



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

Az eredeti szakdolgozati / diplomamunka témabejelentő helye.

Tartalomjegyzék

1. Bevezetés	3
2. Felhasználói dokumentáció	4
2.1. Alkalmazás indítása	4
2.1.1. Alkalmazás indítás böngészőből	4
2.1.2. Alkalmazás indítása saját gépről	5
2.2. Alkalmazás kezelése	6
2.2.1. Webshop felület kezelése	6
2.2.2. Adminisztrációs felület kezelése	9
2.3. Rendelési folyamat	10
2.3.1. Rendelés leadása	11
2.3.2. Vásárlással kapcsolatos információk	14
3. Fejlesztői dokumentáció	15
3.1. Webalkalmazás specifikáció	15
3.2. Kliensoldalon használt technológiák bemutatása	16
3.2.1. Angular keretrendszer	17
3.3. Kliensoldal működése	18
3.3.1. Komponensek	18
3.3.2. Modulok	18
3.3.3. Interfészek	19
3.3.4. Service fájlok	19
3.3.5. Admin mappa	19
3.3.6. Pages mappa	19
3.3.7. Oldalak közötti navigáció	19
3.3.8. Shared mappa	20
3.3.9. Egyéb fájlok és mappák	20
3.4. Szerveroldalon használt technológiák bemutatása	21

TARTALOMJEGYZÉK

3.4.1. Node.js szoftverrendszer	22
3.4.2. Express.js framework és Mongoose programozási könyvtár . .	23
3.4.3. MongoDB adatbázisszerver	24
3.4.4. NoSQL vs SQL adatbázisok összehasonlítása	24
3.5. Szerveroldal működése	25
3.5.1. RESTful API - Végpont tervezés bemutatása	25
3.5.2. Hitelesítés - Adminisztrációs felület védelme	30
3.6. Az alkalmazás megjelenése, web design	31
3.6.1. Figma szoftver - oldalvázlatok	32
3.6.2. Logók és grafikai elemek	33
3.7. Forráskódok	34
3.7.1. Kliensoldali forráskódok	34
3.7.2. Szerveroldali forráskódok	36
3.8. Tesztesetek	38
3.8.1. Kliensoldal tesztelése	38
3.8.2. Szerveroldal tesztelése	38
3.9. Továbbfejlesztési lehetőségek	39
4. Összegzés	40
Irodalomjegyzék	41
Ábrajegyzék	41
Táblázatjegyzék	42
Forráskódjegyzék	43

1. fejezet

Bevezetés

A **WebBeauty**(továbbiakban **WB**) lehetőséget kínál a kisvállalkozók által gyártott termékek bemutatására és árusítására. Mivel napjainkban menőt a kereslet a személyes webáruházak létrehozása iránt, ahol a saját termékeiket kívánják értékesíteni, ezt pedig a **WB** megfelelően kiszolgálja. Az webalkalmazás nem csak a felhasználók számára könnyen kezelhető, hanem egyben a tulajdonosnak 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 értékesítene az általam készített webáruházon keresztül a termékeit. Az alap problémát az vetette fel részéről, hogy egyénileg nem tudnám kiszolgálni az üzlettulajdonos által érkező folyamatos frissítési kéréseit. Ezt a felmerülő nehézséget úgy próbáltam megoldani, hogy a vállalkozó által is kényelmesen elérhetővé tegyem azokat a funkciókat, ami a webalkalmazás aktualizáltságát biztosítja. Továbbá a program képes a tulajdonos és a vásárló közötti közvetlen kapcsolat létszatát kialakítani a **chatbot**¹ funkció segítségével. Mivel ez egy egyszerűbb webalkalmazás, ezért nem egy teljesen egyénileg gondolkozó mesterséges intelligencia(MI)² alapú chatbotról esik szó, hanem egy adatbázisban tárolt előre generált válaszból álló szöveges adathalmazról beszélhetünk, ami kulcsszavas keresés segítségével zajlik.

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

²sokféle megközelítést találhatunk a definícióját illetően. Személy szerint azt gondolom, hogy az MI egy tudatos gondolkozásra alkalmas, emberi beavatkozás nélküli cselekvőképes létfelvétel, amit a legtöbbször számítástechnikai eszközökhez/gépekhez 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, ami nem feltétlenül egyértelmű 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. 2.1.1. fejezet
2. Megnyitni localhostról a projektet. 2.1.2 fejezet

Mindkettő technikát részletes bemutatásra kerül a következő alfejezetekben.

2.1.1. Alkalmazás indítás böngészőből

Az első és legkönnyebben alkalmazható stratégia, hogy valamilyen előre telepített webböngésző (Chrome, Firefox, Opera..stb.) segítségével megnyitjuk az előre Amazon (AWS) szerverére telepített weboldalt. A felület ezen az URL-en (rövidítés feloldása: Uniform Resource Locator) 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 alkalmazás nem csak le kell klónoznunk azt Github segítségével, hanem több különböző szoftvert kell telepítenünk mielőtt eltudnánk indítani magát a projektet. Magához a program fordításához szükségünk lesz a Node.js szoftverre, amely letölthető ingyenesen a nodejs.org eredeti honlapjáról. Továbbá még elengedhetetlen a gépünkről az Angular CLI. Ez egy parancssori interfész az Angular szoftverhez. Mivel a kód futtatásához szükségünk van rengetek eszközre, amely lefordítja és optimalizálja a kódot. A CLI ezt biztosítja a projekt számára. A kód fordításához szükségük lesz egy integrált fejlesztői környezetre. Én személy szerint a Visual Studio Code(rövidítve VS Code) nyílt forráskódú kód szerkesztőjét használtam az alkalmazás elkészítése során, így ezt fogom bemutatni. A következő felsorolásban összefoglalom azokat a lépéseket, amik segítségével eljutunk a program saját gépről való indítását.

1. <https://nodejs.org>-ról töltsük le és telepítsük a számítógépre megfelelő .exe kiterjesztésű fájlt.
2. <https://github.com/magyardor/szakdolgozat2021> webcímen elérhető GitHub repository-t klónozzuk le az eszközünkre
3. töltsük le, és telepítsük a Visual Studio Code nevezetű programot a <https://code.visualstudio.com> címről.
4. nyissuk meg a VS code alkalmazást és installáljuk a következő bővítményeket: Angular Essentials, Material Icon Theme
5. a VS code segítségével töltsük be a leklónozott projekt mappáját, és nyissuk meg egy új terminált
6. a terminálba lépjünk be a szakdolgozat nevezetű mappájába és futtassuk le a következő parancsot: `npm install -g @angular/cli`
7. ha ez sikeresen megtörtént akkor futtassuk le az `npm i` parancsot, melynek segítségével letöltődnek azok a szükséges fájlok, amik nem szerepelnek az Angular CLI interfészben, de használatban vannak

8. mielőtt elindítanánk az alkalmazást be kell lépnünk a projekthez tartozó MongoDB adatbázisba és hozzá kell adnunk a hálózati hozzáférés nevezetű menüpont alatt a gépünk IP címét, különben nem tud csatlakozni a szerveroldalunk az alkalmazás által megjelenítendő adatokhoz
9. a meglévő terminálunk mellé nyissunk meg egy újat és futtassuk le az egyikbe az npm run start a másikba az npm run start:server parancsokat, az előbbi a kliensoldalt, míg az utóbbi a szerveroldali kódokat futtatja és fordítja le
10. ha sikeresen lefordult a kód, akkor a böngésző URL helyére a localhost:4200 címet begépelve megkapjuk a webáruház oldalát

2.2. Alkalmazás kezelése

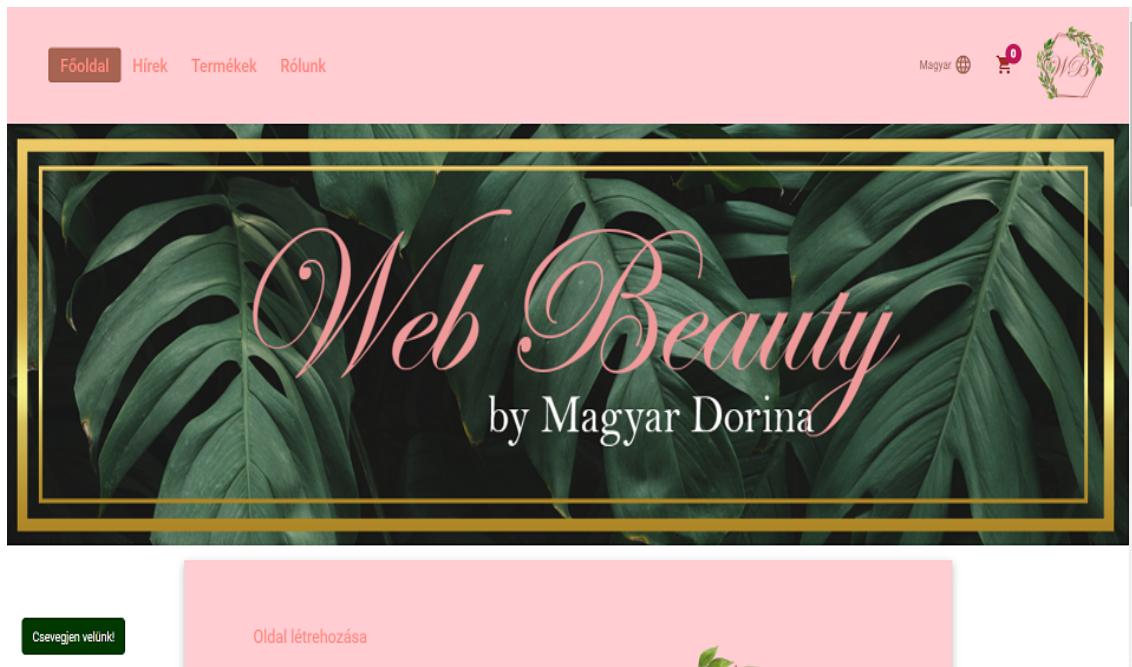
Az alkalmazás felületét két részre bonthatjuk. Az első rész maga a webáruház, amin a vásárlók megtekinthetik a termékeket és megrendelhetik őket, továbbá megnézhetik az üzemeltető által közzétett híreket, ezen felül különböző forrásokból információkat érhetnek el a vásárlással kapcsolatban. A második rész az üzemeltető által karbantartott adminisztrációs oldal. Ennek a felület használatához hitelesítés szükséges, ezzel is véde a vásárlók és a webáruház adatait. Az admin felületen lehetőség van ezen adatok kezelésére, szerkesztésére.

2.2.1. Webshop felület kezelése

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

Menüsor vagy más néven a toolbar. A 2.1-es ábrán a webáruház főoldalának egy részét láthatjuk. A képen tetején található az alkalmazás webshopjához tartozó vezérlő felület, ami a programban toolbar néven található meg. A vezérlő felület bal oldalán látható különböző menüpontok, amelyek segítségével navigálhatunk a differens oldalak között. A jobb oldalán pedig különböző funkciókkal bíró ikonokat és az oldal logóját. Az első ilyen ikon a nyelvválasztó, aminek a segítségével megváltoztathatjuk az oldal nyelvezetét. Jelenleg az angol és a

magyar nyelv közül lehet választani. A mellette lévő bevásárló kocsit ábrázoló ikon segítségével érhető el a felület kosár funkciója, ami tartalmazza az eddig hozzáadott termékeket. A kosárban szereplő termékek számáról egy előzetes információt kaphatunk a felette lévő jelvény számból.



2.1. ábra. Az alkalmazás megjelenése

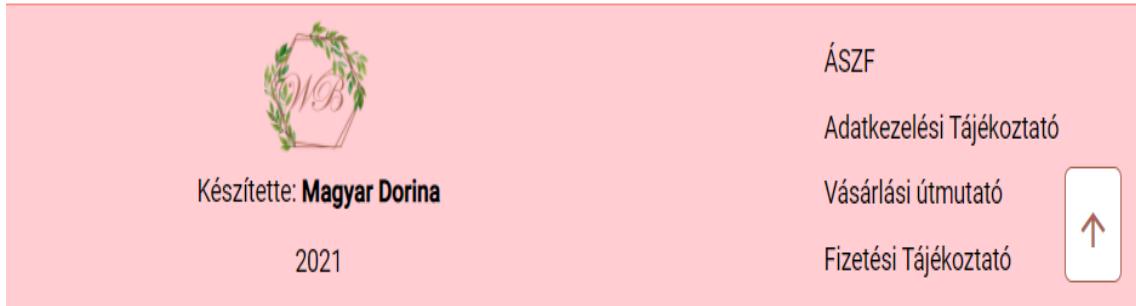
A webáruház tartalma, amit szakmai nyelven body-nak nevezünk. Ez a menüsor alatt található meg, amit a fentebb kifejtett vezérlő felület segítségével különböző oldalak tartalmi részét jeleníthetjük meg. Példaként 2.2-es ábra a és b részén láthatunk.

(a) Termékek oldala

(b) Hírek oldal

2.2. ábra. Webáruház tartalmi része

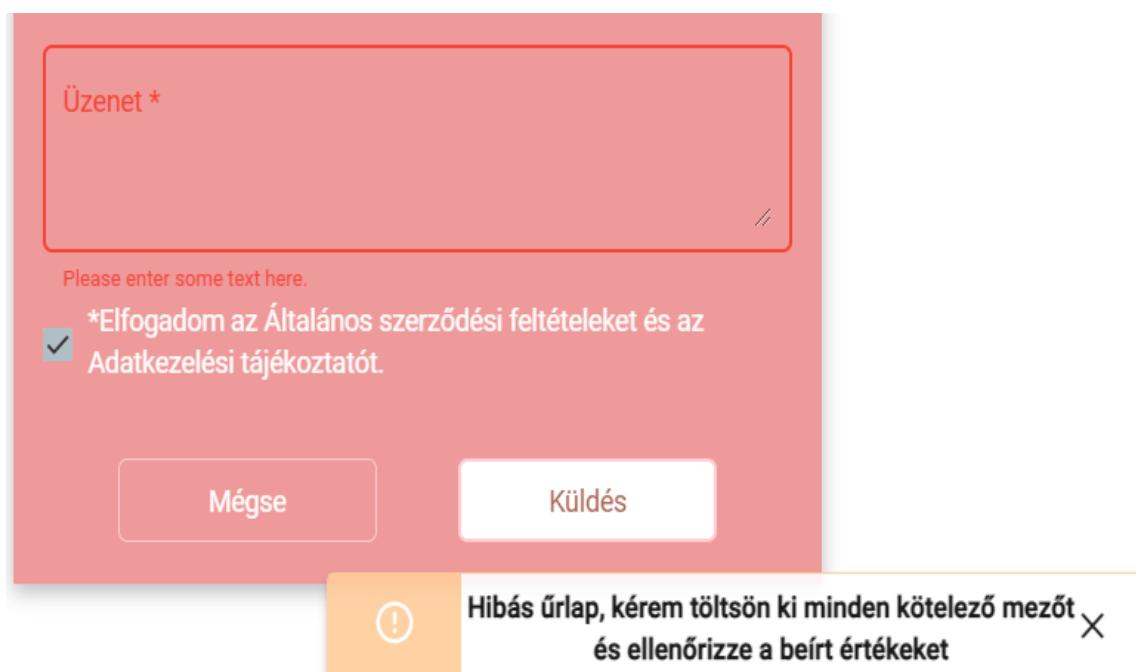
Lábrész vagy ahogy a programban található footer néven. A 2.3-as képen ez a felület látható. Itt olyan oldalak linkjei jelennek meg, amik tartalmazzák a vásárlókra és az oldalra vonatkozó fogyasztóvédelmi törvényeket és általános tájékoztatókat.



2.3. ábra. Az oldal lábrésze

Chatbot funkció a 2.1-es ábra legalján található zöld gomb segítségével érhető el, aminek a bemutatását a 2.3.1-es Vásárlással kapcsolatos információk alfejezetben kívánok kifejteni.

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 2.4-es ábráján is láthatjuk.



2.4. 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/amdin/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, akkor a 2.5-ös ábrán látott oldalt láthatjuk.



2.5. 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 emailcím és jelszó párossal az előbb emlegetett címek valamelyikén. Sikeres bejelentkezással az oldal átnavigálásra kerül az admin felületre. Ez az oldal két fő részből áll: a vezérlési és maga a tartalmi részéből.

Navigációs felület, aminek segítségével jeleníthetjük meg az alkalmazás tartalmi részét. A 2.6-os képen ez a navigációs rész a baloldalon található. Az admin felületen képesek vagyunk új termékeket vagy híreket hozzáadni és a meglévőket lista formájában megtekinteni. Továbbá a fentebbiekben már említett kapcsolat felvétel céljából a felhasználó által elküldött üzeneteket is kilistázásra kerülnek az oldalon. Az alkalmazáson belül lehetőségünk van új felhasználói

fiókot hozzáadni, viszont ezeket a fiókokat adatvédelmi okokból 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 felület részét.

Id	Név	Leírás	Ár	Kép	Menü
61b62241b415f9f7cc07e7b4	Fém szívószál	Környezetkimélő 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		...

2.6. ábra. Adminisztrációs felület megjelenése

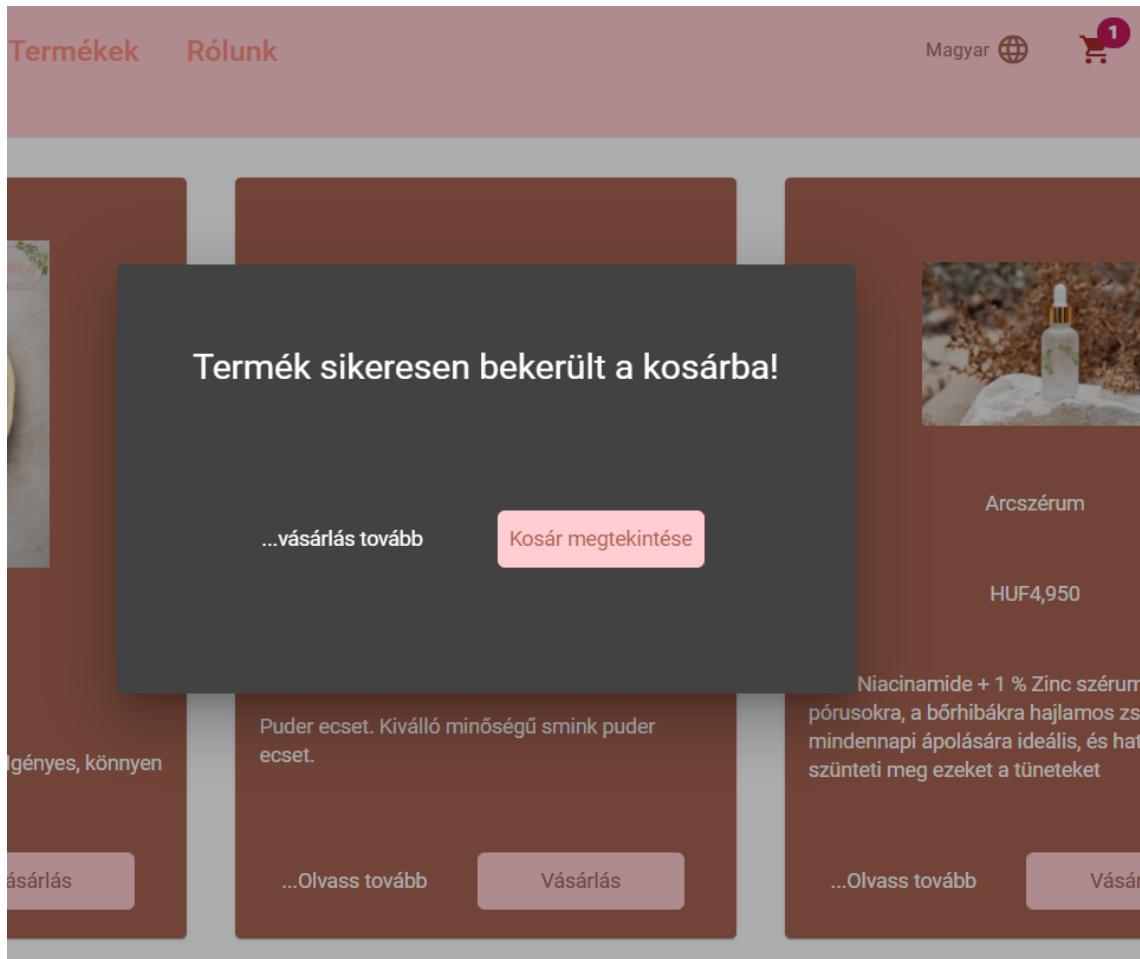
Tartalmi rész például a 2.6-os ábra jobboldalán látott táblázat, ami a termékek listáját tartalmazza. Az oldal ezen részén lehetőségünk van az adatok módosítására, törlésére vagy a fentebb említett termék hozzáadására a megjelenő form kitöltésével.

2.3. Rendelési folyamat

A rendelés folyamata hasonlóan zajlik, mint a legtöbb webáruháznak a rendelése történik. A kiválasztott termékek bekerülnek a kosárba és személyes adatok megadásával és vásárlás befejezésével elküldésre kerül az oldal üzemeltető számára, aki előkészíti azt. A következő két alfejezetben bemutatom egy példán keresztül hogyan is kerül egy rendelés leadását a termék kiválasztásától kezdve és milyen feltételei, információ vonzatai vannak a webáruházon történő vásárlásnak.

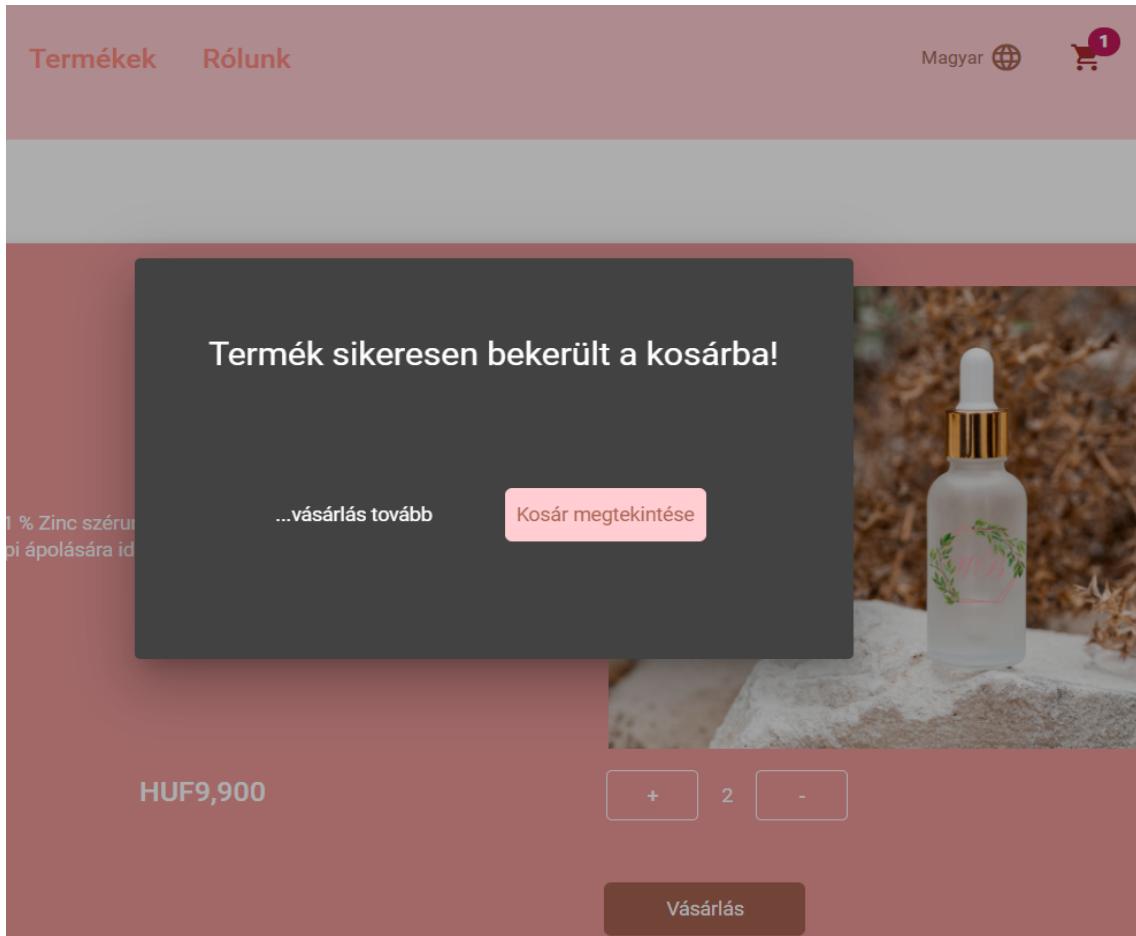
2.3.1. Rendelés leadása

A rendelés leadásához először is szükséges, hogy valami bekerüljön a bevásárlókósár tartalmába. Termék kosárba helyezésére két lehetőségünk van. Az első, hogy ha a termékek listájánál a kiválasztott termék kártyáján a vásárlás gombra kattintunk, mint ahogy a 2.7-es ábrán is megfigyelhető.



2.7. ábra. Termék vásárlása első példa

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 vásárolandó áru számát. Ahogy azt a 2.8-as második kép példáján is láthatjuk.



2.8. ábra. Termék vásárlása második példa

Ha a kiválasztott termékek a kosárba kerültek, azokat megtekinthetjük úgy, hogy ha a 2.7-es és a 2.8-as ábrákon is látott Kosár megtekintése gombot kiválasztjuk, vagy ha magára a bevásárlókocsi ikonjára kattintunk. Mindkettő lehetőséggel átirányít a kosár oldalára, ahol láthatjuk táblázat formájában a megvásárolandó árukat. Mint azt a 2.9-es képen is láthatjuk.



[← Vissza](#)

Bevásárlókocsi

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

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

[Tovább vásárolok](#)

[Vásárlás](#)



2.9. ábra. Beváráslo kosár

A bevásárlókosár tartalmának módosításra lehetőségünk van a mennyiség oszlopában látható gombok segítségével. Ha csökkentjük vagy növeljük egy áru darab mennyiségét, akkor azok az adatok, amiket ez befolyásolja dinamikusan változnak.

A táblázat alatti Vásárlás gombot kiválasztva az oldal átnavigál a fizetési felületre. Az alábbi 2.10-es képeken látható, hogy ezen a felületen négy stepper button magyarul lépegető gomb található. minden lépegetőnek külön funkciója van, amit a nevével azonosíthatunk.

(a) Első stepper button

(b) Második stepper button

2.10. ábra. Fizetési felület

Az első ilyen gombon adhatóak meg a szállítás és számlázási adatokkal kapcsolatos űrlap, aminek a helyes kitöltésével léphetünk a következő lépésre. A második stepper 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

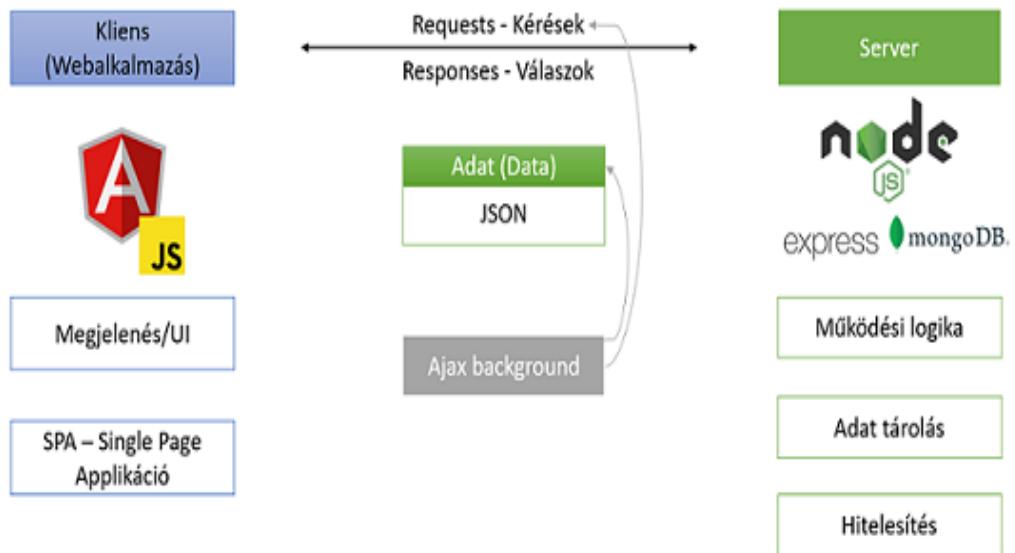
A vásárlással kapcsolatban

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) nevezetű szoftverkötégek segítségével készült. A **MEAN** megoldásverem egy ingyenes nyílt forráskódú szoftverek halmaza, ami lehetőséget kínál dinamikus weboldalak készítésére. A webalkalmazás két főrészre 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(rövidítés feloldása: 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 következő ábrán szeretném reprezentálni minden módon épül fel a webalkalmazás, továbbá megismertetni a két oldal kommunikációs kapcsolatát.



3.1. ábra. Az alkalmazás bemutatása

A 3.1-es ábrán látható a két oldal miképpen osztja meg az információkat egymás között. A front-end pontosabban mondva a kliensoldal **Angular** keretrendszerben **TypeScript** segítségével íródott. A front-end kommunikációja úgynevezett requestszövegekkel más néven kérésekkel (json típusú adattovábbítással) a háttérben aszinkron módon történik amire a szerveroldal responsokkal egyszóval válaszokkal felel. A back-end **Node.js** szoftverrendszer alapú, ami **Express** segítségével íródott. Az adatok tárolásáért a **MongoDB** nevezetű adatbázis felel.

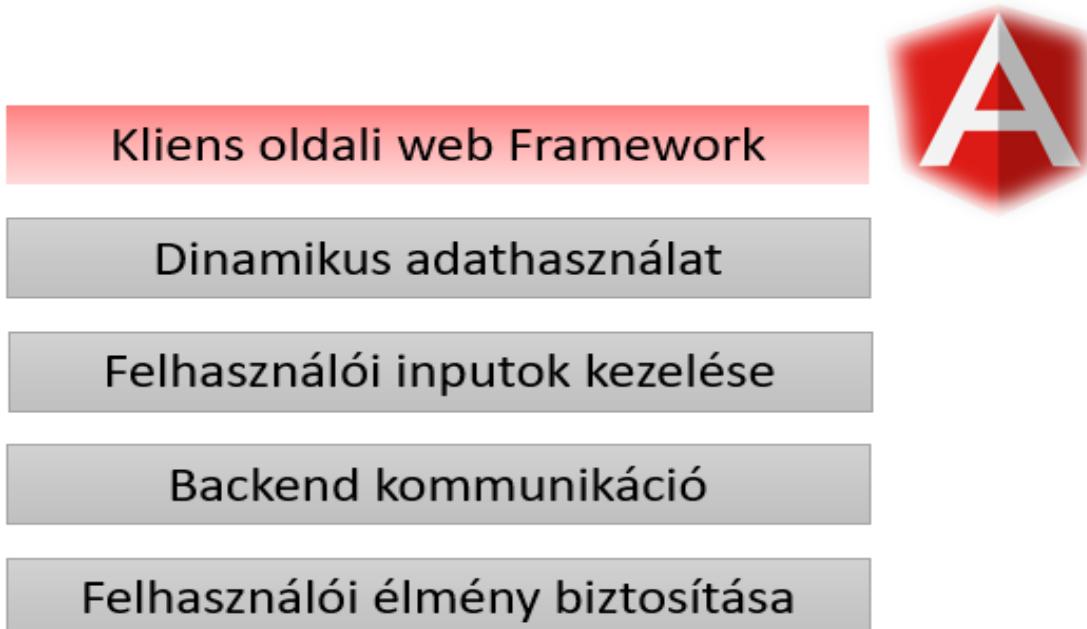
A következő blokkokban szeretném tételesen demonstrálni a fentebb említett front-end és back-end oldalakon használt módszerek alkalmazását és működését. Ezen felül szándékomban áll ismertetni az általam alkalmazott szoftverek technikai tulajdonságaikat.

3.2. Kliensoldalon használt technológiák bemutatása

A dokumentáció és az egész program fő eleme az Angular keretrendszer használata, aminek a segítségével dinamikus webalkalmazások hozhatóak létre. Az Angular egy nyílt forráskódú a 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 egyik legfőbb tulajdonsága, hogy egy kliensoldali keretrendszerrel beszélhetünk az esetében. Ennek köszönhetően képes feldolgozni és megjeleníteni a back-end felől érkező adatokat, így egy dinamikus webalkalmazást kapunk. Ezt úgy tudja biztosítani, hogy képes kapcsolatot kialakítani a szerveroldallal. Továbbá lehetőséget nyújt a felhasználó által beérkezett adatokat fogadására és kezelésére, ezen funkciója segítségével egy modern UI/UX felület készítésére alkalmazhatjuk. A program megírása során a 13.0.4 legújabb verziójú Angular CLI-t telepítettem. minden ezen jellemzői hozzátesznek, ahhoz, hogy Single Page Applikációnak(továbbiakban: SPA) nevezzük az általa támogatott weboldalakat. Olyan webhelyeket hívhatunk SPA-nak, amelyek egyetlen oldalra tölti be dinamikusan az adatokat, más szóval minden eleme egy oldalon található. Ennek köszönhetően a weboldalon való navigáláshoz nem kell betölteni külön DOM vagyis Dokumentumobjektum-modellek.

3.3. Kliensoldal működése

A weboldal felépítéséért, megjelenéséért és kinézetéért a HTML, TypeScript és SCSS hármas programozási nyelv felel. A HTML feladata az alkalmazás tartalmi megjelenítése, a scss pedig ezen tartalom formázása. A TypeScript pedig biztosítja a felhasználó által kiadott utasítások végrehajtását. Az Angular keretrendszerben ez a három nyelv egy-egy komponens darabjai.

3.3.1. Komponensek

A komponensek a programkód logikai darabjai. Két fő eleme van. Az első ilyen elem a templat-ek, ami az alkalmazás megjelenítéséért felel, ez tartalmazza a HTML-t. A második elem az osztályok amiben szerepelnek a metódusok és tulajdonságok, ez a rész a TypeScript fájlokban van definiálva.

3.3.2. Modulok

Az Angular alkalmazások modulárisak és saját moduláris rendszerrel rendelkezik, amit NgModules-nak nevezünk. Ennek a modulnak a segítségével különböző komponenseket, direktívákat és service fájlokat csoportosíthatunk a metaadatai segítségével. Az NgModule-nak öt ilyen metaadata van, aminek a használatával kategorizálhatjuk a komponenseinket felhasználás szerint.

NgModule metadatai:

- deklarációk (declarations): az itt szereplő komponensek kifejezetten ahhoz a module-hoz 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 deklarációnál implementált komponensek használnak
- szolgáltatók (providers): olyan osztályok szerepelnek itt, amelyek létrehozzák és menedzslik a service objektumokat első alkalomkor, amikor az Angularnak szüksége van a függőségek feloldásához.

3.3.3. Interfészek

Az interfészek olyan specifikáció az angular frameworkben, amelyek egy osztály által megvalósítandó tulajdonságok és metódusok összefüggő halmazát határozza meg. Tehát a segítségével létrehozható pár alapvető szabály a tulajdonságokra és a metódusokra amiket használunk az osztályon belül.

3.3.4. Service fájlok

A projektben a servise fájlok tartalmazzák azokat a függvényeket, amiknek a segítségével kapcsolatot alakíthatunk ki a szerveroldallal. Ezek a fájlok tartalmaznak bizonyos request kéréseket, amiknek a segítségével a felhasználó elindíthatja az adatlekérés folyamatát. A függvények kigyűjtésének célja, hogy egyszerűbben elérhetőek legyenek több komponens számára.

3.3.5. Admin mappa

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

3.3.6. Pages mappa

A webalkalmazás összes olyan komponense található ebben a mappában, ami nem kapcsolódik hitelesítési funkció az eléréséhez. Az áruház öt fő oldala található meg itt.

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

3.3.7. Oldalak közötti navigáció

Az oldalak közötti koordinálást az előbbiekben kifejtett modul fájlok egyike kezeli. Angularban a legjobb mód az, hogy ha a routerbe betöltést és a konfigurálást

különállóan történik. A konfigurálás az AppRoutingModuleban zajlik, míg a betöltés a legfelső szintű module azaz az AppModuleban van importálva, ami útválasztóként is szolgál.

3.3.8. Shared mappa

A shared mappa tartalmazza azokat a komponenseket és angular kiegészítő csomagokat, amiket több oldalon is megjelenítésre kerülnek. Ezek a komponensek és packagek a következők:

Angular Material egy felhasználói felület (UI) komponens könyvtár. Segítségével gyorsítja a fejlesztési folyamatot, konzisztens és elegán felületet biztosítva.

Alert üzenetek segítségével a felhasználó által elindított folyamatok állapotáról nyújthatunk információt.

Nyelvválasztás lehetőséget biztosít az oldalon található adatok többnyelvű megjelenítését.

Vissza gomb és a go to top gomb a nevéből kiindulva olyan gombok amiknek lehetőségével egyszerűbb használatot biztosít 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 Forráskódok 3.7.1-es Kliensoldali forráskódok alfejezetében található,

3.3.9. Egyéb fájlok és mappák

Ebben a blokkban szerepel minden olyan mappa és fájl amit nem tudtam az előző fejezetekekhez kapcsolni funkciójuk különbségük miatt.

- Assets mappa: minden olyan képfájlt tartalmaz, amit nem dinamikusan kapunk a szervertől. Ilyen képek például az oldalon használt logók, vagy 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 stílus elem leírása amit a komponensek közösen használnak

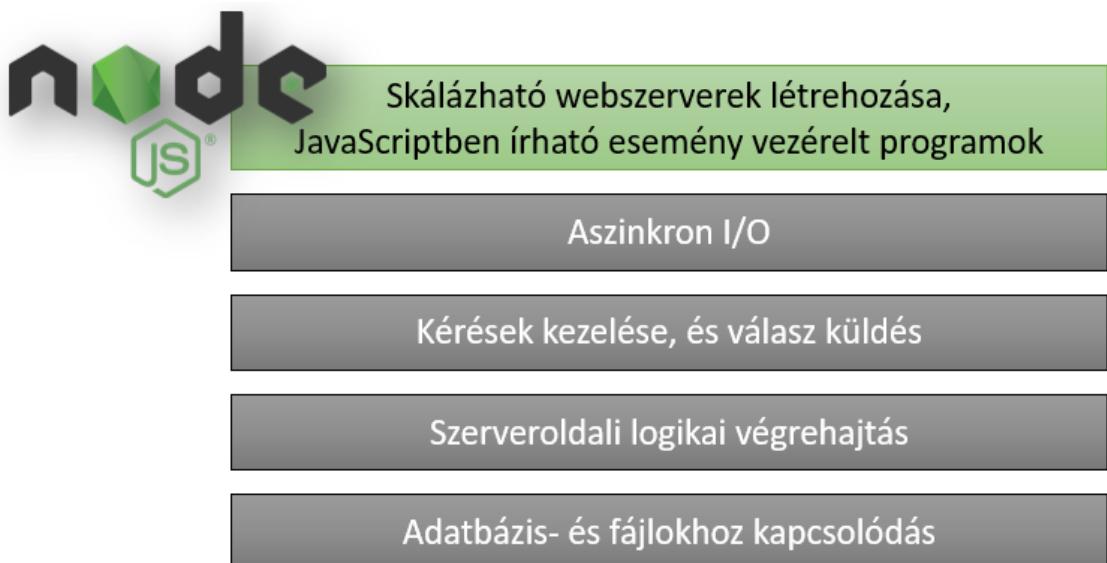
3.4. Szerveroldalon használt technológiák bemutatása

A szerveroldalon használt (3.1-es blokkban már említésre kerülő) módszerek kiválasztásuk előtt kulcsfontosságú szempontjuknak tartottam, hogy az Angular keretrendszerhez megfelelő kompatibilitással rendelkezzenek és alkalmazni tudjam őket a szakdolgozat elkészítése során. A következő technológiák mind olyan szoftverek, frameworkök vagy szerverek amikhez számtalan monográfia 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 tulajdonságaikat, főbb jellemzőiket további ábrák segítségével.

3.4.1. Node.js szoftverrendszer



3.3. ábra. NodeJS bemutatása

A webáruház back-end megírásánál 14.15.4-es verziójú Node.js szoftverrendszert használtam. A 3.3-as ábrán látható a Node.js bemutatása, ami összefoglalja a szoftverrendszer fontosabb jellemzőit. Az illusztráció első dobozában olvasható miszerint a Node.js skálázható webszerverek létrehozására alkalmas más szóval egy olyan rendszert tudunk létrehozni a támogatásával, ami több felhasználót képes egyidejűleg ki-szolgálni. Ezenfelül JavaScript nyelv segítségével olyan programok írhatóak, amely a komponensek közötti esemény interakciókat tekinti alapul, mászóval eseményvezérelt programok megírására alkalmas (ilyen például egy egérkattintás vagy billentyű leütés). Folytatólag a Node.js aszinkron tulajdonságával lehetővé teszi, hogy a kliensoldalról érkező kérések várakozási sorrendbe kerüljenek, ennek következtében a kliensoldal tovább folytathatja a feladatát.

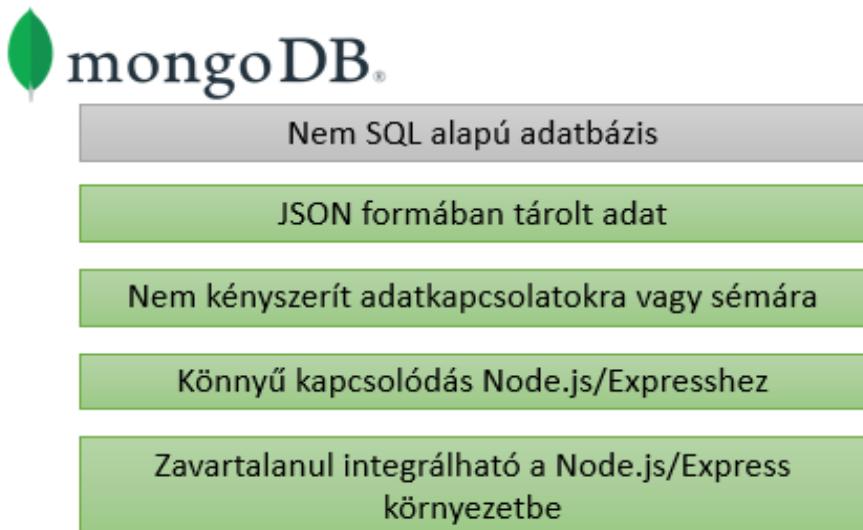
3.4.2. Express.js framework és Mongoose programozási könyvtár



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 a Node.js nehézség nélküli használatára lett fejlesztve. A 3.4-es ábrán látható az Express attribútumainak ismertetése. Az illusztráción felvetettem, hogy az Express egy Middleware-típusú rendszer következésképpen lehetővé teszi a MongoDB adatbázis szoftverhez való zavartalan kapcsolódást a Mongoose 6.0.12 verziójával kiegészítve, ami egy JavaScript-ben írt objektum-orientált programozási könyvtár.

3.4.3. MongoDB adatbázisszerver

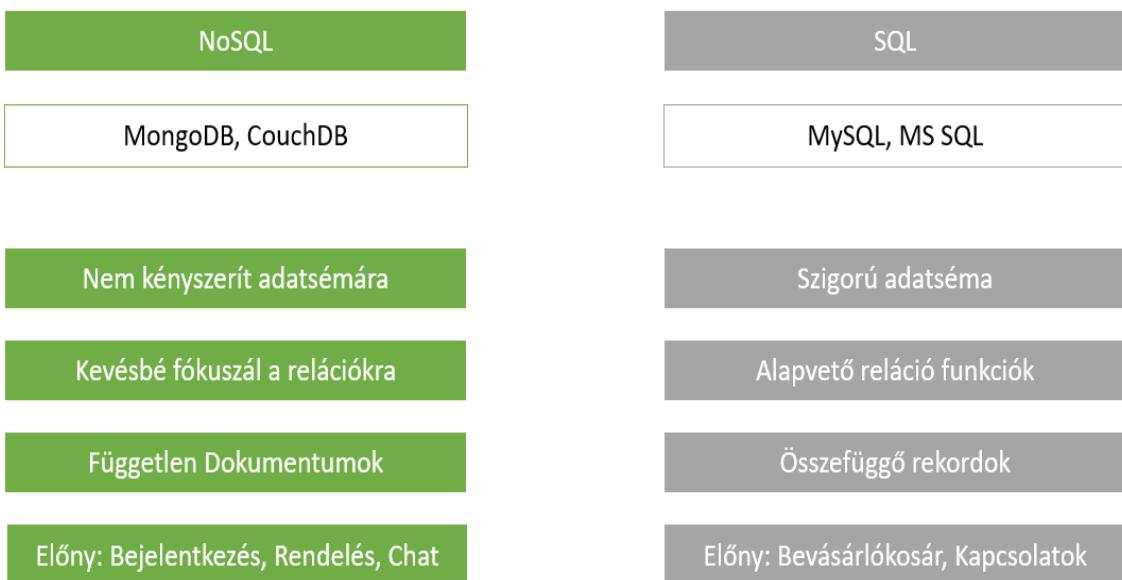


3.5. ábra. MongoDB bemutatása

A MongoDB egy nyílt forráskódú, NoSQL adatbázisszerverek közé sorolt szoftver. A NoSQL magyarán Nem SQL típusú adatbázisrendszer jelent. Jellemzően nem rekordokat és táblázatokat tárolnak mint az SQL típusú szerverek, hanem független dokumentumokat és gyűjteményeket archiválnak. Személyes véleményem szerint a 3.5-ös illusztrációval alátámasztva egyik legfőbb pozitív tulajdonságának éreztem a alkalmazás írása során, hogy az adatok JSON formátumban képes tárolni. Következésképpen a request és response folyamatok egyszerűsített és gyors működését képes biztosítani, mindenkorban lehetővé teszi a kliensoldalon megjelenítendő információk könnyebb feldolgozását.

3.4.4. NoSQL vs SQL adatbázisok összehasonlítása

A következő grafikai ábrán szándékozom röviden bemutatni és összehasonlítani a Nem SQL és az SQL alapú adatbázisokat jellegzetes tulajdonságaik szerint.



3.6. ábra. NoSQL vs SQL adatbázisok összehasonlítása

Mint a 3.6-os ábrán megfigyelhető szempontok szerint egy webáruházban kezelt adatok tárolására a NoSQL adatbázisok is kifejezetten alkalmasok. A NoSQL adatbázisszerverek jellemző tulajdonságai kulcsfontosságú szempontokkal szolgált a webáruház adatbázisának kiválasztásánál.

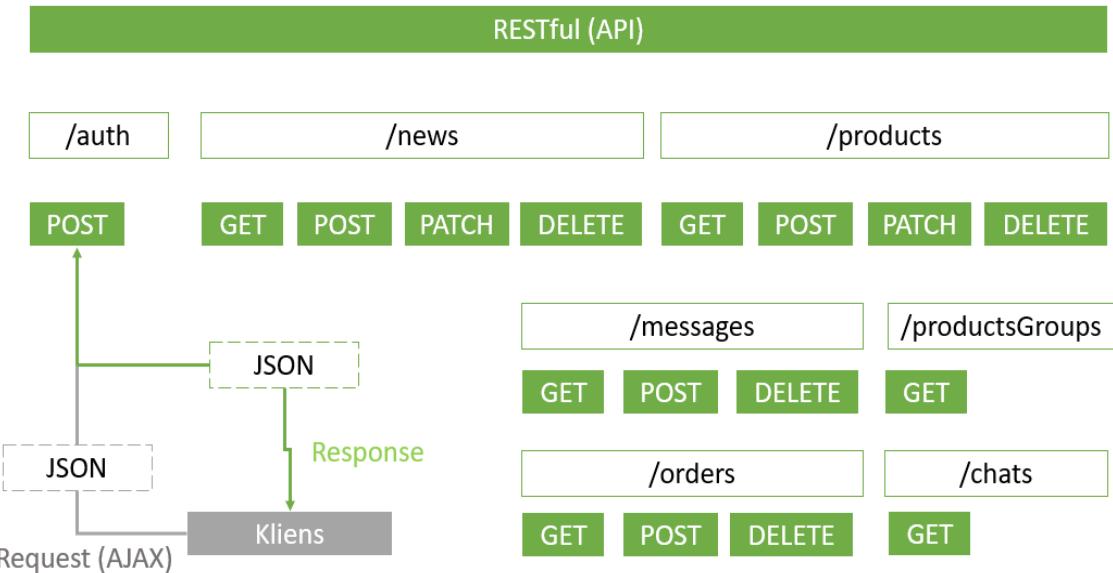
3.5. Szerveroldal működése

Ebben a fejezetben részletesen prezentálom az alkalmazás szerveroldali működését példának okáért milyen request és response hívások találhatóak a kódban, hogyan létesít kapcsolatot a webalkalmazás az adatbázissal...stb., valamint külön kitérek az adminisztrációs oldalon található hitelesítésére alkalmazott technikára is.

3.5.1. RESTful API - Végpont tervezés bemutatása

A webáruház megírása során RESTful API-t (feloldva: Representational State Transfer Application programming interface) magyarán reprezentación alapuló állapotátvitel nevezetű architekturális módszert használtam. Az API-k segítségével a felhasználó számára elérhető a weboldalon számos interakció. Ilyen interakcióknak számít például a bejelentkező felület.

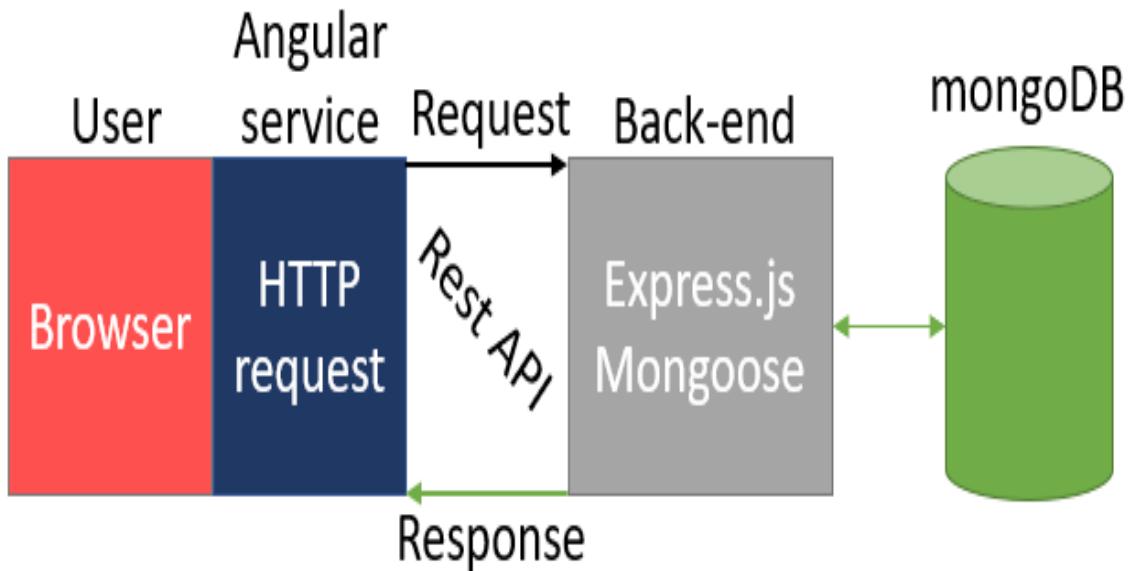
Az alábbi illusztráció szeretném bemutatni az alkalmazásban használt REST API hívásokat, ennek okán a 3.7-es ábrán láthatók a programban megírt végpontok.



3.7. ábra. Adatkezelés

A grafikán megfigyelhető hét különböző végpont ami az auth, hírek, termékek, termékcsoporthoz, üzenetek, chatek és rendeléseket fejezi ki. Az illusztrációt figyelemmel kísérve identifikálható, hogy nem minden végpont rendelkezik ugyan azokkal a kérésekkel. Szemléltetésképp vegyük figyelembe a hírekhez vonatkozó responsokat amik a GET, POST, PATCH és DELETE függvények, ezzel szemben a auth-hoz kizártan POST függvény tartozik. Ennek kifejezetten egy oka van, mégpedig az, hogy a webáruház egyes adatait nem szükséges módosítani tudni kliens oldalról.

A REST API-t megismerve és ezt az információt felhasználva szemléltethető és az alábbi ábrán látható miképpen éri el a felhasználó által kezdeményezett kérés az adatbázist és hogyan kerül válaszra.



3.8. ábra. Kapcsolat a szerverrel

A fenti 3.8-as 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 hozzá.

Felhasználó interakcióba lép a felülettel (például: egy gombra kattintva 3.1-es forráskód), ennek következményeként meghívódik egy a gombhoz tartozó TypeScript nyelven írt függvény 3.2-es forráskód.

```

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. Interakció 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 következő 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 Angular package POST request segítségével a megadott útvonalon küld egy REST API kérést a szerveroldal felé.

A szerveroldal fogadja ezt a kérést és továbbítja a MongoDB felé. Az alábbi 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 }
```

3.4. forráskód. Express POST végpont - JavaScript

Kliensoldalon és Szerveroldalon egyaránt látható a felhasználó által elindított kérés státuszát tartalmazó információ. A front-end-en egy úgynyvezett AlertService

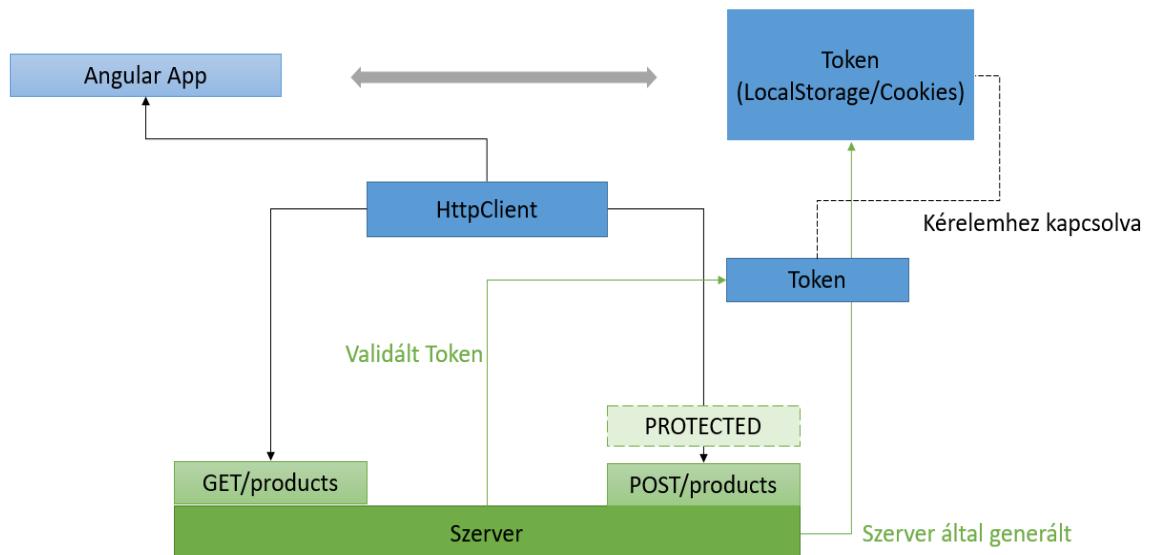
componens segítségével kap visszaigazolást az elindított kérés befejezéséről, míg a back-end-en az adatbázis felé elküldött kérés kódrészletében található ez az információ.

A webalkalmazásban szereplő további REST API hívásokat tartalmazó példakód-sorokat a 3.7-es Forráskódok 3.7.2-es Szerveroldali forráskódok nevezetű alfejezetben találhatóak.

3.5.2. Hitelesítés - Adminisztrációs felület védelme

Az alkalmazás megírása során akaratlagosan egy olyan webáruház létrehozása volt a végcélom, ami időszerű adatok kezelését tudja biztosítani. Következésképpen egy olyan felület megvalósítása gyakorlatban is, amely nem igényel programozón keresztüli beavatkozást. Ennek okán a szakdolgozat tartalmaz egy adminisztrációs oldalt, amely biztosítja a webalkalmazásban dinamikusan megjelenő adatok aktualitását, továbbá a leadott rendelések vásárlók által elküldött üzenetek megjelenítésére is szolgál. Kifejezetten ennek a blokk védelmében került bele külön hitelesítési felület.

Az alább található 3.9-es grafikán ez az engedélyezési folyamat elméleti működése látható.



3.9. ábra. Hitelesítés

Az illusztráció ábrázolt hitelesítési folyamat a következőképpen történik. Az alkalmazás front-end-ről érkező GET/products kérésre engedélyezés nélkül kap választ

a szervertől, mivel a webáruházban bejelentkezés hiányában is hozzáférhetővé kell tenni ezt az információt. Ezzel szemben a POST/products kérés nem következhet be hitelesítés nélkül. Tehát bejelentkezés során a szerver generál egy úgynevezett tokent és LocalStorage-ba elmenti, amit az új termék hozzáadás kérésnél ellenőriz. Az alkalmazásban ez a hitelesítési funkciót következőképpen valósul meg:

Ennek a funkciának létrehozásánál két kiegészítő package-et úgynevezett bcryptjs-t és jsonwebtoken-t használja a programkód. Az előbbi lehetőséget nyújt bizonyos adatok titkosítására (ilyen adat például a belépési jelszó), az utóbbi pedig bizonyos REST API kérések érvényesítésére alkalmas. Mikor létrehozásra kerül egy új felhasználó a bcrypt package egy úgynevezett hash metódusát hívja meg a program, aminek segítségével az új felhasználó jelszava titkosítva kerül be az adatbázisba. Ennek oka az, hogy ha véletlenül sikerül egy idegennek belépnie az adatbázisba, akkor ne tudja megszerezni az adott felhasználók jelszavát. Viszont, ha titkosítva kerül be az adat, a későbbiekben belépésnél nem lehet ellenőrizni, hogy a megfelelő jelszó került-e beírásra. Erre a problémára szolgál megoldásképp a bcryptjs compare metódusának használata, aminek segítségével összehasonlításra kerül az adott felhasználó adatbázisban szereplő jelszava és a begépelt jelszó hashelt változata, mivel ha pont a megfelelő adat kerül beírásra akkor a titkosított információ megegyezőnek kell lennie az adatbázisban található jelszóval. Sikeres bejelentkezésnél a szerveroldal generál egy tokent a jsonwebtoken package segítségével és minden olyan kérésnél amihez szükséges autentikáció, ellenőrzésre kerül ennek az érvényes tokennek a létezése. A programkódban generált tokenek hitelessége egy óráig él.

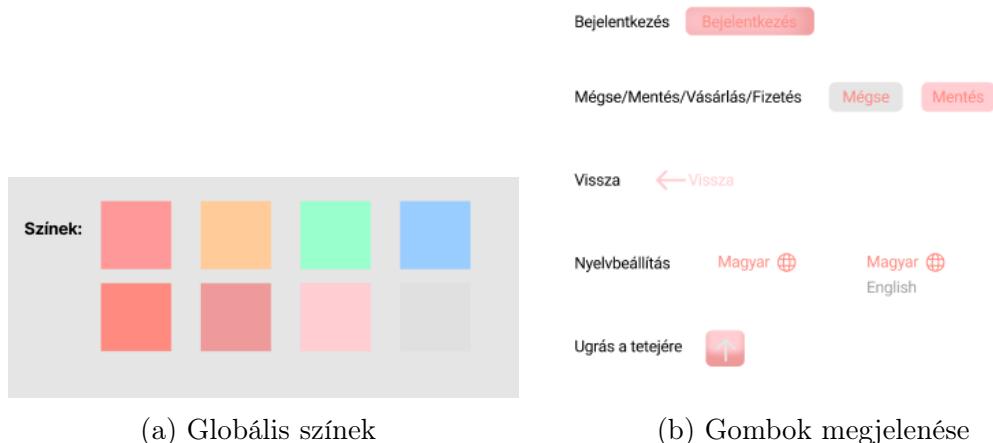
Mindent összevetve az adminisztrációs felület védve van az illetéktelen felhasználók belépésétől és bizonyos funkciók végrehajtásától.

3.6. Az alkalmazás megjelenése, web design

Az oldal megjelenéséhez kapcsolatos színvilág, és vizuális elemek nagy részben a saját munkáim. Az összes látvány elem mint a logó vagy egyes képek az Adobe Photoshop 2019-es programjával készültek, ezen kívül az alkalmazásban megjelenő ikonok a Google Icons - Material ikonjai kerültek felhasználásra. A program megírása előtt készítettem az oldalakhoz és különböző elemekhez drótváztervet. A fejlesztés során az oldalak nem teljesen követik az eredetileg tervezet felület megjelenését.

3.6.1. Figma szoftver - oldalvázlatok

A fentiekben említett drótvázterveket a Figma nevezetű vektorgrafikus szerkesztő segítségével készítettem. A tervezés során meghatároztam, és összegyűjtöttem az alkalmazás színvilágát, ami a 3.10-es ábra a-részeként látható. Ezeket a színeket globálisan került implementálásra a kódon belül. Ez azt jelenti, hogy ha bármikor lecserélek egyes színeket, akkor az adott szín mindenhol az újonnan megadott színként jelennek meg. Ezen felül olyan elemek kerültek tervezésre, mint például az értesítő üzenetek vagy a 3.10-es ábra b-részén az alkalmazásban megjelenő gombokat.



3.10. ábra. Figma kiegészítő elemek tervezete

Ezeken az összefoglaló elemeken felül vázlatosan elkészítettem az egyes oldalak megjelenési tervét, amit a 3.11-es képein fellelhető. Ahogy a két képet összehasonlítjuk az alkalmazásban megjelenő oldalakkal észrevehetően nagy változásokat eszközöltem egyes megjelenéseken.



3.11. ábra. Figma kiegészítő elemek tervezete

3.6.2. Logók és grafikai elemek

Az alkalmazás grafikai elemei elkészítése során próbáltam figyelembe venni a program letisztult megjelenését és termézeszeten annak színvilágát. Az webáruházban két különböző színű logó kerül felhasználásra. A 3.12-es ábra a. képén látható sötétebb kerettel és szöveggel rendelkező logót a másik b. logón szereplő hasonló színét alkalmazó háttereken alkalmazom, mint például az oldalak menüsora. Amíg a b. logót a fehér hátterekkel rendelkező oldalakon kerül megjelenítésre.



3.12. ábra. Grafikai elemek

A webshopon megjelenő további grafikai elemek mint például a 3.13-as ábrán található képek célja, hogy változatosabb megjelenést kölcsönözzön az alkalmazás számára.



3.13. ábra. Grafikai elemek

3.7. Forráskódok

3.7.1. Kliensoldali forráskódok

Ebben az alfejezetben található a kliensoldal működésénél említett chatbot bemutatása forráskódok segítségével. Ezt a funkciót úgy került megvalósításra, hogy előre megírt és adatbázisban eltárolt kérdéseket és válaszokat kérdezek le és jelenítek meg animációk segítségével. Az alábbi forráskódok ezt tartalmazzák.

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 ebben a listában szereplő kérdéseket HTML fájlban 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     ...

```

3.7. forráskód. Chatbot kérdéshez török szöveg megjelenítése - HTML

A beszélgetés imitálásához scss-ben írt stílus fájlt használtam az alábbi 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

Alább található forráskódok a 3.5.1-es fejezetben már (POST request segítségevel) kifejtett JavaScriptben írt kérés mellett a többi példa lekérés látható. A programban használt végpontok, amik bemutatásra kerülnek a következők: GET, DELETE és PUT műveletek.

GET/news 3.9 végpont és a visszaérkező json fájl 3.10:

```

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 letrehozása",
7       "description": "Az oldal letrehozása Magyar Dorina  
Szakdolgozata elkeszítése celjából tortent. Jelenleg ez az  
oldal nem üzemel! Nem kerülnek értékesítésre azok a termékek  
, amik az aruházban találhatóak! Ha további kérdése lenne a  
Rólunk feliratú menünek keresztül kapcsolatba lephet velem és  
 minden kérdésre email formájában válaszolok. Megérteseteket  
 előre is kérlek!",
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 végpont és a visszaérkező json fájl 3.12:

```

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 letrehozasa valtozas",
7       "description": "Az oldal letrehozasa Magyar Dorina
                     Szakdolgozata elkeszitese celjabol tortent. Jelenleg ez az
                     oldal nem uzemel! Nem kerulnek ertekesitesre azok a termek
                     , amik az aruhazban talalhatoak! Ha további kerdesse lenne a
                     Rolunk feliratu menuek keresztul kapcsolatba lephet velem es
                     minden kerdesre email formajaban valaszolok. Megerteseteket
                     elore is koszonom!",
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 végpont:

```
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

TBA

3.8.2. Szerveroldal tesztelése

Szerveroldal REST API kérések tesztelése Postman program segítségével. A táblázat első oszlopában a requestekre vonatkozó hitelesítési kötelezettségéről található információ. A második oszlopban a kérések URL címe olvasható, ezzel a címmel érhető el a back-end-en megírt request függvények. A harmadik és negyedik oszlopban pedig a kérés végrehajtásáról nyújt információkat. 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	TBA	TBA
POST			
Token	Request URL	Status	Message
igen	api/products	500 Server Error	Unexpected field
igen	api/news	TBA	TBA
nem	api/messages	TBA	TBA
nem	api/orders	TBA	TBA
PUT			
Token	Request URL	Status	Message
igen	api/products	500 Server Error	Unexpected field
igen	api/news	TBA	TBA
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	TBA	TBA

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

3.9. Továbbfejlesztési lehetőségek

4. fejezet

Összegzés

Lorem ipsum dolor sit amet, consectetur adipiscing elit. In eu egestas mauris. Quisque nisl elit, varius in erat eu, dictum commodo lorem. Sed commodo libero et sem laoreet consectetur. Fusce ligula arcu, vestibulum et sodales vel, venenatis at velit. Aliquam erat volutpat. Proin condimentum accumsan velit id hendrerit. Cras egestas arcu quis felis placerat, ut sodales velit malesuada. Maecenas et turpis eu turpis placerat euismod. Maecenas a urna viverra, scelerisque nibh ut, malesuada ex.

Aliquam suscipit dignissim tempor. Praesent tortor libero, feugiat et tellus porttitor, malesuada eleifend felis. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nullam eleifend imperdiet lorem, sit amet imperdiet metus pellentesque vitae. Donec nec ligula urna. Aliquam bibendum tempor diam, sed lacinia eros dapibus id. Donec sed vehicula turpis. Aliquam hendrerit sed nulla vitae convallis. Etiam libero quam, pharetra ac est nec, sodales placerat augue. Praesent eu consequat purus.

Ábrák jegyzéke

2.1.	Az alkalmazás megjelenése	7
2.2.	Webáruház tartalmi része	7
2.3.	Az oldal lábrésze	8
2.4.	Alert üzenet	8
2.5.	Bejelentkezási oldal	9
2.6.	Adminisztrációs felület megjelenése	10
2.7.	Termék vásárlása első példa	11
2.8.	Termék vásárlása második példa	12
2.9.	Bevárászló kosár	13
2.10.	Fizetési felület	14
3.1.	Az alkalmazás bemutatása	16
3.2.	Az Angular keretrendszer bemutatása	17
3.3.	NodeJS bemutatása	22
3.4.	Express bemutatása	23
3.5.	MongoDB bemutatása	24
3.6.	NoSQL vs SQL adatbázisok összehasonlítása	25
3.7.	Adatkezelés	26
3.8.	Kapcsolat a szerverrel	27
3.9.	Hitelesítés	30
3.10.	Figma kiegészítő elemek tervezete	32
3.11.	Figma kiegészítő elemek tervezete	32
3.12.	Grafikai elemek	33
3.13.	Grafikai elemek	33

Táblázatok jegyzéke

3.1. Szerveroldali végpontok tesztelése 39

Forráskódjegyzék

3.1.	Felhasználói interakció - HTML fájl	27
3.2.	Interakció függvényhívás - TypeScript fájl	27
3.3.	HttpClient POST request - Service TypeScript fájl	28
3.4.	Express POST végpont - JavaScript	29
3.5.	GET/chat lista lekérés - TypeScript	34
3.6.	Chatbot megjelenítése - HTML	34
3.7.	Chatbot kérdéshez török szöveg megjelenítése - HTML	34
3.8.	Chatbot beszélgetés animáció - scss	35
3.9.	GET/news végpont	36
3.10.	GET/news JSON	36
3.11.	PUT/news végpont	37
3.12.	PUT/news JSON	37
3.13.	DELETE/news végpont	38