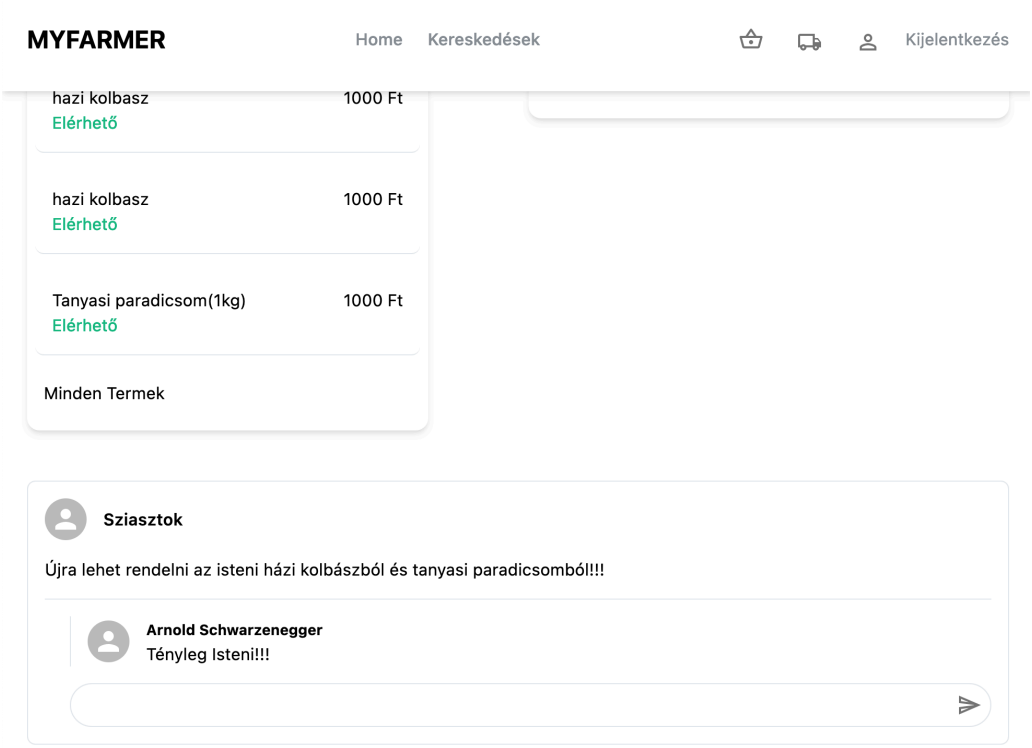
7.1. ábra: A felhasználó kosara

Ha a felhasználó ezután a „Tovább a fizetésre” gombra kattint, akkor átirányításra kerül a fizetési oldalra. Itt egy háromlépcsős rendelés leadási felület fogadja, aminek első oldala egy áttekintés, a második oldalon a rendeléshez szükséges adatait kell megadnia, a harmadik lépésben pedig ezek összesítését láthatja.



7.2. ábra: A rendelés leadásának fizetés előtti lépése

Ha az utolsó lépésnél a fizetés gombra kattint, akkor a Stripe saját felületére irányítja őt az alkalmazás, ahol biztonságosan megadhatja a fizetéshez szükséges adatait és végrehajthatja a tranzakciót. A szolgáltatást csak tesztmódban tudtam használni ingyenesen. Ennek okán valódi fizetést nem tudtam végrehajtani, csak a Stripe CLI adta lehetőségeket tudtam alkalmazni. Ez a CLI lehetőséget ad egy hamis bankkártya használatára teszteléshez, ebben az esetben is ezt alkalmaztam. A sikeres fizetés végeztével a felhasználót a sikeres fizetés oldalra irányítja az alkalmazás. Ezután az emailküldő szolgáltatás egy emailt küld a felhasználó email címére a rendelésről. Ezenkívül a rendelés fülön láthatja a felhasználó a rendeléseit. Ha szeretne módosítani adatain, azt a profiljára kattintva teheti meg. Ha elégedett volt a termékkel és a kereskedővel, akkor könnyedén véleményezheti a kereskedőt egy komment formájában. Ezt a kereskedő saját oldalán teheti meg.



7.3. ábra: Komment írása a kereskedő oldalára

7.1. Az alkalmazás bemutatása egy kereskedő szemszögéből

Az elkészült alkalmazás egyik fő fókuszcsoportja a kiskereskedők. A platform lehetőséget ad számukra saját oldal létrehozására, ahol bemutathatják kereskedésüket, termékeiket, illetve írhatnak különféle bejegyzéseket, amiket utána szerkeszthetnek vagy akár el is távolíthatnak. A kereskedők számára saját oldal létrehozásához nélkülözhetetlen a saját fiók. Hasonlóképpen, mint a felhasználók esetében, számukra is van egy bejelentkezési felület, illetve regisztrációs oldal is. Sikeres regisztráció, illetve bejelentkezést követően, a kereskedő ellátogathat saját oldalára. Ez az oldal első idelátogatáskor jön számára létre. Az oldal létrejötte után már egyből megtalálható lesz a felhasználók számára a kereskedő.

Ha termékeire beérkezik egy rendelés, akkor azt a rendelések fülön láthatja a kereskedő. Itt egy átfogó áttekintést láthat a rendelésről, és sikeres teljesítés esetén teljesítettnek jelölheti meg az adott rendelést. Ezután a rendelés továbbra is meg fog jelenni ezen az oldalon, de már csak áttekintés miatt, megfelelően el fog különülni a még nem teljesített rendelésektől, és megjelölésre kerül egy „Teljesítve” jelölővel.

8. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

Szakdolgozatomban elkészített alkalmazás implementálja a tőle elvárt funkciókat. Viszont egy modern szoftverhez méltón, nyitott a bővítésre. Azt gondolom, idővel az alkalmazás több helyen is bővíthető, ezen lehetőségeket priorizálni viszont nehéz. Véleményem szerint a felhasználók véleménye lehet az, ami a priorizálás alapját adja. Mivel az alkalmazás még nem került kipróbálásra éles körülmények között, ebben a kérdésben mégis a saját véleményemet kell, hogy szem előtt tartsam. A legnagyobb továbbfejlesztési faktor a szűrési és keresési lehetőségek bővítése. Ezekből már most is megtalálható bizonyos fajta szűrés a termékek között, viszont ez mérvadóan sok termék esetén lehetséges, hogy egyes felhasználóknak nem lesz elegendő. Ezen felül a kereskedő számára is implementálnék szűrési lehetőségeket, mind a saját oldalukon, mind pedig a rendelések áttekintése oldalon. Ezzel a kereskedők könnyedén tudnák ezeket az oldalakat, akár teendőlistaként is használni, és segítene a napjuk megtervezésében, hogy mikor milyen rendeléssel kell még foglalkozniuk.

Egy másik fontos továbbfejlesztési faktor lenne, ha az emailküldő szolgáltatást nem csak teszt üzemmódban tudnám alkalmazni. Ez sajnos anyagi vonzatokkal járna. Ebben az esetben lenne lehetőségem, mind a kereskedőknek, mind pedig a vásárlóknak, emailértesítéseket küldeni a különböző státuszváltozásokról, megrendelésekről, vagy ha éppen új komment érkezett valamelyik kereskedő oldalára. Ezek mellet a fizetési szolgáltatást is szeretném a későbbiekben nem csak tesztmódban alkalmazni. Ebben az esetben a kereskedő számára is azonnal tudnék értesítést küldeni, illetve a kifizetés is hatékonyabban végbemenne. Ezek mellett átfogó adminisztrátori oldallal bővíteném tovább a platformot, hogy az adminisztrátori jogosultságú felhasználók a különböző tevékenységeket egyszerűbben tudják végrehajtani.

9. ÖSSZEGZÉS

Szakdolgozatom elkészítésének kezdetén kitűztem alapvető célokat, amiket a folyamat végén mindenképp célom volt, hogy elérjek. Ezek a célok egyaránt érintettek technikai és felhasználási célokat is. Olyan platformot terveztem létrehozni, ami a dolgozatban ismertetett problémákra egy átfogó és egyszerűen használható megoldást kínál. Ennek a megvalósításának a céljából sorra vettem a legnépszerűbb és legelterjedtebb platformokat, és igyekeztem meglátni azok legnagyobb erősségeit és gyengeségeit. Technikai oldalról megkerestem azokat a keretrendszereket és eljárásokat, amik már sok helyen bizonyítottak a nagyvilágban, és a fejlesztők előszeretettel választják őket a feladatok elvégzésére. Törekedtem arra, hogy a választott technológiák nyújtotta előnyöket a legnagyobb mértékben kiaknázhassam és a fejlesztési normáknak megfelelően alkalmazzam őket.

Szakdolgozatomban egy webes alkalmazást készítettem. Ez az alkalmazás megoldást jelent a kiskereskedők egyik alapvető problémájára: termékeiket nem tudják megfelelően népszerűsíteni és értékesíteni az online térben. Egy olyan platformot sikerült létrehoznom, ahol a vásárlók és a kereskedők megtalálhatják egymást, és többé nem szükséges hosszú utánajárással megtalálni azokat - a sokszor csak termelőknél megtalálható - termékeket, amikre a vásárló vágyik. A kereskedők nem csak termékeiket tölthetik fel az oldalra, de van lehetőségük saját kis oldalt is létrehozni az alkalmazáson belül. Ez további segítség egy-egy bolt felvirágoztatására. A vásárlók számára van lehetőség véleményt nyilvánítani a kereskedőkről vagy épp azok valamely termékéről.

A munkám - saját értékelésem alapján - egy felhasználóbarát és letisztult alkalmazás lett, amely teljesíti a vele szemben támasztott elvárásokat. A célcsoportoknak megfelelő szolgáltatásokat nyújt és használata egyszerű. Úgy gondolom, a platform megfelelő segítség azok számára, akik a már korábban ismertetett célcsoportok valamelyikébe tartoznak. Egyszerűbbé teszi a vásárlást és megkíméli a kereskedőket a különböző technikai terhektől, amit egy saját platform vagy egy saját online piactér fenntartása jelentene. Ezek mellett, könnyen követhetővé teszi a rendelések kezelését, ami több jövőbeli fejlesztési lehetőséget is előrevetít. Azt gondolom, hogy ez a szoftver új utakat nyithat meg mindazok számára, akik az online tér adta lehetőségeket szeretnék kihasználni a mindennapos vásárlások folyamán.

10. SUMMARY

At the beginning of my thesis, I set basic goals, which I definitely wanted to achive. These purposes covered both technical and operational purposes. I have planned to create a platform that offers a comprehensive and easy-to-use solution to the problems described in the thesis. To accomplish this, I listed the most popular and widespread platforms and tried to see their greatest strenghs and weaknesses. On the technical side, I have been looking for frameworks and produces that have already been proven in many places around the world, and developers prefer to choose them to perform tasks. I have striven to maximize the benefits of the technologies I have chosen and to apply them in accordance with development standards. In my thesis I created a web application.

This app is a solution to one of the fundamental problems of retailers: they are not able to properly promote and sell their products online. I have managed to create a platform where buyers and traders can find each other, and it is no longer necessary to spend hours to find the wanted products. Merchants can upload their products to the platform and they also have option to create their own page within the app. This is an additional help to make a store flourish. Buyers have the opportunity to comment on merchants sites.

My work – based on my own assessment – has become a user-friendly and clean application that meets the expectations placed on it. It provides services that are appropiate for the target groups and easy to use. I think the platform is a good help for those who belong to one of the target groups. It simplifies shopping and saves traders from the various technical burdens of maintaining their own platform or online marketplace.

In addition, it makes order management easy to track, which predicts more future development opportunities. I think this software could open up new avenues for anyone who wants to take advantage of the opportunities offered by online space in their day-to-day shopping.

IRODALOMJEGYZÉK

[1] https://www.grubtech.com/blog/history-of-food-delivery (utoljára megtekintve: 2021.11.08.)

[2] http://static.hlt.bme.hu/semantics/external/pages/szemantikus_web/hu.wikipedia.org/wiki/HTTP.html (utoljára megtekintve: 2021.12.05.)

[3] https://en.wikipedia.org/wiki/GraphQL (utoljára megtekintve: 2021.11.08.)

[4] https://developer.mozilla.org/en-US/docs/Web/JavaScript (utoljára megtekintve: 2021.12.06.)

[5] https://developer.mozilla.org/en-US/docs/Web/CSS (utoljára megtekintve: 2021.12.06.)

[6] https://szit.hu/doku.php?id=oktatas:programozas:fejlesztesi_modellek_es_modszertanok (utoljára megtekintve: 2021.12.05.)

[7] https://devblogs.microsoft.com/dotnet/announcing-net-6/ (utoljára megtekintve: 2021.12.03.)

[8] https://en.wikipedia.org/wiki/Next.js (utoljára megtekintve: 2021.11.23.)

[9] https://hdi.uni-nke.hu/document/hdi-uni-nke-hu/tezisfuzet-gerevich-janos.pdf (utoljára megtekintve: 2022.04.22.)

[10] https://developer.mozilla.org/en-US/docs/Glossary/SPA (utoljára megtekintve: 2022.04.25.)

[11] https://nextjs.org/ (utoljára megtekintve: 2022.04.29.)

ÁBRAJEGYZÉK

6.2.: ábra: HotChocolate keretrendszer GraphQL API teszteléshez használható végpontja

Saját készítésű ábra

6.3.: ábra: Kliensoldali alkalmazás bejelentkezési felülete

Saját készítésű ábra

6.4: ábra: A kereskedő saját oldala

Saját készítésű ábra

6.5.: ábra: Stripe fizetés áttekintő felülete

Saját készítésű ábra

6.6.: ábra: Az elkészült alkalmazás oldalainak mappája

Saját készítésű ábra

6.7.: ábra: Next.js alkalmazás a böngészőben

Saját készítésű ábra

6.8.: ábra: Fő entitások létrehozásának és szerkesztésének kiemelt egységtesztjei

Saját készítésű ábra

6.9.: ábra: Szociális tevékenységek kiemelt egységtesztjei

Saját készítésű ábra

7.1.: ábra: A felhasználó kosara

Saját készítésű ábra

7.2.: ábra: Rendelés leadásának fizetés előtti lépése

Saját készítésű ábra

7.3.: ábra: Komment írása a kereskedő oldalára

Saját készítésű ábra