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. 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 témaválasztás célja egy személyesen ismert vállalkozó megkeresésén alapult, aki szívesen értékesítene az általam készített webáruházon keresztül a termékeit. Az alap problémát az vetette fel részemről, 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 látszatát 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 legenerált válaszokból álló szöveges adathalmazról beszélhetünk, ami kulcsszavas keresés segítségével zajlik.

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

[2] sokféle megközelítést találhatunk a definícióját illetően. Személy szerint azt gondolom, hogy az MI egy tudatos gondolkozásra alkalmas, emberi beavatkozás nélküli cselekvőképes létforma, amit a legtöbbször számítástechnikai eszközökhöz/gépekhez társítunk.

2. fejezet

Felhasználói dokumentáció

Az alkalmazás készítése során fontos szempont volt, hogy egy felhasználóbarát webáruházat hozzak létre. A webshop használata egyszerű letisztult felülettel rendelkező program olyan funkciókkal kiegészítve, amik megkönnyítik 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, amik a következőek:

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-címen érhető el: http://webbeauty.us-east-2.elasticbeanstalk.com

2.1.2. Alklamazás indítása saját gépről

Az előző módszert azért neveztem könnyebben alkalmazhatónk, 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, ami 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 kódszerkesztőjét 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 url-címen 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 parancssort: npm install -g @angular/cli

7. ha ez sikeresen megtörtént akkor futtassuk le az npm i parancssort, aminek a 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 védve a vásárlók és a webáruház adatait. Az admin felületen lehetőség van ezen adatok kezelésére, szerkesztésére.

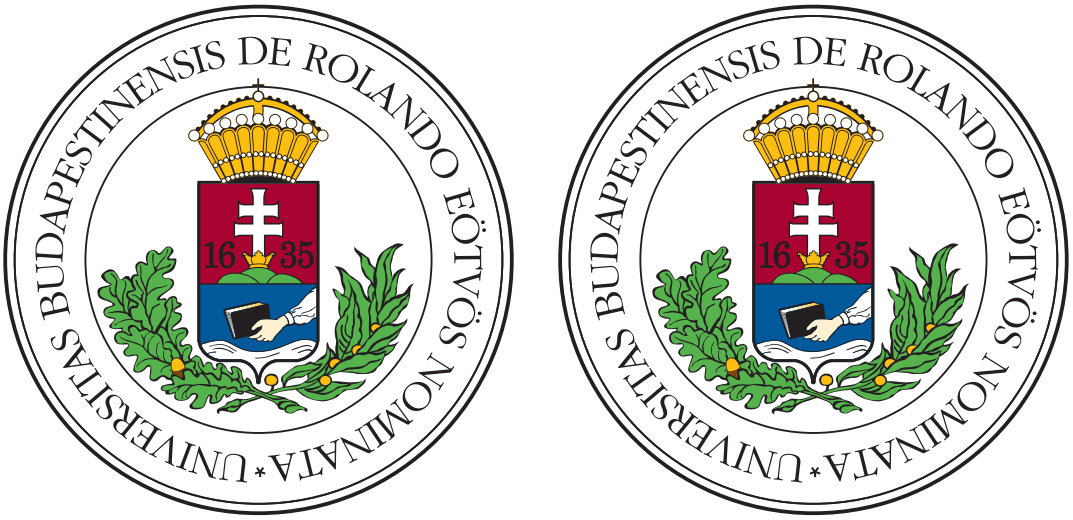
2.2.1. Webshop felület kezelése

Ut aliquet nec neque eget fermentum. Cras volutpat tellus sed placerat elementum. Quisque neque dui, consectetur nec finibus eget, blandit id purus. Nam eget ipsum non nunc placerat interdum.

2.1. ábra. Quisque ac tincidunt leo

2.2.2. Adminisztrációs felület kezelése

In non ipsum fermentum urna feugiat rutrum a at odio. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Nulla tincidunt mattis nisl id suscipit. Sed bibendum ac felis sed volutpat. Nam pharetra nisi nec facilisis faucibus. Aenean tristique nec libero non commodo. Nulla egestas laoreet tempus. Nunc eu aliquet nulla, quis vehicula dui. Proin ac risus sodales, gravida nisi vitae, efficitur neque, Figure 2.2:



(a) Vestibulum quis mattis urna

(b) Donec hendrerit quis dui sit amet venenatis

2.2. ábra. Aenean porttitor mi volutpat massa gravida

Nam et nunc eget elit tincidunt sollicitudin. Quisque ligula ipsum, tempor vitae tortor ut, commodo rhoncus diam. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Phasellus vehicula quam dui, eu convallis metus porta ac.

2.3. Rendelési folyamat

Nam magna ex, euismod nec interdum sed, sagittis nec leo. Nam blandit massa bibendum mattis tristique. Phasellus tortor ligula, sodales a consectetur vitae, placerat vitae dolor. Aenean consequat in quam ac mollis.

Phasellus tortor	**Aenean consequat**
Sed malesuada	Aliquam aliquam velit in convallis ultrices.
Purus sagittis	Quisque lobortis eros vitae urna lacinia euismod.
Pellentesque	Curabitur ac lacus pellentesque, eleifend sem ut, placerat enim. Ut auctor tempor odio ut dapibus.

2.1. táblázat. Maecenas tincidunt non justo quis accumsan

2.3.1. Vásárlással kapcsolatos információk

Mauris a dapibus lectus. Vestibulum commodo nibh ante, ut maximus magna eleifend vel. Integer vehicula elit non lacus lacinia, vitae porttitor dolor ultrices. Vivamus gravida faucibus efficitur. Ut non erat quis arcu vehicula lacinia. Nulla felis mauris, laoreet sed malesuada in, euismod et lacus. Aenean at finibus ipsum. Pellentesque dignissim elit sit amet lacus congue vulputate.

Quisque	**Suspendisse**		**Aliquam**		**Vivamus**	
	Proin	Nunc	Proin	Nunc	Proin	Nunc
Leo	2,80 MB	100%	232 KB	8,09%	248 KB	8,64%
Vel	9,60 MB	100%	564 KB	5,74%	292 KB	2,97%
Auge	78,2 MB	100%	52,3 MB	66,88%	3,22 MB	4,12%

2.2. táblázat. Vivamus ac arcu fringilla, fermentum neque sed, interdum erat. Mauris bibendum mauris vitae enim mollis, et eleifend turpis aliquet.

2.3.2. Rendelés leadása

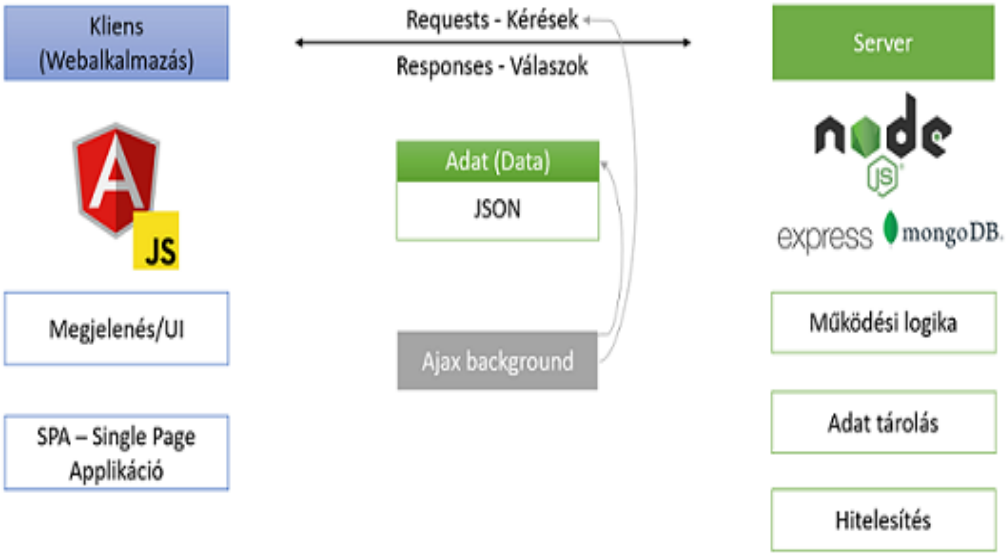
Nunc porta placerat leo, sit amet porttitor dui porta molestie. Aliquam at fermentum mi. Maecenas vitae lorem at leo tincidunt volutpat at nec tortor. Vivamus semper lacus eu diam laoreet congue. Vivamus in ipsum risus. Nulla ullamcorper finibus mauris non aliquet. Vivamus elementum rhoncus ex ut porttitor.

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 szoftverköteg 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 milyen 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 requestekkel 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.

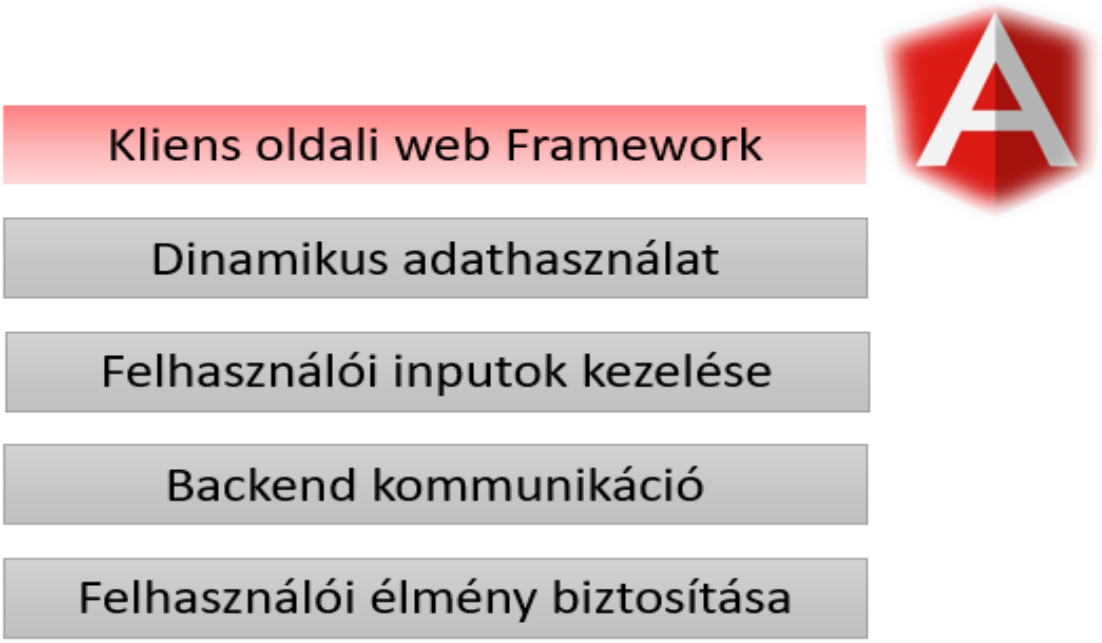
A következő blokkokban szeretném tételesen demonstrálni a fentebb említett front-end és back-end oldalakon használt módszerek alkalmazását és működését. Ezen felül szándékomban áll ismertetni az általam alkalmazott szoftverek technikai tulajdonságaikat.

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 keretrendszerről 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-modelleket.

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 menedzselik 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 packegek 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 fejezetekhez 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

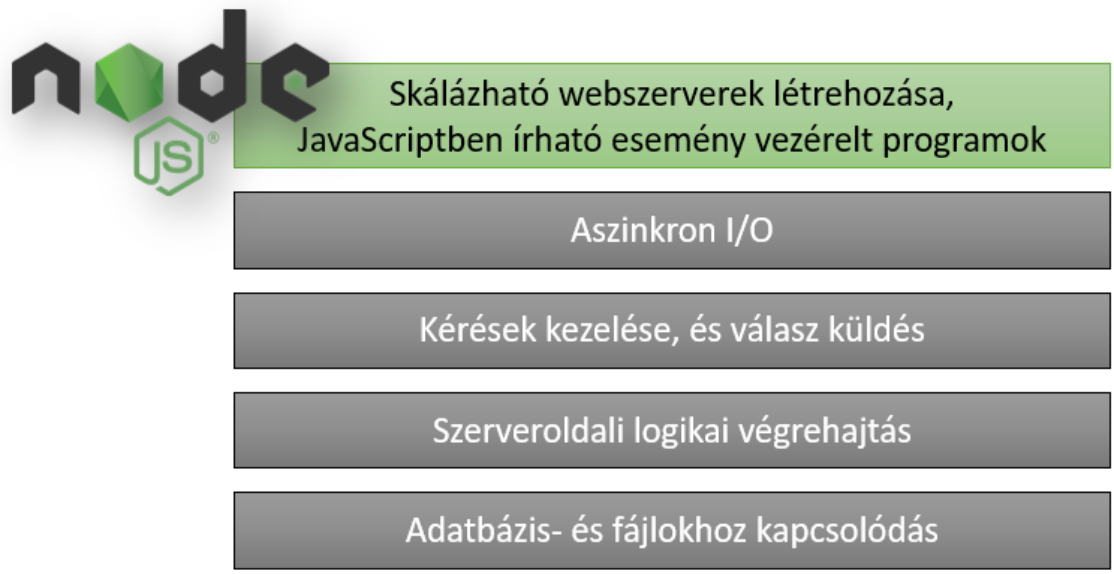
3.4. Szerveroldalon használt technológiák bemutatása

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

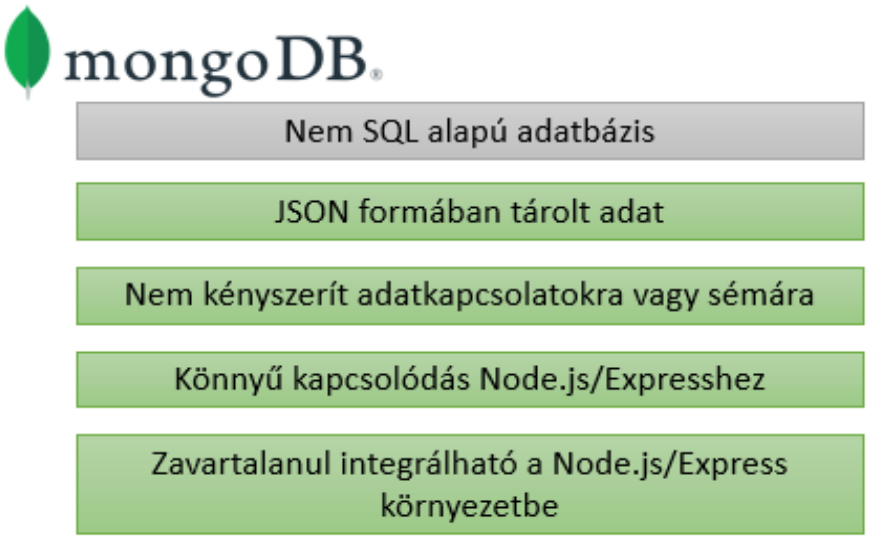
A webáruház back-end megírásánál 14.15.4-es verziójú Node.js szoftverrendszert használtam. A 3.3-as ábrán látható a Node.js bemutatása, ami összefoglalja a szoftverrendszer fontosabb jellemzőit. Az illusztráció első dobozában olvasható miszerint a Node.js skálázható webszerverek létrehozására alkalmas más szóval egy olyan rendszert tudunk létrehozni a támogatásával, ami több felhasználót képes egyidejűleg kiszolgálni. 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

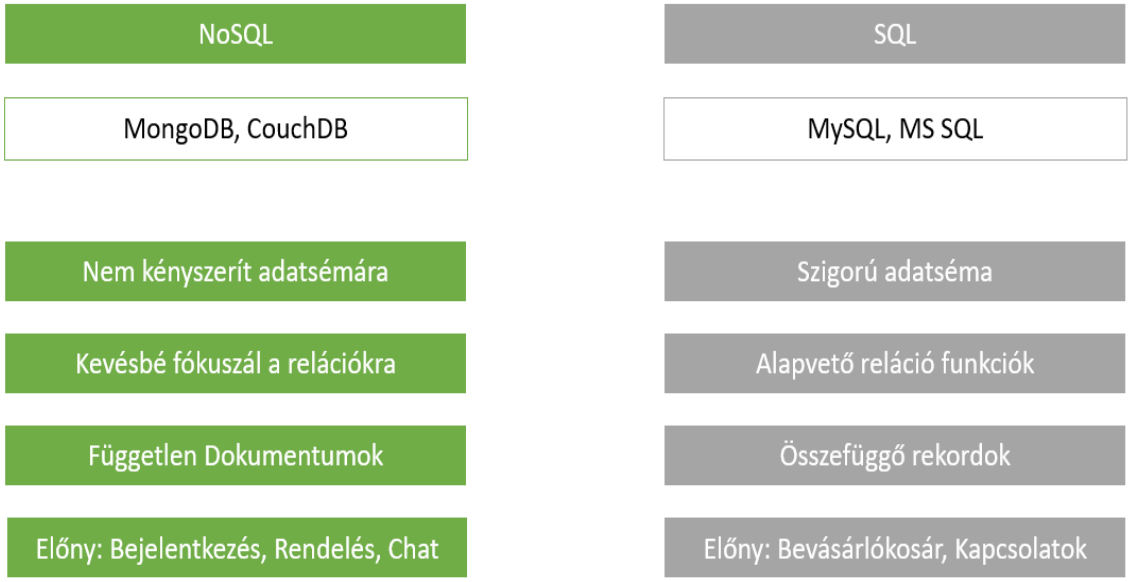


3.5. ábra. MongoDB bemutatása

A MongoDB egy nyílt forráskódú, NoSQL adatbázisszerverek közé sorolt szoftver. A NoSQL magyarán Nem SQL típusú adatbázisrendszert 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, mindeközben 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 No SQL adatbázisok is kifejezetten alkalmasok. A NoSQL adatbázisszerverek jellemző tulajdonságai kulcsfontosságú szempontokkal szolgált a webáruház adatbázisának kiválasztásánál.

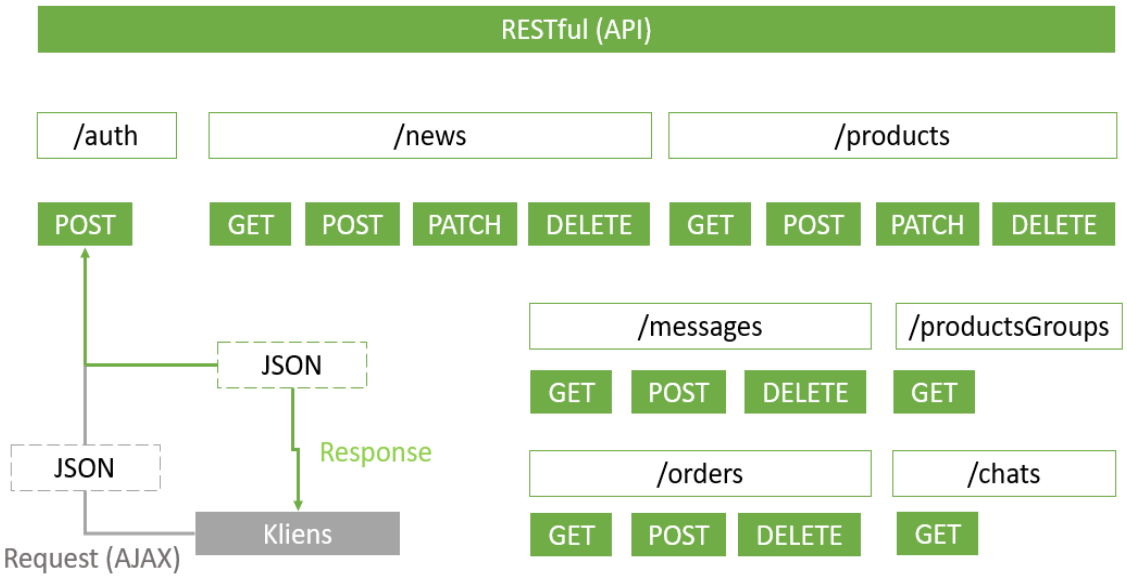
3.5. Szerveroldal működése

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 tervek bemutatása

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

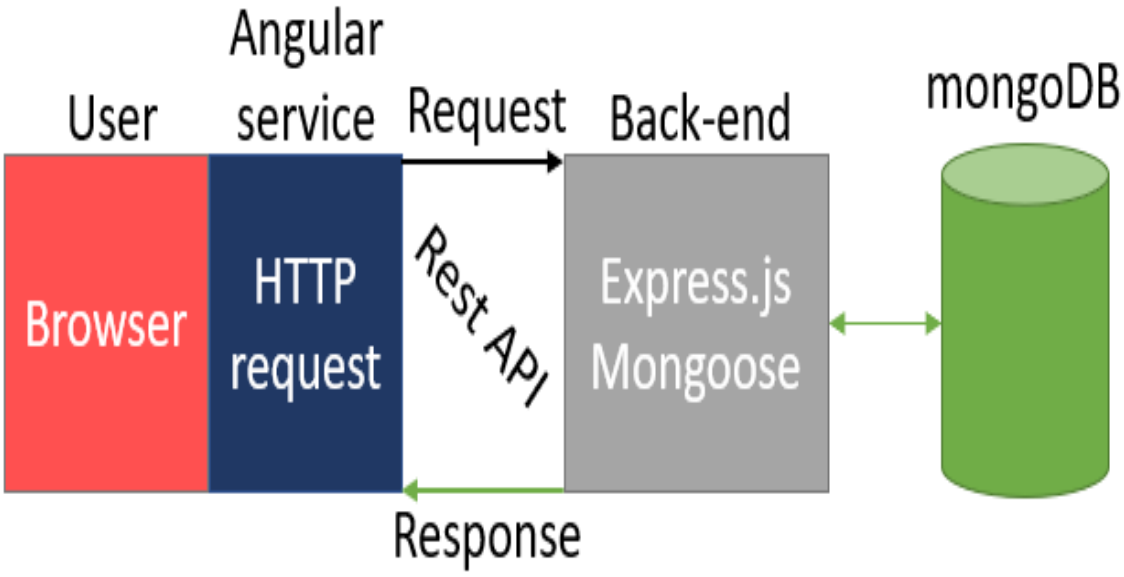
Az alábbi illusztráción szeretném bemutatni az alkalmazásban használt REST API hívásokat, ennek okán a 3.7-es ábrán láthatók a programban megírt végpontok.



3.7. ábra. Adatkezelés

A grafikán megfigyelhető hét különböző végpont ami az auth, hírek, termékek, termékcsoportok, 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 kizárólag 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 alább 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)="
       onAddMessage()" *ngIf="!isLoading">
2  ....
3  <button class="btn-secondary" type="submit" [disabled]="!
       confirmed.checked">
4    {{'GENERIC.ACTION.SEND' | translate}}
5  </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;
```

```
    this.contactService.sendMessage(
      this.form.value.firstName,
      this.form.value.lastName,
      this.form.value.email,
      this.form.value.description,
      this.form.value.image,
    )
    this.isLoading = false
    this.form.reset();
  }
```

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.

```
sendMessage(
  firstName: string,
  lastName: string,
  email: string,
  description: string,
  image: File | string
){
  const messagesData = new FormData();
  messagesData.append("firstName", firstName);
  messagesData.append("lastName", lastName);
  messagesData.append("email", email);
  messagesData.append("description", description);
  messagesData.append("image", image, firstName);
  this.http.post<{message: string, messages: Messages}>
  (environment.apiUrl + "messages", messagesData)
    .subscribe(responseData => {
      const messages: Messages = {
        id: responseData.messages.id,
        firstName: firstName,
        lastName: lastName,
        email: email,
        description: description,
        imagePath: responseData.messages.imagePath,
      };
      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.postMessages = (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 úgynevezett AlertService

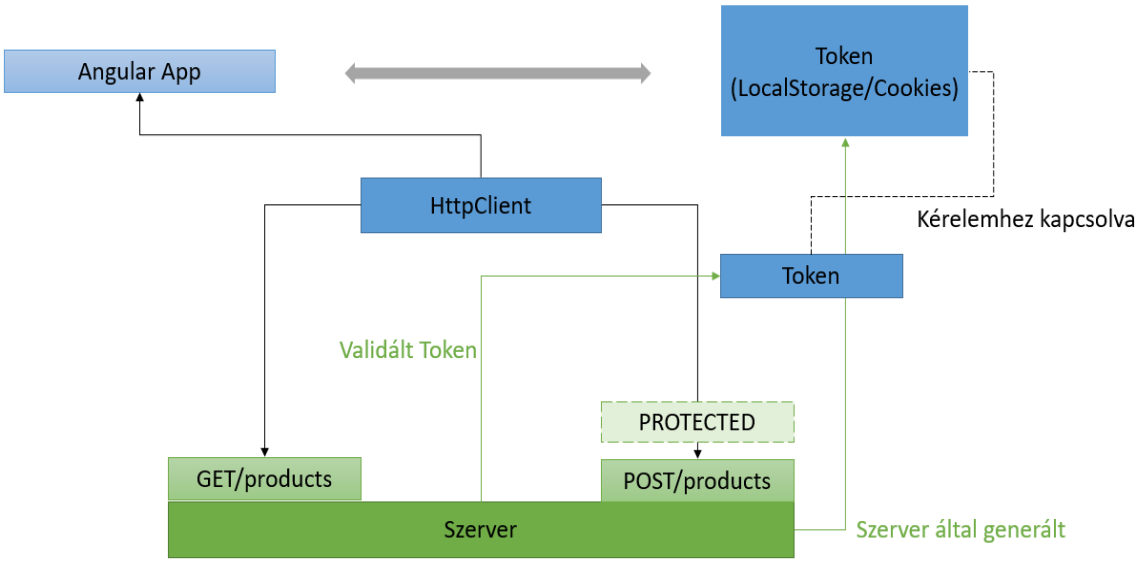
componens segítségével kap visszaigazolást az elindított kérés befejezéséről, míg a back-end-en az adatbázis felé elküldött kérés kódrészletében található ez az információ.

A webalkalmazásban szereplő további REST API hívásokat tartalmazó példakódsorokat 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 illusztráción 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 funkciónak 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 információnak 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

3.6.1. Figma szoftver - oldalvázlatok

A programkód kliensoldal elkészítése előtt készítettem pár oldalvázlatot a Figma nevezetű vektorgrafikus szerkesztő segítségével

3.6.2. Logók és 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.

```
async ngOnInit() {
  await this.chat.getChat();
  this.chatSub = this.chat.getUpdateListener()
  .subscribe(chat => {
    this.chatList = chat;
  });
}
```

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.

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

3.6. forráskód. Chatbot megjelenítése - HTML

```
<div class="animated display-message">
  <div class="chat__message chat__message_B" style="--delay: 2s">
    <div class="chat__content">
      {{selectedChat.title}}
    </div>
  </div>
  <div class="chat__message chat__message_A" style="--delay: 6s">
    <div class="chat__content">
```

```
9           {{selectedChat.description}}
10        </div>
11      </div>
12      ...
```

3.7. forráskód. Chatbot kérdéshez tatoró 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 segítségével) 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:

```
exports.getChat = (req, res, next) => {
  Chat.find().then(result => {
    res.status(200).json({
      message: "Chat fetched successfully",
      chat: result,
    });
  });
}
```

3.9. forráskód. GET/news végpont

```
{
  "message": "News fetched successfully!",
  "news": [
  {
    "_id": "616ece72537ef3531fde6d47",
    "title": "New News",
    "description": "Lorem Ipsum is simply dummy text of the
       printing and typesetting industry. Lorem Ipsum has been the
       industry's standard dummy text ever since the 1500s, when an
        unknown printer took a galley of type and scrambled it to
       make a type specimen book. It has survived not only five
       centuries, but also the leap into electronic typesetting,
       remaining essentially unchanged.",
    "imagePath": "http://localhost:3000/images/news/new-news
       -1634651762652.jpg",
    "__v": 0
  }, ...
  ]
}
```

3.10. forráskód. POST/news JSON

PUT/news 3.11 végpont és a visszaérkező json fájl 3.12:

```
exports.putNews = (req, res, next) => {
  let imagePath = req.body.imagePath;
  if(req.file) {
    const url = req.protocol + "://" + req.get("host");
    imagePath = url + "/images/news/" + req.file.filename
  }
  const news = new News({
    _id: req.body.id,
    title: req.body.title,
    description: req.body.description,
    imagePath: imagePath,
    startDate: req.body.startDate,
    endDate: req.body.endDate,
  });
  News.updateOne({_id: req.params.id}, news).then(result => {
    res.status(200).json(
    {message: "Update succsessful!"}
    );
  });
}
```

3.11. forráskód. PUT/news végpont

```
{
  "message": "Update succsessful!",
  "news": [
  {
    "_id": "616ece72537ef3531fde6d47",
    "title": "New News",
    "description": "Lorem Ipsum is simply dummy text of the
       printing and typesetting industry. Lorem Ipsum has been the
       industry's standard dummy text ever since the 1500s, when an
        unknown printer took a galley of type and scrambled it to
       make a type specimen book. It has survived not only five
       centuries, but also the leap into electronic typesetting,
       remaining essentially unchanged.",
    "imagePath":"http://localhost:3000/images/news/new-news
       -1634651762652.jpg",
    "__v": 0
  }]
}
```

3.12. forráskód. PUT/news JSON

DELETE/news 3.13 végpont:

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

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.

A. függelék

Szimulációs eredmények

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque facilisis in nibh auctor molestie. Donec porta tortor mauris. Cras in lacus in purus ultricies blandit. Proin dolor erat, pulvinar posuere orci ac, eleifend ultrices libero. Donec elementum et elit a ullamcorper. Nunc tincidunt, lorem et consectetur tincidunt, ante sapien scelerisque neque, eu bibendum felis augue non est. Maecenas nibh arcu, ultrices et libero id, egestas tempus mauris. Etiam iaculis dui nec augue venenatis, fermentum posuere justo congue. Nullam sit amet porttitor sem, at porttitor augue. Proin bibendum justo at ornare efficitur. Donec tempor turpis ligula, vitae viverra felis finibus eu. Curabitur sed libero ac urna condimentum gravida. Donec tincidunt neque sit amet neque luctus auctor vel eget tortor. Integer dignissim, urna ut lobortis volutpat, justo nunc convallis diam, sit amet vulputate erat eros eu velit. Mauris porttitor dictum ante, commodo facilisis ex suscipit sed.

Sed egestas dapibus nisl, vitae fringilla justo. Donec eget condimentum lectus, molestie mattis nunc. Nulla ac faucibus dui. Nullam a congue erat. Ut accumsan sed sapien quis porttitor. Ut pellentesque, est ac posuere pulvinar, tortor mauris fermentum nulla, sit amet fringilla sapien sapien quis velit. Integer accumsan placerat lorem, eu aliquam urna consectetur eget. In ligula orci, dignissim sed consequat ac, porta at metus. Phasellus ipsum tellus, molestie ut lacus tempus, rutrum convallis elit. Suspendisse arcu orci, luctus vitae ultricies quis, bibendum sed elit. Vivamus at sem maximus leo placerat gravida semper vel mi. Etiam hendrerit sed massa ut lacinia. Morbi varius libero odio, sit amet auctor nunc interdum sit amet.

Aenean non mauris accumsan, rutrum nisi non, porttitor enim. Maecenas vel tortor ex. Proin vulputate tellus luctus egestas fermentum. In nec lobortis risus,

sit amet tincidunt purus. Nam id turpis venenatis, vehicula nisl sed, ultricies nibh. Suspendisse in libero nec nisi tempor vestibulum. Integer eu dui congue enim venenatis lobortis. Donec sed elementum nunc. Nulla facilisi. Maecenas cursus id lorem et finibus. Sed fermentum molestie erat, nec tempor lorem facilisis cursus. In vel nulla id orci fringilla facilisis. Cras non bibendum odio, ac vestibulum ex. Donec turpis urna, tincidunt ut mi eu, finibus facilisis lorem. Praesent posuere nisl nec dui accumsan, sed interdum odio malesuada.

Ábrák jegyzéke

Táblázatok jegyzéke

Forráskódjegyzék