



EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

PROGRAMOZÁSI NYELVEK ÉS FORDÍTÓPROGRAMOK

TANSZÉK

## Kisvállalatokat segítő webáruház Angular keretrendszerben

*Témavezető:*

Fekete Anett

PhD hallgató, MSc

*Szerző:*

Magyar Dorina

programtervező informatikus BSc

*Budapest, 2021*

**EÖTVÖS LORÁND TUDOMÁNYEGYETEM**  
**INFORMATIKAI KAR**

**SZAKDOLGOZAT TÉMABEJELENTŐ**

**Hallgató adatai:**

Név: Magyar Dorina

Neptun kód: JOMFGK

**Képzési adatok:**

**Szak:** programtervező informatikus, alapképzés (BA/BSc/BProf)

**Tagozat :** Nappali

Belső témavezetővel rendelkezem

**Témavezető neve:** Fekete Anett

*munkahelyének neve, tanárkör: ELTE-IK, Programozási nyelvek és Fordítóprogramok Tanszék  
munkahelyének címe: 1117, Budapest, Pázmány Péter sétány 1/C.      beosztás és iskolai végzettsége:  
PhD hallgató, MSc*

**A szakdolgozat címe:** Kisvállalkozásokat segítő webáruház Angular keretrendszerben

**A szakdolgozat téma:**

(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témajának leírását )

A cél egy kozmetikai termékeket értékesítő webshop alkalmazás megvalósítása. A weboldal tartalmazna egy vásárlói felületet, amin olyan mai oldalak elvárt funkcióit tartalmaznának, mint a chat funkció vagy a csomagküldés és fizetési felület választékos kínálata. Ezen felül a regisztrált felhasználóknak profil oldal biztosítása, amin láthatnák rendeléseiket és kedvelt termékeiket.

Az oldal kiegészülne egy adminisztrációs felülettel, amin keresztül a tulajdonos könnyedén és felhasználó barát felületen tudná szerkeszteni a webshop termékeit esetleges akciókat megadni és a híroldal cikkeit frissíteni. Továbbá analitikus szoftver segítségével a tulajdonos számára részletes statisztikát kapna a weboldal látogatottságáról és egyes oldalak népszerűségéről.

A szakdolgozat magába foglalja az adatháztartás kezelést, szerver és kliensoldali fejlesztéseket. A kliensoldal az Angular nevezetű keretrendszert felhasználásával valósulna meg.

A webalkalmazás végső célja, hogy kisvállalkozásoknak kezdeti felületet biztosítson a termékeik értékesítésére.

Budapest, 2021.06.01.

# Tartalomjegyzék

<b>1. Bevezetés</b>	<b>3</b>
<b>2. Felhasználói dokumentáció</b>	<b>4</b>
2.1. Alkalmazás indítása . . . . .	4
2.1.1. Alkalmazás indítása böngészőből . . . . .	4
2.1.2. Alkalmazás indítása saját gépről . . . . .	5
2.2. Alkalmazás kezelése . . . . .	6
2.2.1. Webshop felület kezelése . . . . .	7
2.2.2. Adminisztrációs felület kezelése . . . . .	9
2.3. Rendelési folyamat . . . . .	11
2.3.1. Rendelés leadása . . . . .	12
2.3.2. Vásárlással kapcsolatos információk . . . . .	16
<b>3. Fejlesztői dokumentáció</b>	<b>19</b>
3.1. Webalkalmazás specifikáció . . . . .	19
3.2. Kliensoldalon használt technológiák bemutatása . . . . .	20
3.2.1. Angular keretrendszer . . . . .	21
3.3. Kliensoldal működése . . . . .	21
3.3.1. Komponensek . . . . .	22
3.3.2. Modulok . . . . .	22
3.3.3. Interfészek . . . . .	22
3.3.4. Service files . . . . .	23
3.3.5. Admin mappa . . . . .	23
3.3.6. Pages mappa . . . . .	23
3.3.7. Oldalak közötti navigáció . . . . .	23
3.3.8. Shared mappa . . . . .	24
3.3.9. Egyéb fájlok és mappák . . . . .	24
3.4. Szerveroldalon használt technológiák bemutatása . . . . .	25

3.4.1. Node.js szoftverrendszer . . . . .	25
3.4.2. Express.js keretrendszer és Mongoose programozási könyvtár .	26
3.4.3. MongoDB adatbázisszerver . . . . .	27
3.4.4. NoSQL és SQL adatbázisok összehasonlítása . . . . .	27
3.4.5. Alkalmazás adatbázisának bemutatása . . . . .	28
3.5. Szerveroldal működése . . . . .	29
3.5.1. REST API - Végpont tervezés és implementáció . . . . .	29
3.5.2. Hitelesítés - Az adminisztrációs felület védelme . . . . .	34
3.6. Az alkalmazás megjelenése . . . . .	36
3.6.1. Figma szoftver - oldalvázlatok . . . . .	36
3.6.2. Logók és grafikai elemek . . . . .	37
3.7. Forráskódok . . . . .	38
3.7.1. Kliensoldali forráskódok . . . . .	38
3.7.2. Szerveroldali forráskódok . . . . .	40
3.8. Tesztelések . . . . .	43
3.8.1. Kliensoldal tesztelése . . . . .	43
3.8.2. Szerveroldal tesztelése . . . . .	44
3.9. Továbbfejlesztési lehetőségek . . . . .	45
<b>4. Összegzés</b>	<b>46</b>
<b>Irodalomjegyzék</b>	<b>47</b>
<b>Ábrajegyzék</b>	<b>47</b>
<b>Forráskódjegyzék</b>	<b>49</b>

# 1. fejezet

## Bevezetés

A WebBeauty (továbbiakban WB) lehetőséget kínál a kisvállalkozók által gyártott termékek bemutatására és árusítására. Napjainkban megnőtt a kereslet a személyes webáruházak létrehozása iránt, ahol a kereskedők saját termékeiket kívánják értékesíteni, ezt pedig a WB megfelelően kiszolgálja. Az webalkalmazás nem csak a felhasználók számára könnyen kezelhető, hanem egyben a tulajdonosoknak is. Lehetőségük van arra, hogy egyszerűen menedzselhessék termékeiket és kapcsolatot tartsanak a lehetséges vásárlókkal.

Számomra a téma választás célja egy személyesen ismert vállalkozó megkeresésén alapult, aki szívesen értékesítené az általam készített webáruházon keresztül a termékeit. Az alapproblémát az vetette fel részéről, hogy egyénileg nem tudnám kiszolgálni az üzlettulajdonos által érkező folyamatos frissítési kéréseket. Ezt a felmerülő nehézséget úgy próbáltam megoldani, hogy a vállalkozó által is kényelmesen elérhetővé tettem azokat a funkciókat, ami a webalkalmazás aktualizáltságát biztosítja. Továbbá a program képes a tulajdonos és a vásárló közötti közvetlen kapcsolat létszatát kialakítani a chatbot<sup>1</sup> funkció segítségével. Mivel ez egy egyszerűbb webalkalmazás, ezért nem egy teljesen egyénileg gondolkozó mesterséges intelligencia (MI)<sup>2</sup> alapú chatbotról esik szó, hanem egy adatbázisban tárolt, előre generált válaszokból álló szöveges adathalmazról beszélhetünk, ami kulcsszavas keresés segítései révén működik.

---

<sup>1</sup>egy szoftver alkalmazás, aminek a támogatásával közvetlen emberi kapcsolat helyett egy virtuális 'asszisztenssel' kommunikáljon.

<sup>2</sup>sokféle megközelítést találhatunk a definícióját illetően. Személy szerint azt gondolom, hogy az MI egy tudatos gondolkozásra alkalmas, emberi beavatkozás nélkül cselekvőképes létféle, amit a legtöbbször számítástechnikai eszközökhez/társítunk.

## 2. fejezet

# Felhasználói dokumentáció

Az alkalmazás készítése során fontos szempont volt, hogy egy felhasználóbarát webáruházat hozzák létre. A webshop használata egyszerű, letisztult felülettel rendelkező program olyan funkciókkal kiegészítve, amelyik megkönnyíti az átlagos vásárlók számára az oldal kezelését. A fejezet célja, hogy bemutassa az alkalmazás azon tulajdonságait, amelyek nem feltétlenül egyértelműek egy hétköznapi kliens számára, ezzel is elősegítve a webalkalmazás gördülékeny felhasználását.

### 2.1. Alkalmazás indítása

Egy hétköznapi felhasználó számára talán ez a legnagyobb kihívás a program használatával kapcsolatban. Az alkalmazás indítására két módszer közül választhatunk, melyek a következők:

1. Megnyitni böngésző segítségével a weboldalt: 2.1 fejezet
2. Megnyitni localhostról a projektet: 2.1.1 fejezet

Mindkettő technika részletes bemutatásra kerül a következő alfejezetekben.

#### 2.1.1. Alkalmazás indítása böngészőből

Az első és legkönnyebben alkalmazható stratégia, hogy valamilyen előre telepített webböngésző (Chrome, Firefox, Opera..stb.) segítségével megnyitjuk az Amazon AWS[aws] szerverére telepített weboldalt. A felület ezen az URL-en érhető el: <http://webbeauty.us-east-2.elasticbeanstalk.com>

### 2.1.2. Alkalmazás indítása saját gépről

Az előző módszert azért neveztem könnyebben alkalmazhatónak, mert ha saját gépről szeretnénk indítani az alkalmazást, nem csak le kell klónoznunk azt a GitHubról, hanem több különböző szoftvert kell telepítenünk, mielőtt el tudnánk indítani a projektet. A program fordításához szükségünk lesz a *Node.js* szoftverrendszerre [**nodejs:online**]. Továbbá elengedhetetlen a gépünkről az Angular CLI[**cli**]. Ez egy parancssori interfész az Angular keretrendszerhez. A kód futtatásához szükségünk van rengeteg eszközre, amely lefordítja és optimalizálja a kódot, az Angular CLI ezt biztosítja a projekt számára. A kód fordításához szükségük lesz egy integrált fejlesztői környezetre (IDE). Én a Visual Studio Code[**vs:code**] (VS Code) nyílt forráskódú kódszerkesztőjét használtam az alkalmazás elkészítése során, így ezt fogom bemutatni. A következő felsorolásban összefoglalom azokat a lépéseket, amelyek segítségével eljutunk a program saját gépről való indításáig.

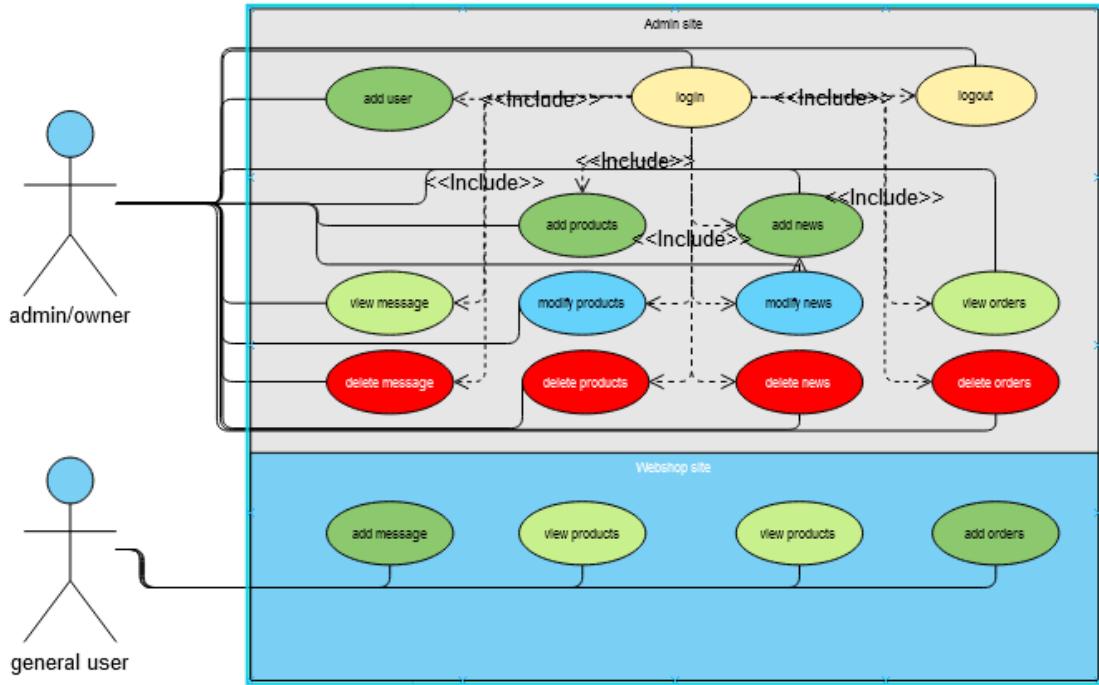
1. A *Node.js* oldaláról[**nodejs:online**] töltsük le és telepítsük a számítógépre megfelelő .exe kiterjesztésű fájlt.
2. Az általam felöltött és publikusan elérhető GitHub repositoryt[**szakdolgozat2021**] klónozzuk le az eszközünkre.
3. Töltsük le és telepítsük a Visual Studio Code[**vs:code**] nevezetű programot.
4. Nyissuk meg a VS Code alkalmazást és telepítsük a következő bővítményeket: Angular Essentials, Material Icon Theme.
5. A Visual Studio Code segítségével töltsük be a leklónozott projekt mappáját, és nyissunk meg egy új terminált.
6. A terminálban lépjünk be a *szakdolgozat* nevezetű mappába és futtassuk le a következő parancsot: `npm install -g @angular/cli`
7. Mivel az Angular CLI önmagában nem elegendő, futtassuk le a következő parancsot: `npm i`. Ennek a parancsnak a segítségével letöltődik minden kiegészítő fájl, ami használatban van a programon belül.
8. Mielőtt parancssor segítségével elindítanánk az alkalmazást be kell lépnünk a projekthez tartozó MongoDB[**mongo**] adatbázisba. Itt hozzá kell adnunk a

*hálózati hozzáférés* nevezetű menüpont alatt a gépünk IP címét, különben nem tud csatlakozni a szerveroldalunk az adatbázishoz.

9. A meglévő terminálunk mellé nyissunk meg egy újat, és futtassuk le az egyikben az `npm run start`, a másikban az `npm run start:server` parancsot. az utóbbi a kliensoldalt, míg az előbbi a szerveroldali kódokat futtatja és fordítja le.
10. Ha sikeresen lefordult a kód, akkor a böngésző URL helyére a `localhost:4200` címet begépelve láthatjuk a webáruház oldalát.

## 2.2. Alkalmazás kezelése

Az alkalmazás felületét két részre bonthatjuk. Az alábbi 2.1-es ábrán láthatjuk ezt a két rendszert, amiben a felhasználó, tulajdonos és az admin által elért funkciókat foglalja össze use case diagram segítségével. Az első rész maga a webáruház, amin a vásárlók megtekinthetik a termékeket és megrendelhetik őket, továbbá megnézhetik az üzemeltető által közzétett híreket, ezen felül különböző forrásokból információkat érhetnek el a vásárlással kapcsolatban. A második rész az üzemeltető által karbantartott adminisztrációs oldal. Ennek használatához hitelesítés szükséges, ezzel is védve a vásárlók és a webáruház adatait. Az admin felületen lehetőség van ezen adatok kezelésére, szerkesztésére.

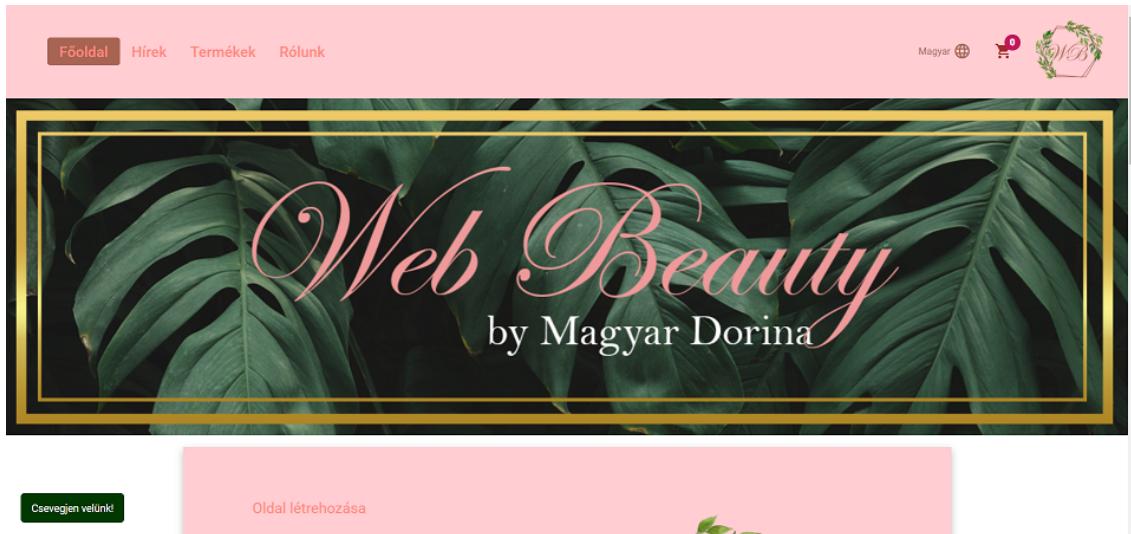


2.1. ábra. Use case diagram - Alkalmazás funkciói

### 2.2.1. Webshop felület kezelése

A webáruház kezelése igen egyszerű az átlagos felhasználó számára. Számos funkcióval rendelkezik az alkalmazás, amelyek elősegítik, hogy egy letisztult, felhasználobarát programot használjanak a vásárlók. Az áruház szerkezeti felépítése három fő részből áll:

**Menüsor** vagy más néven *toolbar*. A 2.2-es ábrán a webáruház főoldalának egy részét láthatjuk. A kép tetején található az alkalmazás webshopjához tartozó vezérlőfelület, ami a programban *toolbar* néven található meg. A vezérlőfelület bal oldalán láthatóak különböző menüpontok, amelyek segítségével navigálhatunk a különböző oldalak között. A jobb oldalán különböző funkciókkal bíró ikonokat és az oldal logóját láthatjuk. Az első ilyen ikon a nyelvválasztó, ami nek a segítségével megváltoztathatjuk az oldal nyelvét. Jelenleg az angol és a magyar nyelv közül lehet választani. A mellette lévő bevásárlókocsit ábrázoló ikon segítségével érhető el a felület kosár funkciója, ami tartalmazza az eddig hozzáadott termékeket. A kosárban szereplő termékek számáról előzetes információt kaphatunk a felette lévő jelvény számból.



2.2. ábra. Az alkalmazás megjelenése

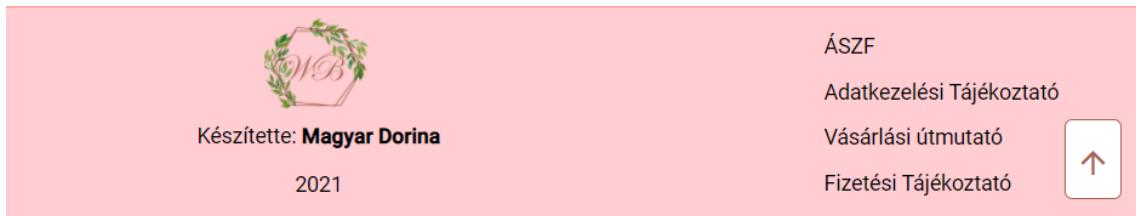
**A webáruház tartalma** (*body*). Ez a menüsor alatt található meg, amit a fentebb kifejtett vezérlőfelület segítségével különböző oldalak tartalmi részét jeleníthetjük meg. Erre példákat 2.3-as ábra a és b részén láthatunk.

(a) Termékek oldala

(b) Hírek oldal

2.3. ábra. Webáruház tartalmi része

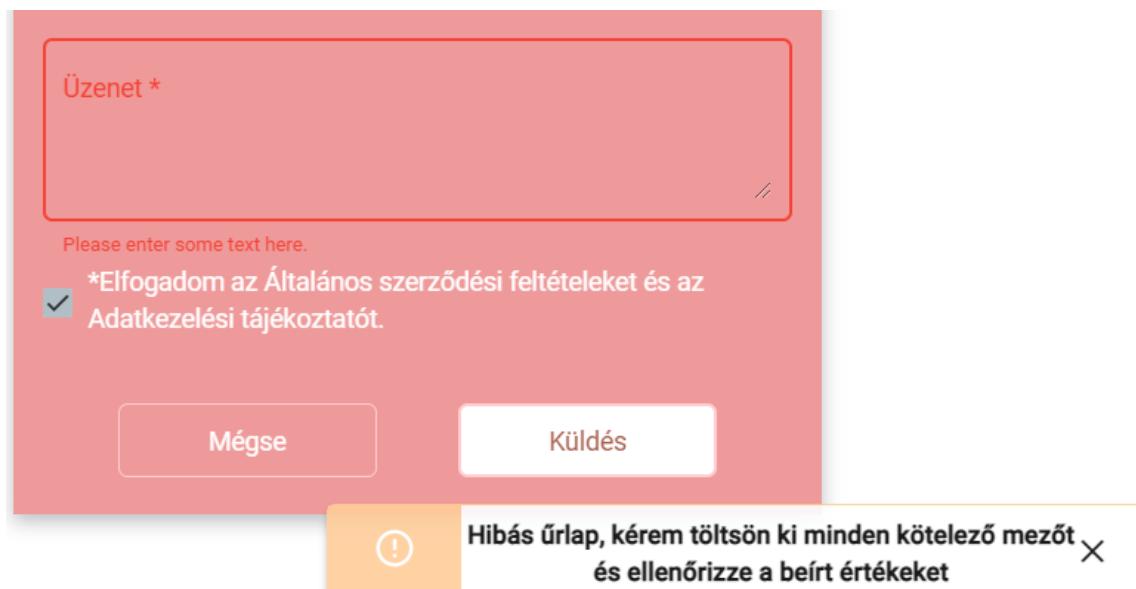
**Lábrész** (*footer*). A 2.4-es képen ez a felület látható. Itt olyan oldalak linkjei jelennek meg, amik tartalmazzák a vásárlókra és az oldalra vonatkozó fogyasztóvédelmi törvényeket és általános tájékoztatókat.



2.4. ábra. Az oldal lábrésze

**A chatbot funkció** a 2.2-es ábra legalján található *Csevegjen velünk!* feliratú zöld gomb segítségével érhető el, aminek a bemutatását a 2.3.1-es Vásárlással kapcsolatos információk alfejezetben kívánok kifejteni.

**Az alert üzenet** olyan információs felület, amik a felhasználó számára értesítést küld egy-egy feladat befejezésével kapcsolatban. Ilyen üzenetek lehetnek például, ha nem sikerül a betöltött űrlap feldolgozása, mint ahogy azt a 2.5-ös ábrán is láthatjuk.



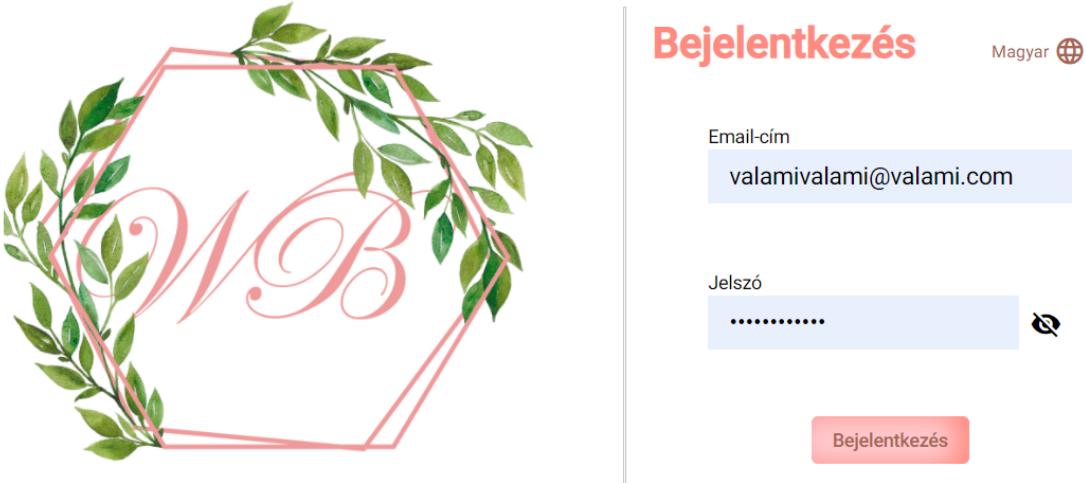
2.5. ábra. Alert üzenet

### 2.2.2. Adminisztrációs felület kezelése

Az oldal eléréséhez szükség van a helyesen begépelt URL megadására a böngészőnkön keresztül, ami

- localhost esetén: <http://localhost:4200/admin/login>
- AWS szerverre telepített weboldal esetén: <http://webeauty.us-east-2.elasticbeanstalk.com/admin/login>

Ha helytelenül gépeljük be a megadott webcímét, akkor az alkalmazás visszanavigál minket az oldal kezdőlapjára, viszont ha sikeresen begépeltük az adminisztrációs felület eléréséhez szükséges URL-t, akkor a 2.6-os ábrán látott oldalt láthatjuk.



2.6. ábra. Bejelentkezási oldal

Az adminisztrációs felület kezeléséhez szükségünk van az oldalt védő autentikáció feloldásához. Ez azt jelenti, hogy először be kell lépnünk a fejlesztő által megadott, vagy az üzemeltető által hozzáadott email cím és jelszó párossal az előbb említett címek valamelyikén. Sikeres bejelentkezással az oldal átnavigálásra kerül az admin felületre. Ez az oldal két fő részből áll: a vezérlési és a tartalmi részből.

**Vezérlési** rész, aminek segítségével megjeleníthetjük az alkalmazás tartalmi részét.

A 2.7-es képen ez a navigációs rész a bal oldalon található. Az admin felületen képesek vagyunk új termékeket vagy híreket hozzáadni, és a meglévőket lista formájában megtekinteni. Továbbá a fentebbiekben már említett kapcsolatfelvétel céljából a felhasználó által elküldött üzenetek is listázásra kerülnek az oldalon. Az alkalmazáson belül lehetőségünk van új felhasználói fiókot hozzáadni, viszont ezek a fiókok adatvédelmi okból nem megtekinthetők és nem törölhetők, csak adatbázis szinten. Ezen felül lehetőség van a bejelentkezett fiókot kijelentkeztetni, így visszalépve bejelentkezés hiányában nem tekinthetjük meg újból ezt a felületrészt.

	Id	Név	Leírás	Ár	Kép	Menü
	61b62241b415f9f7cc07e7b4	Fém szívószál	Környezetkímélő fém szívószál. Igényes, könnyen kezelhető tartós szívószál.	HUF4,200		...
	61b62316b415f9f7cc07e7bc	Smink ecset	Puder ecset. Kiválló minőségű smink puder ecset.	HUF2,100		 Módosítás  Törlés
	61b62474b415f9f7cc07e7c0	Arcszérum	10 % Niacinamide + 1 % Zinc szérum a kitágult pórusokra, a bőrhibákra hajlamos zsíros bőr minden nap ápolására ideális, és hatékonyan szünteti meg ezeket a tüneteket	HUF4,950		...
	61b625b4b415f9f7cc07e7c4	Arcápolás csomag	10 % Niacinamide + 1 % Zinc szérum és hidratálókrém egybe	HUF8,500		...
	61b6269bb415f9f7cc07e7c8	Hajápolás csomag	Hajápolási csomag most kedvező áron elérhető. Shampoo, Balzsam és hajmaszk	HUF12,500		...

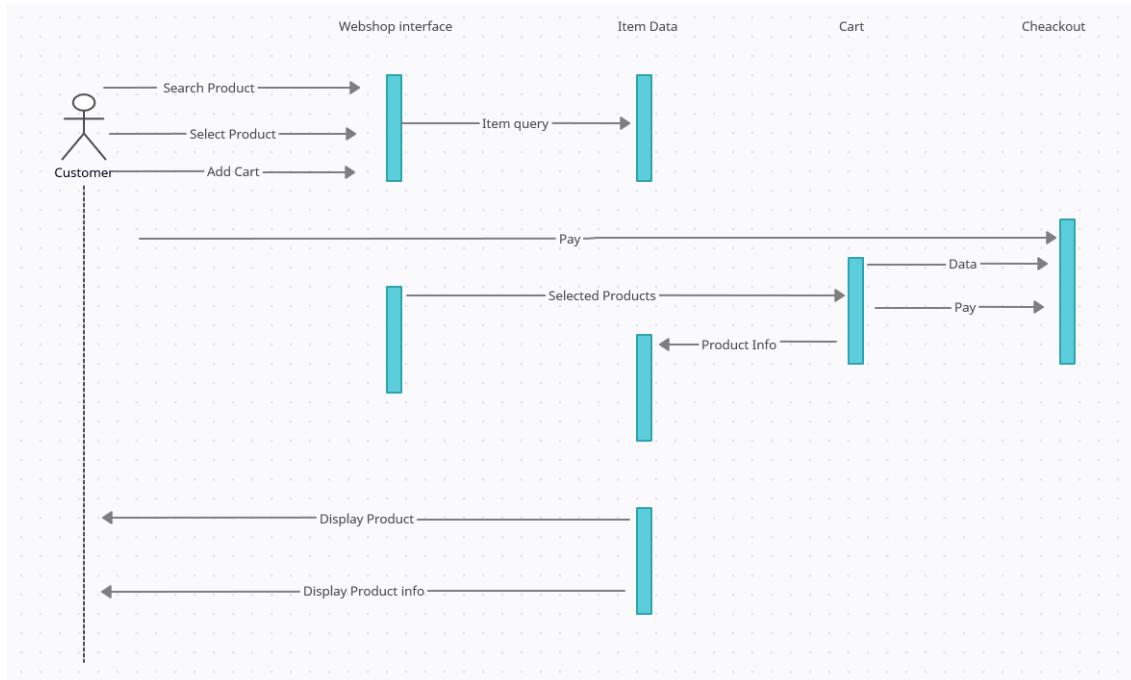
Items per page: 5 ▾ 1 – 5 of 6 | < < > >|

2.7. ábra. Adminisztrációs felület megjelenése

**A tartalmi rész** például a 2.7-es ábra jobb oldalán látott táblázat, ami a termékek listáját tartalmazza. Az oldal ezen részén lehetőségünk van az adatok módosítására, törlésére vagy a fentebb említett termék hozzáadására a megjelenő űrlap kitöltésével.

### 2.3. Rendelési folyamat

A rendelés folyamata hasonlóan zajlik, mint a legtöbb webáruházból való rendelés. A kiválasztott termékek bekerülnek a kosárba, majd a személyes adatok megadásával és a vásárlás befejezésével elküldésre kerül az oldal üzemeltetője számára, aki előkészíti azt. Az alábbi 2.8-as kép pontosan ezt a folyamatot mutatja be szekvencia diagram segítségével.

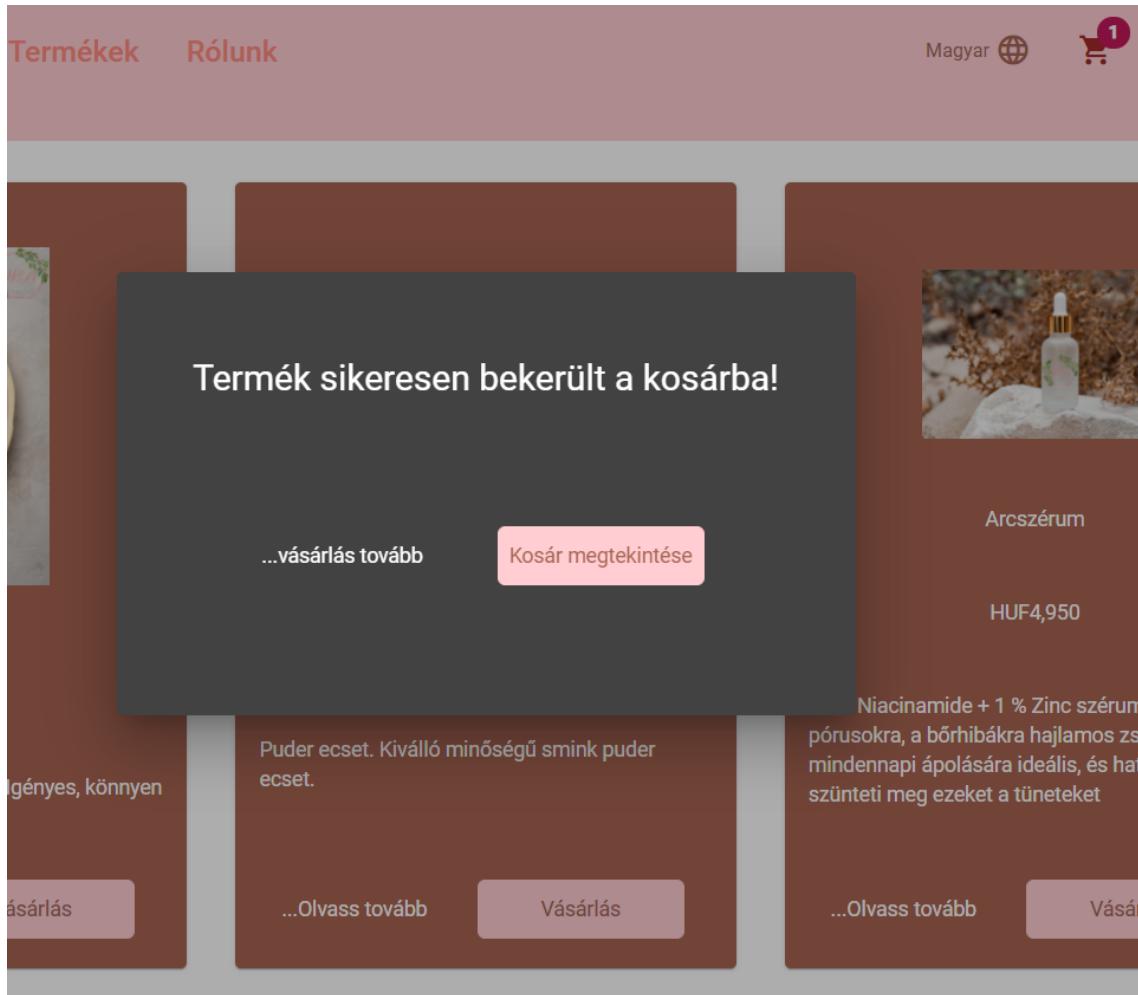


2.8. ábra. Rendelési folyamat bemutatása - szekvencia diagram

A következő két alfejezetben bemutatom egy példán keresztül, hogyan is kerül egy rendelés leadása a termék kiválasztásától kezdve. Továbbá milyen feltételei és információ vonzatai van a webáruházon történő vásárlásnak.

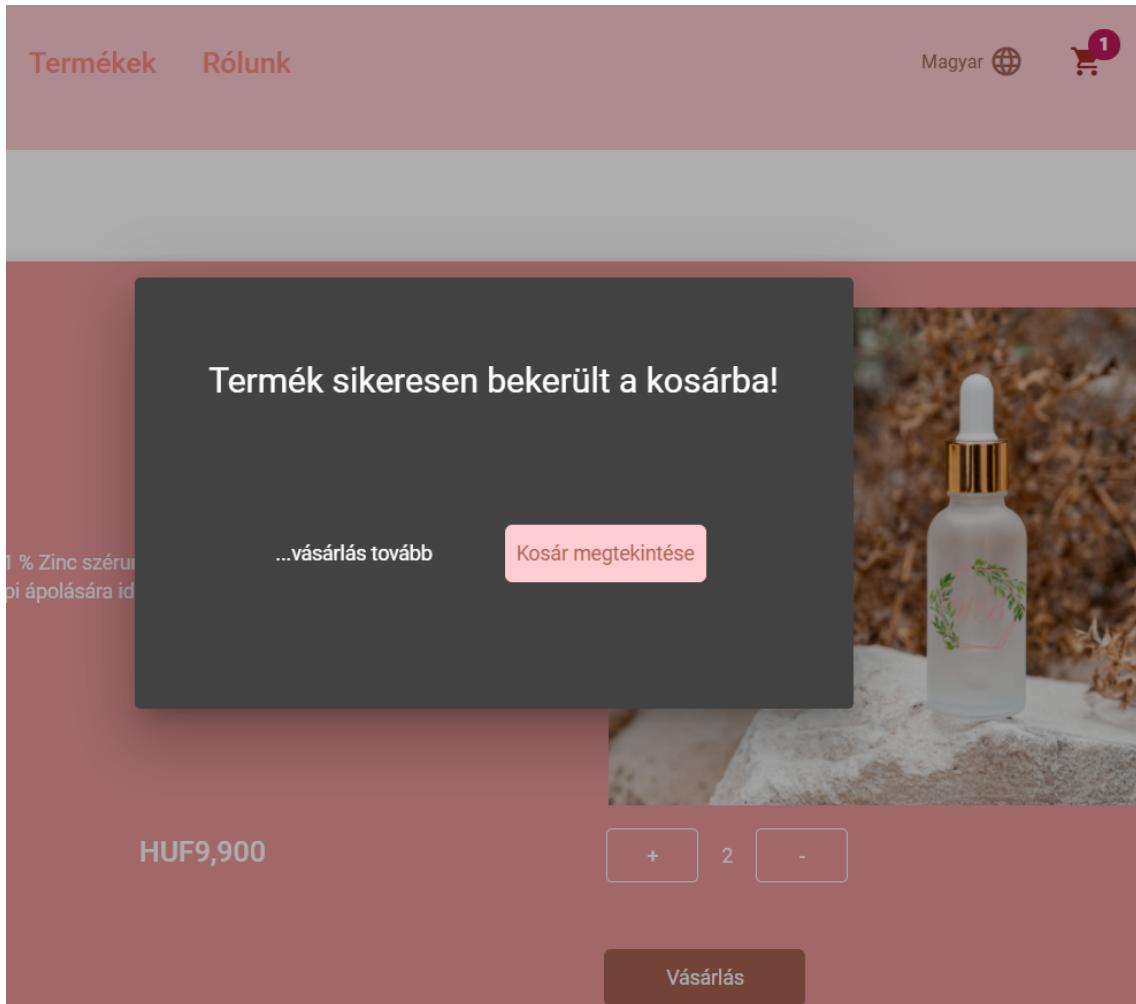
### 2.3.1. Rendelés leadása

A rendelés leadásához először is szükséges, hogy legalább egy termék bekerüljön a bevásárlókosár tartalmába. Egy termék kosárba helyezésére két lehetőségünk van. Az első, ha a termékek listájánál a kiválasztott termék kártyáján a Vásárlás gombra kattintunk, mint ahogyan a 2.9-es ábrán is megfigyelhető.



2.9. ábra. Termék vásárlása a termékek listáján keresztül

A második lehetőségünk az áru bevásárlókocsiba helyezésére, hogy a kiválasztott termék profil oldalán kattintunk a Vásárlás gombra. A különbség a két alternatíva között, hogy a második hozzáadás során megadhatjuk a kiválasztott áru mennyiségett, ahogy azt a 2.10-es második kép példáján is láthatjuk.



2.10. ábra. Termék vásárlása a termék profilján keresztül

Ha a kiválasztott termékek a kosárba kerültek, azokat megtekinthetjük úgy, ha a 2.9-es és a 2.10-es ábrákon is látott Kosár megtekintése gombot kiválasztjuk, vagy ha magára a bevásárlókocsi ikonjára kattintunk. Mindkettő lehetőség átirányít a kosár oldalára, ahol láthatjuk táblázat formájában a megvásárolandó árukat, mint azt a 2.11-es képen is láthatjuk.

[← Vissza](#)

## Bevásárlókocsi

Név	Ár	Kép	Mennyiség	Ár
Smink ecset	HUF2,100		<a href="#">+</a> <input type="text" value="1"/> <a href="#">-</a>	HUF2,100
Arcszérum	HUF4,950		<a href="#">+</a> <input type="text" value="2"/> <a href="#">-</a>	HUF9,900

Összes termék darab száma és ára: 3 db HUF12,000

[Tovább vásárolok](#)

[Vásárlás](#)

2.11. ábra. Bevásárlókosár

A bevásárlókosár tartalmának módosításra lehetőségünk van a mennyiség oszlopában látható gombok segítségével. Ha csökkentjük vagy növeljük egy áru mennyiségét, akkor egyes adatok dinamikusan megváltoznak.

A táblázat alatti Vásárlás gombot kiválasztva az oldal átnavigál a fizetési felületre. A 2.12-es képeken látható, hogy ezen a felületen négy lépegető gomb található. minden lépegetőnek külön funkciója van, amit a nevével azonosíthatunk.



(a) Első lépegető gomb



(b) Második lépegető gomb

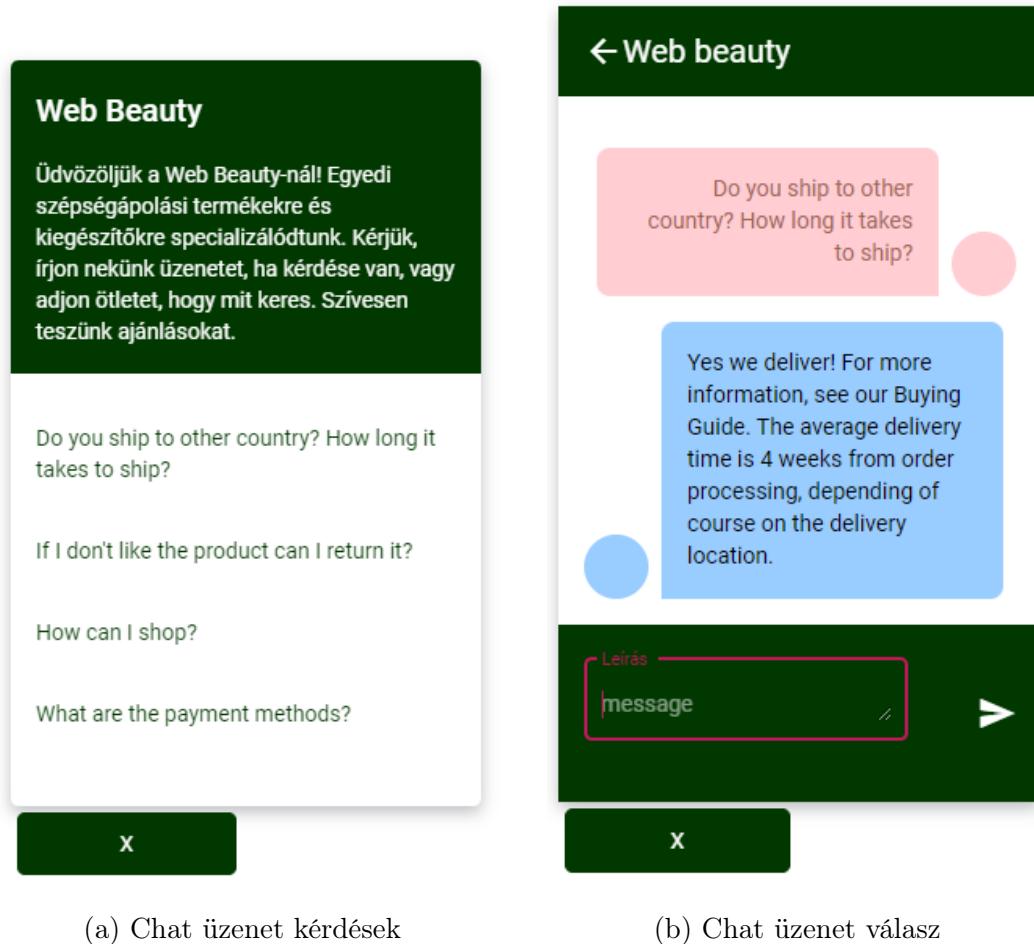
2.12. ábra. Fizetési felület

Az első ilyen gombon adhatóak meg a szállítási és számlázási adatok űrlapon keresztül, aminek a helyes kitöltésével léphetünk a következő lépésre. A második lépegető gombon a szállítással és fizetéssel kapcsolatos információkat adhatjuk meg. A további lépegetőkön egy összefoglalót és visszaítézt kapunk a leadott rendeléssel kapcsolatban.

### 2.3.2. Vásárlással kapcsolatos információk

Az alkalmazáson belül számos információ rendelkezésre áll a vásárlók számára. A programban használt vásárlással kapcsolatos információforrások a következők:

- Chatbot: segítségével lehetőség nyílik a vásárlás során felmerülő kérdések feltevésére. Ezekre a kérdésekre a válaszadás dinamikus formában történik. Tekintsük meg az alábbi 2.13-as ábrán illusztrált generált válaszadás folyamatát. A kép baloldalán látható, ahogy kiválasztjuk a feltevő kérdést és a kép jobb oldalán megválaszolásra kerül. Ha további kérdése van a vásárlónak a program megkéri, hogy forduljon az ügyfélszolgálathoz a megadott elérhetőségeken.



(a) Chat üzenet kérdések

(b) Chat üzenet válasz

2.13. ábra. Chat funció bemutatása

- Általános Szerződési Feltételek minta: az Általános Szerződési Feltételek (rövidítve: ÁSZF) a vásárló és a szolgáltató közötti megállapodási szerződés. A szerződésbe foglalt két fél jogairól szól lényegében. Az oldalon található ÁSZF csak mintaként szolgál, mivel ennek megírása igen magas jogi ismereteket, akár jogász által elkészített dokumentumot igényel. Jelen esetben az oldalon található ÁSZF tartalma lementhető PDF formátumba. A 2.14-es ábrán látott Letöltés PDF gombra kattintva az aszf.pdf kiterjesztésű fájl letöltésre kerül a gépünkre.

The screenshot shows a pink header bar with a menu icon, language selection (Magyar), a shopping cart icon with '0' items, and a logo. Below the header, there's a red button labeled 'Letöltés PDF'. The main content area has a white background and features a title in bold: 'Általános Szerződési Feltételek (ÁSZF) minta'. Below the title is a detailed text about the document's purpose and scope. A section titled 'Szolgáltató adatai:' lists various company details. At the bottom of this section, there's a green button with white text: 'Csevegjen velünk! Elkezések'. The footer contains a PDF icon and the file name 'aszf.pdf'.

Az Általános Szolgáltatási Feltételek („ÁSZF”) tartalmazzák "Web Beauty" cég (székhely: ..... adószám: .....), mint szolgáltató ("Szolgáltató") által üzemeltetett webáruház használatára vonatkozó általános szerződési feltételeket. Kérjük, hogy csak akkor vegye igénybe szolgáltatásainkat, amennyiben minden pontjával egyetért, és kötelező érvényűnek tekinti magára nézve. Jelen dokumentum nem kerül iktatásra, kizárolag elektronikus formában kerül megkötésre (nem minősül írásba foglalt szerződésnek), magatartási kódexre nem utal.

**Szolgáltató adatai:**

- A szolgáltató neve:
- A szolgáltató székhelye:
- A szolgáltató elérhetősége, az igénybe vevőkkel való kapcsolattartásra szolgáló, rendszeresen használt elektronikus levelezési címe:
- Cégjegyzékszáma:
- Adószáma:
- Nyilvántartásban bejegyző hatóság neve (cégbíróság):
- Telefonszámai:
- Adatvédelmi nyilvántartási száma:
- Engedély száma:
- A szerződés nyelve: magyar

Csevegjen velünk! Elkezések

2.14. ábra. ÁSZF letöltése

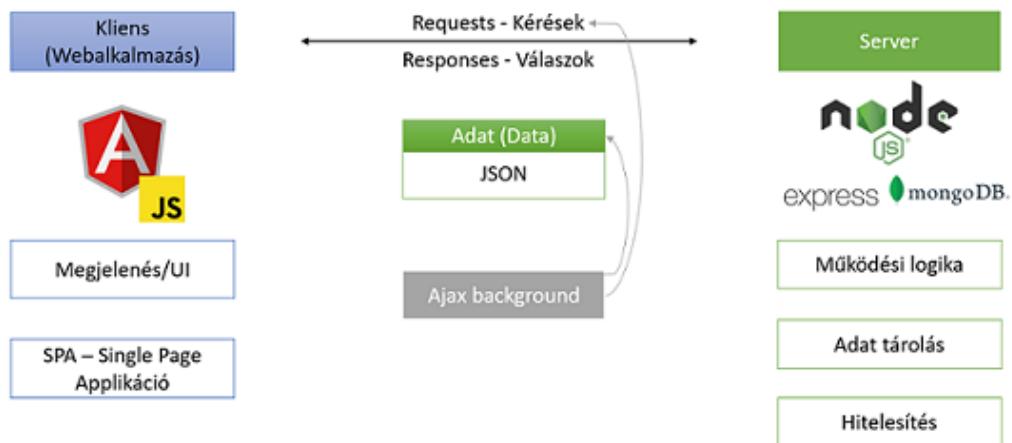
- Adatkezelési tájékoztató: esetében az oldalon történő adatok tárolásáról szóló dokumentum. Az oldalon található adatkezelési tájékoztató ugyan úgy mintaként szolgál, mint az ÁSZF dokumentum. Erre is komoly jogszabályok vonatkoznak, aminek megírásához jogi tudás vagy egy jogász segítsége elengedhetetlen.
- Vásárlási útmutató: ez az oldal a vásárlás lépéseinak részletes leírására szolgál.
- Fizetési tájékoztató: az oldalon található vásárláskor igénybevehető szolgáltatások leírását tartalmazza, aminek segítségével információt szerezhet a vásárló a szállítással kapcsolatos díjazásokra és fizetési lehetőségekre.

### 3. fejezet

## Fejlesztői dokumentáció

### 3.1. Webalkalmazás specifikáció

Az alkalmazás fő célja egy olyan működő webáruház bemutatása, ami MEAN (MongoDB, Express.js, Angular, és Node.js - solution stack)**[mean:stack]** nevezetű szoftverköteg segítségével készült. A MEAN megoldásverem ingyenes, nyílt forrás-kódú szoftverek halmaza, ami lehetőséget kínál dinamikus weboldalak készítésére. A webalkalmazás két fő részre osztható: kliensoldali és szerveroldali (idegen nyelven: front-end és back-end) részre. A kliensoldal leglényegesebb feladata, hogy a felhasználó által is látott weboldalt megjelenítse grafikai UI/UX (User Interface/User Experience) dizájn implementálásával. Nevezetesen egy olyan rendszer, ami képes a felhasználó számára felületet és élményt biztosítani. Miközben a szerveroldal elsődleges feladata az alkalmazás úgynevezett business logikájának a megvalósítása. Ezen felül képes az adatok feldolgozására és hitelesítésére is. A 3.1-es ábrán szeretném bemutatni, milyen módon épül fel a webalkalmazás, továbbá megismertetni a két oldal kommunikációs kapcsolatát.



3.1. ábra. Az alkalmazás rétegeinek bemutatása

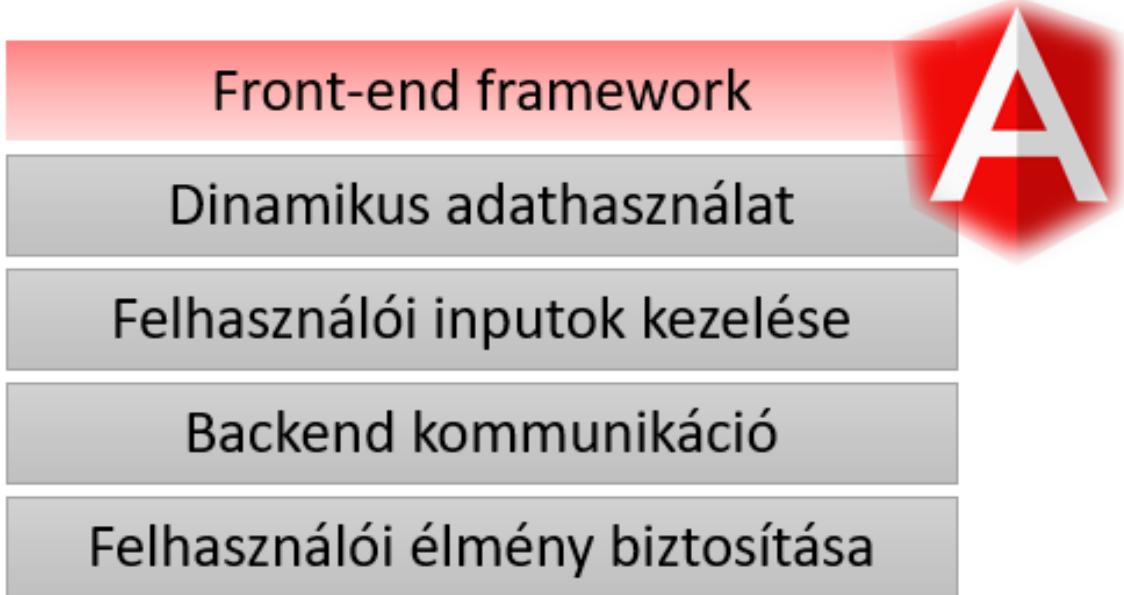
A 3.1-es ábrán látható, hogy a két oldal miképpen osztja meg az információkat egymás között. A front-end, pontosabban mondva a kliensoldal Angular[angular:online] keretrendszerben, TypeScript[ts] segítségével íródott. A front-end kommunikációja úgynevezett requestekkel, más néven kérésekkel (JSON formátumú adattovábbítással) a háttérben aszinkron módon történik, amire a szerveroldal response-okkal, vagyis válaszokkal felel. A back-end Node.js[nodejs:online] szoftverrendszer alapú, ami Express.js[express] keretrendszer segítségével íródott. Az adatok tárolásáért a MongoDB[mongo] nevezetű adatbázis felel.

A következő blokkokban szeretném tételesen bemutatni a fentebb említett front-end és back-end oldalakon használt módszerek alkalmazását és működését. Ezen felül szándékomban áll ismertetni az általam alkalmazott szoftverek technikai tulajdonságait.

## 3.2. Kliensoldalon használt technológiák bemutatása

A klienoldal megjelenítéséhez az Angular keretrendszer használtam, aminek a segítségével dinamikus webalkalmazások hozhatóak létre. Az Angular egy nyílt forráskódú, Google által fejlesztett, JavaScript nyelven írt front-end keretrendszer. Ebben a fejezetben szeretném kellőképpen kifejteni, miért ezt a rendszert választottam az alkalmazás megírásához, ezen felül alaposabban bemutatni a működését és főbb tulajdonságait a 3.2-es ábra segítségével.

### 3.2.1. Angular keretrendszer



3.2. ábra. Az Angular keretrendszer bemutatása

Az Angular egy kliensoldali keretrendszer, ennek köszönhetően képes feldolgozni és megjeleníteni a back-end felől érkező adatokat, aminek segítségével egy dinamikus webalkalmazást láthatunk a böngészőnkön keresztül. Ezt úgy tudja biztosítani, hogy képes kapcsolatot kialakítani a szerveroldallal. Továbbá lehetőséget nyújt a felhasználó által beérkezett adatok fogadására és kezelésére, ezen tulajdonga segítségével egy modern UI/UX felület készítésére alkalmazhatjuk. A program megírása során a 13.0.4 legújabb verziójú Angular CLI-t telepítettem. Mindezen jellemzői hozzátesznek ahhoz, hogy Single Page Application-nek (rövidítve: SPA)[spa] nevezzük az általa támogatott weboldalakat. Olyan webhelyeket hívhatunk SPA-nak, amelyek egyetlen oldalra töltik be dinamikusan az adatokat, más szóval minden elemük egy oldalon található. Ennek köszönhetően a weboldalon való navigáláshoz nem kell betölteni külön DOM, vagyis Dokumentumobjektum-modelleket.

## 3.3. Kliensoldal működése

A weboldal felépítéséért és megjelenéséért a HTML, TypeScript és SCSS programozási nyelvek felelnek. A HTML feladata az alkalmazás tartalmi megjelenítése, az SCSS feladata pedig ezen tartalom formázása. A TypeScript biztosítja a felhaszná-

ló által kiadott utasítások végrehajtását. Az Angular keretrendszerben ez a három nyelv együttesen segít az oldalak megjelenítésében.

### 3.3.1. Komponensek

A komponensek[**components**] a programkód logikai darabjai. Két fő eleme van:

1. Sablon: az alkalmazás megjelenítéséért felel, ez tartalmazza a HTML-t
2. Osztály: ebben szerepelnek a metódusok és a tulajdonságok, ez a rész a TypeScript fájlokban van definiálva

### 3.3.2. Modulok

Az Angular alkalmazás moduláris és saját moduláris rendszerrel rendelkezik, amit NgModulesnak[**ngmodules**] nevezünk. Ennek a modulnak a metaadatai segítségével különböző komponenseket, direktívákat és szolgáltatási(angolul: service) fájlokat csoportosíthatunk. Az NgModule-nak öt ilyen metaadata van, aminek a használatával kategorizálhatjuk a komponenseinket felhasználás szerint.

Az NgModule metaadatai:

- deklarációk (declarations): az itt szereplő komponensek kifejezetten ahhoz a modulhoz tartoznak, ahol létrehoztuk őket.
- exportok (exports): a deklarációban használt komponensek azon részhalmaza, amiknek láthatónak kell lennie máshol létrehozott komponensek számára.
- importok (imports): olyan modulokat tartalmaz, amiket a deklarációtól implementált komponensek használnak.
- szolgáltatók (providers): olyan osztályok szerepelnek itt, amelyek létrehozzák és menedzslik a service objektumokat első alkalomkor, amikor az Angularnak szüksége van a függősségek feloldásához.

### 3.3.3. Interfészek

Az interfészek[**interface**] olyan specifikációk az Angular keretrendszerben, amelyek egy osztály által megvalósítandó tulajdonságok és metódusok összefüggő halmozát határozzák meg. Tehát a segítségével létrehozható pár alapvető szabály a tulajdonságokra és a metódusokra, amiket használunk az osztályon belül.

### 3.3.4. Service files

A projektben a service files[**service**] tartalmazza azokat a függvényeket, amiknek a segítségével kapcsolatot alakíthatunk ki a szerveroldallal. Ezek a fájlok tartalmaznak bizonyos kéréseket, amiknek a segítségével a felhasználó elindíthatja az adatlekérés folyamatát. A függvények kigyűjtésének célja, hogy egyszerűbben elérhetőek legyenek több komponens számára, ezzel lehetőséget biztosít a többszöri felhasználásra. Továbbá egyszerűsíti a metódusok megváltoztatását más keretrendszer használata esetén.

### 3.3.5. Admin mappa

Ez a mappa tartalmazza az adminisztrációs oldal megjelenítéséhez szükséges fájlokat, elkülönítve a többi komponenstől. Erre azért volt szükség, mert a két oldal szerkezeti felépítése eltérő egymástól, továbbá segítette a fejlesztés során ezeket elszeparálva tartani.

### 3.3.6. Pages mappa

A webalkalmazás olyan komponensei találhatóak ebben a mappában, amik nincsenek bejelentkezés funkció mögé rejtve. Tehát az elérésükhez nem szükséges autentikáció. Az áruház öt fő oldala található meg itt.

- Főoldal/Kezdőlap: összefoglalja a hírek és a termékek oldalát.
- Hírek: az oldal üzemeltetője által megosztott fontosabb információk.
- Termékek: minden olyan termék, amit eladásra szánnak.
- Rólunk: az oldal üzemeltetőjével kapcsolatos információk, elérhetőségek.
- Bevásárlókosár: a felhasználó által kiválasztott termékek.

### 3.3.7. Oldalak közötti navigáció

Az oldalak közötti koordinálást[**navigation**] az előbbiekbén kifejtett modulok egyike kezeli. Angularban a legjobb mód az, hogy ha a routerbe betöltést és konfigurálást különállóan kezeljük. A router egy szolgáltatás, ami biztosítja az oldalak közötti navigációt. A konfigurálás az AppRoutingModule-ban zajlik, míg a betöltés a legfelső szintű modulban azaz az AppModule-ban van importálva, ami útválasztóként is szolgál.

### 3.3.8. Shared mappa

A Shared mappa tartalmazza azokat a komponenseket és Angular kiegészítő csomagokat, amik több oldalon is megjelenítésre kerülnek. Ezek a komponensek és package-ek a következők:

- Angular Material[**material**]: egy felhasználói felület (UI) komponens könyvtár. Az Angular Material használata gyorsítja a fejlesztési folyamatot és egy konzisztens, elegáns felületet biztosít.
- Alert üzenetek: segítségével a felhasználó által elindított folyamatok állapotáról nyújthatunk információt.
- Nyelvválasztás: lehetőséget biztosít az oldalon található adatok többnyelvű megjelenítésére.
- Vissza gomb és az Ugrás a tetejése gomb: a nevüköből kiindulva olyan gombok amik lehetőséget biztosítanak az oldalak navigálásra a felhasználók számára.
- Chatbot: animációval rendelkező beszélgetési felület, ami lehetőséget nyújt arra, hogy a felhasználó gyors információt szerezzen az adott szolgáltatásokkal kapcsolatban. A chatbot részletes bemutatása a (3.7)-es fejezet Kliensoldali forráskódok (3.7.1) alfejezetében található.

### 3.3.9. Egyéb fájlok és mappák

Ebben a blokkban szerepel minden olyan fájl, ami kiegészítő funkcióként szolgál a többi fájl számára.

- Assets mappa: minden olyan képfájlt tartalmaz, amit nem dinamikusan kapunk a szervertől. Ilyen képek például az oldalon használt logók és borítóképek.
- Enviroments mappa: az ebben szereplő fájlok tartalmazzák a szerveroldal eléréséhez szükséges URL címet.
- Theme mappa: olyan stílusfájl, ami a komponensekben használt színek gyűjteményét tartalmazza.
- **style.scss**: minden olyan stíluselem leírása, amit a komponensek közösen használnak.

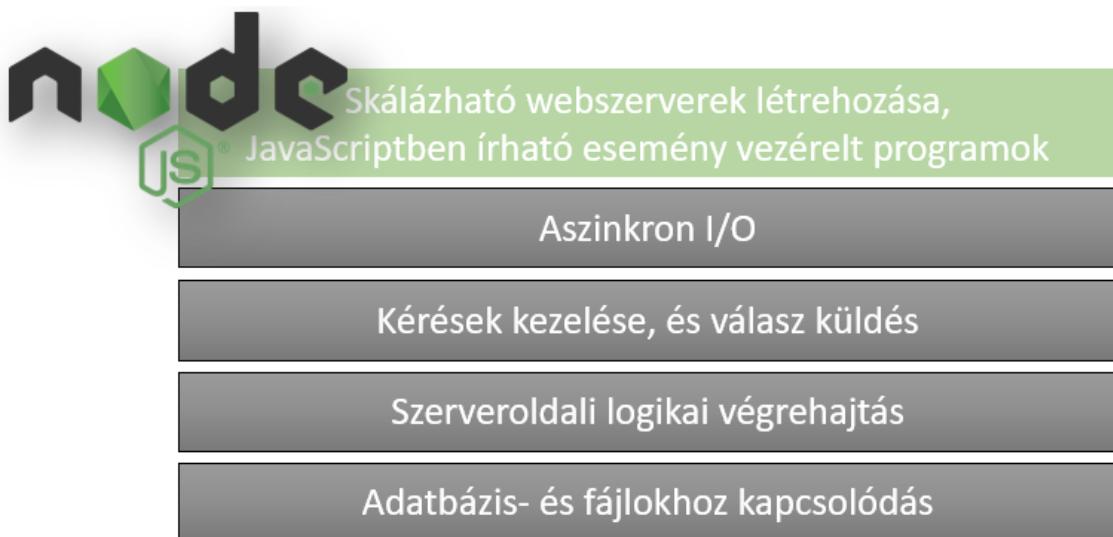
## 3.4. Szerveroldalon használt technológiák bemutatása

A szerveroldalon használt (a 3.1-es alfejezetben már említett) módszerek kiválasztása előtt kulcsfontosságú szempontnak tartottam, hogy az Angular keretrendszerhez megfelelő kompatibilitással rendelkezzenek, és alkalmazni tudjam őket a szakdolgozat elkészítése során. A következő technológiák minden olyan szoftverek, keretrendszerek vagy adatbázisszerverek, amikhez számtalan dokumentáció elérhető az interneten, ezzel támogatva a későbbiekben létrehozott projekteket. A következő felsorolásban összefoglalásképpen összegyűjtöttem az alkalmazásban fellelhető, általam használt szoftvereket és verziószámukat:

- Node.js: 14.15.4
- Express.js: 4.17.1
- Mongoose: 6.0.12
- MongoDB Atlas

A fejezet további részében szeretném ismertetni a fentebb felsorolt technológiák jellegzetes tulajdonságait, főbb jellemzőit további ábrák segítségével.

### 3.4.1. Node.js szoftverrendszer

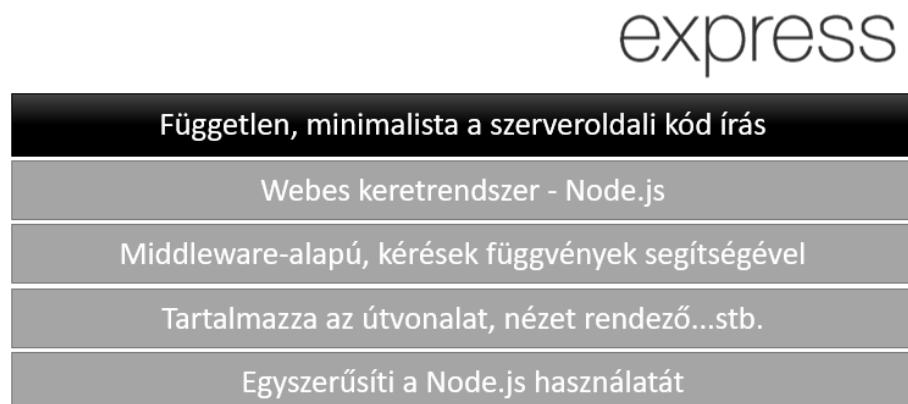


3.3. ábra. Node.js bemutatása

A webáruház back-endjének megírásánál 14.15.4-es verziójú Node.js szoftverrendszeret használtam. A 3.3-as ábrán látható a Node.js bemutatása, ami összefoglal-

ja a szoftverrendszer fontosabb jellemzőit. Az illusztráció első dobozában olvasható, hogy a Node.js skálázható webszerverek létrehozására alkalmas, más szóval egy olyan rendszert tudunk létrehozni a támogatásával, ami több felhasználót képes egyidejűleg kiszolgálni. Ezenfelül JavaScript nyelv segítségével olyan programok írhatóak, amelyek a komponensek közötti eseményinterakciókat tekintik alapul, más szóval eseményvezérelt programok megírására alkalmasak (ilyen például egy egérkattintás vagy billentyűleütés). Folytatónak a Node.js aszinkron tulajdonságával lehetővé teszi, hogy a kliensoldalról érkező kérések várakozási sorrendbe kerüljenek, ennek következtében a kliensoldal tovább folytathatja a feladatát.

### 3.4.2. Express.js keretrendszer és Mongoose programozási könyvtár



3.4. ábra. Express bemutatása

A szerveroldal áttekinthetőbb és olvashatóbb kódírása érdekében a webáruház fejlesztése során az Express.js 4.17.1-es verzióját használtam. Az Express egy webes keretrendszer, amely a Node.js nehézség nélküli használatára lett fejlesztve. A 3.4-es ábrán látható az Express attribútumainak ismertetése. Az Express egy Middleware-típusú rendszer, következésképpen lehetővé teszi a MongoDB adatbázisszoftverhez való zavartalan kapcsolódást a Mongoose[**mongoose**] 6.0.12 verziójával kiegészítve, ami egy JavaScript-ben írt objektumorientált programozási könyvtár.

### 3.4.3. MongoDB adatbázisszerver

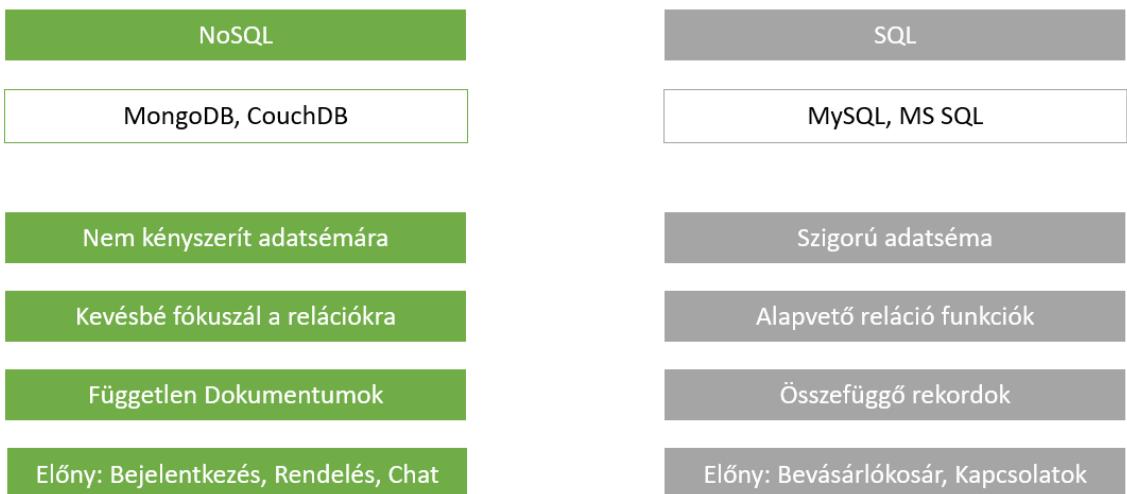


3.5. ábra. MongoDB bemutatása

A MongoDB egy nyílt forráskódú, NoSQL[**nosql**] adatbázisszerverek közé sorolt szoftver a 3.5-ös ábrán látottak szerint. A nevéről következtetve ez nem egy SQL típusú adatbázisrendszer. Jellemzően nem rekordokat és táblázatokat tárolnak, mint az SQL típusú szerverek, hanem független dokumentumokat és gyűjteményeket archiválnak. A NoSQL típusú adatbázisok többségében JSON[**json**] formátumú adatok tárolására alkalmasak. A JSON(JavaScript Object Notation) nyelvfüggetlen kapcsolatot biztosító, szöveges alapú szabványt alkalmazó programozási nyelv. Következésképpen a request és response folyamatok egyszerűsített és gyors működését képes biztosítani, mindeközben lehetővé teszi a kliensoldalon megjelenítendő információk könnyebb feldolgozását.

### 3.4.4. NoSQL és SQL adatbázisok összehasonlítása

A 3.6-os ábrán szándékozom röviden bemutatni és összehasonlítani a nem SQL és az SQL alapú adatbázisokat jellegzetes tulajdonságai szerint.

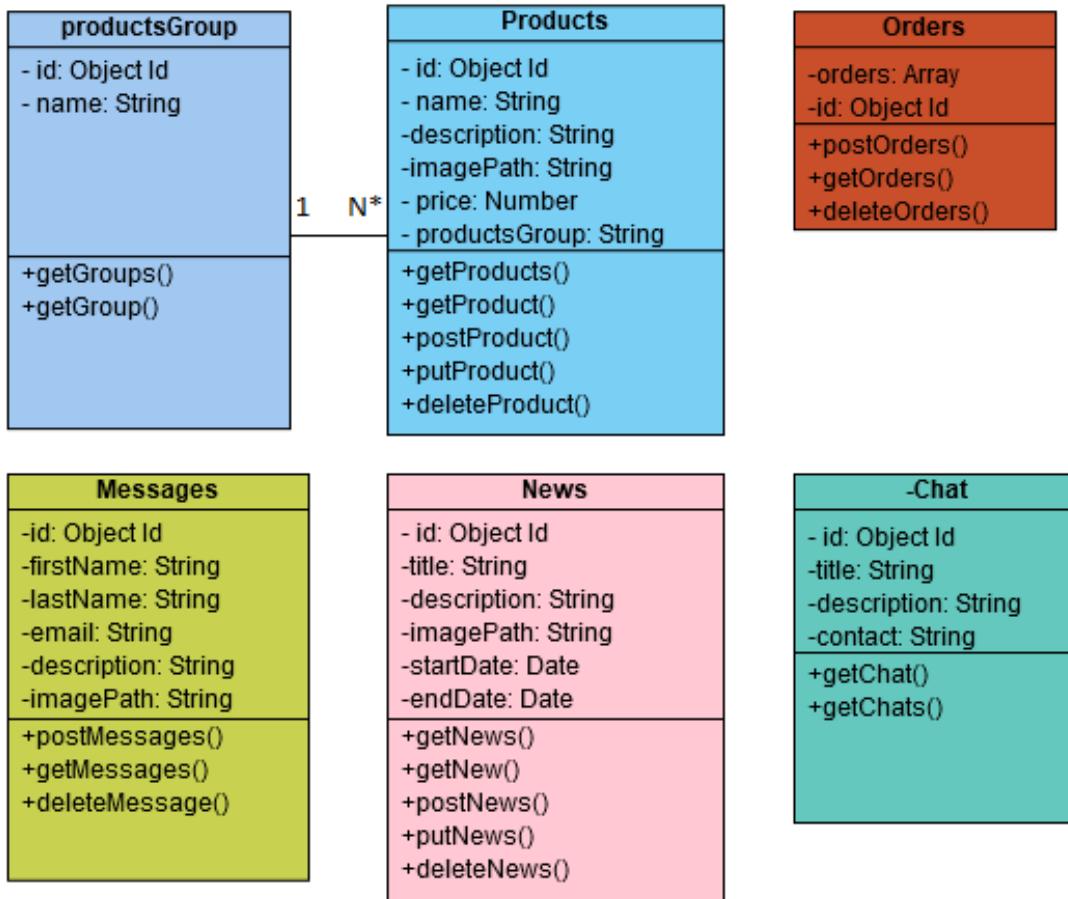


3.6. ábra. NoSQL és SQL adatbázisok összehasonlítása

A 3.6-os ábrán megfigyelhető szempontok szerint egy webáruházban kezelt adatok tárolására a NoSQL adatbázisok kifejezetten alkalmasak. A NoSQL adatbázis-szerverek jellemző tulajdonságai kulcsfontosságú szempontokkal szolgáltak a webáruház adatbázisának kiválasztásánál.

#### 3.4.5. Alkalmazás adatbázisának bemutatása

Az alábbi 3.7-es ábrán látható az alkalmazásban használt adatbázis tervezete. Az illusztrációt osztály diagramként készítettem el. A NoSQL adatbázis típusnak köszönhetően az osztályoknak nem feltétlen kell kapcsolatot kialakítaniuk egymás között. Az adatbázisban hat darab gyűjtemény található. A hat gyűjtemény közül csak kettő áll kapcsolatban egymással, mégpedig a termékek lista és a termékek csoportja. A kapcsolat egy a sokhoz típusú, mivel egy csoporthoz több termék kapcsolódhat, de egy termékhez csak egy csoport.



3.7. ábra. Adatbázis - class diagram

### 3.5. Szerveroldal működése

Ebben a fejezetben részletesen prezentálom az alkalmazás szerveroldali működését, például milyen request hívások találhatóak a kódban, hogyan létesít kapcsolatot a webalkalmazás az adatbázissal, stb., valamint külön kitérek az adminisztrációs oldalon található autentikációra is.

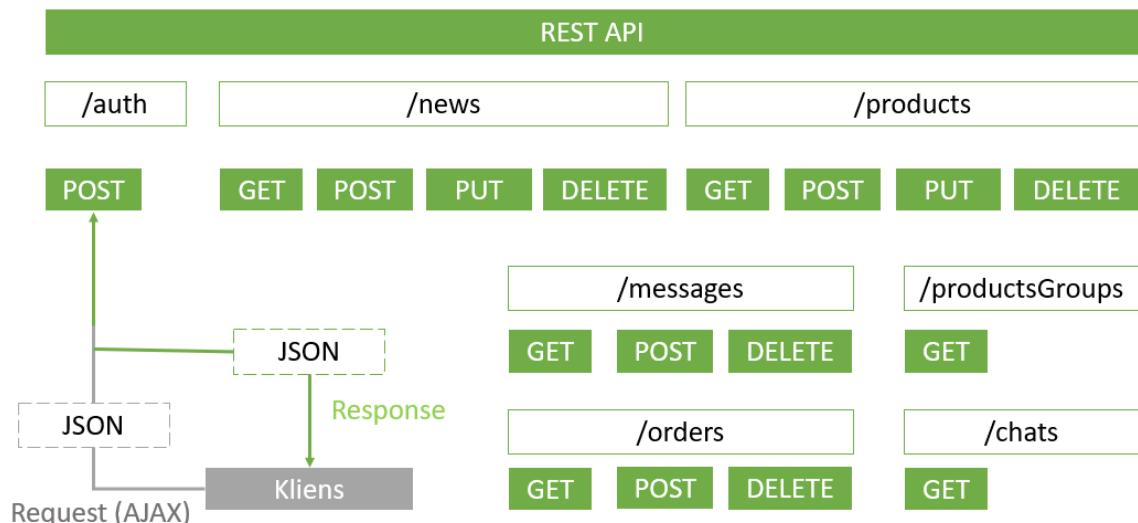
#### 3.5.1. REST API - Végpont tervezés bemutatása

A webáruház megírása során REST API-t (Representational State Transfer Application Programming Interface)[[restapi](#)], architekturális módszert használtam. A REST API-k komponensek összekapcsolására alkalmazzák, könnyű és rugalmas szolgáltatást biztosítva. Az API olyan szabályrendszert jelent, ami meghatározza az alkalmazások számára ilyen módon kapcsolódjanak és kommunikáljanak egymással.

Lehetővé teszi a kliensoldalon található szolgáltatások számára, hogy hozzáférjen a szerveroldali erőforrásokhoz. Az alkalmazás tervezési fázisában ennek a két oldalnak függetlennek kell lennie egymástól. A kliensoldal számára egyetlen információnak a szerveroldal eléréséhez szükséges URL címének kell elérhetőnek lennie. Ugyan úgy, ahogy a szerveroldalnak sem szabad módosításokat végeznie a kliensoldalon, azon kívül, hogy az adatokat HTTP-n keresztül továbbítják egymásnak. Az adatok továbbítására az Angular által biztosított HttpClient[[http](#)] szolgáltatási osztályt használja az alkalmazás. Ennek az osztálynak a segítségével fogja el a kimenő kérések és bejövő válaszokat az alkalmazás számára. A kliensoldal számos kérést képes elindítani, ezek közül az alkalmazásban használt kérések a következők:

- GET: lekérdezi az adatbázisban szereplő listákat
- POST: megkéri az adatbázist, hogy adjon hozzá a listához az elküldött adatokat
- PUT: megkéri az adatbázist, hogy módosítsa a listában szereplő adatot az elküldött adatokra
- DELETE: megkéri az adatbázist, hogy az elküldött tulajdonságúval rendelkező adat kerüljön törlesre

Az alábbi illusztráció szeretném bemutatni az alkalmazásban használt REST API hívásokat, ennek okán a 3.8-as ábrán láthatók a programban megírt végpontok.

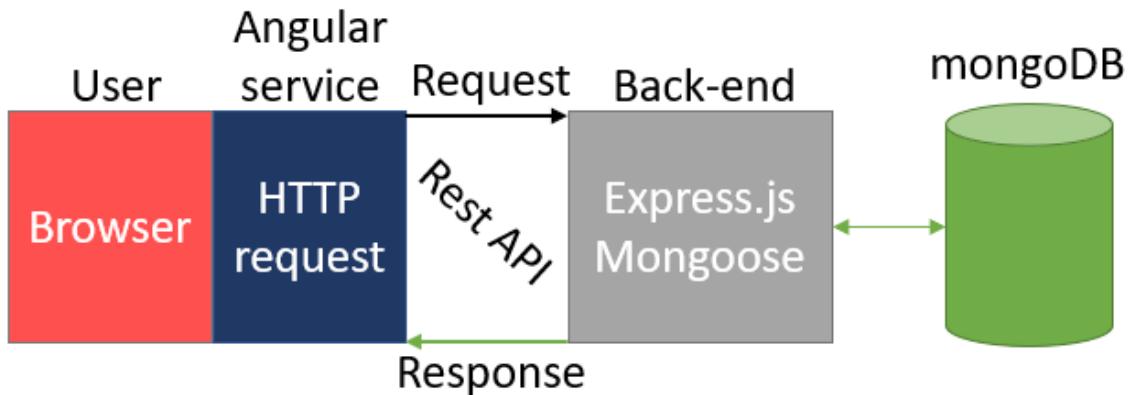


3.8. ábra. Adatkezelés

A grafikán megfigyelhető hét különböző végpont, amelyek az autentikáció, hírek, termékek, termékcsoportok, üzenetek, chatek és rendelések funkciókhöz kapcsolódnak. Az illusztrációt figyelemmel kísérve látható, hogy nem minden végpont ren-

delkezik ugyanazokkal a kérésekkel. Szemléltetésképp vegyük figyelembe a hírekhez vonatkozó kéréseket, amik a GET, POST, PATCH és DELETE függvények, ezzel szemben a autentikációhoz kizárolag POST kérés tartozik. Ennek kifejezetten egy oka van, mégpedig az, hogy a webáruház egyes adatait nem szükséges módosítani tudni kliensoldalról.

A REST API-t megismerve és ezt az információt felhasználva a 3.9-es ábrán látható, miképpen éri el a felhasználó által kezdeményezett kérés az adatbázist és hogyan kap választ.



3.9. ábra. Kapcsolat a szerverrel

A 3.9-es ábrával és az alább található alkalmazásban szereplő kódsorok segítségével szeretném bemutatni, hogy a felhasználó által indított kérés milyen sorrendben jut el az adatbázishoz és miképpen tér vissza a felhasználóhoz.

A felhasználó interakcióba lép a felülettel. Ilyen interakció lehet például a 3.1-es forráskódban szereplő gomb megnyomása. A gombra kattintás következményeként meghívásra kerül a hozzá tartozó TypeScript programozási nyelv segítségével megírt a 3.2-es forráskódban szerepelő függvény. Ez a függvény meghívja egy szolgáltatási fájl metódusát és adatokat ad át a számára.

```

1   <form style="margin-top: 2rem;" [formGroup]="form" (submit)=""
2     onAddMessage()" *ngIf="!isLoading">
3     ...
4     <button class="btn-secondary" type="submit" [disabled]="!
5       confirmed.checked">
6       {{'GENERIC.ACTION.SEND' | translate}}
7     </button>
  
```

3.1. forráskód. Felhasználói interakció - HTML fájl

```

1  onAddMessage(): void {
2      if(this.form.invalid) {
3          this.alertService.warn('ALERT.WARN.INVALID_FORM');
4          this.form.markAllAsTouched();
5          this.isLoading = false;
6          return;
7      }
8      this.isLoading = true;
9      this.contactService.sendMessage(
10         this.form.value.firstName,
11         this.form.value.lastName,
12         this.form.value.email,
13         this.form.value.description,
14         this.form.value.image,
15     )
16     this.isLoading = false
17     this.form.reset();
18 }
```

### 3.2. forráskód. Függvényhívás - TypeScript fájl

Az előbb említett 3.2-es forráskódban szereplő függvény átadja az adatokat a kliensoldalon található Service fájl `sendMessage()` nevezetű függvényének. Ez a fájl tartalmazza a 3.3-as forráskódban szereplő kódsort.

```

1  sendMessage(
2      firstName: string,
3      lastName: string,
4      email: string,
5      description: string,
6      image: File | string
7  ){
8      const messagesData = new FormData();
9      messagesData.append("firstName", firstName);
10     messagesData.append("lastName", lastName);
11     messagesData.append("email", email);
12     messagesData.append("description", description);
13     messagesData.append("image", image, firstName);
14     this.http.post<{message: string, messages: Messages}>
15       (environment.apiUrl + "messages", messagesData)
16       .subscribe(responseData => {
```

```

17     const messages: Messages = {
18         id: responseData.messages.id,
19         firstName: firstName,
20         lastName: lastName,
21         email: email,
22         description: description,
23         imagePath: responseData.messages.imagePath,
24     };
25     this.messages.push(messages);
26     this.msgUpdate.next([...this.messages]);
27     this.alert.success('ALERT.SUCCESS.ADD');
28     this.router.navigate(['/contact']);
29 }, error => {
30     this.alert.error(error.error.message);
31 });
32 }
```

### 3.3. forráskód. HttpClient POST request - Service TypeScript fájl

Ebben a kódrészletben látható, ahogyan a kliens oldal HttpClient szolgáltatási osztály POST kérés segítségével a megadott útvonalon küld egy REST API hívást a szerveroldal felé.

A szerveroldal fogadja ezt a kérést és továbbítja a MongoDB felé. A 3.4-es forráskódban Express - JavaScript nyelv segítségével megírt kódrészletben ez szerepel.

```

1 exports.postMessage = (req, res, next) => {
2     const url = req.protocol + "://" + req.get("host");
3     const msg = new Messages({
4         firstName: req.body.firstName,
5         lastName: req.body.lastName,
6         email: req.body.email,
7         description: req.body.description,
8         imagePath: url + "/images/messages/" + req.file.filename,
9     });
10    msg.save().then(result => {
11        res.status(201).json({
12            message: "Message added successfully",
13            messages: {
14                ...result,
15                id: result._id
16            }
17        })
18    })
19 }
20 }
```

```
17      }) ;  
18  }) ;  
19 }
```

#### 3.4. forráskód. Express POST végpont - JavaScript

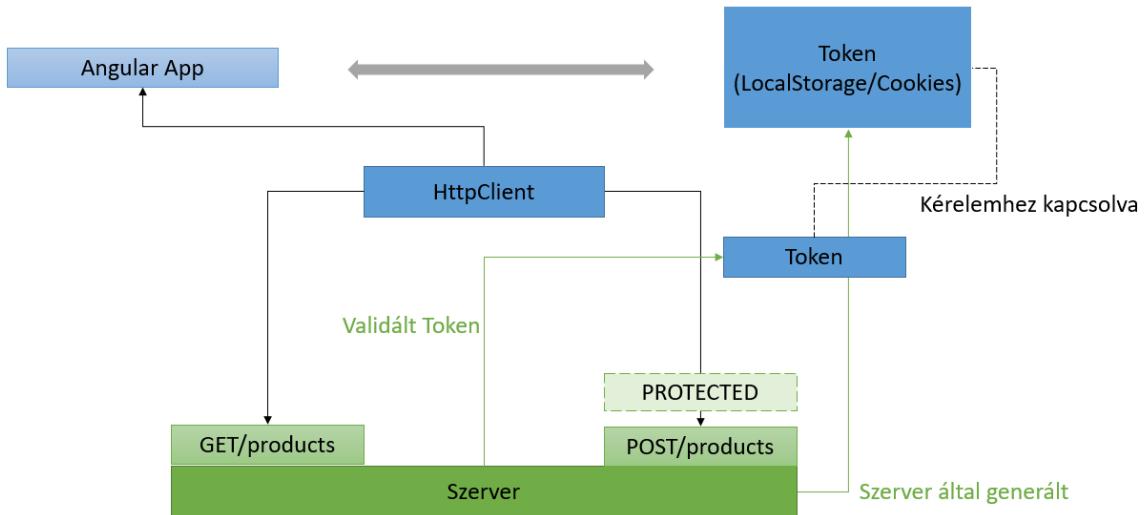
Kliensoldalon és szerveroldalon egyaránt látható a felhasználó által elindított kérés státuszát tartalmazó információ. A front-enden egy úgynevezett AlertService komponens segítségével kap visszaigazolást az elindított kérés befejezéséről. A backend oldalon a `res.status(201)` kódrészlet segítségével kapunk visszaigazolást a kérés befejezésével kapcsolatban.

A webalkalmazásban szereplő további REST API hívásokat tartalmazó példakódsorok a 3.7-es fejezet Szerveroldali forráskódok (3.7.2) nevezetű alfejezetében találhatóak.

### 3.5.2. Hitelesítés - Az adminisztrációs felület védelme

Az alkalmazás megírása során szándékosan egy olyan webáruház létrehozása volt a célom, ami időszerű adatok kezelését tudja biztosítani. Következésképpen egy olyan felület megvalósítása, amely nem igényel az oldal üzemeltetéséhez programozói segítséget. Ennek okán a program tartalmaz egy adminisztrációs oldalt, amely biztosítja a webalkalmazásban dinamikusan megjelenő adatok aktualitását, továbbá a leadott rendelések és vásárlók által elküldött üzenetek megjelenítésére is szolgál. Kifejezetten ennek a blokknak védelmére került bele külön hitelesítési[hitelesites] felület.

A 3.10-es grafikán található az engedélyezési folyamat elméleti működése.



3.10. ábra. Hitelesítés

Az illusztrációt ábrázolt hitelesítési folyamat a következőképpen történik. Az alkalmazás front-endről érkező GET/products kérésre engedélyezés nélkül kap választ a szervertől, mivel a webáruházból bejelentkezés hiányában is hozzáférhetővé kell tenni a termékek listáját. Ezzel szemben a POST/products kérés nem következhet be hitelesítés nélkül. Tehát bejelentkezés során a szerver generál egy úgynevezett tokenet és elmenti a LocalStorage-ba, amit az új termék hozzáadása kérésnél ellenőriz. Az alkalmazásban ez a hitelesítési funkció a következőképpen valósul meg:

Ennek a funkciónak a létrehozásánál két kiegészítő package-et, *bcryptjs*-t és *jsonwebtoken*-t használ a programkód. Az előbbi lehetőséget nyújt bizonyos adatok titkosítására (ilyen adat például a belépéshez szükséges jelszó), az utóbbi pedig bizonyos REST API kérések érvényesítésére alkalmas. Mikor létrehozásra kerül egy új felhasználó, a bcrypt package egy úgynevezett hasító metódusát hívja meg a program, aminek segítségével az új felhasználó jelszava titkosítva kerül be az adatbázisba. Ennek az az oka, hogy idegenek számára olvashatatlanná váljon ez az információ. Viszont ezzel probléma adódik a későbbiekben. Mivel, ha titkosítva kerül be az adatbázisba, az oldalra történő belépéskor nem elegendő összehasonlítani a begépelt adatot az adatbázisban tárolt adattal. Erre a problémára szolgál megoldásképp a *bcryptjs compare* metódusának használata. Ennek a metódusnak segítségével összehasonlításra kerül az adott felhasználó adatbázisban szereplő titkosított jelszava és a begépelt jelszó hasított változata. Ennek köszönhetően, ha pont a megfelelő karaktereket gépeljük be, akkor a begépelt adat titkosított változatának megegyezőnek kell lennie az adatbázisban található adattal.

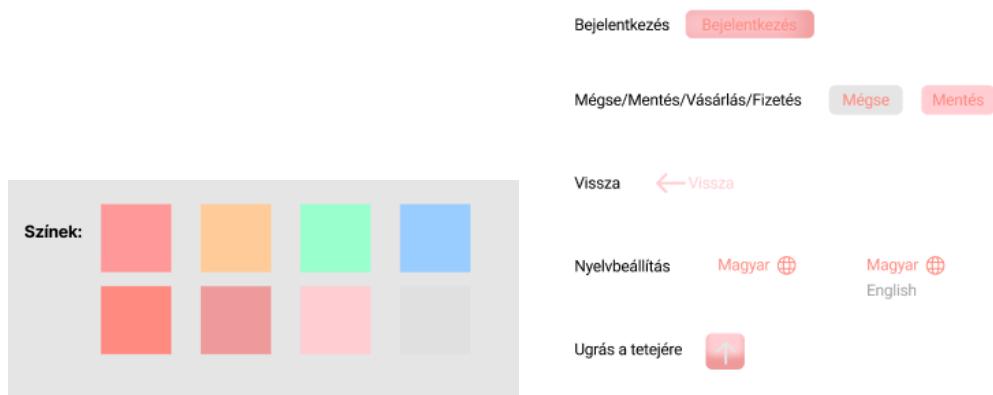
Mindent összevetve az adminisztrációs felület védve van az illetéktelen felhasználók belépésétől és bizonyos funkciók végrehajtásától.

## 3.6. Az alkalmazás megjelenése

Az oldal megjelenéséhez kapcsolatos színvilág, és vizuális elemek nagy részben a saját munkáim. Az összes látványelem, mint például a logó vagy az oldalon található képek, az Adobe Photoshop[photoshop] 2019-es programjával készültek. Az alkalmazásban szereplő ikonok a Google Icons - Material icons[material] segítségével kerültek megjelenítésre.

### 3.6.1. Figma szoftver - oldalvázlatok

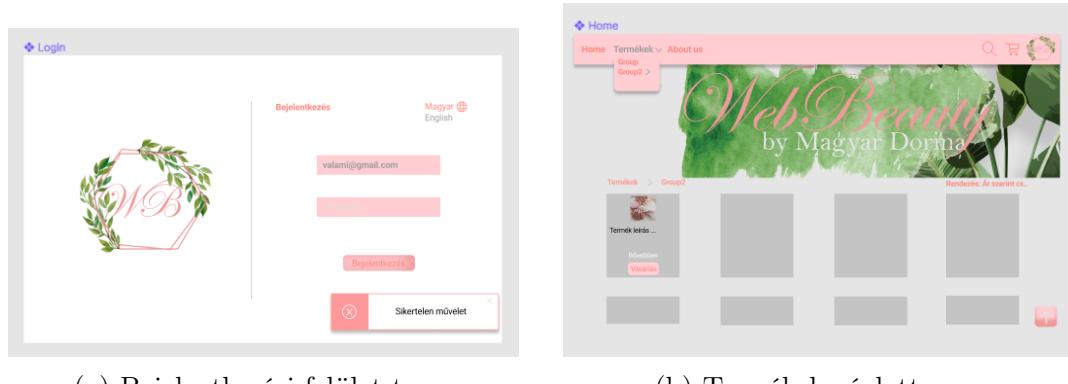
Az alkalmazás fejlesztése előtt a megjelenő oldalakra tervezett drótvázterveket a Figma[figma] nevezetű vektorgrafikus szerkesztő segítségével készítettem. A tervezés során meghatároztam, és összegyűjtöttem az alkalmazás színvilágát, ami a 3.11-es ábra bal oldalán látható. Ezek a színek globálisan kerültek implementálásra a kódon belül. Ez azt jelenti, hogy ha egyes színek lecserélésre kerülnek, akkor az alkalmazáson belül az adott színek az újonnan megadott színekként jelennek meg. Ezen felül olyan elemek kerültek megtervezésre, mint például az értesítő üzenetek vagy a 3.11-es ábra jobb oldalán látható gombok.



3.11. ábra. Figma kiegészítő elemek tervezete

Az alkalmazásban megjelenő oldalak vázlatát is elkészítettem, ami a 3.12-es ábrán látható. Ha az ábrán szereplő két képet összehasonlítjuk az alkalmazásban meg-

jelenő oldalakkal, akkor észrevehető mekkora különbség van az oldalvázlatok és az alkalmazásban szereplő végleges megjelenéssel rendelkező oldalak között.



(a) Bejelentkezési felület terv

(b) Termékek vázlatterve

3.12. ábra. Figma kiegészítő elemek tervezete

### 3.6.2. Logók és grafikai elemek

Az alkalmazás grafikai elemeinek elkészítése során próbáltam figyelembe venni a program letisztult megjelenését és természetesen annak színvilágát. A webáruházban végül két különböző színű logó kerül felhasználásra. A 3.13-as ábra bal oldalán sötétebb kerettel és szöveggel rendelkező logó látható, amit a jobb oldali képpel megegyező színű háttereken használok. Ilyen háttérrel rendelkezik például az oldal menüsora. A jobb oldali logót fehér háttereken használom, mint például az adminisztrációs rész bejelentkezési felülete.



(a) Sötétebb logó

(b) Logó

3.13. ábra. Grafikai elemek - logók

A webshopon megjelenő további grafikai elemek, például a 3.14-es ábrán található képek célja, hogy változatosabb megjelenést kölcsönözzenek az alkalmazás számára.



(a) Rólunk felület fejléce

(b) Főoldal felület fejléce

3.14. ábra. Grafikai elemek

## 3.7. Forráskódok

### 3.7.1. Kliensoldali forráskódok

Ebben az alfejezetben található a 3.3-as fejezetben említett chatbot bemutatása forráskódok segítségével. Ez a funkció úgy került megvalósításra, hogy előre megírt és az adatbázisban eltárolt kérdéseket és válaszokat jelenítek meg animációk segítségével. Az alábbi forráskódrészletek ennek a megjelenítését írják le.

Először is lekérdezem és eltárolom ezt a chat listát a GET/chat 3.5 forráskódban szereplő függvény segítségével.

```

1  async ngOnInit() {
2      await this.chat.getChat();
3      this.chatSub = this.chat.getUpdateListener()
4          .subscribe(chat => {
5              this.chatList = chat;
6          });
7      }

```

3.5. forráskód. GET/chat lista lekérés - TypeScript

Ezután megjelenítem a listában szereplő kérdéseket HTML fájlból a 3.6 forráskódban látottak szerint. Ezek a kérdések linkként szolgálnak, amik átirányítanak a kérdés animált profil oldalára, ami a 3.7 forráskódban látható.

```

1  <div class="chat-body">
2      <div class="chat-link" *ngFor="let chat of chatList">
3          <div (click)="loadChatProfile(chat.id)">{{chat.title}}</div>

```

```
4   </div>
```

### 3.6. forráskód. Chatbot megjelenítése - HTML

```
1  <div class="animated display-message">
2    <div class="chat__message chat__message_B" style="--delay: 2s">
3      <div class="chat__content">
4        {{selectedChat.title}}
5      </div>
6    </div>
7    <div class="chat__message chat__message_A" style="--delay: 6s">
8      <div class="chat__content">
9        {{selectedChat.description}}
10     </div>
11   </div>
12   ...
13
14
15
16
17
18
19
20
21
```

### 3.7. forráskód. Chatbot kérdéshez tatoró szöveg megjelenítése - HTML

A beszélgetés imitálásához SCSS-ben írt stílusfájlt használtam a 3.8 forráskódban leírtak alapján.

```
1 .chat__message {
2   ...
3   transform-origin: 0 100%;
4   padding-top: 0;
5   transform: scale(0);
6   ...
7   animation: message 0.15s ease-out 0s forwards;
8   animation-delay: var(--delay);
9   --bgcolor: var(--info-color);
10  --radius: 8px 8px 8px 0;
11 }
12
13 .chat__message_B{
14   color: var(--light-color);
15   flex-direction: row-reverse;
16   text-align: right;
17   align-self: flex-end;
18   transform-origin: 100% 100%;
19   --bgcolor: var(--main-color);
20   --radius: 8px 8px 0 8px;
21 }
```

```

22
23     .chat__message::before {
24         content: "";
25         flex: 0 0 40px;
26         aspect-ratio: 1/1;
27         background: var(--bgcolor);
28         border-radius: 50%;
29     }

```

3.8. forráskód. Chatbot beszélgetés animáció - SCSS

### 3.7.2. Szerveroldali forráskódok

Az alább található forráskódok a 3.5.1-es fejezetben említett JavaScript programozási nyelv segítségével írt példa lekérések láthatóak. A programban használt végpontok, amik bemutatásra kerülnek ebben a fejezetben a következő kérések: GET, DELETE és PUT műveletek.

GET/news 3.9-es forráskódban látható a GET végpont lekérése, aminek segítségével lekérésre kerül az adatbázisból a hírek listája. Az adatbázis válaszüzenetben visszaküld egy JSON fájlt. Ez a JSON fájl a 3.10-es forráskód leírásában látható.

```

1   exports.getChat = (req, res, next) => {
2       Chat.find().then(result => {
3           res.status(200).json({
4               message: "Chat fetched successfully",
5               chat: result,
6           });
7       });
8   }

```

3.9. forráskód. GET/news végpont

```

1   {
2       "message": "News fetched successfully!",
3       "news": [
4           {
5               "_id": "61b61edfb415f9f7cc07e7a8",
6               "title": "Oldal letrehozasa",
7               "description": "Az oldal letrehozasa Magyar Dorina
8                   Szakdolgozata elkeszitese celjabol tortent. Jelenleg ez az
9                   oldal nem uzemel! Nem kerulnek ertekesitesre azok a

```

```

        termekek, amik az aruhazban talalhatoak! Ha további
        kerdeze lenne a Rolunk feliratu menünek keresztül
        kapcsolatba lephet velem és minden kérdésre email
        formájában válaszolok. Megérteseteket előre is köszönöm!",

8      "imagePath": "http://localhost:3000/images/news/oldal-
         letrehozasa-1639325407045.png",
9      "startDate": "2021-08-31T22:00:00.000Z",
10     "endDate": "2021-12-14T23:00:00.000Z",
11     "__v": 0
12   },
13   ...
14 }

```

### 3.10. forráskód. GET/news JSON

PUT/news 3.11-es forráskódban szereplő módosítás művelet végpontja látható. Ennek a végpont segítségével képesek vagyunk módosítani az adatbázisban szereplő lista elemeit. Az adatbázis válaszüzenetben visszaküldi a művelet státuszát. Ha sikeresként tér vissza akkor a 3.12-es forráskódban szereplő JSON objektumot kapjuk meg.

```

1  exports.putNews = (req, res, next) => {
2    let imagePath = req.body.imagePath;
3    if(req.file) {
4      const url = req.protocol + "://" + req.get("host");
5      imagePath = url + "/images/news/" + req.file.filename
6    }
7    const news = new News({
8      _id: req.body.id,
9      title: req.body.title,
10     description: req.body.description,
11     imagePath: imagePath,
12     startDate: req.body.startDate,
13     endDate: req.body.endDate,
14   });
15   News.updateOne({_id: req.params.id}, news).then(result => {
16     res.status(200).json(
17       {message: "Update successful!"}
18     );
19   });
20 }

```

---

### 3.11. forráskód. PUT/news végpont

```
1  {
2      "message": "Update successful!",
3      "news": [
4          {
5              "_id": "61b61edfb415f9f7cc07e7a8",
6              "title": "Oldal letrehozása változás",
7              "description": "Az oldal letrehozása Magyar Dorina  
Szakdolgozata elkeszítése celjából történt. Jelenleg ez az  
oldal nem üzemel! Nem kerülnek értékesítésre azok a  
termékek, amik az aruházban találhatóak! Ha további  
kerdése lenne a Rolunk feliratú menünek keresztül  
kapcsolatba lephet velem és minden kérdésre email  
formajában válaszolok. Megérteseteket előre is köszönöm!",
8              "imagePath": "http://localhost:3000/images/news/oldal-  
letrehozasa-1639325407045.png",
9              "startDate": "2021-08-31T22:00:00.000Z",
10             "endDate": "2021-12-14T23:00:00.000Z",
11             "__v": 0
12         }]
13     }
```

### 3.12. forráskód. PUT/news JSON

DELETE/news 3.13-as forráskódban a DELETE kérés végpontja szerepel. Ezzel a műveettel lehetőségünk van kitörölni az adatbázisból az átadott paraméterrel rendelkező adatokat. Válaszüzenetként csak a kérés státuszát kapjuk vissza.

```
1  exports.deleteNews = (req, res, next) => {
2      News.deleteOne({_id: req.params.id}).then( result => {
3          res.status(200).json({
4              message: "News deleted!"
5          });
6      });
7  }
```

### 3.13. forráskód. DELETE/news végpont

## 3.8. Tesztesetek

### 3.8.1. Kliensoldal tesztelése

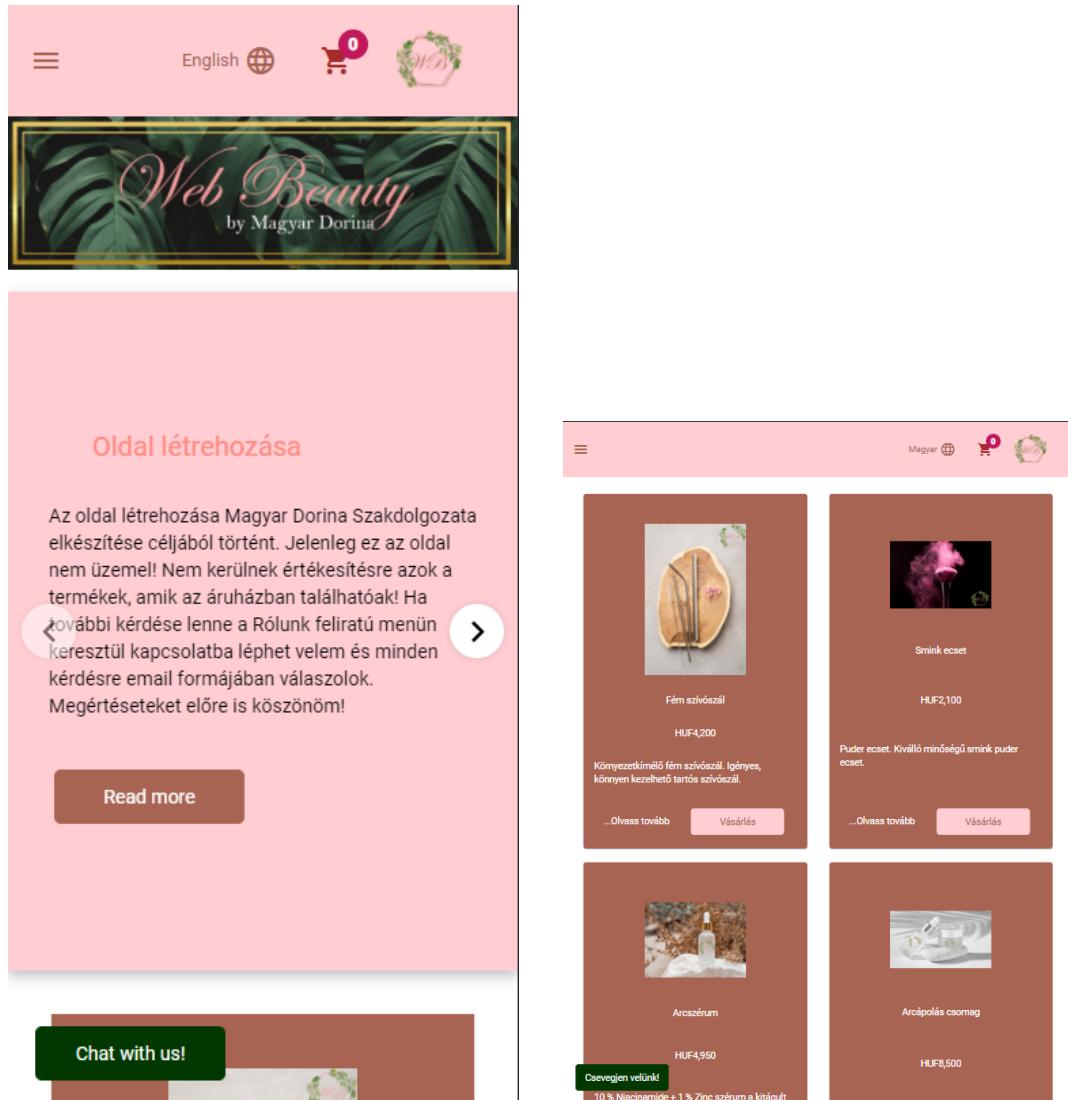
A kliensoldal tesztelésére általam végrehajtott felülettesztést végeztem. A felhasználói felülettesztést gyakran használják webes alkalmazások tesztelésére. Ennek a módszernek a lényege, hogy a felületet állandó stressznek tesszük ki.

1. A bejelentkezési felület tesztelése: a 3.15-ös ábra bal oldalán látható képen helyesen megadott emailcím és jelszó párossal próbálok belépni, ami sikeres üzenettel tér vissza. A sikeres teszt eredményeként átnavigál az adminisztrációs felület termékek listája oldalra. Az ábra jobb oldalán helytelen adatok megadásával sikertelen művelet üzenettel tér vissza az oldal. A sikertelen teszt eredményeként a bejelentkezési kérelem elutasításra kerül és ezen a felületen marad.



3.15. ábra. Bejelentkezési felület tesztelése

2. Reszponzivitás: az oldal telefonon, táblagépen és asztaligépen is megfelelően tölti be az oldalt. A megjelenése eszközspecifikusan reszponzív. A 3.16-os ábra bal oldala a telefonos nézetet jeleníti meg, míg az ábra jobb oldala a táblagépről látott képet mutatja be. Mindkettő kép esetében láthatjuk, hogy a menüsor elrejtésre kerül. Ezzel biztosítva azt, hogy nem csúszik össze az ikonokkal. Az ábrán továbbá megfigyelhető, hogy a hírek és a termékek listája módosítva jelenik meg. Mindkét eszköz feliületén a termékek kártyái láthatóan újra rendeződnek. Továbbá mobilnézetben a főoldalon megjelenő hírek és az azokhoz tartozó képek elrejtésre kerülnek.



3.16. ábra. Reszponzív felület tesztelése

### 3.8.2. Szerveroldal tesztelése

Az alábbi fejezetben látható a szerveroldalon megírt REST API kérések tesztelése látható táblázat formájában. A végpontok tesztelését Postman[**postman**] program segítségével készítettem. A 3.1 táblázat első oszlopában látható a kérésekre vonatkozó hitelesítési kötelezettségek. A második oszlopban a kérések URL címe olvasható, ezekkel a címekkel érhetőek el a back-enden megírt függvények. A harmadik és negyedik oszlopban ezeknek a kéréseknek a végrehajtás státuszáról látható válaszüzenetek. Ilyen információ például az, hogy sikeres volt-e a művelet, és milyen üzenet társul hozzá.

Szerveroldali végpontok tesztelése			
GET			
Token	Request URL	Status	Message
nem	api/products	200 OK	Products fetched successfully
nem	api/productsGroups	200 OK	Group fetched successfully
nem	api/news	200 OK	News fetched successfully
nem	api/chat	200 OK	Chat fetched successfully
nem	api/messages	401 Unauthorized	Auth failed
igen	api/messages	200 OK	Message fetches successfully
igen	api/orders	200 OK	Order fetches successfully
POST			
Token	Request URL	Status	Message
igen	api/products	500 Server Error	Unexpected field
igen	api/products	200 OK	Products added successfully
igen	api/news	200 OK	News added successfully
nem	api/messages	200 OK	Message added successfully
nem	api/orders	200 OK	Products added successfully
PUT			
Token	Request URL	Status	Message
igen	api/products	500 Server Error	Unexpected field
igen	api/products	200 OK	Update successful
igen	api/news	200 OK	Update successful
DELETE			
Token	Request URL	Status	Message
igen	api/products/6186..	200 OK	Products deleted
igen	api/news/61ae..	200 OK	News deleted
igen	api/messages/61a4..	200 OK	Message deleted
igen	api/orders/6197..	200 OK	Order deleted

3.1. táblázat. Back-end REST API kérés végpontok tesztelése.

### 3.9. Továbbfejlesztési lehetőségek

Az alkalmazás továbbfejlesztés céljaként szeretném a webáruház felületén is megjeleníteni az autentikációt. Ezzel lehetővé téve, hogy a vásárlók létrehozzák a saját fiókjukat, aminek a segítségével láthatják a vásárlási előzményeiket. Bejelentkezés után a vásárlók, egy ajánlott terméklistát láthatnának az előzményeik alapján leszűrve, továbbá mások által együtt vásárolt termékek intelligens összekötése és ajánlása további vásárlók számára. A program mostani verziójába azért nem került be ez a funkció a webshop felületére, mert jelenleg nem tudnám különbözőket az adminisztrációs és a webáruház felület bejelentkezését. A későbbiekben erre a problémára megoldásként szolgálhat a szerepkörök bevezetése.

## 4. fejezet

### Összegzés

A program sikeresen bemutat egy működő webáruházat, amit kisvállalkozók saját termékeik értékesítésére alkalmas. Az alkalmazást első sorban egy vállalkozó általi megkeresés céljából készítettem el. De a fejlesztés során számos új technológiákat és új módszereket sikerült megismernem, amiket az egyetemi tanulmányaim során nem találkoztam. Az alkalmazás megvalósítására használt MEAN szoftverköteg nem egyedi, hiszen rengeteg projektnek az alapjaként szolgál, még is a kliensoldalon használt kiegészítő csomagjainak segítségével egy hangulatos elsősorban kozmetikai eszközök értékesítését szolgáló webáruházat sikerült lefejlesztenem. Viszont a későbbiekben az alkalmazás képes kiszolgálni más megjelenést és tartalmi igényeket kérő projekteket, mivel ezek az adatok dinamikusan vannak kezelve. Az adminisztrációs rész megkönnyíti a vállalkozó számára a felület karbantartását, hiszen felhasználóbarát módosítási lehetőségek kerültek implementálásra. A használt technológiák megfelelő kompatibilitással rendelkeznek egymással szemben. Ennek köszönhetően nem ütköztem nagyobb akadályokba a fejlesztés során.

Külön megszeretném köszönni Nemesné Balla Júlia Évának, hogy lehetőséget kaptam az alkalmazás elkészítésére. Továbbá szeretném megköszönni Fekete Anett PhD hallgatónak, hogy elvállalta a szakdolgozatom témavezetését és végig támogatta a program elkészítését.

# Ábrák jegyzéke

2.1.	Use case diagram - Alkalmazás funkciói . . . . .	7
2.2.	Az alkalmazás megjelenése . . . . .	8
2.3.	Webáruház tartalmi része . . . . .	8
2.4.	Az oldal lábrésze . . . . .	9
2.5.	Alert üzenet . . . . .	9
2.6.	Bejelentkezési oldal . . . . .	10
2.7.	Adminisztrációs felület megjelenése . . . . .	11
2.8.	Rendelési folyamat bemutatása - szekvencia diagram . . . . .	12
2.9.	Termék vásárlása a termékek listáján keresztül . . . . .	13
2.10.	Termék vásárlása a termék profilján keresztül . . . . .	14
2.11.	Bevásárlókosár . . . . .	15
2.12.	Fizetési felület . . . . .	16
2.13.	Chat funkció bemutatása . . . . .	17
2.14.	ÁSZF letöltése . . . . .	18
3.1.	Az alkalmazás rétegeinek bemutatása . . . . .	20
3.2.	Az Angular keretrendszer bemutatása . . . . .	21
3.3.	Node.js bemutatása . . . . .	25
3.4.	Express bemutatása . . . . .	26
3.5.	MongoDB bemutatása . . . . .	27
3.6.	NoSQL és SQL adatbázisok összehasonlítása . . . . .	28
3.7.	Adatbázis - class diagram . . . . .	29
3.8.	Adatkezelés . . . . .	30
3.9.	Kapcsolat a szerverrel . . . . .	31
3.10.	Hitelesítés . . . . .	35
3.11.	Figma kiegészítő elemek tervezete . . . . .	36
3.12.	Figma kiegészítő elemek tervezete . . . . .	37
3.13.	Grafikai elemek - logók . . . . .	37

3.14. Grafikai elemek . . . . .	38
3.15. Bejelentkezési felület tesztelése . . . . .	43
3.16. Reszponzív felület tesztelése . . . . .	44

# Forráskódjegyzék

3.1.	Felhasználói interakció - HTML fájl . . . . .	31
3.2.	Függvényhívás - TypeScript fájl . . . . .	32
3.3.	HttpClient POST request - Service TypeScript fájl . . . . .	32
3.4.	Express POST végpont - JavaScript . . . . .	33
3.5.	GET/chat lista lekérés - TypeScript . . . . .	38
3.6.	Chatbot megjelenítése - HTML . . . . .	38
3.7.	Chatbot kérdéshez török szöveg megjelenítése - HTML . . . . .	39
3.8.	Chatbot beszélgetés animáció - SCSS . . . . .	39
3.9.	GET/news végpont . . . . .	40
3.10.	GET/news JSON . . . . .	40
3.11.	PUT/news végpont . . . . .	41
3.12.	PUT/news JSON . . . . .	42
3.13.	DELETE/news végpont . . . . .	42