A félév során az Önálló labor 2 tárgy keretein belül a feladatom egy-egy webes frontend és backend technológia megismerése és ezek segítségével egy webalkalmazás/webshop kialakítása volt.

Egy létező kisvállalkozás számára készítettem el 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 volt egy saját weboldalra. A webalkalmazásom igazából egy REST API-ból és egy webes frontendből áll. A backendet ASP.NET-tel, a frontendet Angularral készítettem el. Az adatbáziskezelésre az EntityFramework Core-t a felhasználókezelésre az Identity plaftormot használtam.

A backend fejleszése során a hagyma architektúrára alapoztam, amellyel lazán csatolt megoldások készíthetők. A hagyma architektúra előnye emellett a jobb karbantarhatóság, mivel az összes függőség csak egyetlen irányba, kívűlről befelé mutathat. A hagyma architektúra nagy mértékben épít a függőségek megfordításának alapelvére, ezért a rétegek közötti kommunikáció interfészekkel történik. Ez magával hordozza azt is, hogy a konkrét implementációk csak futásidőben vannak biztosítva az alkalmazás számára. 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 kívülről kommunikálni. Az ötödik réteg igazából maga a frontend, vagyis egy böngészőben futó webes alkalmazás.

A dián látható osztálydiagramon láthatóak az implementáció során létrehozott osztályok és a kapcsolataik. Az ApplicationUser és ApplicationRole osztályok a felhasználók és a szerepek modellezésére szolgálnak. Ezek az Identity által definiált megfelelő ősosztályokból származnak le. Ezek az ősök tartalmazzák azokat a tulajdonságokat, amelyekre az autentiákcó és az autorizáció során szükség volt. Jelenleg az alkalmazásba csupán az admin tud bejelentkezni, a sima felhasználók kezelése nem volt prioritás, azonban szándékosan modelleztem előre a szerepköröket is, mivel erre a későbbi továbbfejlesztés során szükség lesz. Az EntityBase egy absztrakt ős, az összes entitás számára szükséges azonosítót tartalmazza. A ProductCategory osztály egy termékkategóriát reprezentál, a webshopban ez pl. hajgöndörítők, hajgumik, hajcsatok stb. A Product osztály egy adott terméket reprezentál, ennek van neve, ára, leírása stb. Az Order osztály egy megrendelést reprezentál, amelyet egy vásárló tud leadni, és az adatbázisban rögzítjük. Az OrderItem pedig egy kapcsolatot reprezentál egy termék és egy megrendelés között. Erre azért volt szükség, mert pl ez segít modellezni azt, hogy ha egy termékből több darabot rendel a vevő, illetve az árkedvezményeket is könnyebb így modellezni.

Az alkalmazásban az admin autentikációját Json webtokenekkel oldottam meg. Érvényes felhasználó és jelszó páros megadásával hozzáférési tokent kérhetnek az alkalmazástól, amelyeket a további kérések fejlécében kell feltüntetni. 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 csővezeték következő elemébe így, ha a csővezeték 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.

Magát a webalkalmazást az Angular keretrendszerrel készítettem el. A reszponzív felhasználói felület kialakítása érdekében Bootsrap keretrendszer CSS osztályait és az Angular Material komponenseit használtam. A kliensalkalmazás modell osztályait az NSwagStudio nevű program segítségével hoztam létre, amely OpenAPI dokumentáció alapján képes kliens osztályokat generálni. Itt megemlítenék néhány általános fejlesztői megoldást, amit a kliens fejlesztése során alkalmaztam. Az eseményekre fel és leiratkozás egységes kezelésére létrehoztam egy közös ősosztályt, amely nyilvántartja az eseménykre való feliratkozást, és az adott Angular komponens törlődésekor leiratkozik a megfelelő observable objektumokról, és ezáltal elkerüli a memória szivárgást. A bejelentkezett felhasználó autentikációs tokenjét egy http Interceptor illeszti a kimenő kérések Authorization fejlécébe.

Mutatnék néhány képernyőképet a webshopról. Ezen a képen a termékkategóriák oldalt láthatjuk. Ezen pedig a megrendelésnél kitöltendő Form látható. Ezen a képen pedig az leadott megrendelések és azok adatai látszanak, ezt csak az admin tekintheti meg.

A webalkalmazásban rengeteg továbbfejlesztési lehetőség rejlik. Látható volt a képernyőképek alapján, hogy a felhasználói felület még elég kezdetleges és egyszerű, ahhoz, hogy a felhasználói élmény jobb legyen, még rengeteg részén lehetne fejleszteni, de ezek főleg designer szempontok, így eddig nem élveztek prioritást a fejlesztés során. Emellett a felhasználó kezelést is lehetne tovább fejleszteni. Lehetővé kéne tenni a vevők számára a regisztrációt és bejelentkezést, amely megkönnyítené a rendelés menetét. Emellett az admin számára ki lehetne alakítani külön felületet, ahol hozzáadhat új termékeket, illetve szerkesztheti a meglévőket.