VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2018 Konstantin Kozhukhov

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

FRONT-END FRAMEWORKY PRO ASP.NET CORE WEBOVÉ APLIKACE
FRONT-END FRAMEWORKS FOR ASP.NET CORE WEB APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Konstantin Kozhukhov**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Ing. Ivo Lattenberg, Ph.D.**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Konstantin Kozhukhov ***ID:*** 177266
Ročník: 3 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Front-end frameworky pro ASP.NET Core webové aplikace

POKYNY PRO VYPRACOVÁNÍ:

Vytvořte ASP.NET Core webovou aplikace sloužící jako informační systém skladu elektronických součástek. Použijte framework DotVVM. Aplikace bude umožňovat pracovat se skladem třem skupinám uživatelů. První skupinou budou administrátoři - spravují uživatele a jejich role (Administrátor, Skladník nebo Konstruktér). Druhou skupinou budou skladníci. Ti mohou zakládat nové součástky (definovat kategorii, parametry, přidat obrázek a datasheet). Dále mohou nastavovat stav skladu (počet součástek na skladě) a v neposlední řadě mají možnost vygenerovat objednávku součástek (textový soubor). Poslední skupinou budou konstruktéři, kteří mohou prohlížet databázi součástek, hledat dle různých parametrů či stáhnout datasheet.

DOPORUČENÁ LITERATURA:

[1] MACDONALD, Matthew, Adam FREEMAN a Mario SZPUSZTA. ASP.NET 4 a C# 2010: tvorba dynamických stránek profesionálně. Přeložil Jan POKORNÝ. Brno: Zoner Press, 2011. Encyklopedie Zoner Press. ISBN 978-80-7413-131-8.

[2] VIRIUS, Miroslav. C# 2010: hotová řešení. Brno: Computer Press, 2012. K okamžitému použití (Computer Press). ISBN 978-80-251-3730-7.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 29.5.2018

Vedoucí práce: doc. Ing. Ivo Lattenberg, Ph.D.
Konzultant:

prof. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

V této bakalářské práci se zabývám návrhem a realizací webové aplikace sloužící jako informační systém skladu elektronických součástek. V prvních dvou kapitolách této práce je zahrnuta teoretická analýza a popis technologií, které budou použity v aplikaci, zatímco poslední kapitola je věnována jejich implementaci. Cílem výsledného programu je jednoduchá demonstrace použití platformy ASP.NET Core s frameworkem DotVVM.

KLÍČOVÁ SLOVA

ASP.NET Core, aplikace, DotVVM, Entity Framework, LINQ, C#

ABSTRACT

In this bachelor thesis I deal with design and implementation of a web application serving as information system for storage of electronic components. The first two chapters of this work deal with the theoretical analysis and description of technologies that will be used in the application, while the last chapter is dedicated to implementation. The goal of the resulting program is a simple demonstration of the use of the platform ASP.NET Core with a framework DotVVM.

KEYWORDS

ASP.NET Core, aplication, DotVVM, Entity Framework, LINQ, C#

KOZHUKHOV, K. *Front-end frameworky pro ASP.NET Core webové aplikace*. Brno, Rok, 58 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: doc. Ing. Ivo Lattenberg, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Front-end frameworky pro ASP.NET Core webové aplikace“ jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu doc. Ing. Lattenbergovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora(-ky)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno ....

podpis autora(-ky)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM VÝPISŮ

ÚVOD

Informační systémy se dnes vyskytují téměř v každé společnosti. Ať už se jedná o školní portál, webové stránky pro nákup letenek nebo složitý bankovní systém. Pomáhají řešit administrativu, sbírat a analyzovat informace a také usnadňují práci uživatelům.

Autora této bakalářské práce zaujala možnost použití moderních technologií spojených s vývojem webových aplikací. Výsledkem této práci by měla být aplikace sloužící jako informační systém pro sklad elektronických součástek. Pro realizaci projektu budou použity platformy ASP.NET Core a JavaScript framework DotVVM. Dále bude použit dotazovací jazyk LINQ pro přístup do databáze MSSQL.

Celá aplikace bude napsána v jazyce C# s využitím objektově orientovaného přístupu. Předpokládá se tedy čtenářova základní znalosti při práci s objekty, stejně jako dostatečná znalost jiných programovacích jazyků jako Visual Basic, C++ nebo Java.

Práce bude rozdělena do tří kapitol. První popisuje platformu .NET a její technologie, které budou použity při vytváření aplikace, druhá popisuje webové technologie a architekturu frameworku DotVVM, ve třetí kapitole je popsána realizace a rozbor programu.

1 PLATFORMA .NET

Platforma .NET je softwarový produkty a nástroje pro vývoj aplikací vyvinutý firmou Microsoft. Platforma nabízí vývojářům možnost vytvářet aplikace v různých prostředích jako je desktop, web nebo mobilní vývoj. Základní komponentou platformy je .NET Framework, který se používá pro běh aplikací. Zahrnuje spouštěcí rozhraní a všechny základní knihovny. Pro tvorbu .NET aplikací je nejčastěji využíváno vývojové prostředí Visual Studio.[1]

1.1 ASP.NET

V této kapitole je popsaná ASP.NET platforma. Čtenář v ní nalezne informace o její struktuře a historickém vývoji.

1.1.1 Úvod a popis platformy

ASP.NET je webový framework pro vytváření webových aplikací pomocí HTML, CSS a JavaScript. Název je odvozen od staré technologie ASP - Active Server Pages (česky aktivní serverové stránky). Platforma obsahuje funkce, knihovny a hotové komponenty, které pomáhají vývojářům ulehčit práci se základními problémy, jako je cashování, autorizace, validace technologií v reálném čase, přístupy k databází apod. Hlavní programovací jazyky pro psaní ASP.NET aplikací jsou C# a Visual Basic. Po postupném zdokonalování v současnosti platforma zahrnuje tyto technologie:

- *Web Forms* je nejstarší technologie, která je určena pro vytváření webových formulářů. V této technologii je programátorovi k dispozici mnoho hotových komponentů, které lze přesunout na stránku a následně jim přiřadit logiku události pro pozdější interakci. Po vytvoření stránky server provede kompilaci HTML kódu s komponenty, najde a přidá všechny potřebné logické funkcí a zašle uživateli vygenerovanou HTML stránku. Tento způsob vývoje se podobá technologii Windows Forms pro vytváření desktopových aplikací.[2]

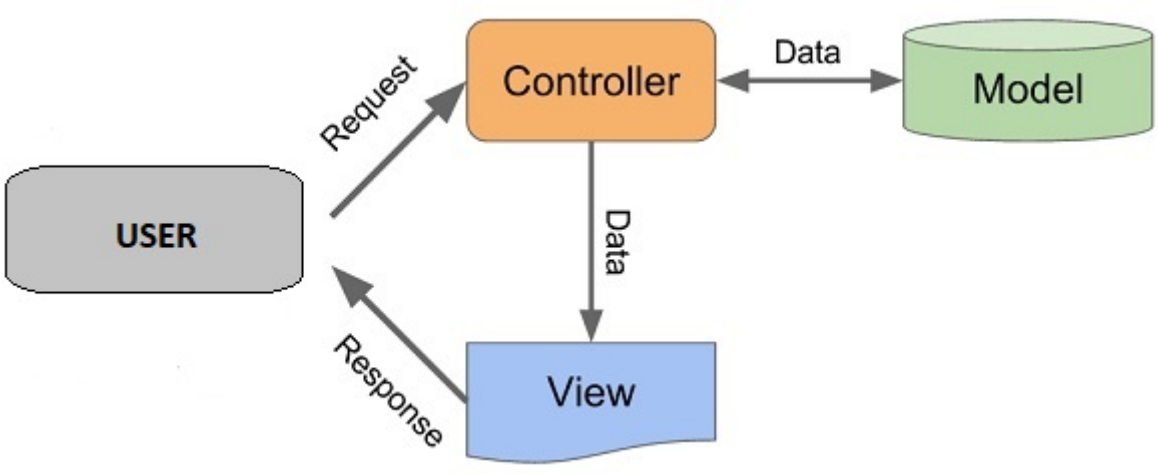
- *Web Pages* je technologie zaměřená na tvorbu jednoduchých dynamických webových aplikací. Využívá nástroj *Razor*, umožňující vkládat serverový kód (C# nebo Visual Basic) přímo do HTML stránky. Hlavním cílem technologie je zjednodušení vývoje webových stránek. [3] V současnosti je po takové technologii malá poptávka a stále více vývojářů se snaží používat framework,

který autor popisuje v následujícím odstavci.

- *MVC* - je framework pro vytváření webových aplikací pomocí implementace návrhového vzoru MVC [4]. Koncept předpokládá rozdělení aplikace na tři části:

  1. Model (*Model*) obsahuje logiku používaných dat, kde se provádějí výpočty, dotazy na databázi atd.
  2. Pohled (*View*) je vizuální část, uživatelské rozhraní aplikace. Obvykle se jedná o HTML stránku, kterou uživatel vidí v prohlížeči.
  3. Řadič (*Controller*) poskytuje spojení mezi uživatelem a systémem (pohledem a modelem). Typicky dostává data z uživatelského vstupu, zpracovává je a posílá modelu. V této bakalářské práci bude použit tento návrhový vzor z důvodu jeho funkčnosti a poptávky.

  Na obrázku 1.1 je znázorněn průběh fungovaní MVC. Uživatel odešle data na server, *controller* je příjme a provede jejich změny v modelu. Poté obdrží data nazpět a pošle je na *view*, ten pak zobrazí odpověď uživateli v prohlížeči. Tento vzor bude použitý v navrhované aplikaci.



Obr. 1.1: Blokové schéma návrhového vzoru MVC

1.1.2 ASP.NET Core

ASP.NET Core je dalším krokem ve vývoji platformy ASP.NET - není však jen další verzí, je to úplně nová platforma. V červnu 2016 byla vydána její první verze a v srpnu 2017 už byla představená verze druhá, která v době psaní této práce

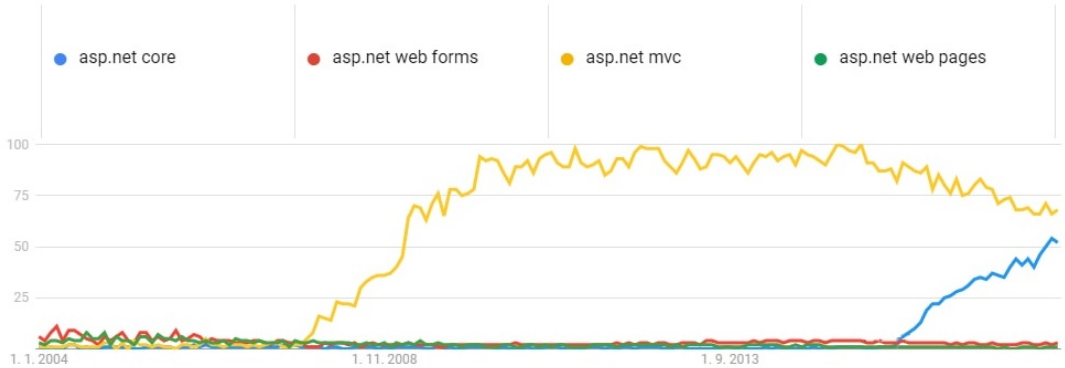
byla nejaktuálnější. ASP.NET Core je open-source technologie a všechny zdrojové soubory jsou k dispozici v depozitáři GitHub.

ASP.NET Core se stal cross-platformní technologií, která funguje na základních operačních systémech: Windows, Mac OS X a Linux. A díky nové modulární struktuře frameworku lze všechny potřebné komponenty webové aplikace stáhnout jako jednotlivé moduly pomocí správce balíčků Nuget. V tabulce 1.1 je znázorněno porovnávaní funkcí a možnosti nového ASP.NET Core a ASP.NET.

Tab. 1.1: Porovnání technologií ASP.NET a ASP.NET Core [5]

ASP.NET	ASP.NET Core
Funguje na Windows	Funguje na Windows, Linux, Mac
Používá Web Forms, SignalR, MVC, Web API, Web Pages	Používá Razor Pages (doporučeno), MVC, Web Api, SignalR
Jedna verze na zařízení	Více verzí na jednom zařízení
Dobrý výkon	Výkon je větší než ASP.NET
Používá .NET framework	Používá .NET Core nebo .NET framework

Na obrázku 1.2 vidíte graf srovnání dotazů ve vyhledávači Google od roku 2004 do roku 2018. Na horní křivce, která zobrazuje framework MVC, je vidět, že se jedná o nejpoužívanější ASP.NET framework. Stejně tak je patrné, že s příchodem ASP.NET Core v roce 2016 využití této technologie rychle roste a dnes se těší relativně stejné popularity jako MVC.



Obr. 1.2: Graf porovnání vyhledávaných výrazů v Google od roku 2004 až do roku 2018.

1.2 Entity Framework

Entity Framework (EF) je rozšíření ADO.NET[1], objektově orientovaná technologie od firmy Microsoft pro přístup k datům.

První verze EF vyšla v roce 2008 a měla omezenou funkčnost - základní podporu ORM[2] a jediný přístup ke komunikaci s databází - Database First. S vydáním verze 4.0 v roce 2010 se mnohé změnilo - od té doby se EF stal doporučenou technologií pro přístup k datům a do frameworku byly zavedeny nové možnosti interakce s databází – přístupy Model First a Code First.[6]

Další zlepšení funkčnosti se dostavilo s vydáním verze 5.0 v roce 2012. A konečně, v roce 2013 se objevil EF 6.0 s možností asynchronního přístupu k datům, s lepší podporou transakcí, procedur a s mnoha dalšími vylepšeními. Nejnovější verze v době psaní této práce byla EF 6.1.2.

Hlavním pojmem EF je **entita**. Entita představuje sadu dat, spojených s určitým objektem, proto v této technologii pracujeme s objekty, ne přímo s tabulkami. Entity mají klíče a mohou být spojeny asociativními komunikacemi jeden-k-mnoha, jeden-k-jednomu, mnoho-k-mnoho. Stejně jako v reálné databázi, probíhá komunikace mezi tabulkami přes cizí klíče.

Neméně důležitým pojmem je **Entity Data Model**. Tento model spojuje třídy entit se skutečnými tabulkami v databázi. Entity Data Model se skládá ze tří úrovní: *koncepční*, *úroveň úložiště dat* a *úroveň spojení* (*mapování*).[7]

- Na *koncepční* úrovni se definují třídy entit používaných v aplikaci.
- Úroveň *úložiště dat* určuje tabulky, sloupce, vztahy mezi tabulkami a typy dat.
- *Úroveň spojení* (*mapování*) slouží jako prostředník mezi dvěma výše popsanými, definuje mapování mezi vlastnostmi třídy entity a sloupci tabulek.

1.2.1 Způsoby interakce s databází

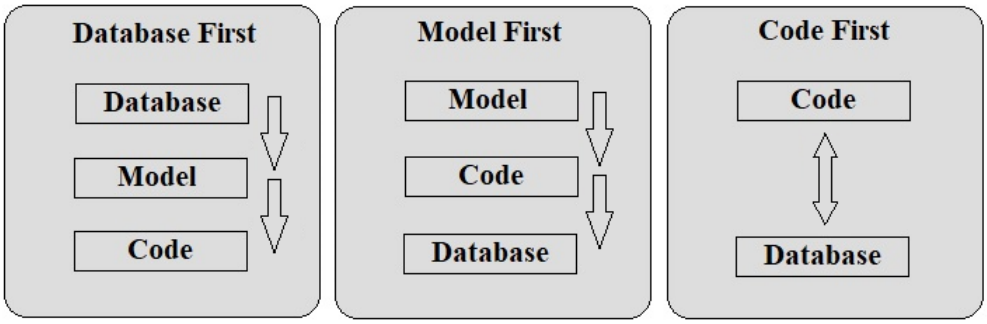
EF nabízí tři možné způsoby interakce s databází:

- **Database First**
  V tomto přístupu se na začátku vytváří databázová architektura s využitím různých nástrojů (např. SSMS), následně se v aplikaci generuje **.edmx** (Entity Data Model Wizard) - datový model databáze, který poskytuje grafické rozhraní pro komunikaci s databází v podobě grafů a objektový model ve formě vlastností a tříd C#.

[1] ADO.NET - technologie pro přístup k datům. Tato technologie představuje sadu tříd, pomoci kterých můžeme odeslat dotazy do databáze, nastavit připojení, obdržet odpověď od databáze a provádět řadu dalších operací.

[2] ORM (object relational mapping) je nastroj zajišťující komunikaci mezi databází a objekty v aplikaci. V .NET mezi největší ORM nástroje patří NHibernate a Entity Framework.

- **Model First**
  Nejprve se vytváří grafický model **.edmx** ve Visual Studio (na pozadí se vytvoří třídy C#). Výsledkem je schéma, podle kterého se následně vygeneruje databáze.
- **Code First**
  Při tomto přístupu **.edmx** se model vůbec nepoužívá a vývojář píše třídy a vlastnosti v C#, které budou odpovídat tabulkám. O generování databází se následně postará EF. Každá provedená změna v modelu se projeví i v databázi. V naší aplikaci budeme používat právě tento přístup.



Obr. 1.3: Schéma možných způsobů interakce s databází pomocí EF

1.3 LINQ

LINQ (Language Integrated Query) je jednoduchý dotazovací jazyk ke zdroji dat [8], integrovaný do jazyka C#. Jako zdroj dat může sloužit objekt implementující rozhraní IEnumerable (například standardní kolekce nebo pole), sada dat DataSet, XML dokument atd. Bez ohledu na typ zdroje LINQ umožňuje použít ke všem stejný přístup pro dotazování. Tento jazyk zjednodušuje práci při psaní SQL dotazů a snižuje množství kódu. Existuje několik různých typů LINQ:

- **LINQ to Objects**: používá se pro práci s poli a kolekcemi objektů.
- **LINQ to Entities**: používá se při přístupu k databázi. Zde se využívají ORM a EDM.
- **LINQ to Sql**: používá se pro dotazování na databázi v MSSQL Server.
- **LINQ to XML**: umožňuje práci s XML dokumenty.
- **LINQ to DataSet**: používá se při práci s objektem DataSet.

1.3.1 Seznám klíčových slov v LINQ

Klíčová slova v LINQ jsou podobná obyčejné syntaxi SQL - jsou to **SELECT, FROM, WHERE, ORDER BY, JOIN, MIN** atd. Ale mají jiné pořadí a strukturu při psaní dotazu. V následujícím kódu se podíváme na dotazovací výraz:

Výpis 1.1: Příklad LINQ dotazu

```
   List<User> users = new List<User>
{
    new User {Name="Adam", Age=22, Languages = new
    List<string> {"čeština", "angličtina" }},
    new User {Name="Michal", Age=37, Languages = new
    List<string> {"čeština", "francouzština" }},
    new User {Name="Tomáš", Age=49, Languages = new
    List<string> {"čeština", "němčina" }},
};
    var selecteduser = from u in users
                       from l in user.Languages
                       where u.Age < 49
                       where l == "angličtina"
                       select user;

    foreach (User user in selectedUsers)
    Console.WriteLine("{0}-{1}", user.Name, user.Age);

// vysledek Adam-22.
```

Předpokládejme, že máme třídu **User**, která má vlastnosti **Name**, **Age** a seznam **Languages**. Vytvoříme seznam users a přes konstruktor třídy User přidáme tři nové objekty a nastavíme jim vlastnosti. Poté vybereme uživatele, který je mladší 49 let a mluví anglicky. V kódu používáme dvakrát klíčový vyraz **Where** z důvodu nutnosti přístupu k seznamu **Languages** nějakého objektu. Následně pomocí cyklu **Foreach** vypíšeme uživatele na obrazovku.

Velkou výhodou v LINQ je styl psaní dotazu pomocí lambda výrazů. Značně zjednodušuje rychlost vývoje a umožňuje psát flexibilní kód. Následující kód je stejný jako výše uvedený, ale zapsaný pomocí lambda výrazu.

Výpis 1.2: Příklad LINQ dotazu pomocí lambda výrazu

```
var selectedUsers = users.SelectMany(u => u.Languages,
    (u, l) => new { User = u, Lang = l })
    .Where(u => u.Lang == "angličtina" && u.User.Age < 49)
    .Select(u=>u.User);
```

Seznam klíčových slov[9]:

- **From** – zdroje dat. Ukázka:
  from p in Predmety
- **Where** - podmínka. Ukázka:
  from p in Predmety where p.Nazev <> „XAN4“ select p.Nazev
  Vybereme předměty, které se nerovnají předmětu XAN4.
- **Select** - upřesnění, co konkrétně chceme vypsat.
  “*” - projde všechny sloupce v tabulce / “název sloupce” – konkretní sloupec
- **Distinct** - vypisuje každý unikátní prvek pouze jedenkrát, čímž se můžeme vyhnout duplicitě dat.
- **Orderby (asccending/descending)** - řazení dat (vzestupně/sestupně). Ukázka:
  from p in Predmety orderby p.Rocnik select p.Nazev.
  Vypíšeme všechny názvy předmětu a seřadíme je vzestupně podle ročníku.
  Ascending - jedná se o defaultní hodnotu, nemusíme ji tedy psát.
- **Group** - seskupovaní dat. Ukázka:
  from sudent in Students group student.Name by student.Rocnik
  Vypíšeme jména student a rozdělíme jejich do skupin podle ročníku.
- **Join** - spojení tabulek. Ukázka:
  from student in Students join predmet in Predmety on student.id_predmet equals predmet.id
  Zde se spojují tabulky Studenti a Predmety podle cizího klíče id_premdety a primárního klíče id.
- **Skip** - přeskočit prvních několik záznamů v tabulce. Ukázka:
  (from p in Predmety select p.Nazev).Skip(10)
  Vybereme všechny názvy předmětu po prvních deseti.
- **ToArray** - konvertování vybraných dat do pole. Ukázka:
  (from p in Predmety orderby p.Rocnik select p.nazev).ToArray()
- **ToList** - konvertování vybraných dat do seznamu List<T>. Ukázka:
  (from p in Predmety orderby p.Rocnik select p.Nazev).ToList()
- **Contains** - vrací true/false pokud vybraná data obsahují určité číslo/text. Ukázka:
  From p in Predmety where p .Contains(„BOOP“) select p.Nazev
- **Count** - počet záznamů. Ukázka:
  (From student in Students select student.Name).Count();

2 WEBOVÉ TECHNOLOGIE

V této kapitole budou popsány technologie pro vytváření webových rozhraní. Od struktury DOM do seznámení čtenáře s SPA, JavaScript, frameworkem DotVVM. Všechny tyto technologie budou použity v aplikaci, která je popsána ve třetí kapitole.

2.1 DOM

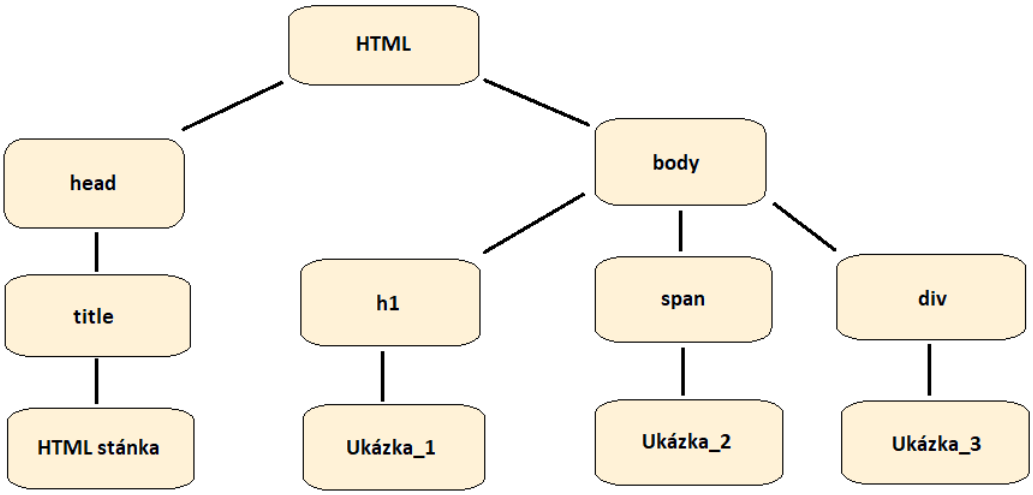
Hlavním nástrojem pro tvorbu dynamických změn na stránce je DOM (Document Object Model) – objektový model, používaný pro XML/HTML-dokumenty. Podle DOM modelu má každý dokument stromovou strukturu. Každý HTML tag je uzlem a objektem reprezentujícím část dokumentu[10].

Webová stránka – je dokument, který může být představen ve formě HTML kódu. DOM nabízí jiný způsob reprezentace, uchovávání a správy tohoto dokumentu. Podporuje objektově orientovanou reprezentaci webové stránky tak, aby bylo možné ji změnit pomocí jazyka pro tvorbu scénářů typu JavaScript.
Jako vzor vezmeme obyčejnou HTML stránku:

Výpis 2.1: Vzor HTML kódu jednoduché webové stránky

```
<!DOCTYPE html>                                    1
<html>                                             2
    <head>                                         3
        <title>HTML stránka</title>                4
    </head>                                        5
    <body>                                         6
        <h1>Ukázka_1</h1>                          7
        <span>Ukázka_2</span>                      8
        <div>                                      9
            <p>Ukázka_3</p>                        10
        </div>                                     11
    </body>                                        12
</html>                                            13
```

Výše uvedený kód bude mít následující stromovou strukturu:



Obr. 2.1: Stromová struktura jednoduché webové stránky

Z obrázku výše je vidět, že všechny prvky jsou seřazeny v DOM hierarchickým způsobem, kdy každý prvek představuje samostatný uzel. To znamená, že každý prvek, například prvek **span**, představuje uzel. Ale také třeba text (Obr.2.1 **Ukázka_2**) uvnitř prvku představuje samostatný uzel, se kterým můžeme pracovat a dynamicky jej upravovat.

2.2 JavaScript

JavaScript je dominantní, objektově orientovaný webový skriptovací jazyk. Vyvinul ho Brendan Eich v roce 1995 ve společnosti Netscape jako skriptovací jazyk v prohlížeči **Netscape Navigator 2**. Původně se jazyk jmenoval LiveScript, ale díky tehdejší popularitě jiného jazyka Java, byl přejmenován na JavaScript. V listopadu roku 1996 společnost Netscape představila JavaScript ECMA International[1], aby vytvořil standardní specifikaci. Tato skutečnost vedla k oficiálnímu vydání specifikací jazyka ECMAScript v červnu 1997. [11]

Do dnešního dne bylo vyvinuto několik standardů jazyka ECMA. Aktuální verze standardu je ECMAScript 2017 (ES 8) [12]. Vývoj jazyka natolik pokročil, že se používá k vytvoření nejen webových stránek, ale i mobilních a desktopových aplikací

[1]ECMA International - (European Computer Manufacturers Association International) - mezinárodní organizace pro normalizaci informačních a komunikačních systémů, která byla založená v roce 1961. Sídlem organizace je Ženeva, Švýcarsko.

2.2.1 Web

Hlavním účelem JavaScriptu je tvorba dynamických webových stránek. Jeho velkou výhodou je absence nutnosti stálého připojení k serveru. Při otevření webové stránky uživatel stahuje spolu s HTML a CSS Javascript instrukce a celý kód je pak interpretován na straně klienta v prohlížeči. Mezi funkce, které JavaScript nabízí patří:

- Změna a odstraňovaní všech HTML prvků a atributů na stránce.
- Změna všech CSS stylů na stránce.
- Reagování na všechny existující události na stránce.
- Možnost vytvářet nové události v rámci stránky atd.

2.2.2 Server

V květnu 2009 platforma Node.js podnikla své první kroky ve světe vývoje serverových častí aplikací. Dnes bude velmi obtížné najít vývojáře, který by neslyšel o této technologii. Node.js umožňuje vytvoření webových serverů, síťových nástrojů a různých modulů pomocí JavaScriptu a jiných jazyků jako CoffeScript a TypeScript, které se pak kompilují do "čistého"JavaScriptu. Moduly poskytují přistup k síťovým protokolům (TCP, HTTP, DNS, UDP atd.), kryptografickým funkcím, datovému streamování a dalším základním funkcím. Jádrem platformy je open-source V8 od společnosti Google, který může běžet na operačních systémech Linuxu, MacOS a Microsoft Windows serverů. [13]

2.2.3 Mobil

V poslední době rychle roste oblast mobilního vývoje. Zvýšení výkonu zařízení a šíření standardu HTML5 přispělo k tomu, že pro vytváření aplikací pro mobilní zařízení můžeme taktéž použít JavaScript. Mezi nejpopulárnější frameworky patří Jquery Mobile, React Native, Angular a Meteor.

2.2.4 Desktop

V roce 2013 GitHub představil open-source framework pro tvorbu desktopových aplikací pomoci HTML, CSS a JavaScript, s názvem Elektron. Ten odlišuje fakt, že jako Java může běžet na libovolném operačním systému. Framework zahrnuje Node.js, pro práci se serverem, a Chromium pro vykreslování dat na obrazovku. Tato technologie rychle získala popularitu a začaly ji využívat velké společnosti. Mezi známými aplikacemi jsou: Skype a Visual Studio Code od Microsoft, Atom od GitHub, Slack, WhatsApp a další. [14]

2.3 MV* - návrhové vzory

Návrhové vzory (Design patterns) - jsou obecným řešením často se vyskytujících problémů nebo slouží jako šablona, která může být použita v různých situacích při návrhu software. Návrhové vzory můžou snížit celkovou velikost kódu díky zamezení opakování. V teto kapitole budou představeny tři hlavní návrhové vzory MVC, MVP a MVVM.

2.3.1 MVC

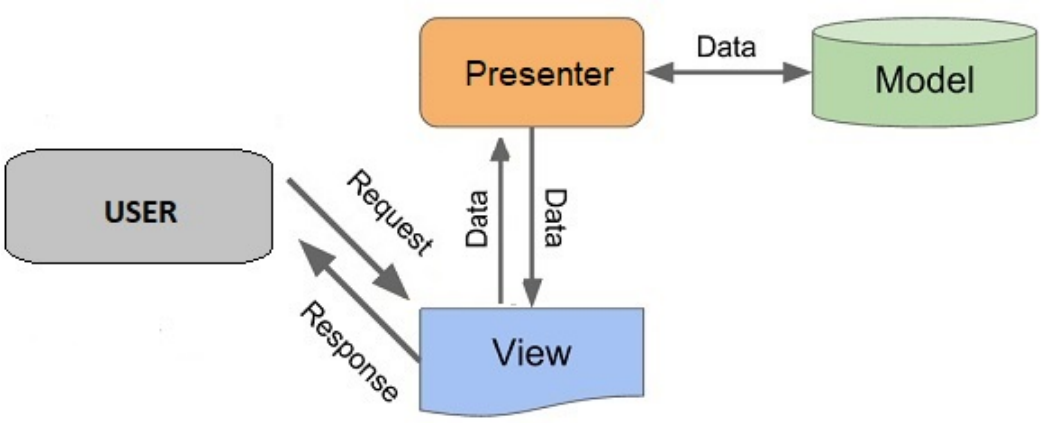
MVC (Model View Controller) je architektonický návrhový vzor, který zprostředkovává lepší aplikační organizaci prostřednictvím rozdělení uživatelského rozhraní a logiky do tří samostatných komponent: model (datový model), view (uživatelské rozhraní) a controller (řídící logika).

Návrhový vzor byl vytvořen norským počítačovým vědcem Trygve Reenskaugem v roce 1979, když pracoval na programovacím jazyku Smalltalk-80. [15] Podrobný popis všech tří komponent a jejich propojení nalezneme v kapitole 1.1.1.

2.3.2 MVP

MVP (Model View presenter) je návrhový vzor odvozený od MVC, jehož hlavním cílem je zlepšení prezentační logiky. Vznikl ve společnosti Taligent na počátku devadesátých let minulého století. [15]
V rámci frameworku je logika rozdělena následujícím způsobem:

- *Model* má podobnou funkčnost jako u MVC. Zajišťuje přístupy k datům a obsahuje logiku aplikace.
- *View* je uživatelským vstupem a výstupem. Spolupracuje přímo s funkcí *Presenter* prostřednictvím volání příslušných funkcí nebo událostí. Na rozdíl od MVC zde probíhá oboustranná komunikace *View* - *Presenter*.
- *Presenter* reaguje na události přicházející od uživatele a zajišťuje změny ve *View* nebo v *Modelu*. *View* a *Presenter* jsou od sebe zcela odděleny a veškerá komunikace probíhá pomocí rozhraní.



Obr. 2.2: Blokové schéma návrhového vzoru MVP

2.3.3 MVVM

MVVM (Model View View Model) je architektonický návrhový vzor umožňující oddělit logiku aplikace od vizuální části (View). Tento model byl představen zaměstnancem firmy Microsoft Johnem Grossmanem v roce 2005 jako modifikace MVP a byl původně zaměřen na vývoj WPF aplikací. [15] Návrhový vzor začal rychle získávat na popularitě v komunitě a brzy se začal používat i v jiných programovacích jazycích. Jedny z nejznámějších JavaScript frameworků realizujících MVVM šablonu jsou Knockout.js, Ember.js a Angular.js.
MVVM se skládá ze tří komponent: model (*Model*), pohled (*View*) a prostředník mezi View a Modelem (*ViewModel*):

- *Model* popisuje data používaná v aplikaci. Modely mohou obsahovat logiku, přímo související s těmito daty, například logiku validace vlastností modelu. Zároveň by model neměl obsahovat žádnou logiku, týkající se zobrazení dat a interakce s vizuálními ovládacími prvky.
- *View*, podobně jako MVC, definuje vizuální rozhraní, přes které uživatel komunikuje s aplikací.
- *ViewModel* spojuje *Model* a *View* přes mechanismus vázání dat. Pokud ve *ViewModelu* dojde ke změně, vlastnosti uživatelských prvků se aktualizují. Stejně je tomu i naopak: pokud dojde ke změně vlastností v prvku uživatelského rozhraní, *ViewModel* se automaticky aktualizuje. Této funkcionalitě se říká Two-way data binding (obousměrné vázání dat).

Obr. 2.3: Blokové schéma návrhového vzoru MVVM

2.4 JSON

JSON (JavaScript Object Notation) je odlehčený textový formát pro ukládání dat. Popisuje strukturu a organizaci dat v JavaScript. Jednoduchost JSON přivedla k tomu, že v současné době je nejpopulárnějším formátem pro přenos dat v prostředí webu, který nahrazuje jiný, kdysi populární formát XML. [16] Objekty JSON jsou velmi podobny objektům JavaScript, díky tomu, že JSON je podmnožinou JavaScriptu. Zároveň je důležité rozlišovat: JavaScript je programovací jazyk, JSON je formát dat. JSON podporuje tři datové typy: primitivní hodnoty, objekty a pole. Primitivní hodnoty představují standardní řetězce, čísla, hodnotu **null** nebo logické hodnoty **true** a **false**. Příklad formátu JSON: Výpis 2.2

Výpis 2.2: Zapis dat ve formatu JSON

```
Objekt v JavaScript:
var employee = {
    name: "Tomas",
    age: 40,
    isworking: true
}
JSON:
{
    "name": "Tomas",
    "age": 40,
    "isworking": true
}
```

Jak vidíme z kódu výše, v JSON jsou názvy vlastností v uvozovkách jako JavaScript řetězce. Avšak na rozdíl od objektů JavaScriptu, objekty v JSON nemůžou obsahovat funkce a proměnné.

2.5 Single Page Aplikace

Single Page Aplikace (jednostránkové aplikace) je typ webových aplikací nebo webových stránek, které používají jediný HTML dokument jako šablonu pro všechny webové stránky v aplikaci. Další potřebné soubory pro zobrazení obsahu jsou stahovány dynamicky pomocí AJAX[2] po interakci s uživatelem. Není nutné stále načítat stejná data, z čehož vyplývá zmenšení datového toku od uživatele k serveru. Aplikace načítá pouze ta data, která uživatel požaduje v danou chvíli. Tento princip vývoje bude použit v předkládané aplikaci. [17]

2.6 DotVVM

V roce 2005 se pro tvorbu webové aplikace používala populární platforma Web Forms. V té době by zkušenému vývojáři postačily znalosti jazyka C#, platformy .NET nebo HTML. Samozřejmě tyto aplikace nelze srovnávat s tím, co nám mohou dát moderní technologie, ale díky hotovým komponentům a funkcím zapouzdřeným do platformy byl proces vývoje jednoduchý na pochopení pro začínající vývojáře. [18]

Postupem času se moderní web vyvíjel, a objevily desítky nových frameworků a kniho-ven jako jsou Jquery, Angular.js nebo React.js. Dnes by vývojář musel znát minimálně některé z těchto uvedených frameworků, stejně tak jako knihovny pro běžné záležitosti. Velkým problémem však je, že většina vývojářů nerozumí JavaScriptu do hloubky. Dokáží jen deklarovat Jqeury funkce, ale nemají téměř tušení, jak tento jazyk na pozadí funguje. Problémem je i fakt, že většina frameworků obsahuje velké množství JavaScript kódu, který neděla nic zásadního, jen transformuje data, volá určité Web API, ošetřuje chyby nebo aktualizuje View. Ideálním řešením by mohla být nějaká konzistentní platforma, která by byla schopna řešit většinu JavaSript problému, byla by rozšiřitelná a dobře fungovala s platformou ASP.NET. [18]

Takovou platformou stal nový open-source framework DotVVM vytvořený českou společností Riganti. První verze frameworku byla představena v roce 2015, a všechny jeho zdrojové kódy jsou veřejně přístupné na GitHub. DotVVM aplikace obsahují hodně JavaScript kódu, nicméně vývojáři při tvorbě aplikacích nemusí napsat ani řádek JavaScriptu. DotVVM je inspirovaný ASP.NET Web Forms, a všechny JavaScript funkce jsou zapouzdřeny do komponent. Názvy komponent jsou podobné i v prostředí Web Forms, což děla framework více pochopitelným pro .NET vývojáře.

[2] AJAX (Asynchronous JavaScript And XM) je sada webových technologií pro tvorbu asynchronních webových aplikací. Pomocí ní můžeme aktualizovat jednotlivé části webové stránky bez nutnosti ji celou načítat znovu.

Také je třeba podotknout, že DotVVM funguje s poslední verzi ASP.NET Core 2.0. [18]

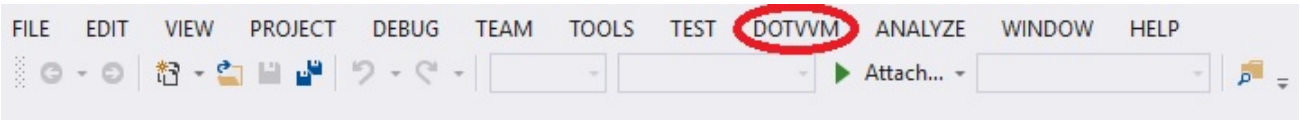
2.6.1 Princip fungování

DotVVM používá návrhový vzor MVVM, jehož ViewModel má dvě funkce. První obsahuje stav stránky (View), například hodnoty, zadané uživatelem do komponentu, musejí být zachovány ViewModelem, který reprezentuje stav stránky. Druhá obsahuje definice chování, které může uživatel vyvolat například kliknutím na nějaké tlačítko. Propojení je realizováno pomocí provázaní dat (data-binding). [19]

Na straně klienta se využívá knihovna Knockout.js, která implementuje návrhový vzor MVVM. Funkce a vlastnosti ve ViewModelu jsou napsány v jazyce C#. Abychom mohli tato data zobrazit ve View, DotVVM generuje C# kód do JavaScript. Výsledkem je HTML kód obsahující JavaScript se syntaxí Knockout.js. [20]

Ukázka webové aplikace využívající DotVMM

Před vlastním vytváření aplikace je třeba stáhnout DotVVM z oficiálních stránek[3] nainstalovat jej. Jakmile je instalace dokončena, v panelu Visual Studio se objeví nová položka DotVVM.



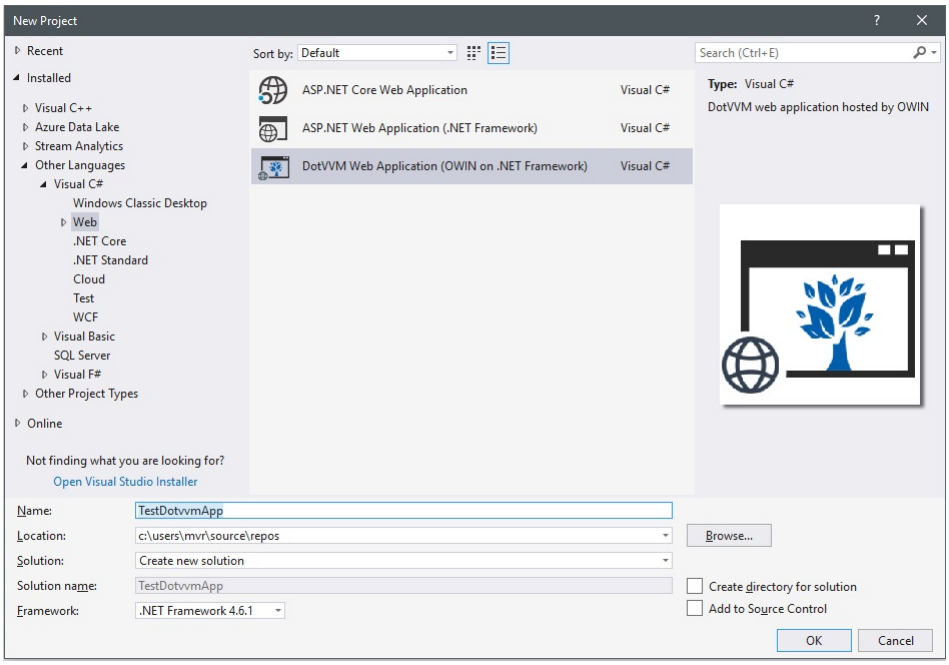
Obr. 2.4: Panel Visual Studio

Ve Visual Studiu založíme nový projekt: File – New – Project. Z nabídky vlevo vybereme záložku Web a v ní zvolíme typ projektu DotVVM Web Application. Do pole níže napíšeme název aplikace – „TestDotvvmApp“. Dialogové okno pro výběr typu projektu vidíme na obr. 2.5. Volbu potvrdíme stiskem tlačítka „OK“.

Po vytvoření aplikace DotVVM vytvoří několik složek a souborů. Nalezneme je vpravo v okně Solution Explorer. Hlavními jsou složky View a ViewModel (realizace MVVM patternu). Po otevření těchto složek vidíme, že ke každému .dothml souboru, který je stránkou (View), náleží jedna třída ze složky ViewModels. MasterPage.dotmaster je hlavní stránkou () šablony, od níž ostatní stránky dědí CSS styly a HTML (více informací viz kapitola 2.5). Startup.cs je soubor, který se stará o spouštění aplikace. Nyní si ukážeme, jak vytvořit postupně jednoduchou kalkulačku

[3] https://www.dotvvm.com/install

pomocí hotových DotVVM komponentů. Nejprve otevřeme soubor DefaultViewModel.cs. V textovém okně se zobrazí DefaultViewModel - třída, která se dědí od třídy DotvvmViewModelBase. DotvvmViewModelBase () poskytuje základní funkce frameworku. Odstraníme vlastnost Title a Konstruktor třídy, pak Number1, Number2, Result a čtyři funkce, které provádějí matematické operace a vracejí výsledek do vlastnosti Result, stejným způsobem jako je vidět na obr. 2.6.



Obr. 2.5: Panel Visual Studio

Otevřeme default.dothtml ve složce View a přidáme dvě komponenty TextBox a čtyři komponenty typu Button, podobně jako je zobrazeno na obr. 2.7 Komponenty v DotVVM mají vždy příponu dot, podobně jako je přípona asp vždy v ASP.NET Web forms. Pomocí **Click="command: Add()"** můžeme zavolat metodu Add() z View- model při kliknutí na tlačítko. Pomocí **Text="value: Number1"** můžeme svázat data získaná z Viewmodel a automaticky je aktualizovat na stránce.

Kliknutím na tlačítko F5 se spustí program a všechny dot komponenty se začnou generovat do formátu HTML. Otevře se také webový prohlížeč a na stránce se zobrazí dva textboxy a čtyři tlačítka. Poté, co uživatel přidá hodnoty do textboxů, klikne na jedno z tlačítek. Pomocí knihovny knockout se odešlou data na ViewModel, kde probíhá zpětné generování kódu do C#, stejně jako volání příslušných metod. Pro náhled na vygenerovaný HTML kód, stiskneme klávesu F12 a v prohlížeči otevřeme konzoli. Vyberme položku Network a obnovíme stránku. Po obnovení otevřeme soubor localhost, ve kterém je vygenerovaný HTML kód. Komponenty textbox budou změněny na syntaxi knihovny Knockout: **<input data-**

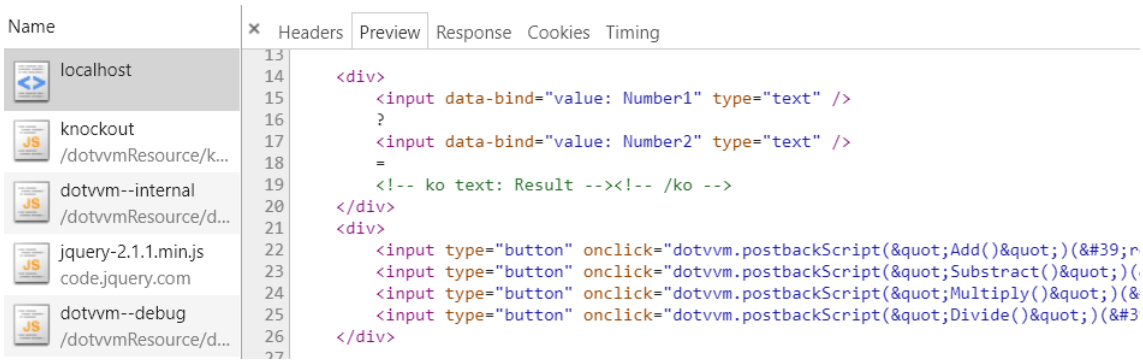
bind="value: Number1"type="text"/ > a komponenty button na **<input type="button"onclick=""value=""/>**. Příklad konzole je zobrazen na obr. 2.8.

```
namespace TestDotvvmApp.ViewModels
{
    public class DefaultViewModel : MasterPageViewModel
    {

        public int Number1 { get; set;}

        public int Number2 { get; set;}

        public int Result { get; set;}

        public void Multiply()
        {
            Result = Number1 * Number2;
        }

        public void Divide()
        {
            Result = Number1 / Number2;
        }

        public void Add()
        {
            Result = Number1 + Number2;
        }

        public void Substract()
        {
            Result = Number1 - Number2;
        }

    }
}
```

Obr. 2.6: ViewModel ukazkové aplikace

```
@viewModel TestDotvvmApp.ViewModels.DefaultViewModel, TestDotvvmApp
@masterPage Views/MasterPage.dotmaster
<dot:Content ContentPlaceHolderID="MainContent">

    <div>
        <dot:TextBox Text="{value: Number1}" />
        ?
        <dot:TextBox Text="{value: Number2}" />
        =
        {{value: Result}}
    </div>
    <div>
        <dot:Button Text="Složit" Click="{command: Add()}" />
        <dot:Button Text="Odečíst" Click="{command: Substract()}" />
        <dot:Button Text="Vynásobit" Click="{command: Multiply()}" />
        <dot:Button Text="Rozdělit" Click="{command: Divide()}" />
    </div>
</dot:Content>
```

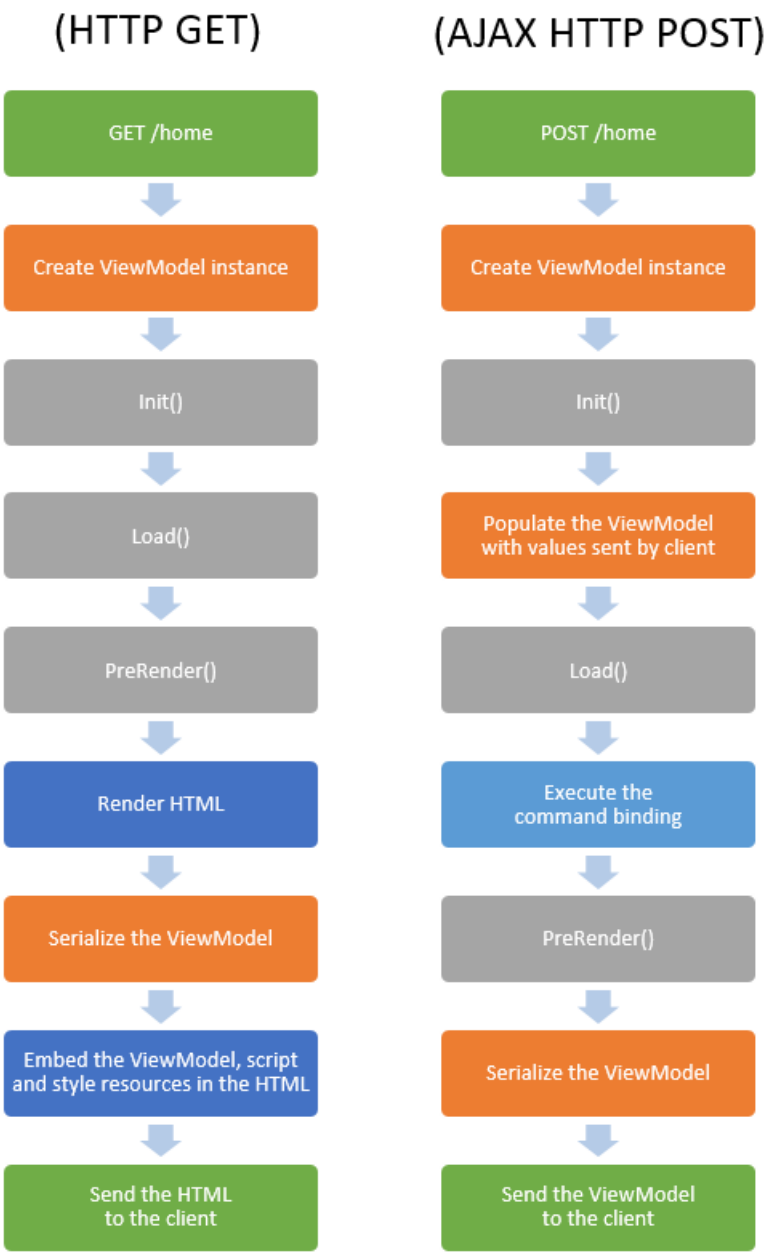
Obr. 2.7: View ukazkové aplikace



Obr. 2.8: Konzole prohlížeče ukázkové aplikace

2.6.2 Komunikace ve DotVVM

Životní cyklus frameworků je podobný jako u Web Forms. Komunikace probíhá pomocí HTTP Get a Post požadavků. Při první návštěvě stránkyse najde příslušný ViewModel, který patří ke View. Vytvoří se instance ViewModelu, a pokud tato instance dědí z DotvvmViewModelBase, tak se zavolá konstruktor a v něm metody Init(), Load() a PreRender(). Do těchto metod můžeme doplňovat logiku nebo je úplně přepsat. Následně se zavolá renderování komponent a vytvoří se HTML, do kterého se přidá reference na dotvvm.js a jiné potřebné skripty. Na konci se objekt ViewModelu vyserializuje do JSON a přidá se do HTML stránky. V momentě, kdy uživatel začne interakci s komponenty, které vyvolávají postback, ViewModel se vyserializuje a odešle AJAX Post požadavek na server. Poté je nalezena příslušná třída ViewModel, ve které se spustí metoda Init(), ta přidá do objektu ViewModelu hodnoty, které přišli od uživatele a zavolá se metoda Load() a PreRender(). ViewModel se znovu serializuje do JSON a uživateli se odešle jenom to, co se ve ViewModelu změnilo. Celý průběh komunikace je znázorněn na obr. 2.9. [19]



Obr. 2.9: Komunikace ve DotVVM

2.6.3 Výhody použití DotVVM

Možnost tvorby vlastních komponent

Každý vývojář webových aplikaci ví, jakou výhodou je možnost v technologii vytvářet vlastní moduly, komponenty či patterny. Tato řešení se pak dají použít v několika dalších projektech. V ASP.NET Web Forms, na rozdíl od ASP.NET MVC, byla dobře realizovaná možnost zkombinovat kaskádové styly, skripty a serverový

kód samotné komponenty. DotVVM nabízí díky své podobě Web Forms také stejnou možnost. [18]

Validace

V jakékoliv velké webové aplikaci je potřeba využívat validaci. DotVVM nabízí validaci jak na serveru, tak i pomocí JavaScriptu na klientu. Je to velká výhoda - uživatel nemusí dlouho čekat na odpověď ze serveru, když mu na klientské straně validátor během půl vteřiny oznámí, že má chybu. Občas muže nastat situace, že potřebujeme provést validaci na serveru kvůli přístupu do databáze, aby například zkontrolovat unikátnost e-mailové adresy. Z výše uvedeného je patrné, jak užitečné je mít technologii, která umí validovat oboustranně. [18]

Lokalizace

Většina webových stránek má v současnosti možnost měnit jazyky. Velmi dobrým příkladem je sociální sít Facebook, která podporuje velké množství jazyků. V ASP.NET aplikacích mají soubory typu RESX přeložené texty a jsou lokalizovány na serveru. Existují i knihovny pro lokalizovaní na klientské straně, ale pak je potřeba využít API, aby se spojilo s RESX daty, což nás nutí psát zbytečný kód. DotVVM poskytuje snadnou práci s lokalizací textu, kde stačí jen vytvořit RESX soubor a referenci na něj z nějaké komponenty. [18]

Datum a čas

Často se ve vývoji setkáváme s problémem zobrazení data a času v určitém formátu. Ideálním řešením by bylo stejné formátovaní jak na serveru, tak na klientu. DotVVM nám takové formátování poskytuje, má je dokonce ve stejném formátu, na jaké jsou zvyklí .NET vývojáři[4]. [18]

[4]Formátovaní Data a času v DotVVM - https://www.dotvvm.com/docs/tutorials/basics-formatting-dates-and-numbers/1.1

3 IMPLEMENTACE PROJEKTU

3.1 Definice problému

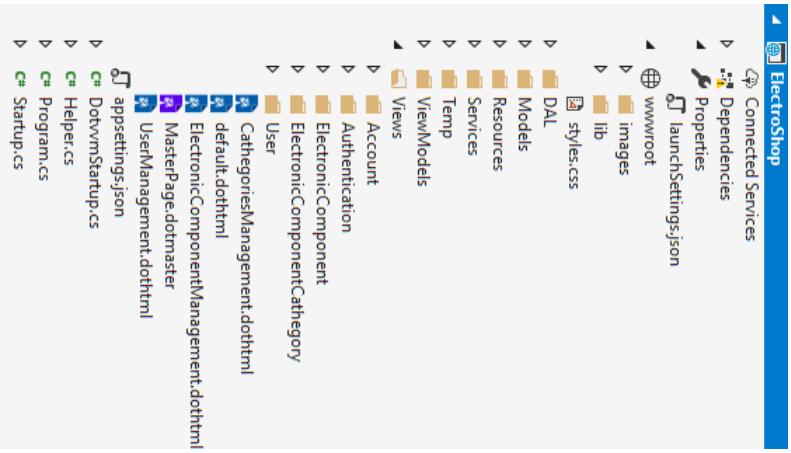
Cílem práce bylo vytvořit aplikaci, sloužící jako informační systém skladu elektronických součástek. Aplikace bude poskytovat přístup pouze třem skupinám uživatelů, s různými přístupovými právy. První skupinou jsou administrátoři, kteří mají možnost přidávat, upravovat a mazat v systému pracovníky všech skupin. Druhou skupinu tvoří skladníci, kteří mohou měnit stav skladu. Mají možnost definovat kategorie a elektrické součástky, upravovat je, přidávat k nim obrázky a datasheet. Mají právo také vygenerovat **.txt** soubor s objednávkami součástek. Poslední skupina jsou konstruktéři, kteří mají k dispozici jen seznam součástek k prohlížení.

Aplikace je přizpůsobená pro činnost pracovníka tak, aby zjednodušila jeho práci. K vytvoření aplikace je potřeba znalost programovacího jazyka C#, technologie Entity Framework, dále pak IDE Visual Studio a MS SQL Server. Firmou Riganti byla poskytnuta obchodní verze frameworku DotVVM, jehož komponenty byly použity v aplikaci. Podrobnější popis technologií a instalační soubory lze nalézt na oficiálních stránkách firmy.

Aplikace poslouží jako příklad implementace aplikace pomocí platformy ASP.NET Core v kombinaci s frameworkem DotVVM. V této kapitole bude popsána struktura a funkce jednotlivých části aplikace.

3.2 Adresářová struktura

V této kapitole je podrobně popsána adresářová struktura aplikace. Ta poslouží lepší orientaci čtenáře v projektu, kde každá složka obsahuje sadu určitých dat. Adresářová struktura bude také praktickým podkladem při rozšiřování programu a implementování nových funkcí.



Obr. 3.1: Adresářová struktura aplikace

- *Dependencies*
  Dependencies (závislosti) obsahují všechny nainstalované NuGet balíčky v aplikaci. Pojďme se projít některé z nich:
  - *Microsoft.AspNetCore*
    Hlavní baliček pro tvorbu ASP.NET Core aplikací. Obsahuje funkce jako routování, konfigurace projektu atd.
  - *Microsoft.AspNetCore.Authentication.Cookies*
    Balíček pro práci s autentizací s využitím **cookies**. [21]
  - *Microsoft.AspNetCore.Identity*
    Tento balíček nám poskytuje systém autorizace a autentizace Identity.
  - *Microsoft.EntityFrameworkCore*
    Balíček nám poskytuje plnou funkčnost Entity Frameworku - při práci s databází.
  - *DotVVM.AspNetCore*
    Hlavní balíček práci s DotVVM frameworkem.
  - *DotVVM.AspNetCore.BusinessPack*
    Balíček placené verze DotVVM. Poskytuje rozšířené funkce a přistup k většímu počtu komponent.
- *Properties*
  Uzel Properties (Vlastnosti) zahrnuje **launchSettings.json** soubor, který obsahuje nastavení konfigurace pro spuštění programu. Můžeme také upravovat konfiguraci přímo v nastavení projektu. Klikneme pravým tlačítkem na projekt a zvolíme Properties, pak klikneme v levém menu na Debug a v položce Profile můžeme přepínat nastavení.
- *wwwroot*
  wwwroot slouží jako kořenová složka pro uchování webových souborů. Obecně

platí, že by měly být odděleny složky pro různé typy statických souborů, jako jsou JavaScript, CSS, Obrázky, knihovny, skripty atd. V našem případě máme hlavní CSS soubor **styles.css**, složku *lib* s knihovnami bootsrap a jquery a složku *images*, která obsahuje obrázky.

- Adresář *DAL*
  Tento adresář obsahuje C# třídy různých entit - User (Uživatel), ElectronicComponents (Elektronické součástky) a ElectronicComponentCategories (Kategorie elektronických součástek). Dále jsou obsaženy třídy DbInitializer a ElectroShopDBContext, které inicializují databázi a vytvářejí EF Core kontext dat pro následnou interakci.
- Adresář *Models*
  V adresáři Models jsou uloženy všechny modely použité v aplikaci: UserModel reprezentující uživatele, ElectronicComponentModel reprezentující elektronickou součástku, OrderItem reprezentující objednávku a další.
- Adresář *Resources*
  Tento adresář obsahuje soubor **Texts.resx**, ve kterém se nachází všechny texty a názvy z webových stránek aplikace. Také může posloužit k vytvoření podpory mnohojazyčnosti.
- Adresář *Services*
  Services (služby) obsahuje třídy, které poskytují metody pro dotazování na databázi přes kontext Entity Frameworku. Třida CathegoryService slouží pro dotazy na kategorie elektronických součástek. Třída ElectronicComponentService obsahuje dotazy na elektronické součástky. Třída UserService slouží pro tvorbu, mazaní a upravování uživatelů a identit.
- Adresář *Temp*
  V adresáři Temp si aplikace uchovává různá dočasná data.
- Adresář *ViewModels*
  Tento adresář obsahuje třídy reprezentující logiku webových stránek vytvořených pomocí DotVVM frameworku. Více o ViewModelu nalezneme v kapitole 2.3.3.
- Adresář *Views*
  V adresáři Views jsou umístěny **.dohtml** webové stránky frameworku DotVVM.
- Soubor *appsettings.json*
  Jedná se o jeden ze zaváděcích souborů aplikace. Je to konfigurační soubor, který definuje cestu k databázi na serveru.
- Soubor *DotvvmSturup.cs*
  Jedná se o konfigurační soubor DotVVM frameworku. DotVVM používá kód C# pro nastavení funkcí a parametrů.

Metoda *ConfigureRoutes* definuje cesty v aplikaci. Každá stránka v DotVVM musí být registrována ve směrovací tabulce. Registrace cesty má následující parametry: 1 - název cesty, 2 - URL cesta ke stránce, 3 - cesta k **.dothtml** souboru v projektu a 4 - tento nepovinný parametr definuje implicitní hodnoty cesty.
Metoda *ConfigureControls* slouží pro registraci vlastní DotVVM komponenty vytvořené vývojářem.
Metoda *ConfigureResources* registruje kaskádové styly a skripty v projektu. V našem případe se jedná o výše zmíněný soubor **styles.css**.
V hlavní konfigurační metodě *Configure* jsou obsaženy tři výše uvedené metody. K dispozici jsou také také metody AddBusinessPackConfiguration pro fungování pokročilých DotVVM komponent a registrace knihovny bootstrap - *AddBootstrapConfiguration*.

- Soubor *Helper.cs*
  Pomocná třída obsahující metody parsování dat.
- Soubor *Program.cs*
  Hlavní třída ASP.NET Core aplikace, která je určena pro její spouštění.
- Soubor *Startup.cs*
  Hlavní konfigurační třida ASP.NET Core aplikace. Registruje veškeré služby, role, autorizaci, přístupy k databázi atd.

3.3 Přihlášení a bezpečnost

3.3.1 ASP.NET Identity

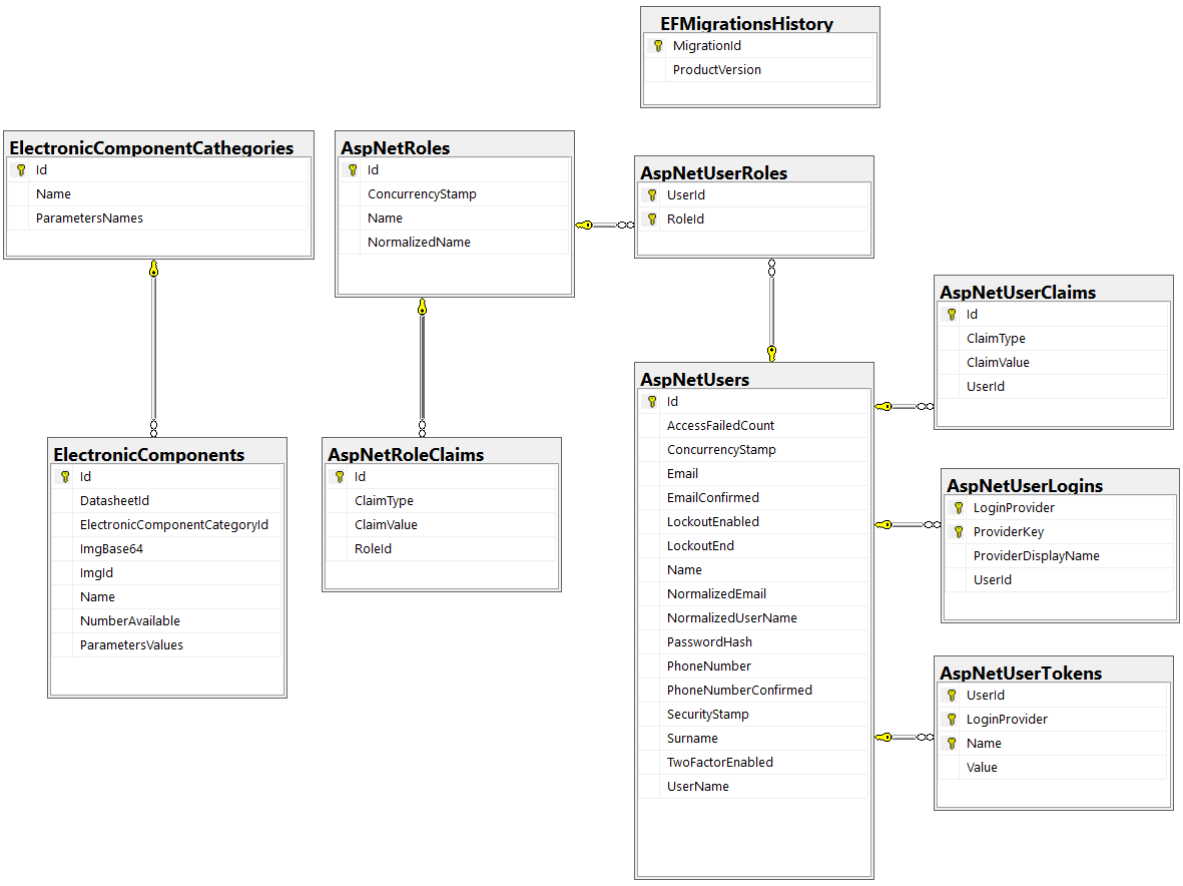
Bezpečnost serveru, který obsahuje důležité záznamy a je volně přístupný z internetu, je klíčová. Právě proto bylo rozhodnuto použit hotové řešení od firmy Microsoft. Technologie ASP.NET Identity slouží k autorizaci a autentizaci uživatelů. Nabízí nám hned několik způsobů přihlášení. Základní možností je vytvoření účtu v databázi, případně se můžeme přihlásit přes účty externí jako je Microsoft, Facebook nebo třeba Google. V našem případě jsme využili možnosti individuálního účtu přímo v databázi, kde na základě uživatelského jména nebo role nám budou přidělena práva pro používání systému. Klíčovými objekty v ASP.NET Identity jsou uživatelé a role. Všechny funkce pro vytvoření či smazání uživatele nebo pro interakci s úložištěm uživatelů jsou uloženy do třídy UserManager. Pro práci s rolemi a jejich řízením je definována třída RoleManager. Třídy UserManager a RoleManager jsou umístěny v knihovně *Microsoft.AspNet.Identity.Core*, a pro komunikaci s databází ve jmenném prostoru Microsoft.AspNet.Identity.EntityFramework je definována třída kontextu *IdentityDbContext*. Podívejme se podrobněji na některé funkce, které nám

jsou k dispozici v těchto třídách:

- *UserManager*
  - *Create(user) / CreateAsync(user)*: vytváří nového uživatele.
  - *Delete(user) / DeleteAsync(user)*: odstraňuje uživatele.
  - *Update(user) / UpdateAsync(user)*: aktualizuje uživatele.
  - *FindById(id) /FindByIdAsync(id)*: hledá uživatele podle id.
  - *ChangePassword(id, old, new) / ChangePasswordAsync(id, old, new)*: mění heslo uživatele. [22]
- *RoleManager*
  - Create(role) / CreateAsync(role): vytváří novou roli.
  - Delete(role) / DeleteAsync(role): odstraňuje roli.
  - Update(role) / UpdateAsync(role): aktualizuje roli.
  - RoleExists(name) / RoleExistsAsync(name): vrací **true**, jestliže role s tímto názvem již existuje. [23]

ASP.NET Identity používá přístup Code-First platformy Entity Framework pro automatické vytváření svých schémat v databázi. Pro generování databáze a rolí byly vytvořeny třídy ApplicationUser, ElectronicComponent a ElectronicComponentCathegory, které reprezentují entity. Třída UserType typu enum obsahuje seznam rolí uživatelů a je součástí vlastností třídy ApplicationUser. Ta dědí od IdentityUser všechny vlastnosti. Logika vytvoření rolí a uživatelů umístěná v metodách *InitRole* a *InitUser*. Třída ElectroShopDBContext reprezentuje databázový kontext a dědí z kontextu Identity.

Nyní zkusíme projekt spustit. Stiskneme klávesu F5 a vyčkáme, než Entity Framework vygeneruje databázi. Až se zobrazí webová stránka, v levé části okna Visual Studio nalezneme vytvořenou databázi. Dostaneme se k ní otevřením SQL Server Object Explorer -> (localdb)\mssqllocaldb -> otevřeme databázi ElectroShop a klikneme na Tables. V této složce jsou uloženy všechny nové tabulky. Vzhledem k tomu, že Enity framework Core nepodporuje zobrazení .edmx diagramů ve Visual Studiu, můžeme využít MS SQL Server. Otevřeme program a v dialogovém okně connect to Server napíšeme (localdb)\mssqllocaldb do položky Server Name. Ostatní položky necháme beze změny. Klikneme na Databases a vybereme položku ElectroShop. Poté klikneme pravým tlačítkem myši na Database Diagrams a vybereme New Database Diagram. V okně Add Table označmíe všechny tabulky a klikneme na tlačítko ADD. Po provedení operací MSSQL Server vygeneruje diagram se závislostmi stejným způsobem, jako je na obr. 3.2.



Obr. 3.2: Diagram vytvořených tabulek

Díky dědičnosti z IdentityDbContext jsou v existující databázi vytvořeny tabulky pro ukládání informací o uživatelích. Ve výchozím nastavení se po přihlášení prvního uživatele Identity vytvoří následující sada tabulek:

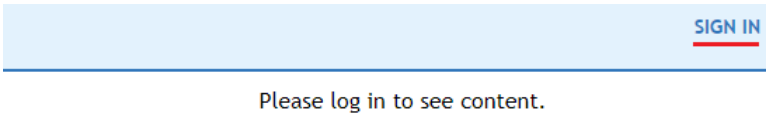
- *EFMigrationHistory*
  Tabulka používaná Code First Migrations pro ukládání informací o migracích, používaných k databázi.
- *AspNetUserClaims*
  Tabulka obsahující sadu *claim*. Tato sada představuje ve srovnání s rolemi jiný Claims-Based Authorization model autorizace. Model autorizace Claim obsahuje základní informace o uživateli, např. email, přihlašovací jméno, věk atd. Tyto informace nám umožňují identifikovat uživatele a přidat mu příslušná oprávnění.
- *AspNetUserLogins*
  Tabulka obsahující přihlašovací údaje (login) uživatele.
- *AspNetUserRoles*
  Tabulka, která uživatelům přiřazuje různé role.

- *AspNetUsers*
  Tabulka obsahující údaje registrovaných uživatelů.

Zbývající dvě tabulky ElectronicComponentCathegories a ElectronicComponents, jak popsáno výše, obsahují informace a součástkách a jejich kategoriích.

3.3.2 Přihlášení

Při prvním spuštění aplikace se v horní části okna webového prohlížeče objeví menu z obr.3.4.



Obr. 3.3: Základní menu webové aplikace

Toto menu je společnou součástí všech stran a jeho kód se nachází v souboru MasterPage.dotmaster. V tomto souboru jsou **.dot** komponenty RoleView a AuthenticatedView. RoleView generuje individuální obsah na stránce uživateli s přiřazenou rolí. V našem případě komponenta pouze přidává název stránky, na nichž se nachází pracovník. AuthenticatedView má podobnou funkci, ale na rozdíl od RoleView, renderuje různý obsah přihlášeným a nepřihlášeným uživatelům.

Po stisknutí tlačítka SIGN IN nás aplikace přesměruje na stránku SignIn.dothtml. Na této stránce je k dispozici formulář pro zadání jména a hesla. Pro počáteční používání aplikace, byly vytvořeny tři osoby každé skupiny: *admin*, *skladnik* a *konstrukter*, které mají stejné heslo *Master1234**. Pro vytvoření uživatele a hesla byla implementovana metoda Initialize() třídy DbInitializer(). Po zadání jména a hesla a stisknutí modrého tlačítka SIGN IN, se zavolá asynchronní operace SignIn() z ViewModelu, která porovnává údaje se záznamy v databázi. V případě shody je uživatel přihlášen a přesměrován na příslušnou stánku. Pokud nedojde ke neshodě, program ohlásí chybu.

Obr. 3.4: Přihlašovací formulář

3.4 Administrátor

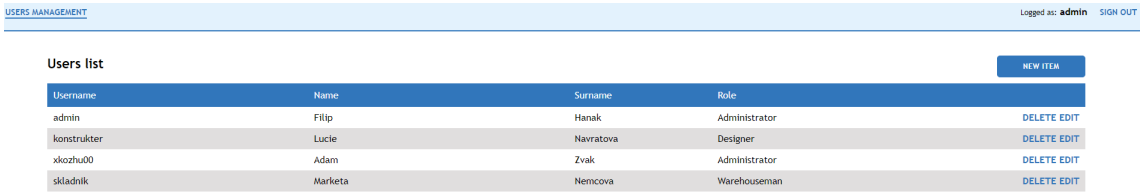
Po přihlášení jako správce nalezneme na hlavní stránce tabulku se seznamem všech uživatelů, obsahující jména, příjmení a role zaměstnanců. Administrátorská práva správce zahrnují vytváření, mazání a změnu popisu uživatelů. Chceme-li vytvořit novou osobu, klikneme na modré tlačítko s nápisem NEW ITEM. Po kliknutí budeme přesměrováni na stránku /user/create.dothtml, kde se nachází hlavní formulář k přidání nové osoby. Pro úspěšné odesílání je třeba vyplnit všechny položky. Typ hesla je nastaven v ASP.NET Identity. Ta vyžaduje, aby heslo obsahovalo alespoň jeden velký symbol, jeden alfanumerický znak, a jeden nealfanumerický znak. Při porušení tohoto pravidla program vypíše varování o chybě. Můžeme také měnit formát hesla. Pro změnu je třeba ve třídě Startup přidat službě AddIdentity (service) další nastavení. Podívejme se na některé z nich:

- *options.Password.RequireDigit*
  Vyžaduje číslo od 0 do 9 v heslu. Výchozí hodnota je **true**.
- *options.Password.RequiredLength*
  Definuje minimální délku hesla. Výchozí hodnota je 6.
- *options.Password.RequireLowercase*
  Vyžaduje aspoň jedno malé písmeno v hesle. Výchozí hodnota je **true**.
- *options.Password.RequireUppercase*
  Vyžaduje aspoň jedno malé písmeno v hesle. Výchozí hodnota je **true**.
- *options.Password.RequireNonAlphanumeric*
  Vyžaduje nealfanumerický znak v heslu. Výchozí hodnota je **true**. [24]

Další funkce, která je k dispozici administrátorovi, je úprava uživatelů. Otevřeme seznam všech pracovníků a vpravo od vybraného klikneme na tlačítko EDIT. Na nové stránce se objeví formulář, kde můžeme měnit jméno, příjmení či roli. Stejně tak je

nám k dispozici tlačítko DELETE, které odstraní pracovníka z databáze. Username osoby měnit není možné.

3.4.1 View



Obr. 3.5: Hlavní stránka administrátora

Pro vytvoření hlavní tabulky uživatelů na View byla použita komponenta GridView.
GridView je komponentou pro řízení tabulkou, která umožňuje přivázat zdroj dat a tyto údaje pak řadit. Atribut DataSource určuje zdroj dat, který nám zprostředkuje ViewModel. Atribut class přidává tabulce kaskádové styly. Každý sloupec tabulky je definován v komponentě GridViewTextColumn, jejíž atribut ValueBinding slouží pro zobrazení záznamů v tabulce, a HeaderText je textem, který vidíme v záhlaví: username, name atd. Dále můžeme vidět komponentu RouteLink. To je jen odkaz na stránky EDIT a DELETE, pomocí které předáváme na ViewModel hodnotu Username vybraného uživatele.
Všechny texty byly přesunuty do souboru **Texts.resx** a přistupujeme k nim pomocí direktiv:

- *@viewModel ElectroShop.ViewModels.UsersManagementViewModel*
  Direktiva, která odkazuje na ViewModel stránky.
- *@masterPage Views/MasterPage.dotmaster*
  Direktiva pro dědění šablony menu z hlavního souboru MasterPage.dotmaster.
- *@import ElectroShop.Resources*
  Direktiva, odkazující na soubor textu ve složce Resources.

3.4.2 ViewModel

Výpis 3.1: Kód ViewModelu UsersManagement

```
namespace ElectroShop.ViewModels                                    1
{                                                                   2
    [Authorize(Roles = "Administrator")]                            3
public class UsersManagementViewModel : MasterPageViewModel        4
    {                                                               5
        private readonly UserService userService;                   6
                                                                    7
public UsersManagementViewModel(UserService studentservice_)       8
        {                                                           9
            this.userService = studentservice_;                     10
        }                                                           11
        [Bind(Direction.ServerToClient)]                            12
        public List<UserListModel> Users { get; set; }              13
                                                                    14
        public override async Task PreRender()                      15
        {                                                           16
            Users = userService.GetUsers();                         17
            base.PreRender();                                       18
        }}}                                                         19
```

Ve výpisu 3.1 je zobrazen kód třídy UsersManagementViewModel. V DotVVM máme k dispozici atribut [Authorize] ze jmenného prostoru DotVVM.Framework.Runtime.Filters. Jeho úkolem je kontrolovat uživatele přicházejících na stránku.

Řádek [**Authorize(Roles = "Administrator")**] nám říká, že oprávnění ke vstupu na stránku mají pouze uživatelé s rolí *Administrator*. Tento atribut lze použit jak pro celou třídu, tak i pro dílčí metody.

Někdy při vývoji aplikace je třeba určit směr, kam budeme přenášet data - jestli ze serveru na klienta anebo naopak. K tomu v DotVVM existuje atribut [Bind], při jeho implementaci kterého můžeme zadat jeden z následujících způsobů přenosu dat:

- *ServerToClient*
  Přenáší hodnoty ze serveru ke klientovi. Jedná se o ideální způsob v případě, kdy máme na stránce komponenty ComboBox nebo GirdView pouze pro zobrazení dat ze serveru.
- *ClientToServer*
  Předává hodnoty pouze od klienta k serveru. Tento způsob lze použít pro hodnoty pole formuláře, které mohou být přečteny pouze na serveru.

- *Both*
  Je výchozím nastavením. Data se přenášejí v obou směrech. Použití tohoto nastavení se nejlépe hodí například pro ovládací prvky formuláře TextBox.
- *None*
  Hodnoty vlastností z ViewModelu nejsou přenášeny ani ze serveru ke klientovi ani naopak.

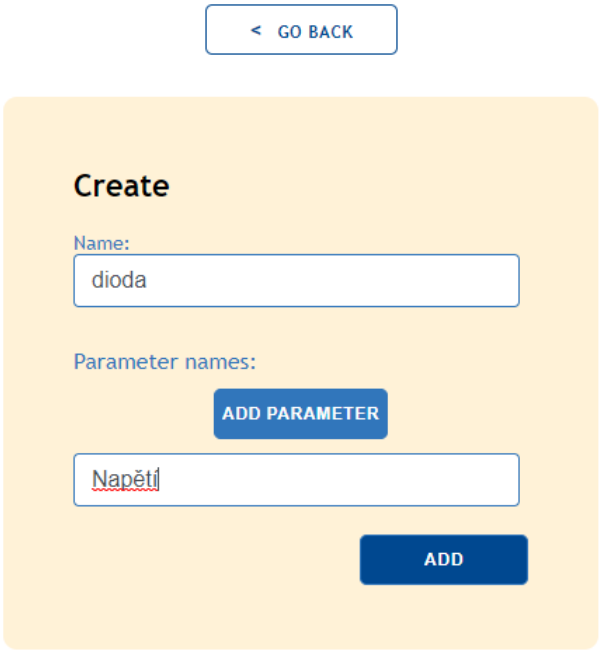
Tento atribut se vždy inicializuje nad vlastnosti v kódu ve ViewModelu. Pro zobrazení uživatelů v GridView na stránce administrátora je typ přenosu ServerToClient nejvhodnějším.

3.5 Skladník

Po úspěšném přihlášení na webovou stránku je uživateli skladník k dispozici celá řada možností: vytvářet kategorie, parametry, součástky a a vytvářet jejich úpravy. Skladník má také možnost přidávat Datasheet ve formátu **.pdf** a obrázek. Podívejme se na všechny tyto možnosti podrobněji. V levé horní části menu nalezneme odkaz Cathegories, kliknutím na něj jsme přesměrováni na stránku CathegoriesManagement. Na této stránce se také nachází tabulka generovaná komponentou GridView, ale pouze s názvy kategorií. Budeme-li chtít definovat novou kategorii, klikneme na modré tlačítko s nápisem NEW ITEM v pravé části obrazovky. Díky tomu se nám zobrazí formulář pro vytvoření nové kategorie. Můžeme zadat název kategorie a kliknutím na tlačítko ADD Parameter přidat parametry. Celé definovaní parametrů závisí na typu kategorie. Doporučuje se přidávat co největší množství parametrů, které co nejpřesněji popisují vlastnosti součástek. Předpokládáme architekturu aplikace, v níž při vytváření nové kategorie není možné měnit její nastavení nebo ji mazat. Všechna pole formuláře jsou povinná a při odeslání prázdného textu vrátí aplikace varování o chybě. Název kategorie nesmí být schodný s jíž existujícími názvy.

Nyní vytvoříme kategorii s názvem *dioda* a přidáme jí dva parametry: *Napětí* stejně, jak to znázorněno na obr. 3.6. Po úspěšné operaci najdeme v levém horním rohu odkaz na stránku Components. Kliknutím na ni se vrátíme zpět na domovskou stránku uživatele skladník. V pravém horním rohu najdeme tlačítko NEW ITEM, po kliknutí se nám zobrazí ještě jeden formulář pro vytvoření součástky. Napíšeme její jméno a v komponentě ComboBox najdeme nově vytvořenou kategorii (dioda). Po kliknutí na kategorii DotVVM automaticky přidá pole parametrů k vyplnění. Do tohoto pole můžeme zadat množství součástek k použití a nahrát obrázek či datasheet. Vyplníme všechna pole a klikneme na modré tlačítko ADD. V GridView na hlavní stránce se objeví námi vytvořená součástka.

V posledním sloupci se nacházejí tři tlačítka. Pokud jsme dříve přidali Datasheet k součástce, uvidíme nyní i další tlačítko Datasheet. kliknutím na které si můžeme stáhnout **.pdf** soubor obsahující tabulky s hodnotami. První tlačítko Detail zobrazí formulář s popisem součástky. Tlačítko EDIT umožňuje změnit název, kategorii nebo obrázek či datasheet komponenty. Tlačítko ADD TO ORDER slouží k vytvoření objednávky. Nad tabulkou jsou k vidění DotVVM komponenty Textbox a ComboBox. První pole filtruje data v tabulce podle zadaného názvu součástky, druhé filtruje vše v závislosti na vybrané kategorii. Díky těmto komponentám rychle najdeme potřebnou informaci v tabulce.



Obr. 3.6: Formulář pro vytvoření kategorie součástky

Poslední možnost, která patří mezi práva skladníka je vytváření objednávky. Klikneme na tlačítko Add to order v posledním sloupci tabulky. Nahoře, vedle tlačítka New Item se objeví tlačítko Show order. Po jeho stisknutí se otevře modální okno s tabulkou objednávek. Kliknutím na Remove one můžeme dekrementovat počet součástek. Abychom vygenerovali textový soubor, klikneme na tlačítko Generate Order, díky němuž začne stahování souboru **.txt**. Pro dokončení procesu klikneme na finální tlačítko Accept & Close, následně se počet vybraných součástek odečte z databáze.

3.5.1 View

Modální okno

Pro použití modálního okna byla vybrána komponenta ModalDialog. DotVVM nabízí velmi jednoduchý způsob, jak pracovat s modálními okny, s minimálním množstvím kódu. Mezi její atributy patří:

- *ID*
  Získá nebo nastaví unikátní ID na komponentu.
- *IsDisplayed*
  Získá nebo nastaví hodnotu označující viditelnost dialogového okna.
- *HeaderText*
  Získá nebo nastaví text záhlaví.
- *CloseOnOutsideClick*
  Získá nebo nastaví, zda okno bude zavřeno v případě, že uživatel klikne mimo dialogové okno. Ve výchozím nastavení tento atribut povolen.
- *CloseOnEscape*
  Získá nebo nastaví, zda bude okno zavřené, když uživatel zmáčkne klávesu Escape. Ve výchozím nastavení je toto povoleno.
- *DataContext*
  Získá nebo nastaví kontext dat pro komponentu a její potomky.

Při otevření stránky DotVVM analyzuje komponentu ModalView spolu s jejími atributy a vygeneruje následující html kód:

Výpis 3.2: Kód ViewModelu UsersManagement

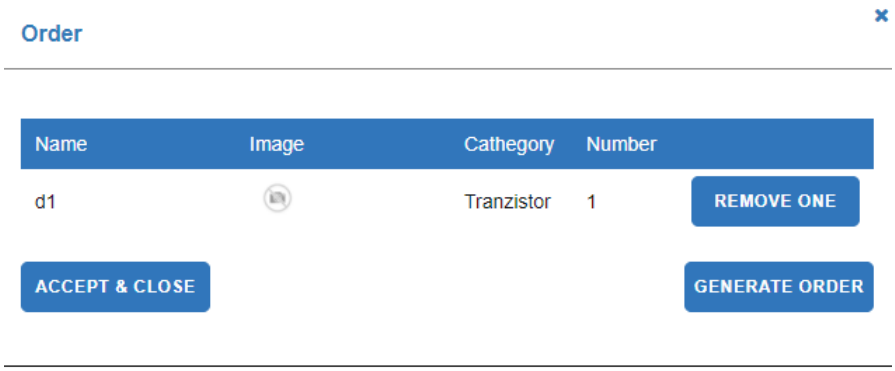
```
<div class="dotvvm-bp-modal-dialog">                    1
    <div class="bp-container">                          2
    <div class="bp-grip"></div>                         3
    <div class="bp-header"></div>                       4
    <div class="bp-contents"><div>                      5
</div>                                                  6
```

Prvek div s CSS třídou *dotvvm-bp-modal-dialog* je rodičovský prvek modálního okna. Má takové atributy jako *role*, nebo *style*, který určuje styly celého okna a atribut *data-bind* obsahující skripty, např. *IsModalDisplay*, který určuje logiku skrytí a zobrazení okna.
Popis další prvků - potomků:

- div s třídou *bp-container* - kontejner, který má podobné styly jako jeho rodič, stylizuje modální okno a pozadí.
- div s třídou *bp-grip* - určuje styl rámců.

- div s třídou *bp-header* - zobrazuje záhlaví, ve kterém se nachází název okna a tlačítko zavřít.
- div s třídou *bp-contents* - hlavní div, ve kterém je celý obsah. V našem případě je to html prvek *table*, který byl vyrenderováný komponentou GridView.

Výsledek výše ukázaného kódu můžemee vidět na obr. 3.7.



Obr. 3.7: Ukázka modálního okna

FileUpload

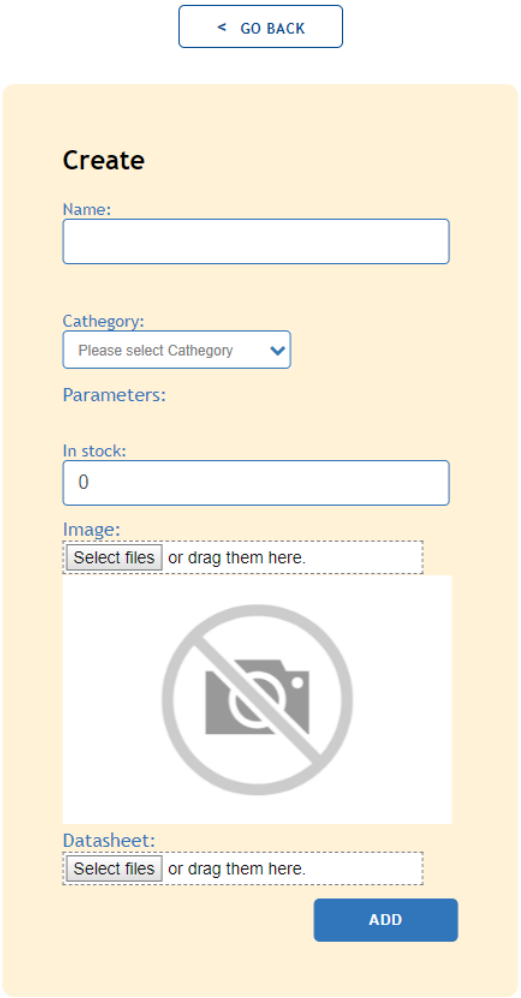
Při implementaci logiky nahrávaní souborů do databáze se nepodařilo najít vhodné řešení, při kterém by proces odesílaní a zpětného získaní obrázku a datasheetu fungoval spolu s DotVVM bez vyplývajících problémů na straně frameworku. Všechny způsoby implementace logiky jsou umístěny a zakomentovány ve třídě CreateViewModel.cs v metodě ProcessUploadedImage.
Proto byla vybrána komponenta FileUpload, která umožňuje uživateli posílat jeden nebo více dočasných souborů asynchronně. Podívejte se na její atributy:

- *Data*
  Vrátí nebo nastaví data k nahraní.
- *AllowedFileTypes*
  Získá nebo nastaví typy souborů, které server akceptuje.
- *AllowMultipleFiles*
  Získá nebo nastaví, zda uživatel může vybrat více souborů najednou. Ve výchozím nastavení tento atribut povolen.
- *MaxFileSize*
  Získá nebo nastaví maximální velikost souborů v megabajtech. Ve výchozím nastavení je velikost neomezená.

- *UploadCompleted*

  Získá nebo nastaví událost, která se spustí, když se nahraje soubor.

DotVVM doporučuje ukládat nahrané soubory v adresáři aplikace. Z toho důvodu byla vytvořena složka Temp, která ukládá dočasné soubory. Cesta k této složce je definovaná v UploadedFileStorage ve třídě Startup.cs.



Obr. 3.8: Ukázka formuláře s DotVVM komponentou FileUpload

3.5.2 ViewModel

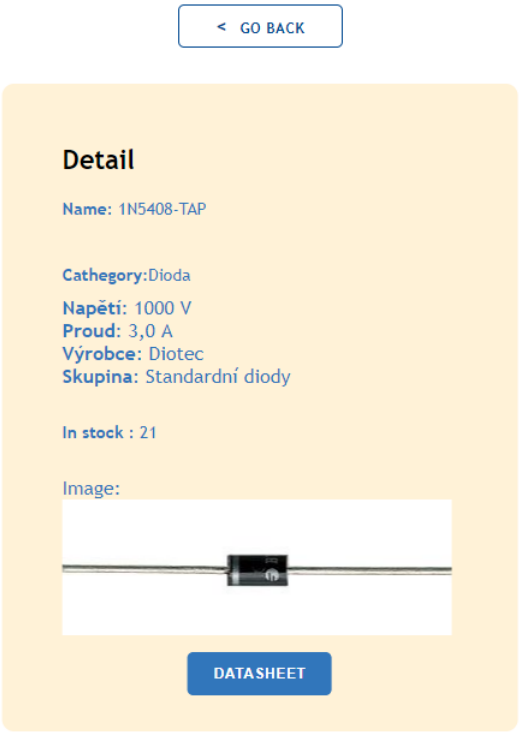
Po otevření ViewModel stránky máme k dispozici celou logiku práce Gridview, ModalDalog atd. V metodě INIT() se provádí počáteční inicializace dat při vstupu na stránku. Například na hlavní GridView se váže nový zdroj dat s možností stránkovaní. V případě, že je počet součástek větší než 10, pod tabulkou se zobrazí tlačítka navigace. Další asynchronní metoda AddToOrder přidává vybrané součástky

do slovníku OrderCount reprezentujícího objednávky. Metoda RemoveOneItemFromOrder odstraňuje po jedné vybrané objednávce z GridView v modálním okně. Jako parametr přijímá id komponenty. Další Metoda SetAllOrderCountsToZero má v sobě cyklus foreach, který projde všechny položky ve slovníku OrderCount a každému prvku přidá nulovou hodnotu. Tato stejná metoda se používá v případě AcceptAndCloseOrder, při dokončení objednávky. Také se pomocí foreach projde slovník objednávek a z dostupných se odečtou objednané součástky. Pak na konci metody se do ElectronicComponents znovu přidá zdroj dat a pomocí *IsModalDisplayed = false* se vypne modální okno.
Metoda GenerateStreamFromString jmenného prostoru Stream nám poskytuje logiku přenosu dat. Její funkčnost využíváme v metodě GenerateOrder pro generovaní textového souboru objednávek. Poslední metoda DownloadDatasheet nám stáhne Datasheet z úložiště. Přijímá dva parametry - id souboru a jeho název.

3.6 Konstruktér

Po úspěšném přihlášení bude uživatel s rolí Konstruktér bude přesměrován ne stejnou stránku jako Skladník. Mají však k dispozice pouze limitované možnosti. Jedná se o oprávnění k prohlížení databáze a možnost zobrazit detaily vybrané součástky. Na hlavní stránce bude dostupná tabulka GridView, v níž můžeme díky komponentě ComboBox záznamy filtrovat podle kategorií. Komponenta TextBox slouží k vyhledávání součástek podle názvu. Napravo od položky se zobrazí tlačítko detail a pokud skladník přidal k součástce datasheet, objeví se také objeví tlačítko pro stažení. Kliknutí na tlačítko detail nás přesměruje na stránku Detail.dothtml, ve které se nachází podrobný popis součástky. Podobně jako na obr. 3.9.

Pro skrytí tlačítek NewItem a Add to Order byla použita komponenta AuthenticatedView, uvnitř které je umístěna další komponenta RoleView. Obě tyto komponenty jsou popsány v kapitolách výše. Zde v atributu Role můžeme přidat roli, pro kterou bude k dispozici obsah složky RoleView. V případě načtení stránky konstruktorem, nebudou tato tlačítka k dispozici.



Obr. 3.9: Ukázka detailu jednotlivé součástky

3.7 Services

V této kapitole jsou pro stručný přehled popsány metody z adresáře Services.

UserService

Metody v této třídě slouží pro práci s uživatelem, jeho tvorbou, odstraněním a prováděním různých změn.

- *GetUserByUsername()*
  Tato metoda vrací objekt user pomocí třídy userManager. Jako vstupní parametr přijímá userName.
- *GetRoleByUsername()*
  Metoda vrací roli uživatele. Jako vstupní parametr přijímá userName.
- *CreateIdentity()*
  Vytváří objekt třídy Claim (identitu). Jako parametry přijímá objekt třídy ApplicationUser a jeho role. V konstruktoru se vytvoří doplňující informace o objektu (name, id, role), a jako typ autentizace bude zvoleno "Cookie".
- *SignIn()*
  Tato metoda slouží pro přihlášení uživatele. Jako parametry přijímá objekt

uživatele, jeho role a heslo. Pokud objekt user existuje, zavolá se metoda CheckPasswordAsync třídy userManager a provede kontrolu validity hesla. Pokud heslo je validní, zavolá se metoda CreateIdentity(), v opačném případě se vratí **null**.

- *CreateUser()*
  Metoda na vytváření nového uživatele. Jako parametry přijímá username, jméno, příjmení, heslo, a typ role.
- *GetUsers()*
  Metoda vrátí seznam všech uživatelů typu UserListModel z databáze přes kontext Entity Frameworku.
- *GetUserModelByName()*
  Vráti objekt třídy UserModel s parametry. Jako vstup přijímá username uživatele.
- *EditUser()*
  Metoda editování uživatele. Jako vstup přijímá objekt typu UserModel. Na výstupu je objekt typu IdentityResult.
- *DeleteUser()*
  Smaže uživatele z databáze přes třídu UserManager.

CathegoryService

Třída CathegoryService obsahuje metody pro dotazování nad kategoriemi elektronických součástek.

- *GetAllCategories()*
  Tato metoda vrátí seznam všech kategorií součástek typu ElectronicComponentCathegoryModel.
- *GetAllCategoriesNames()*
  Metoda vrátí seznam názvů všech kategorií.
- *GetCathegoryByName()*
  Vrátí kategorii podle názvu na vstupu.
- *Create()*
  Metoda vytvoří kategorii součástky s parametrem (parametr - je pasovaný text typu **string**)
- *ContainsAlreadyAName()*
  Tato metoda posílá dotaz na databázi, zda název kategorii již existuje. Vrátí logickou hodnotu **true** nebo **false**;

ElectronicComponentService

Pro práci s elektronickými součástkami byla vytvořena třída ElectronicComponentervice.

- *GetAllQueriable()*
  Používá se v jiné metodě GetData() ve třídě ElectronicComponentManagementViewModel. Metoda vrací objektu typu ElectronicComponentModel.
- *GetById()*
  Vrací objekt elektronické součástky podle jejího ID. Jako vstup přijímá id typu **int**.
- *Create()*
  Vytváří nový objekt třídy ElectronicComponent. Jako vstup přijímá objekty třídy ElectronicComponentModel.
- *Update()*
  Metoda aktualizuje údaje elektronické součástky. Jako vstup přijímá objekty třídy ElectronicComponentModel.
- *Delete()*
  Vymaže objekt elektronické součástky z databáze. Jako vstup přijímá objektu třídy ElectronicComponentModel.
- *ContainsAlreadyAName()*
  Metoda kontroluje, zda ve třídě ElectronicComponent již existuje elektronická součástka se stejným názvem, následně vrátí logickou hodnotu typu **bool**.

4 ZÁVĚR

Cílem této bakalářské práce bylo navrhnout a implementovat aplikaci, sloužící jako informační systém pro sklad elektronických součástek. V první kapitole jsem popsal teoretické základy platformy .NET a její technologie pro tvorbu webových aplikací. Dále jsem dal definice takovým pojmům jako Entity Framework a LINQ a ukázal seznam jejich základních funkcí. Ve druhé kapitole jsem zkoumal technologie pro vytváření grafických rozhraní. Popsal jsem objektový model DOM, architekturu návrhových vzorů a výhody používaní frameworku DotVVM. Na základě prvních dvou kapitol byla vytvořena ukázková webová aplikace. Roz- bor této aplikace a popis všech použitých funkcí se nachází v poslední kapitole. Aplikace plně odpovídá zadání s výjimkou odesílání dat na server. Tento problém je považován za chybu na straně frameworku DotVVM. Při hledání vhodného ře- šení byla vybrána komponenta FileUpload pro nahrávaní dočasných dat do paměti aplikace. Při vývoji bylo použito vývojové prostředí Visual Studio, díky jehož tech- nologii automatického doplňování kódu IntelliSense a jejího propojení s DotVVM byl vývoj aplikace velmi pohodlný. DotVVM je považován za mladou technologii a stále se nachází ve stavu aktivního vývoje. Proto má ze své podstaty více nedokonalostí a chyb, než jeho konkurenti, jako Angular nebo React. Nehledě na tyto skutečnosti, nápad vytvořit framework podobný ASP.NET Web Forms (zároveň fungující na Javascriptu), je perspektivní a má do budoucna velký potenciál. V této práci jsem se naučil jak funguje DotVVM spolu s ASP.NET Core v praxi a také prohloubil své znalosti v Entity Frameworku, C a principech objektově orientovaného programování. Další verzi aplikace bych rozšířil o stránku sběru statistik dat (obejdnávek, uživatelů, atd.), použil bych protokol SMTP pro odesílání e-mailů zaměstnancům, udělal bych mobilní verzi stránek a přidal knihovnu SignalR pro komunikaci v reálném čase.

LITERATURA

[1] .NET – Glosář | Microsoft Docs. [online]. 2017, poslední aktualizace 08. 07. 2017 [cit. 05. 03. 2018] Dostupné z URL: <https://docs.microsoft.com/cs-cz/dotnet/standard/glossary

[2] FREEMAN, Adam. Pro ASP.NET MVC 5. Apress, 2013, 832 s., ISBN 978-1-4302-6530-6.

[3] Představení technologie ASP.NET Web Pages – Začínáme | Microsoft Docs. [online]. 2015, poslední aktualizace 28. 05. 2015 Dostupné z URL: <https://docs.microsoft.com/cs-cz/aspnet/web-pages/overview/getting-started/introducing-aspnet-web-pages-2/getting-started

[4] EVJEN, B., HANSELMA, S., RADER, D. Professional ASP.NET 4 in C# and VB. Wrox Press Ltd, 2010, 1536 s., ISBN: 978-0-470-50220-4.

[5] Choose between ASP.NET and ASP.NET Core | Microsoft Docs. [online]. 2018, poslední aktualizace 03. 14. 2018 [cit. 05. 03. 2018] Dostupné z URL: <https://docs.microsoft.com/en-us/aspnet/core/choose-aspnet-framework?view=aspnetcore-2.1

[6] Entity Framework Version History | MSDN [online]. 2016, poslední aktualizace 23. 10. 2016 [cit. 05. 03. 2018] Dostupné z URL: <https://msdn.microsoft.com/en-us/library/jj574253(v=vs.113).aspx

[7] Entity Data Model | Microsoft Docs. [online]. 2017, poslední aktualizace 30. 03. 2017 [cit. 05. 03. 2018] Dostupné z URL: <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/entity-data-model

[8] Language-Integrated Query (LINQ) (C#) | Microsoft Docs. [online]. 2017, poslední aktualizace 02. 02. 2017 [cit. 06. 03. 2018] Dostupné z URL: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/

[9] LINQ: .NET Language Integrated Query. Learn to Develop with Microsoft Developer Network | MSDN [online]. 2007, poslední aktualizace 01. 02. 2007 [cit. 06. 03. 2018] Dostupné z URL: <https://msdn.microsoft.com/en-us/library/bb308959.aspx

[10] W3C Document Object Model. World Wide Web Consortium (W3C) [online]. 2005, poslední aktualizace 19. 01. 2005 [cit. 11. 03. 2018] Dostupné z URL: <https://www.w3.org/DOM/

[11] A Brief History of JavaScript – Brendan Eich. [online]. 2010, poslední aktualizace 21. 07. 2010 [cit. 13. 03. 2018] Dostupné z URL: <https://brendaneich.com/2010/07/a-brief-history-of-javascript/

[12] Standard ECMA-262 [online]. 2017, poslední aktualizace 01. 06. 2017 [cit. 13. 03. 2018] Dostupné z URL: <https://www.ecma-international.org/publications/standards/Ecma-262.html

[13] hrome V8 | Google Developers. Google Developers [online]. [cit. 15. 03. 2018]. Dostupné z URL: <https://developers.google.com/v8/

[14] Electron | Build cross platform desktop apps with JavaScript, HTML, and CSS [online]. [cit. 15. 03. 2018]. Dostupné z URL: <https://electronjs.org/

[15] STEFANOV, Stoyan. JavaScript Patterns, 2012, 185 s., ISBN 978-1-449-33181-8.

[16] JSON. [online]. [cit. 16. 03. 2018]. Dostupné z URL: <https://www.json.org/

[17] ASP.NET - Single-Page Applications | MSDN [online]. 2013, poslední aktualizace 01. 11. 2013 [cit. 17. 03. 2018] Dostupné z URL: <https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

[18] HERCEG, Tomáš. DotVVM: “Javascript” Apps Without Javascript. [online]. 2013, poslední aktualizace 01. 11. 2013 [cit. 17. 03. 2018] Dostupné z URL: <https://channel9.msdn.com/Series/NET-DeveloperDays-2015-on-demand/dotVVM-Javascript-Apps-With-No-Javascript-Tomas-Herceg

[19] ViewModels | DotVVM Documentation. [online]. [cit. 17. 03. 2018]. Dostupné z URL: <https://www.dotvvm.com/docs/tutorials/basics-viewmodels/latest

[20] HERCEG, Tomáš. DotVVM: "Javascriptové"aplikace bez Javascriptu. [online]. [cit. 18. 03. 2018]. Dostupné z URL: <http://wug.cz/zaznamy/282-MS-Fest-2015-Brno-dotVVM-javascriptoveaplikace-be

[21] ASP.NET Core Authentication | DotVVM Documentation. [online]. [cit. 18. 03. 2018]. Dostupné z URL: <https://www.dotvvm.com/docs/tutorials/advanced-aspnetcore-authentication/1.1

[22] UserManager Class (Microsoft.AspNetCore.Identity) | Microsoft Docs. [online]. [cit. 20. 03. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.usermanager-1?view=aspnetcore-2.0

[23] RoleManager Class (Microsoft.AspNetCore.Identity) | Microsoft Docs. [online]. [cit. 20. 03. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.identity.rolemanager-1?view=aspnetcore-2.0

[24] Configure ASP.NET Core Identity | Microsoft Docs. [online]. [cit. 20. 03. 2018]. Dostupné z URL: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity-configuration?view=aspnetcore-2.0&viewFallbackFrom=aspnetcore-2.0tabs%3Daspnetcore2x&tabs=aspnetcore2x

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

SQL	Structured Query Language (strukturovaný dotazovací jazyk)
LINQ	Language Integrated Query (dotazovací jazyk)
HTML	Hypertext Markup Language (značkovací jazyk)
CSS	Cascading Style Sheets (Kaskádové styly pro tvorbu HTML stránek)
MVVM	Model View View Model (návrhový vzor)
MVC	Model View Controller (návrhový vzor)
MVP	Model-View-Presenter (návrhový vzor)
SSMS	SQL Server Management Studio
EF	Entity Framework
SPA	Single Page Application
JSON	JavaScript Object Notation

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO DVD

- `/` ..... kořenový adresář přiloženého DVD
  - `Dokumentace` ..... elektronická verze bakalářské práce
    - `xkozhu00-bp.pdf`
  - `ElectroShop` ..... webová aplikace pro sklad elektronických součástek
    - `ElectroShop.sln` ..... spouštěcí soubor pro Visual Studio
  - `Ukázky projektů`
    - `TestDotvvmApp` ..... ukázková aplikáce
      - `TestDotvvmApp.sln` ..... spouštěcí soubor pro Visual Studio
  - `ReadMe.txt` ..... textový soubor obsahující komentář ke spuštění programu