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

Berta Balázs

DIPLOMATERV 1

KONZULENS

Benedek Zoltán

BUDAPEST, 2019

Tartalomjegyzék

1 Technológiák bemutatása

Dolgozatom ezen fejezetében szeretném részletesebben bemutatni az általam felhasznált technológiákat. A mai rendszerek komplexitásai miatt szükségszerű, hogy ne mindent a legkisebb elemektől kezdjünk el felépíteni, hanem felhasználjuk a már létező megoldásokat, eszközöket. A webfejlesztés a szoftverfejlesztés egy nagy ága, ezáltal számos segédeszköz született hozzá.

1.1 ECMAScript

Az ECMAScript egy szabványosított programozási nyelv meghatározás, mely tartalmazza a nyelv szintaxisát, szemantikáját, könyvtárait és az egyes kiegészítő technológiákat, amiket a nyelv támogat. Azért született, hogy egységesítse a különböző JavaScript implementációkat. Főként a kliens oldali webes alkalmazások fejlesztéseben használatos, de egyre elterjedtebb szerver alkalmazások írásában is, melyhez a Node.js nyújt segítséget.

1.1.1 JavaScript

A JavaScript (röviden JS), egy magas szintű, interpretált[1] programozási nyelv, mely megfelel az ECMAScript specifikációjának. A HTML[2] és CSS[3] mellett a webfejlesztés egyik magja. Célja gazdag és interaktív alkalmazások készítése.

Támogatja az esemény alapú, funkcionális illetve az imperatív programozási stílusokat. Eszközöket nyújt alap típusokkal való munkára, úgy mint, szövegek, tömbök, dátumok, reguláris kifejezések[4] és a DOM[5] kezelésére. Azonban a nyelv nem tartalmaz

[1] Az interpreter (program) futtatókörnyezetként viselkedik, saját maga hajtja vére a kódból kiolvasott parancsoknak megfelelő műveleteket.

[2] Hypertext Markup Language, leíró nyelv webalkalmazások készítéséhez

[3] Cascading Style Sheets, stíluslap nyelv, dokumentumok megjelenésének leírására

[4] Karakter sorozatok, keresési minták meghatározására

[5] Document Object Model, elemek fa struktúrában való kezelése

I/O, hálózati, tárolási, illetve grafikai eszközöket. Ezekhez a host eszköz beépített szolgáltatásait használja fel. A nyelv dinamikusan típusos.

A webfejlesztés egyik nagy kihívása volt mindig is a különböző fajta böngészők, máshogy oldottak meg egyes dolgokat a saját implementációikban. Mára minden modern böngésző támogatja a JavaScript beépített interpreterrel.

1.1.2 TypeScript

A TypeScript egy nyílt forráskódú programozási nyelv, melyet a Microsoft hozott létre és tart karban. Ez egy szigorú szintaktikai részhalmaza a JavaScriptnek és statikus típusosságot ad a nyelvnek.

Az ebben készített alkalmazások visszafordulnak JavaScriptre. A visszafordításra több lehetőség is van. Alap esetben a TypeScript Checker használatos.

A statikus típusosságot leíró fájlokon keresztül éri el, amiben típus információk vannak megadva a létező JavaScript könyvtárakról, hasonló mint a C++ header állományok.

Fő előnyei a JavaScripthez képest:

- Típus annotáció / fordítás idejű típus ellenőrzés,
- Típus következtetés,
- Interfészek,
- Felsorolás típusok (Enum),
- Generikus típusok,
- Névterek,
- Asyn/ await programozási minta.

1.2 Angular (Angular 2+)

Az Angular egy TypeScript alapú nyílt forráskódú alkalmazásfejlesztési keretrendszer. A korábbi AnglarJS teljesen újragondolt változata.

Ez a keretrendszer főként webfejlesztésre szolgált, de mára már cross platform[6] lett. Segítségével progresszív web- (PWA), natív mobil- és asztali alkalmazások egyaránt készíthetők.

Mivel ez egy keretrendszer, ezért feltétlenül szükséges kitérni az általa nyújtott lehetőségekre.

1.2.1 Teljesítmény

Az egyik fontos eszköze a kód generálás. A fejlesztők által létrehozott mintákat (template) olyan kóddá alakítja, ami optimalizálva van a mai JavaScriptes virtuális gépekre.

Az elkészült alkalmazás univerzális, kiszolgálhatja szinte bármilyen szerver, NodeJs, .NET, PHP. Előkészíti az utat a SEO-ra optimalizáló alkalmazásoknak is.

Az egyes JavaScriptes alkalmazásoknál meg kell várni míg a teljes szkriptek betöltenek. Az Angular megoldja azzal, hogy felosztja a kódot, és csak azok a részletek töltődnek be, amelyek szükségesek a felhasználók kérésének kiszolgálásához.

1.2.2 Produktivitás

UI nézetek gyors létrehozását, fejlesztését adja a beépített egyszerű és hatékony mintakezelésével. A parancssori támogatottsága támogatja a gyors buildelést, komponensek létrehozását, tesztek futtatását és az azonnali telepítést. A fejlesztő környezet gazdag segítséget nyújt a fordításban, hibák megtalálásában.

Egy szoftver teljes életciklusának levezetéséhez eszközöket nyújt. Beépített teszt lehetőség Karma segítségével. A telepítés is egyszerű a beépített konfigurációs lehetőségekkel.

[6] A keresztplatformos szoftverfejlesztés egy olyan folyamat, amikor egy alkalmazás egyszerre több platformra készül el

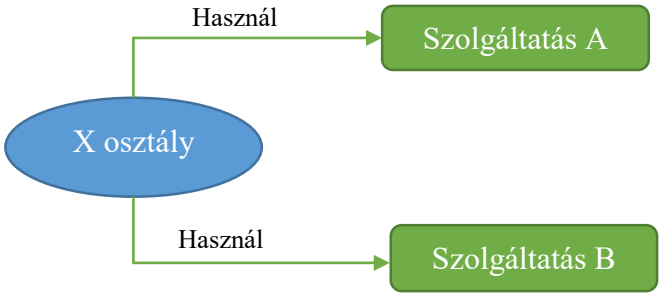
1.2.3 Felépítés

1.2.3.1 Adatkötés

1.2.3.2 Függőségek az osztályok és szolgáltatások között

Ebben a részben egy olyan problémakört mutatok be egy egyszerű példán, ami az én alkalmazásomban is megjelenik és azokat próbáltam csökkenteni szoftverfejlesztési módszerekkel.

Legyen egy X nevű osztályunk, ami kettő vagy több szolgálatatást felhasznál és ezeket ismerni kell tervezési időben.



1. ábra: Az X osztály függőségei

Ennél a megvalósításnál az alábbi problémák merülhetnek fel:

- fordítási időben szükségünk van a függőségek implementációira,
- ha megváltozik a szolgáltatás, vagy azt egy másikkal helyettesítjük, akkor módosítani kell a használó osztályok kódjait,
- benne lesz az osztályokban a függőségek igazgatásáért felelős kód (létrehozás, megtalálás).

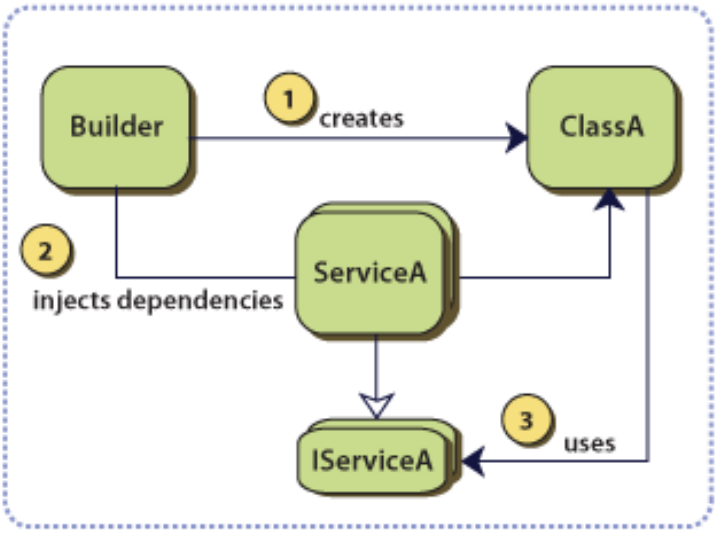
A cél tehát az, hogy ezek a megkötések megszűnjenek, az osztályok legyenek különválasztva a saját függőségeitől. Erre olyan módszerek állnak rendelkezésre, amelyekben egy külső eszköz, komponens szolgáltatja a függőségeket.

Vannak hátrányai is ennek a megközelítésnek, mint például a kód komplexitásának növekedése vagy, hogy biztosnak kell lennünk abban, hogy a keretrendszer számára elérhető a függőség, amit injektálnia kell. Megvalósítására több

implementáció is létezik, ilyenek például a gyártási minta, a szerviz lokátor minta, a függőség injektálás, stratégia tervezési minta stb.

1.2.3.3 Függőség injektálás (Dependency Injection, DI)

Az előbb felvázolt példára nyújt egy megoldást a függőség injektálás. Ahelyett, hogy inicializáljuk a függő szolgáltatást, deklaratívan leírjuk, hogy mire van szüksége. Egy Builder objektum fogja ezután az osztályokat létrehozni és átadni nekik, amikre szükségük van [16].



2. ábra: Függőség injektálás minta működése

1.3 Entity Framework Core

Az Entity Framework Core (EF Core) egy könnyebb, bővíthetőbb, és a platform független verziója a nagy Entity Framework (EF) adat elérési technológiának. Ez egy object-relational mapper (ORM), ami lehetővé teszi, hogy az adatbázis elemeit .NET objektumként használjuk. Ezzel kiváltható sok alacsony szintű adatelérési kódolás, ami egyébként szükséges lenne. Az Entity Framework Core sok fajta adatbázis motort támogat. Ezek közül én az SQLite [18] verziót választottam a mobil klienshez, mivel az szinte minden mobil platformon elérhető, illetve MSSQL adatbázis-kezelő rendszert a szerverhez.

A nagy Entity Framework háromfajta fejlesztési szemléletet támogat alkalmazások készítéséhez úgy, mint a Code First, Model First és Database First megközelítés. Ezek közül Entity Framework Core jelenleg csak az első változatot

biztosítja. Ez esetünkben annyit jelent, hogy először létre kell hozni a modell osztályainkat és ez alapján az adatbázist generáltatni a keretrendszer segítségével.

Az entitások között több fajta kapcsolat lehet, ezek megadására több lehetőséget is biztosít az Entity Framework Core. Az első és legegyszerűbb megoldás a navigációs tulajdonság alapú leképezés. Ezek a kapcsolatok úgy működnek, mint az adatbázisban a külső kulcsok. Ezen felül létezik az annotációs megoldás, ahol az entitás osztályainkat annotáljuk fel a megfelelő attribútumokkal. A harmadik megoldási lehetőség az ún. Fluent API, ahol az adatbázis kontextusunk létrehozásában kell definiálni a megfelelő relációkat, tábla leképezéseket.

1.4 ASP.NET Core

Az ASP.NET Core egy keresztplatformos, nagyteljesítményű és nyílt forráskódú keretrendszer, modern, felhő alapú, internetes alkalmazások készítésére. Ez a nagy ASP.NET újra tervezése szerkezeti változásokkal, amiknek köszönhetően soványabb és modulárisabb a rendszer. Számos előnnyel rendelkezik:

- beépített függőség injektálás (lásd 1.2.3.3),
- könnyű, nagyteljesítményű, moduláris HTTP kérés csővezeték,
- IIS és ön-hosztolásra is képes saját folyamatban(process),
- fejleszthető több operációs rendszeren is (Windows, macOS, Linux),
- nyílt forráskódú,
- stb.

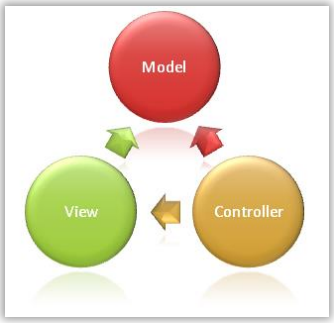
Az ASP.NET Core részei NuGet csomagokon[7] keresztül érhetők el, ezzel is lehetővé téve az optimalizálást, mivel elég csak a szükséges csomagokara hivatkozni.

Az ASP.NET Core MVC sok szolgáltatást nyújt a fejlesztőknek, ezek közé tartozik az út kezelés (routing), modellkötés (model binding), modell validáció, függőség injektálás, szűrők stb. Ezek közül számos komponenst felhasználtam az alkalmazásom elkészítése során.

[7] Ez egy csomagkezelő rendszer .NET-es projektek számára.

1.4.1 Modell-nézet-vezérlő minta

Az modell-nézet-vezérlő (model-view-controller, MVC) minta három komponensből épül fel: modell, nézet és vezérlő. Segítségével elérhetjük a szerepkörök különválasztását. A felhasználó kérései a vezérlőhöz vannak irányítva, ami felelős a modellel való munkáért. A vezérlő választja ki azt a nézetet, ami megjelenik az ügyfélnek és biztosítja az adatot, ami szükséges a felületen való megjelenítéshez.



3. ábra: MVC komponensei és hivatkozásaik egymásra [20]

1.4.2 RESTful webszolgáltatás

A RESTful webszolgáltatás egy olyan webszolgáltatás, ami a REST (Representational State Transfer) szoftver architektúrára épül. Ez hat megkötést ír le, amelyek a következők:

- Egységes interfész,
- Állapotmentesség,
- Gyorsítótárazható,
- Kliens-szerver architektúra,
- Réteges felépítés,
- Igényelt kód (ez a megkötés opcionális).

A HTTP ma már nem csak HTML oldalak kiszolgálására létezik. Ez egy erőteljes platform Web API készítésére, felhasználva az igéit (GET, POST, PUT stb.) és néhány egyszerű elgondolást, mint például az URI és a fejlécek. Az ASP.NET Core Web API egy komponens halmaz, ami leegyszerűsíti a HTTP programozást.

2 Irodalomjegyzék

[1] K. Nahtkasztlija, „Az idegen szavak toldalékolása,” június 2009. [Online]. Available: http://www.pcguru.hu/blog/kredenc/az-idegen-szavak-toldalekolasa/5062.

[2] P. Koopman, „How to Write an Abstract,” október 1997. [Online]. Available: https://users.ece.cmu.edu/~koopman/essays/abstract.html. [Hozzáférés dátuma: 20 október 2015].

[3] W3C, „HTML, The Web’s Core Language,” [Online]. Available: http://www.w3.org/html/. [Hozzáférés dátuma: 20 október 2015].