

Szegedi Tudományegyetem

Informatikai Intézet

SZAKDOLGOZAT

Pál Krisztián Zoltán

2023

Szegedi Tudományegyetem

Informatikai Intézet

Italárusító webshop alkalmazás beépített játékkal

Szakdolgozat

Készítette:

Pál Krisztián Zoltán

programtervező
informatikus szakos
hallgató

Témavezető:

Dr. Bilicki Vilmos

egyetemi adjunktus

Szeged

2023

Feladatkiírás

A szakdolgozat I. és szakdolgozat II. tárgyak keretein belül egy különféle alkoholos és alkoholmentes italokat árusító webáruházat alkot meg a hallgató, amelyben az általános webshopokban található funkciók mellett egyedi funkciók megvalósítása is a feladat része. Többek között beleértve a fizetési rendszer integrálását, emellett az árucikkek közötti gördülékeny böngészést.

A szakdolgozat teljesen öncélú, viszont felhasználhatósága kiterjedhet akár más sajátosan megalkotott, de ettől eltérő árucikkekkel foglalkozó weboldalak fejlesztésének alapjaként. Az elkészített webalkalmazás meg kell, hogy állja a helyét az interneten fellelhető más webshopokkal szemben.

Amióta az internet egyre elterjedtebbé vált, egyre népszerűbbek lettek a webshopok, melyek segítségével a felhasználók saját igényük szerint gyorsan tudnak böngészni a termékek tárházában. Mindig szükség van, és a jövőben is szükség lesz ezen platformokra, melyek megkönnyítik, kényelmesebbé teszik az emberek életét. Az áruház tulajdonosok számára is hatalmas előny, hiszen rengeteg fogyasztóhoz el tudják juttatni termékeiket a helyi áruházaiak termelésén túlmenően.

A hallgató feladatát képezi egy újabb webshop megalkotása. A weboldal azt az egyediséget viseli, hogy egy egyszerű beépített játék alkalmazás juttathatja kedvezményekhez a vásárlókat. Így egyedi módon juttatjuk a felhasználót pénzügyi előnyhöz, emellett a webshopon is több időt tölt el, ami azt eredményezi, hogy új termékeket ismer meg, több energiát fordít a vásárlásra. Ezen üzleti logika elősegítheti, hogy a felhasználók több árucikket, avagy gyakrabban rendeljenek, de még így is szimpatikusabbá téve számukra a webalkalmazás rendszerét.

Tartalmi összefoglaló

- **A téma megnevezése:**

Italárusító webshop alkalmazás beépített játékkal

- **A megadott feladat megfogalmazása:**

A feladat egy olyan webalkalmazás fejlesztése, amely dinamikus, felhasználóbarát, és nem utolsó sorban italok megrendelését teszi lehetővé (természetesen csak képzeletben). A webáruház kényelmes böngészést biztosít a felhasználónak a neki tetsző termékek között. Különlegessége a beépített játék, mely a felhasználóknak lehetőséget biztosít minden egyes nap bizonyos értékű kedvezmények szerzésére.

- **A megoldási mód:**

A keretrendszer által biztosított szolgáltatások jó kihasználása (egységekre szervezés, materialok használata, stb.), a Firebase könnyű elérése és ezáltal a backend megteremtése, a Stripe fizetési rendszer integrálása a Firebase Cloud Functions használatának köszönhetően egy dinamikusan működő hostolt alkalmazás kifejlesztését tette lehetővé.

- **Alkalmazott eszközök, módszerek:**

Az alkalmazás a Visual Studio Code nevű fejlesztői környezetben készült. A logika TypeScript nyelven íródott, a megjelenítés HTML és SCSS használatát követelte meg. A Tailwind könyvtár használata megkönnyítette az elemek stílusainak beállítását, emellett javított az átláthatóságon. A fejlesztésben egy multiplatformos keretrendszer, az Angular 14-es verziója nyújtott segítséget. Adatbázist, tárolót, sőt autentikációt és hostingot biztosít a Firebase felhőalapú szolgáltatás. Ennek a része a Firebase Cloud Functions, amely lehetővé teszi a Stripe fizetési rendszer által biztosított fizetési funkciót, és a játékban kapott életeri pontok visszaállításának időponthoz kötését. A projekt forráskódja GitHubon elérhető.

- **Elért eredmények:**

Az alkalmazás bármikor elérhető az interneten, forráskódja mintaként felhasználható.

- **Kulcsszavak:**

webshop, angular, dinamikus, játék, firebase, stripe

Tartalomjegyzék

Feladatkiírás	1
Tartalmi összefoglaló	2
Tartalomjegyzék	3
1. Bevezetés	4
1.1 Problémafelvetés, motiváció	5
2. Alkalmazott eszközök, technológiák bemutatása	7
2.1 Fejlesztői környezetek	7
2.1.1 Visual Studio Code	7
2.1.2 Böngésző fejlesztői mód	8
2.2 Programozási nyelvek	8
2.2.1 TypeScript	9
2.2.2 HTML	9
2.2.3 SCSS	9
2.2.4 Tailwind	10
2.3 Keretrendszerek	10
2.3.1 Angular	10
2.4 Firebase	11
3. Terület áttekintés	12
3.1. Webshop alkalmazások terület áttekintése	12
3.2. Piackutatás	13
3.3. Piackutatás értelmezése	14
4. Architektúra	17
4.1. Futtatási környezetek	17
4.2. Helyszínek	17
4.3. Eszközök	18
5. Funkcionális specifikáció	20
5.1. Bejelentkezés és regisztráció	20
5.2. Böngészés a termékek között	21
5.3. Termékek részletei	22
5.4. Felhasználói visszajelzések	22
5.5. Kosárkezelés	23
5.6. Integrált fizetési rendszer	24
5.7. Kedvezmények szerzése (Beépített játék, egyszer használatos link)	26
5.8. Felhasználói fiók kezelése	27

5.9. Bejegyzések írása	28
5.10. Admin funkciók	29
5.11. Routing	30
5.12. Use-Case diagramok	31
6. A megvalósítás lépései fontosabb kódrészletek kiemelésével	33
6.1. Komponensek és modulok	33
6.2. Modellek (Interfészek)	41
6.3. Backend kódok	43
6.4. Jogosultságkezelés Firebase-ben	45
7. Tapasztalatok, továbbfejlesztési lehetőség	47
7.1. Tapasztalatok, nehézségek	47
7.2. Továbbfejlesztés	47
Összegzés	48
Irodalomjegyzék	48
Nyilatkozat	49

1. Bevezetés

A webáruházak egyre nagyobb népszerűségnek örvendenek. Az embereknek hatalmas kényelmi funkciót biztosít, hogy személyesen el sem szükséges menniük a megvásárolni kívánt termékekért. Ennek okán az igény erre folyamatosan csak nő. Ezen webalkalmazás többek között sablonként is szolgálhat újabban felépülő, hasonló célokra készített oldalak felépítésekor.

A cél az, hogy minél gyorsabb és minőségibb weboldalakat fejlesszünk, hisz az emberek türelmetlensége csökkentheti a vállalkozók szeme előtt lebegő bevételt, ha nem töltenek elég időt az adott weboldalon. A figyelemfelkeltés céljából ezen webáruház a szokványos technikák mellett egy beépített játékot tartalmaz, mellyel a felhasználó akciókra tehet szert. Ezzel elérhetjük azt, hogy több időt töltsön az oldalon, ezáltal több neki tetsző terméket tárhatunk a szeme elé, ami vásárlásra ösztönzi. Valójában ez egy nyerő-nyerő szituáció, hisz így elérjük, hogy a felhasználó többet vásároljon így profitot termelve a vállalkozónak. Látható, hogy kevesebb lesz a profit az akciók miatt, mégis a rendszer kiépítése maximális nyereséget termel, ha a termékek árai megfelelően behatároltak.

A kész alkalmazás rendkívül felhasználóbarát, nem igényel bármiféle szaktudást a kezeléshez. Egy laikus felhasználó is könnyen boldogulhat egyedül. Az admin felület kiépítése hasonló könnyedséget von maga után, ugyanis fontos, hogy az üzemeltető is gördülékenyen tudja működtetni az alkalmazást. A dinamikus megvalósítás lehetővé teszi, hogy könnyeden, mondhatni akármennyi ideig fusson az alkalmazás. Hisz a hosting-nak köszönhetően a világhálón bármikor megtekinthető. Az esetleges hibák jelzése szintén beépített feladata az alkalmazásnak.

1.1 Problémafelvetés, motiváció

A szakdolgozatom témája az előző fejezetben felvázolt italárusító (mely természetesen más termékekkel is foglalkozhatna) webalkalmazás sablon elkészítése. Fel van vértézve beépített játékkal, mely legalább egy játékmódot tartalmaz. Korszerű technológiát használ, és folyamatosan bekerülő új funkciókat alkalmaz.

A webáruház hozzájárulhat ahhoz, hogy szélesebb körben elterjedjen ez a fajta piaci rés kihasználás, miszerint egy játékkal csalogatjuk oda, sőt inkább tartjuk ott az oldalunkra tévedő felhasználókat.

A motivációm tehát annak a problémának a megoldására ad egy lehetőséget, miszerint a türelmetlen felhasználók túl hamar elhagynak egy adott webáruházat anélkül, hogy vásárolnának. Ez sokszor köszönhető a magas várakozási időnek, míg betöltenek bizonyos részletek, a nem túl felhasználóbarát felületnek vagy épp a magas áraknak. Mindezek megoldása mellett még kedvezményt is biztosítok a vásárlók számára, amennyiben hajlandóak játszani és/vagy regisztrálni. A játékok köztudottan addikciót okozhatnak, és ezen függőség kialakulását lehetővé téve akár napi szinten visszatérő vásárlóink is lehetnek.

2. Alkalmazott eszközök, technológiák bemutatása

2.1 Fejlesztői környezetek

A fejlesztői környezetek (IDE - Integrated Development Environment) olyan szoftvereszközök, amelyek megkönnyítik a programozók munkáját. Olyasfajta segítséget nyújtanak, amik jelentősen rövidíthetik egy-egy alkalmazás elkészítésének az idejét, így produktívabb és gyorsabb munkát eredményezve egyaránt.

Általában egy fejlesztői környezet több mindent magába foglal, többek között tartalmaz fordítót, futtatókörnyezetet, és a szoftver fejlesztéséhez elengedhetetlen szövegszerkesztőt. Mondhatni egy integrált munkakörnyezet, melyben a programozók írhatnak, szerkeszthetnek, tesztelhetnek és hibát is kereshetnek a kódban, mindezt átlátható módon. Javaslatokat tesz, amikor elkezdünk gépelni egy adott parancsot, változónevet, függvénynevet, sőt képes a gépelési hibáink kijelzésére is.

A fejlesztői környezet sok mindent rejt magában, mint például szintaktikai kiemelés, kódszerkesztési segítség, hibakeresési eszközök, verziókezelési integráció és sok más. Különböző programozási nyelveket és platformokat segítő variációi léteznek, melyekből a programozó kiválaszthatja a saját igényeit és preferenciáit legjobban kielégítőt. A teljesség igénye nélkül néhány: Visual Studio Code, Eclipse, Notepad++, PyCharm, WebStorm stb.

2.1.1 Visual Studio Code

Az általam használt TypeScript nyelvhez a leginkább illeszkedő fejlesztői környezet a Visual Studio Code. Ez egy keresztplatformos eszköz, mely rengeteg programozási nyelvet és platformot támogat. Ezen felül sok egyéb fejlesztést segítő funkciót és bővítményt kínál. A Microsoft fejlesztette ki ezt a terméket, mely a kényelem megteremtését szolgálja, mint más fejlesztői környezetek, de tudomásom szerint a legszélesebb körű az alkalmazása.

2.1.2 Böngésző fejlesztői mód

Ha egy böngészőben aktiválni szeretnénk a fejlesztői módot csupán annyit kell tennünk, hogy az aktív ablakban lenyomjuk az F12 billentyűt, vagy jobb gombbal kattintunk az oldalon és kiválasztjuk a „Vizsgálat” menüpontot. Ekkor előjönnek különféle fejlesztést segítő eszközök (a frontend fejlesztőknek nagy segítségére). A legtöbb manapság használatos böngészőben már fellelhető ez a beépített funkció. Mindez lehetővé teszi, hogy az általunk éppen fejlesztett vagy akár mások által készített weboldalakat elemezzük. Könnyen látható, hogy ez gyors hibakeresésre is alkalmas.

Több panellel rendelkezik, amelyek különböző funkciókat tesznek lehetővé: elemek vizsgálatát, a DOM szerkesztését, a CSS (stílusok) vagy akár a JavaScript módosítását, a hálózati lekérdezések monitorozását, a konzol üzeneteinek megtekintését és sok más eszközt, amelyek segítségével fejlesztés közben interaktívan dolgozhatunk a weboldallal.

2.2 Programozási nyelvek

A programozási nyelvek afféle mesterséges nyelvek, melyeket azért hoztak létre, hogy az emberek utasítások megadásával tudjanak kommunikálni a számítógéppel. Értelemszerűen a gépek sokkal gyorsabbak, logikusabbak, mint az ember. Viszont közvetlenül nem képesek kommunikálni az emberekkel, hisz egy számítógép nem tud egyik emberi nyelven sem, mint ahogy nekünk sem egyszerű megérteni a számítógépek nyelvét. Szerencsére a programozási nyelvek megoldást nyújtanak ennek a problémának a megoldására.

A programozási nyelveknek saját szabályaik, szintaxisuk és funkcióik vannak. Ebben hasonlítanak az emberi nyelvekhez. Fontos, hogy egy programozási nyelv a programozni tanulók számára érthetővé váljon, így biztosít könnyebb kommunikációt a számítógéppel. Általuk a fejlesztők írhatnak utasításokat, melyek sorozatából épülnek fel a programok és az alkalmazások, amelyek képesek végrehajtani különféle feladatokat.

2.2.1 TypeScript

A fejlesztés során kiemelt fontosságú, hogy milyen programozási nyelvet használunk, ugyanis előnyt jelenthetnek bizonyos programozási nyelvek az egyes feladatokat megvalósító alkalmazások elkészítésében. A TypeScript pedig rendkívül jól illeszkedik a Frontend fejlesztést segítő keretrendszerekhez, mint például az Angular, amit jelenleg használtam. A webalkalmazás logikája ezen a nyelven íródott így több, mint a fejlesztés felét kitette ennek a nyelvnek a használata.

A TypeScript egy nyílt forráskódú, statikusan típusos objektumorientáltnak is használható programozási nyelv, mely a JavaScript interpretált programozási nyelv hiányosságait terjeszti ki. Javarészt a Microsoft csapata fejlesztette, hogy megkönnyítsék a programozók munkáját. A TypeScriptet a fordító először JavaScript kóddá fordítja át, melyet már képes értelmezni a böngésző.

2.2.2 HTML

A HTML (Hypertext Markup Language) egy leíró nyelv, ami a weboldalak struktúrájának felépítésére szolgál. Az oldal csupasz vázát felépíthetjük a HTML által előre definiált tag-párokkal, melyek egymásba ágyazásával egyedi struktúrát alakíthatunk ki. Értelmszerűen a Frontend fejlesztést segítő keretrendszereknek szüksége van valami ehhez hasonlóra. Többek között az Angular is a HTML használatát igényli azon felül, hogy valamennyi logikát is bevihetünk a használatával a HTML kódba.

2.2.3 SCSS

Az SCSS (Sassy Cascading Style Sheet) egy stílusleíró nyelv, ami kiterjeszti a CSS-t, melybe előre definiált kulcs-érték párok megadásával tudjuk a HTML által felépülő struktúrának a megjelenését változtatni. Az SCSS könnyebb és hatékonyabb CSS stíluslapok írását teszi lehetővé, hisz kisebb logikai funkciókat biztosít, mint például a változók létrehozása. Egy preprocessor dolgozza fel az SCSS fájlokat a fordítás részeként, és így hoz létre a háttérben CSS fájlokat.

2.2.4 Tailwind

A Tailwind CSS egy modern és rendkívül testre szabható CSS keretrendszer, amelyet kifejezetten a gyors és hatékony webfejlesztéshez terveztek. A Tailwind ideológiája az, hogy a CSS osztályok közvetlenül definiálják a stílust, így minimalizálva az előre definiált stílusokat és maximalizálva a fejlesztői rugalmasságot. Az osztályok lehetővé teszik a pontos kontrollt az elrendezési és stíluselemek felett, miközben könnyen testre szabhatók és újrahasznosíthatók. A Tailwind széles körben elterjedt a fejlesztők körében, akik értékelik a gyors és intuitív munkafolyamatot, valamint a könnyen karbantartható és szabványos kimenetet.

2.3 Keretrendszerek

A keretrendszerek arra szolgálnak, hogy a funkcióikat felhasználva lerövidítsék a fejlesztésre szánt időt. Nagyobb projektek esetén rendkívül hasznos, hiszen ahogy a nevéből is adódik egy keretet ad az egész projektnek. Lényegileg olyan funkciókat tartalmaznak, melyeket alap esetben szintén le kellene fejleszteni. Viszont a keretrendszerek adottak, és így bizonyos dolgok fejlesztésén időt spórolhatunk. Ennek a haszna, hogy sokkal inkább koncentrálhatunk azon területek implementálására, amely az adott alkalmazásunk lényegét képezi.

2.3.1 Angular

Az általam is használt nyílt forráskódú webes keretrendszer az Angular, melyet a Google fejlesztett ki, és adott ki 2010. október 20-án még Angular JS néven modern webalkalmazások fejlesztésének céljából. A célja, hogy felhasználóbarát, jól skálázható, hatékony alkalmazást fejlesszünk általa. Eleinte a JavaScript nyelvet használta a logikájához, de már az Angular 2-től áttért a JavaScriptet kiterjesztő TypeScript nyelv használatára. Biztosít adatkötést a felhasználói felület és a logika között, könnyű navigációt a Router segítségével, és a sablonok segítségével dinamikus és interaktív felhasználói felületek hozhatók létre. Komponensekbe, avagy modulokba szervezhetjük az alkalmazásunkat a jobb átláthatóság végett. A segítségével nem csak webalkalmazások építhetők fel, hanem például mobilalkalmazások, de többek között

még asztali alkalmazások is. Ezen keretrendszer 14-es verzióját használtam fel a webalkalmazás elkészítésekor.

2.4 Firebase

A Firebase a Google által kifejlesztett népszerű és teljeskörű fejlesztői platform. A gyors, hatékony fejlesztésben, tesztelésben, sőt az üzemeltetésben is nagy segítségére van a fejlesztőknek. Az eszközei és szolgáltatásai rengeteg segítséget nyújtanak. Idetartozik a teljesség igénye nélkül a felhasználókezelés és hitelesítés, a felhőalapú tárolás, a valós idejű adatbázis, de akár még az alkalmazásnaplózás is. A Firebase könnyen integrálható webes-, avagy mobilalkalmazásokba egyaránt. Az Angular keretrendszer a TypeScript által könnyen képes alkalmazni a Firebase-t ezzel megkönnyítve a programozási munkákat.

3. Terület áttekintés

3.1. Webshop alkalmazások terület áttekintése

A webshopok afféle digitális központok, ahol a fogyasztók könnyedén böngészhetnek, vásárolhatnak, és az árukat a kényelmükből kifolyólag otthonukba szállíttathatják anélkül, hogy el kellene hagyniuk a házat. Ez lényegi hasonlóság minden webshop között, így az én megoldásom is erre épül. Ezen dinamikus online kereskedelmi platformok minden egyes szegmense fontos szerepet játszik a felhasználói élmény, az üzleti hatékonyság és az innováció terén.

A terület áttekintésének fontos része maga a marketing. A webshopok különféle módszerekkel igyekeznek vásárlásra bírni a fogyasztói társadalom tagjait. Személy szerint olyan egyedi megoldásokat választottam, melyek újító jelleggel hatnak a webáruházakra. Ugyanis a piackutatás elvégzése során nem találtam máshol beépített játékot, avagy blogírási lehetőséget. A marketing kontextusában a terület kiterjed a digitális térre is, ahol a versenytársak, a fogyasztók és a technológia együttesen formálják az online kis- és nagykereskedelmet.

A webshop terület áttekintése nem csupán statisztikák összegyűjtése és táblázatok elkészítése a versenytársakról. Fontos volt átemelni a már jól bevált módszereket a mások által épített alkalmazásokból. Ilyen a könnyű böngészés, a kifinomult felhasználói élmény kialakítása, a gördülékeny vásárlási lehetőség. Ez egy átfogó elemzés vizsgálatát takarja, mely a piaci trendek, fogyasztói szokások, és az aktuális technológiai fejlemények feltárására irányul. A cél nem csupán az volt, hogy ismereteket szerezzek a jelenlegi piaci helyzetről, hanem hogy ezekre az ismeretekre alapozva a webshopom maximális hatékonysággal és innovációval szolgálhassa ki a vásárlókat.

A következő alfejezetekben részletesen fogok foglalkozni a webshopok által kínált lehetőségek tárházával, és megvizsgáljuk, hogyan járult hozzá a virtuális italokat árusító webshopom fejlesztéséhez és sikeréhez a világháló már használatban levő webáruházainak részletes kielemezése. A piackutatás során szerzett ismeretek és az áttekintés során feltárt lehetőségek összekapcsolásával teremtettem meg azt a szilárd alapot, amelyen az online kiskereskedelem sikeresen megélhet és fejlődhet a digitális térben.

3.2. Piackutatás

	Saját alkalmazás tervek	Kormorán Bottleshop	DrinkCentrum	TotalDrinks	Drink Online
Linkek:	https://trinkydinky-webshop.web.app/	https://alkoholshop.hu/	https://drinkcentrum.hu/	https://totaldrinks.hu/	https://www.drinkonline.eu/
Interneten elérhető	Igen	Igen	Igen	Igen	Igen
Üdvözlő/Életkor ellenőrző üzenetek implementálása	Nem	Igen	Igen	Igen	Nem
Egyedi témák	Igen	Igen	Igen	Igen	Igen
Regisztrációs lehetőség	Igen	Igen	Igen	Nem	Nem
Böngészés a termékek között	Igen	Igen	Igen	Igen	Igen
Alkoholmentes italok forgalmazása	Igen	Nem	Igen	Igen	Igen
URL átirányítások	Igen	Igen	Igen	Igen	Igen
E-mailben való értesítések rendelés esetén	Nem	Igen	Igen	Igen	Igen
Regisztráció nélküli rendelés lehetőség	Igen	Igen	Igen	Igen	Igen
Megjegyzés/Komment írás a termékekhez	Igen	Igen	Nem	Nem	Nem
Értékelés a termékekhez	Igen	Nem	Nem	Nem	Nem
Saját blog bejegyzés írása	Igen	Nem	Nem	Nem	Nem
Termékhez hasonló termékek megjelenítése	Igen	Igen	Igen	Igen	Nem
Újdonságok gyors elérése	Nem	Nem	Igen	Nem	Nem
Webshop értékelésének megtekintési lehetősége	Nem	Igen	Nem	Nem	Nem
A vásárláshoz Facebook fiók használata	Nem	Nem	Nem	Nem	Igen
Többféle betűtípus alkalmazása több szín mellett	Igen	Nem	Nem	Igen	Igen
Logók, képek szereplése	Igen	Igen	Igen	Igen	Igen
Többnyelvűség	Nem	Igen	Nem	Nem	Nem
Beépített játékkal kedvezmény szerzése	Igen	Nem	Nem	Nem	Nem

3.2.1. ábra: Piackutatás táblázat

3.3. Piackutatás értelmezése

A piackutatás alapján értelemszerűen kivehető, hogy mindegyik webshop megtalálható az interneten. Ez a lényegi értelme az összesnek, hogy kiszolgálja az arra tévedő felhasználókat a saját termékeikkel.

Mivel az alkohol vásárlás egy olyan tevékenység, melyhez szükséges az, hogy a vásárló életkora elérje a 18-at, ezért szokványos, hogy az adott webshop az oldal megnyitásakor felteszi a kérdést, hogy felnőttek vagyunk-e már. Ugyanis ez jogosulttá tesz minket arra, hogy szeszes italt fogyaszthatunk. Ezen felül vagy ehelyett mindenféle (nem felnőtt árucikket tartalmazó oldalak esetén is) szokás egy felugró üdvözlő ablak megjelenése, amely szimplán csak köszönti a lehetséges vásárlót. Ez személyesebbé teheti az élményt, ugyanis egy boltban való vásárlás esetén is lezajlik egy oda-vissza köszönés a fogyasztó és az eladó között. Én mégis úgy gondoltam, hogy ez nem kimondottan fontos funkció. Sőt egyes emberek még idegesítőnek is találják ezt. Ezen okokból nem építettem be az alkalmazásomba üdvözlő ablakot.

Minden webshop habár ugyanazzal a céllal készül, mégis egyedi tematikát, egyedi stílust, egyedi megjelenést kölcsönöz, amellyel megpróbálja felkelteni a lehetséges vásárlói figyelmét. Sokszor megfigyelhető, hogy egy-egy webshop teljesen máshogy épül fel, mint a többi. Ennek a gyakorlati haszna sokszor tényleg a figyelemfelkeltés, emellett a minél gördülékenyebb böngészés megteremtése, melynek többféle megoldása is felhasználóbarát élményt teremthet.

A böngészés a termékek között sokszínű lehet. A leghatékonyabb az, hogy ha könnyebben elszeparálhatóak egymástól a termékek. Akár kategóriák, színek, árak, nevek szerint. A név szerinti szűrés a legpontosabb, ugyanis lehetőségünk nyílik ténylegesen csak arra a termékre keresni, amit kívánunk megvásárolni. Ezen okokból alkalmaztam én is kategóriák, árak, nevek szerinti rendezést, és nevekre való szűrést.

A regisztráció és ezáltal a bejelentkezés lehetősége sokszor új kapukat nyit meg a fogyasztók előtt. Vannak webshopok, ahol csak a regisztrációt követően van lehetőségünk rendelni, valahol el tudjuk menteni bankkártya adatainkat, vagy esetleg kedvezményekhez férhetünk hozzá. A regisztráció egyfajta autentikációt biztosít, mely által könnyen azonosíthatjuk az egyes felhasználókat.

A megjegyzések írása, és az értékelések nagyban segítik a webáruházakat abban, hogy visszajelzéseket kapjanak a termékeik milyenségéről. Ennek segítségével tudják,

hogya mi az, ami bevált, és mi az a termék, amin módosítani kell, esetlegesen teljesen el kell törölni. A vásárlók között kialakuló kommunikáció arra is szolgál, hogy egymást megsegítsék a termék milyenségét, minőségét, hasznosságát illetően. Sok esetben a megjegyzések és értékelések hozzáadása is a regisztrációt követően elérhető funkció, de egyes esetekben lehetőség van akár anonim hozzáfűzésre, értékelés nyújtásra is.

Saját blog bejegyzés írása ugyan nem szokványos egy webáruház esetén. Viszont úgy gondoltam, hogy ez színesítheti, és feldobhatja azt. Ez a funkció csupán arra szolgál, hogy plusz információhoz juttassa a felhasználókat, emellett, hogy lehetőséget kapjanak gondolataik bővebb kifejtésére összeszedettebb formában.

A termékek keresését illetően nagy segítségre lehet az, hogy a megnyitott termék oldalán megjelenjenek hozzá igencsak hasonló más termékek is. Ezzel motiválttá téve a fogyasztókat az esetleges új termékek megismerésének irányába. Így emelkedő vásárlási számra tehet szert a webshop tulajdonosa.

Néhány weboldal kiemelt fontosságúnak tartja az újonnan érkező termékek nyomatékos feltüntetését. Általában jól látható helyeken helyezik el, és szinte egyből az oldal megnyitásakor találkozhatunk velük. Az új termékeknek még szüksége van marketingre, hogy az információ a létezésükről elterjedjen, így növelve a kínálatot.

Az értékelés nem csak egyes termékekre irányulhat. Az én esetemben például lehetőség nyílik a blogbejegyzések értékelésére is. Ezáltal könnyen kikövetkeztethető, hogy a benne található információ mennyire releváns, hasznos. Így arra sarkallva az oldalra tévedőket, hogy érdemes-e elolvasni egy adott bejegyzést vagy sem. Más weboldalakon olyan funkciót is fellelhetünk, amellyel a teljes rendszer egészét értékelhetjük. Így szintén mások segítségére lehet, hogy az adott webshop mennyire hiteles, mennyire jók a termékei.

A vásárlás indításához a legtöbb esetben szükséges létrehozunk egy fiókot a weboldal felületén. Ebben segít a regisztráció. Máshol akár fizethetünk úgy is, hogy csak egyszeri megoldásként adjuk meg a fizetéshez szükséges adatainkat. Hasznos módszert biztosít az, amikor nem terheljük a felhasználókat azzal, hogy újdonsült fiókot hozzanak létre. Csupán csak hozzákapcsolják az egyik közösségi média fiókjukat, mint például a Facebook-ot, avagy a mindenki által előnyben részesített Gmail fiókot.

Az egyedi témákhoz kapcsolódóan a weboldalak rengeteg lehetőséget kínálnak felépítésük folyamán. A készítőnek hatalmas paletta áll rendelkezésére színek, betűtípusok, és más dizájn elemek használatát illetően. Ezek szintén arra szolgálnak, hogy olvashatóvá, felhasználóbaráttá, egyszerűen jól kinézővé tegyék a szemünk elé

tároló oldalakat. Ide tartoznak a weboldalt színesítő képek is, ugyanis sok esetben egy jól elkészített kép, avagy az oldal csábító emblémái, logói, egyéb díszai felhívják a figyelmet. Egy italárusító webshop esetén a termékek íze csak akkor derül ki, amikor már megvásárolták őket, viszont a kinézetük sokszor döntő fontosságú lehet a fogyasztók szemében. Mindemellett a kisebb, könnyen elhelyezhető logók könnyű marketing fogások.

Fontos kiemelni, hogy a weboldalak látogatói milyen nyelven beszélnek, milyen írott szövegeket értenek meg. Kiváló lehetőség az, ha lehetősége van a felhasználónak egyetlen kattintással módosítani a kívánt nyelvre a weboldal tartalmának teljes egészét. Ugyanis a megértés döntő fontosságú a rendelések számát illetően. Senki nem fog megrendelni olyan terméket, amiről azt sem tudja mi az. Habár az internet korában főleg otthoni rendelés esetén már könnyen fordíthatóvá válnak az egyes szövegrészek a különféle fordítóprogramoknak köszönhetően. Azonban a fogyasztói társadalom tagjai rendkívül türelmetlenek és lusták tudnak lenni ezen a téren. A többnyelvűség viszont megoldást nyújt erre a problémára. Lehetővé teszi, hogy a felhasználók kedvük szerint böngészhessenek akár a saját, akár azon a nyelven, amit ismernek. Ezért célszerű az angol nyelvet alapul venni itt Európában, ugyanis közel ez a legismertebb, legtöbb ember által beszélt nyelv.

A webshopok nem szokványos eleme, hogy tartalmazzanak játékokat. Véleményem szerint viszont nagyszerű marketing fogás, amivel az oldalunkra csábíthatjuk nap, mint nap a fogyasztók egy bizonyos hányadát. Főképp, ha jutalom is társul a játékkal elvesztegetett percekért. A játékok köztudottan addikciót válthatnak ki egyes emberekből, és ezt a függőséget kihasználva feltételezhető, hogy akár a játék élvezete hatására megnő a piaci kereslet a termékek irányába. Ez nem egy elismert módszer, sőt nem is alkalmazott, viszont egy próbálkozást megér, hogy hogyan hat ez a fogyasztói réteg gondolkodására. Vajon ezen funkció tényleg meghozza a várt hatást, avagy csak plusz idegesítő elem a vásárlási folyamatban? Erre a kérdésre keresem a választ.

4. Architektúra

4.1. Futtatási környezetek

Alapvetőleg egy Angular projekt létrehozásakor 2 fájl jön létre, amik a környezetért felelnek. Az egyik a *development*, másik a *production*. Az *angular.json* fájlban van lehetőség módosítani, hogy mely lehetőségekben mely környezet legyen érvényben. Az *ng build* parancsot kiadva a *cmd*-be elérhetjük, hogy létrejöjjön a *dist* mappa, amiben az alkalmazás lefordított állományai szerepelnek. Ezt követően a *firebase deploy —only hosting* parancs (természetesen amennyiben már a *firebase*-t beállítottuk a projektben) kirakja az alkalmazást a világhálóra. Esetemben az *ng build* alapértelmezettként a *production* környezetet használja. Viszont helyi futtatás esetén amikor az *ng serve* parancsot használom (ez a parancs le is fordítja és futtatja is a projektet) akkor a *development* környezet lesz érvényben. Egy apró helyen használtam ki ezt. A fizetési oldal, amely egy külső *express* szerveren fut lokálisan és a *hosting*olt formában is ugyanoda mutat. Viszont amennyiben sikeresen fizetünk, avagy visszalépünk akkor a környezet alapján tudja az alkalmazás, hogy a lokális vagy a kitelepített sikeres vagy sikertelen fizetési információs oldalra navigáljon. A környezetért felelős TypeScript fájlokban elhelyeztem egy *production* boolean típusú attribútumot. Ezzel ellenőrzöm, hogy hova is kell navigálni az adott esetben:

```
url: environment.production ? 'https://trinkydrinky-webshop.web.app' : 'http://localhost:4200'
```

4.1.1. ábra: Futtatási környezet szerinti navigáció

4.2. Helyszínek

A különféle helyszíneket maguk a modulok adják. Ezek funkcióit, milyenségét, darabszámát a későbbi fejezetekben részletesen taglalom. Egy kívül álló helyszín van, az pedig a többször emlegetett *express* szerveren futtatott checkout oldal, amely kommunikál a *Stripe* fizetési rendszerrel.

4.3. Eszközök

Az eszközök közé sorolandó építőkövek lehetnek a komponensek, szolgáltatások, guard-ok, pipe-ok, melyek segítségével felépül a teljes alkalmazás.

A komponensek arra szolgálnak, hogy az egyes modulokban fellelhető újrahasznosítható részeket külön egységekbe szervezzük. Például a main oldalon 4 sor látható, amelyek mindegyike valójában ugyanaz a komponens csak újrahasznosítva. Más értékhez hozzájutva az input direktíván keresztül más termékeket jelenít. Emellett a blogok és a kommentek mindegyike is ugyanazokon a komponenseken alapul, azonban így újrafelhasználhatóvá váltak az adott kódrészletek. A komment komponensnek például két input direktívája is van:

```
<span *ngFor="let comment of this.comments">
  <app-comment [comment]="comment" [productId]="this.actProduct?.id"></app-comment>
</span>
```

4.3.1. ábra: ngFor és komment komponensek felhasználása a kódban

A szolgáltatások arra szolgálnak, ha két modulban szükség van azonos funkcionalitásra akkor ne végezzünk kód duplikációt, hanem így kiszervezve felhasználhatjuk mindegyik modulban az adott kódrészletet függvények formájában. Ide sorolhatók többek között a Firebase adatbázisából, tárolójából való adatlekérések. Ugyanis például a termékek adatait több modulban is felhasználhatjuk, frissíthetjük többféle módon, törölhetjük, újat kreálhatunk.

```
updateQuantity(productId: string, productQuantity: number){
  const productRef = this.afs.collection<Product>(this.collectionName).doc(productId);
  return productRef.update({ quantity: productQuantity });
}
```

4.3.2. ábra: Termékmennyiség frissítése függvény a megfelelő service-ben

A guard-ok segítségével testreszabhatóbbá tehetjük a routing-ot. Ugyanis bizonyos feltételeket köthetünk bizonyos elérési útvonalakhoz. Ilyen példa a csak bejelentkezett

```
const user = JSON.parse(localStorage.getItem('user') as string);
if (user){
  return true;
} else {
  this.router.navigateByUrl('/login').then(() => {
    this.toastr.info("Jelentkezz be a játékkal szerzett akciókért");
  }).catch(error => {
    console.error(error);
  });
  return false;
}
```

4.3.3. ábra: canActivate függvény átirányítással

felhasználók számára elérhető oldalak az adott guard *canActivate* függvényét kihasználva.

A pipe-ok pedig arra szolgálnak, hogy bizonyos adattípusokat megfelelő formátumban jelenítsünk meg a weboldalon. Két egyedi csővezetékét alkalmaztam. Az egyik azért felel, hogy a kedvezményes árak megfelelő formátumban jelenjenek meg, a másik pedig a *number* típusként tárolt dátumokat konvertálja tényleges dátummá a transform függvény segítségével.

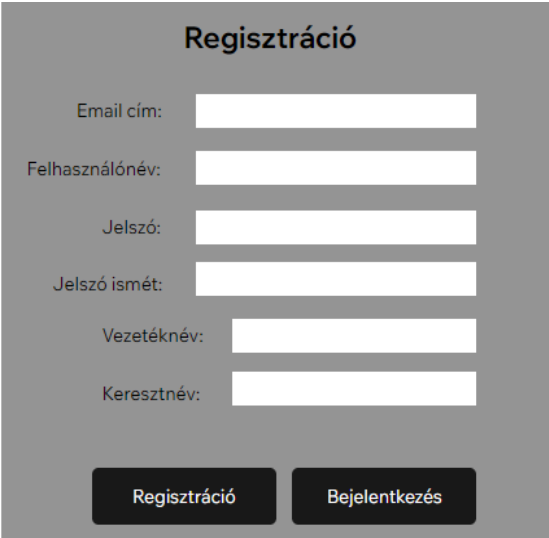
```
transform(value: number): string {  
  const roundedValue = Math.round(value);  
  return roundedValue.toString();  
}
```

4.3.4. ábra: kerekítő transform függvény egy pipe-ban

5. Funkcionális specifikáció

5.1. Bejelentkezés és regisztráció

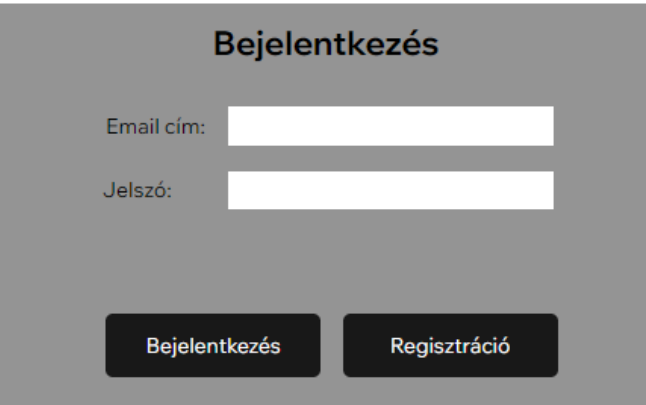
A Firebase és a hozzá kapcsolódó Angular-os technikák segítségével könnyedén megoldható az autentikáció egy webalkalmazás esetében. A Firebase gördülékenyen megjegyzi az újonnan regisztrált felhasználókat az Authentication fülön. A jelszavakat hash-elte formában tárolja el így nem tudjuk őket visszafejteni, szinte lehetetlen, az alkalmazásunk biztonságosabb lesz az adathalászokkal szemben. A Firebase egy egyedi azonosítót (*User UID*) is hozzárendel minden újonnan regisztrált felhasználóhoz. Mindezt a kódban egy külön Service-be kiszervezve oldottam meg néhány függvény segítségével. Ezen felül a Firestore Database-ben még egy külön kollekció is rendelkezésre áll, ahol a felhasználókról szóló egyéb adatokat is le tudjuk tárolni dokumentumok formájában.



A regisztrációs felület tervezet egy sötét szürke háttérrel rendelkező form. A cím "Regisztráció" középsőre van igazítva. Alatta hat fehér beviteli mező található: "Email cím:", "Felhasználónév:", "Jelszó:", "Jelszó ismét:", "Vezetéknév:" és "Keresztnév:". A mezők mellett vannak jelölő ikonok. Alul két fekete gomb van: "Regisztráció" és "Bejelentkezés".

5.1.1. ábra: Regisztrációs felület terve

Természetesen adatvédelmi szempontból a jelszót ekkor sem érdemes csak úgy eltárolni. A kódban ez úgy szerepel, hogy van egy külön modul létrehozva a regisztrációnak és a bejelentkezésnek is egyaránt, amelyek így külön oldalként jelennek meg, és igencsak könnyen használhatóak. A bejelentkezésben egy gombhoz kötött függvény hívja meg az `authService login` függvényét, és így történik meg a hitelesítés. A regisztrációhoz már arra is szükség van, hogy egy másik Service-t használjunk, melyben a CRUD (Create, Read, Update,



A bejelentkezési felület tervezet egy sötét szürke háttérrel rendelkező form. A cím "Bejelentkezés" középsőre van igazítva. Alatta két fehér beviteli mező található: "Email cím:" és "Jelszó:". Alul két fekete gomb van: "Bejelentkezés" és "Regisztráció".

5.1.2. ábra: Bejelentkezési felület terve

Delete) műveletek vannak megvalósítva *User* objektumokra vonatkozólag. A create művelet segítségével a Firebase Database megfelelő kollekciójában is megjelenik az

újonnan regisztrált felhasználó, és annak adatai azon felül, hogy az Authentication fülön is megtalálható lesz. Képernyőtervek láthatóak az oldalsó ábrákon.

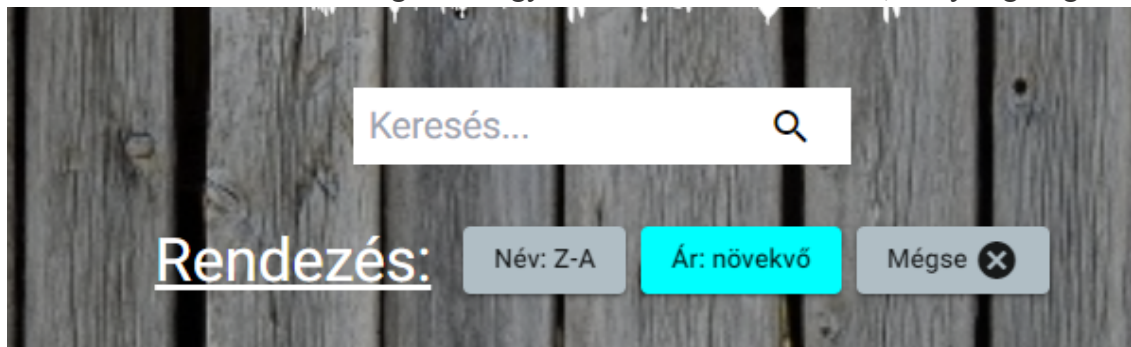
5.2. Böngészés a termékek között

A webáruházak elsődleges célja, hogy eladják termékeiket, és ezáltal profitot termeljenek. Elengedhetetlen, hogy a vásárlók könnyedén tudjanak böngészni közöttük. Ezen okból kifolyólag alakítottam ki a menüsávon egy részt, melyen a különféle italok kategóriái között könnyedén lehet navigálni. Az ábra szemléltet is néhányat a kódból a teljesség igénye nélkül. A főoldalon a hamarosan érkező, az új, az akciós és a kifutóban

```
</ul>
<ul fxLayout fxLayoutGap="15px" fxLayoutAlign="center" class="navigation-items" id="nav-item2">
  <li><a [routerLink]="'/category/' + 'All'" (click)="setActiveLink('/category/All')"
    [ngClass]="{'active': activeLink === '/category/All'}">Minden</a></li>
  <li><a [routerLink]="'/category/' + 'Beer'" (click)="setActiveLink('/category/Beer')"
    [ngClass]="{'active': activeLink === '/category/Beer'}">Sörök</a></li>
  <li><a [routerLink]="'/category/' + 'Wine'" (click)="setActiveLink('/category/Wine')"
    [ngClass]="{'active': activeLink === '/category/Wine'}">Borok</a></li>
  <li><a [routerLink]="'/category/' + 'Champagne'" (click)="setActiveLink('/category/Champagne'"
    [ngClass]="{'active': activeLink === '/category/Champagne'}">Pezsgők</a></li>
```

5.2.1. ábra: A menüsáv routing logika a kódban

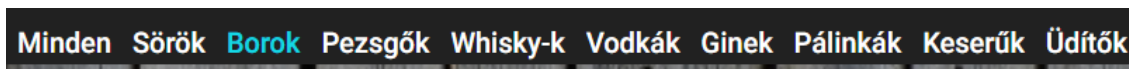
lévő termékek vannak szerepeltetve, melyek felhívhatják a vásárlók figyelmét. Emellett ezen az oldalon a dizájn és a helytakarékosság szempontjából a termékek között lapozgatni lehet, így megkeresve a számunkra legszimpatikusabbat. A kategóriák oldal és a főoldal is egy-egy különálló modulban helyezkedik el. Amennyiben kiválasztunk egy számunkra szimpatikus kategóriát minden termék, amely az adott kategóriába tartozik megjelenik a képernyőn bármiféle rendezési logika nélkül. A fogyasztónak lehetősége nyílik abc sorrendbe, és annak fordítottjába vagy akár ár szerint növekvő, avagy csökkenő sorrendbe helyezni a termékeket. Így sokkal gördülékenyebbé válik a vásárlás. A rendezési lehetőség fölött egy keresési funkció található, mely segítségével



5.2.2. ábra: Rendezés és dinamikus keresés a Kategória oldalakon

dinamikusan tudunk keresni a felsorakozott termékek között, amennyiben elkezdjük gépelni egy adott termék nevét. A termékek nevét alapul vevő funkció már a gépelés

pillanatában meghányadolja a termékeket. Csak azok maradnak a listában, amelyek *name* attribútuma tartalmazza a begépelte stringet. A feljebb látható ábrán látható a megjelenése ezen funkcióknak. A kategória sáv a menüben a kiválasztott jelzésével:



5.2.3. ábra: Kategória sáv a menüben

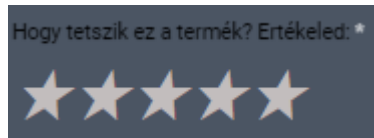
5.3. Termékek részletei

Kiemelt prioritással bír, hogy az egyes termékek rendelkezzenek felhívó leírással, kedvező árral, és minden fontos információval, mely arra sarkallja a felhasználókat, hogy azonnal rendeljék meg az adott terméket, hisz mindenképp jól járnak vele. Ezen okokból kifolyólag a termékekre kattintva egy új lap tárul elénk, melyen csak a fókuszba helyezett termék képe szerepel nagyobb változatban, emellett sok érdekes információ róla, mint például egy frappáns leírás. A kiválasztott termékhez hasonló termékek kisebb képei is megjelennek ezen az oldalon, amik szintén csak könnyítik a böngészést, emellett segítik az újdonsült termékek megismerésének folyamatát. A felhasználók felé visszajelzésként egy „HAMAROSAN ELFOGY” feliratú kiemelt szöveg tárul, amennyiben már csak kevés van az adott termékből, de még az is jelzésre kerül, ha egy termék éppen nincsen raktáron. A részleteket vizsgálva van lehetősége a fogyasztónak többféle módon értékelni az adott terméket.

5.4. Felhasználói visszajelzések

Az előbb említett felületen lehetőségük van a vásárlóknak hozzászólni az adott termékhez. Így visszajelzéseket küldhetnek minden egyes termékről a webáruház üzemeltetője felé. Ezek a kommentek szintén tárolódnak a Firebase Database-ben egy kollekcióba, mint a termékek és a felhasználók adatai. Minden felhasználó felhatalmazást kap arra, hogy a saját maga által írt hozzászólást bármikor eltávolítsa vagy frissítse. Az admin felhasználóknak jogukban áll bírálni bizonyos kommenteket, ezért törölhetik őket attól függetlenül, hogy ki a szerzője. Az aktív adminok így kulturált közösséget teremthetnek, avagy az üzemeltető szempontjából, habár átverés, de hasznos lehetőség a negatív visszhang kommentek kiszűrése, és eltávolítása. Az ilyesfajta értékelések azért relevánsak, mert az üzemeltető láthatja, hogy melyik az a termék, amelyre nagyobb az igény, melyik az, amire nem. Természetesen nyilván

tarthatjuk a szállítási mennyiségeket is, hogy még tisztább képet kapjunk az adott árucikk fogyasztását illetően. Emellett a vásárlók a pozitív visszajelzésekkel egymást is vásárlásra ösztönözhetik, ugyanis egy olyan termék, amely sok pozitív megjegyzéssel van elárasztva nagyobb valószínűséggel vonzza be új fogyasztókat. Másik lehetőség az

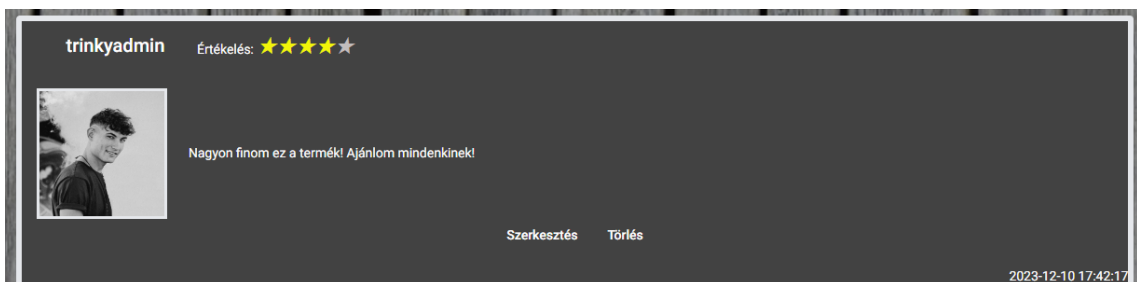


5.4.1. ábra: Üres értékelés



5.4.2. ábra: Értékelés használva

értékelésre, hogy a felhasználók egy pontozási rendszerben tudják értékelni a nekik tetsző, vagy épp nem kívánatos termékeket. A skála 1-től 5-ig terjed, melyet csillagok jeleznek (lásd: ábrák). Minden felhasználó minden egyes termékre csak egy értékelést nyújthat be, viszont ezt a későbbiekben lehetősége van módosítani. Az összes felhasználó egyes termékekre adott értékeléseinek átlaga a termék részleteinél megtekinthető. Kép egy kommentről:



5.4.3. ábra: Komment az egyes termékek felületén

5.5. Kosárkezelés

A felhasználóknak a minél kényelmesebb vásárlás érdekében több helyen is elérhető a kosárba tétel lehetősége. A főoldal, a kategóriák között való böngészés és az egyes termékekre kattintás után is lehetőség nyílik arra, hogy a kívánt darabszámú terméket elhelyezzük a kosarunkban. A sikeres kosárba helyezést egy toastr segítségével jelezzük is a felhasználók felé, mindamelllett, hogy a kosár ikon a menüsorban jelzi a benne elhelyezett termékek darabszámát. Ilyenkor az elhelyezett termékek a



5.5.1. ábra: Kosár ikon

localStorage-ban is tárolódnak, azért, hogy amennyiben a felhasználó frissíti a weboldalt, akkor az addig kiválasztott árucikkek ne vesszenek el a kosárból. A kosár egy önálló modul, ami mindig az adott felhasználóra vonatkozóan tárolja az információt. A bevásárlókocsi ikonra kattintva navigálhatunk arra az oldalra, ahol a

fizetést is lebonyolíthatjuk. Itt látjuk összesítve az elhelyezett árucikkek árát. Lehetőségünk nyílik csökkenteni és növelni az elhelyezett árucikkek darabszámát, eltávolítani egy adott terméket a listából, törölni a teljes bevásárlókocsi tartalmát, visszainavigálni a főoldalra a további vásárlás érdekében, de akár már fizethetünk is egyetlen gomb megnyomását követően. Kosár kinézete a weboldalon:

Termékek	Név	Egységár	Mennyiség	Teljes ár	Kosár ürítése
	Kék Whisky	7496 Ft	– 1 +	7496 Ft	
	Am Cneblee-i Vörösbor	3031 Ft	– 2 +	6062 Ft	
Vásárlás folytatása				13558 Ft	Tovább a fizetéshez

5.5.2. ábra: A kosár lehetséges tartalma

5.6. Integrált fizetési rendszer







A webáruházban kizárólag bankkártyás fizetésre van lehetőség, melyet a kosár alapos megvizsgálása után érhetünk el. A Stripe fizetési rendszer beépítése teszi lehetővé a vásárlást. Mivel ez nem egy valódi webshop, ezért természetesen tesztkártya adatokkal nyílik lehetőségünk fizetni, mint például a 424242424242-es számú bankkártya. Ilyenkor egy külön API-ból származó oldal tárul a szemünk elé, ahol a fizetési adataink megadása után (amik természetesen továbbra is hamis adatok, kisebb ellenőrzésekkel, mint például az email cím formátumának ellenőrzése, a kártya lejárat dátumának valószerűsége oly módon, hogy bármilyen jövőbeli dátumot megadhatunk) a „Fizetés” gombra kattintva visszajelzést kapunk a vásárlás sikerességét illetően. A Stripe rendszer a Firebase Cloud Functions segítségével van integrálva, ugyanis habár az Angular 100%-ban egy frontend keretrendszer, itt szükség van a backend továbbfejlesztésére a Firebase alapvető funkcióin túl. Egy Express szerver működése teszi lehetővé azt, hogy a kitelepített alkalmazásban is (anélkül, hogy lokálisan bármilyen szervert működtetni kellene) megjelenjen a fizetési felület. A sikeres fizetést követően a Stripe hivatalos

oldalán belépve meg is kapjuk a vásárlás adatait, melyeket továbbítani szerettünk volna. A megadott e-mail cím, a bankkártya titkosítva, a termékek képei, nevei, árai és az összár feltüntetése könnyen beazonosíthatóvá tesz egy adott rendelést azon túl, hogy

ID	pm_10BKe0BErcCUqQ7GHf5SK2fx	Owner	asdsada
Number	**** 4242	Owner email	test@gmail.com
Fingerprint	pVBgeEUDQWL0nwzU	Address	HU
Expires	12 / 2023	Origin	United States 🇺🇸
Type	Visa credit card	CVC check	Passed ✅
Issuer	Stripe Payments UK Limited		

5.6.1. ábra: Rendelési adatok a Stripe-ban

természetesen az időpont is szerepel a megjelenő sorban. A képeken látható egy példa rendelés eredménye a Stripe-ból, a feltüntetett rendelési adatokkal, és egy másik példa, hogy hogyan vannak feltüntetve az egyes megrendelt termékek adatai, és a forintban

Customer	asdsada Hungary		
ITEMS	QTY	UNIT PRICE	AMOUNT
 Kékes Whisky	1	Ft7,690.00	Ft7,690.00
 Kék Whisky	2	Ft7,890.00	Ft15,780.00
 Am Cneblee-i Vörösbor	1	Ft3,190.00	Ft3,190.00
 Anikai pálinka	1	Ft3,679.00	Ft3,679.00
 Narancs pálinka	1	Ft5,169.00	Ft5,169.00
 Bechkva B62 Vodka	1	Ft6,990.00	Ft6,990.00
Total			Ft42,498.00

5.6.2. ábra: Rendelt termékek adatai a Stripe-ban

értendő teljes ár is. A felhasználó természetesen ezt nem láthatja, azonban visszajelzésként egy újabb oldalra navigálódik melyen megjelenik, hogy a vásárlása sikeres volt, emellett a saját email címem, amelyre kattintva lehetősége nyílik az ügyfélnek megosztani az észrevételeit a vásárlással kapcsolatban. Ebben az esetben a kosár is kiürül, emellett a termékek darabszáma is csökken a megvásárolt mennyiségek számával.

5.7. Kedvezmények szerzése (Beépített játék, egyszer használatos link)

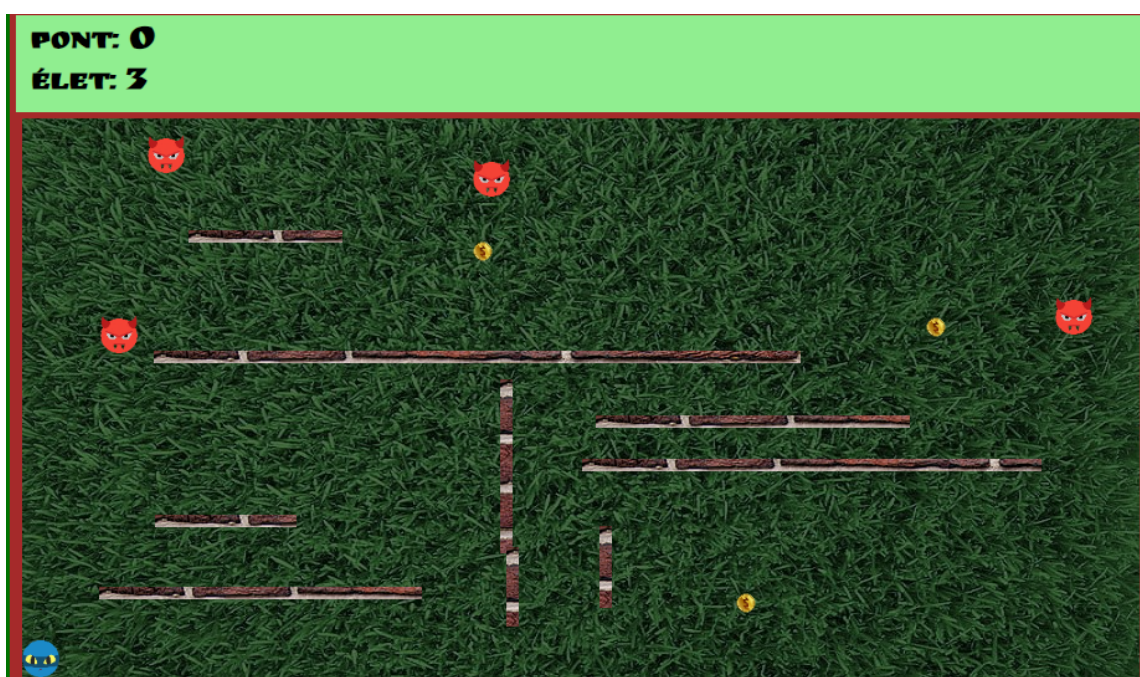
A játék egy újabb modul a rendszer teljes egészét illetően. A felhasználónak előbb be kell jelentkeznie, hogy ez a funkció elérhetővé váljon. Amennyiben egy odatévedő felhasználó rákattint a Játék gombra a menüsávban a jól kiépített routingnak hála a bejelentkezési oldalon találja magát, ahol egy figyelmeztető üzenet is felvillan, hogy a játék opció, csak bejelentkezett felhasználók számára érhető el. Ezen funkció egy nagyon egyszerű, a pacman mintájára készült egypályás kalandozást tesz lehetővé, ahol a feladat csupán annyi, hogy mindhárom pénzért felszedjük főhősünkkel. Természetesen akadályokba is ütközhetünk, melyeken nem tud áthaladni a játékos, csakúgy, mint a játéktér falának ütközve sem tudja elhagyni a pályát. Ezen falak elhelyezkedése a véletlenre van bízva, ugyanis véletlenszám generátorok segítenek a pálya felépítésében. Ez nem minden, ugyanis gonosz ellenségek sokasága is megfertőzi a térképet, akiknek elhelyezkedése, darabszáma, sőt még a mozgásuk milyensége is teljesen randomizált folyamat. Ha szerencsések vagyunk, vagy elég türelmesek, és elégszer frissítünk rá a weboldalra akkor kaphatunk egyszerűen végigvihető pályákat is. A nehézségi szint csupán a játékos szerencséjén, avagy türelmén múlik. Minden regisztrált felhasználó 3 életet kap a játékhoz, amely életek minden 24 órában újra jóvá íródnak a fiók számláján. Ha nekiütközik egy ellenségnek, vagy ő ütközik nekünk akkor veszítünk egy életerő pontot, visszakerülünk a kiindulási pozícióba, ami a bal alsó sarok, és elveszítjük az addig már megszerzett pénzértéket, amelyek szintén random helyeken újra generálódnak a pályán. A játék akkor ér véget, ha elveszítjük az összes életpontunkat, avagy megszerezzük mind a három pénzértét. Amennyiben elbuktunk akkor nem kapunk semmiféle kedvezményt, viszont, ha sikeresen teljesítjük a pályát, akkor 3% kedvezményre leszünk jogosultak, mely minden termékre érvényes. Nincs lehetőségünk az életerő pontokat gyűjteni, ha sokáig nem játszunk, mindig 3 lehet a maximális a számlánkon. Ehhez a folyamathoz szintén a Firebase Cloud Functions használatára volt szükség, ahol egy 24 óránként lefutó függvény automatikusan frissíti az összes fiók életerő pontját 3-ra amennyiben kevesebb volt ennél az értéknél. A megszerzett kedvezmény minden 24 órában lenullázódik, ami arra sarkallja a vásárlókat, hogy mindennap amikor csak rendelni szeretnének újra játszanak. Mivel a fogyasztókat regisztrálásra szeretnénk sarkallni ezért van egy másik lehetőség is, ahol

5% kedvezményt szerezhetünk minden termékre. Így már összesen 8% kedvezmény kerülhet jóváírásra minden egyes termékre a regisztrált, s ügyesen játszó felhasználók

Újszerűségünk okán ezen linkre kattintva 5% kedvezményt biztosítunk minden regisztrált felhasználónak minden termékünkre a feltüntetett és a játékkal szerzett akciókhoz bónuszként!
Kattints az akcióhoz!
Köszönjük, hogy elolvastad a bemutatkozásunkat. Kellemes vásárlást kívánunk.

5.7.1. ábra: Kedvezményes link a Rólunk oldalon

javára. Az 5% kedvezmény megszerzése érdekében csupán csak a Rólunk fülön megtalálható linkre kell kattintani, amely bejelentkezetlen fogyasztók esetén a routingnak köszönhetően szintén a bejelentkezési felületre navigál. A kedvezmények biztosítása és fenntartása növelheti a vásárlói elégedettséget. A képeken látható az egyszer használatos link, és a játék végleges formái.



5.7.2. ábra: Játék felülete egy lehetséges pályán

5.8. Felhasználói fiók kezelése

A felhasználói fiók kezelése kulcsfontosságú része egy weboldal életének, hiszen ez az interaktív felület hozza létre a személyes kapcsolatot a vásárlók és a webshop között. A „Profil” fül, mely bejelentkezést követően érhető el, egy átfogó modult kínál minden felhasználónak a saját fiók személyre szabására és naprakészen tartására. A jelenleg használt e-mail cím, felhasználónév,

5.8.1. ábra: Jelszóváltoztató felület

vezetéknév és keresztnév mellett megtekinthetik az aznapra már megszerzett kedvezményüket is, melyet százalékban kifejezve láthatnak. Az itt elérhető információk átláthatóvá teszik a felhasználók számára a fiókjukkal kapcsolatos részleteket. A felhasználóknak lehetőségük van a kevésbé kritikus adatok, mint például a felhasználónév, vezetéknév és keresztnév módosítására, ami rugalmasságot biztosít a személyes beállításai terén. Az említett adatok egyszerű módosítása mellett a weboldal fejlett biztonsági intézkedéseket kínál a jelszóváltoztatásra is. Bár ez egy összetettebb folyamat, de kulcsfontosságú, hogy a felhasználók számára biztosítsunk egy hatékony és biztonságos módot jelszavuk frissítésére. A profilkép feltöltése további lehetőséget kínál a felhasználóknak az egyedi azonosításra. Ezen képek bármikor módosíthatóak, mely szintén hozzájárul a felhasználói fiók naprakészen tartásában. Ezek minden komment mellett megjelennek, így a felhasználók könnyen felismerhetővé válnak a közösség számára. Ezáltal a webshop nem csupán egy online vásárlási platform, hanem egy olyan közösségi tér, ahol az azonosítást követően a blogbejegyzések és kommentek írásával a személyes élmények is megoszthatóvá válnak. A felhasználói fiókok, avagy a profilok kezelése így nem csupán egy



5.8.2. ábra: Profilkép csere

adminisztratív funkció, hanem egy olyan eszköz, amely közelebb hozza az ügyfeleket egymáshoz és a weboldal egészéhez.

5.9. Bejegyzések írása

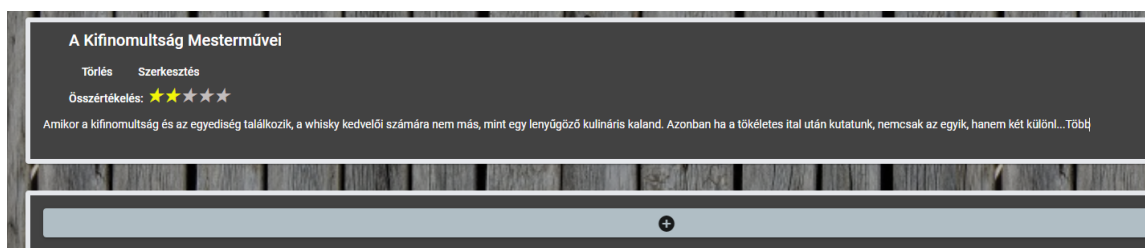
A blogbejegyzés funkció egyedülálló lehetőséget kínál a webshop regisztrált felhasználóinak, hogy ne csupán vásároljanak, hanem személyes tapasztalataikat, véleményüket és szakértelmüket is megosszák a közösséggel. Ez az interaktív és dinamikus funkció nem csupán kiegészíti a webshop kínálatát, de egy teljesen új dimenziót ad az online vásárlási élménynek.

Miután a felhasználók regisztráltak egyedi blogfelületet kapnak, ahol szabadon kifejezhetik gondolataikat és tapasztalataikat, mindemellett mások által írt blogbejegyzések sokasága között böngészhetnek. Ahogy az a termékeknél is fellelhető funkció, itt is lehetőség nyílik a mások által írt bejegyzések értékelésére szintén egy 1-től 5-ig terjedő skálán, melyet csillagok jelölnek.

A blogbejegyzések írása és megosztása során a felhasználók szabadságot élveznek, ugyanis értékelhetik a termékeket, megoszthatják azokkal kapcsolatos történeteiket, vagy akár szakértői tanácsokat adhatnak a különböző italokról. A rugalmasság az egyik kulcsfontosságú jellemzője ennek a funkciónak. A felhasználók bármikor szerkeszthetik a blogbejegyzéseiket, így mindig aktuális és releváns információkat oszthatnak meg a közösséggel.

Emellett a bejegyzések akár el is távolíthatóak, így a felhasználók teljes kontrollt gyakorolhatnak a megosztott tartalom felett. A blogbejegyzés lehetősége nem csupán egy szokványos véleménynyilvánítási platform, hanem egy interaktív tudásmegosztási eszköz is.

Ezáltal a webshop nem csupán egy hely a termékeinek értékesítésére, hanem egy közösségi tér, ahol az ügyfelek egymás közötti kapcsolatokat építhetnek, és érdekesebbnél érdekesebb információkkal gazdagíthatják az italkultúrát és a webshop hosszú távú vonzerejét. A képen részlet látható a bejegyzések oldalról:



5.9.1. ábra: Példa blog és hozzáadás gomb

5.10. Admin funkciók

A webshop adminisztrátorai különleges jogosultságokkal rendelkeznek, amelyek elengedhetetlenek a rendszer hatékony üzemeltetéséhez. Az adminisztrátorok számára elérhető kulcsfontosságú funkciók közé tartozik a termékek kezelése. Csakis az adminisztrátorok jogosultak új termékek hozzáadására, meglévők törlésére és azok adatainak frissítésére, biztosítva ezzel a webshop aktuális kínálatát. Ezen kívül az adminisztrátorok felelősek a kommentek és blogbejegyzések felülvizsgálatáért is.

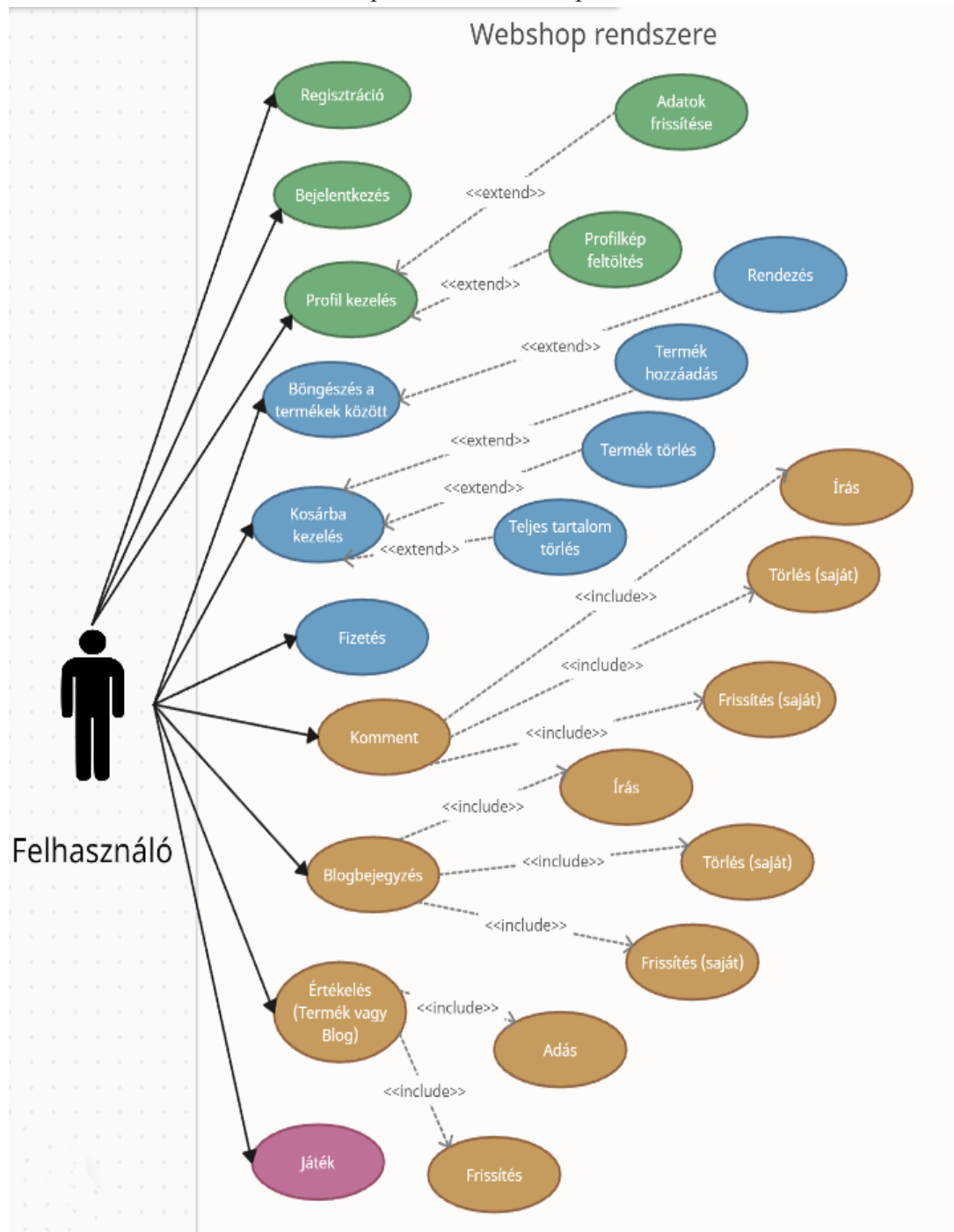
Jogosultságuk révén ellenőrizhetik a felhasználók által beküldött tartalmakat, és eltávolíthatják azokat, amelyek nem felelnek meg a webshop irányelveinek vagy amelyek nem kívánatosak.

5.11. Routing

A routing azért felel, hogy navigációt biztosítson a felhasználóknak. Minden oldalt egy-egy modul épít fel, amely kisebb komponensekre lehet osztva. Amikor egy felhasználó megnyitja a webáruházat akkor a main, avagy Főoldal nyílik meg. Megjelenik felül egy menüsor minden oldalon, ami könnyű navigációt tesz lehetővé. A Főoldalról navigálni lehet a Rólunk, a Bejelentkezés, a Regisztráció, a Kosár, az egyes Termékek és az egyes Kategóriák oldalaira. A háttérben a Category és a Product is egyetlen modul csak dinamikusan váltogatják a megjelenítendő tartalmat az ActivatedRoute-nak köszönhetően. Ez a hét oldal között bármelyikről bármelyikre végrehajtható a megfelelő menüsor elem kiválasztásával a navigáció. A kosár oldalról a fizetés gombra kattintva érhető el egy külső oldal, mely a checkout névre hallgat. Ez egy külső Express szerveren fut, amely kommunikál a Stripe fizetési rendszerrel. Erről az oldalról visszalépve a Sikertelen fizetés oldalra navigálunk, sikeres fizetés esetén pedig a Sikeres fizetés oldalra. Bejelentkezést követően elérhető a Profil, a Játék és a Blog oldalak is a menüsáv újdonsült elemeiként. Az admin felhasználóknak elérhető egy új modulban az új termék felvétele funkció. Összesen így 13 modulból áll a teljes alkalmazás.

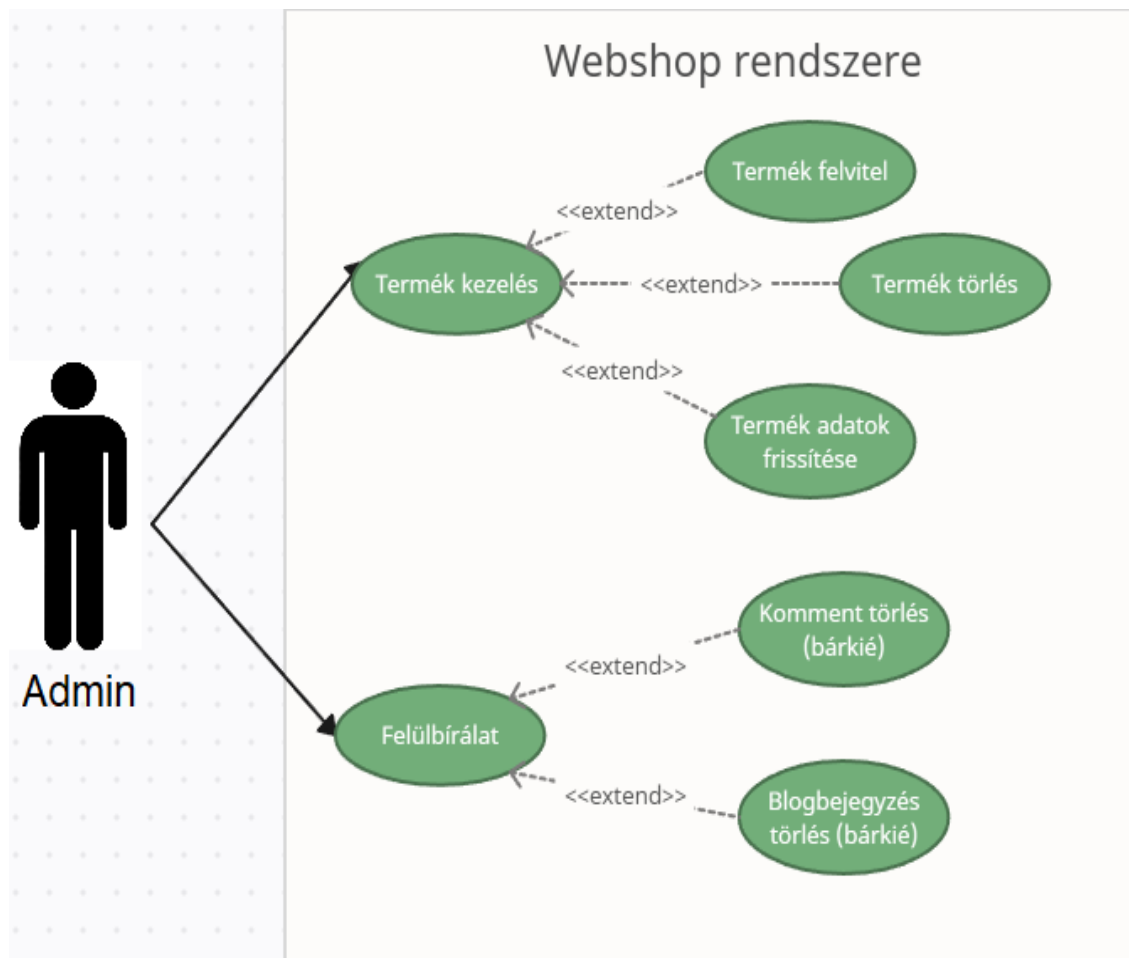
5.12. Use-Case diagramok

Felhasználók használati eset kapcsolata a webshop rendszerével:



5.12.1. ábra: Felhasználó használati eset diagram

Admin felhasználók lehetséges további használati eset tevékenységeinek feltüntetése az alapvető felhasználói funkciókon túlmenően:



5.12.2. ábra: Admin használati eset diagram

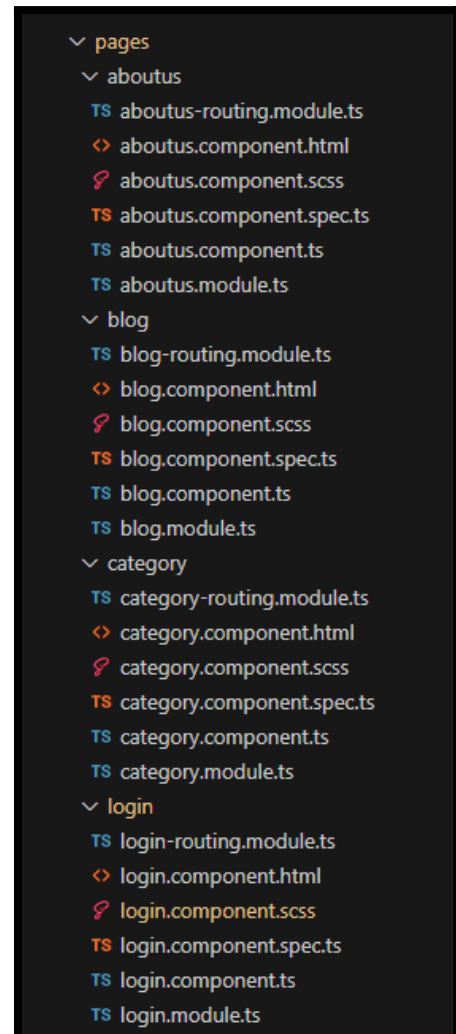
6. A megvalósítás lépései fontosabb kódrészletek kiemelésével

6.1. Komponensek és modulok

A komponensek és modulok hierarchikus szervezése kulcsfontosságú lépés az Angular alkalmazás megtervezésében. A komponensek tisztázottan elkülönülő egységeket képviselnek, amelyek lehetővé teszik az újrafelhasználhatóságot és az egyszerűbb karbantarthatóságot. Minden komponens tartalmaz egy TypeScript fájlt a logika kezelésére, valamint HTML és SCSS (vagy más stílus) fájlokat a megjelenésért és stílusért. Modulokba szervezve ezek a komponensek még nagyobb egységeket alkotnak, ahol a modulok általában önálló oldalakat jelenítenek meg. A modulok további tartalmakat is magukba foglalhatnak, mint például szolgáltatások, csövezetékek és más fájlok. A routing szintén elengedhetetlen részegység (egy különálló ts kiterjesztésű fájlban helyezkedik el), amely lehetővé teszi a modulok közötti hatékony átirányítást. Így könnyedén létrehozható egy

Single Page applikáció. Az alkalmazás a *6.1.1. ábra: Modulok a fájlrendszerben* legeneráláskor létre is hozza a gyökermodult AppModule néven. Az ábrán néhány modul látható a saját alkalmazásomból. Célszerű volt először megtervezni, hogy mennyi, és milyen céllal elkészülő modult hozzak létre. A tervezési folyamatban kiemelt a külső forrásból származó modulok importálásának fontossága az alkalmazás megjelenésének és működésének optimalizálása érdekében.

A teljes alkalmazás a gyökermodult is beleértve 14 modulra bomlik szét, melyek mind különféle feladatokat hivatottak ellátni. A gyökermodul app névre hallgat, és a



többi modul felett áll. A fő feladatai közé tartozik a testvér modulok közötti navigáció megvalósítása, és a menüsáv kialakítása. A menüsáv, amennyiben telefonon, avagy kisebb méretű kijelzőn látogatjuk az alkalmazást egy hamburger menüvé alakul át. Ehhez egy külön komponens kapcsolódik a tagolhatósága végett, melyet az AppModule definiál. A képen a routing megvalósításának részlete látható:

```
{
  path: 'main',
  loadChildren: () => import('./pages/main/main.module').then(m => m.MainModule)
},
{
  path: 'login',
  loadChildren: () => import('./pages/login/login.module').then(m => m.LoginModule),
  canActivate: [NotAuthGuard]
},
},
```

6.1.2. ábra: Routing példák

A MainModule (avagy főoldal) az áttekinthető felhasználói élmény érdekében különböző jelzésekkel ellátott termékeket kínál. A termékek jól láthatóan különböző címkékkel vannak ellátva, mint például "hamarosan érkező", "új", "akciós" vagy "kifutó". Az egyes termékekhez való hozzáférés és a kosárba helyezés lehetősége azonnal a felhasználók rendelkezésére áll a MainModule-ban, így felgyorsítva és megkönnyítve a vásárlást. A MainModule komponensekre van tagolva, ezzel optimalizálva a fejlesztést és elősegítve a karbantartást. A komponensek egy-egy sorért felelősek, ugyanis a főoldalon lényegében 4 sornyi terméket láthatunk a fent említett címkéknek megfelelően. Ez a komponensalapú megközelítés lehetővé teszi, hogy a kód

```
<div id='div-flex'>
  <span class="plexi-layer">
    <div class='title'>Hamarosan kapható termékek</div>
    <app-product-list [marker]='soon'></app-product-list>
    <div class='title'>Új termékek</div>
    <app-product-list [marker]='new'></app-product-list>
    <div class='title'>Kedvezményes termékek</div>
    <app-product-list [marker]='discount'></app-product-list>
    <div class='title'>Kifutó termékek</div>
    <app-product-list [marker]='sale'></app-product-list>
  </span>
</div>
```

6.1.3. ábra: Product-list komponensek, input direktíva használata

tisztább, könnyebben érthető legyen, és lehetővé teszi a fejlesztők számára a funkcionalitás egyszerű módosítását vagy bővítését. A első képen látható a MainModule csekély tartalma, melyben a komponenseknek átadott input direktíva segítségével

osztjuk részekre az oldalt. A következő apró kép pedig ábrázolja, hogy a komponensekben miként kapjuk meg az input direktívában elhelyezett szöveges értéket:

```
@Input() marker?: string;
```

6.1.4. ábra: Input direktíva változóba helyezés

A CategoryModule szintű egy kifinomult megoldás, ahol az egyes termékek kategóriák szerint vannak csoportosítva. Mindössze egyetlen modulból áll, és dinamikusan még is 11 oldalnak ad otthont, mely a kategóriák számával egyezik meg, beleértve azt is amikor minden terméket megjelenítünk az oldalon, avagy nem létező kategóriára keresünk. Ezen dinamikusság szintén a routingnak köszönhető, mert az url-ben azon felül, hogy megjelenik az információ miszerint a „category” oldalon vagyunk, további részegységként fellelhető maga a kategória megnevezése, amely a termékek egy attribútumaként van eltárolva minden egyes árucikk esetén. Ezáltal a felhasználók könnyen navigálhatnak a különféle kategóriák között, és a CategoryModule segítségével gyorsan és hatékonyan megtalálhatják az érdeklődésüknek megfelelő termékeket. Ezzel a megközelítéssel a webshop rugalmasan alkalmazkodik a különböző kategóriákhoz, miközben fenntartja a könnyű kezelhetőséget és az intuitív felhasználói élményt. A felhasználónak ezeken az oldalakon is lehetősége nyílik az egyes termékekhez való hozzáférésre és az azonnali kosárba helyezésre. A böngészést nagyban megkönnyíti, hogy ár- és abc szerint lehetőségünk van rendezni az árucikkeket. Emellett egy keresősáv biztosít szűrési lehetőséget.

Szorosan kapcsolódik a fent említett két modulhoz a ProductModule, amely minden termékhez önálló oldalt létrehozva biztosítja az árucikkek részleteiben való elmélyedést. A felhasználóknak lehetősége nyílik nagyobb ábrán megtekinteni az egyes termékeket, és az azokhoz tartozó információkat. Ezeken az oldalakon biztosított a termékek értékelése, emellett a kommentek hozzáfűzése. A gördülékeny böngészés érdekében a termékhez hasonló más termékek is fellelhetőek a mat-card-ban elhelyezett termékinformációk mellett. A *product* oldalak a kategóriához hasonló dinamikusságot hordozzák magukban.

A kosár egy szintén önálló, kényelmes és felhasználóbarát modul, ahol könnyedén és átláthatóan kezelhetjük a virtuális bevásárlókosarunk tartalmát. Ez a modul lehetővé teszi, hogy a felhasználók a már kosárba helyezett tetszőleges számú terméket egyben láthassák, átvizsgálhassák, és módosíthassák a megvásárolni kívánt darabszámot amennyiben meggondolják magukat ezt illetően. Az oldalon belül lehetőség van a kosár

tartalmának teljes törlésére, valamint az egyes termékek kosárból való eltávolítására is. A kosár oldalon minden egyes termékhez tartozó egységarak és a teljes ár is jól látható, így a vásárlók mindig tisztában lehetnek a felmerülő költségeikkel. Az oldal intuitív és könnyen áttekinthető, lehetőséget biztosítva a felhasználóknak a vásárlás további folytatására vagy a fizetési folyamat elindítására. A kosár tartalmát egy erre a célra kifejlesztett modell segíti, melyet a későbbiekben fogunk részletesen megnézni.

A fizetéshez kapcsolódik két kisebb modul, melyek a fizetés sikerességét jelölik, és a routingnak köszönhetően csak a kosár oldalon látható „Tovább a fizetéshez” gomb lenyomása után érhetőek el. Az egyik akkor kerül a felhasználók elé, amennyiben a fizetési kísérletük sikertelen. Ekkor semmi különösebb nem történik. A felhasználók tovább böngészhetnek a weboldalon, bővíthetik kosarukat, újabb fizetési akciót indíthatnak, és így tovább. A másik modul viszont kritikus fontossággal bír. Ugyanis, amennyiben sikeresen a vásárlás, akkor azt szintén jeleznünk kell a felhasználó felé. Ami még fontosabb, hogy ki kell üríteni a kosarat, és a raktárban lévő árucikkek számát is módosítani kell a kiszállításra küldött mennyiségek levonásával. Mivel a kosár tartalma a localStorage-ban van, ezért ennek a tartalmát kell törölnünk. Emellett kritikus, hogy ténylegesen sehogy ne lehessen elérhető ez az oldal fizetés nélküli interakció esetén. A lejjebb látható kód (6.1.5. ábra) értelmezése: Az *ngOnInit()* élekciklusbeli függvény azonnal lefut, amikor az oldal megnyílik. A biztonság kedvéért először leellenőrizzük, hogy a localStorage-ban elhelyezett *cart* item értéke létezik-e. Amennyiben igen, akkor a *JSON.parse()* függvény segítségével kinyerjük az értékét a *cart* objektum *items* attribútumába, melynek típusa *Array<CartItem>*, amely azt jelenti, hogy a kosárba helyezett termékek tömbjét tartalmazhatja. Ezt követően végig iterálunk ezen a tömbön, és mivel a *CartItem*-ek nem egyenlőek az adatbázisban tárolt *Product*-okkal, ezért a *productService* egy függvénye segítségével feliratkozunk, hogy

```
ngOnInit() {
  if(localStorage.getItem('cart') !== null){
    this.cart.items = JSON.parse(localStorage.getItem("cart"));
    for (let i = 0; i < this.cart.items.length; i++) {
      this.productService.loadImageMetaByProductID(this.cart.items[i].id).pipe(take(1)).subscribe(data => {
        data[0].quantity -= this.cart.items[i].quantity;
        if(data[0].quantity <= 10){
          data[0].marker = "sale";
        }
        this.productService.update(data[0]).catch(error => {
          console.error(error);
        });
      });
    }
  }
  this.cartService.clearCart(true);
}
```

6.1.5. ábra: Termékmennyiségek csökkentése, kosár törlése a localStorage-ból

lekérjük az adatbázisban található megegyező azonosítóval szereplő termékeket. A `.pipe(take(1))` rész segítségünkre van abban, hogy mindig csak egyszer fusson le a lekérés. Alapvetőleg ez egy folyam, melyben a `data` lokális változó a termékek egy tömbjeként érhető el. Viszont mivel az `id`, avagy azonosító egyedi attribútum ezért a `data` tömb egy egyelemű tömb lesz. Ez már az adatbázisban is letárolt terméket fogja tartalmazni. Ezt követően pedig a `productService` egy másik függvényével frissítjük az adatbázisban a megfelelő termékeket egyesével. A `for` ciklus lefutását követően pedig a teljes kosarat kiürítjük egy másik szolgáltatás használatával, melyben a `localStorage` `cart` nevű elemét töröljük.

A regisztrációt és a bejelentkezést szintén önálló modulok alkotják. Mindkettő egyszerű, letisztult, könnyen kezelhető felület. A Firebase biztosította autentikáció van megvalósítva a weboldalon. Mindemellett a Firebase Database kollekciónak közé tároljuk le a felhasználók regisztráció során megadott adatait. Egy különálló szolgáltatás használja az `AngularFireAuth` által biztosított függvényeket. Ezek teszik lehetővé, hogy az új felhasználó email címe, és jelszavának hashelt formája mentésre kerüljön az Authentication fülön, avagy a már mentett értékek alapján azonosítsa a felhasználót. Ennek a

```
constructor(private auth: AngularFireAuth) {}

login(email: string, password: string) {
  return this.auth.signInWithEmailAndPassword(email, password)
}

registration(email: string, password: string) {
  return this.auth.createUserWithEmailAndPassword(email, password);
}
```

6.1.6. ábra: Bejelentkezés és regisztráció szolgáltatás függvény

szolgáltatásnak a használata mindkét modulban fellelhető a megfelelő függvényeinek meghívásával.

Van egy önálló modul, amely kisebb kiegészítésként funkcionál, ahol az ügyfeleknek lehetősége van közelebbről megismerni a webshop milyenségét. A menüsávban „Rólunk” néven jelenik meg, és egy kattintással azonnal elérhetővé válik. Ezen a felületen rövid összefoglaló és bemutatkozás olvasható, amely hozzájárul az üzleti történetünk és értékeink jobb megértéséhez. Az oldal alján egy link található, amelyre kattintva a regisztrált felhasználók 5%-os kedvezményre tehetnek szert. Fontos megérteni, hogy ez a kedvezmény a termékek 100%-os árára vonatkozik, függetlenül az esetlegesen már meglévő más kedvezményektől. Tehát, ha egy termék alapból 50%-os kedvezménnyel szerepel, akkor az 5% kedvezmény ezen termék teljes árából kerül levonásra. Ezáltal biztosítom, hogy az ügyfeleim minden esetben részesülhessenek az extra előnyökből, és ezáltal növelve a weboldal iránti elégedettségüket.

Az eddig említett modulok megtekintései nem igényelnek regisztrációt. Persze már ezek között is fellelhető olyan funkcionalitás, amelyhez szükséges a bejelentkezett fiók, mint például a kedvezményt szerző link használata. A

```
path: 'blog',
loadChildren: () => import('./pages/blog/blog.m
canActivate: [AuthGuard]
```

következő modulok azonosított felhasználókat feltételeznek, ugyanis a guard-oknak hála nem érhetőek el bejelentkezés nélkül. A guard-ok speciális szolgáltatások, melyekben többféle függvényt írhatunk meg, és ezután ezeket szabadon hozzárendelhetjük a routinghoz. Ezen példában a guard canActivate nevű függvényében szerepel a logika, amely a bejelentkezetlen felhasználókat elnavigálja a Bejelentkezés oldalra.

Elsőként vegyük figyelembe a BlogModule-t. Ez a modul lehetővé teszi a felhasználóknak, hogy blogbejegyzéseket írjanak, frissítsenek, és töröljenek. Természetesen mindenki csak a saját bejegyzése fölött bír efféle jogokkal. Az olvasás művelete viszont mások által írt bejegyzésekre is

```
<span *ngFor="let blog of this.blogs">
  <app-blog-content [blog]="blog"></app-blog-content>
</span>
```

6.1.8. ábra: Blog-content komponens használata a kódban

kiterjed. Minden regisztrált felhasználó azonnal rendelkezik a fent említett jogok mindegyikével a belépést követően. Az admin felhasználók törlési jogot kapnak minden egyes blogbejegyzésre, a mások által készítettekre is, ezt nevezzük felülbírálni funkciónak. A blogok mindegyike egy *mat-card*-on helyezkedik el egy-egy külön komponensben. A modul HTML kódjában egy *for* ciklus által kerülnek egyesével átadásra a blogok adatai, mint az a képen is látható. Hasonlóan a MainModule-hoz itt is input direktívát használunk. A blogbejegyzés hozzáadása a HTML-ben egy form használatával valósul meg. A

```
blogsForm = this.createBlogForm({
  id: '',
  author: '',
  title: '',
  text: ''
});
```

6.1.9. ábra: Új blog hozzáadása a kódban (részlet)

TypeScript kód adja a logikáját, melyben egy belső függvény segítségével validátorokat rendelünk az egyes attribútumokhoz, ezzel szabályozva, hogy a felvitt blogbejegyzés

```
createBlogForm(model: Blog) {
  let formGroup = this.fBuilder.group(model);
  formGroup.get('title')?.addValidators([Validators.required, Validators.maxLength(100)]);
  formGroup.get('text')?.addValidators([Validators.required, Validators.minLength(200)]);
  return formGroup;
}
```

6.1.10. ábra: Validátorok a kódban

megfeleljen a meghatározott kritériumoknak (például, hogy a blog címe maximum 100 karakter hosszú lehet, a bejegyzés tartalma pedig legalább 200 karakter hosszú kell, hogy legyen). Ezt követően amennyiben a felhasználó rákattint a létrehozás gombra szintén egy függvény hívódik meg, amely elvégzi az adatbázisba való tárolást egy

```
if(this.blogsForm.valid){
  this.blogsForm.get('author')?.setValue(this.user?.username);
  this.blogService.create(this.blogsForm.value).then(_ => {
    this.toastr.success("Sikeresen rögzítetted a blog bejegyzésedet!", "Blog");
    this.blogsForm.get('title')?.reset();
    this.blogsForm.get('text')?.reset();
  });
}
```

6.1.11. ábra: Validátorok felhasználás, blog létrehozása a kódban

szolgáltatás segítségével. Üressé teszi újra a mezőket, és egy toastr segítségével jelzi a felhasználó felé egy felugró ablakban a felvitel sikerességét. A validátorok használata igencsak egyszerű, ugyanis a *form*-ot ellenőrizve, mintha a *valid* szó egy adattag lenne megvizsgálhatjuk, hogy megfelelt-e a megadott kritériumoknak. Ez egy logikai igaz avagy hamis értéket ad vissza.

A beépített játék is egy külön modulban helyezkedik el. A játékhoz tartozik egy önálló *models* mappa, amiben az összes felhasznált elem, és azoknak a funkcionalitása szerepel a könnyebb átláthatóság végett. Ezen objektumok vannak létrehozva a játék moduljában olyan számban amennyire szükség van. A játékteret egy canvas biztosítja, amely szélessége és magassága a kódban

- models
- TS Barrier.ts
- TS Coin.ts
- TS Enemy.ts
- TS Player.ts

6.1.13. ábra: Modellek a játékhoz

```
<div (window:keydown)="onKeyDown($event)">
  <canvas #gameCanvas width="{{WINDOW_WIDTH}}" height="{{WINDOW_HEIGHT}}" class="gameCanvas"></canvas>
</div>
```

6.1.14. ábra: Canvas a HTML-ben

tetszőleges módosítható, azonban a játékos számára fixálva van. Írnom kellett egy random számokat generáló függvényt, amely elősegíti, hogy

```
getRandomInt(min: number, max: number): number {
  min = Math.ceil(min);
  max = Math.floor(max);
  return Math.floor(Math.random() * (max - min + 1)) + min;
}
```

6.1.15. ábra: Random függvény a játékhoz

a pályán lévő ellenségek, az akadályok, és a megszerezhető pénzürmék helyzete is véletlenszerű legyen. Ezenfelül a

```
onWin(){
  if (this.user) {
    this.userService.updateDiscount(this.user.id, this.user.discount, this.player.point);
    this.userService.updateHealth(this.user!.id, 0);
    this.router.navigateByUrl('/main').then(() => {
      this.toastr.success("Gratulálunk! Minden termékre " + this.player.point + "% kedvezményt kaptál!",
    ));
  }
}
```

6.1.16. ábra: Győzelem függvény a játékhoz a kódban

játék saját TypeScript fájljában azt határozza meg, hogy mennyi legyen az ellenségek, és az akadályok darabszáma. A bejelentkezett felhasználó adatai között szerepel a kedvezmény százaléka, amely érték 3-mal nő amennyiben a játékos sikeresen felvette az összes pénzért, és ezt követően rákattintott a jutalom megszerzését jelző gombra. Végül visszavigyünk a főoldatra.

Az utolsó modul kizárólag admin felhasználók számára elérhető. Ez az a felület, ahol gördülékenyen tudnak új terméket hozzáadni, amely azonnal feltöltődik az

```
<form [formGroup]="productsForm" (ngSubmit)="addProduct()">
  <mat-form-field>
    <mat-label for="id">ID:</mat-label>
    <input matInput type="text" formControlName="id">
  </mat-form-field>
```

6.1.17. ábra: Termék hozzáadása funkció meghívása a kódban

```
async addProduct() {
  this.loginLoading = true;
  if (this.productsForm.valid && this.imageFile) {
    const product: Product = {
      id: this.productsForm.get('id')?.value,
      name: this.productsForm.get('name')?.value,
      photo_url: 'images/' + this.productsForm.get('id')?.value + '.png',
      short_description: this.productsForm.get('short_description')?.value,
      long_description: '-',
      category: this.productsForm.get('category')?.value,
      price: this.productsForm.get('price')?.value,
      alcohol: this.productsForm.get('alcohol')?.value | 0,
      quantity: this.productsForm.get('quantity')?.value,
      marker: this.productsForm.get('marker')?.value
    };
    this.imageFilePath = 'images/' + this.productsForm.get('id')?.value + '.png';
    const task = this.storage.upload(this.imageFilePath, this.imageFile);
    try {
      await task;
    } catch {}
    this.toastr.error("Hiba a képfeltöltés során!", "Termékkép");
  }
  await this.productService.create(product).then(_ => {
    this.router.navigateByUrl('/main');
    this.toastr.success("Sikeres termék felvitel!", "Termék");
    this.loginLoading = false;
  }).catch(_ => {
    this.toastr.error("Sikertelen termék felvitel!", "Termék");
    this.loginLoading = false;
  });
} else {
  this.toastr.error("Sikertelen termék felvitel!", "Termék");
  this.loginLoading = false;
}
}
```

6.1.18. ábra: Termék hozzáadása funkció a kódban

adatbázisban a megadott adatok alapján, és ezt követően meg is jelenik a megfelelő oldalakon. A hozzáadás függvény magában hordozza az új termék adatainak felvitelét a Firestore Database megfelelő kollekciójába és a Storage-ba is megfelelő módon tárolja a képet. A fájl feltöltés kapcsán eltároljuk a képet egy változóban, az elérési útvonalat pedig a megadott termékazonosító alapján állítjuk be. A függvény ellenőrzi, hogy van-e kiválasztott kép és a *productsForm* megfelel-e a validációs feltételeknek. Amennyiben igen, akkor létrehoz egy új *Product*-ot a megfelelő attribútumok értékeinek megadásával. A képet megpróbálja feltölteni a fent említett Storage-ba, ezután tölti csak fel a termékadatokat is a megfelelő kollekcióba. Végül történik egy navigáció a Főoldal irányába, amennyiben minden sikeresen zajlott.

6.2. Modellek (Interfészek)

A használandó modellek kialakítása kulcsfontosságú lépés a webalkalmazás tervezésekor, mivel ezek határozzák meg az alkalmazás adatstruktúráit. A modelltervezés során figyelembe vettem az alkalmazás üzleti logikáját és a kívánt funkciókat, hogy az adatkezelés hatékony és átlátható legyen.

Elsőként a *User* modellt terveztem meg, mely a megvalósítás során átesett kisebb módosításokon is. Át kellett gondolni, hogy melyek azok a releváns információk, amelyeket mindenképp el kell tárolni egy felhasználóról. Ez magában foglalja a létrehozott fiók felhasználónevét, e-mail címét, keresztnévét, vezetéknévét, jelszavát, jogosultságainak kezelését egy admin attribútumban, a profilképének url-jét. A felhasználó modellt úgy kellett megalkotni, hogy támogassa a regisztrációt, a bejelentkezést és az esetleges profil frissítéseket. A korábban említett módosítások a következők (új attribútumok): a fiókhoz tartozó életerő pontok mennyisége a játékhoz, az összesen megszerzett kedvezmény százaléka, a kedvezményes link használatba vétele egy logikai attribútumban.

A *Comment* és *Blog* modellek kapcsán érdemes volt megtervezni azokat az attribútumokat, amelyek segítségével egyértelműen azonosíthatók ezek az entitások. Például egy

```
export interface Blog {  
  id: string;  
  author: string;  
  title: string;  
  text: string;  
}
```

6.2.1. ábra: Blog modell attribútumai

```
export interface Comment {  
  id: string;  
  username: string;  
  comment: string;  
  // Könnyebb így használni itt  
  date: number;  
  productId?: string;  
}
```

6.2.2. ábra: Komment modell attribútumai

komment esetében a szerző felhasználóneve, a komment szövege, a

dátum és annak a terméknek az azonosítója, amelyhez vonatkozóan megírták. A blog modellje esetén a cím, a tartalom és a szerző felhasználónevének letárolása tűnt releváns információnak.

A „Rating” modellje magában foglalja az értékelt termék azonosítóját, a felhasználó által adott értékelést, mint szám érték, és magának az értékelőnek a felhasználónevét. A későbbiekben ugyanezt a modellt használtam fel, amikor a blogbejegyzések értékelését valósítottam meg. Itt a termékazonosító helyett, a bejegyzés azonosítója kerül átadásra, a többi viszont teljesen hasonlóan zajlik, mint a termékek esetében.

Végül a „Product” modellje a legkritikusabb fontosságú a webshop szempontjából. Ebben a modellben tárolom a termék nevét, rövid leírását (eleinte terveztem egy hosszabb leírást is adni minden termékhez, azonban végül a Főoldalt más kialakításban hoztam létre, így csak egyetlen leírás maradt benne a végleges verzióban), árát, raktári készletét, a termék képének url-jét, és egyéb releváns információkat (alkoholtartalom, kategória, valamilyen jelzés). A kategóriák attribútum nagyban segíti a termékek hatékony csoportosítását és keresését.

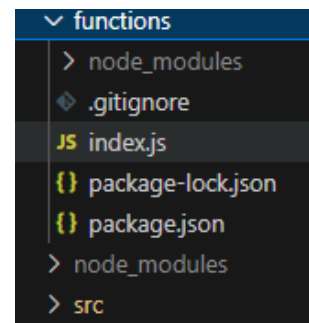
Eredetileg nem tervezett modell, de kiemelt fontossággal bír, és rendkívül megkönnyítette a munkát a *CartItem* modellje, mely a *Cart* modell egyetlen attribútumaként egy tömböt is alkot. Ebben a modellben tárolásra kerülnek a kosárba helyezett árucikkek megfelelő attribútumai. Ez az egyetlen olyan modell, mely csak a frontend oldalon van felhasználva, ugyanis segíti az átláthatóságot. A *Product* modell megfelelő attribútumait használja fel, amellet, hogy tartalmazza a kosárban elhelyezett mennyiséget is, mint adattag.

A modellek létrehozása során kiemelt, hogy azok összhangban legyenek az Angular alkalmazás tervezési mintájával és az alkalmazás üzleti logikájával. A megfelelő modellek segítenek a hatékony adatkezelésben és hozzájárulnak az alkalmazás átláthatóságához és karbantarthatóságához. A backend-en, melyet a Firebase biztosít ezen modellek mindegyike a *CartItem* és *Cart* kivételével egy-egy kollekcióban szerepelnek, és így tárolódnak le a Firebase Database-ben. Fontos, hogy a modellek összhangban legyenek a tényleges adatbázisban létrehozott kollekciókkal.

6.3. Backend kódok

Habár alapvetőleg ez egy frontend fejlesztés, mely már egy meglévő backend-et használ a Firebase segítségével, még is vannak olyan részek melyekhez muszáj volt egyedi kódsorok írására. A Firebase Cloud Functions nagyban hozzájárult ehhez a művelethez. A teljes frontend alkalmazás lényegi kódja az src mappán belül helyezkedik el. Ehhez a művelethez viszont ezen kívül kellett tekinteni, egy külső functions mappába tárolt *index.js* fájl biztosítja a backend-hez írt újdonsült funkciókat. *6.3.1. ábra: Functions mappa tartalma*

Ezen mappa jól láthatóan egyedi node_modules mappával és package.json fájlokkal is rendelkezik a konfiguráció érdekében.



Az első különálló függvény kódjának magyarázata: Elsőként importáltam magát a

```
const functions = require('firebase-functions');
// Stripe function
const express = require('express');
const cors = require('cors');
const bodyparser = require('body-parser');

const app = express();
app.use(express.static('public'));
app.use(bodyparser.urlencoded({extended: false}));
app.use(bodyparser.json());
app.use(cors({origin: true, credentials: true}));

const stripe = require('stripe')('sk_test_..._Q7Gj...pGOMYwQ3r...');

app.post('/checkout', async(req, res, next) => {
  try {
    const session = await stripe.checkout.sessions.create({
      line_items: req.body.items.map((item) => ({
        price_data: {
          currency: 'huf',
          product_data: {
            name: item.name,
            images: [req.body.images.find(i => i.includes(item.id))]
          },
          unit_amount: item.price * 100,
        },
        quantity: item.quantity,
      })),
      mode: 'payment',
      success_url: req.body.url + '/success-payment',
      cancel_url: req.body.url + '/cancel-payment',
    });

    res.status(200).json(session);
  } catch (error) {
    next(error);
  }
});

exports.api = functions.https.onRequest(app);
```

6.3.2. ábra: Stripe-pal kommunikáló backend függvény

Firestore modul, amely lehetővé teszi a Firestore modul használatát.

használatát. Ezt követően importáltam az Express modult, amely egy webalkalmazás keretrendszer. Átala tudunk egy külön álló Express szervert működtetni, ami kiszolgálja a fizetési rendszert. A CORS (Cross-Origin Resource Sharing) importálása segít a keresztezett eredetű HTTP kérések kezelésében. Ez azt jelenti, hogyha a frontend és a backend nem ugyanazokon a domain-eken van telepítve, mégis tudnak kommunikálni egymással (Ez nagy segítség volt a lokális tesztelésnél is, ahol különböző port-on kellett futtatni az Express szervert). A Body-parser a HTTP kérések és válaszok tartalmának könnyű kezelése végett került importálásra. Az importok a require kulcsszót használják. Mindezek után következik az Express alkalmazás inicializálása az *app* nevű változóban. Ezután ezen állítottam be néhány funkciót, hogy gördülékenyebben kezeljen bizonyos típusú adatokat, használja a CORS-t. A következő változó *stripe* névre hallgat, ebbe a stripe rendszer titkos kulcsának értékét tároltam le, melyet biztonsági okokból olvashatatlaná tettem. Ezt követi a meghívott *post* metódus, amelyben létrehozuk, hogy hogyan nézzen ki, és miket jelenítsen meg az az oldal, ahol a fizetést bonyolíthatjuk le a bankkártya adatok megadásával. Itt látható a sikeres és a sikertelen fizetés esetén bekövetkező útvonal navigáció is. Végül az utolsó sorban látszik, hogy a függvény *api* néven fog megjelenni a felhőszolgáltatás függvényei között.

A másik függvény a játékhoz kapcsolódik. Itt szükséges a *firebase-admin* felületének importálása és inicializálása, ugyanis most backend oldalról közelítjük meg, tehát más úton van lehetőség lekérni, és módosítani az adatbázis egyes adatait. Jól látható, hogy a függvény a *scheduledFunction* névre hallgat. A nevéből adódóan látszik, hogy ez a függvény ütemezett, azaz bizonyos időközönként fut le. Mégpedig 1440 percenként, azaz minden 24 órában pontosan egyszer mindig ugyanabban az időpontban. A teljes *Users*

```
// Time scheduledFunction
const admin = require('firebase-admin');
admin.initializeApp();

exports.scheduledFunction = functions.pubsub
  .schedule('every 1440 minutes')
  .timezone('Europe/Budapest')
  .onRun(async (_) => {
    const db = admin.firestore();

    const collectionRef = db.collection('Users');
    const querySnapshot = await collectionRef.get();

    querySnapshot.forEach(async (doc) => {
      const userData = doc.data();
      if(userData.discountToLink){
        await doc.ref.update({ discount: 5});
      } else {
        await doc.ref.update({ discount: 0});
      }

      if(userData.gameHealth !== 3){
        await doc.ref.update({ gameHealth: 3 });
      }
    });

    return null;
  });
```

6.3.3. ábra: Időzítő backend függvény

kollekciót lekérjük, végig iterálunk rajta, és módosítjuk a megfelelő értékeket.

6.4. Jogosultságkezelés Firebase-ben

A táblázatban látszik, hogy mely kollekciókhoz milyen jogosultságai vannak az egyes felhasználóknak. Ezek közül néhány pont magyarázatra szorul. Az értékeléseket nem lehet csak úgy törölni. Amennyiben már adtunk értékelést, akkor csak módosítani tudjuk azt, visszavonni már nem. Azonban, ha a blogbejegyzés, avagy a termék törlésre kerül, amelyre adtunk le értékelést, akkor ez is feleslegessé válik. A saját blogjainkat törölve a mások által leadott értékeléseket, melyek erre a bejegyzésre érkeztek is töröljük a háttérben a tudtukon kívül. A másik érdekesség, hogy a regisztrált és az anonim felhasználók is a tudtukon kívül szerkeszthetik a termékeket, ugyanis amikor sikeresen rendelnek, akkor a háttérben a raktárban lévő mennyiségek száma csökken az adott termékre vonatkozólag. A többi információ relevánsan értelmezhető a táblázat alapján.

	Anonim felhasználó	Regisztrált felhasználó	Admin felhasználó
Kommentek	Olvasás	Olvasás, hozzáadás, szerkesztés (saját), törlés (saját)	Olvasás, hozzáadás, szerkesztés (saját), törlés (bárkié)
Blogok	-	Olvasás, hozzáadás, szerkesztés (saját), törlés (saját)	Olvasás, hozzáadás, szerkesztés (saját), törlés (bárkié)
Értékelések	Olvasás	Olvasás, hozzáadás, szerkesztés (saját), törlés (saját blogra érkezőt)	Olvasás, hozzáadás, szerkesztés (saját), törlés (bárkié)
Termékek	Olvasás, szerkesztés (darabszám)	Olvasás, szerkesztés (darabszám)	Olvasás, hozzáadás, szerkesztés, törlés

6.4.1. ábra: Jogosultságkezelés táblázat

Minden kollekcióra különféle jogosultságok vannak beállítva, amelyek által a Firebase is ellenőrzi, hogy adott esetben van-e jog olvasásra, törlésre, létrehozásra, módosításra. A képen látható ezek közül néhány a teljesség igénye nélkül.

```

match /Comments/{document=**} {
  // Mindenki olvashatja mindenki kommentjét
  allow read;
  // Csak a bejelentkezett felhasználó írhatja/törölheti a saját kommentjét
  // Vagy egy admin felhasználó törölheti bárkijét
  allow write: if request.auth != null &&
    request.resource.data.username ==
      get(/databases/${database}/documents/Users/${request.auth.uid}).data.username;
  allow delete: if request.auth != null &&
    (resource.data.username ==
      get(/databases/${database}/documents/Users/${request.auth.uid}).data.username ||
      get(/databases/${database}/documents/Users/${request.auth.uid}).data.admin == true);
}

match /Ratings/{document=**} {
  // Mindenki láthatja mindenki értékelését
  allow read;
  // Csak a bejelentkezett felhasználó módosíthatja a saját értékelését
  allow write: if request.auth != null &&
    request.resource.data.username ==
      get(/databases/${database}/documents/Users/${request.auth.uid}).data.username;
  // Egy bejelentkezett felhasználó törölheti bárkijét
  // (amennyiben törli a blogját törlődnek a rajta levő értékelések is)
  allow delete: if request.auth != null;
}

```

6.4.2. ábra: Jogosultságkezelés a Firebase Rules-ban

Jól látható, hogy a kommentek esetén mindenki, még a nem regisztrált felhasználók is rendelkeznek olvasási joggal. Ez csupán annyit jelent, hogy bárki képes elolvasni bárkinek a kommentjét. Ez az értékelésre ugyancsak igaz, hiszen a weboldalon a kommentek mindegyikénél megjelenik a termékre leadott értékelés is. A write engedélyezése lehetővé teszi mind a létrehozást, mind a módosítást is. Kommentek és értékelések esetén is előfeltétel az, hogy a felhasználó autentikálva legyen. Ezért felel az első sor az *if* után. Ezt követően ellenőriztem, hogy az érkező adat *username* tulajdonsága megegyezik-e a bejelentkezett felhasználó *username* tulajdonságával. Az adatbázisból a *uid* alapján könnyen kinyerhető ez az információ. A delete rész engedélyezése két fő szempontban tér el az előzőekben ismertetett write-tól. Az adminok bárki kommentjét törölhetik, ha az valamiért nem felelt meg a weboldal irányelveinek. Avagy, ha törölnek egy terméket vagy egy blogbejegyzést akkor a háttérben az ezekre leadott értékelések is feleslegessé válnak az adatbázisban, tehát törlődnek. Azaz az adminoknak a mások által leadott értékeléseket is törölniük kell rekurzívan. A saját blogbejegyzések törléséhez nem kell adminisztrátori jogosultságokkal rendelkeznie a felhasználónak, így a fenti állítás ebben az esetben is igaz. Másik különbség, hogy ilyenkor nem a request-ből származó adatot vizsgáljuk, hanem az eredeti resource-t, ugyanis törlés esetén pont az a lényeg, hogy a requestbe nem érkezik meg az adat. Így ezt az előtagot elhagyva tökéletesen működik a kívánt jogosultság.

7. Tapasztalatok, továbbfejlesztési lehetőség

7.1. Tapasztalatok, nehézségek

Tapasztalataim szerint az Angular egy meglehetősen összetett, és nehezen kezelhető keretrendszer. Viszont összetettségéből fakadóan rengeteg lehetőséget rejt magában. Ami a fő nehézséget okozta az még is a fizetési rendszer integrálása volt. Sikeresen működtetve egy Express szerveret már lokálisan tudtam működtetni, azonban ez nem volt elég. Meg kellett ismerkednem a Firebase Cloud Functions funkcióval, amely lehetővé tette a könnyű kezelést a hostolt verzió számára is. Ezen funkció elérése, habár igényli a bankkártya adatok megadását, mégis egy teljes ingyenesen használható megoldás.

Az Angular okozta nehézségeket továbbfűzve, eleinte nehéz volt átlátni, hogy milyen részegységekre oszlik fel egy adott modul. Mi az, amit érdemes lehet kisebb komponensekre tagolni, így logikusabbá téve a kódot.

7.2. Továbbfejlesztés

Úgy hiszem semmi sem tökéletes, ezért minden alkalmazás esetén folyamatosan ott rejlik a továbbfejlesztés lehetősége. Újabb funkciók beépítése, melyek bővítő jelleggel hatnak az alkalmazásra, avagy a régiek felfrissítése, továbbgondolása.

Az italokat áruló webalkalmazásom nem tartalmaz többnyelvűséget, viszont ennek az előnyeit egy korábbi fejezetben részleteztem. Ez egy fontos új funkció lehetőség a továbbfejlesztésre. Megoldási variációként akár fordító API importálása, akár manuálisan írt pipe-szerű fájl használat a szavakra ugyanazt a hatást érheti el. Ezen két megoldási mód közül érdemes megfontolni, hogy melyiket egyszerűbb, érdemesebb beépíteni, és melyik fejleszthető, karbantartható könnyebben.

Másik aspektus egy már elhelyezett funkció továbbfejlesztése. A játék funkció rengeteg lehetőséget rejt magában, például többféle pálya felépítése, akár más- és másfajta stílusban. Így kielégíthető lenne a fogyasztók csaknem teljes állománya az egyéni preferenciáknak megfelelően. Mindezzel magasabb értékű kedvezmény szerzése lenne elérhető. A több kedvezmény pedig még nagyobb vonzerővel bír a weboldal érdekeltségének irányába.

Összegzés

A szakdolgozat megvalósításának keretein belül megismerkedtem azokkal a technológiákkal és eszközökkel, amelyek segítségével elkészült egy dinamikus interneten elérhető képzeletbeli termékeket árusító webalkalmazás.

A megvalósítás során áttekintettem a grandiózus Angular keretrendszer és a Firebase felhőszolgáltatás előnyeit, hátrányait és különbségeit más keretrendszerekkel és felhőszolgáltatásokkal szemben. Megismerkedtem a Stripe fizetési rendszerrel, melynek köszönhető, hogy az alkalmazás dinamikus tesztfázisú fizetési lehetőséget biztosít. A frontend alkalmazás dinamikusan kommunikál a Firebase-sel, amely folyamatosan lehetővé teszi a kollekciók adatainak és a tárolóban raktározott képek gyors elérését.

A késznek nyilvánított alkalmazás megvalósított funkciói többszörösen tesztelve voltak, melyek mindegyikének működése bizonyosságot nyert.

Irodalomjegyzék

- [1] Angular, <https://angular.io>, 2023. 12. 08.
- [2] Visual Studio, <https://visualstudio.microsoft.com/>, 2023. 12. 08.
- [3] Firebase, <https://firebase.google.com/docs/>, 2023. 12. 08.
- [4] Tailwind, <https://tailwindcss.com/docs/>, 2023. 12. 08.
- [5] Stripe, <https://stripe.com/docs/>, 2023. 12. 08.
- [6] Canvas, <https://www.javascripttutorial.net/web-apis/javascript-canvas/>, 2023. 12. 08.
- [7] Gencraft, <https://gencraft.com/>, 2023. 12. 08.
- [8] Toastr, <https://www.npmjs.com/package/@types/toastr/>, 2023. 12. 08.
- [9] TypeScript, <https://www.typescriptlang.org/docs/>, 2023. 12. 08.
- [10] HTML, <https://www.w3schools.com/html/>, 2023. 12. 08.

Nyilatkozat

Alulírott Pál Krisztián Zoltán programtervező informatikus szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet Szoftverfejlesztési Tanszékén készítettem, Programtervező informatikus BSc diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat a Szegedi Tudományegyetem Diplomamunka Repozitóriumban tárolja.

Szeged, 2023. december 15.

Pál Krisztián Zoltán