Szegedi Tudományegyetem

Informatikai Intézet

Diplomamunka

Fürstahl Bence Dániel

2023

Szegedi Tudományegyetem

Informatikai Intézet

**Általánosítható webshop létrehozása react/firebase segítségével**

Diplomamunka

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Készítette: |  | Témavezető: |  |
|  | Fürstahl Bence Dániel |  | **Dr. Bilicki Vilmos** |  |
|  | Mérnökinformatikus mesterszakos hallgató |  | Egyetemi adjunktus |  |

Szeged

2023

# Feladatkiírás

A diplomamunkám célja egy az újonnan létrejövő cégek számára gyorsan, olcsón kitelepíthető weblap minta létrehozása. Ezen cél elérése érdekében létre kell hoznunk egy könnyedén átlátható weblapot, mely rendelkezik regisztrációs illetve regisztráltalan áruátlátással, áru kategória lapokkal, kosárral illetve fizetési lehetőségekkel. Ezek kiegészítésére egy a céges adminok számára használható aloldalakat is létre kell hoznunk, melyek segítségével a cég képes nyomon követni az oldalon látható árukat regisztrált felhasználókait illetve az egyes rendeléseket és azok állapotát.

A projekt megvalósításához a kliens oldalon JavaScriptes React illetve Tailwind CSS keretrendszereket használtam, míg a backendhez a Google Firebase szolgáltatást használom.

# Tartalmi összefoglaló

**Téma megnevezése:** A szakdolgozatom kereteiben egy egyszerű kiinduló webshop weblap template létrehozása a célom React, Taiwind CSS és Google Firebase segítségével, amely könnyedén adaptálható az adott webshop igényeihez, illetve a Tailwind CSS-nek köszönhetően akár mobilra is.

**Feladat megfogalmazása:** A webalkalmazásom lehetővé teszi, hogy egy újonnan létrejött cég könnyedén, gyorsan létre tudjon hozni egy olyan webes felületet, melynek segítségével megkezdhetik árujuk online árusítását. Ezen célból a weblapnak tartalmaznia kell alapvető felhasználó és áru kezelést, az áruk megvásárlásának lehetőségét illetve az egyes rendelések nyomon követesét.

**Megoldási mód:** A feladat megvalósításához egy webes felület létrehozását választottam, mely a Google Firebase szolgáltatásra épül. Ezen szolgáltatás használatának előnye, hogy a Google egy univerzálisan ismert és elfogadott szolgáltató így a technológiában nem járatos személyek is relatív bizalommal fogadják el adataik tárolására. Ezen felül a szolgáltatás sok pluginje nagyméretű specializációt tesz lehetővé. A weblap megvalósításához a React keretrendszert választottam, melynek köszönhetően a weboldal könnyedén adaptálható a felhasználó elvárásaihoz illetve a kinézet megvalósításához a Tailwind CSS adja, talán az egyik leggyorsabb ás átláthatóbb adaptációs opciókat.

**Alkalmazott eszközök, módszerek:** A webalkalmazás megvalósításához a React illetve a Tailwind CSS keretrendszereket használtam, míg a backendet a Google firebase szolgáltatás biztosította. Email küldéshez a Google Smtp szolgáltatását használtam a Firestore Trigger Email from Firebase extensionjével, míg fizetéshez a Stripe integrációjával próbálkoztam, Ezenfelül a fejlesztéshez használtam a NodeJS környezetet illetve Visual Studio Code fejlesztői eszközt. Verziókövetéshez GitHub-ot alkalmaztam.

**Elért eredmény:** Úgy érzem munkám során sikerült egy olyan alapvető weblapot létrehoznom mely elegendően kielégíti az alapvető elvárásokat, amiket egy új cég támaszthat webalkalmazásával szemben. Munkám során próbáltam ügyelni arra, hogy olyan technikákat alkalmazzak melyek későbbi munkám során is hasznomra lehetnek.

**Kulcsszavak:** *React, Tailwind CSS, Google Firebase, NodeJS, Webshop*

# Tartalomjegyzék

[Feladatkiírás 1](#_Toc153447455)

[Tartalmi összefoglaló 2](#_Toc153447456)

[Tartalomjegyzék 3](#_Toc153447457)

[Motiváció 5](#_Toc153447458)

[1. Területi áttekintés 6](#_Toc153447459)

[1.1. Navigációs sáv 6](#_Toc153447460)

[1.2. Jogosultságok 7](#_Toc153447461)

[1.3. Termékkezelés és kosár 7](#_Toc153447462)

[1.4. Fizetés 8](#_Toc153447463)

[1.5. Kapcsolattartási lehetőségek 8](#_Toc153447464)

[2. Felhasznált technológiák 9](#_Toc153447465)

[2.1. React framework 9](#_Toc153447467)

[2.2. TailwindCSS 9](#_Toc153447468)

[2.3. Google Firebase 10](#_Toc153447469)

[2.4. NodeJS 10](#_Toc153447470)

[2.5. Stripe 11](#_Toc153447471)

[2.6. Verziókövetés 12](#_Toc153447472)

[3. Fontosabb folyamatok 13](#_Toc153447473)

[3.1. Regisztráció és profil kezelés 13](#_Toc153447474)

[3.2. Kosár és fizetés 13](#_Toc153447475)

[4. Funkcionális specifikáció 16](#_Toc153447476)

[4.1. Alkalmazás Use-Case Diagramja 17](#_Toc153447477)

[4.2. Regisztráció és autentikáció 17](#_Toc153447478)

[4.3. Termék kategória lapok 19](#_Toc153447479)

[4.4. Kosár 21](#_Toc153447480)

[4.5. Fizetés opciók 23](#_Toc153447481)

[4.6. Profil 25](#_Toc153447482)

[4.7. Admin felületek 25](#_Toc153447483)

[4.7.1. Termékek 25](#_Toc153447484)

[4.7.2. Rendelések 26](#_Toc153447485)

[4.7.3. Felhasználók 26](#_Toc153447486)

[4.8. Rólunk oldal 26](#_Toc153447487)

[4.9. Routing 27](#_Toc153447488)

[5. Architektúra 28](#_Toc153447489)

[6. Adatmodell 29](#_Toc153447490)

[6.1. Felhasználó 29](#_Toc153447491)

[6.2. Termék 30](#_Toc153447492)

[6.3. Rendelés 31](#_Toc153447493)

[7. Jogosultságkezelés 32](#_Toc153447494)

[8. Fontosabb kódrészek 33](#_Toc153447495)

[8.1. Authentikáció 33](#_Toc153447496)

[8.1.1. Regisztráció 33](#_Toc153447497)

[8.1.2. Bejelentkezés 33](#_Toc153447498)

[8.1.3. Kijelentkezés 34](#_Toc153447499)

[8.2. State managment 34](#_Toc153447500)

[8.3. Kosár 36](#_Toc153447501)

[8.4. Fizetés 37](#_Toc153447502)

[9. Tapasztalatok, továbbfejlesztési lehetőségek 40](#_Toc153447503)

[Felhasznált irodalom, források 41](#_Toc153447504)

[Nyilatkozat 42](#_Toc153447505)

# Motiváció

Diplomamunkám elkészítésének két fontos motivációja volt. Elsődleges motivációm az, hogy jelenleg gyakran dolgozok különböző weblapokkal illetve webshopokkal igy egy ilyen téren elkészített diplomamunka sok tapasztalatot nyújthat számomra jövőbeli projektjeim során.

Diplomamunkámban azért céloztam meg egy általános webshop létrehozását, mert ez az egyik legyakoribb kérés, ami munkám során szembejön. Egy gyorsan kitelepíthető webshop ami képes addig üzemelni amíg egy specializáltabb weblap megépül, így az adott üzlet tud profitot termelni ezen idő alatt is, ugyanis egy újonnan alapított cég manapság a nagy multinacionális cégek világában képtelen akár rövidtávon is éltben maradni valamilyen alapvető webes felület nélkül.

Diplomamunkám második motivációja, hogy a közeljövőben több alkalommal is a Google Firebase szolgáltatásával kell majd dolgoznom, igy szerettem volna minél előbb tapasztalatot szerezni ezen szolgáltatás működésével kapcsolatban.

Diplomamunkámat e két fő motiváló ok alapján próbáltam meg elkészíteni.

# Területi áttekintés

Mivel diplomamunkám célja egy átlátható és jól működő webshop megvalósítása volt, ezért első lépésben azt kellett átgondolnom, hogy mik azok az elégethetetlen részek, amik nélkül egy webshop nem képes üzemelni, illetve hogy ezek megvalósítása hogyan lehetséges.

Ehhez első lépésként végignéztem több különböző áruházlánc webshopját és összevetettem őket csak webes árusítással foglalkozó cégek webshopjaival. Ezen kutatásból könnyen leszürhető volt, hogy vannak olyan funkciók és aloldalak, amelyek minden webshopban megtalálhatók. Ezen eredmények alapján indult el gondolatmenetem webshopom megvalósítása során.

## Navigációs sáv

Az első és talán legfontosabb eleme minden webshopnak egy navigációs sáv melynek segítségével a felhasználók könnyedén megtalálhatják az általuk keresett termékeket. Egy ilyen navigációs sávnak érdemes minden oldalon egyformán megjelennie, igy a felhasználó könnyedén láthatja, megértheti és használhatja. A navigációs sávnak tartalmaznia kell a felhasználó számára legfontosabb menüpontokat, illetve ha ez olyan, akkor akár jelezheti is a változásokat pl.: a kosár menüpontja mellet megjeleníthetjük, hogy éppen hány terméket tartalmaz.

A navigációs sávnál fontos, hogy az egyes menüpontok nevei egyértelműek legyenek, hiszen a felhasználónak könnyedén értenie kell, hogy mit tesznek az egyes ezek, illetve fontos, hogy a navigációs sáv csak a legfontosabb elemeket tartalmazza, ugyanis, ha túl sok menüpontunk van akkor az könnyedén megzavarhatja a felhasználót, illetve az átláthatóságot is csökkenti. Ezen problémának a legegyszerűbb megoldása, ha egyes menüpontokat lenyílóvá tesszük és a sok aloldal helyett egy gyűjtőnevet teszünk csak ki a navigációs sávra.

## Jogosultságok

A navigáció végig gondolása során könnyedén beláthatjuk, hogy a weblapunknak lesznek olyan oldalai, amit nem szeretnénk, hogy akárki, aki az oldalra látogat, elérhessen. Ezen okból kifolyólag szükségünk van arra, hogy megkülönböztethessük az egyes felhasználókat és az egyes oldalak csak az adott jogosultsággal rendelkező felhasználók érhessék el.

Ezen probléma megoldása érdekében a webshop felhasználóit három kategóriába osztottam: vendég, regisztrált felhasználó, admin. Ezek relatív nagyméretű kategóriák melyeket a webshop továbbfejlesztése során érdemes lehet tovább osztani illetve gondolni, hiszen nem feltétlen érdemes minden admin felhasználónak ugyanolyan jogosultságokat és hozzáféréseket adni, ugyanis az megnövelheti a kockázatok mértékét, illetve az sem biztos, hogy mindig érdemese-e nem regisztrált felhasználóknak is engedni, hogy vásároljanak az oldalunkon. Jelen felosztásom alapjául az a meggondolás szolgált, hogy egy üzlet oldalát nem feltétlen csak azok látogatják, akik online akarnak feltétlenül vásárolni, hanem azok is, akik szeretnék megnézni, hogy az adott üzlet árulja-e az általuk keresett termékeket, így nem célszerű csak azokat a felhasználókat az oldalra engedni, akik regisztráltak. A másik oldalról viszont érdemes előnyöket biztosítani azon felhasználók számára, akik regisztráltak pl.: leárazások.

## Termékkezelés és kosár

A különböző jogosultsági szintek megállapítása után hamar beláttam, hogy komoly problémát okozhat a kosár termékeinek és a fizetési lehetőségeknek a megvalósítása. Ha azt szeretnénk, hogy a weboldal emlékezzen a kosárunk tartalmára azt valahol tárolnunk kell. Azon esetben, ha az oldalt csak regisztrált felhasználók használják ez könnyedén megvalósítható a backenden a felhasználó azonosítójával, viszont, ha nem regisztrált felhasználóknak is akarunk ilyen funkciót, akkor létre kell hoznunk egy ideiglenes felhasználót backenden, amelyet megfelelő időközönként tisztítunk. Itt viszont egy olyan problémába ütköztem, hogy a Firebase nem támogatja az ilyen típusú műveleteket, azaz nem tudjuk automatizáltan törölni a backendről az ideiglenes felhasználókat, ezért a második legegyszerűbb megoldást választottam és a kosár tartalmát egy cookie-ban tárolom lokálisan 2 nap élettartammal így a nem regisztrált felhasználóknak sem kell amiatt aggódniuk, hogy elveszik a kosaruk tartalma, ha frissül az oldal.

## Fizetés

A következő lépés, ami komolyabb átgondolást érdemel az a weblapon történő fizetési opciók. A legegyértelműbb fizetési opció az úgynevezett utánvét, vagyis amikor a vásárló a termék kiszállításakor készpénzzel fizet az áruért. Ennek megvalósítása a weblap szempontjából nem igényel komoly átgondolást, de a webshopoknak érdemes bankkártyás fizetési opciót is biztosítaniuk. Ehhez a Stripe nevezetű pénzügyi fizetési infrastruktúrát implementáltuk. Azon esetben, ha a vásárló szeretne bankkártyával fizetni fontos, hogy az adott tranzakció biztonságos csatornákon haladjon keresztül, hiszen rendkívül nagy probléma az, ha ellopják valakinek a pénzügyi adatait. A Stripe ilyen biztonságos fizetési környezetet biztosít, támogatja az egyszeri és ismétlődő fizetéseket, nyomon követhető rajta az oldalon áthaladt pénzmozgások, támogat lokalizációt, különböző kártyatípusokat és kezel e-mail alapú visszajelzéseket is az egyes tranzakciókról. Így elmondhatjuk, hogy a Stripe integrációja egy remek megoldásnak tűnik a projekt megvalósítására.

## Kapcsolattartási lehetőségek

Minden webshopnál nagyon fontos hogy legyenek feltüntetett elérhetőségek, hiszen ha valami probléma történik a rendeléssel, akkor a vásárlónak szükséges, hogy képes legyen valahogy kapcsolatot teremteni magával a céggel. Ezen elérhetőségek alatt általában fizikai címet, email címet, telefonos elérhetőséget értünk. A webshopok többségénél az ilyen elérhetőségek a lábjegyzetben vannak feltüntetve, amik megjelennek minden egyes oldal alján, illetve ha úgy érezzük ez nem elég akkor még egy külön kapcsolattartási oldalt is létrehozhatunk ahonnan a felhasználó akár direktbe is írhat a cégnek. Azon esetben, ha külön oldalt hozunk létre az elérhetőségeknek érdemes elgondolkodni azon, hogy ezen az oldalon akarjuk-e ugyanazokat az információkat meg egyszer a láblécben feltüntetni.

# Felhasznált technológiák

Diplomamunkám során sok különböző technológiát használtam. Ezek nagyrésze piacon aktívan működő alkalmazásokban is megtalálhatóak, feladatuk a weboldal működéséért felel, de vannak olyanok is, amelyeket a fejlesztési folyamat megsegítésére használtam. A következő fejezetben ezen technológiákról szeretnék adni egy rövid összefoglalót!



## React framework

A React egy ingyenes open-source front end fejlesztési JavaScript könyvtár melynek fejlesztése és fenntartása a Meta csoporthoz köthető. Működésé komponenseken alapul melyek segítségével könnyedén lehet single-page vagy mobil vagy server-render applikációkat felépíteni. Open-source mivoltának köszönhetően sok könyvtárral rendelkezik, melyek szintén nagyban megkönnyítik a fejlesztési folyamatot. Diplomamunkám fejlesztése során több ilyen könyvtárat alkalmaztam:

* react-modal: Segítségével könnyedén hozhatunk létre modal-okat weblapunk során melyek megkönnyítik az adatfelviteleket.
* react-router-dom: Ezen könyvtár segítségével könnyedén létrehozhatunk router elemeket melyek segítenek az egyese aloldalak közötti mozgásban.
* redux: Ezen könyvtár segítségével létrehozhatunk egy store-t, amivel könnyedén tudjuk a state-ben tárolt adatokat frissíteni, illetve olvasni.

## TailwindCSS

A TailwindCSS egy open-source css framewoek melynek segítségével a fejlesztő könnyedén és gyorsan befolyásolhatja az egyes elemek kinézetét. Egyik legnagyobb előnye, hogy más css framework-ökkel szemben nem tartalmaz előre definiált classok-at elemekhez, így sokkal nagyobb szabadságot biztosít a programozónak.

TailwindCSS alkalmazásával könnyedén és átláthatóan módosíthatjuk programunk elemeinek kinézetét, mivel a classokat magában a murkupban hívhatjuk meg, igy nem kell keresgélnünk, hogy az egyes elemekhez mely css komponensek kötődnek. Segítségével könnyen építhetünk reszponzív designokat weblapjaink számára.

## Google Firebase

A Google Firebase egy a Google által biztosított szolgáltatás, amely egy teljes platformot biztosít az alkalmazásunknak. Segítségével egyszerűbb alkalmazásoknál elhagyhatjuk a szerver oldali implementációt, hiszen ezt biztosítja számunkra a Google.

A Google Firebase egyik legnagyobb előnye, hogy valós idejű adatbázist biztosít, melynek Firestore Database a neve. Ezen database elérése websoketeken keresztül történik, így nem kell http requetekkel foglalkoznunk, ez nagyban megnöveli az applikációnk működésének sebességét. Egyetlen problémája, hogy csak NoSQL adatbázist hozhatunk létre benne.

A Google Firebase másik nagy előnye, hogy van beépített Authentikációs szolgáltatása, ahol több különböző authentikáció közül választhatunk. Ezek lehetnek email/jelszó, telefon, de akár közösségi média fiókkal pl.: Facebook, Twitter stb. történő bejelentkezés is. Ezen felül a szolgáltatás nyújt programunk számára verifikációs és jelszó reset opciókat és ezekhez email templateket melyek, összeköthetők saját smtp providerrel.

Ezenfelül a Google Firebasehez sok különböző extension található, amik segíthetnek különböző problémák megoldásában, mint e-mail küldés vagy egyéb külső szolgáltatással való összekötésben.

## NodeJS

A NodeJS egy open-source cross-platform JavaScript futtatási környezet. Ezen környezetnek köszönhetően a fejlesztés során hozzáférésem volt a Node Package Manager-hez (npm) melynek segítségével képes voltam a több különböző csomag telepítésére a projektemhez.

Diplomamunkámhoz elengedhetetlen volt a NodeJS környezet használata, hiszen ezen keresztül tudtam letölteni mind a TailwindCSS-t mind a React csomagokat, illetve a Firebase-nek is van a Google által kiadott csomagjai melyek segítségével az alkalmazásom képes volt a Firebase egyes funkcióihoz csatlakozni.

## Stripe

A Stripe egy univerzálisan elfogadott és használt pénzügyi szolgáltatás melynek segítségével automatizáltan fogadhatunk fizetéseket. A Stripe integrálásával programunk egyszeri vagy megújuló bankkártyás fizetéseket fogadhat. A szolgáltatás automatizálja a pénzügyi tranzakciókat, felügyeli a megújuló fizetések lejárási dátumait és automatizáltan jelzi vásárlók felé, ha fizetési szükségletük van.

A Stripe szolgáltatás segítségével nagymértékben megkönnyebbül üzletünk pénzügyi tevékenységeinek követése, hiszen a szolgáltatás nagymértékű áttekintést nyújt összes az oldalunkon történt bankkártyás tranzakcióról. A tranzakciókról a Stripe automatikus értesítéseket küld a felhasználónak melyek testreszabhatóak, hogy megfeleljenek üzletünk elvárásainak.

A Stripe Google Firebasebe való integrálását megkönnyíti, hogy a Stripe csapata létrehozott és fenntart egy Firebase extension-t (Run Payments with Stripe), melynek segítségével automatikusan képesek vagyunk Stripe-ban tárolt termékeinket illetve Firebase felhasználóinkat egymás között szinkronizálni.

Azon esetben, ha a Firebase extension nem megfelelő a weblapunk számára a Stripe integrálása könnyen megoldható a Google Cloud Functions segítségével, melyeknek köszönhetően létrehozhatunk egy saját a Google által hostolt backend funkciót, ami a Firebase programunkból hívható.

## Verziókövetés

Minden fejlesztési folyamatról elmondható, hogy szerves része a verziókövetés, hiszen ennek hála láthatjuk, hogy milyen változások következtek be kódunkban illetve egy áttekintést is nyújthat számunkra projektünk állapotáról.

Ezen célra több különböző segédprogram is létezik, melyek közül talán a leghíresebb a Git. Ennek segítségével a programunk kódját képesek vagyunk feltölteni egy központi tárolóba (repositroy). Ilyen feltöltés (commit) esetén az adott változtatásokat egy névvel látjuk, el melyek segítenek abban, hogy később visszakeressük az egyes változásait a programnak. Ezenfelül az egyes fejlesztési folyamatokat szét tudjuk választani külön ágakra (branch), amelyekbe a feltöltések nem változtatnak más ágakon. Ezen ágak az adott fejlesztési ciklus befejezésével egyesíthetők (merge) a központi projekt ággal. Ez lehetőséget ad a kód felülvizsgálatára mielőtt az az éles verzióba kerülhetne. Ilyen egyesítésekkor előfordulhat, hogy konfliktusok lépnek fel melyeket a verzió követők le tudnak kezelni, de végül a programozóknak kell megoldaniuk

# Fontosabb folyamatok

Diplomamunkám ezen fejezetében szeretném ismertetni a weblapon történő fontosabb folyamatokat, ezzel könnyebb átláthatóságot biztosítva annak működésére.

## Regisztráció és profil kezelés

Azon esetben, ha egy felhasználó úgy érzi, hogy weblapunkat többször is használni fogja, nagy valószínűséggel nem akarja majd ismételten megadni adatait minden egyes vásárláskor, hanem szeretné azokat tárolni és automatikusan használni vásárlásaihoz. Ehhez nyújt megoldást a weblapon egy profil regisztrálása opció.

Ezen regisztrálási folyamat első lépése a regisztrációs link megnyomása a navigációs sávban a bejelentkezés gomb alatt. Ez a link a felhasználót tovább viszi egy regisztrációs oldalra, ahol a felhasználónak meg kell adnia azon adatait, amivel egy fiók létrehozható számára (név, e-mail, jelszó). Azután, hogy ez megtörtént a felhasználó számára a Firebase automatikusan kiküld egy megerősítő e-mailt, amivel megerősítheti regisztrációját.

A regisztráció megtörténte után a felhasználó a bejelentkezés gombbal beléphet weblapunkban fiókjába. Ekkor lehetősége van profilja alatt adatai módosítására, szállítási címek megadására, képes visszanézni korábbi rendeléseit, illetve ha úgy érzi akár törölheti is. Azon esetben, ha a felhasználó itt megad kiszállítási címeket, később azokat vásárláskor könnyedén felhasználhatja

## Kosár és fizetés

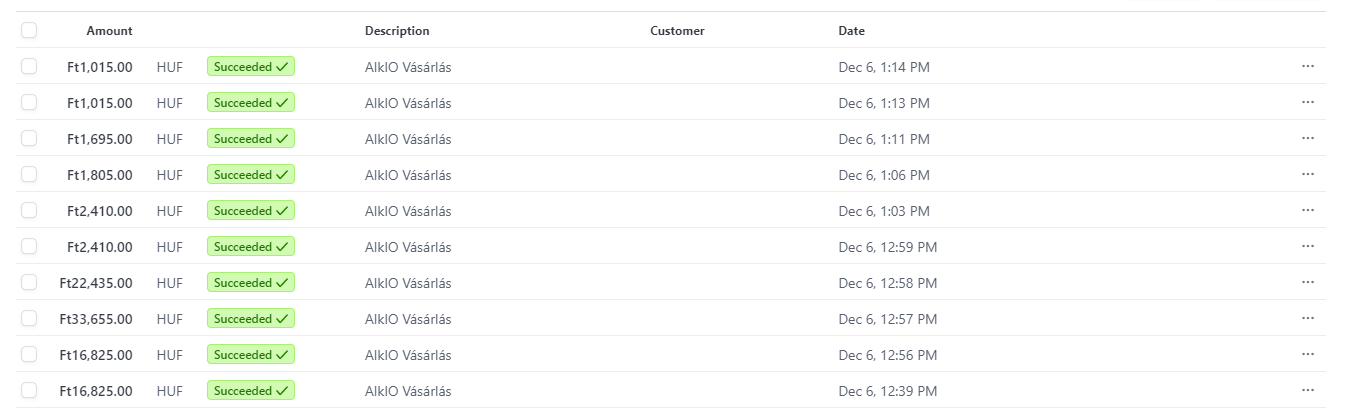
Minden webshop legfontosabb része a vásárlási és fizetési opciók. Ennek megvalósításának legfontosabb folyamatai a termékek kosárba tétele, a kosár összesítése a fizetés és a rendelések kezelése. Ezekből egyértelműen látható, hogy az első háromra minden egyes felhasználónak képesnek kell lennie, míg az utolsóhoz csak a céges adminoknak szabad, hogy hozzáférése legyen.

A vásárlási folyamat első lépése az egyes termékek kosárhoz adása, ehhez a felhasználó az egyes termékkategória oldalakon kikeresi az általa megvásárolni kívánt termékeket és azokat a kosárhoz adja. A kosár lokálisan state-ben és egy cookie-ban menti ezen adatokat annak érdekében, hogyha az oldal frissülne, azok ne vesszenek el és keljen a felhasználónak újrakezdenie a vásárlást. Az adatok cookie-ban két napig vannak tárolva, így a felhasználónak van ideje visszatérni a vásárlásához, ha azt valamilyen okból fogva félbehagyná.

Miután a vásárló összegyűjtötte, az általa megvásárolni kívánt termékeket a kosár oldalra navigálva láthat róluk egy összesítőt. Ennek segítségével a felhasználó meg egyszer a termékek megvétele előtt leellenőrizheti, hogy mindent, amit meg akart vásárolni megtalált-e, azok egyesével mennyibe kerülnek, illetve, hogy összesen mennyibe kerül az egész vásárlása. Ezen az oldalon a vásárló képes, ha szeretné a már kosarába helyezett termékeinek mennyiségét változtatni, azokból többet a kosárba helyezni, vagy csökkenteni azok mennyiségét.

Amikor a vásárló elégedett a kosarában levő termékekkel, egy megerősítés gombbal tovább kerül egy adatmegadó aloldalra, ahol kiválaszthatja fizetési módját, illetve ha bejelentkezett felhasználó választhat szállítási módot, vagy ha nem bejelentkezett felhasználó, megadhatja a szállításhoz szükséges elérési adatokat (lásd: 4.7 ábra). Miután a vásárló megadta a szükséges adatokat a rendeléshez attól függően, hogy milyen fizetési módot választott folytatódik a rendelés. Azon esetben, ha a felhasználó utánvétet (készpénzes fizetést) választott a megfelelő adatokból létrejön egy rendelés, amiről a vásárló egy automatikus e-mailt kap, ami tartalmazza vásárlása részleteit.

Azon esetben, ha a felhasználó a bankkártyás fizetést választotta, a rendelést a Stripe fizetési szolgáltatás veszi át. Ilyenkor a felhasználónak meg kell adnia bankkártya adatait, aminek a segítségével létrehozunk egy Stripe payment method-ot amit tovább küldök egy Google Cloud Function-nek. Ezen Cloud function létrehoz egy Stripe payment intent-et aminek a segítségével a Stripe levonja a vásárló számlájáról a megfelelő összeget és azt továbbítja a Stripe weboldalán megadott céges számlára. Amikor a fizetés sikeresen megtörtént a rendelés létrejön és a felhasználó egy automatizált megerősítő emailt kap, amely tartalmazza a rendelése adatait.



*3.1 Ábra – Sikeres tesztfizetések Stripe weblapon*

Miután a fizetés végbement és létrejöttek a rendelések az utolsó lépés a rendelések követése és állapotuk figyelése. Ezen feladatra létrehoztam egy admin aloldalt, amely az egyes rendelések követésére szolgál. Itt az egyes admin jogosultságú felhasználók képesek hónapokra bontva megnézni az adott hónapban leadott rendeléseket, azok állapotát és tartalmát. Az adminok itt tudják az egyes rendeléseket státuszát állítani (*megrendelve, átadva futárnak, kiszállítva*) és ezzel követni a rendeléseket. Amikor egy rendelés *átadva futárnak* állapotra lesz állítva az adott megrendelő automatikusan kap egy e-mailt a rendeléshez használt e-mail címre ezzel jelezve, hogy az adott rendelés termékei kiszállításra fognak kerülni.

# Funkcionális specifikáció

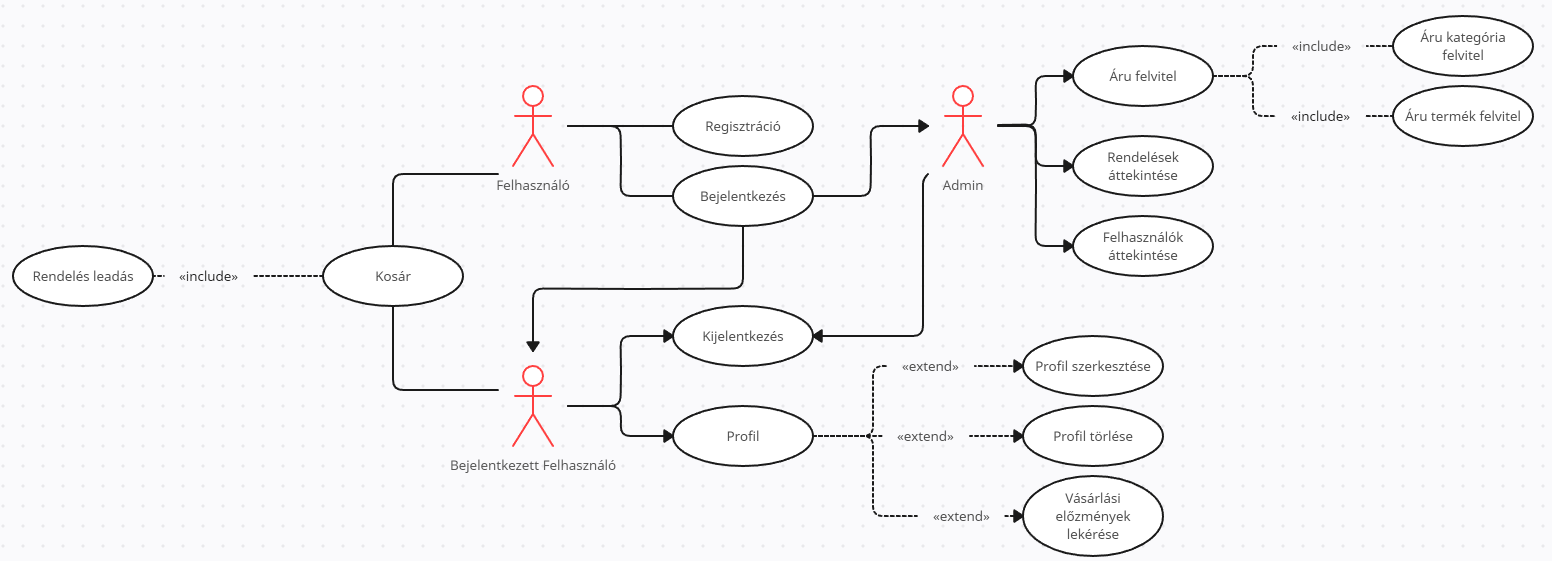
Ezen fejezetben Diplomamunka alkalmazásom működésének főbb elemeit szeretném összefoglalni.

Diplomamunkám során egy olyan webes alkalmazást hoztam létre amivel egy újonnan alapult üzlet, könnyen olcsón létrehozhat magának egy webes felületet felhasználói számára, illetve ezen webes felületen rendeléseket is fogadhat.

A Frontend implementálása React-ban történt aminek, a funkcionalitásainak a megsegítéséhez több package-t is használtam (react-modal, react-redux, react-icons, react-router-dom) melyek segítségével az oldal használata a felhasználó számára könnyebb lett. A Frontend designjának kialakításához TailwindCSS-t használtam.

A backend létrehozásához a Google Firebase nevű szolgáltatását vettem igénybe. Az adatok tárolására a NoSQL alapú Firestore Database felel, míg email küldéshez a Trigger Email from Firestore extensiont használtam. Bankkártyás fizetésekhez a Run Payments with Stripe extensiont probáltam meg implementálni, de miután ezzel több problémába is ütköztem a Stripe végleges implementálásához Google Cloud Functions szolgáltatást használtam.

## Alkalmazás Use-Case Diagramja



*4.1. Ábra – Webshop Use-Case diagrammja*

A felhasználó a webshop nyitóoldalán kezdi meg az alkalmazás használatát, innen kiindulva az oldal tetején megtalálható navigációs sáv segítségével elérheti az egyes termékkategóriák oldalait, a kosarat és a bejelentkezési opciót. Azon esetben, ha a felhasználó még nem regisztrált a webshop honlapján átléphet egy regisztrációs oldalra.

## Regisztráció és autentikáció

Ha a felhasználó regisztrálva akarja használni a weboldalt, akkor a navigációs sávban, talál egy linket a bejelentkezés gomb alatt, amely átviszi őt a regisztrációs oldalra. Ezen az oldalon a felhasználónak meg kell adnia a vezeték, illetve a keresztnevét, az e-mail címét és a jelszavát megerősítéssel. Ezen mezők alapvető ellenőrzéssel rendelkeznek, ezzel garantálva azt, hogy a felhasználó csak megfelelő karaktereket használ illetve, hogy a jelszó megfelel az elvárásainknak. A jelszóval szemben támasztott alapvető elvárásunk, hogy legalább 8 karakter hosszú legyen, tartalmazzon kis- és nagybetűt illetve, hogy legalább egy speciális karaktert. Azon esetben, ha a felhasználó által megadott jelszó nem felel, meg a kívánt feltételeknek arról a felhasználó számára visszajelzéseket nyújtunk. A regisztráció mindaddig nem lehetséges, amíg a két jelszó meg nem egyezik és a regisztrációs feltételeknek nem felel meg.

Azután, hogy a felhasználó megadta az általunk kért adatokat a Firebase Authentikáció segítségével létrehozunk egy új felhasználót e-mail és jelszó bejelentkezési típussal, illetve az authentikáció lefutása után a Google Firestore Databas-ben a *users* kollekcióban létrehozunk az authentikáció által generált egyedi uid-val egy új dokumentumot, mely tárolja a felhasználónk adatait melyeket a felhasználó később a profilja alatt módosíthat.

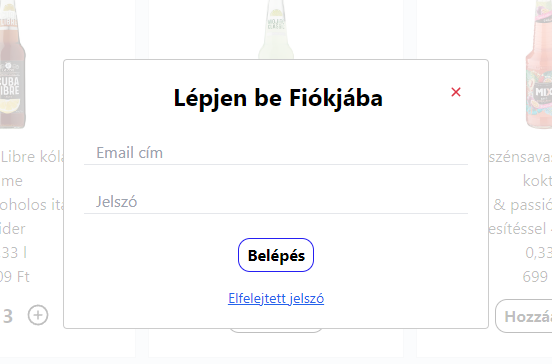
Ha a felhasználó véletlenül lépett volna, a regisztrációs oldalra a regisztrációs oldal alja tartalmaz, egy linket melynek segítségével a felhasználó könnyen visszatérhet a webshop nyitóoldalára.



*4.2 Ábra – Webshop regisztrációs forma*

Ezen regisztráción keresztül csak vásárlói felhasználót lehet létrehozni melyeket később szükség esetén egy már létező admin adminisztratív jogokkal láthat el. A regisztráció végbemenetele után az újonnan létrehozott felhasználó automatikusan beléptetésre kerül és visszatér a webshop nyitóoldalára.

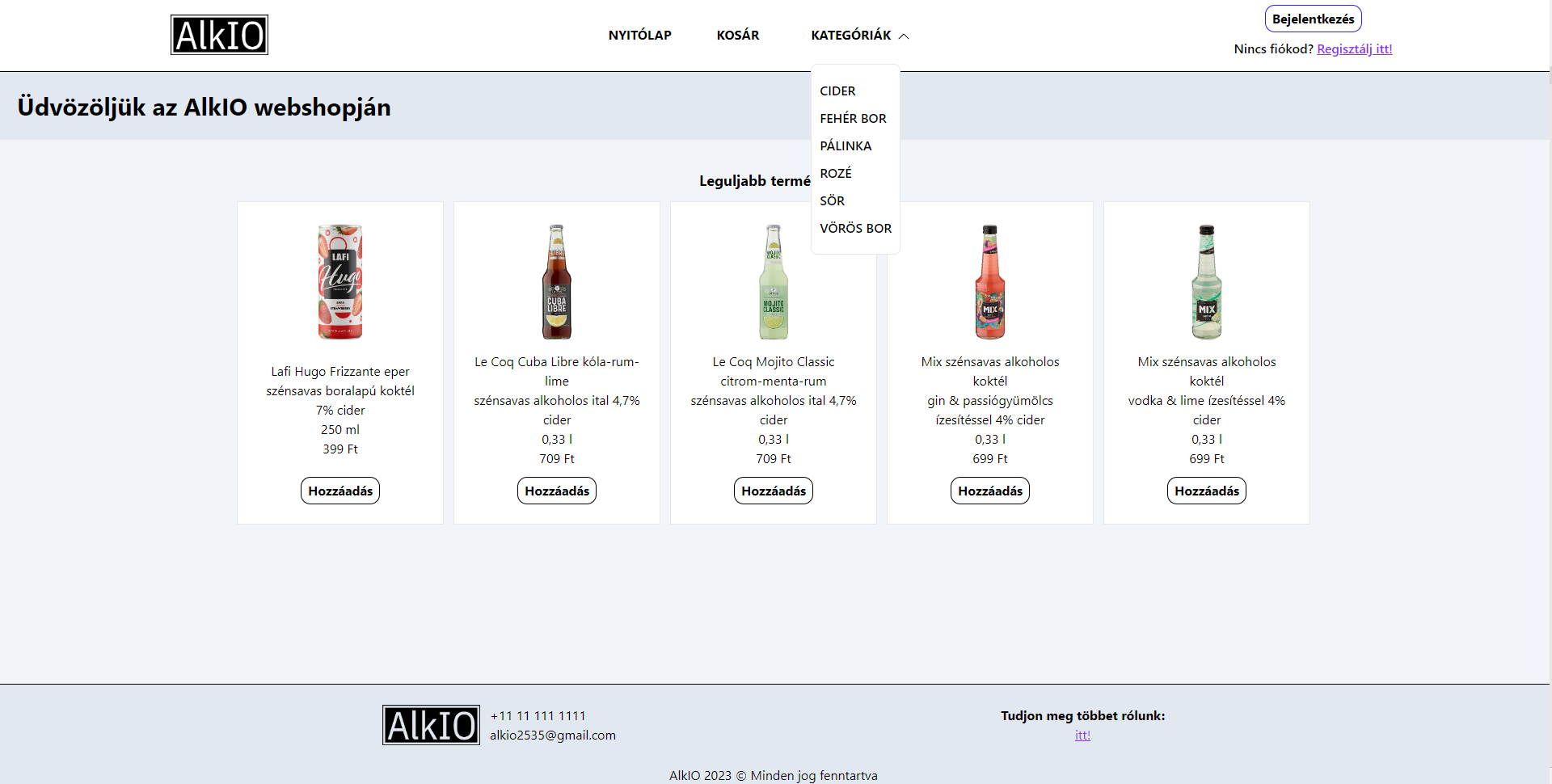
Ha a felhasználó már regisztrált az oldalra a navigációs sávban talál egy bejelentkezés gombot melynek hatására megjelenik az oldalon egy modal melyen keresztül az adott felhasználó az e-mail címe és jelszava megadásával bejelentkezhet az oldalra. A modal megvalósításában a react-modals package volt segítségemre, amivel létrehozhatam egy egyszerű modalt. Azon esetben, ha a felhasználó véletlen nyitotta meg a modalt azt könnyedén bezárhatja a jobb felső sarkában található X gombbal vagy azáltal, hogy a modalon kívüli régióban bárhova kattint.



*4.3 Ábra – Belépési modál webshop nyitó oldalával a háttérben*

## Termék kategória lapok

Minden webshop szerves része a termékek és e termékek kategóriái, melyek alapján a vásárlók megtalálhatják általuk keresett termékeiket. Egy dinamikusan bővülő weblapnál fontosnak ítéltem meg, hogy a weblapon megtalálható termékek kategóriákban legyenek és a navigációs sávban ezen kategóriák láthatóak legyenek. Viszont fontos szempont volt, hogy a dinamikus bővülés ellenére a navigációs bár ne váljon túlzottan zsúfolttá. Ezen probléma megoldására a navigációs sávba egy kategóriák lenyíló menüpontot helyeztem el mely alatt a Firebase *productCategories* kollekciójában található kategórianevek kerülnek megjelenítésre. Így létre tudtam hozni egy dinamikusan bővíthető termékkategória listát, aminek a segítségével a felhasználók elérhetik az általuk keresett termékkategóriák lapjait és azon kategóriák áruit.

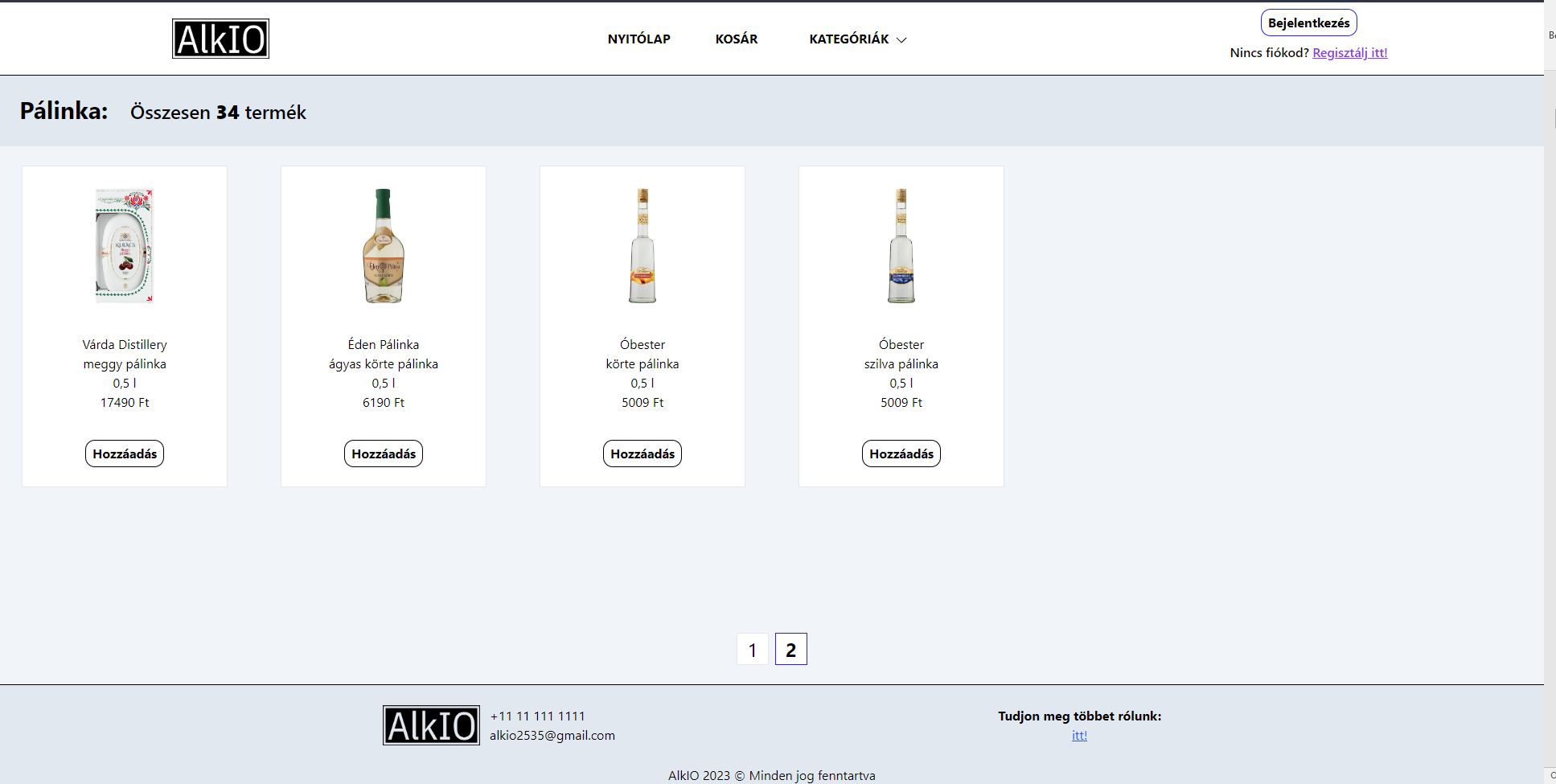


*4.4 Ábra – Termékkategória lista*

Az egyes kategóriák oldalainak a felépítése ugyanazon sémát követi. Az oldal tetején megtalálható a navigációs sáv, alatta egy kiemelt régióban a kategória neve illetve a kategóriában megtalálható összes termék száma, Ezen régió alatt egy méretfüggő gridben megtalálható az adott kategória által tartalmazott termékek kártyái melyeket a felhasználó belehelyezhet a kosarába.

Ennek megvalósításában sokat segített a *react-router-dom* és a *redux* bővítmények melyeknek köszönhetően létrehoztam egy minta oldalt, ami a termékek alapján változik a különböző termék aloldalakra. Ezt egy *ShopLayout* nevezetű keretoldal megvalósításával kezdtem. Ezt a *react-router-dom Outlet* nevezetű könyvtáraknak köszönhetően tudtam létrehozni. Ezen layout lényege, hogy statikusan megjelenít dolgokat a megadott részein az oldalon, míg az *Outlet* az adott oldal tartalmára változik. Ennek köszönhetően nem kell minden egyes oldalra külön elhelyeznem a navbar és footer komponenseket, mert azok minden a layout alatti routing oldalon jelen lesznek.

A másik segítség a kategória oldalak megvalósításába a state managment volt. Minden egyes kategória oldalon ugyanazok a komponensek kerülnek meghívásra az egyetlen technikai különbség közöttük, hogy melyik kategória termékeit kérjük le a state-be tárolásra. Az egyes kategóriák és a benne lévő adatok a Firebase *Products* nevezetű kollekciójában helyeződnek el. Amikor a felhasználó belép, a megfelelő oldalra a weblap innen kéri le a megfelelő termékkategóriához tartozó termékeket és azt redux store segítségével a state-be tárolja. Innen az oldal megnézi, hogy harmincnál több termék található-e az adott kategóriában, és ha igen akkor egy chunk-olást hajt végre, azaz az első harminc elemet áthelyezi egy *shownProductsArray* nevezetű tömbbe. Az oldalon csak ezen tömb elemei kerülnek megjelenítésre, illetve a lap alján megjelenik egy oldalszám jelző gomb melynek segítségével a felhasználó léphet a termékkategória egyes „oldali” között. Azon esetben, ha a felhasználó másik „oldalra” akarna lépni akkor a lap alján levő oldal gombjának megnyomásával a *shownProductsArray* feltöltődik az éppen választott darab termékkel, vagy ha már nincs harminc termék, akkor a maradék termékkel és az kerül megjelenítésre.



*4.5 Ábra – AlkIO webshop Pálinka kategória második „oldal”*

Diplomamunkám megírása során azért döntöttem amellett, hogy az egyes termékkategória oldalakat harmincasával szétszedjem, mert úgy éreztem, hogy a felhasználónak így egyszerűbb áttekintenie a weblap egyes áruit, illetve így a felhasználó nem szembesül azzal a problémával, hogy sokat kell visszatekernie az oldal tetejére, ha másik termék kategóriába, a profiljába stb. akar átlépni.

## Kosár

Minden webshopnak szüksége van egy kosár funkcióra, ami segítségével a vásárlók át tudják nézni milyen termékeket, akarnak venni, azokból hány darabot vesznek, illetve mennyibe fog kerülni a rendelésük összeségében.

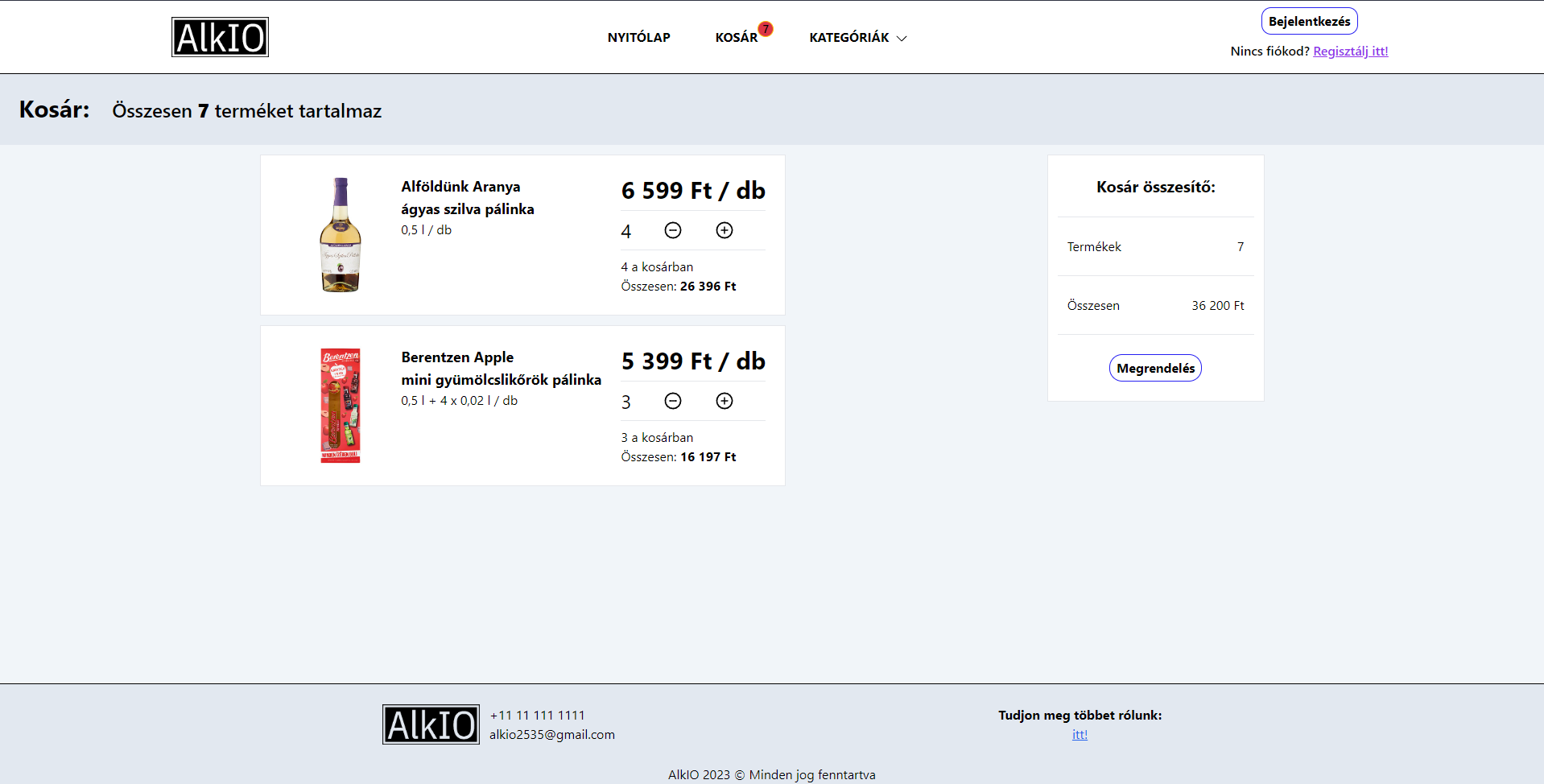
A kosár funkcióm legnagyobb kihívása a nem regisztrált vásárlók voltak. A kosár tartalmát akárcsak a termékeket, illetve a Firebaseről lehívott adatok összeségét redux store segítségével state-ben tárolom, viszont a state a weblap frissülése után kitisztul, így, ha csak itt tároljuk a kosár tartalmát, akkor az minden frissítés hatására elveszik.

Ezen probléma megoldására az első ötlet a kosár mentése a backenden, ami regisztrált felhasználóknál nem lenne problémás, de nem regisztrált felhasználók esetén nem lehetséges, így vagy egy ideiglenes felhasználót kéne létrehoznom, amivel dolgozhatok, vagy más megoldást kell keresni. Az ideiglenes felhasználó létrehozásának legnagyobb problémája, hogy Firebaseben nincs lehetőség ütemezett adattörlésre így, ha létrehozunk egy ideiglenes felhasználót akkor az mindaddig létezni fog amíg azt valaki manuálisan ki nem törli. Ezen gondolatmenet után arra jutottam, hogy a probléma megoldása az, hogy egy sütiben tárolom az adatokat az adott felhasználó böngészőjében. Ezen sütik létrehozásához a js-cookie packaget használom, mely segít a sütik biztosításában, illetve a sütiknek 2 napos életidőt adtam, így nem marad örökre a kosarukban az áru.

A kosár oldal felépítése a termék kategória oldalakhoz hasonlóan egy a navigációs sáv alatti egy kiemelt régióval kezdődik, ami mutatja, hogy a felhasználó a kosár oldalon van, illetve, hogy a kosár jelenleg hány terméket tartalmaz. Azon esetben, ha a kosár tartalmaz termékeket, akkor a navigációs sávon a kosár kategória mellet megjelenik egy indikátor, amely jelzi, hogy a felhasználó már hány terméket helyezett a kosárba.

A kosár oldalon bal oldalt látható a kosárban megtalálható termékek felsorolása, ezek hasonlóan a termékkategóriákban megtalálható árukhoz tízesével vannak chunkokra bontva és közöttük a felhasználó oldaljelző gombokkal léptethet. Az egyes termékek kártyái tartalmazzák a termék adatait, lehetőséget adnak az adott termékből megvenni kívánt mennyiség módosítására illetve jelzik, hogy az adott áruk összesen mennyibe kerülnek.

A kosár oldal jobb oldalán egy összesítő panel jelzi a felhasználó számára, hogy a kosara mennyi terméket tartalmaz illetve, hogy azok mennyibe kerülnek összesen. Ha a felhasználó preferált státuszú, akkor it látja, hogy mennyi kedvezményt kap az összegre illetve itt található a megrendelés gomb is, mely tovább viszi a vásárlót a fizetési opciókhoz és mindaddig disabled amíg nincs áru a kosárban.



*4.6 Ábra – AlkIO webshop kosár termékekkel*

## Fizetés opciók

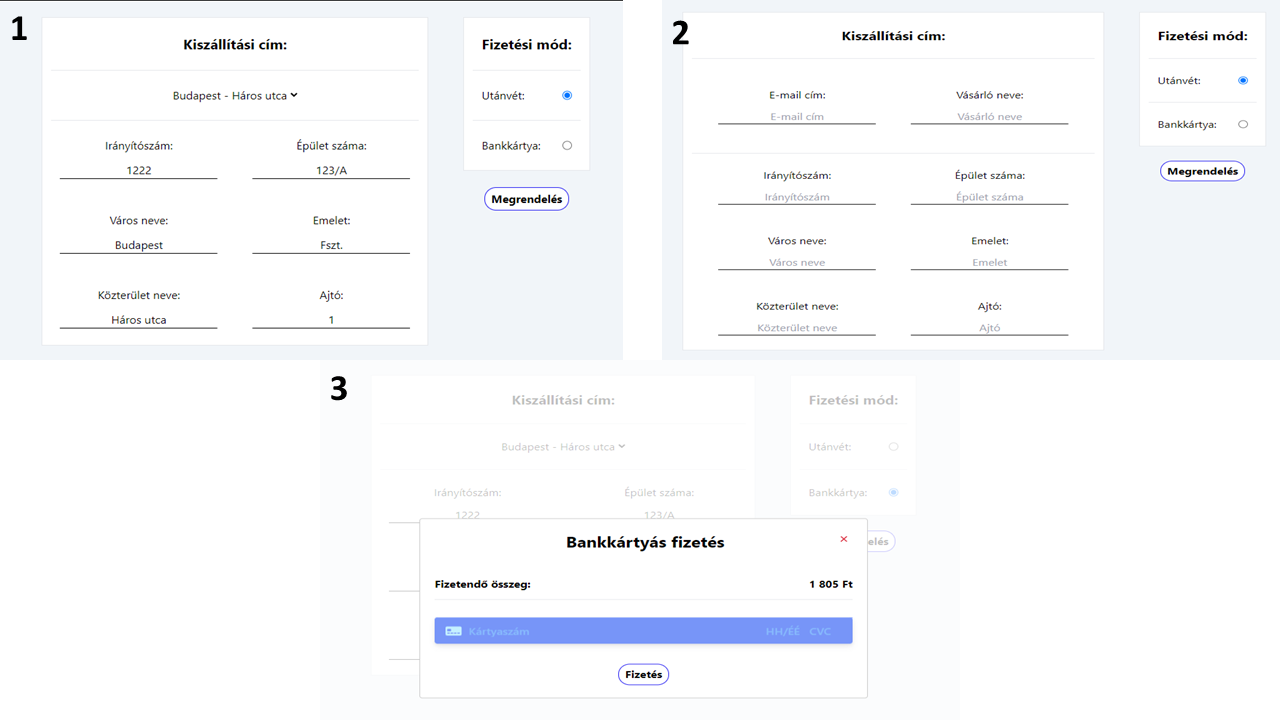
A kosár elkészítése után a logikus következő opció annak a kifizetése. Fizetéshez két opciót terveztem: bankkártyás fizetés és utánvét (készpénz) fizetés. A bankkártyás fizetéshez a Stripe nevezetű szolgáltatás integrációját terveztem.

A fizetési oldalra a kosárból juthatunk át, ha a kosarunk tartalmaz termékeket. Azon esetben, ha nincs bejelentkezett felhasználónk a kiszállítási oldalon egy form lekéri a felhasználó teljes nevét és e-mail címét, illetve a kiszállítási címet. Akkor, ha a felhasználó be van jelentkezve fiókjába a vásárló választhat a profiljához mentett szállítási címek közül vagy megadhat egy új címet ahova a rendelés majd ki lesz szállítva.

Ezen adatok megadása mellet a felhasználó egy rádiógomb segítségével kiválaszthatja, hogy milyen fizetési móddal szeretne vásárolni. Az utánvét esetében a vásárlónak nincs további feladata, a rendelés felvételre kerül egy *orders* kollekcióba Firebasen ami az összes megrendelést tartalmazza illetve ha a felhasználó be van jelentkezve akkor az adott felhasználó profiljába is mentésre kerül. Megrendelésre kattintás esetén a Firebase a Trigger Email from Firestore extension segítségével kiküld egy emailt a felhasználónak, ami a rendelés tartalmát részletezi, mint például, hogy mit tartalmaz a rendelése, mennyibe fog kerülni és mikorra várható a rendelés megérkezése. Az email elküldéséhez létrehoztam egy Google fiókot az adott minta webshopomnak és a Google SMTP szolgáltatása segítségével ezen a fiokon keresztül küldöm ki az e-maileket.

A bankkártyás fizetéshez a Stripe nevezetű szolgáltatást próbáltam meg integrálni. Ezt egy a Stripe által fenntartott és fejlesztett Run Payments with Stripe nevezetű extensionnel lehetséges megoldani Firebaseben. Ezen extension képes automatikusan létrehozni stripe vásárlókat a Firebaseben tárolt felhasználókból feltéve, hogy azok egy kollekcióban az Authentication által generált *uid*-val vannak tárolva, illetve automatikusan képes a Stirpeban tárolt árukat átszinkronizálni egy Firebase kollekcióba. Ezen adatok segítségével a Stripe képes teljes tranzakciókat létrehozni bankkártyás fizetéshez. Normál backend esetén a Stripe képes payment intenteket létrehozni általunk megadott pénzösszegekkel, viszont sajnálatos módon Firebaseben az intentek csak a Stripe által beszinkronizált árukra lehetségesek. Ezen ok miatt sajnos kiderült, hogy az egyetlen módja annak, hogy egy webshop áruit igy bankkártyával kifizessük az, ha egyesével megveszi a vásárló az árukat annyiszor ahány db árut akart az adott termékből venni, ami eléggé hamar belátható, hogy egy fizetéshez akár tíz, húsz tranzakciót is létrehozhat. Ez se a vásárlónak se a webshopnak nem előnyös, hiszen így minden egyes tranzakció során a Stripe levon valamennyit az átutalt pénzből és az adóztatás is annyiszor történik meg ahány külön tranzakció volt, arról nem is beszélve, hogy nincs olyan vásárló, aki hajlandó több tízszer megadni újra bankkártya adatait. Így megállapítottam, hogy a Firebasen keresztül történő Stripe fizetés csak akkor lehetséges és éri meg egy weblap számára ha subscription terveket árulnak (pl.: egy streaming oldalon).

Ezen probléma megoldását végül a Google Cloud Functions szolgáltatása szolgáltatta. Ezzel képesek vagyunk létrehozni és müködtetni általunk létrehozott backend funkciókat, vele Google Firebase alkalmazásunkat akár teljes fullstack projekté is átalakíthatjuk. A Google Cloud Function segítségével képes voltam a Stripe saját api-jának segítségével saját Stripe payment intentek létrehozására, így már csak annyi volt a dolgom, hogy létrehozzak a frontenden egy Stripe Fragmentet ami begyűjti a felhasználó bankkártya adatait, majd azokat tovább küldi a Cloud Function-nek. Ezután a Cloud function létrehozza a szükséges payment intentet, azt továbbítja a Stripe szolgáltatásnak és így megtörténik a bankkártyás fizetés.



*4.7 Ábra – AlkIO rendelés megerősítési oldal felhasználóval (1) és nélkül (2),*

*illetve kártya adatok megadása (3)*

## Profil

A bejelentkezetett felhasználók neve megjelenik a navigációs sávon, erre kattintva elérhetik a saját profiljukat. A felhasználó profilja két Fő részre bontható, az első tartalmazza a profil adatait. Itt a felhasználó látja, és ha akarja, megváltoztathatja a profilhoz tartozó nevet, illetve kiszállítási címeket vehet fel, ahova majd később az általa rendelt termékek kiküldésre kerülnek, illetve ezen címekhez tartozó adatokat módosíthatja is. Ha a felhasználó úgy döntene, hogy nem akarja, hogy az adott webshop fenntartsa a profilját és tárolja adatait, itt lehetősége van a profil törlésére is.

Azon esetben, ha a felhasználónak már voltak korábban rendelései, akkor lehetősége van azok megjelenítésére is ezen az oldalon. A rendelések fül alatt a felhasználó láthatja korábban leadott rendeléseit, azok árát, illetve az egyes rendelések lenyithatók, ahol látható a rendeléshez tartozó áruk és azoknak a mennyisége.

## Admin felületek

Ha a weboldalra egy admin felhasználó jelentkezik be a navigációs sáv egy új admin navigációs elemmel bővül. Akár csak a termékkategóriáknál ez a navigációs elem is lenyitható és alatta megtalálhatóak a különböző admin oldalak.

### Termékek

Az admin termékek oldal alatt egy admin felhasználó képes termékek és termékkategóriák felvitelére, módosítására, és törlésére a webshopban Ezen az oldalon baloldalt megtalálható termékkategóriák nevei egymás alatt. Ha az admin szeretne, újat felvinni a sor alján egy plusz gombbal megnyílik egy modal aminek a segítségével megteheti. Létező kategória nevének módosítása hasonlóképpen ezzel a modallal történik. A termék oldal jobb oldalán egy táblázatban megtalálhatóak az adott kategóriához tartozó termékek. A termékek a termék kategóriák termékeihez hasonlóan húszasával chunk-olva oldalakra vannak bontva Az admin felhasználó itt képes módosítani a termékek adatait, illetve új termékek felvitelére melyek modallal történnek meg. Ha az admin másik kategóriát akar választani a bal oldalt felsorolt termékkategóriák nevére kell kattintania.

### Rendelések

Ezen oldalon az admin hónapokra bontva láthatja az adott hónapban bejött rendelések adatait táblázat formában. A rendelések szintén chunk-olva vannak húszasával külön oldalakra ezzel megkönnyítve az átláthatóságot. Itt az admin átállíthatja a rendelés állapotát, ami lehet megrendelve, átadva futárnak, illetve kiszállítva. Ha egy rendelés állapota „átadva futárnak” -ra változik rendelésleadáshoz hasonlóan egy értesítő e-mail kerül kiküldésre az adott rendeléshez tartozó e-mail címre ezzel jelezve a vásárlónak a rendelése állapotát.

### Felhasználók

Az admin ezen az oldalon láthatja az weblapra regisztrált összes felhasználót. A felhasználók egy select segítségével admin és vásárlói felhasználókra vannak bontva. Az admin ezen a weboldalon képes az egyes felhasználók szintjét átállítani (sima és preferált, a preferált 15% kedvezményt kap minden vásárlásból az oldalon), illetve képes admin jogokat adni felhasználóknak, illetve egyéb adminoktól ezen jogokat képes megvonni.

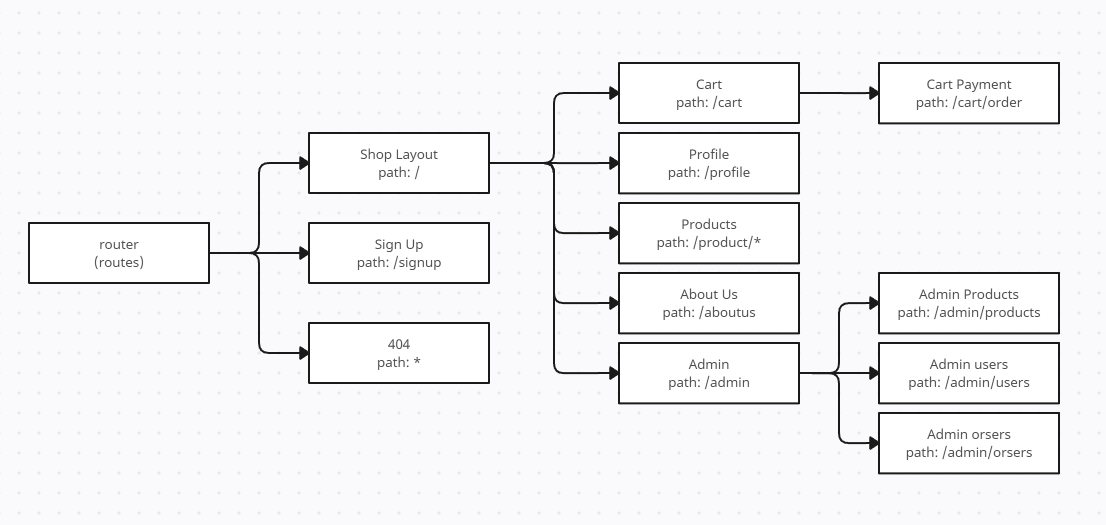
## Rólunk oldal

Minden weboldal alján megtalálható egy footer amiben megtalálható az üzlet elérhetőségei, illetve egy link, ami tovább viszi a felhasználót a rólunk oldalra. Ezen oldal tartalmaz egy rövid leírást az adott üzletről illetve egy kontakt formot ahol a felhasználó megadhatja a nevét, telefonszámát és email címét illetve írhat egy szöveges üzenetet, amit a lap alján való küldés gombbal a Firebase elküldi a webshop email fiókjára, ezzel is egy kontakt lehetőséget biztosítva a felhasználónak.

## Routing

A weblap routing-ja a *react-router-dom* könyvtár segítségével lett megvalósítva. Ennek segítségével létrehoztam egy *RouterProvider* komponenst, ami magába foglalta a webshopot és kapott egy *router* nevű paraméter-t, amely tartalmazta az egyes routok adatait.

A routing három fő path-ból áll. Az első a ShopLayout element-re mutató / path amely magába foglalja a weblap összes aloldalát (kosár, profil, termékek stb.). A második a */signup* amely a weblapra történő regisztrációért felelős. A harmadik a \* path amely a 404 oldalra visz azon esetben, ha nem létező path-re akarunk menni.



*4.8 Ábra – Alkio webshop routing*

# Architektúra

A diplomamunkámhoz készített alkalmazás két részre bontható. Az első a felhasználók által elérhető webes alkalmazás és az ehhez az adatokat szolgáltató szerver, amit a Google Firebase szolgáltatása biztosított.

A frontend weblap megvalósításához a React-ot és a TailwindCSS-t választottam, amivel egy modern single-page weboldalt hoztam létre. A tervezés és a megvalósítás alatt próbáltam ügyelni a könnyed átláthatóságra ezzel is elősegítve későbbi fejlesztéseket.

A Backendhez a Google által biztosított Firebase szolgáltatást választottam. A szolgáltatás előnyei, hogy egy bizonyos mértékig ingyenes, van beépített autentikáció, NoSQL adatbázissal rendelkezik és sok extension-je megkönnyíti az egyes felmerülő problémák megoldását. A Stripe-al történő fizetés megvalósításához a Google Cloud Functions szolgáltatását választottam.

A projekt forráskódját Githubon tároltam, a verziókövetést is ezen keresztül bonyolítottam le.

A webshop host-olását a Google Firebase szolgáltatás biztosította.

# Adatmodell

## Felhasználó

A weblapra történő regisztrálással létrejövő felhasználó adatait tárolja. Létrehozásakor a felhasználó regisztrációjakor történik meg, ilyenkor az összes adattag létrehozásra kerül, de nem mindegyik tárol tényleges adatot még.

* *id* – string

Az authentikációval létrejött uid a Firebasből

* *addresses* – array

Egy tömb, ami tartalmazza a felhasználóhoz mentett szállítási címek adatait, regisztrációkor egy üres tömbként jön létre

* + *postalCode* – az adott cím irányítószáma
  + *city* – az adott címhez tartozó város
  + *street* – az adott címhez tartozó utca
  + *building* – az adott cím épület száma
  + *floor* – az adott címhez tartotó épület szintje (ha társasház)
  + *doorNumber* – az adott címhez tartozó épület ajtaja (ha társasház)
* *email* – string

Az adott authentikációhoz használt e-mail cím

* *familyName* – string

Az adott személy családneve

* *surname* – string

Az adott személy keresztnevei

* *lvl* – number

Az adott felhasználó tipusa (sima vagy preferált) 1 vagy 2, a nem regisztrált felhasználóé 0-a

* *admin* – bool

Az adott felhasználó admin-e

* *orders* – object

Az adott felhasználó által leadott rendeléseket tartalmazó objektum, az objektum kulcsa a rendelés időpontja, a felhasználó létrejöttekor üres

* + *address* – a rendeléshez használt cím
  + *cartProduct* – a rendelésben szereplő termékek
    - *id* – a megrendelt termék azonosítója
    - *name* – a megrendelt termék teljes neve
    - *quantity* – a megrendelt termék mennyisége
  + *price* – a rendelés ára (Ft-ban)

## Termék

A weblapon árusított termék adatait tárolja, admin felhasználó hozhatja létre, a leíráson és a mennyiségen és az src-n kívül minden adat kötelező megadású.

* *id* – string

Az adott termékhez tartozó azonosító, a termék létrehozásakor generáljuk

* *name* – string

A termék neve

* *number* – number

Az adott termékből a raktáron rendelkezésre álló mennyiség

* *price* – number

Az adott termék ára (Ft-ban)

* *quantity* – string

Az adott termék tárolási képessége (pl.: italoknál)

* *type* – string

Az adott termék típusa

* *description* – string

Az adott termék leírása

* *src* – string

Kívülről importált kép esetén a kép elérési útvonala

## Rendelés

A leadott rendelésekhez tartozó adatok, létrejöttükkor az összes benne levő adat létrejön.

* *address* – string

Az adott rendelés kiszállítási címe

* *cartProducts* – object

Az adott rendeléshez tartozó termékek adatai

* + *id* – a megrendelt termék azonosítója
  + *name* – a megrendelt termék teljes neve
  + *quantity* – a megrendelt termék mennyisége
* *date* – string

A rendelés dátuma

* *email* – string

A megrendelő e-mail címe

* *name* – string

A megrendelő teljes neve

* *orderState* – string

A megrendelés állapota

* *price* – number

A rendelés teljes ára (Ft-ban)

* *user* – string vagy null

Ha bejelentkezett felhasználó rendelt, akkor annak az *uid*-je egyébként null

# Jogosultságkezelés

Minden egyes adatbázis egyik legnagyobb kihívása a biztonságos adatkezelés, vagyis, hogy csak az férhessen hozzá az adatainkhoz, akit tényleg akarjuk, hogy hozzáférhessen. Szerencsére ezen probléma megoldására számomra a Google Firebase szolgáltatása beépített megoldásokat nyújt.

A Google Firebase Firestore szolgáltatásához tartozik egy *Rules* nevezetű szolgáltatás rész, melynek köszönhetően könnyedén lekezelhetjük, hogy az egyes adattábláinkhoz kik és milyen helyzetben férhetnek hozzá. Az egyes hozzáférési szabályok beállíthatók külön dokumentumokra, dokumentum részekre, azok írására, olvasására, létrehozás, törlésére stb. Ezek használatával adatbázisom jogosultságkezelése megoldhatónak bizonyult.

Diplomamunkám során a hozzáférést három szintre bontottam, ezek a nem regisztrált felhasználók, a regisztrált felhasználók és az adminok. Az egyes adattáblák hozzáféréseit ezek a szintek határozzák meg. Annak érdekében, hogy a nem regisztrált felhasználók is képesek legyenek vásárolni a termék és termékkategória tábláknak olvashatóknak kell lennie mindenki számára, míg a rendelések táblába rendelésüket fel kell tudniuk vinni.

Az egyes regisztrált felhasználóknak tudják az előbb említett adatok olvasása mellet a saját profiljuk adatainak eléréséhez, illetve azok módosítására vagy akár törlésére. Minden egyéb adat módosításához csak az adminoknak van módosítási hozzáférése, de emellett az adminok képesek más felhasználók adatainak módosítására is.

# Fontosabb kódrészek

## Authentikáció

Az autentikációért felelős kód három főbb részre bontható a regisztrációra a bejelentkezésre és a kijelentkezésre. Ezek mind a Firebase auth metódusait használják.

### Regisztráció

Regisztrációkor a Firebase *createUserWithEmailAndPassword()* metódusa hívódik meg a felhasználó által megadott e-mail és jelszó segítségével. A feliratkozáskor előre ellenőrizzük, hogy a jelszó megfelel az általunk kívánt elvárásoknak, miután a felhasználó létrejött létrehozunk neki egy felhasználót a FireStore adatbázisban.

### Bejelentkezés

Bejjelentkezéskor a Firebase *signInWithEmailAndPassword()* metódusa hívódik meg a felhasználó által megadott jelszóval és e-mail címmel. Sikeres bejelentkezés esetén lekérjük az adott felhasználóhoz tartozó adatokat. Ha a felhasználónak nem emlékszik a jelszavára a bejelentkezési modal-on van lehetősége a jelszójának lecserélésére. Ehhez a Firebase *sendPasswordResetEmail()* metódusa hívódik meg a felhasználó e-mail címével. Ez küld a felhasználónak egy e-mailt, aminek a segítségével új jelszó adható meg.

### Kijelentkezés

Kijelentkezéskor a Firebase *signOut()* metódusa kerül meghívásra. Ez töröli a Firebase által használt frissítési tokeneket illetve a meghívás után a stateben tárolt felhasználói adatokat eldobjuk.

## State managment

A webshop működésének fontos része a redux store-ba tárolt adatok és az ehhez tartozó state managment. Az egyes termékek, felhasználók, kosár adatait a redux storeban tároljuk. A storeban lévő adatok lekérése *AsyncThunk*-ok által történik amiket *dispatch()* hívásokkal érünk el különböző helyeken a kódban.

A példa során a termékekhez létrehozott productSlice-ot mutatom be, de a többi adat tárolása is ilyen módon történik. Az adatok tárolása egy Slice létrehozásával kezdődik, aminek van egy neve, kezdeti állapota reducerei és extra reducerei.

export const productSlice = createSlice({

  name: "productItems",

  initialState,

  reducers: {},

  extraReducers: (builder) => {

    builder.addCase(getNewProduct.pending, (state) => {

      state.newProductLoading = true;

    });

    builder.addCase(getNewProduct.fulfilled, (state, action) => {

      state.newProductsObj = action.payload;

      state.newProductsArray = Object.values(action.payload).sort(

        compareNewProducts

      );

      state.newProductLoading = false;

    });

    builder.addCase(getNewProduct.rejected, (state) => {

      state.newProductLoading = false;

    });

Extra reducereket használok az adatok lekérésére. Ennek a megoldásnak több előnye is van, a legfontosabb, hogy az egyes adatelérésekhez külön le tudom kezelni a három állapotot, amik a pending, a fulfilled, és a rejected. Amikor az adatlekérés elkezdődik akkor egy loader boolt átállítok *true*-ra, ennek a segítségével tudok az adott oldalon pl.: egy spinnert megjeleníteni, amíg az adatok töltődnek, majd, mikor az adatok megérkeznek akkor azokat a statebe tárolom és a loading-ot *false*-ra állítom. Ha az adatok megérkezésében hiba történik megjeleníthettek pl.: toast segítségével egy hibaüzenetet ezzel is jelezve a felhasználónak, hogy hiba történt.

Az *extraReducerek asyncThunkokra* figyelnek amelyeket dispatchek segítségével lehet a kódon belül meghívni. Ezek a thunkok azok, amik ténylegesen elvégzik az adatok lekérését az adatbázisból.

export const getProduct = createAsyncThunk(

  "productItems/getProduct",

  async (categoryName) => {

    const response = await firebase

      .firestore()

      .collection("Products")

      .doc(categoryName)

      .get();

    return response.data();

  }

);

Ennek a megoldásnak másik nagy előnye, hogy ha valamiért le kell cserélni a backendjét az alkalmazásnak, akkor elég itt átírni az egyes adatlekéréseket a program működésének más részén nem kell komolyan változtatni. A thunk meghívása kódban annak nevével és a megfelelő paraméterekkel történik pl.: *dispatch(getProduct(categoryName))*

Miután megvannak a Slice-aink a megfelelő thnuk-okkal azokat hozzá kell, adjuk a storehoz. Erre több megoldás is létezik én a *reduxjs/toolkit combineReducers()* metódusát választottam.

const rootReducers = combineReducers({

  ordersReducer,

  usersMapReducer,

  userReducer,

  productCategoryReducer,

  productReducer,

  cartReducer,

});

const store = configureStore({

  reducer: rootReducers,

});

Ha a reducerjeink hozzá vannak adva a storehoz és a storet az app köré wrappeltük akkor az egyes az egyes Sliceok statejeiben tárolt adatok elérése nagyon egyszerűvé vállik. A megfelelő oldalakon először *dispatch()* segítségével lehívom az adott oldalhoz kellő adatokat pl.: termékeket. Amig azok megérkeznek a statebe levő loading bool segítségével egy spinnert jelenítek meg a képernyőn, és amint megérkeznek az adatok a *useSleector()* metódus segítségével lehívom őket a state-ből és megjelenítem őket.

const productItems = useSelector((state) => state.productReducer);

const productCategory = useSelector((state) => state.productCategoryReducer);

Ennek a megoldásnak köszönhetően az oldal reszponzív és könnyen átköthető lesz.

## Kosár

A kosár működésé state-re alapszik, a *cart* state tartalmazza az adott vásárló kosarának tartalmát. Minthogy a state az oldal frissülésével visszaáll az *initialState*-be ezért itt fontos, hogy az egyes adatokat ne csak statebe tároljuk, hanem cookie-ban is. A weblapra történő belépéskor megnézzük, hogy van-e *cart* névű sütink eltárolva a gépen, és ha igen akkor az abban lévő adatokat átadjuk a statenek.

  useEffect(() => {

    dispatch(getProductCategories());

    if (Cookies.get("cart") !== undefined) {

      dispatch(getCart(JSON.parse(Cookies.get("cart"))));

    }

    dispatch(getNewProduct());

  }, [dispatch]);

Ha a felhasználó megrendeli a kívánt termékeket, mind a statet visszaállítjuk *isitialState* állapotába, míg a sütit töröljük. Emellett egy előre megírt template emailt is elküldünk a felhasználónak, amit a Firebase *mail* nevű kollekciójába történő adat felvitellel ahonnan a Trigger Email with Firebase extensiuon automatikusan elküldi az általunk felállított SMTP szerveren keresztül az emailt (a mi esetünkben a webshophoz létrehozott Google fiók Google SMTP funkciójával).

## Fizetés

A fizetés megvalósítását a Stripe integrálásával valósítottam meg. Ehhez első megközelítésem a Firebase Run Payments with Stripe extensionje volt, de hamar be kellett látnom, hogy ez sajnos nem alkalmasa arra, amire a Stripeot használni akartam, így másik megoldást kellet keresnem.

A Stripe integrálásának megvalósítását végül a Google Cloud Functions szolgáltatása nyújtotta. A megvalósításhoz létrehoztam egy Cloud Functiont amely a *requestbe* várja a Payment Mehtod *id* paraméterét, illetve várja a fizetendő *amount* összeget. Itt fontos megjegyezni, hogy a Stripe habár támogatja a Forintot, mint valutát a fizeteni kivánt *amount* mennyiség utolsó két számjegye a Payment Intent létrehozásakor tizedes értékké válik, azaz az összes érkező pénzértékünk utolsó két számjegye 0 kell, hogy legyen ugyanis a Forintnak technikailag nincs váltó valutája. A létrehozott *payment* backend endpointot egy általánosabb api endpointhoz kötöttem annak érdekében, ha további fejlesztéseket akarnánk tenni, akkor elég legyen a frontendről az api endpointon belüli megfelelő function-t meghívnunk.

app.post("/payment", cors(), async (req, res) => {

  let { price, id } = req.body;

  try {

    await stripe.paymentIntents.create({

      amount: price,

      currency: "huf",

      description: "AlkIO Vásárlás",

      payment\_method: id,

      confirm: true,

      return\_url: "https://webshop-d4135.firebaseapp.com/",

    });

    res.json({

      message: "Payment successful",

      success: true,

    });

  } catch (error) {

    res.json({

      message: "Payment failed",

      success: false,

      error,

    });

  }

});

exports.api = functions.region("europe-west1").https.onRequest(app);

A frontenden a Stripe integrálása egy *Elements* nevű provider-be történő foglalással kezdődik. Az *Elements* provider elvárja, hogy a *stripe* nevű változójába megadjuk a Stripe fiókunkhoz tartozó *Publishable key* nevű api kulcsot. Ezután a Stripe használható a provider által körbefogott komponensekben.

        <Elements stripe={stripePromise}>

          <CartStripeModal

            setOpenModal={setOpenModal}

            openModal={openModal}

            paymentInfo={paymentInfo}

            contact={contact}

          />

        </Elements>

A bankkártya adatok lekérését egy a Stripe által biztosított *CardElement* nevű komponens biztosítja. Ez a komponens automatikusan lekéri, egy bankkártyás fizetéshez szükséges összes információt majd ezeket továbbítja fizetéskor a *createPaymentMethod* metódusnak, ami ezek segítségével létrehoz egy payment method *id*-t amit ezután *axios* segítségével tovább tudunk küldeni a Cloud Function-nek, ami létrehozza a szükséges payment intentet, amit a Stripe azután automatikusan tranzakcióként lekezel.

  const handleSubmit = async (event) => {

    event.preventDefault();

    if (!stripe || !elements) {

      return;

    }

    const { error, paymentMethod } = await stripe.createPaymentMethod({

      type: "card",

      card: elements.getElement(CardElement),

    });

    if (!error) {

      try {

        const { id } = paymentMethod;

        const response = await axios.post(

          "https://europe-west1-webshop-d4135.cloudfunctions.net/api/payment",

          {

            price,

            id,

          }

        );

Ha a fizetés sikeres volt az adott megrendelést hozzáadjuk a megrendelésekhez, illetve küldünk egy emailt a felhasználónak, hogy megrendelését köszönjük. Ellenkező esetben jelezük a felhasználónak, hogy fizetési kísérleté sikertelen volt és, hogy próbálkozzon újra.

# Tapasztalatok, továbbfejlesztési lehetőségek

A diplomamunkám elkészítése alatt számos új technológiát ismertem meg és számos technológiába nyertem bele mélyebb belelátást. Megismerkedtem a Google Firebase szolgáltatásának rejtelmeivel illetve a Stripe tranzakciós szolgáltatás működésével. Mindemellett már korábbról ismert technológiákban is újabb ismereteket sajátítottam el, ilyen a React redux és a TailwindCSS.

Diplomamunkám fejlesztése folyamatos ötletek sorával járt számomra, az egyes problémák megoldására és azok általánosítására. Ennek hála sokszor leragadtam olyan dolgok kidolgozásával, amik nem okoztak komoly változást a weblapon, de egy nagyobb méretű project kereteiben valószínűleg hasznosak lennének. Úgy érzem sok olyan tanulsággal gazdagodtam mely a jövőben segíteni fog munkámban.

Diplomamunkám továbbfejlesztését további Cloud Function-ök megírásával kezdeném, így a projekt teljes fullstack fejlesztéssé válhatna, és így hatékonyabbá és biztonságosabbá is lehet az adatkezelése a projektnek. Ezenfelül remek fejlesztési opció lehet még különböző adminisztrátori szintek létrehozása, azok jogosultságának kezelése és ezzel extra backoffice funkciók létrehozása. Egyéb fejlesztési opció lehet még komplexebb leértékelés kezelés az egyes termékekre.

# Felhasznált irodalom, források

1. React JS - <https://react.dev/>
2. TailwindCSS – <https://tailwindcss.com/>
3. Visual Studio Code – <https://code.visualstudio.com/>
4. Stripe – <https://stripe.com/en-hu>
5. Github – <https://github.com/>
6. mdn web docs – <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
7. Google Firebase - <https://firebase.google.com/>
8. React icons – <https://react-icons.github.io/react-icons/>
9. React modal – <https://reactcommunity.org/react-modal/>
10. Redux – <https://react-redux.js.org/>
11. React spinners – <https://www.davidhu.io/react-spinners/>
12. React router – <https://reactrouter.com/en/main>
13. Google Cloud Functions – <https://cloud.google.com/functions/docs>
14. Axios – <https://axios-http.com/docs/intro>

A forráskód elérése: <https://github.com/InquisitorSolen/webshop>

# Nyilatkozat

Alulírott, **Fürstahl Bence Dániel**, **Mérnökinformatikus** szakos hallgató, kijelentem, hogy a szakdolgozatban ismertetettek saját munkám eredményei, és minden felhasznált, nem saját munkából származó eredmény esetén hivatkozással jelöltem annak forrását.

Dátum (Szeged, év, hó, nap) \_Fürstahl Bence Dániel\_\_

aláírás