Az én diplomamunka témám egy webshop fejlesztése backend alkalmazással, webes és mobil klienssel. Erre a félévre a kitűzött cél a backend alkalmazás elkészítése és a webes kliens elkezdése volt.

Egy létező kisvállalkozás számára készítem a webshopot, a vállalkozás különböző hajgöndörítők, hajgumik és hasonló termékek értékesítésével foglalkozik. Eddig csupán közösségi médián, és emailes rendeléseken keresztül értékesítették a termékeiket, és igény merült fel egy saját webshopra. A webshopom egy REST API-val rendelkező backend alkalmazásból, egy webes és egy mobil kliensalkalmazásból áll. A backendet ASP.NET Core-ral, a webes kliens appot Angularral készítettem el, a mobil kliens pedig Android platformon fogom. Az adatbáziskezelésre az EntityFramework Core-t a felhasználókezelésre az Identity plaftormot használtam.

A backend fejlesztése során a hagyma architektúrára alapoztam, amellyel lazán csatolt megoldások készíthetők. A hagyma architektúra előnye emellett a jobb karbantarthatóság, mivel az összes függőség csak egyetlen irányba, kívülről befelé mutathat. A hagyma architektúra nagy mértékben épít a függőségek megfordításának alapelvére, ezért a rétegek közötti kommunikáció interfészekkel történik. A fejlesztés során 5 réteget alakítottam ki. A legbelső a domén entitások rétege, ez az alkalmazás központi része, itt vannak definiálva az alkalmazásban létező entitások. Ezt követi az adatelérési réteg. Ez absztrakciós célt szolgál, lehetővé teszi az üzleti logikai réteg számára az egyéges adatelérést. A következő a szolgáltatások rétege. Itt az alkalmazás funkcióinak implementációja található, ezt szokták még üzleti logikai rétegnek is hívni. A következő a webes réteg, ebben találhatóak az API végpontok, a kliensek ezzel a réteggel tudnak kommunikálni. Az ötödik réteg igazából maga a frontend, vagyis a kliens alkalmazások.

Az osztálydiagramon láthatóak a domén entitások osztályai, és ezek kapcsolatai. Az ApplicationUser és ApplicationRole osztályok a felhasználók és a szerepek modellezésére szolgálnak. Az alkalmazásomban admin és sima felhasználó szerepek vannak. Ezek az Identity által definiált megfelelő ősosztályokból származnak le. Ezek az ősök tartalmazzák azokat a tulajdonságokat, amelyekre az autentiákcó és az autorizáció során szükség van. Az Order egy rendelést reprezentál, az OrderItem egy kapcsoló tábla a termékek és rendelések között. A ProductCategory és a Product a termékkategóriákat és a termékeket reprezentálják. Minden felhasználónak lehet egy kosara, amit a Cart reprezentál. Azonban egy kosár létezhet felhasználó nélkül is, mivel bejelentkezés nélkül is lehet kosárba rakni termékeket, és rendelést leadni. A CartItem egy kapcsoló tábla a termékek és a kosarak között. Létrehoztam egy külön Address osztályt, ami tartozhat egy rendeléshez és egy felhasználóhoz is. Az Address nem az Entity ből hanem a ValueObject ősből származik, mivel ez egy olyan adategység aminek nincs saját identitása.

Az alkalmazás Json access és refresh token alapú authentikációval rendelkezik. A kliensek egy érvényes felhaszáló és jelszó páros megadásával kapnak egy hozzáférési és egy refresh tokent. A hozzáférési tokent a kérések fejlécében tünteti fel a kliens, majd amikor lejár a token, akkor a refresh token segítségével kérhet új hozzáférési és refresh token párost. Ennek lényege, hogy ha egy rosszindulatú félnek sikerül megszereznie a kliens hozzáférési tokenjét, akkor azt csak rövid ideig képes használni. Épp ezért az access token érvényességi ideje rövid, általában 1-3 perc, az én alkalmazásomban ez 1 perc. Mielőtt a beérkezett kérések tényleges feldolgozása megkezdődne ellenőrzésre kerülnek a klienstől kapott adatok (DTO objektumok) egy validációs osztályban. Hibát találva a kontrollerek érintése nélkül validációs hibaüzenetet küld vissza a rendszer. A kontrollereket igyekeztem vékonyan tartani, szinte pár sorosak, és delegálnak az üzleti logikai réteg felé.

A kéréseket feldolgozó pipeline-ba regisztráltam egy globális hibakezelő middleware-t, ez egyszerűen egy try-catch blokkal körülvett áthívás a pipeline következő elemébe így, ha a pipeline bármely további részében kivétel keletkezik, az legkésőbb ebben a függvényben el lesz kapva és le lesz kezelve. Több kivételt definiáltam, ezekre építettem a hibakezelést és a válaszhoz a megfelelő státuszkód beállítását. Az alkalmazás működése során folyamatos naplózást végeztem egy konzolos felületre.

Itt egy példát mutatnék egy kérés kiszolgálására. Látszik hogy a Controller réteg vékony, hisz egysoros a függvény. Ez áthív a megfelelő Service osztályba, ami a Repositoryn keresztül elkéri az adatbázistól a megfelelő adatokat, majd elvégzi az entitás letérképezését egy ViewModel objektumra és visszaadja azt.

Magát a webalkalmazást az Angular keretrendszerrel készítettem el. A reszponzív felhasználói felület kialakítása érdekében az Angular Flex Layout könyvtárat használtam, ez egy Flexbox és mediaQuery alapú könyvtár amit a hivatalos Angular csapat fejleszt, és nagyon jól használható Angularra. A felhasználói felülethez az Angular Material komponenseit használtam. A kliensalkalmazás modell osztályait az NSwagStudio nevű program segítségével hoztam létre, amely OpenAPI dokumentáció alapján képes kliens osztályokat generálni. A bejelentkezett felhasználó autentikációs tokenjét egy http Interceptor illeszti a kimenő kérések Authorization fejlécébe. A token frissítését guardokkal oldottam meg. Ha a user lejárt access tokennel probál egy védett routera navigálni, akkor a guard meghívja a refresh tokennel a backend refresh végpontját és a visszakapott érvényes hozzáférési tokent elmenti, majd folytatja a navigációt. Ha nem sikerül a frissítés, mert pl lejárt a refresh token is, akkor a bejelentkezés képernyőre juttatja a felhasználót. A kosár kezelését is megemlíteném itt, ez egy komplexebb feladatot volt. Azt akartam, hogy a webshopot bejelentkezés nélkül is lehessen használni, illetve hogy a kosár tartalma el legyen mentve, ha bezárja az alkalmazást a felhasználó. Ezt úgy oldottam meg, hogy az alkalmazás a LocalStorageben eltárolja a felhasználó adatait és a kosár id-ját. Induláskor ha el van mentve egy kosár id akkor a szerver alkalmazástól lekéri a kosár tartalmát, illetve ha van felhasználó, akkor automatikusan bejelentkezteti.

Bejelentkezés képernyő, profil képernyő.

Mint említettem, a webes kliens implementációjának egy része, és a teljes mobil kliens lefejlesztése még hátravan, ezeket nyáron és ősszel tervezem elkészíteni. A fő részek amik hiányoznak még a webes kliensből az a webes fizetés implementálása, valamilyen 3rd party szolgáltatóval pl Barion. Illetve az admin felület elkészítése, ahol az admin megtekintheti a leadott rendeléseket, illetve kezelheti a webshopban lévő termékeket és termékkategóriákat.