**FEJLESZTŐI DOKUMENTÁCIÓ**

**ELITECARPEST WEBSHOP WEBALKALMAZÁS**

**2024. április 12.**

**A WEBALKALMAZÁS CÉLJA**

Az EliteCarpets webshop szőnyegek árusításával foglalkozó cég számára jött létre. Az alkalmazás ötletét az adta, hogy a készítők környezetében van olyan, aki webshopot szeretne indítani, de még nem rendelkezik online felülettel. A projekt elsődleges célja egy kísérleti platform, ahol olyan funkciókat lehet kikísérletezni és tesztelni, amik a későbbiekben értékes elemei lehetnek az aktív projektnek, mert adaptálható és jó alapanyagot jelent a következő projekt számára.

**PROJEKT LEÍRÁSA**

A projekt egy modern webshop alkalmazás, melyet Angular keretrendszerrel fejlesztettünk. Az alkalmazás a Firebase felhőalapú adatbázist és autentikációs rendszert használja az adatok tárolására és felhasználói kezelésre. A fejlesztés során a Visual Studio Code fejlesztői környezetet és a Git verziókezelőt alkalmaztuk.

**FELHASZNÁLT TECCHNOLÓGIÁK**

A projekt frontend és backend részekre van felosztva, ahol a frontend az felhasználó/vásárló által látható webes felületet jelenti, míg a backend az adatok tárolását, kezelését és az üzleti logikát végzi.

Frontend

* Angular CLI 16.2.7. keretrendszer
* HTML, CSS
* ng-bootstrap UI komponensek
* ngx-toaster könyvtár
* date-fns könyvtár

Backend:

Firebase felhőszolgáltatások:

* Firesbase Realtime Database adatbázis
* Firebase Authentichation
* Firebase Cloud Functions

Fejlesztői környezet:

* Visual Studio Code

Egyaránt elérhető macOS-ra,, Linux-ra és Windows-ra is. Rendelkezik egy beépített támogatással mind JavaScripthez, TypeScript-hez és Node.js-hez is. Előnyei a kódformázás, könnyű navigáció, kiegészítő lehetőségek.

* Firebase SDK

Firebase szolgáltatások integrálása, például a Firebase Authentication, Realtime Database (adatok tárolására és lekérdezésére)

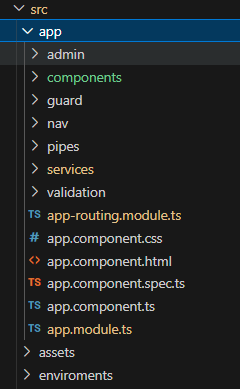
* Git Verziókezelés

A verziókezelő volt segítségünkre a fejlesztés során az együttműködésben, valamint a kód verzióinak kezelésében, nyomon követésében és dokumentálásában.

**ÁLTALÁNOS MŰKÖDÉS**

Az Angular Routing modullal definiált útvonalak segítségével navigálunk az egyes oldalak között. Az ng-bootstrap komponenseket használjuk az alkalmazás felhasználói felületének kialakítására és interakcióinak kezelésére. A Firebase Auth modullal végzünk felhasználói autentikációt és jogosultságkezelést. Az adatkezelését Firebase a backend rész végzi, biztosítva ezzel az adatok integritását és biztonságát. A Firebase Cloud Functions-t használjuk a HTTP-végpontok implementálásához. Az **admin felület az alábbi bejelentkezési adatokkal érhető el**: E-mail cím: [admin@admin.com](mailto:admin@admin.com), Jelszó: Admin1234

**KÖNYVTÁRSZERKEZET**

* Komponensek, funkciók mappákba rendezve funkcionalitás alapján
* Services az implementálásához
* Pipe-ok az adatok formázásához
* Segédfunkciók a közös műveletek elvégzéséhez
* A Firebase konfigurációt és szolgáltatásokat külön modulokba szervezzük az átláthatóság érdekében

/src

/app

/admin

/components

/guards

/nav

/pipes

/services

app.module.ts

app-routing.module.ts

/assets

/environments

A következő **vizuális komponensek** lettek létrehozva:

* app.component
* nav.component – menüsor, navigációs mező
* home.component – a termékek megjelenítésére szolgáló componens
* search.component – keresőmező a nav componensben, a termékek név alapú kereséséhez
* cart.component – a kosár tartalmának vizuális megjelenítésére és változtatására
* checkout.compoent – a vásárlás véglegesítésére létrehozott componens, számlázási, fizetési, szállítási adatok megadására
* footer.component – a webshop footer része, melyről a kötelező és hasznos információk, illetve a kapcsolatfelvétel érhető el
* packet-point-info,component – a checkout componensben használt, csomagpontra történő szállítás esetén a csomagpont információit tartalmazza
* product.component – egy kiválasztott termék megjelenítésére használt componens, mely véleményezési lehetőséget is biztosít a felhasználó számára
* terms-conditions.component – az általános szerződési feltételek tárolására
* user-login.component - beléptető felület
* user-sign-up.component – regisztrációs felület
* forgotpassword.component- Elfelejtett jelszó igénylésének űrlapja
* verifyemail.component – Tájékoztató felület az elfelejtett jelszó linkjének elküldéséről
* profil.component – felhasználók adatainak megjelenítése, kezelése
* profil-data.component – a profil felület alcomponense, a felhasználó adatainak kezelésére
* admin.component
* order-list.component – az admin felület alcomponense, a rendelések kezelésére
* rating-list.component – az admin felület alcomponense, a termékvélemények kezelésére
* product-list.component – az admin felület alcomponense, a termékek kezelésére
* user-list.component – az admin felület alcomponense a felhazsnálók kezelésére

A következő **nem vizuális komponensek** lettek beépítve:

* services
  + auth.service – felhasználók azonosítása
  + base.service – a termékek adatbázisból történő elérését segíti
  + cart.service – a kosártartalmat kezeli
  + oder.service – az adatbázis orders végpontjának elérésére
  + packet-point.service – a csomagpont információinak elérését teszi lehetővé
  + rating.service – az adatázis velemenyek végpontjának elérésére
  + search.service – a keresőmezőbe beírt adatokat szolgáltatja
  + user.service – a bejelentkezett user adatait szolgáltatja az adatbázisból, illetve regisztrációnál menti a user adatokat a users táblába
* guard – útvonalak védelme, ellenőrzik, hogy a felhasználó miként van vagy nincs bejelentkezve mielőtt hozzáférne bizonyos oldalakhoz
* login-guard
* not-logged.guard
* sadmin.guard
* pipes:
  + key.pipe – kulcs alapú filter termékekhez
  + rating.pipe – product ID alapú filter a véleményekhez
  + serch.pipe – név alapú filter a termékekhez

**KOMPONENSEK**

**APP COMPONENT**

Az AppComponent a webalkalmazás gyökérkomponenseként szolgál.

**<app-nav>:** A komponens a navigációs sávot reprezentálja, amely linkeket tartalmaz az alkalmazás különböző részeihez, funkcióihoz, illetve egy kereső mezőt a termékek név szerinti kereséséhez

**#mainContainer:** Ez a div elem az alkalmazás tartalmának fő konténereként szolgál.

**<router-outlet>:** Helyörző, mely dinamikusan betöltött komponensek számára, amelyeket az Angular Router kezel. Az alkalmazás különböző útvonalai a saját nézeteiket jelenítik meg a router-outlet-en belül.

**<app-footer>:** Ez a komponens az alkalmazás lábléce, amely a kötelező jogi információkat és egyéb releváns részleteket tartalmazza

Az AppComponent osztály az alapvető konfigurációt biztosítja a gyökérkomponens számára.

Összességében az AppComponent az alkalmazás alapjaként szolgál, meghatározza az elrendezést, és beépíti a navigációs és lábléc komponenseket. Az egyéni CSS stílusok fokozzák a fő tartalomtároló vizuális megjelenését.

**ADMIN COMPONENT**

Az AdminComponent az adminisztrációs felület központi eleme. Lehetővé teszi a rendszergazdának, hogy kiválassza, melyik adattáblában szeretne dolgozni (felhasználók, termékek, rendelések, vélemények), és a választás alapján az adott táblához tartozó listázó komponenst jeleníti meg. Az **admin felület az alábbi bejelentkezési adatokkal érhető el**: E-mail cím: [admin@admin.com](mailto:admin@admin.com), Jelszó: Admin1234

**#selectorContainer:** Ez a csoport tartalmazza a választóelemeket, **#selectorText:** Kijelző szöveg, amely arra kéri a felhasználót, hogy válasszon táblát, #selector: A legördülő lista segítségével lehet kiválasztani a kívánt táblát. A [(ngModel)] irányított adatkötés biztosítja a kétirányú szinkronizálást a selectorValue komponens tulajdonság és a kiválasztott érték között.

option-bg osztályú option elem: Üres alapértelmezett opció, amelyet a [value]="" értékhez kötünk.

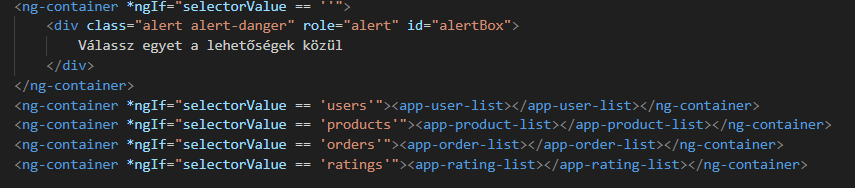
Több option elem: Mindegyik egy-egy alcomponenst (felhasználók, termékek, rendelések, vélemények) reprezentál, a [value] direktiva segítségével az értékhez kötve ('users', 'products', stb.).

ng-container direktívák feltételes megjelenítéshez:

* Ha a selectorValue üres, akkor megjelenik egy riasztás (#alertBox), amely arra kéri a felhasználót, hogy válasszon egy lehetőséget.
* A kiválasztott érték (selectorValue) alapján a megfelelő lista komponens kerül megjelenítésre:
  + app-user-list: Felhasználó lista komponens (selectorValue == 'users')
  + app-product-list: Termék lista komponens (selectorValue == 'products')
  + app-order-list: Rendelés lista komponens (selectorValue == 'orders')
  + app-rating-list: Vélemény lista komponens (selectorValue == 'ratings')

selectorValue: A kiválasztott tábla értékét tároló tulajdonság.

Összességében az AdminComponent egy rugalmas adminisztrációs felületet biztosít, amely lehetővé teszi a rendszergazdának, hogy különböző adattáblák között navigáljon és azokat kezelje.



**ADMIN: ORDER-LIST COMPONENT**

Az app-order-list komponens a webáruház megrendeléseinek kezelését teszi lehetővé. A komponens megjeleníti a megrendelések listáját, melyek szűrhetők rendelés azonosító alapján. Lehetővé teszi a megrendelések módosítását, és törlését.

Funkciók:

* Megrendelések lekérése: A komponens az OrderService segítségével lekéri az összes megrendelést az adatbázisból.
* Megrendelések megjelenítése: A lekérdezett megrendelések listáját jeleníti meg a komponens. A lista tartalmazza a rendelés azonosítót, felhasználó ID-t, rendelés időpontját, rendelési adatokat (címzett, telefonszám, adószám - ha van, szállítási cím), fizetési módot (online vagy utánvét), rendelés összegét, és a megrendelés státuszát.
* Megrendelés keresés: A felhasználó az oldal tetején található mezőbe beírt rendelés azonosító alapján szűrheti a listát.
* Megrendelés módosítása: A felhasználó a lista elemeire kattintva szerkesztheti a megrendelés adatait (címzett adatai, szállítási cím, telefonszám, státusz). A módosítások elvégzése után a "Módosítás" gombra kattintva menti a változtatásokat az adatbázisba a OrderService segítségével.
* Megrendelés törlése: A "Törlés" gombra kattintva törölhető egy rendelés. A törlés előtt a komponens megerősítést kér a felhasználótól. A törlést a OrderService segítségével hajtja végre.
* Értesítés: A komponens a ngx-toastr könyvtár segítségével megjeleníti a felhasználónak a különböző műveletek (pl.: módosítás, törlés) eredményességét.

Metódusok:

* **ngOnInit()**: Az újratölti a rendeléseket az OrderService segítségével, és frissíti a ordersArray tömböt az új rendelésadatokkal.
* **modifyOrder(order: any)**: Egy meglévő rendelés adatait módosítja. Elkészíti a módosítandó rendelés adatait tartalmazó objektumot és meghívja az OrderService writeOrderData metódusát a rendelés frissítésére.
* **deleteOrder(key: string)**: Egy rendelést töröl az adatbázisból az adott kulcs alapján. Meghívja az OrderService deleteOrder metódusát a törlés végrehajtására.
* **toastMsgOutlet(result: boolean, type: string)**: Egy segédmetódus, amely megjeleníti a műveletek (módosítás vagy törlés) sikerességét vagy sikertelenségét a ToastrService segítségével. A metódus különböző üzeneteket jelenít meg aszerint, hogy a művelet sikerült-e vagy sem.

**ADMIN: PRODUCT-LIST COMPONENT**

Az app-product-list komponens a szőnyegek webáruház termékeinek kezelését teszi lehetővé. A komponens megjelenít egy terméklistát, mely szűrhető kategória és terméknév alapján. Lehetővé teszi új termékek felvitelét, meglévő termékek módosítását és törlését.

Funkciók:

* Termékek lekérése: A komponens a BaseService segítségével lekéri az összes szőnyeget az alkalmazás inicializálásakor.
* Termék Lista Megjelenítése: A lekérdezett termékeket kategóriák alapján csoportosítva jeleníti meg a komponens. A felhasználó választhat a kategóriák között, hogy az adott kategóriához tartozó termékeket lássa.
* Új Termék Felvétele: A felhasználó az űrlap kitöltésével vehet fel új terméket. Az űrlap mezői a termék nevét, árát, darabszámot, láthatóságot, anyagát, mintázatát, méretét, színét, szálmagasságát, vastagságát és stílusát kérdezik le. Az űrlap adataiból a komponens egy új terméket hoz létre, majd a BaseService segítségével elmenti az adatbázisba.
* Termék Módosítása: A felhasználó a listából választhat ki egy terméket, melynek adatait ezután módosíthatja az űrlapon. A módosított adatok elküldésével a komponens a BaseService segítségével frissíti a termék adatait az adatbázisban.
* Termék Törlése: A felhasználó a "Törlés" gombra kattintva törölhet egy terméket. A törlés előtt a komponens megerősítést kér a felhasználótól. A törlést a BaseService segítségével hajtja végre.
* Készlet Mennyiség Módosítása: A felhasználó a termék mellett található mezőbe beírt értékkel növelheti a kiválasztott termék készlet mennyiségét. A módosítást a "Hozzáadás" gomb megnyomásával véglegesítheti.
* Értesítés: A komponens a ngx-toastr könyvtár segítségével megjeleníti a felhasználónak a különböző műveletek (pl.: felvitel, módosítás, törlés) eredményességét.

Metódusok:

* **ngOnInit()**: Újratölti a termékek listáját a BaseService szolgáltatás segítségével.
* **setShowedProducts()**: Beállítja a showedProducts tömböt az aktív kategóriához tartozó termékekkel.
* **setChoosenProduct(product: any)**: Beállítja a kiválasztott terméket a choosenProduct objektumba.
* **changeActiveCategory(i: number)**: Megváltoztatja az aktív kategóriát, és frissíti a megjelenített termékek listáját.
* **setNewProdToDefault()**: Visszaállítja az új termék adatait az alapértelmezett üres értékekre.
* **qtyPlus()**: Növeli a kiválasztott termék mennyiségét az adatbázisban, és frissíti a megjelenített adatokat.
* **postProduct()**: Új terméket tölt fel az adatbázisba a megadott kategóriába.
* **deleteProduct()**: Törli a kiválasztott terméket az adatbázisból.
* **modifyProduct()**: Módosítja a kiválasztott termék adatait az adatbázisban.
* **toastMsgOutlet(result: boolean, type: string)**: megjelenít egy toast üzenetet a ToastrService segítségével, tájékoztatja a felhasználót a művelet sikerességéről vagy sikertelenségéről

**ADMIN: RATING-LIST COMPONENT**

Az app-rating-list komponens megjeleníti a felhasználók által beküldött termékértékeléseket. Az értékelések egy táblázatban jelenik meg, mely a következő oszlopokat tartalmazza:

* ID: Az értékelés azonosítója.
* Felhasználó ID: Az értékelést beküldő felhasználó azonosítója.
* Termék ID: Az értékelt termék azonosítója.
* Pontszám: A felhasználó által adott pontszám (1-5).
* Értékelés Szövege: A felhasználó által írt értékelés szövege.
* Cenzúrázás: Minden egyes értékelésnél van egy "Cenzúrázás" gomb.

Funkciók, metódusok:

* Értékelések lekérése: A komponens a RatingService segítségével lekéri az összes értékelést az alkalmazás inicializálásakor. A lekérdezett adatokat a ratingsArray tömbben tárolja.
* Értékelés megjelenítése: A komponens a \*ngFor direktiva segítségével iterál a ratingsArray tömbön, és minden egyes értékeléshez létrehoz egy sort a táblázatban. A sorok a fenti oszlopokban megjelenítik az értékelés adatait.
* Értékelés Cenzúrázása: A "Cenzúrázás" gombra kattintva a komponens meghívja a **modifyRating(rating:any)** metódust. Ez a metódus módosítja az értékelés szövegét egy előre definiált cenzúrázott szövegre ("CENZÚRÁZOTT VÉLEMÉNY: A vélemény kívül esett a jó ízlés határain"). A módosított értékelést a RatingService segítségével menti vissza az adatbázisba.
* **toastMsg(result: boolean)**: Értesítés. A modifyRating metódus eredménye alapján a komponens megjelenít egy értesítést a felhasználónak ("Siker" vagy "Hiba") a ngx-toastr könyvtár segítségével.

**ADMIN: USER-LIST COMPONENT**

Az app-user-list komponens a webáruház felhasználóinak kezelését teszi lehetővé. A komponens megjeleníti a felhasználók listáját, melyek szűrhetők felhasználó azonosító alapján. Lehetővé teszi a felhasználói adatok módosítását és törlését.

Funkciók:

* Felhasználók lekérése: A komponens az ProfilService segítségével lekéri az összes felhasználót az adatbázisból.
* Felhasználók megjelenítése: A lekérdezett felhasználók listáját jeleníti meg a komponens. A lista tartalmazza a felhasználó ID-t, felhasználói adatokat (e-mail cím, név, cím, telefonszám).
* Felhasználó keresés: A felhasználó az oldal tetején található mezőbe beírt rendelés azonosító alapján szűrheti a listát.
* Felhasználó módosítása: A felhasználó a lista mezőibe kattintva szerkesztheti az adatokat (név, cím, telefonszám). A módosítások elvégzése után a "Módosítás" gombra kattintva menti a változtatásokat az adatbázisba a ProfilService segítségével.
* Felhasználó törlése: A "Törlés" gombra kattintva törölhető egy felhasználó. A törlés előtt a komponens megerősítést kér a felhasználótól. A törlést a ProfilService segítségével hajtja végre.

Metódusok:

* **modifyUser(userData: any, key: string)**: Egy adott felhasználó adatait módosítja. A userData objektumból kiválasztott adatokat egy új body objektumba gyűjti össze, és ezt a profilService.writeUserData() metódus segítségével frissíti az adatbázisban.
* **deleteUser(userData: any)**: Törli a megadott felhasználót az adatbázisból. A metódus először megerősítést kér a felhasználótól egy confirm() hívással, és ha a felhasználó megerősíti a törlést, akkor meghívja a profilService.deleteUser() metódust a felhasználó kulcsával.

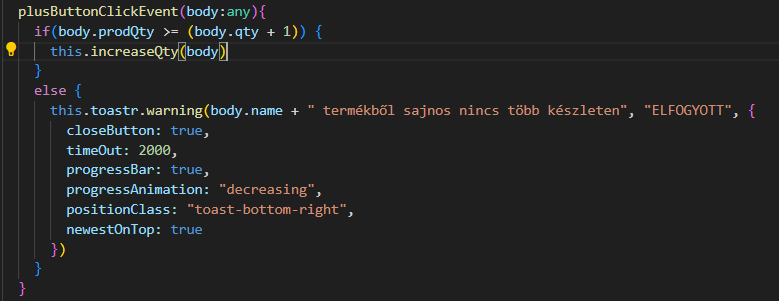
**CART COMPONENT**

Az app-cart komponens megjeleníti a felhasználó kosarát egy Bootstrap offcanvas modal ablakban, a teljes kosárértékkel és az összes darabszámmal. A kosár tartalma a CartService szolgáltatásból származik. A komponens lehetővé teszi a termékek darabszámának módosítását, valamint a termékek eltávolítását a kosárból.

Funkciók:

* Kosár tartalmának megjelenítése: A komponens a \*ngFor direktíva segítségével iterál a cart tömbön, és megjeleníti a kosárba tett termékek adatainak egy részét (név, egységár, darabszám, összes ár), továbbá a termékhez tartozó képet.
* Kosár összesítő: A komponens lekéri a CartService segítségével a kosár teljes értékét és az összes darabszámot.
* Visszalépés a termékekhez: A "Vissza a termékekhez" gomb bezárja a kosár modalt, és visszaviszi a felhasználót a termékek oldalára.
* Tovább a véglegesítéshez: A "Tovább a véglegesítéshez" gomb a pénztár oldalra navigál, de csak akkor aktív, ha van legalább egy termék a kosárban.

Metódusok:

* **removeProduct(body: any)**: Töröl egy terméket a kosárból. A CartService.removeProduct() metódust hívja a termék eltávolítására, majd a ToastrService segítségével megjelenít egy hibaüzenetet az eltávolításról.
* **plusButtonClickEvent(body: any)**: Kezeli a "+" gomb eseményét, amikor a felhasználó növelni szeretné a termék mennyiségét a kosárban. Ellenőrzi, hogy a készlet elegendő-e a mennyiség növeléséhez, és vagy meghívja az increaseQty() metódust, vagy figyelmeztető üzenetet jelenít meg, ha nincs több termék készleten.
* **increaseQty(body: any)**: Növeli a megadott termék mennyiségét a kosárban eggyel, frissíti a kosárban lévő mennyiséget a CartService.updateProductQty() metódus segítségével, majd sikerüzenetet jelenít meg.
* **minusButtonClickEvent(body: any)**: Kezeli a "-" gomb eseményét, amikor a felhasználó csökkenteni szeretné a termék mennyiségét a kosárban. Ha a mennyiség nagyobb mint 1, meghívja a decreaseQty() metódust, ha 1, akkor a terméket törli a kosárból a removeProduct() metódus segítségével.
* **decreaseQty(body: any)**: Csökkenti a megadott termék mennyiségét a kosárban eggyel, frissíti a kosárban lévő mennyiséget a CartService.updateProductQty() metódus segítségével, majd figyelmeztető üzenetet jelenít meg.

**CHECKOUT COMPONENT**

Ez a komponens a felhasználói felületen a pénztár funkciót valósítja meg, ahol a vásárlók a kosár tartalmát megrendelhetik.

A komponens számos tulajdonságot definiál az adatok tárolására:

* cart: A kosár tartalma.
* totalQty: A kosárban lévő termékek összes mennyisége.
* total: A kosár összesítő ára.
* user: A felhasználó adatai (opcionálisan kitöltendő).
* data: A szállítási adatokat tartalmazó objektum.
* entity: A számlázási entitás típusa ("person" vagy "company").
* deliveryOpt: A szállítási mód ("home" vagy "packetpoint").
* paymentOpt: A fizetési mód ("before" vagy "after").
* termsValue: Kijelöli, hogy a felhasználó elfogadta-e az ÁSZF-et.
* packetPointList: A csomagpontok adatai tömbje.
* selectedPacketPointCity: A kiválasztott csomagpont városa.
* selectedPacketPoint: A kiválasztott csomagpont adatai.
* cities: A legördülő menüben megjelenő városok listája.
* errorShow: Kijelöli, hogy hibaüzenet jelenjen-e meg.
* errorMsg: A megjelenítendő hibaüzenet szövege.

Metódusok:

* **selectPacketPointCity(newValue: any)**: Beállítja az aktuálisan kiválasztott csomagpont városát.
* **selectPacketPoint(newValue: any)**: Kiválasztja a konkrét csomagpontot a felhasználó által megadott városban.
* **qtyCheck()**: Ellenőrzi, hogy a kosárban lévő összes termék mennyisége nem haladja-e meg a raktáron lévő mennyiséget. Ha minden rendben, igazzal tér vissza, egyébként hamissal.
* **removeUnusedProp()**: Távolítja el a felesleges tulajdonságokat a kosárból, amikor a rendelést előkészíti a felhasználó.
* **orderProducts()**: A rendelési folyamat végrehajtását kezeli. Először is ellenőrzi a termékmennyiségeket, majd létrehozza a rendeléshez szükséges test adatokat. Ellenőrzi, hogy minden szükséges mező kitöltött-e, és ha igen, elküldi a rendelést az OrderService-en keresztül. Sikeres rendelés esetén üzenetet jelenít meg és üríti a kosarat.
* **callToasts(result: boolean, toastBodyTxt: string, toastHeaderTxt: string)**: Toast üzenetek megjelenítéséért felelős, amelyek tájékoztatják a felhasználót a műveletek eredményéről. A sikeres és sikertelen műveletekhez különböző üzeneteket jelenít meg.

**FOOTER COMPONENT**

Ez a dokumentáció az EliteCarpets webáruház láblécében található linkekhez kapcsolódó modális ablakok tartalmát foglalja magában. A footer célja, hogy a felhasználók könnyen, e-mail útján kapcsolatba léphessenek a kereskedővel, illetve könnyen elérhetővé tegye számukra az alábbi információkat:

* Általános Szerződési Feltételek (ÁSZF)
* Gyakran Ismételt Kérdések (GYIK)
* Adatkezelési Tájékoztató
* Szállítási Információk
* Fizetési Információk
* Kapcsolat

A modális ablakok tartalma az EliteCarpets webáruház aktuális verziójának megfelelően van dokumentálva.

A dokumentáció frissítése a webáruház frissítéseivel összhangban történik.

**SEARCH COMPONENT**

A Search komponens egy egyszerű keresőmezőt biztosít a weboldal nav komponensében. A felhasználó által beírt kifejezést továbbítja a SearchService szolgáltatásnak.

expression: String típusú tulajdonság, amely a keresőmezőbe beírt aktuális szöveget tárolja.

Metódusok:

* **onKeyUp(value: string)**: A metódus a keresőmező keyup eseményére fut le, és a beírt szöveget továbbítja a SearchService-nek. Ez a metódus frissíti a expression változót a megadott értékkel, amely a felhasználó által beírt szöveget tartalmazza, és meghívja a search.setSearchTerm(value) metódust a SearchService-en keresztül. Ez a metódus felelős azért, hogy továbbítsa a keresési kifejezést a szolgáltatásnak, amely kezeli a keresési kifejezéseket az alkalmazásban.

**TERMS-CONDITIONS COMPONENT**

A Terms-conditions egy egyszerű modális ablakot biztosít, melyben a felhasználó megtalálja az általános szerződési feltételeket.

**PACKET-POINT-INFO COMPONENT**

A Csomagpont Infó komponens egy moduláris elem, amely egy adott csomagpont adatait jeleníti meg. A komponens bemenete egy JSON objektum, amely a csomagpont adatait tartalmazza.

A Csomagpont Infó komponens használatához az alábbi lépéseket kell követni:

* Importáld a komponenst a modulba, ahol használni szeretnéd
* Add hozzá a komponenst a sablonodhoz: <app-packet-point-info [data]="packetpointData"></app-packet-point-info>
* Határozd meg a packetpointData változót, amely a bemeneti JSON objektumot fogja tartalmazni.

A komponens a következőket jeleníti meg:

* A csomagpont neve
* A csomagpont üzemeltetőjének azonosítója
* A csomagpont címe
* A csomagpont megtalálásához szükséges információk
* Link a Google Térképhez, amely a csomagpontot mutatja
* A csomagpont nyitvatartási ideje

**FORGOT-PASSWORD COMPONENT, VERIFY-EMAIL COMPONENT**

Az elfelejtett jelszavak kezelése, amelyet az alkalmazásoknak kezelniük kell, hogy segítsék a felhasználókat a hozzáférésük visszaszerzésében anélkül, hogy új fiókot kellene létrehozniuk vagy a támogatási szolgálattal kellene kapcsolatba lépniük. A Firebase Authentication szolgáltatása automatizálja ezt a folyamatot, így a fejlesztőknek nem kell manuálisan kezelniük a jelszó-visszaállítási e-mailek küldését és a jelszavak biztonságos tárolását.

Funkciók:

* Megjelenítés: A komponens egy űrlapot jelenít meg, ahol a felhasználó beírhatja az e-mail címét majd a Küldés gombra kattintania.
* Adatok Beküldése: Az űrlap beküldésekor a komponens a AngularFireAuth szolgáltatás sendPasswordResetEmail metódusát hívja meg, ami elküldi a jelszó-visszaállítási e-mailt a megadott címre.
* Visszajelzés a Felhasználónak A felhasználó visszajelzést kap arról, hogy a jelszó-visszaállítási linket elküldték (vagy ha hiba történt).

Metódusok:

* **sendlink()**: Ez a metódus meghívja az AuthService által biztosított forgotpassword() metódust az aktuálisan megadott e-mail címmel. Ez a metódus felelős azért, hogy a felhasználónak jelszó-helyreállítási linket küldjön az adott e-mail címre. A művelet végrehajtása után az e-mail mezőt üresre állítja, így felkészítve a komponenst egy újabb lehetséges műveletre.

**HOME COMPONENT**

Az app-home komponens a weboldal főoldalát jeleníti meg, mely két fő részből áll:

* Kategória Választó: A bal oldali oszlopban a kategória választó található. A kategória választó felsorolja a különböző szőnyeg kategóriákat (pl.: Padlószőnyeg, Szőnyeg, Futószőnyeg). A felhasználó egy kategóriára kattintva szűrheti a megjelenített termékeket. A kategóriaválasztó továbbá végig követi a felhasználó útját az oldal tetejétől az oldal aljáig és lehetőséget biztosít utolsó gombjával az oldal tetejére történő gyors visszaugrásra.
* Termékek és Szűrők: A jobb oldali oszlop a termékeket és a szűrőket jeleníti meg. A szűrők segítségével a felhasználók az anyag, mintazat, méret, szín, szálmagasság, vastagság és stílus alapján szűrhetik a termékeket. A szűrők felett egy gomb található a szűrők törléséhez. A termékek kártyákon jelennek meg, melyeken a termék képe, neve, ára, valamint opcionálisan a színe és mérete is látható. A felhasználó a termékkártyára kattintva megnyithat egy modált, mely a kiválasztott termék részletes adatait jeleníti meg (app-product komponens). A kosárba gombbal a felhasználó hozzáadhatja a kiválasztott terméket a kosárhoz.

Funkciók:

* Kategória váltás: A felhasználó a kategória választó segítségével válthat a különböző kategóriák között. A kategória váltáskor a komponens frissíti a megjelenített termékeket.
* Termék szűrés: A felhasználók a szűrők segítségével szűrhetik a megjelenített termékeket. A szűrők az összes termék tulajdonságai alapján működnek (anyag, mintazat, méret, szín, stb.).
* Termékek megjelenítése: A komponens a getProducts metódus segítségével lekéri a termékek adatait a BaseService-ből. A lekérdezett adatokat a komponens a products tömbben tárolja. A szűrés és a kategória váltás után a komponens a showedProducts tömböt frissíti a megjelenítendő termékekkel.
* Kosárba Rakás: A "Kosárba" gombra kattintva a komponens a cartService.addProduct metódusát hívja meg, hogy hozzáadja a kiválasztott terméket a kosárhoz. A metódus visszatérési értékétől függően a komponens megjelenít egy értesítést a felhasználónak ("Siker" vagy "Hiba").
* Termék modal: A termék kártyára kattintva a komponens megnyit egy modált (app-product), mely a kiválasztott termék részletes adatait jeleníti meg.
* Visszalépés az oldal tetejére: A kategória választó alatt található egy gomb, melyre kattintva a felhasználó visszalapozhat az oldal tetejére.

Metódusok:

* **ngOnInit()**: Angular életciklus metódus, amely a komponens inicializálásakor fut le. Itt történik a felhasználói bejelentkezés állapotának ellenőrzése és az adatok betöltése.
* **addToCart(product: any)**: Hozzáad egy terméket a bevásárlókosárhoz a CartService segítségével, és megjelenít egy toast üzenetet az eredményről.
* **changeActiveCategory(i: number)**: Megváltoztatja az aktív termékkategóriát és újraindítja a terméklista megjelenítését a választott kategória szerint.
* **setShowedProducts()**: Beállítja a megjelenítendő termékek listáját az aktív kategóriából, figyelembe véve a termék kulcsait.
* **setChoosenProduct(product: any)**: Kiválasztja és beállítja a részletezésre szánt terméket.
* **getChossenProduct()**: Visszaadja a kiválasztott terméket.
* **filterProducts()**: A termékek szűrését végzi a beállított szűrők alapján. Csak azokat a termékeket jeleníti meg, amelyek megfelelnek az összes aktív szűrőfeltételnek.
* **changeFilter(newValue: any)**: Frissíti az aktív szűrők állapotát és újra elvégzi a termékek szűrését.
* **clearFilters()**: Törli az összes aktív szűrő beállítást, és újra beállítja a megjelenítendő termékek listáját.
* **showResultToastMsg(result: boolean, body: any)**: Eredmény alapján toast üzenetet jelenít meg, amely tájékoztatja a felhasználót a kosárba helyezés sikerességéről vagy sikertelenségéről.
* **scrollToTop()**: Gördülékenyen görgeti az oldalt a tetejére, amikor a felhasználó ezt kéri (például a szűrők feletti részhez).

**NAV COMPONENT**

Az app-nav komponens a weboldal tetején található navigációs sávot jeleníti meg. A navigációs sáv tartalmazza a weboldal logóját, egy keresőmezőt, valamint linkeket a főbb oldalakhoz. A megjelenő linkek a felhasználói bejelentkezési állapottól függően változnak.

Funkciók:

* Navigációs linkek: A komponens linkeket jelenít meg a kezdőlaphoz (/home), a kosárhoz (#offcanvasCart), a bejelentkezéshez (/signin), a regisztrációhoz (/signup), a felhasználói profilhoz (/profil), és a felhasználók kezeléséhez (/users - adminisztrátorok számára).
* Bejelentkezés állapota: A komponens a AuthService segítségével határozza meg, hogy a felhasználó be van-e jelentkezve. A bejelentkezett felhasználók számára a profiloldalra és a kijelentkezésre vonatkozó linkek láthatók.
* Adminisztrátor jogosultság: A komponens az AuthService segítségével ellenőrzi, hogy a bejelentkezett felhasználó rendelkezik-e adminisztrátor jogosultsággal. Csak az adminisztrátorok láthatják a felhasználók kezelése linket. Későbbiekben kerül fejlsztésre, jelenleg az admin felület a localhost:4200/admin útvonalon érhető el.
* Kosár termék darabszám: A komponens a CartService segítségével lekéri a kosárban lévő termékek összesített darabszámát, és megjeleníti a kosár ikon mellett.
* Keresés: A komponens az <app-search> komponenst ágyazza be a keresőmező megjelenítéséhez, mely a termékek név szerinti keresését szolgálja.

**PRODUCT COMPONENT**

A ProductComponent egy egyéni termék részleteit jeleníti meg egy modális ablakban és lehetőséget biztosít a bejelentkezett felhasználó számára, hogy értékelje azt.

Bemenő tulajdonság (@Input)

* data: any - Ez egy bemeneti tulajdonság, amely a termék adatait tartalmazó objektumot kapja meg.

Metódusok:

* **sumScores()** - Kiszámítja a termék átlagos értékelését a ratingsArray alapján. Végigiterál az értékeléseken, és összeadja a pontszám értékeket. Az összesített pontszámot elosztja az értékelések számával, így megkapva az átlagot.
* **addToCart(product: any)** - A termék kosárba helyezéséért felelős metódus. Létrehoz egy objektumot (body), amely a termék kosárba kerüléshez szükséges adatokat tartalmazza (kulcs, ár, mennyiség, név, kategória, készletin lévő mennyiség). A cartService.addProduct(body) metódust meghívja a termék kosárba helyezésére. A toastMsgOutlet metódust meghívja a művelet eredményének felhasználónak történő visszajelzésére.
* **postRating()** – lehetőséget biztosít a felhasználó számára, hogy a termékről véleményt írjon, amit eltárol az adatbázisban

**PROFIL COMPONENT**

A ProfilComponent a profil felület központi eleme. Lehetővé teszi a felhasználóknak, hogy kiválasszák, melyik adattáblában szeretne dolgozni (felhasználó adatai, rendelései, véleményei), és a választás alapján az adott táblához tartozó listázó komponenst jeleníti meg.

#selectorContainer: Ez a csoport tartalmazza a választóelemeket.

#selectorText: Kijelző szöveg, amely arra kéri a felhasználót, hogy válasszon táblát.

#selector: A legördülő lista segítségével lehet kiválasztani a kívánt táblát. A [(ngModel)] irányított adatkötés biztosítja a kétirányú szinkronizálást a selectorValue komponens tulajdonság és a kiválasztott érték között.

option-bg osztályú option elem: Üres alapértelmezett opció, amelyet a [value]="" értékhez kötünk.

ng-container direktívák feltételes megjelenítéshez:

* Ha a selectorValue üres, akkor megjelenik egy riasztás (#alertBox), amely arra kéri a felhasználót, hogy válasszon egy lehetőséget.
* A kiválasztott érték (selectorValue) alapján a megfelelő lista komponens kerül megjelenítésre:
  + profil-data: Felhasználó adatai komponens
  + my-orders: Rendeléseim komponens
  + myopinin: Véleményeim komponens

selectorValue: A kiválasztott tábla értékét tároló tulajdonság.

**PROFIL: PROFIL-DATA COMPONENT**

A profil-data komponens a felhasználó személyes adatainak kezelését teszi lehetővé. A komponens megjeleníti a felhasználók adatait és lehetővé teszi a felhasználói adatok módosítását.

Funkciók:

* Felhasználó lekérése: A komponens az UserService segítségével lekéri az felhasználó adatait az adatbázisból.
* Adatok megjelenítése: A lekérdezett felhasználók adatok tartalmazzák a e-mail címet, nevet, címet, telefonszámot.
* Felhasználói adatok módosítása: A "Módosítás" gombra kattintva módosíthatóak az adatok kivéve az e-mail cím. A módosítást a ProfilService segítségével hajtja végre.

Metódusok:

* **ngOnInit()**: Az Angular életciklus hookja, amely inicializáláskor fut le. Ez a metódus a ProfilService reload() metódusát hívja meg, ami valószínűleg frissíti vagy újratölti a profil adatokat a szerverről.
* **modifyUser(user: any)**: Ez a metódus felelős a felhasználói adatok módosításáért. Elkészíti a módosítandó felhasználói adatokat tartalmazó objektumot, és meghívja a ProfilService writeUserData() metódusát az adatok frissítésére a UserService által biztosított kulcs segítségével. Sikeres művelet esetén frissíti a felhasználói felületen megjelenítendő üzenetet és annak típusát, hiba esetén pedig hibaüzenetet jelenít meg.

**PROFIL: ORDER-DATA COMPONENT**

Az order-data komponens a felhasználó rendeléseinek kezelését teszi lehetővé. A komponens megjeleníti a felhasználók rendeléseit.

Funkciók:

* Megrendelések lekérése: A komponens az OrderService segítségével lekéri az felhasználó megrendeléseit az adatbázisból.
* Adatok megjelenítése: A lekérdezett megrendelések tartalmazzák a megrendelt terméket, a rendelés összegét, a szállítási és számlázási címet, megrendelőt, telefonszámot és a rendelés státuszát.

Metódusok:

* ngOnInit() metódus feladata az adatok betöltése . Ebben az esetben a ngOnInit() metódus felelős azért, hogy lekérje a bejelentkezett felhasználóhoz tartozó rendeléseket az OrderService segítségével. Itt a orderService.loadMyOrders() metódust hívja meg a felhasználó e-mail címével, mint paraméterrel. A metódus visszatérési értékét (this.myOrders) használja a komponensben a rendelések tárolására, amit aztán felhasználhat a HTML sablonban a rendelések megjelenítésére.

**PROFIL: RATINGS-DATACOMPONENT**

A ratings-data komponens a felhasználó értékeléseinek kezelését teszi lehetővé. A komponens megjeleníti a felhasználók egyes termékekre adott értékeléseit, melyek módosíthatók.

Funkciók:

* Értékelések lekérése: A komponens az OrderService segítségével lekéri az felhasználó megrendeléseit az adatbázisból.
* Adatok megjelenítése: A lekérdezett értékelések tartalmazzák a felhasználó nevét, az értékelés időpontját, az értékelés pontszámát, valamint az értékelés szövegét.

Metódusok:

* + **ngOnInit():** ametódus, akkor hívódik meg, amikor a komponens példányosítva és inicializálva van. Itt történik az értékelések adatok kezdeti betöltése a loadData() metódus segítségével.
  + **loadData():** Ez a metódus felelős azért, hogy betöltse a felhasználóhoz tartozó értékeléseket a RatingService loadMyRatings() metódusának használatával, amely az aktuális felhasználó e-mail címét használja azonosítóként.
  + **delRating(key: string):** Ez a metódus a RatingService deleteRating() metódusát hívja meg, hogy törölje az adott kulccsal rendelkező értékelést. Sikeres törlés esetén egy értesítő üzenetet jelenít meg, és újratölti az értékeléseket a loadData() metódus segítségével.
  + **modify(rating: any):** Ez a metódus frissíti a meglévő értékelést. Először eltávolítja a kulcsot az értékelési objektumból, majd meghívja a RatingService writeRatingData() metódusát, hogy frissítse az adatbázisban. Sikeres módosítás után értesítő üzenet jelenik meg, és újra betölti az értékeléseket.
  + **toastMsgOutlet(result: boolean, type: string):** Ez a metódus kezeli az értesítési üzenetek megjelenítését a toastr szolgáltatás segítségével. Különböző típusú üzeneteket jelenít meg az értékelés módosítása vagy törlése során bekövetkező események alapján. A type paraméter alapján eldönti, hogy módosítási vagy törlési műveletről van-e szó, és a result paraméter alapján sikeres vagy sikertelen volt-e a művelet.

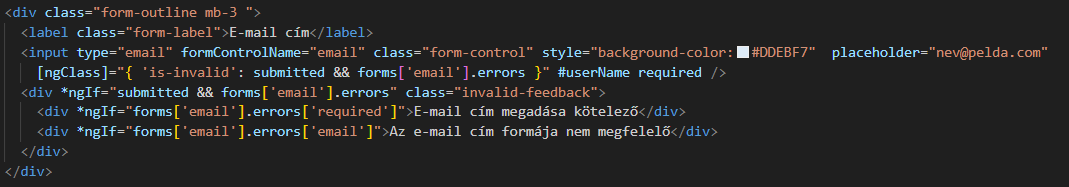
**USER-LOGIN COMPONENT**

Ez a komponens lehetővé teszi a felhasználók számára, hogy bejelentkezzenek az alkalmazásba a Firebase Authentication szolgáltatását használva, biztonságos módon hitelesítve a felhasználókat. A komponens e-mail/jelszó alapú hitelesítést használ.

Megjelenítés: A komponens egy űrlapot jelenít meg, ahol a felhasználóknak meg kell adniuk az e-mail címüket és jelszavukat.

Bejelentkezési adatok feldolgozása: Amikor a felhasználó beküldi az űrlapot, a komponens meghívja a Firebase AngularFireAuth` szolgáltatás `signInWithEmailAndPassword` metódusát, átadva az e-mail címet és a jelszót. A Firebase ellenőrzi a bejelentkezési adatokat, és sikeres hitelesítés esetén visszaad egy felhasználói objektumot. Ha hiba történik (pl. helytelen jelszó), a rendszer hibaüzenetet jelenít meg. Sikeres bejelentkezés esetén a felhasználói interfész frissül. A hitelesítési logikát az AuthService tartalmazza.

A validációs szabályok FormBuilder segítségével lettek beállítva, melyek ellenőrzik a pl. az e-mail érvényes formátumát, illetve a jelszó megfelelőségét, melynek kötelező legalább hat karakternek lenni.



Az alábbi segítő metódus, amely könnyebbé teszi az egyes űrlapmezők vezérlőinek elérését a sablonban.

  get forms(): { [key: string]: AbstractControl } {

    return this.form.controls;

  }

Metódusok:

* **ngOnInit()**: Az Angular életciklus metódusa, amely a komponens inicializálásakor hívódik meg. Itt állítja be az űrlap kezdeti struktúráját és validációs szabályait a FormBuilder segítségével. Az e-mail mezőhöz kötelező megadás és e-mail formátum validációt, míg a jelszó mezőhöz kötelező megadás, minimális (6) és maximális (20) karakterhossz validációt rendel.
* **onSubmit()**: Ez a metódus fut le, amikor a felhasználó beküldi az űrlapot. Ellenőrzi, hogy az űrlap valid-e. Ha igen, akkor további logikát (pl. hitelesítési hívás) lehetne itt implementálni, de jelen esetben csak kiírja a konzolra az űrlap adatát JSON formátumban. Ha az űrlap nem valid, akkor nem tesz semmit.

**USER-SIGN-UP COMPONENT**

felhasználói felületi elem, amely lehetővé teszi az új felhasználók számára, hogy regisztráljanak vagy létrehozzanak egy új fiókot az alkalmazásban a Firebase Authentication segítségével és Firebase Realtime Database szolgáltatásokat használja az új felhasználók adatainak mentéséhez.

Megjelenítés

A komponens egy űrlapot jelenít meg, amelyben a felhasználóknak meg kell adniuk a regisztrációhoz szükséges adatokat (pl. e-mail cím, jelszó, név, cím).

Bejelentkezési Adatok Feldolgozása: Amikor a felhasználó beküldi az űrlapot, a komponens meghívja a Firebase createUserWithEmailAndPassword metódusát, átadva az e-mail címet és a jelszót.

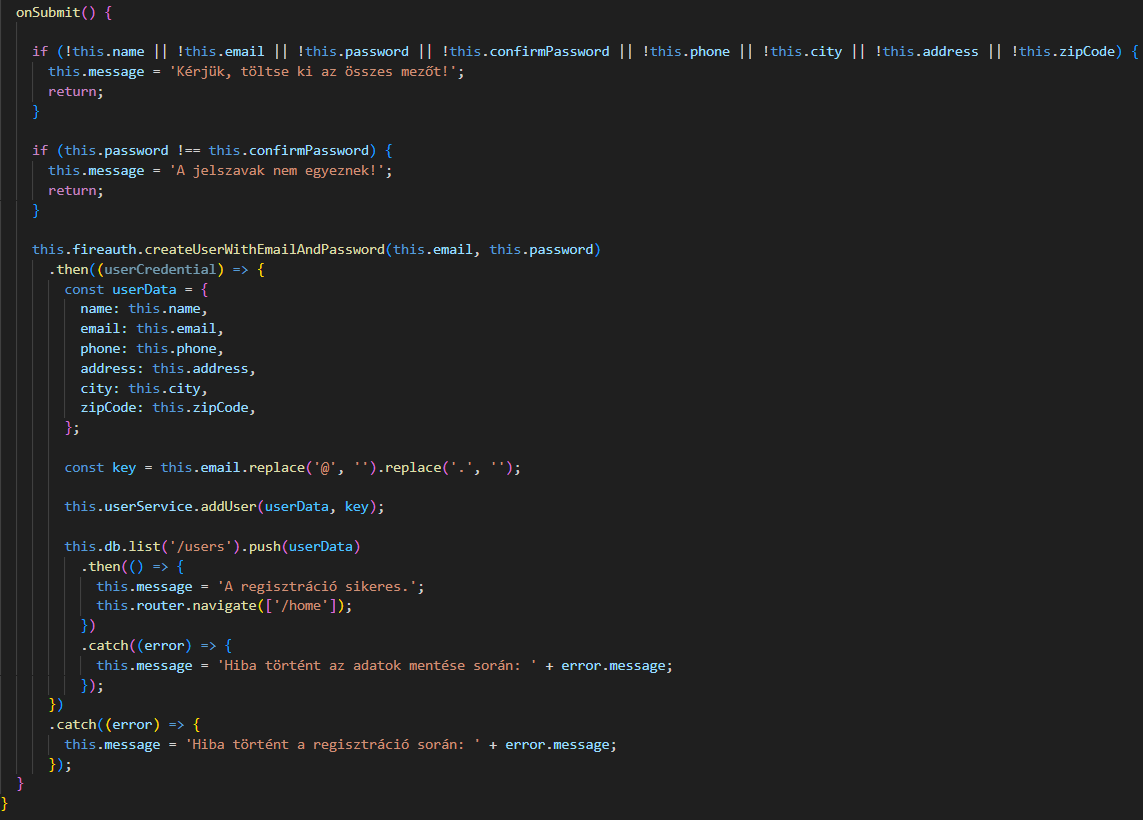
Felhasználói fiók létrehozása: A Firebase ellenőrzi az adatokat, és sikeres regisztráció esetén létrehoz egy új felhasználói fiókot, ahol a felhasználó e-mail címe lesz az azonosító.

Visszajelzés a felhasználónak: A felhasználó visszajelzést kap arról, hogy a regisztráció sikeres volt-e, és hiba esetén hibaüzenet jelenik meg. A message változót, ami tárolja az esetleges sikeres vagy hibás műveletek visszajelzéseit.

Metódusok:

onSubmit():

* Az onSubmit metódus a regisztrációs űrlap beküldését kezeli. Először hozzáadja az új felhasználót a Firebase Authentication rendszeréhez az email és password mezők használatával.
* Sikeres regisztráció esetén létrehoz egy userData objektumot a felhasználói adatokkal, majd egy egyedi kulcsot generál az e-mail címből a Firebase Realtime Database-ben való azonosításra.
* A userService.addUser metódusát meghívva hozzáadja a felhasználói adatokat egy alkalmazásspecifikus tárolóhoz vagy kollekcióhoz.
* A db.list('/users').push(userData) hívással hozzáadja a felhasználói adatokat a Firebase Realtime Database /users útvonalának gyűjteményéhez.
* Sikeres adatmentés esetén frissíti a message változót egy sikerüzenettel. Ha hiba történik, akkor a message változót frissíti a hibainformációval.



**PIPOK**

**KEY PIPE**

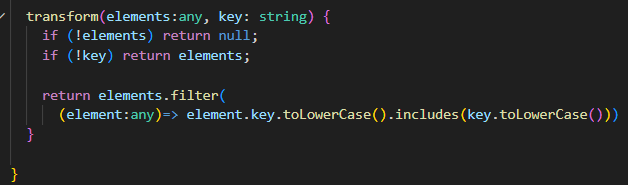
A key pipe egy objektum vagy tömb tulajdonságainak értékeit szűri a megadott kulcsszó alapján. A szűrés a kulcs alapján történik, kisbetű-nagybetű érzéketlen módon.

Bement:  
 elements: Az a szűrendő objektum vagy tömb

key: A szűrés alapjául szolgáló kulcsszó (string)

Kimenet:

A pipe egy új tömböt ad vissza, amely az eredeti objektum vagy tömb azon elemeit tartalmazza, amelyek kulcsa tartalmazza a megadott kulcsszót (kisbetű-nagybetű érzéketlen módon). Ha az elements bemenet null vagy undefined, akkor a pipe szintén null értéket ad vissza.



**RATING PIPE**

A rating pipe egy értékelések tömböt szűr a megadott termék alapján. A pipe csak azokat az értékeléseket adja vissza, amelyek a megadott termékhez tartoznak.

Bemenet:

ratings: Az a szűrendő értékelések tömb

productId: A szűrés alapjául szolgáló termék azonosító

Kimenet:

A pipe egy új tömböt ad vissza, amely az eredeti értékelés tömb azon elemeit tartalmazza, amelyek a megadott termék azonosítójú terméket értékelték. Ha az eredeti bemenet null vagy undefined, akkor a pipe szintén null értéket ad vissza.

**SEARCH PIPE**

A search pipe termék tömböt szűri a megadott kifejezés alapján. A szűrés a termék nevében történik, kisbetű-nagybetű érzéketlen módon.

Bemenet:

products: Az a szűrendő termék tömb

expression: A szűrés alapjául szolgáló kifejezés

Kimenet:

A pipe egy új tömböt ad vissza, amely az eredeti termék tömb azon elemeit tartalmazza, amelyek neve tartalmazza a megadott kifejezést (kisbetű-nagybetű érzéketlen módon). Ha az eredeti bemenet null vagy undefined, akkor a pipe szintén null értéket ad vissza.

**SERVICES**

**AUTH SERVICE**

Az AuthService egy Angular Injectable szolgáltatás, mely a felhasználói hitelesítési folyamatokat kezeli. Ez a szolgáltatás központi helyet biztosít a hitelesítéssel kapcsolatos műveletek kezelésére, és segít az alkalmazás többi részével való integrációban.

Funkciók és metódusok:

Autentikációs metódusok: **(signIn(email: string, password: string), signOut(), forgotpassword(email: string)):**

* Ezek a metódusok kezelik a felhasználói bejelentkezési folyamatot, a kijelentkezést és a jelszó-visszaállítást a Firebase Authentication szolgáltatással.
* Már regisztrált felhasználók hitelesítése, hogy hozzáférjenek az alkalmazás védett részeihez.
* Bejelentkezett felhasználók session-jének vagy tokenének törlése, így megszüntetve a hozzáférésüket a védett tartalmakhoz. signOut kijelentkezési funkció, amely törli a felhasználói munkamenetet.
* Jelszó-visszaállítás forgotpassword: Segítségnyújtás a felhasználóknak elfelejtett jelszavak esetén, jelszó-visszaállító e-mailek küldése.

Felhasználói állapot követése:

* Ellenőrzi, hogy a felhasználó jelenleg be van-e jelentkezve, és esetlegesen hozzáfér a felhasználói profil adatokhoz.
* **getLoggedUser()**: Lekéri a jelenleg bejelentkezett felhasználó állapotát az AngularFireAuth authState által.
* **isNotLogin()**: Ellenőrzi, hogy van-e bejelentkezett felhasználó. Ha nincs, true értéket ad vissza.
* **isLogin()**: Ellenőrzi, hogy van-e bejelentkezett felhasználó. Ha van, true értéket ad vissza.
* A bejelentkezett felhasználó adatait, beleértve az egyedi jogosultságokat (claims), a user objektumban tárolja és kezeli. Az egyedi jogosultságok kezelése az **getClaims(uid: string)** és **setCustomClaims(uid: any, claims: any)** metódusokkal történik, amelyek az HTTP klienst használják a szerverrel való kommunikációhoz.

Konstruktor:

* Inicializálja a Firebase Authentication (`fireauth`), a router és a HttpClient példányokat. A getLoggedUser-re feliratkozva kezeli a bejelentkezett felhasználó állapotát és jogosultságait.

**BASE SERVICE**

A BaseService egy Angular Injectable szolgáltatás, amely általános adatkezelési funkciókat biztosít az webshop termékeihez.

Metódusok:

* Termékek lekérdezése: A **getProducts()** metódus egy Subject objektumot ad vissza, amely a termékek listáját tartalmazza a Firebase adatbázisból. A komponensek feliratkozhatnak erre az adatfolyamra, hogy értesüljenek a frissítésekről (pl. új termékek, törölt termékek).
* Adatok frissítése: A **reload()** metódus frissíti a getProducts által elérhető adatfolyamot a legfrissebb adatokkal a Firebase adatbázisból.
* Termék lekérése: **loadProducts()**: metódus, ami lekéri a termékek adatbázisban lévő adatait az és a választ továbbítja a productsSub Subject-re.
* Termékmennyiség: **updateQty(qty: number, category: string, key: string)**: Frissíti a termék mennyiségét az adatbázisban, csökkentve azt a megadott mennyiséggel.
* Új termék felvétele: A **postProduct(body: any, category: string**) metódus új terméket ad hozzá a Firebase adatbázis megadott kategóriájához. Aszinkron metódus, ami Promise<boolean> típussal tér vissza, jelölve, hogy a művelet sikerült-e vagy sem.
* Termék törlése: A **deleteProduct(key: string, category: string)** metódus töröl egy terméket a Firebase adatbázis megadott kategóriájából. Ez szintén egy aszinkron metódus, ami Promise<boolean> típussal tér vissza.
* Termék adatainak módosítása: A **writeProductData(body: any, key: string, category: string)** metódus módosítja egy meglévő termék adatait a Firebase adatbázisban. Ez is egy aszinkron művelet, ami Promise<boolean> visszatérési értékkel rendelkezik.

**PROFIL SERVICE**

A ProfilService egy Angular Injectable szolgáltatás, amely a felhasználókkal kapcsolatos adatok kezeléséért felelős, különösen a Firebase Realtime Database használatával.

Funkciók és működés

* a felhasználói adatok betöltése, mentése és törlése
* A szolgáltatás az HttpClient-et használja a felhasználói adatok lekérésére egy REST API-n keresztül, és a usersSub Subject-et arra, hogy értesítse az alkalmazás többi részét az adatok változásáról. Az AngularFireDatabase-t és a Firebase Database SDK-t használja az adatok írására és törlésére a Firebase adatbázisban.

Metódusok:

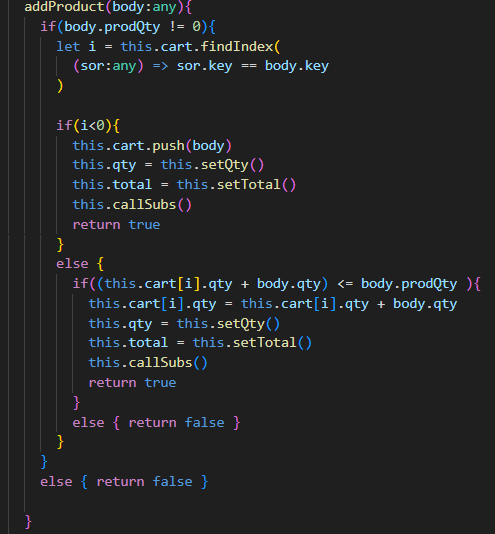
* **writeUserData():** Felhasználói adatokat ír a Firebase Realtime Database-be, egy adott key alapján. A `body` tartalmazza a menteni kívánt adatokat.
* **getUsers():** Visszaadja a usersSub Subject-et, amelyre más komponensek feliratkozhatnak a felhasználói adatok fogadására.
* **loadUsers():** Az HttpClient segítségével GET kérést indít a korábban megadott URL-re, és a választ továbbítja a usersSub Subject-en keresztül.
* **deleteUser(), deleteOrder(), deleteRating()**: Törli a megadott key-hez tartozó adatot a Firebase Realtime Database megfelelő részéből (users, orders, velemenyek).

**CART SERVICE**

A CartService egy Angular Injectable szolgáltatás, amely a felhasználói bevásárlókosarat kezeli az alkalmazásban. A szolgáltatás a bevásárlókosár tartalmát, a termékek összmennyiségét és az összértéket BehaviorSubject objektumok segítségével teszi elérhetővé a komponensek számára.

Metódusok:

* Termék hozzáadása a kosárhoz: A **addProduct(body: any)** metódus új terméket ad a bevásárlókosárhoz, vagy frissíti a meglévő termék mennyiségét, ha a termék kulcsa már szerepel a kosárban.
* Kosár ürítése: A **emptyCart()** metódus kiüríti a bevásárlókosarat, és nullára állítja a termékek összmennyiségét és összértékét.
* Termék mennyiségének frissítése: Az **updateProductQty(body: any)** metódus frissíti a bevásárlókosárban lévő termék mennyiségét.
* Termék eltávolítása a kosárból: A **removeProduct(body: any)** metódus eltávolít egy terméket a bevásárlókosárból.
* Bevásárlókosár lekérdezése: A **getCart()** metódus egy BehaviorSubject objektumot ad vissza, amely a bevásárlókosár aktuális tartalmát tartalmazza.
* Összmennyiség lekérdezése: A **getTotalQty()** metódus egy BehaviorSubject objektumot ad vissza, amely a bevásárlókosárban lévő termékek összmennyiségét tartalmazza.
* Összérték lekérdezése: A **getCartTotal()** metódus egy BehaviorSubject objektumot ad vissza, amely a bevásárlókosár termékeinek összértékét tartalmazza.



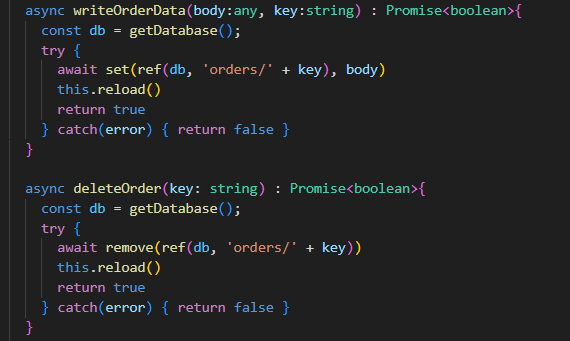
**ORDER SERVICE**

Az OrderService egy Angular Injectable szolgáltatás, amely a rendelések kezelését végzi az alkalmazásban. A szolgáltatás több funkciót biztosít a rendelések kezelésére:

* Rendelés feladása: postOrder
* Rendeletek módosítása az adatbázisban: writeOrderData
* Rendeletek lekérdezése: getOrders
* Rendeletek törlése: deleteOrder

Metódusok:

* **postOrder(body: any)**: Elküld egy új rendelést az adatbázisba. A metódus az HttpClient post metódusát használja a megadott URL-re, és JSON formátumban küldi el a body-ban kapott adatokat. A választ egy Observable<boolean> formában adja vissza, jelölve, hogy a művelet sikeres volt-e.
* **loadMyOrders(userEmail: string)**: Lekérdezi a megadott felhasználóhoz tartozó rendeléseket a Firebase adatbázisból, az e-mail címe alapján. A kapott adatokat egy tömbben gyűjti össze, amit vissza is ad.
* **writeOrderData(body: any, key: string)**: Frissít vagy hoz létre egy rendelést az adatbázisban a megadott kulcs alapján. Sikeres művelet esetén igaz (true) értékkel tér vissza, különben hamissal (false).
* **deleteOrder(key: string)**: Törli a megadott kulcsú rendelést az adatbázisból. Sikeres törlés esetén igaz (true) értékkel tér vissza, különben hamissal (false).
* **reload()**: Újratölti a rendeléseket az adatbázisból a loadOrders() metódus meghívásával.
* **getOrders()**: Visszaadja a orderSub Subjectet, amely az aktuális rendelések állapotát tartalmazza.
* **loadOrders()**: Betölti a rendeléseket az adatbázisból az URL használatával, és frissíti a orderSub Subjectet a kapott válasszal.

****

**PACKET-POINT SERVICE**

A PacketPointService egy Angular Injectable szolgáltatás, amely a FoxPost csomagautomaták listáját kezeli az alkalmazásban. A szolgáltatás egyetlen metódust biztosít a csomagautomaták adatainak lekérésére.

A getPacketPointList metódust használd a csomagautomaták listájának megszerzésére.

**RATING SERVICE**

A RatingService egy Angular Injectable szolgáltatás, amely a felhasználói értékeléseket kezeli az alkalmazásban.

Két fő funkciója van:

* Értékelések küldése a szervernek: HTTP POST kérést küld a felhasználó által megadott értékeléssel.
* Adott értékelés módosítása a Firebase Realtime Database-ben.
* Értékelések lekérdezése az adatbázisból.

Metódusok:

A **postRating(body: any)** metódust használjuk az értékelés adatbázisba történő feltöltésére, a **loadMyRatings(userEmail: string)** metódus szolgálaz adatbázisból való lekérésre, **a**  **writeRatingData(body: any, key: string)** metódust hasznájuk egy adott értékelés módosítására, a **deleteRating(key: string)** metódust az értékelés törlésére, **getRatings()** metódust pedig az értékelések lekérdezésére az adatbázisból.

**SEARCH SERVICE**

A SearchService egy Angular Injectable szolgáltatás, amely a kereséssel kapcsolatos adatokat és eseményeket kezel az alkalmazásban.

Felelősség:

* A keresőkifejezés tárolása.
* A komponensek értesítése a keresőkifejezés változásáról.
* A törlési állapot kezelése (opcionális).

Metódusok:

A **setSearchTerm(searchTerm: string)** metódust használd a keresőkifejezés beállítására, a **getSearchTerm()** metódust pedig az aktuális keresőkifejezés megszerzésére. A **clearTerm()** metódus a keresőkifejezés törlésére szolgál.

A SearchService a BehaviorSubject osztályt használja a keresőkifejezés és a törlési állapot tárolására.

**USER SERVICE**

A UserSevice egy Angular Injectable szolgáltatás, amely a felhasználót kezeli az adatbáziban.

Funkciók és Metódusok:

* Bejelentkezett felhasználó lekérdezése: A **loadUserData()** megkeresi és betölti a bejelentkezett felhasználó adatait, amit eltárol és tovább adja egy BehaviorSubjectnek
* A bejelentkezett user adatainak lekérdezése: A **getUserData()** visszaadja az a userSub BehaviorSubject objectumot, amire a komponensek feliratkozhatnak
* **setUser(data: any)** Beállítja a felhasználói adatokat a user változóban, és frissíti a userSub BehaviorSubject-et az új adatokkal. Ez a metódus biztosítja, hogy a felhasználói adatok mindig naprakészek legyenek az alkalmazásban.
* Új felhasználó felvitele az adatbázisba: Az **addUser(user:any, key:string)** metódus új felhasználót ad hozzá az adatbázishoz, egy általunk meghatározott kulccsal.

**GUARD**

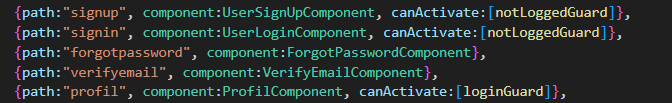
Routing keretében használtunk a navigációs műveletek ellenőrzésére és kezelésére. Ezekkel szabályoztuk bizonyos útvonalak elérését. A Guard típusok közül a CanActiveFn -t használtuk, mellyel ellenőriztük a hogy egy felhasználó jogosult-e az adott útvonal megtekintésére.

Három guard került létrehozásra:

* loginGuard, notLoggedGuard

ellenőrzik, hogy a felhasználó be van-e jelentkezve vizsgálják, ha nincs bejelentkezve, van-e érvényes munkamenete

A CanActivate függvénytípusvisszatérési értéke egy boolean amely jelzi, hogy az adott útvonalhoz hozzáférhet-e a felhasználó



* sAdminGuard

az admin bejelentkezéséhez lett létrehozva, annál a componensnél, amelyiknél az app-routing modulban a CanActive: [sAdminGuard] szerepel, azokat az útvonalakat csak admin jogosultsággal érjük el.

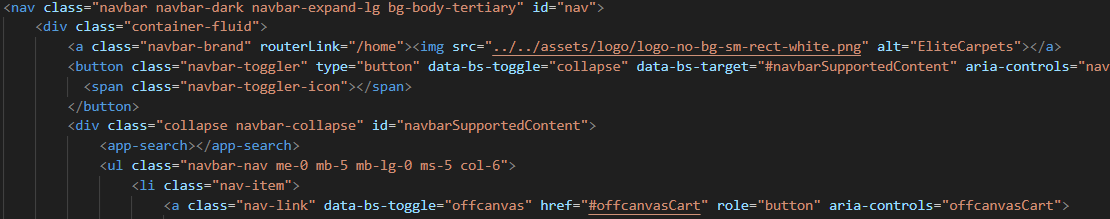
**ROUTING**

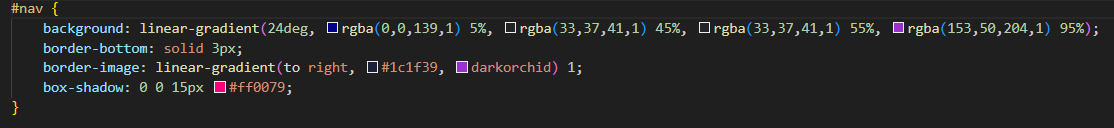
Az Angular Router modulját használjuk a navigációért és az útvonalak kezelésére szolgál. Az útvonalak konfigurációja egyúttal definiálja az alkalmazás szerkezetét. Az útvonalak definiálását külön modulban kezeljük. A routing rendszer integráltuk "guard" típusokkal is, amelyek segítettek a hozzáférés kezelésében.



**STÍLUSOK ÉS MEGJELENÉS**

Ennek kialakításához és az alkalmazás megjelenésének javításához CSS-t, React Native StyleSheet-et valamint a Bootstrap 5.3 verzióját használtuk, mivel a Bootstrap személyre szabható, az egyedi módosításokat jól kezeli és saját CSS-sel felülírható.





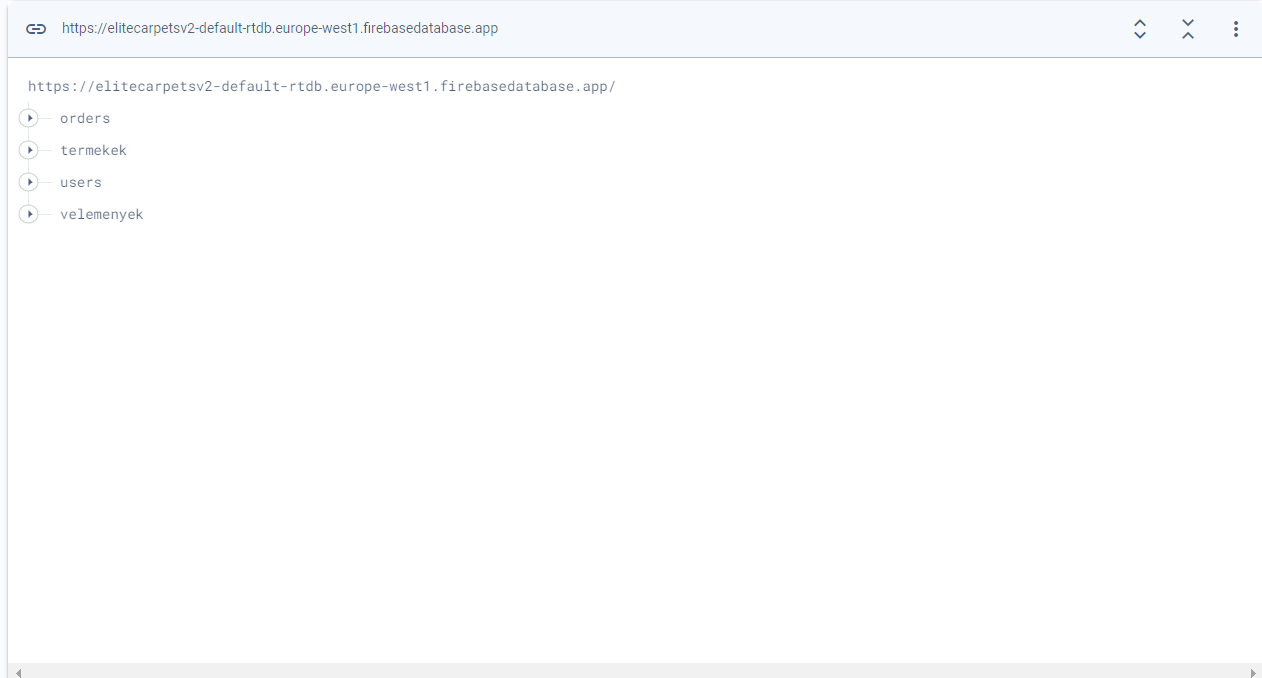
**ADATSZERKEZETEK, ADATMODELLEK**

Az adatok tárolását valós idejű adatbázis az ún. "real-time databases" (Firebase Realtime Database) végzi. Az adatok tárolása és szinkronizálása valós időben valósulhat meg. Az adatbázis automatikusan szinkronizálja az adatokat a szerver és a kliens között, így a változások azonnal láthatóak mindkét oldalon.

A termékek, rendelések és felhasználók adatait strukturált formában kezeljük és tároljuk, például objektumok vagy tömbök formájában ezzel lehetővé téve az adatok rugalmas tárolását és könnyű hozzáférhetőségét. A Firebase Realtime Database az adatokat JSON objektumok formájában tárolja.

A felhasználók azonosítása a Firebase Authentication szolgáltatásával lett megvalósítva. A belépéshez az Email/Password kombináció használtunk.

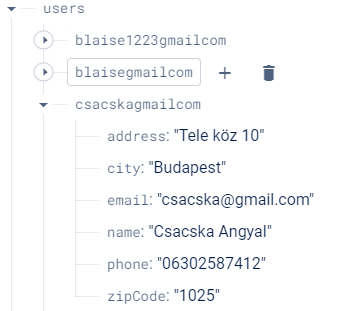
Az alábbiakban látható az adatbázis szerkezete és a tárolt adatok típusai:



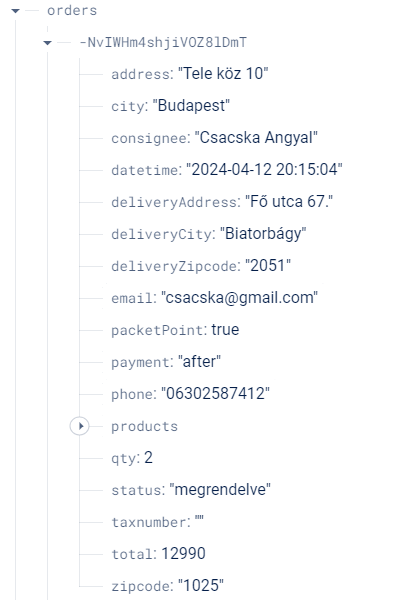
* A ’termekek’ csomópontban tároljuk a termékeket, és mindegyiket egyedi azonosítóval azonosítjuk. Minden terméknek vannak tulajdonságai, mint például a név, leírás, ár, kategóriák és elérhetőség.



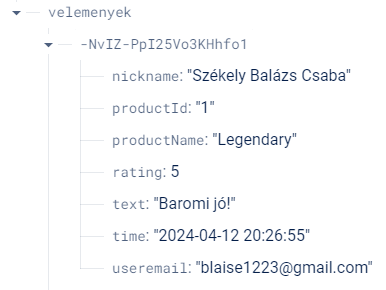
* A ’users’ csomópontban tároljuk a felhasználókat, és szintén egyedi azonosítóval azonosítjuk. Minden felhasználónak vannak tulajdonságai, mint például a név, email cím és cím.



* Az ’orders’ csomópontban tároljuk a rendeléseket, és szintén egyedi azonosítóval azonosítjuk. Minden rendelésnek vannak tulajdonságai, mint például a felhasználó azonosítója, dátuma, rendelt termékek és mennyiségek, teljes ár és státusz.



* A ’velemenyek’ csomópontban tároljuk a véleményeket, és szintén egyedi azonosítóval azonosítjuk. Minden véleménynek vannak tulajdonságai, mint például a felhasználó azonosítója, dátum, értékelés száma, szövege.



Adatbázis szabályok:

Ezek a szabályok az adatbázis biztonságát és hatékonyságát szolgálják, biztosítva, hogy csak megfelelő jogosultságokkal rendelkező felhasználók férjenek hozzá az adatokhoz, és hogy az adatbázis gyorsan tudjon reagálni a lekérdezésekre.

Jelenleg a következő Firebase adatbázis szabályok vannak érvényben:



**TESZTELÉS ÉS HIBAKERESÉS**

A program fejlesztése során az egyes befejezett elemek külön kerültek feltöltésre és a fejlesztéssel párhuzamosan bizonyos szintű tesztelés is megvalósult. Többször előfordultak szintaxishibák, gépelési vagy írásjel hibák, például lemaradt vessző, pontosvessző, vagy szerkesztés közben törölt egység után fennmaradt kapcsos zárójel, vagy ennek hiánya. Előfordult, hogy kimaradt a service injektálás a konstruktorba vagy a metódus meghívása.

A webfejlesztéshez használt alapértelmezett böngésző a Google Chrome. Az oldal végső tesztelése is a Google Chrome és Mozilla Firefox böngészőkkel valósult meg.

A két böngésző eredményei között nagy eltérés nem volt tapasztalható. A CSS, képbetöltés és oldalfrissítés a különböző böngészőknél más-más időt vett igénybe. A Bootstrappel szabályoztuk a containerek és gombok tulajdonságait, így ezeket a böngészők beállításai nem befolyásolták. Az egyetlen különbség a böngészők felugró warning üzenete volt, mely böngésző tulajdonság.

**EGYSÉGTESZT**

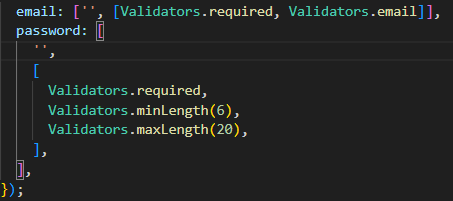
A fejlesztés során a metódusok kimeneti értéke, először a fejlesztői konzolra került kiíratásra, amennyiben a kapott visszatérési értékek és működés megfelelő volt, akkor a metódus megjelenítésre került a weboldalon.

Mivel a projekt fejlesztése még nem zárult le teljesen, néhány esetben még megjegyzések és konzolos kiíratások találhatóak a kódban a későbbi fejlesztésekhez, tesztek futtatásához.

**INTEGRÁCIÓS TESZT**

A teszt folyamán vizsgáltuk különböző komponensek közötti helyes adatátadást és interakciót. A bejelentkezési és regisztrációs felület tesztelése során hibás adatokat rögzítettünk vagy egyáltalán nem rögzítettünk semmit a beviteli mezőkbe. Erre azért volt szükség mert a userLogin és a userSignUp komponensek esetében előfordulhat, hogy hibás adatokat adnak meg. Az e-mail cím és jelszó megadása kötelező vizsgálva az e-mail cím formátumát és a jelszó hosszát is.

Az email és jelszó mező ellenőrzése a bejelentkezési felületen:



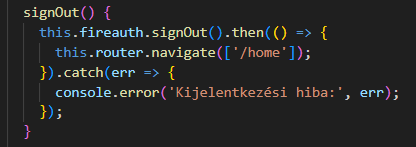
**ROBOTFRAMEWORK, SELENIUMLIBRARY TESZTEK**

A Robot Framework-höz kapcsolódó SeleniumLibrary segítségével automatizált funkcionális és felhasználói felület teszteket végeztünk a webalkalmazáson. Ezek az eszközök lehetővé tették a webes interakciók széles skálájának hatékony tesztelését, beleértve az űrlapok kitöltését, gombokra kattintást és dinamikus tartalmak kezelését.

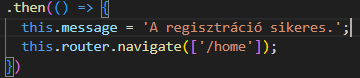
A tesztek kiterjedtek az alapvető felhasználói műveletekre, az adatbeviteli űrlapokra, navigációs elemekre és az adatintegritás ellenőrzésére. A tesztek során a fő hangsúly a felhasználói interakciók pontos és megbízható végrehajtásán volt, valamint az adatkezelési folyamatok hibamentes működésén.

A tesztek során jelentősebb hibák nem fordultak elő. Azonban észleltünk kisebb inkonzisztenciákat és pontatlanságokat, amelyek elsősorban az adatkezeléssel kapcsolatosak voltak, a termékek adatai nem töltődtek fel adatbázisba. Az adatok helytelen tárolódása vagy a beérkező adatok pontatlanságai állhatnak a problémák hátterében. Megoldására a backend adatkezelési logikáját és a frontend adatbeviteli validációs folyamatait alaposabban kellett vizsgálni. A hibát az okozta, hogy, ha a termék valamelyik tulajdonsága undefined értéket kapott akkor az adatok tárolása nem valósult meg, így a kódban szükséges volt átírni, hogy, ha nincs adat az adott tulajdonságnál akkor egy üres stringet adjon vissza.

A tesztelés során a kilépés gombra való kattintással nem az elvárt Home-ra navigálta vissza automatikusan, hibát nem jelzett csak nem teljesítette az elvárt eredményt, az éppen megtekintett oldalon maradtunk. A megoldás az auth.services.ts signOut() metódusának kibővítése volt a programozott útvonalindítással



Navigációs probléma volt észlelhető a regisztrációs űrlap kitöltése utána is, mivel a sikeres regisztrációt követően még mindig a regisztrációs űrlapot láttuk, annak ellenére, hogy profilunkba be voltunk jelentkezve. A probléma abból adódott, hogy az onSubmit() metódus nem kezelte sikeres regisztrációt követő navigációt, melyet az alábbi programozott útvonalindítás elindításával kezeltünk.



**FEJLESZTÉSI LEHETŐSÉGEK**

Jelenlegi állapota szerint a weboldalon sikeresen működik a regisztráció, bejelentkezés, felhasználói profil, termékek megjelenítése, kiválasztása és a vásárlási folyamat. A webalkalmazáshoz adminisztrációs felület is készült.

A webalkalmazást tovább szeretnénk fejleszteni annak érdekében, hogy optimalizáljuk a kódunkat és javítsuk a rendszer teljesítményét. További funkcionalitások hozzáadását is tervezzük, amelyeket az eredeti fejlesztési fázisban idő vagy erőforrás hiányában nem tudtunk implementálni. Ezen felül néhány technikai kihívás is akadályozta a fejlesztést, amelyek megoldása komoly gondolkodást és időráfordítást igényelt. A közeljövőben tervezett fejlesztések között szerepel pl. az adminisztrációs felület admin jogainak változtathatóságával, a felhasználói felület további bővítése még több felhasználóbarát megoldással, mint pl. rendelések törlésének megoldása és annak kezelése törlés esetén, hogy erről a szállító is abban a pillanatban értesüljön, ahogy a rendelés törlésre kerül a felületen, online fizetés bevetése.

**Felhasznált irodalom:**

Angular dokumentáció: <https://angular.io/docs>

ng-bootstrap dokumentáció: <https://ng-bootstrap.github.io/>

Firebase dokumentáció: <https://firebase.google.com/docs/>

Visual Studio Code dokumentáció: <https://code.visualstudio.com/docs/>

Git dokumentáció: <https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository>