Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Schütz Máté

WEBSHOP FEJLESZTÉSE WEBES ÉS MOBIL KLIENSRE

KONZULENS

Albert István

BUDAPEST, 2022

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Schütz Máté**, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző, cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2022. 05. 22.

...…….

Schütz Máté

1 Bevezetés

Manapság a kisvállalkozások számára elengedhetetlen, hogy valamilyen formában jelen legyenek az online térben, hiszen a fogyasztók jelentős része már online rendeléseken keresztül intézi el a különböző vásárlásait. Ez a vállalkozás típusától függően általában egy weboldal, webshop, és/vagy mobil alkalmazás formájában testesül meg.

1.1 A dolgozat szerkezete

Ebben az alfejezetben a dolgozat szerkezetét, felépítését mutatom be. A dolgozat 7 fő fejezetből áll.

Az első fejezet a bevezetés, ebben a diplomamunkám téma választását és ennek a motivációját mutatom be.

A második fejezetben hasonló, már létező megoldásokat mutatok be, illetve azt, hogy ezekből miket merítettem, ültettem át a saját megoldásomba, illetve miket csináltam másképpen. Emellett bemutatom a fejlesztés során használt technológiákat.

A harmadik fejezetben az elkészített rendszer architektúráját ismertetem és megmutatom, hogy hogyan épül fel a webshop a három különböző alkalmazásból.

A negyedik, ötödik és hatodik fejezet a szerveralkalmazásról, a webes kliens alkalmazásról és a mobil kliens alkalmazásról szól, ebben a sorrendben. Ezekben a fejezetekben egyesével bemutatom a három különböző alkalmazást. Mindegyiknél kitérek a fontosabb fejlesztői és tervezői döntésekre, amelyeket az alkalmazások fejlesztése során hoztam meg. Emellett bemutatom az alkalmazások lényegesebb részeit, és az érdekesebb fejlesztői megoldásokat.

A hetedik fejezet az irodalomjegyzéket tartalmazza, amelyekben a dolgozat elkészítéséhez források találhatóak.

1.2 Témaválasztás

A szoftverfejlesztés területén a webfejlesztés egyre nagyobb teret nyert az elmúlt évtizedek során. Ma a fejlesztők többségének a programozás egyet jelent a webfejlesztéssel [1]. Éppen ezért én is fontosnak tartottam, hogy jelentősebb tapasztalatot szerezzek a webfejlesztés terén, hogy később ezt karrierem során kamatoztathassam.

Egyetemi tanulmányaim során néhány tárgy keretén belül már megismerkedtem a webfejlesztés különböző aspektusaival, azonban még jobban el akartam mélyíteni a tudásomat ezen a területen. Ebből az okból kifolyólag mindenképpen egy webfejlesztéshez kapcsolódó témát akartam választani a diplomamunkámnak.

Egy ismerősöm az elmúlt évben elindított egy kisvállalkozást, amely kézzel készített hajgöndörítő, illetve hajápoló termékek eladásával foglalkozik. Eddig a vevőket különböző szociális média platformokon keresztül érte el – Instagram és Facebook. A vevők itt tudták leadni a rendeléseiket személyes üzeneteken keresztül. A vállalkozás sikerességéből adódóan felmerült az igény egy saját webshop készítésére, amely megkönnyítené a vevők számára a rendelések leadását, javítaná a vásárlási élményt, illetve levenné a vállalkozó válláról a rendelések manuális kezelésének terhét.

Úgy gondoltam, hogy ennek az alkalmazásnak a megvalósítása egy tökéletes lehetőség arra, hogy elmélyítsem a tudásom és megfelelő tapasztalatot szerezzek a webfejlesztés területén. Egy webshop alapvetően két alkalmazásból áll. Az egyik a szerver oldali alkalmazás, amely a webshop üzleti logikáját valósítja meg, és tartalmazza a szükséges adatokat. A másik maga a kliensoldali webalkalmazás, amely a felhasználó böngészőjében fut, és egy megfelelő felületet biztosít a vevő számára. Emellett én úgy döntöttem, hogy megvalósítok egy második kliens oldali alkalmazást is, amely egy Android platformra készített mobil alkalmazás. Ennek oka, hogy már évek óta érdekel a mobil szoftverfejlesztés és a vállalkozás számára ez is egy hasznos bővülési lehetőség, amellyel még több vevőt tud elérni.

2 Hasonló megoldások, technológiák bemutatása

Ebben a fejezetben bemutatok különböző létező megoldásokat, amelyekből ihletet merítettem, illetve amelyeket átültettem a saját alkalmazásomba. Emellett ismertetem a fejlesztés során felhasznált technológiákat, amelyek az alkalmazásom alapját képezik.

2.1 Hasonló megoldások

Az alkalmazásom megtervezésekor egyrészt a saját felhasználói tapasztalataimra támaszkodtam, másrészt rengeteg különböző webshopot áttekintettem. Ezekből merítettem ihletet, illetve megtapasztaltam, hogy mik azok a tipikus hibák amelyek a felhasználói élményt csökkentik.

Az egyik ilyen fő hiba az, hogy sok webshop különböző megfontolásokból és okokból adódóan nem enged regisztráció, illetve bejelentkezés nélkül rendelést leadni. Ez sokszor el tudja tántorítani a potenciális vásárlót, hiszen általában egy fiók létrehozása teljesen feleslegesnek tűnik a felhasználó számára, főleg ha csak egy egyszeri vásárlásról van szó. Sokan azzal érvelnek, hogy azzal, hogy regisztrációra kényszerítjük a vevőt, nagyobb eséllyel konvertálhatjuk visszatérő vásárlóvá. Véleményem szerint azonban egy modern e-commerce oldal esetében sokkal fontosabb a felhasználói élmény emelése. Ha a felhasználó úgy érzi, hogy feleslegesen kell időt töltenie a regisztrációval, sokkal nagyobb az esélye, hogy egy másik webshophoz fog fordulni, vagy egyszerűen elmegy a kedve a vásárlástól.

Egy másik jellegzetes hiba, ami általában a kisebb, kevesebb erőforrással rendelkező webshopokra jellemző, hogy miután a felhasználó hozzáadta a megrendelni kívánt termékeket a kosarához, majd ezután regisztrált, illetve bejelentkezett, elveszíti a kosarának a tartalmát. Ez rendszerint nagy mértékben csökkenti a felhasználói élményt, és frusztrációhoz vezethet.

2.2 Felhasznált technológiák

2.2.1 Verziókezelő eszközök

Verziókezelésre és forráskód kezelésre Gitet használtam, amely manapság a legelterjedtebb eszköz erre a feladatra. [2] A Git egy ingyenes és nyílt forráskódú elosztott verzió kezelő rendszer, amely egyaránt használható effektíven kis és nagy méretű

projekteknél. Tanulmányaim során mindig Gitet használtam verziókezelésre, így egyértelmű volt ennek az eszköznek a kiválasztása. A forráskódom biztonságos tárolására GitHub-ot használtam, amely a Git segítségével verziókezelésre és forráskód tárolásra használható. Emellett hozzáférés-kezelést és számos együttműködési funkciót nyújt, amelyeket természetesen diplomamunkám során nem vettem igénybe, hiszen egyszemélyes projekten dolgoztam.

2.2.2 Backend alkalmazás technológiák

A backend alkalmazásomat a Visual Studio [3] fejlesztői eszköz segítségével készítettem el C# nyelven. [4] A munkám során főleg .NET platformon és C# nyelven fejlesztettem, így ebben van a legtöbb tapasztalatom. Emiatt úgy döntöttem, hogy a backend alkalmazást ASP.NET Core platformon készítem el. [5] Az ASP.NET Core egy nyílt forráskódú keretrendszer, amely segítségével modern, internet kapcsolattal rendelkező alkalmazásokat lehet készíteni. A backend alkalmazásom gyakorlatilag egy web API (Application Programming Interface). Ez egy webszerveren futó alkalmazás, amely egy interfészt biztosít a külvilág számára, és ezen az interfészen keresztül, kérések segítségével kommunikálnak vele a kliens oldali programok.

A backend alkalmazásom rendelkezik egy adatbázissal, ami tárolja a webshop adatait, ezek például a megrendelhető termékek, a leadott rendelések, és a felhasználók adatai. Ezen adatok tárolására MSSQL [6] adatbázist használtam, amely egy relációs adatbázis kezelő rendszer, amelyet a Microsoft fejleszt. Az adatbázis elérésére, és az adatbázis struktúra diagram előállításához a Microsoft SQL Server Management Studio [7] szoftvert használtam.

Az adat hozzáféréshez és az objektum-relációs letérképezéshez az Entity Framework Core [8] keretrendszert használtam. Ez egy nyílt forráskódú, cross-platform verziója a népszerű Entity Framework keretrendszernek. Ez a keretrendszer nagyban megkönnyíti az adatbázis hozzáférést .NET Core platformon, hiszen a segítségével .NET objektumokon keresztül kezelhetjük az adatbázist és nincs szükség szinte semmilyen adathozzáférési kód megírására.

Az API leírásához a Swashbuckle.AspNetCore [9] NuGet csomagot használtam, amely egy Swagger leíró implementációt tartalmaz. Ez automatikusan generál egy OpenAPI specifikációnak [10] megfelelő API dokumentációt az ASP.NET Core

projektből. Emellett a felhasználói felületről közvetlenül kipróbálható az API, azaz küldhetünk teszt kéréseket a backend alkalmazásnak.

A backend alkalmazás Input validálásához a FluentValidation [11] NuGet csomagot használtam. Ez egy .NET könyvtár, amely segítségével a segítségével attribútumok megadása helyett fluent módon írhatunk validációs kódot a Data Transfer Object (DTO) osztályaink számára. Így sokkal tisztább maradt a DTO osztályok kódja, nem szennyezi be a sok validációs attribútum. Emellett a validációs kódom egy helyen található, így könnyebben karban tartható.

Az objektumok letérképezéséhez az AutoMapper [12] könyvtárat használtam. Ennek a segítségével egyszerűen írhatunk letérképezési szabályokat és ez leveszi a vállunkról a letérképezési kód írásának terhét.

A backend alkalmazásban az Inversion of Control (IoC) [13] minta megvalósításához az Autofac [14] könyvtárat használtam. Az IoC minta lényege, hogy ahelyett, hogy az alkalmazásunkban az osztályok hozzák létre a saját függőségeiket, az osztályok létrehozásakor „odaadjuk” nekik ezeket a függőségeket, ezt hívják függőség injekciónak. Így az alkalmazásunk lazábban csatolt marad, és könnyebb kezelni a függőségeket. Az Autofac lényege, hogy egy konténert szolgáltat, amelybe beregisztrálhatjuk a függőségeinket, és így ő később szolgáltatni tudja ezeket az osztályaink számára.

2.2.3 Webes alkalmazás technológiák

A webes kliens alkalmazás elkészítését a Visual Studio Code [15] fejlesztői eszköz segítségével végeztem el TypeScript [16] nyelven.

A webalkalmazásomat egy modern JavaScript keretrendszer segítségével szerettem volna elkészíteni, és végül az Angular [17] mellett döntöttem. Az Angular egy webes keretrendszer, fejlesztői platform, amelyet a Google fejleszt, és Single-Page Application-ök (SPA) készíthetőek vele.

A felhasználói felület elkészítéséhez az Angular Material [18] könyvtárat használtam. Ez egy komponens könyvtár, amelyet a hivatalos Angular csapat fejleszt és megfelel a Material Design [19] specifikációnak. Segítségével könnyen tudtam integrálni modern és szép UI komponenseket az Angular alkalmazásomba.

Az alkalmazás függőségeinek telepítéséhez a Node Package Managert [20] (npm) használtam. Az npm egy csomagkezelő szoftver JavaScript programozáshoz.

2.2.4 Mobil alkalmazás technológiák

Az Android alkalmazás fejlesztéséhez az Android Studio [21] szoftvert használtam. Ez a Google által is hivatalosan támogatott integrált fejlesztői környezet (IDE) Android alkalmazások fejlesztéséhez. Az alkalmazást Kotlin [22] nyelven készítettem el. 2019 óta a Kotlin a Google által is preferált és ajánlott nyelv Android platformra készített alkalmazásokhoz.

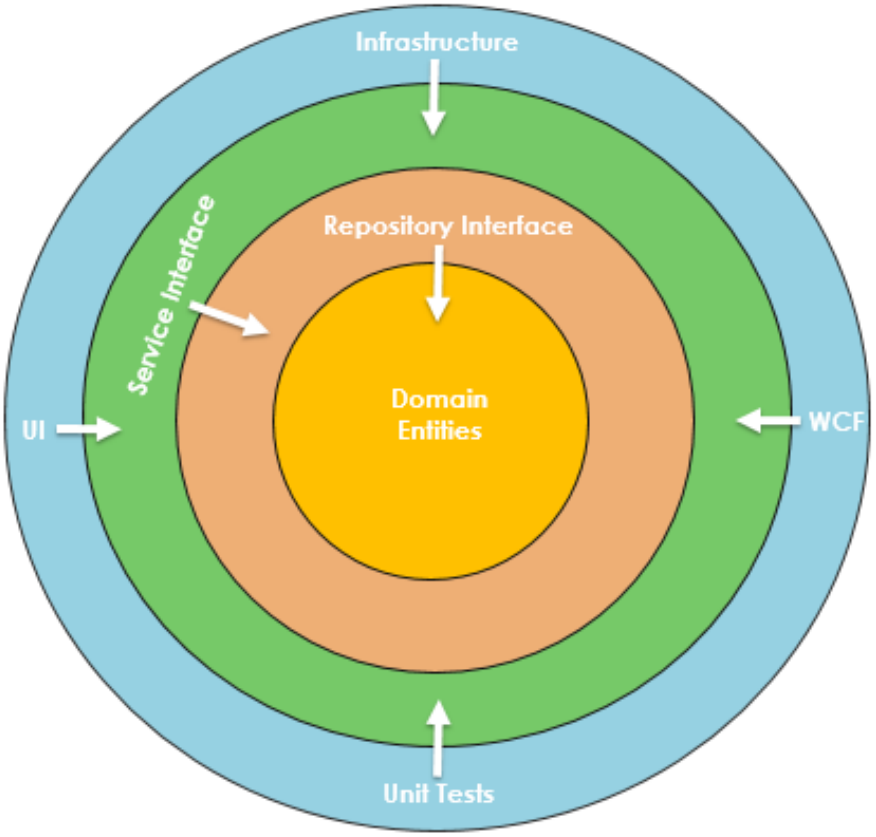
2.2.5 Egyéb technológiák

A dolgozatot Microsoft Word segítségével készítettem el, míg a diplomamunka bemutatásához, prezentálásához a Microsoft PowerPoint szoftvert használtam.

3 Architektúra

3.1 Backend alkalmazás architektúrája

A fejlesztés során az ASP.NET Core projektek esetében gyakran alkalmazott hagyma architektúrára építettem, amellyel lazán csatolt megoldások készíthetők.



1. ábra: A hagyma architekúra rétegei

Szoros csatolásnak azt nevezzük, amikor egy osztály egy másik osztály konkrét implementációjától függ. Ezzel ellentétben a laza csatolás fennállása esetén egy osztály anélkül használhat egy másik osztályt, hogy függene tőle. Ennek elérésével csökkenthető a komponensek közötti függőség és ezzel együtt az átterjedő módosítások kockázata.

A hagyma architektúra előnye emellett a jobb karbantarthatóság, mivel az összes függőség csak egyetlen irányba, kívülről-befelé mutathat. Ezért adott réteg módosítása esetén biztosak lehetünk benne, hogy a tőle belsőbb rétegek nem szorulnak korrigálásra. Ha valahol máshol is változást vált ki a módosításunk, az csak külsőbb rétegekben lehet.

Az architektúra nagy mértékben épít a függőségek megfordításának alapelvére, a rétegek közötti kommunikáció interfészekkel történik. Ez magával hordozza azt is, hogy a konkrét implementációk futásidőben vannak biztosítva az alkalmazás számára. Ennek lehetővé tételét az Autofac kontrol megfordítási keretrendszer segítségével oldottam meg. Az architektúra az ábrán látható módon négy rétegből áll, amiket a saját implementációnkba is átvezettünk:

1. Domén entitások rétege: az alkalmazás központi része, a domén entitásokat tartalmazza.

2. Adatelérési réteg: absztrakciós réteg, amely egységes adatelérést biztosít az üzleti logikai réteg számára. Ez a réteg foglalkozik az adatforrás lekérdezésével, a kapott adatok objektumokká történő leképezésével, illetve a módosítások adatforráson történő érvényesítésével.

3. Szolgáltatások rétege: üzleti logikai rétegnek is hívható, az alkalmazás funkcióinak implementációját tartalmazza. A web réteg által jelzett műveleteket végzi el, eközben, ha szükség van nem elérhető adatokra akkor az adatelérési rétegtől kérdezi le azokat. Az adatok módosítása után az adatelérési réteget bízza meg a változtatások tárolásával is.

4. Web réteg: az alkalmazást teszi elérhetővé, hogy kliensek segítségével a felhasználók élni tudjanak azokkal a funkciókkal, szolgáltatásokkal, amiket a szoftver nyújt. A felhasználók által kezdeményezett műveleteket továbbítja az üzleti logikai réteg megfelelő részére.

3.2 Webes kliens alkalmazás architektúrája

A webes kliens alkalmazás egy tipikus Angular alkalmazás architektúrájával rendelkezik. Az Angular keretrendszer segítségével flexibilis és moduláris alkalmazásokat hozhatunk létre. Ezt segíti elő a beépített függőség injektáló keretrendszer. Ennek lényege, hogy amikor létrehozunk egy új *Service* osztályt, akkor automatikus megjelöli ezt az osztályt az *Injectable* kulcsszóval és ezzel jelzi az Angular számára, hogy felhasználhatja ezt az osztályt függőség injektálásra. A *Service* osztályok feladata általában adatok fogadása, és küldése a külvilág felé – az én esetemben a backend alkalmazás felé. Az Angular alapvetően konstruktor injekcióval dolgozik. Ennek lényege, hogy amikor egy komponens osztályban fel akarjuk használni egy *Service* osztály

szolgáltatásait, akkor a komponens osztály konstruktorában kell ezt jeleznünk egy, a *Service* osztálynak megfelelő típusú argumentummal.

Az Angular alkalmazásomban a backend alkalmazás API-jának megfelelően hoztam létre a Service osztályokat. Ez azt jelenti, hogy egy Service osztály egy adott típusú feladatot végez el. Például az OrderService osztály a rendelésekkel kapcsolatos kérések küldéséért és fogadásáért felelős.

Az *AuthService* emellett még a webes kliens autentikációs logikájáért is felelős. Egy *BehaviorSubject*-ben tárolja az épp aktuálisan bejelentkezett felhasználót, és erre feliratkozva tud reagálni az alkalmazás többi része az autentikációs eseményekre. Emellett az AuthService a LocalStorage segítségével eltárolja a felhasználó böngészőjében a felhasználó adatait, így ha bezárja, majd később újra megnyitja az alkalmazást, nem lesz szüksége újra bejelentkezni.

A *CartService* a kosár kezeléséért felelős logikát tartalmazza. Az alkalmazást a mai modern webshopokhoz hasonlóan alakítottam ki. Ennek lényege, hogy a felhasználó akár bejelentkezés nélkül is rakhat termékeket a kosarába, adhat le rendeléseket. Azonban ha bejelentkezik, akkor az addigi kosarának tartalma nem szabad hogy elvesszen. További elvárás, hogy ha a felhasználó bezárja az alkalmazást, és később újra megnyitja, akkor legyen elmentve a kosarának tartalma. Ezeknek az elvárásoknak megfelelő megoldást a CartService valósítja meg.

Az alkalmazásom egy fő modullal rendelkezik, illetve létrehoztam egy külön modult, amely az Angular Material komponensek és függőségek behúzásáért felelős. Ezt csupán abból az okból tettem, hogy átlátható maradjon a fő modul kódja.

3.3 Mobil kliens alkalmazás architektúrája

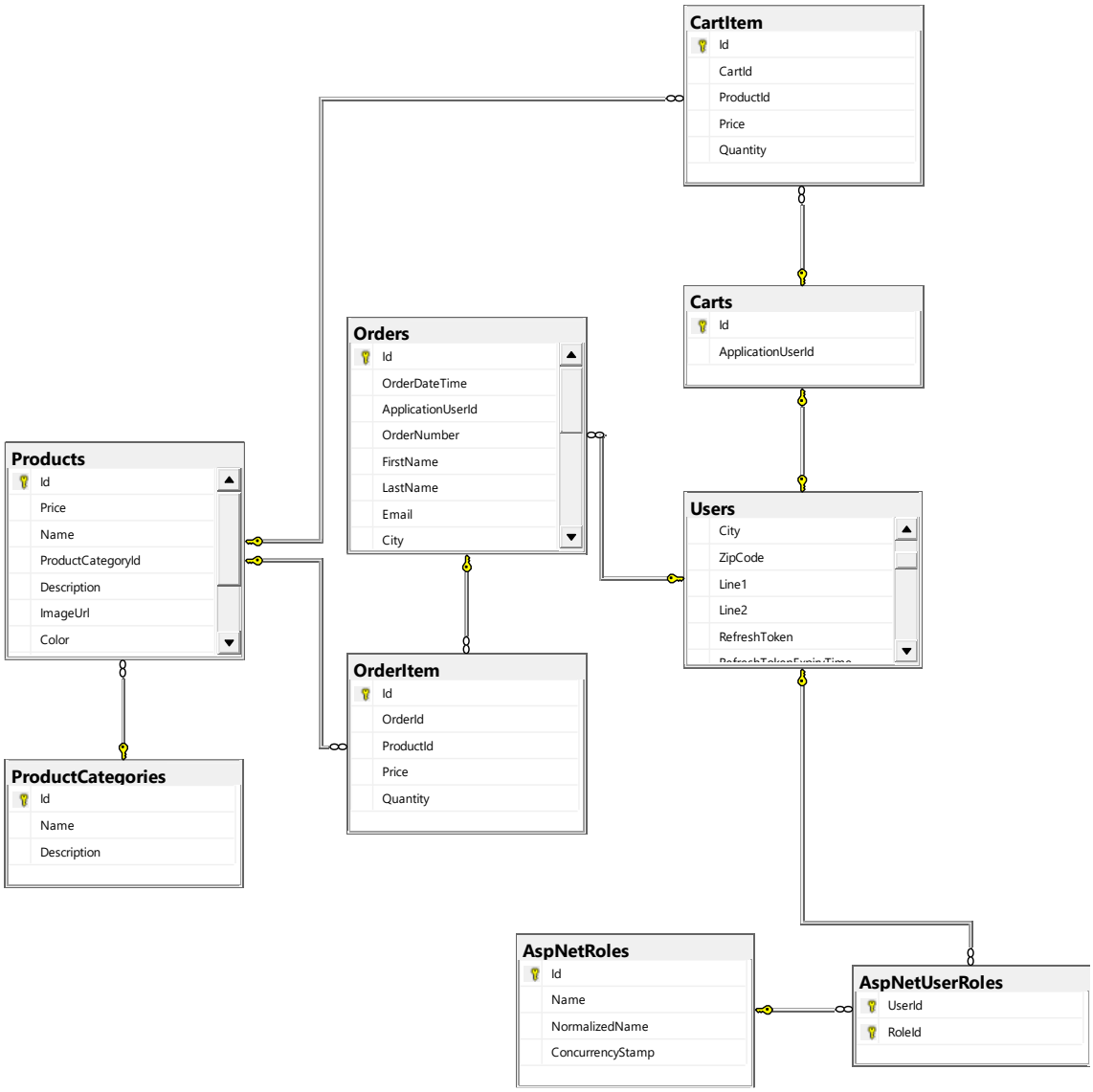
4 Backend alkalmazás

Ebben a fejezetben ismertetem a backend alkalmazás fejlesztésének és tervezésének fontosabb aspektusait, a lényegesebb fejlesztői döntéseket és az érdekesebb megoldásokat.

4.1 Adatbázis felépítése

Az adatok tárolására MSSQL adatbázist használtam, melyben a 2. ábraán látható struktúra alakult ki. A Code-First megközelítést alkalmazva a domén entitások és a kontextus osztály implementálása, konfigurálása után a séma elkészítését az Entity Framework technológiára bíztam.

A webshopnak szüksége van a leadott rendelések adatainak tárolására. Ezt az adatbázisban az *Orders* tábla



2. ábra - Az adatbázis struktúrája

4.2 Rétegek

4.2.1 Domén entitások rétege

Ebben az alfejezetben az adatmodell és azon entitásai kerülnek ismertetésre, ami alapján az adatbázis struktúrája is kialakult. Nem sorlom fel az összes tulajdonságot, amikkel ezek az entitások rendelkeznek, csak azokat, amelyek az alkalmazás szempontjából fontosabb szerepet játszanak.

4.2.2 Adatelérési réteg (Data Access Layer)

Az adatelérési réteg feladata az adathozzáférés biztosítása a felsőbb rétegek számára. Ez a réteg tartalmazza az ApplicationDbContext osztályt, amely az IdentityDbContext osztályból származik. Ez reprezentálja az adatbáziskapcsolatot, és arra használtam, hogy lekérdezzek, illetve adatokat mentsek az adatbázisba. A DbContext osztály a "Unit of work" és a "Repository" minták kombinációja. Itt definiáltam az adatbázisomban lévő táblákat, és az entitások konfigurációit.

Az adatelérési réteg tartalmazza továbbá az alkalmazás Repository osztályait. Ezek az osztályok az ApplicationDbContext osztály példányának segítségével kommunikálnak az adatbázissal.

Az adatelérési rétegben végeztem el az adatbázis seed-elését, azaz alapadatokkal való feltöltését is. Ez a fejlesztési és tesztelési folyamatokat könnyítette meg. Ezt a ModelBuilderExtensions osztályban valósítottam meg.

4.2.3 Szolgáltatások rétege

Az üzleti logika megvalósítását tartalmazza, itt találhatók az alkalmazás lényegi funkcionalitását nyújtó osztályok. Ebben a rétegben vannak definiálva a DTO-k, az ezekhez tartozó validátorok és a ViewModel-lek is.

A DTO-k, ViewModel-lek és a domain entitások között szükség van valamilyen leképezésre. Amikor például egy lekérdező kérést szolgálunk ki, akkor a megfelelő domain entitásokból kell ViewModel-leket készíteni és ezeket visszaküldeni a válaszban. Amikor pedig egy domain entitást módosító kérést kell kiszolgálunk, akkor a DTO-ból kell a módosításnak megfelelő domain entitást előállítani. Ezeket az átalakításokat időpazarlás minden alkalommal manuálisan elvégezni, hiszen nagyon repetitív feladatról van szó. Ezen feladat elvégzésére az AutoMapper könyvtár szolgáltatását vettem igénybe, mely előre definiált mappelési szabályok alapján elvégzi helyettem az átalakítást.

Az üzleti logikát megvalósító kódot Service osztályokba szerveztem, a webes Controllerek a megfelelő Service interfészen keresztül történő meghívásával tudják igénybe venni az általa nyújtott szolgáltatásokat. A szolgáltatások működésük során lekérdeznek, létrehoznak, módosítanak, illetve törölnek domain entitásokat az üzleti logikai szabályoknak megfelelően. Ezen műveletek elvégzéséhez az adatelérési réteg Repository osztályait használják, amelyek az adatbáziselérést absztraktálják.

4.2.4 Web réteg

A web rétegben található Controller osztályok felelősek a klienstől érkező HTTP kérések fogadásáért, feldolgozásáért, majd a megfelelő válasz visszaküldéséért.

3. ábra - OpenApi leíró

A Controller osztályokból a Swashbuckle könyvtár segítségével automatikusan egy OpenAPI specifikációnak megfelelő Swagger JSON leírót is generáltam. A könyvtár figyelembe veszi a web API alkalmazásban lévő route-okat, controllereket és modell osztályokat. Emellett a könyvtár rendelkezik egy Swagger UI eszközzel, amely a létrehozott Swagger JSON-ből képes készíteni egy gazdag felhasználói felületet, amelyen keresztül átláthatóan meg lehet tekinteni a web API végpontjait, illetve akár ki is lehet próbálni őket teszt kérések indításával.

5 Webes kliens

6 Mobil kliens

7 Irodalomjegyzék

[1] „StackOverflow 2021 Developer Survey,” 2021. [Online]. Available: https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies. [Hozzáférés dátuma: 22. május 2022.].

[2] „Git,” [Online]. Available: https://git-scm.com/. [Hozzáférés dátuma: 22. május 2022.].

[3] „Visual Studio,” Microsoft, [Online]. Available: https://visualstudio.microsoft.com/. [Hozzáférés dátuma: 22. május 2022.].

[4] „C# documentation,” Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/. [Hozzáférés dátuma: 22. május 2022.].

[5] „ASP.NET Documentation,” Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/?view=aspnetcore-6.0. [Hozzáférés dátuma: 22. május 2022.].

[6] „SQL Server 2019,” Microsoft, [Online]. Available: https://www.microsoft.com/en-gb/sql-server/sql-server-2019. [Hozzáférés dátuma: 22. május 2022.].

[7] „Download SQL Server Management Studio (SSMS),” Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15. [Hozzáférés dátuma: 22. május 2022.].

[8] „Entity Framework Core,” Microsoft, [Online]. Available: https://docs.microsoft.com/en-us/ef/core/. [Hozzáférés dátuma: 22. május 2022.].

[9] Swashbuckle.AspNetCore. [Online]. Available: https://github.com/domaindrivendev/Swashbuckle.AspNetCore. [Hozzáférés dátuma: 22. május 2022.].

[10] „OpenAPI Specification,” [Online]. Available: https://swagger.io/specification/. [Hozzáférés dátuma: 22. május 2022.].

[11] J. Skinner, „FluentValidation,” [Online]. Available: https://docs.fluentvalidation.net/en/latest/. [Hozzáférés dátuma: 22. május 2022.].

[12] „AutoMapper,” [Online]. Available: https://automapper.org/. [Hozzáférés dátuma: 22. május 2022.].

[13] M. Fowler, „Inversion of Control Containers and the Dependency Injection pattern,” [Online]. Available: https://martinfowler.com/articles/injection.html. [Hozzáférés dátuma: 22. május 2022.].

[14] „Autofac,” [Online]. Available: https://autofac.org/. [Hozzáférés dátuma: 22. május 2022.].

[15] „Visual Studio Code,” Microsoft, [Online]. Available: https://code.visualstudio.com/. [Hozzáférés dátuma: 22. május 2022.].

[16] „TypeScript,” Microsoft, [Online]. Available: https://www.typescriptlang.org/. [Hozzáférés dátuma: 22. május 2022.].

[17] „Angular,” Google, [Online]. Available: https://angular.io/. [Hozzáférés dátuma: 22. május 2022.].

[18] „Angular Material,” Google, [Online]. Available: https://material.angular.io/. [Hozzáférés dátuma: 22. május 2022.].

[19] „Material Design,” Google, [Online]. Available: https://material.io/design. [Hozzáférés dátuma: 22. május 2022.].

[20] „Node Package Manager,” [Online]. Available: https://www.npmjs.com/. [Hozzáférés dátuma: 22. május 2022.].

[21] „Android Studio,” Google, [Online]. Available: https://developer.android.com/studio. [Hozzáférés dátuma: 22. május 2022.].

[22] „Kotlin,” JetBrains, [Online]. Available: https://kotlinlang.org/. [Hozzáférés dátuma: 22. május 2022.].

Függelék