TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Vývoj desktopové aplikace pro obsluhu obráběcího stroje

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Lukáš Honke**
Vedoucí práce: Ing. Pavel Herajn

Liberec 2017

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Desktop application development for the machine tool

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Lukáš Honke**
Supervisor: Ing. Pavel Herajn

Liberec 2017

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Lukáš Honke**
Osobní číslo:	**M14000028**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Vývoj desktopové aplikace pro obsluhu obráběcího stroje**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se s tvorbou grafických aplikací v jazyce C# přizpůsobených pro dotykové ovládání na platformě .NET Framework, s ukládáním dat do SQL databáze a možnostmi práce s PDF soubory.
2. Navrhněte tvorbu nadstavbového grafického prostředí, uživatelského systému i aplikací Dokumentace, Kalendář a Technologická kalkulačka.
3. Realizujte aplikace.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **30–40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **AGARWAL, Vidya Vrat a James HUDDLESTON. Databáze v C# 2008: průvodce programátora. Vyd. 1. Brno: Computer Press, 2009. ISBN 978-80-251-2309-6.**

[2] **C#: programujeme profesionálně. 1. vyd. Brno: Computer Press, 2003. ISBN 80-251-0085-5.**

Vedoucí bakalářské práce: **Ing. Pavel Herajn**
Ústav mechatroniky a technické informatiky

Konzultant bakalářské práce: **Ing. Jiří Švéda, Ph.D.**
ČVUT FS

Datum zadání bakalářské práce: **10. října 2016**

Termín odevzdání bakalářské práce: **15. května 2017**

Technická univerzita v Liberci
L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2016

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 12.5.2017

Podpis:

Poděkování

Tímto bych rád poděkoval panu doktorovi Ing. Jiřímu Švédovi za poskytnuté konzultace a rady z praxe. Rád bych také poděkoval panu Ing. Tomášovi Kozlokovi za zprostředkování celého projektu a umožnění mi být jeho součástí. Dále bych rád poděkoval svému vedoucímu práce, panu Ing. Pavlu Herajnovi, za poskytnuté rady a připomínky. V neposlední řadě děkuji své přítelkyni a rodině za trpělivost a podporu během mého studia.

Abstrakt

Cílem bakalářské práce je navrhnout a realizovat vybrané dílčí aplikace informačního systému pro nově vyvinutý obráběcí stroj společnosti TOS VARNSDORF a.s. Jedná se o aplikaci pro správu a zobrazení dokumentů, aplikaci pro správu a plánování událostí a aplikaci pro zobrazení obrazu z IP kamery a její ovládání. Součástí práce je také vytvoření hlavní obrazovky, která bude sloužit jako rozhraní mezi dílčími aplikacemi. Je kladen důraz na jednoduchost a intuitivnost ovládání.

Klíčová Slova:

Vývoj aplikací, informační systém, desktopové aplikace, .NET Framework, C#, WPF

Abstract

The goal of this thesis is to design and implement selected component applications for information system for new machine tool developed by company TOS VARNSDORF a.s.. Following applications are required: application for managing and displaying documents, application for managing and scheduling events and application for rendering video from IP camera and controlling its movement. Part of the thesis is focused on creation of main screen, which will be used as an interface between component applications. Emphasis is placed on making the user interface simple and intuitive.

Keywords:

Application development, information system, desktop applications, .NET Framework, C# WPF

Obsah

Seznam zkratek

- .NET soubor technologií v softwarové platformě .NET Framework
- UI uživatelské rozhraní („user interface“)
- WPF Windows Presentation Foundation
- WinForms Windows Forms
- XML Extensible Markup Language
- XAML Extensible Application Markup Language
- SDK Software development kit
- SW Software
- PDF Portable Document Format

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

V dnešní době se výrobci nových obráběcích strojů snaží odlišit a nabídnout ke svým strojům přidanou hodnotu. U zavedených výrobců, kteří mají mechanickou strukturu dostatečně propracovanou dlouholetým vývojem, se v samotné stavbě strojů příliš velkých změn nedá očekávat, mění se zejména funkcionalita jejich ovládání. Jednou z cest, jak zaujmout zákazníky, je pak vybavit stroj softwarovým informačním systémem, který obsluze stroje usnadní práci. Řada výrobců toto už nabízí, například:

- Hyundai Wia – systém iTrol (inteligent control) [1]
- DMG MORI – systém CELOS [2]
- Okuma – systém OSP control [3]

Cílem této bakalářské práce je navrhnout a vytvořit vybrané dílčí aplikace obdobného systému pro nově vyvíjený obráběcí stroj WHT 110 C od společnosti TOS VARNSDORF a.s. [4].

Zadavatelem požadavků pro tento systém byla firma TOS VARNSDORF a.s. Vyvíjený systém byl pojmenován pracovním názvem TOSControl. Na základě rešerše zadavatele byla identifikována potřeba následující funkcionality a modulů:

1. Hlavní obrazovka s přehledem dostupných aplikací. Tato obrazovka bude pro uživatele výchozím bodem v systému TOSControl. Uživatel bude schopný na této obrazovce spustit jednotlivé aplikace, které po spuštění hlavní obrazovku překryjí. Obrazovka bude také umožňovat uživateli přepínání se mezi těmito aplikacemi.
2. Aplikace pro správu a zobrazení dokumentů. Tato aplikace bude schopná umožnit operátorovi obráběcího stroje otevírat technickou dokumentaci přímo v ovládacím panelu stroje. Dokumenty jsou obvykle k dispozici ve formátu PDF. Aplikace by měla být schopná nahradit konvenční aplikace pro zobrazování dokumentů v tomto formátu, například Adobe Acrobat. Mimo to by měla poskytovat možnost správy dokumentů, včetně přidávání nových dokumentů a omezování práv na určité skupiny dokumentů jen pro dané uživatele. Aplikace by měla být přizpůsobena pro jednoduché dotykové ovládání. Předpokládá se neznalost uživatele s prací v této aplikaci, je tedy nutné, aby bylo navržené uživatelské rozhraní co nejvíce jednoduché a intuitivní.
3. Aplikace pro správu události – kalendář. Tato aplikace bude schopná poskytnout uživateli možnost plánování událostí spojených s obráběcím strojem. Aplikace bude notifikovat uživatele o nadcházejících událostech. Nutné je ukládání události na sdílené databázové úložiště, aby bylo do budoucna možné doprogramovat vzdálený přístup ze strany společnosti TOS VARNSDORF a.s. Vzhled aplikace by měl být intuitivní a připomínat zjednodušenou verzi aplikace Microsoft Outlook.
4. Vizuální kontrola technologie (označeno jako technologická kalkulačka). Uživatel by si měl být schopný pomocí této aplikace zobrazit obraz z IP kamery, která je namontována na obráběcím stroji a k PCU jednotce je připojena pomocí ethernetového kabelu. Využívá se kamera typu FS-WD652-6105 (specifikace v [5]). Tato kamera umožňuje horizontální i vertikální otáčení, tato funkcionalita by měla v této aplikaci být zahrnuta také.

Tyto aplikace budou implementovány do průmyslového PC s operačním systémem Windows 7, společně s grafickým uživatelským prostředím řídicího systému Siemens Sinumerik 840D sl (Sinumerik Operate). Data pro komunikaci mezi řídicím systémem a navrhovanými aplikacemi jsou sdílena formou objektů využívajících systémové knihovny výrobce. Tvorba rozhraní pro výměnu dat s řídicím systémem ale není součástí této bakalářské práce.

2 Rešerše a seznámení s problematikou

2.1 Obecné SW nástroje

Pro vývoj aplikace bylo zvolena platforma Microsoft .NET Framework ve verzi 4.5. V této verzi jsou zahrnuty technologie, které jsou v současné době ve vývoji moderních aplikací velmi využívané. Jedná se například o LINQ (*Language Integrated Query*), což je knihovna, která umožní využívání funkcí, které efektivně manipulují s kolekcemi dat ve stylu příkazů v jazyce SQL. Zjednodušeně se dá říct, že LINQ poskytuje nový přístup k práci s kolekcemi dat. Jeho kód je tzv. deklarativní (definuje podobu dat, nedefinuje jakým způsobem se to má provést) a je protikladem ke klasickému imperativnímu přístupu pomocí *for* cyklů. [6]

Základem práce s LINQ je rozhraní *IEnumerable* [7], které je rozhraním pro všechny kolekce v jazyku C# a poskytuje enumerátor, který umožňuje iteraci přes prvky kolekce. Výhodou LINQ je, že jde o ucelený princip pro manipulaci s daty z různých zdrojů, k jehož psaní je potřeba znát pouze jazyk C#. Pro LINQ jsou totiž k dispozici různá rozšíření: LINQ to XML (transformuje LINQ výraz do dotazu, který přistupuje k DOM modelu XML formátu), LINQ to SQL (transformují LINQ výraz do SQL dotazu – je tedy možné přistupovat k databázovým entitám stejně, jako ke standardním kolekcím v jazyce C#), LINQ to Entities (užití LINQ dotazů v Entity Frameworku pro přístup do databáze), a další. Tyto rozšíření tedy umožňují pracovat s kolekcemi dat z různých zdrojů jednotným, přehledným a efektivním způsobem. Zdrojový kód 1 znázorňuje příklad užití LINQ to Entities pro vytáhnutí dat z relační databáze. Zdrojový kód 2 pak ukazuje, jak by se podobný problém dal řešit pomocí standardního SQL dotazu.

```
public static ObservableCollection<TosAppointment> GetCalendarAppointments(TosUser
user)
{
	var result = new ObservableCollection<TosAppointment>();
	var mapper = new TosAppointmentMapper();

	// vytvoření databázového kontextu (Entity Framework)
	using (var context = new TosEntities2())
	{
		// formulace databázového dotazu pomoci LINQ
		var entities = context.Events.Include(e =>
e.Category1).ToList().Where(c => c.Category1 == null || CanView(user,
c.Category1));

		// přemapování databázových entit do tříd, které je možné zobrazit
v Kalendáři
		IEnumerable<TosAppointment> models = mapper.Map(entities);
		result.AddRange(models);
	}

	return result;
}
```

Zdrojový kód 1: příklad užití LINQ to Entities pro získání událostí z databáze

```
public static ObservableCollection<TosAppointment>
GetCalendarAppointmentsAdo(TosUser user)
{
  var result = new ObservableCollection<TosAppointment>();
  var mapper = new TosAppointmentMapper();

  string sqlQuery = "SELECT * FROM[dbo].[Events] AS E LEFT OUTER " +
                    "JOIN[dbo].[Category] AS C ON[E].[Category] = [C].[Id]";

  SqlConnection connection = new SqlConnection(dbConnectionString);
  connection.Open();

  SqlCommand command = new SqlCommand(sqlQuery, connection);
  SqlDataReader dataReader = command.ExecuteReader();
  List<Event> entities = new List<Event>();

  if (dataReader.HasRows) {
    while (dataReader.Read())
    {
      // Create a new event
      Event eventEntity = new Event();
      eventEntity.Id = dataReader.GetInt32(0);
      eventEntity.Name = dataReader.GetString(1);
      // … načíst další atributy

      entities.Add(eventEntity);
    }
  }

  command.Dispose();
  connection.Close();
  connection.Dispose();

  result.AddRange(mapper.Map(entities.Where(c => c.Category1 == null ||
CanView(user, c.Category1)).ToList()));
  return result;
}
```

Zdrojový kód 2: příklad získávání dat z databáze pomocí klasického SQL dotazu

Další užitečnou technologií je PLINQ. Jde o rozšíření LINQ, které umožňuje spouštět nad *IEnumerable* objekty paralelní operace, čímž se celý proces může rozložit do více vláken a proběhne tak rychleji. Implementace takového procesu je mnohem jednodušší, přehlednější a mnohem méně náchylná na chyby, než klasické vytváření vláken a programování provádění paralelních operací nad kolekcí manuálně. Pro využití PLINQ stačí zavolat u příslušné implementace *IEnumerable* metodu *AsParallel()*, další klasická LINQ volání nad tímto objektem pak budou prováděna paralelně. [8]

Ve verzi 4.5 je navíc velmi dobře vyřešená podpora asynchronního programování, využívá se pro to klíčových slov *async* a *await*. Toho se dá využít při implementaci kódu, který při čekání na načtení či uložení vstupně-výstupních dat (nebo jiného procesu na pozadí aplikace), vrátí řízení zpět vláknu zajišťujícímu obsluhu uživatelského rozhraní, tím se jednoduše a efektivně zajistí, že se uživatelské rozhraní aplikace nezablokuje a zůstane pro uživatele responzivní. [9]

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
  // operace náročná na CPU
  // spustí se vláknem na pozadí v ThreadPool, UI vlákno asynchronně vyčká na
výsledek – nedojde k zablokování UI vlákna
  await Task.Run(() => CalculateData(data));

  // dlouho trvající I/O operace
  // samotnou uložení dat provede např. databázový server, aplikace asynchronně
vyčká na výsledek – nedojde k zablokování UI vlákna
  await this.SaveDatabaseData(data);

  MessageBox.Show(„Finished“);
}
```

Zdrojový kód 3: přiklad užití await pro asynchronni provedení metod v OnClick eventu

S rozhodnutím využít Microsoft .NET Framework se pojí použití programovacího jazyku C# jako primárního jazyku při implementaci aplikací. Vývoj probíhal ve vývojovém prostředí Microsoft Visual Studio 2015. Pro stahování a správu externích knihoven v projektu se využíval nástroj NuGET [10] - externí knihovny jsou pomocí tohoto distribuovány jako balíčky, které lze z oficiálního serveru stáhnout. Velkou výhodou tohoto nástroje je, že se automaticky postará o vyřešení případných závislostí jednotlivých knihoven na jiných knihovnách.

2.2 Nástroje pro tvorbu uživatelského rozhraní

Pro návrh a vykreslení uživatelského rozhraní byla využita technologie WPF (*Windows Presentation Foundation*) od Microsoftu. Prakticky jedinou moderní alternativou v tomto případě byla technologie Windows Forms, která je starší a existuje k ní spousta materiálů a prověřených postupů, které by se v práci mohly využít. WPF je však z hlediska návrhu modernější (umožňuje data binding – techniku automatické synchronizace dat, například mezi uživatelským rozhraním a datovou třídou v aplikaci), flexibilnější (umožňuje lépe přizpůsobit vzhled jednotlivých komponent aplikace, využívá k tomu jazyk XAML, v podstatě dovoluje přepsat většinu částí dané grafické komponenty) a obecně využívanější při vývoji nových aplikací [11] [12]. Při použití návrhového vzoru MVVM (*Model–view–viewmodel*) lze snadno oddělit vývoj grafického rozhraní (které se ve WPF píše v jazyce XAML, který je odvozený od jazyku XML) od hlavní logiky programu. Tyto důvody rozhodly o použití WPF. Je možné, že některé externí knihovny poskytující rozšířenou funkcionalitu pro grafické uživatelské rozhraní můžou své přídavky nabízet pouze pro technologii WinForms. U WPF je však toto vyřešené – WPF nabízí komponentu *WindowsFormsHost* [13], která umožňuje hostovat WinForms komponentu uvnitř WPF okénka. [14]

Definice vzhledu jednotlivých WPF komponent je obvykle obsažená v jejich příslušném XAML kódu. K upravení některých základních parametrů vzhledu (např. barva pozadí) stačí změnit hodnotu konkrétních parametrů, což se obvykle provádí vytvořením nového stylu (reprezentovaný typem *Style*), který aplikuje změny hodnot atributů na cílové komponenty (zdrojový kód 4) Upravit lze buď pouze konkrétní komponenty, nebo lze styl nastavit tak, aby se aplikoval na všechny komponenty určitých typů. [15]

```
<Style x:Key="CustomButtonStyle" TargetType="Button">
    <Setter Property="Background" Value="Black"/>
</Style>

<Button Content="Upravené tlačítko“ Style="{StaticResource
CustomButtonStyle}"/>
```

Zdrojový kód 4: úprava barvy pozadí tlačítka pomocí aplikování vlastního Style

Někdy je pro pokročilejší úpravy nutné pozměnit samotný ControlTemplate komponenty. ControlTemplate definuje, jakým způsobem je daná komponenta vykreslená v uživatelském rozhraní a z jakých částí (jiných komponent) se skládá a hodnoty pro parametry spjaté s grafickým vzhledem obvykle přejímá ze svých stylů. [16]

Jelikož je cílem bakalářské práce vytvořit aplikaci, která má jednoduché a intuitivní ovládání přizpůsobené pro dotyky prstem, bylo vhodné zvážit využití externích knihoven, které se věnují implementaci dodatečných a pokročilých komponent grafického uživatelského rozhraní. Příkladem může být pokročilá komponenta pro výběr data. Zadávání data uživatelem je obecným problémem z hlediska validace a parsování zadaných dat do jednotného formátu, se kterým se dále dá v programu pracovat (v .NET ramework jde konkrétně o strukturu *DateTime*). Použitím takovéto komponenty se však zajistí, že uživatel nemůže zadat nevalidní hodnotu, ověřování totiž provádí samotná komponenta během zadávání dat (datum navíc uživatel obvykle zadává klikáním, které je nastavené tak, aby bylo co nejvíce intuitivní). Obrázek 1 ukazuje, jak taková komponenta může vypadat v knihovně Telerik UI for WPF [17].

Mezi nejpopulárnější knihovny, které tyto komponenty poskytují pro WPF, patří Telerik UI for WPF [17], DevExpress pro WPF [18] nebo SyncFusion Essential Studio for WPF [19]. Všechny tyto knihovny mají pozitivní reference, nabízejí velmi podobný výběr komponent a žádné by nebyly špatnou volbou pro tento projekt. Na základě vyhodnocení vlastností, aktuálních zkušeností a preferencí zadavatele byly zvoleny komponenty od společnosti Telerik. Kompletní seznam komponent, které tento dodavatel poskytuje pro WPF, je popsaný v [20].

Obrázek 1: komponenta RadDateTimePicker z knihovny Telerik UI for WPF

Jako alternativu k vývoji v platformě Microsoft .NET Framework bylo možné zvážit využití technologie Java, resp. JavaFX [21] pro vykreslení uživatelského rozhraní. JavaFX umožňuje z části podobnou funkcionalitu, jako WPF (např. psaní uživatelského rozhraní v jazyce FXML, který je podobný jazyku XAML z WPF, techniku databinding, aj.). JavaFX pro nastavení grafických stylů uživatelských komponent využívá jazyk CSS (oproti WPF, kde se styly definují v jazyce XAML). Pro JavaFX však nejsou dostupné žádné profesionální uživatelské komponenty podobné např. Telerik UI for WPF, které by vyhovovaly požadavkům projektu. Tvorba uživatelsky atraktivního uživatelského rozhraní by tak byla náročnější. Cílem práce jsou navíc aplikace pouze pro operační systém Windows, takže multiplatformní schopnosti jazyka Java by se nevyužili. Z tohoto důvodu, a s přihlédnutím k aktuálním zkušenostem a preferencím zadavatele bylo rozhodnuto, že by tato možnost neposkytovala žádnou výraznější výhodu při realizaci projektu oproti použití Microsoft .NET Framework.

2.3 Databázový systém

Dále bylo nutné vybrat konkrétní databázový systém pro ukládání relačních dat. Mezi nejzajímavější a aktuálně nejvyužívanější systémy patří Windows SQL Server, MySQL a Oracle Database. [22] Pro práci s databází je efektivní využít techniku zvanou ORM (*Object - relational mapping*), což je princip konvertování dat mezi relačními databázovými systémy a objektově orientovanými programovacími jazyky. Konkrétně jde o třídy v kódu napsaném v jazyce C#. Využitím principu ORM je možné pracovat s databázovými entitami jako s objekty a odstraňuje se nutnost psát konkrétní databázové dotazy. Jedna z nejpopulárnějších technologií pro realizaci ORM v platformě Microsoft .NET Framework je Entity Framework [23], jehož autorem je Microsoft. Jelikož tento framework nativně podporuje především Windows SQL Server (a podpora pro jiné databázové systémy se musí ručně přidávat), bylo rozhodnuto ho v projektu využít právě v kombinaci se systémem Windows SQL Server.

2.4 Nástroje pro zobrazení PDF dokumentů

Součástí bakalářské práce je vytvořit aplikaci, která bude umožňovat vykreslování obsahu dokumentů ve formátu PDF. Vykreslování (tzv. renderování) PDF dokumentu je ale obecně poměrně složitý problém. První verze PDF formátu byla vydaná již v roce 1993 [24] a od té doby prošel tento formát obrovským vývojem, během kterého se stával čím dál více komplexním. Důkazem složitosti může být i to, že oficiální příručka obsahující kompletní technické specifikace formátu má 1310 stránek [25]. Navrhnout vlastní spolehlivé řešení pro vykreslování PDF dokumentů by tak překračovalo rozsah této bakalářské práce. Proto se rozhodlo o využití externích knihoven.

Na základě průzkumu trhu byly vyhodnoceny následující možnosti:

- Foxit PDF SDK [26] – rozsáhlé populární profesionální SDK pro komplexní práci s PDF soubory. Toto SDK je vyvíjené společností Foxit Software Inc., která se vývoji software pro zpracování PDF souborů věnuje již od roku 2001.
- Adobe PDF Library SDK [27] – profesionální SDK od tvůrců PDF formátu, společnosti Adobe Inc. Nabízí podobný výčet možností, jako Foxit PDF SDK, avšak za vyšší cenu.
- Debenu Quick PDF Library [28] – profesionální SDK od zavedené společnosti Debenu, která je vlastněná společností Foxit Software Inc. Toto SDK tedy využívá podobných technologií pro vykreslování PDF souborů, jako Foxit PDF SDK.
- Foxit Viewer for .NET SDK [29] – jednodušší profesionální řešení dostupné pouze pro .NET Framework. Nejde o klasické SDK, ale o funkční WinForms komponentu, která umí vykreslovat PDF soubory. Staví na technologiích od společnosti Foxit Software Inc., jde tedy o rychlé a spolehlivé řešení. Negativem může být horší přizpůsobitelnost grafického vzhledu – jednotlivé části komponenty jsou již naprogramované a je obtížné je změnit. Jelikož také nejde o SDK, je další práce s PDF soubory pomocí této knihovny velmi omezená (nelze například extrahovat text nebo vytvořit obrazový náhled z dokumentu).
- Knihovna PDFiumViewer [30] – open-source knihovna pro zobrazení PDF souborů, distribuovaná pod licencí Apache 2.0. Vychází z open-source projektu PDFium, jehož vývoje se účastní i společnost Google. Zobrazovací technologie projektu PDFium je částečně založená na profesionálních technologiích od Foxitu [31]. Knihovna je pak charakterem podobná Foxit Viewer for .NET SDK, neboť poskytuje z velké části funkční WinForms komponenty, uvnitř kterých lze PDF soubory zobrazovat. Dle provedeného testování na vzorku vlastních PDF dokumentů bylo i toto řešení vyhodnoceno jako kvalitní a bylo možné ho využít. Nespornou výhodou profesionálních komerčních knihoven je ale to, že se s jejich zakoupením váže nárok na technickou podporu v případě problémů a budoucí aktualizace od dodavatele. Po konzultaci se společností TOS VARNSDORF a.s. se proto rozhodlo o zakoupení komerční knihovny.

Na základě porovnání vlastností a testování knihoven bylo zvoleno využití knihovny Foxit PDF SDK. Tato knihovna poskytuje obrovský výčet možností pro práci s PDF soubory a během testování poskytovala nejlepší výstup při extrakci textu. Levnějším alternativním řešením by pak mohla být knihovna Foxit Viewer for .NET SDK, ta ale nepokrývá všechny oblasti požadované funkcionality, musely by se tak vytvářet různé menší kompromisy.

3 Cíle práce

Na základě zadání práce a rešerše byly stanoveny následující cíle bakalářské práce:

1. Seznámení se s tvorbou grafických aplikací v jazyce C# pro potřeby dotykového ovládání na platformě .NET Framework, včetně propojení do SQL databáze
2. Návrh, realizace a implementace dílčích aplikací pro nově vyvíjený informační systém stroje
   - Hlavní obrazovka a uživatelský systém
   - Dokumenty
   - Kalendář
   - Technologická kalkulačka – zobrazení technologického procesu

Základní požadavky na dílčí aplikace:

Hlavní obrazovka

- Obrazovka bude sloužit jako vstupní bod do systému TOSControl, zobrazí se tedy defaultně po jeho spuštění. Na obrazovce budou vidět ikony aplikací, které po klepnutí otevřou odpovídající aplikační modul.
- Součástí obrazovky bude oddělený boční panel, který bude vždy viditelný, nezávisle na tom, jaký aplikační modul je zrovna spuštěný. Tento panel slouží k rychlému přechodu zpět na hlavní obrazovku, pomocí něj je tedy možné přepínat mezi aplikacemi. V panelu je dále informace o tom, pod jakým uživatelským účtem je aktuální uživatel přihlášený (viz kapitola 4.4) a tlačítko, pomocí kterého se uživatel může přihlásit či odhlásit.
- Nad všemi moduly TOSControl bude fungovat uživatelský systém, kde uživatelé jsou členové předem definovaných skupin. Jejich skupina poté určuje, jaká mají práva v systému.

Dokumenty

- Aplikace bude schopná otevírat PDF dokumenty a zobrazovat jejich obsah v uživatelsky přívětivé formě. Měla by být plnohodnotnou náhradou za konvenční čtečky PDF dokumentů (např. Adobe Acrobat), mezi umožňované funkce tedy bude patřit: prohledávání v dokumentu, extrahování textu, přibližování, oddalování, rychlý přechod na konkrétní stránku v dokumentu a načtení stromového obsahu dokumentu, včetně rychlého přechodu na kapitoly obsahu. Pokud je načtený dokument interaktivní (obsahuje hypertextové odkazy), bylo by vhodné, aby interaktivita fungovala i při zobrazení dokumentu v této aplikaci.
- Dále bude možné v rámci aplikace dokumenty spravovat: přidávat nové, upravovat a mazat stávající, třídit do kategorií (kategorie dokumentu udává, jaké uživatelské skupiny budou moct editovat nebo zobrazit daný dokument), jednotlivým dokumentům bude možné přiřadit klíčová slova, na základě kterých potom půjde nad seznamy dokumentů vyhledávat. Uživatel si může dokumenty přidávat do své složky „Oblíbené dokumenty“. Uvnitř dokumentů může uživatel vytvořit „záložky“, která budou vázané na konkrétní stránku, může si tak ukládat zajímavá místa v dokumentu a později se k nim vrátit.
- Bude možné otevírat pouze dokumenty přidané v rámci této aplikace, dokumenty budou zobrazené v „mřížkovém“ seznamu, kde u každého dokumentu bude vygenerovaný obrazový náhled.

Kalendář

- Aplikace bude umožňovat plánování událostí ve stylu kalendáře. Události bude možné pojmenovat, přiřadit jim popis, definovat čas počátku a čas konce (případně nastavit na „celý den“). Bude možné definovat, jak brzy před začátkem události aplikace upozorní uživatele na nadcházející událost. Události bude možné kategorizovat, kategorie událostí budou definovat, které uživatelské skupiny mají práva událost vidět a editovat. Události musí být uloženy v databázi, aby byla do budoucna umožněna implementace „vzdáleného přístupu“ (vzdálené plánování události do kalendáře zákazníkům).
- Aplikace musí umožňovat alespoň následující pohledy na seznam událostí: měsíční přehled, týdenní přehled, denní přehled.
- Aplikace musí umět notifikovat uživatele o nadcházející události bez ohledu na to, jestli je aktuálně spuštěná, nebo se uživatel nachází v jiném aplikačním modulu systému TOSControl.

Technologická kalkulačka – zobrazení technologického procesu

- Pod pojmem technologická kalkulačka zadavatel práce definuje možnost sledování technologického procesu pomocí reálného obrazu z IP kamery. Toto představuje základní kámen technologické kalkulačky, na kterém bude zadavatel dále stavět aplikaci pro stanovení řezných podmínek procesu. Algoritmy pro stanovení vhodných řezných podmínek nejsou součástí bakalářské práce.
- Aplikace bude umožňovat zobrazit obraz ve formě videa z IP kamery připojené k obráběcímu stroji. Tuto kameru bude možné v aplikaci otáčet v horizontálním i vertikálním směru. Implementace v této bakalářské práci počítá s kamerou typu FS-WD652-6105 [5].
- Součástí aplikace bude plovoucí „pop-up“ okno, které bude zobrazovat zmenšený obraz z IP kamery. Toto okno půjde vyvolat z postranního panelu Hlavní obrázky a bude tedy k dispozici vždy, nezávisle na tom, jaký modul je v systému TOSControl právě aktivní.

Všechny aplikace by měly být uzpůsobené pro ovládání na dotykové obrazovce, která navíc nepodporuje snímání dotyku více prstů na obrazovce najednou. Také se předpokládá, že uživatel není s aplikacemi předem seznámen, aplikace je tedy nutné navrhnout tak, aby její ovládací prvky byly co nejvíce intuitivní.

4 Návrh architektury softwarového řešení

Jednotlivé aplikace systému TOSControl jsou obvykle řešeny v oddělených třídách odvozených od *WPF třídy Window* [32]. *Window* je základní WPF třída, která definuje aplikační okno v OS Windows. Vnitřní struktura těchto tříd se nejčastěji řeší pomocí WPF komponenty *Grid* [33], která umožňuje definovat flexibilní mříž složenou z řádků a sloupců. Do jednotlivých buněk se pak vkládají vlastní komponenty. Šířka a výška jednotlivých buněk může být buď pevně daná (v pixelech), nebo může být adaptivní a buňka se roztáhne podle rozměru komponent uvnitř buňky. Počítá se s využitím jednoho typu dotykové ovládací obrazovky, která má neměnné rozlišení, proto i většina aplikací tvořených v rámci této bakalářské práce přiřazuje grafickým komponentám pevně dané rozměry.

V tomto projektu se textový vstup od uživatele obvykle řeší pomocí WPF komponenty *TextBox* [34] či *ComboBox* [35], pokud je výčet možností předem definován. Pro tlačítka se využívá WPF *Button* [36], jednotlivé akce se implementují pomocí eventu *OnClick*, který podporuje i dotykové ovládání. Event je metoda, která se zavolá, pokud se v uživatelském rozhraní stane nějaká definovaná událost, například uživatel klepne na tlačítko. Viditelnost prvků na stránce ve WPF ovlivňuje parametr *Visibility*, pokud je potřeba v aplikaci nějaký prvek za běhu skrýt, nastaví se na hodnotu *Collapsed*, v opačném případě platí hodnota *Visible*. Větší části uživatelského rozhraní, které je potřeba za běhu dynamicky měnit (například přechod mezi seznamem dokumentů a zobrazením jednoho dokumentu), se v tomto projektu umístí do samostatné WPF komponenty *Page* [37]. Pro zobrazení *Page* se pak využívá WPF komponenta *Frame* [38], ta umožňuje svou *Page* za běhu dynamicky měnit.

Alternativně se v projektu využívají vybrané WPF komponenty z knihovny UI for WPF od společnosti Telerik. Konkrétně jde o:

- RadComboBox [39] – podobné klasické WPF komponentě *ComboBox*, ale umožňuje tzv. „auto-complete“, tedy na základě postupného psaní uživatele se bude aplikace snažit uhádnout, co chce napsat a bude mu to ve formuláři předvyplňovat.
- RadMaskedInput [40] – podobné klasické WPF komponentě *TextBox*, ale umožní uživateli zadat vstupní data pouze v povoleném formátu.
- RadPasswordBox [41] – zadávání hesla, každé písmeno je zobrazeno jako znak hvězdy, dojde k skrytí skutečného obsahu.
- RadRibbonView [42] – přidává do aplikace lištu Ribbon [43]. Ribbon má podobu karet a tlačítek, které jsou horizontálně rozložené ve vrchní části okna, pomocí nich se dá aplikace ovládat. Tento ovládací prvek je uživatelům známý z aplikací Microsoft Word, Microsoft Outlook, aj.
- RadCalendar (RadTimePicker) [44] – komponenta umožňující pohodlně zadávat datum a čas pomocí klikání (resp. dotyků), uživatel nemusí nic psát ručně (tím není nutné vstupní data dále validovat).
- RadScheduleView [45] – umožňuje vizualizaci událostí kalendáře do mřížky ve stylu aplikace Microsoft Outlook. Umožňuje změnit aktuální časové období (měsíc, týden, den,…).

Pro přístup k třídám, které uchovávají kolekce dat určitého typu (například seznam uživatelů, uživatelských skupin, dokumentů, aj.) se využívá návrhový vzor

Singleton [46], který zabezpečí, že v celém programu nevznikne více, než jedna instance těchto tříd.

Pro zobrazení a práci s PDF soubory byla během rešeršní části (kapitola 2.4) zvolena knihovna Foxit PDF SDK. Po několika měsících vývoje (pomocí poskytnuté zkušební licence) se ale změnily rozpočtové podmínky projektu a zjistilo se, že se bude muset pro potřeby projektu zakoupit levnější knihovna. Rozhodlo se tak o implementaci čtení PDF souborů pomocí knihovny Foxit Viewer for .NET SDK. Důsledkem této změny je, že v projektu je implementované funkční řešení pomocí obou knihoven. Do budoucna je tak v případě zájmu možné přejít zpět na Foxit PDF SDK.

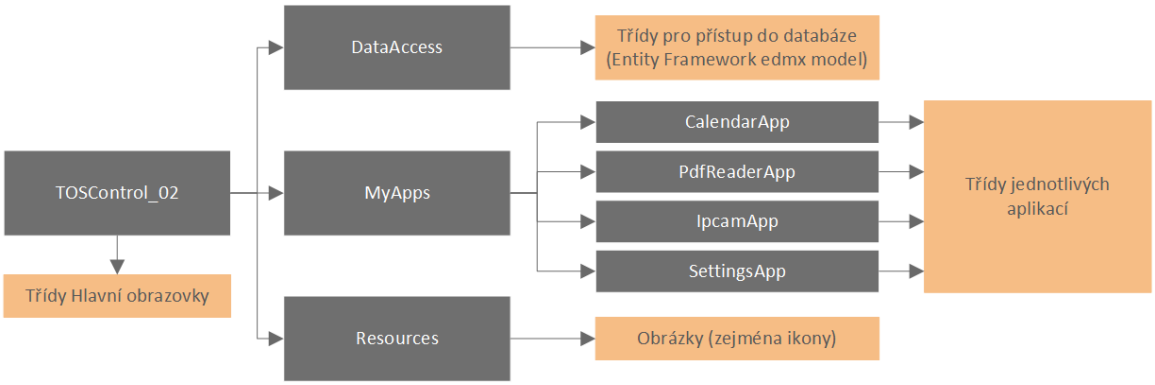
Správným způsobem vývoje by bylo oddělit návrh jednotlivých aplikací systému TOSControl od jejich implementace a návrh v této kapitole podrobněji rozepsat. Z důvodu postupného vývoje požadavků na funkcionalitu aplikací od zadavatele práce se však návrh aplikací nepodařilo spolehlivě oddělit od jejich implementace a oba procesy se z velké části prováděly současně.

4.1 Základní struktura programu

Třídy systému TOSControl jsou, jako u většiny aplikací v .NET Framework, členěny do jmenných prostorů (anglicky „namespaces“). Struktura jmenných prostorů pak odpovídá adresářové struktuře projektu. V rámci této bakalářské práce byl navržen a realizován následující způsob členění tříd:

- TOSControl_02: obsahuje hlavní třídu aplikace Úvodní obrazovka a další třídy s ní spojené. Jde o výchozí obrazovku systému TOSControl, proto se nachází v kořenovém jmenném prostoru.
- TOSControl.DataAccess: jmenný prostor pro třídy, které definují přístup k datům, převážně k relační databázi. Jde převážně o třídy automaticky generované pomocí Entity Framework, včetně databázového EDMX modelu.
- TOSControl.MyApps: jmenný prostor pro jednotlivé dílčí aplikace systému TOSControl. Každá aplikace má v tomto prostoru svou vlastní složku, ve které se nacházejí její třídy, způsob členění v rámci této aplikace je pak individuální pro každou aplikaci.
- TOSControl.Resources: v této složce jsou mediální soubory využívané pro stylování aplikace, zejména pak ikony. Některé ikony pochází z volně dostupného balíčku Modern UI Icons [47], jiné byly ručně vytvořeny pro potřeby tohoto projektu.

Z důvodu relativní jednoduchosti aktuální verze systému TOSControl zatím nebylo nutné členit aplikace do separátních projektů. V případě budoucího vývoje (například přidání nových aplikačních modulů) by bylo vhodné o tomto způsobu zauvažovat.

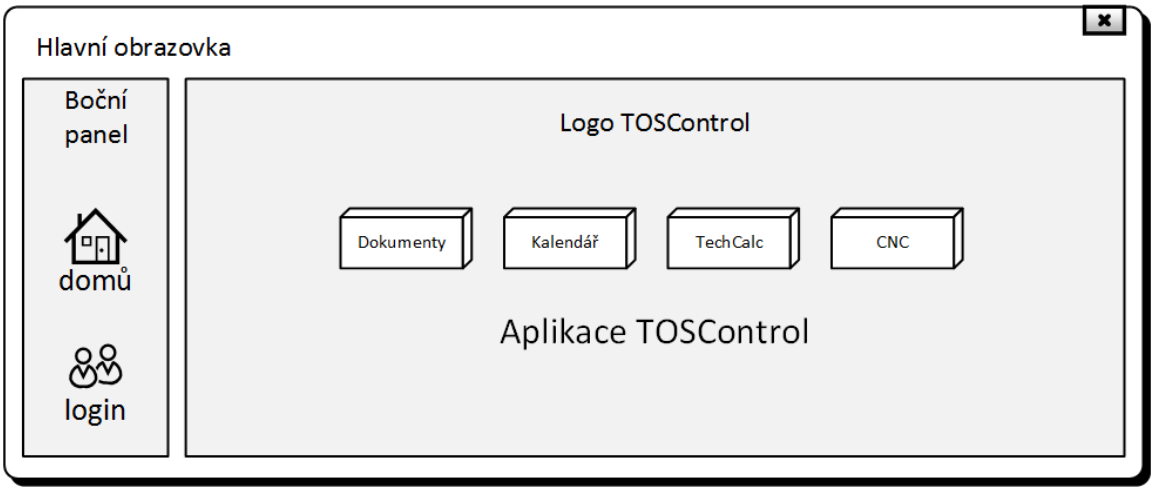


Obrázek 2: blokový diagram struktury TOSControl

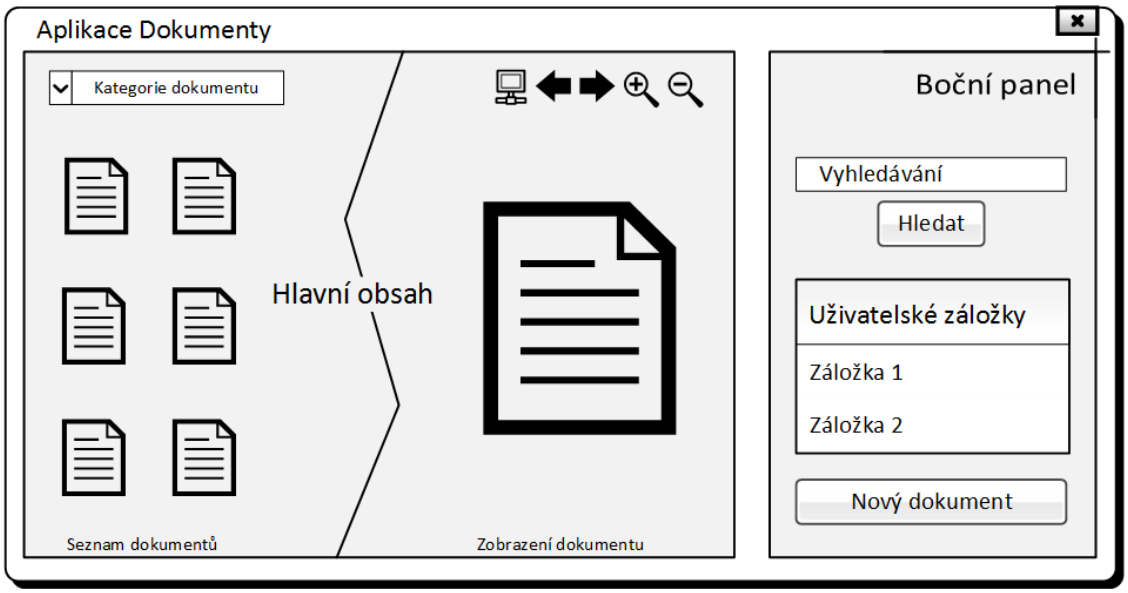
4.2 Uživatelské rozhraní

V rešeršní části byl popsán ideální způsob, jakým se ve WPF upravuje vzhled jednotlivých komponent – při jednoduchých úpravách se využívají styly a změny konkrétních atributů, při pokročilejších se využívá přímých úprav *ControlTemplate* komponent. Aby se v projektu udržel jednotný vzhled i pro budoucí přidané komponenty, byly navrženy styly, které se v projektu automaticky aplikují na vybrané komponenty: *Button*, *Label*, *TextBox*, *ComboBox*, *ScrollBar*. Tyto styly přednastaví zejména barvy pozadí prvků a barvy fontu textu, který se v nich nachází. V aplikacích projektu se využívají tmavé barvy pro pozadí komponent a světlé barvy pro text.

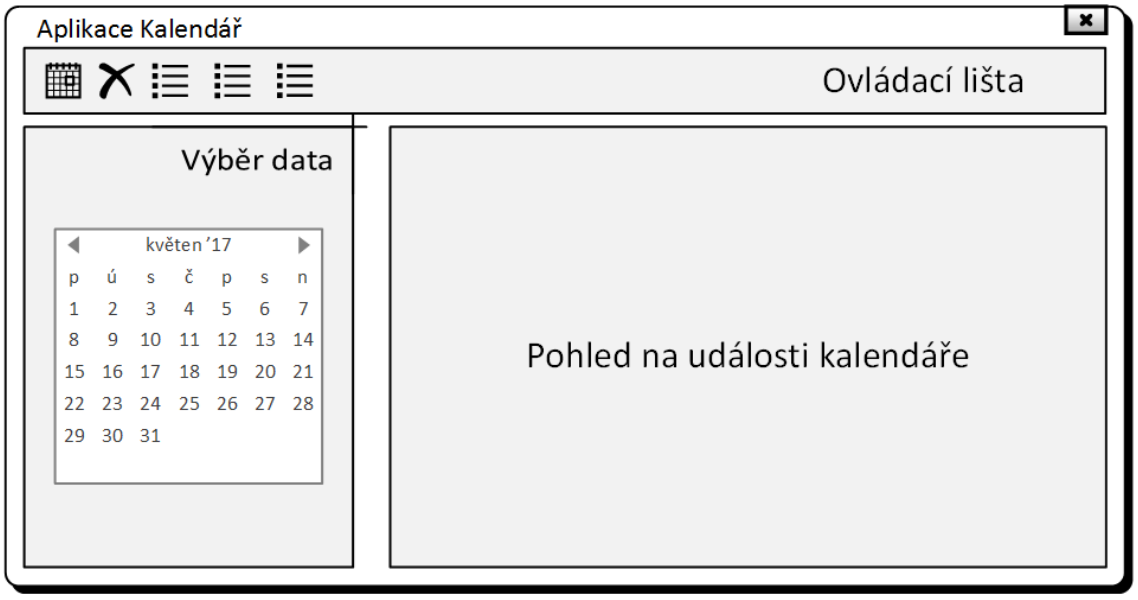
Při návrhu uživatelského rozhraní se muselo počítat především s tím, že aplikace budou mít dotykové ovládání, je tedy nutné, aby k tomu jednotlivé ovládací prvky byly velikostně přizpůsobené. Dotykový panel, pomocí kterého budou aplikace ovládané, navíc neumožňuje některé moderní funkce, zejména „multi-touch“, který uživatelé znají z chytrých mobilních telefonů. Bylo tedy nutné s tímto omezením při návrhu uživatelského rozhraní počítat. Výsledný návrh uživatelského rozhraní po konzultacích se zadavatelem práce je zobrazený v podobě drátěných modelů na obrázcích 3, 4 a 5.



Obrázek 3: drátěný model návrhu Hlavní obrazovky systému TOSControl



Obrázek 4: drátěný model návrhu uživatelského rozhraní aplikace Dokumenty



Obrázek 5: drátěný model návrhu uživatelského rozhraní aplikace Kalendář

4.3 Ukládání dat

Prvotní návrh systému ukládání dat nepočítal s relační databází, neboť se předpokládalo, že instalace databázového serveru na PCU jednotku obráběcího stroje nebude proveditelná. V tomto návrhu se všechna data ukládala do souborů typu XML. Nakonec se podařilo na PCU jednotku doinstalovat Microsoft SQL Server 2013, bylo tedy možné způsob ukládání dat přepracovat tak, aby se využíval tento databázový server.

Finální návrh provádí ukládání všech typů dat do SQL databáze. Výhodou tohoto přístupu je zachování integrity databáze pomocí definovaných integritních omezení, zejména cizích klíčů mezi jednotlivými tabulkami. Návrh databáze je diagramově popsán v kapitole 4.3.1. Jedinou výjimkou jsou datové soubory, které obsahují metadata k ukládaným dokumentům aplikace Dokumenty (popsáno v kapitole 5.2.1), ty jsou ukládané do souborů typu XML a umístěny ve stejné složce, ve které jsou i samotné

dokumentové soubory. Zadavatel chtěl zachovat možnost zasílat svým klientům dodatečné dokumenty s přednastavenými metadaty, to je díky tomuto způsobu ukládání možné, klient přijaté soubory pouze překopíruje do adresáře s ostatními dokumenty.

Na základě provedené rešerše bylo zvoleno využití databázového systému Microsoft SQL Server. K práci s daty je použita knihovna Entity Framework a tzv. „Database First" přístup [48]. Jde o přístup, kdy se nejprve navrhnou a vytvoří databázové tabulky (obvykle pomocí externích nástrojů a přímých SQL dotazů), poté se vytvoří propojení mezi knihovnou Entity Framework a existující databází a z ní se vytvoří tzv. „EDMX model". Jde o soubor, který překreslí strukturu existujících databázových tabulek a jejich vztahů do diagramového modelu (pomocí metody reverse engineering), tento diagramový model je zároveň plně editovatelný z Visual Studia a lze jeho prostřednictvím provádět v databázové struktuře změny, je-li to nutné. Po vytvoření tohoto modelu dojde k automatickému vygenerování tříd nutných pro práci s databází, pomocí kterých lze k jednotlivým databázovým entitám přistupovat jako ke klasickým objektům. Připojení do databáze je realizované třídou odvozenou od třídy DbContext [49] (implementaci pro konkrétní databázi vygeneruje Entity Framework), která v sobě obsahuje reference na všechny typy entit databáze, každý typ entity je reprezentována objektem typu DbSet<T>, kde T je typ samotného objektu [50]. S tímto objektem se pracuje podobně jako s klasickou kolekcí objektů. Zdrojový kód 5 znázorňuje implementaci DbContext pro databázi systému TOSControl, samotný návrh struktury databáze je v následující kapitole.

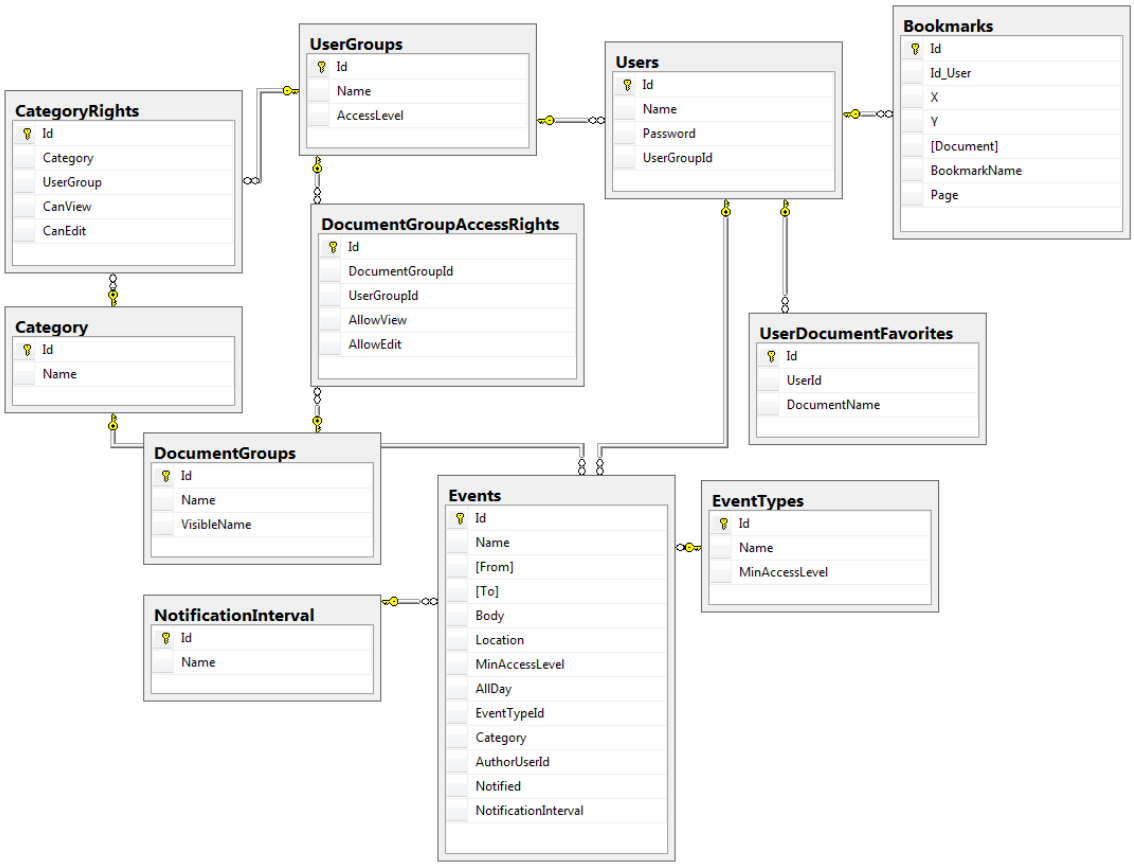
```
public partial class TosEntities2 : DbContext
{
    public TosEntities2()
        : base("name=TosEntities2"){}

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }

    public virtual DbSet<Bookmark> Bookmarks { get; set; }
    public virtual DbSet<Event> Events { get; set; }
    public virtual DbSet<EventType> EventTypes { get; set; }
    public virtual DbSet<UserGroup> UserGroups { get; set; }
    public virtual DbSet<User> Users { get; set; }
    public virtual DbSet<Category> Categories { get; set; }
    public virtual DbSet<NotificationInterval> NotificationIntervals { get; set; }
    public virtual DbSet<CategoryRight> CategoryRights { get; set; }
    public virtual DbSet<DocumentGroupAccessRight> DocumentGroupAccessRights
{ get; set; }
    public virtual DbSet<DocumentGroup> DocumentGroups { get; set; }
    public virtual DbSet<UserDocumentFavorite> UserDocumentFavorites { get; set; }
}
```

Zdrojový kód 5: implementace DbContext pro databázový model TOSControl

Pro manipulaci s daty se využívá technologie LINQ to Entities [48], který konkrétní programové dotazy interně přetransformuje do SQL příkazu, který odešle na SQL Server. Příklad použití této technologie je znázorněný ve zdrojovém kódu 1.

4.3.1 Návrh databáze

Dle schématu na obrázku 6 jsou uživatelé uloženi v tabulce „Users“. Každý uživatel patří k uživatelské skupině (tabulka „UserGroups“). Uživatel může mít vytvořené záložky v aplikaci Dokumenty (tabulka „Bookmarks“), své oblíbené dokumenty (tabulka „UserDocumentFavorites“) a vytvořené události v aplikaci Kalendář (tabulka „Events“). Každá událost má dobu, do které musí být uživatel před jejím začátkem upozorněn (tabulka „NotificationInterval“). Událost má dále svou kategorii (tabulka „Category“), kategorie události pak pomocí vazebné tabulky „CategoryRights“ definuje, jaké uživatelské skupiny mají právo k zobrazení a editování událostí konkrétních kategorií. Obdobně vazebná tabulka „DocumentGroupAccessRights“ slouží k definici práv uživatelských skupin k manipulaci s dokumenty v aplikaci Dokumenty.



Obrázek 6: UML diagram struktury databázových tabulek

4.4 Uživatelský systém

Navržen byl uživatelský systém, ve kterém je každý uživatel reprezentován jedním uživatelským účtem, ke kterému se přihlašuje pomocí uživatelského jména a hesla. Každý uživatelský účet má přidělený svou uživatelskou skupinu, která definuje jeho práva v jednotlivých aplikacích, tedy práva jsou řešena na úrovni uživatelských skupin. Uživatelské skupiny jsou předdefinované na výčet: „Worker“, „Technician“, „Admin“, do budoucna se však počítá s tím, že administrátor (člen skupiny „Admin“) bude schopný uživatelské skupiny vytvářet i upravovat jejich práva.

5 Realizace

V této bakalářské práci byly realizovány celkem 4 aplikační moduly TOSControl, které jsou popsány v následujících podkapitolách.

5.1 Úvodní obrazovka

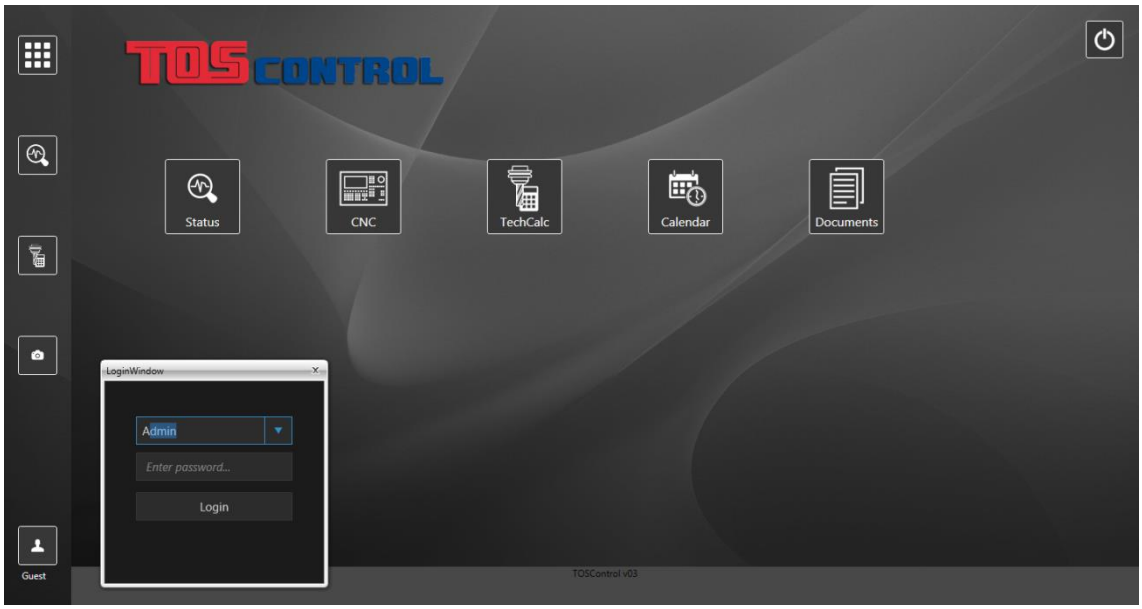
Aplikace úvodní obrazovka je realizována ve třídě *MainWindow*, která je odvozena od WPF třídy *Window*. Spustí se ihned při načtení systému TOSControl a po celou dobu zůstává aktivní, pouze se v případě potřeby, při přepínaní do jiných aplikačních modulů, přesouvá do pozadí či do popředí.

Její uživatelské rozhraní je realizované pomocí WPF komponenty *Grid*, která obrazovku rozděluje na fixní počet řádků a sloupců. První sloupec a první řádek jsou využity jako okrajové oblasti – nachází se v nich logo „TOSControl“ a zápatí aplikace. V dalších buňkách se pak nachází ikony, které reprezentují jednotlivé aplikace. Ikony jsou realizované pomocí WPF komponenty *Button*, které po kliknutí spustí odpovídající aplikační modul. Jednotlivé aplikační moduly zůstávají obvykle v paměti (ve třídě *MainWindow* jsou uloženy reference na jejich primární třídy) a nevypínají se, i pokud dojde k přepnutí aktivního aplikačního modulu. Výjimkou je pouze situace, kdy se uživatel odhlásí ze svého uživatelského účtu v systému TOSControl, pak se spuštěné aplikační moduly ukončit musí, důvodem pro to jsou načtená aplikační data, která by mohla být přístupná jen z konkrétního uživatelského účtu a je třeba je v případě odhlášení z aplikace skrýt. V případě, že aplikace je již načtená v paměti a uživatel znovu klikne na její ikonu na úvodní obrazovce, tak se její okno pouze zaktivuje (metodou *Window.Activate()*) a není třeba vytvářet novou instanci. Tímto způsobem je řešená logika přepínání mezi aplikačními moduly.

5.1.1 Postranní panel

Důležitou součástí úvodní obrazovky je postranní panel, který je pevně umístěný v levé části obrazovky. Je realizovaný jako samostatná třída *SidePanel*, která je odvozená od WPF *Window*. Má pevně danou šířku i výšku. Postranní panel je vždy viditelný, toho je docíleno především tak, že žádný jiný aplikační modul nezobrazí část svého aplikačního okna do oblasti, kde se postranní panel nachází. U každého aplikačního modulu systému TOSControl je ve třídě reprezentující jeho aplikační okno (tj. třída odvozená od WPF *Window*) nastavená hodnota parametru *WindowStartupLocation* na „*Manual*“ (a zároveň *Top=0*, *Left=86*), díky tomu je pak každé aplikační okno zobrazené těsně vedle postranního panelu a nedojde k jeho překrytí. Jednotlivá okna zároveň nelze přesouvat ani měnit jejich velikost (nastaveno pomocí parametru *ResizeMode*=„*NoResize*“).

Samotný obsah postranního panelu je vizuálně strukturován pomocí WPF komponenty *StackPanel*, která je nastavená tak, aby obsah řadila vertikálně. Obsahuje především tlačítka pro přechod zpět na úvodní obrazovku (zavolá metodu *Window.Activate()* u instance třídy *MainWindow*), otevření tzv. „pop-up“ okna pro zobrazení video obrazu z připojené IP kamery (viz kapitola 5.4.2) a přihlášení do systému pomocí uživatelského účtu (zobrazí okénko, kde uživatel vyplní uživatelské jméno a heslo). Obrázek 7 zobrazuje podobu spuštěné aplikace Úvodní obrazovka v prostředí systému TOSControl, včetně postranního panelu a otevřeného přihlašovacího okna.



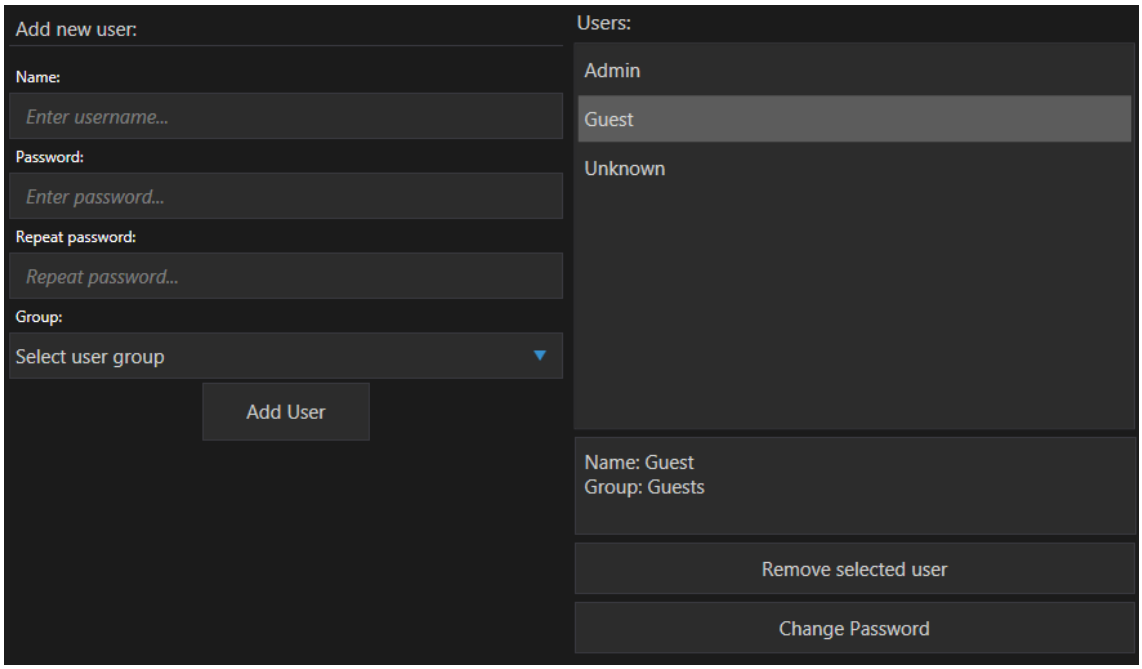
Obrázek 7: aplikace úvodní obrazovka, včetně otevřeného přihlašovacího okna

5.1.2 Uživatelský systém

Aplikace Hlavní obrazovka v sobě zahrnuje implementaci navrženého uživatelského systému. Uživatel je reprezentován třídou *TosUser*, načítání a správa uživatelů se provádí ve třídě *TosUserDatabase*, ta obsahuje zejména metody:

- *AddUser(string userName, string password, string groupName)*: slouží k přidání nového uživatele do databáze (pokud neexistuje uživatel se stejným uživatelským jménem).
- *RemoveUser(string userName)*: smaže uživatele podle uživatelského jména.
- *ChangePassword(TosUser user, string newPassword)*: změní heslo uživatele.
- *LoadUsers()* a *LoadUserGroups()*: načte existující uživatele a skupiny z databáze do paměti, toho je využíváno v předvyplňování formulářů.
- *Login(string name, string password)*: nastaví aktuálního uživatele používajícího systém TOSControl.
- *LogOut()*: resetuje aktuálního uživatele systému TOSControl.

Každý uživatel patří do nějaké uživatelské skupiny, která definuje jeho práva v aplikačních modulech TOSControl. Tato skupina je reprezentována třídou *TosUserDatabase*, každý uživatel si na svou skupinu uchovává referenci, současně je seznam skupin také ve třídě *TosUserDatabase*. Součástí realizovaného uživatelského systému jsou formuláře pro přihlášení do systému (zobrazený je na obrázku 7) a pro správu uživatelských účtů v systému administrátorem (obrázku 8).



Obrázek 8: jednoduché prostředí pro správu uživatelských účtů

5.2 Aplikace Dokumenty

Všechny hlavní třídy této aplikace jsou uloženy ve jmenném prostoru TOScontrol_02.MyApps.PdfReaderApp. Hlavní okno aplikace je řešené ve třídě *MainDocumentWindow*, která je odvozená od *WPF* třídy *Window*.

Struktura *MainDocumentWindow* je řešena pomocí komponenty *Grid*. Ta rozděluje celé okno na dva sloupce v poměru 3:1. První sloupec je vyhrazen pro hlavní obsah aplikace Dokumenty, druhý sloupec je pak vyhrazený prostor pro boční panel. Hlavním obsahem je buď pohled na seznam dostupných dokumentů (popsán v kapitole 5.2.2), nebo pohled na obsah jednoho konkrétního dokumentu (popsán v kapitole 5.2.3). Obsah bočního panelu v druhém sloupci okna se při běhu aplikace dynamicky mění (popsáno více v dalších kapitolách). Boční panel lze skrýt, čímž dojde k nastavení šířky prvního sloupce na 100% šířky okna aplikace.

Implementace bočního panelu je v samotné třídě *MainDocumentWindow*. Veškerý obsah je v ovládacím prvku *StackPanel*, který je nastavený tak, aby vnitřní prvky vykresloval pod sebe (vertikální řazení). V bočním panelu jsou zahrnuté následující ovládací prvky:

- vyhledávání (pokud je aktuálně zobrazený pohled na seznam dokumentů, pak se prohledává celý seznam dle klíčových slov dokumentů, pokud je aktuálně vykreslen jeden konkrétní dokument, pak se provádí fulltextové vyhledávání pouze v něm). Vstupní data se zadávají komponentou *TextBox*, provedení vyhledávání a přechod na další či předchozí výsledek se provede klepnutím na odpovídající *Button*.
- Přehled záložek aktuálního uživatele (využívá se *ListBox* pro vykreslení seznamu záložek, na kolekci jeho dat se aplikuje filtrování, které v případě, že je v aplikaci otevřený jeden konkrétní dokument, tak v tomto seznamu zobrazí pouze záložky

vztažené k tomuto dokumentu). Součástí přehledu je *TextBox*, který zobrazuje informace o aktuálně vybrané záložce, dále tlačítko pro přechod na vybranou záložku a tlačítko pro smazání vybrané záložky. Při pokusu o smazání záložky je pomocí *MessageBox.Show* dialogovým oknem ověřeno, jestli uživatel chce záložku skutečně smazat.

- Tlačítko na přidání dokumentu – po klepnutí zobrazí formulář, který umožní přidat nový dokument do aplikace. Realizace tohoto formuláře je popsána v kapitole 5.2.1. Tlačítko je viditelné (parametr *Visibility* má hodnotu *Visible*) pouze v případě, že je v aplikaci aktivní pohled na seznam dokumentů.
- WPF komponenta *TreeView*, která zobrazuje obsah aktuálního dokumentu načtený přímo z PDF souboru. Je skrytá, pokud je v aplikaci aktivní pohled na seznam dokumentů. Obsah se načítá pomocí knihoven na zpracování PDF souborů. Jelikož je obsah dokumentu stromová struktura, jejíž každý prvek obsahuje reference na své podkapitoly, je naplnění *TreeView* daty pro zobrazení obsahu dokumentu řešeno rekurzivně. Způsob plnění daty (pro užitou knihovnu Foxit PDF Viewer for .NET SDK) je popsaný ve zdrojovém kódu 6.

V prvním sloupci *MainDocumentWindow* (levá část okna) je jediným prvkem WPF komponenta *Frame*, jejímž obsahem je objekt WPF třídy *Page*. Aktuální objekt *Page* se mění podle toho, který pohled v aplikaci je momentálně zobrazený (pohled na seznam dokumentů či pohled na obsah konkrétního dokumentu) a tento pohled zároveň realizuje. Obsah *Frame* se nastavuje metodou *Frame.Navigate(object content)*. Tím se docílí toho, že se do levé části *MainDocumentWindow* mohou dynamicky dosazovat různá podokna. Realizovaná podokna aplikace Dokumenty jsou popsaná v kapitolách 5.2.2 a 5.2.3.

```
...
// Process top-level (root) bookmarks
foreach (Foxit.PDF.Viewer.Bookmark rootBm in pdfViewerDocument.Bookmarks)
{
	var addedItem = new BookmarkItem() { Header = rootBm.Text, Page =
(rootBm.PageNumber-1) };
	ProcessBM(rootBm, addedItem);
}
...

// Processes single Bookmark node
private void ProcessBM(Foxit.PDF.Viewer.Bookmark bm, TreeViewItem parentItem)
{
	// Save bookmark to custom BookmarkItem class
	var addedItem = new BookmarkItem() { Header = bm.Text, Page =
(bm.PageNumber-1)};

	// Put it to the TreeView
	parentItem.Items.Add(addedItem);

	// Process child Bookmarks
	foreach (Foxit.PDF.Viewer.Bookmark childBm in bm.Bookmarks)
	{
		ProcessBM(childBm, addedItem);
	}
}

// Wraps a single Bookmark which contains info about page number, inherits from
TreeViewItem
public class BookmarkItem : TreeViewItem
{
	public int Page { get; set; }
}
```

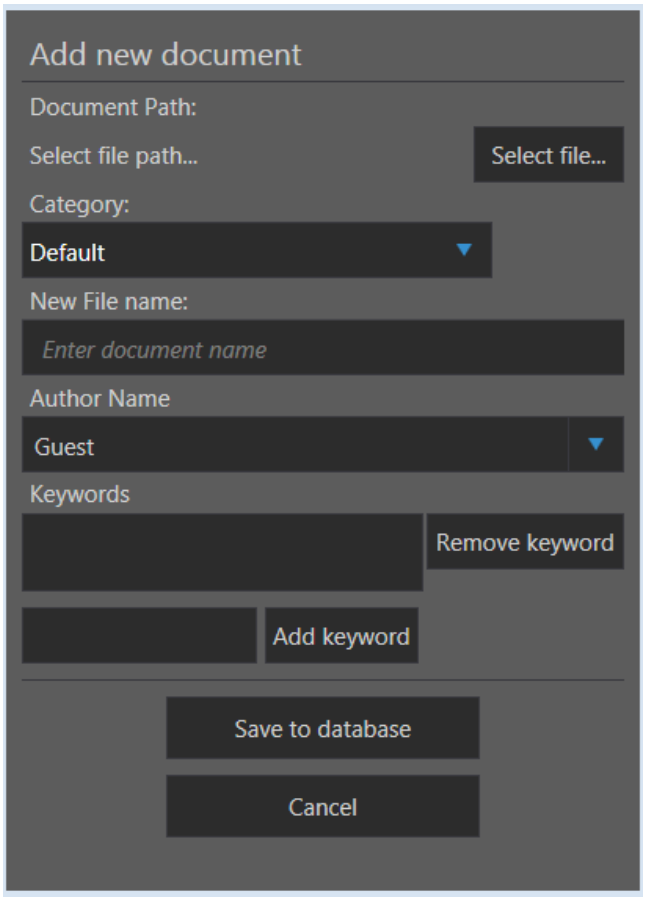
Zdrojový kód 6: způsob plnění TreeView pro zobrazení stromového obsahu dokumentu

5.2.1 Přidání nového dokumentu

Pro přidání nového i upravení stávajícího dokumentu se využívá stejný formulář. Pokud se ve formuláři nějaký stávající dokument upravuje, pak jsou určité hodnoty před zobrazením uživateli předem vyplněny.

Obsah formuláře je strukturován pomocí *StackPanel*, který je nastaven tak, aby prvky řadil pod sebe. Kromě nastavení jména dokumentu (řešeno pomocí TextBox) je třeba od uživatele získat cestu k originálnímu PDF souboru dokumentu, to se řeší (po kliknutí na tlačítko „Select file“) vytvořením instance třídy *OpenFileDialog*, které se nastaví parametr *Filter* na hodnotu *„PDF files (*.pdf)|*.pdf|All files (*.*)|*.*“*, poté se dialog zobrazí uživateli (zavoláním *OpenFileDialog.ShowDialog()*) a následně je z vráceného objektu *DialogResult* získána cesta k souboru. V případě, že se formulářem přidává nový dokument, tak se tento soubor zkopíruje do datového úložiště aplikace Dokumenty a zároveň se přejmenuje na ve formuláři uživatelem zadané jméno dokumentu. Výběr kategorie dokumentu je řešený pomocí komponenty *ComboBox*, které se předá seznam všech dostupných dokumentových kategorií v aplikaci. K dokumentu se dále dají přidat klíčová slova, to je řešené pomocí kombinace komponent *ListBox* (zobrazuje aktuálně přidaná slova), *TextBox* pro zadávání nového klíčového slova uživatelem a tlačítka „Add keyword“. Klíčová slova je možné po označení mazat tlačítkem „Remove keyword“.

Po potvrzení formuláře se k danému dokumentu vytvoří XML soubor, který obsahuje informace načtené z formuláře – metadata dokumentu. Tento XML soubor je pojmenován stejně, jako soubor dokumentu a je umístěn ve stejné složce. Pokud ve formuláři dojde k upravení metadat existujícího dokumentu, je jeho původní XML soubor novým souborem přepsán. Je-li je účelem formuláře upravení metadat stávajícího dokumentu, pak je součástí formuláře také tlačítko „Delete document“, které dokument smaže (včetně zdrojového PDF souboru v datovém úložišti aplikace).



Obrázek 9: formulář pro přidání a úpravu dokumentu

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
  <visible_name>SINUMERIK 840D sl / 828D</visible_name>
  <author>Admin</author>
  <group>Default</group>
  <added_date>18.09.2016 23:28:20</added_date>
  <keywords>
    <keyword>Siemens</keyword>
    <keyword>Machine</keyword>
    <keyword>Manual</keyword>
  </keywords>
</document>
```

Zdrojový kód 7: reprezentace metadat dokumentu v XML souboru

5.2.2 Seznam dokumentů

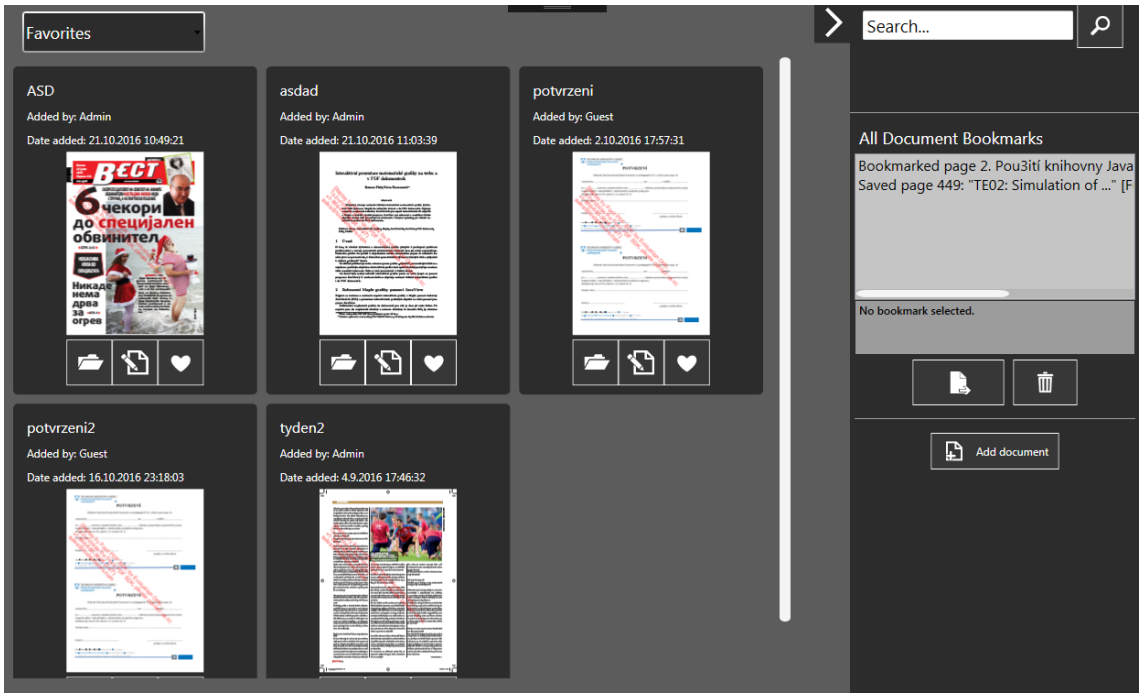
Realizace zobrazení pohledu na seznam dokumentů je řešena ve třídě *ViewDocuments*, která je odvozena od WPF třídy *Page*. Tato třída v levém horním rohu zobrazuje *ComboBox*, který obsahuje všechny dostupné kategorie dokumentů. Zbytek stránky pak tvoří seznam dokumentů uložený do čtvercové matice, která je řešená pomocí WPF komponenty *UniformGrid*. Ta je zároveň obalená ovládacím prvkem *ScrollViewer*, čímž dojde k zajištění toho, aby v seznamu dokumentů šlo „scrollovat" (pokud je seznam rozměrově větší než je velikost aplikačního okna). Seznam dokumentů se dynamicky mění podle toho, která kategorie je zrovna vybraná v *ComboBoxu* v levém horním rohu. Pokud uživatel zadá v postranním panelu aplikace do vyhledávacího pole nějaký výraz, pak dojde k vyfiltrování seznamu dokumentů jen na dokumenty, které tento výraz obsahují buď ve svém názvu, nebo v nějakém ze svých klíčových slov – k porovnávání se využívá metoda *string.Compare* s parametrem *InvariantCultureIgnoreCase*. Současně se zobrazují pouze dokumenty, které má aktuálně přihlášený uživatel oprávnění vidět. Uživatelská práva se v aplikaci Dokumenty řeší na úrovni skupin. Uživatelé mají pouze ta práva, která má jejich uživatelská skupina a zároveň všechny dokumenty v jedné skupině dokumentů mají stejnou důležitost. Uživatel je může vidět buď všechny, nebo nemůže vidět žádný.

Každý dokument v seznamu je reprezentován jedním objektem typu *StackPanel*, který obsahuje následující informace o dokumentu: viditelné jméno dokumentu, informaci o uživateli, který dokument přidal, datum přidání dokumentu, vygenerovaný náhled na dokument a trojici tlačítek „Open document" (otevře dokument, přejde do režimu zobrazení jednoho dokumentu; dokument lze otevřít i po kliknutí na jeho obrázkový náhled), „Edit document" (otevře formulář na úpravu dokumentu, viz kapitola 5.2.1), „Save to favorites" (uloží dokument do seznamu oblíbených dokumentů aktuálně přihlášeného uživatele). Tlačítko „Edit document" je viditelné pouze v případě, že má uživatel právo daný dokument upravovat. Aby došlo k vizuálnímu odlišení, je každý StackPanel umístěn do WPF komponenty Border, která okolo vykreslí grafický rámec.

V pravém bočním panelu aplikačního okna je seznam záložek, který, pokud je v aplikaci aktivní pohled na seznam dokumentů, zobrazuje sjednocení všech záložek ze všech dokumentů, které aktuálně přihlášený uživatel vytvořil.

K načítání dostupných dokumentů se využívá třída *TosDocumentDatabase.* Tato třída po spuštění aplikace Dokumenty načte do paměti všechny informace o všech dokumentech, které má k dispozici ve svém datovém úložišti. Každý dokument musí mít k dispozici svůj odpovídající XML soubor, který obsahuje metadata (název dokumentu, datum přidání, aj.). Pokud takový soubor pro daný dokument neexistuje, tak se při načítání všech dokumentů vytvoří nový a jednotlivým atributům se přiřadí defaultní hodnoty (název dokumentu se zkopíruje ze jména souboru, autor se nastaví na „Unknown", přiřadí se výchozí dokumentová skupina, datum přidání dokumentu se nastaví na aktuální datum). Tím je možné přidávat dokumenty do aplikace i tak, že se soubor ručně zkopíruje do složky datového úložiště aplikace Dokumenty. Třída *TosDocumentDatabase* také načte seznam existujících skupin dokumentů ze souboru DocumentGroups.xml. Součástí *TosDocumentDatabase* jsou také metody na manipulaci s dokumenty v datovém úložišti, konkrétně jde o:

- *bool CopyAddDocument(string path, string customName, string categoryName, string newAuthor, List<string> keywords)* – přidání nového dokumentu, včetně zkopírování do datového úložiště aplikace a kontroly validity vstupních údajů. Pokud se dokument nepodaří přidat, metoda vrátí hodnotu *false*.
- *void DeleteDocument(TosDocument document)* – odstranění dokumentu z aplikace, včetně odstranění z datového úložiště.
- *bool EditDocument(TosDocument document, string newName, string newCategory, string newAuthor, List<string> keywords)* – upravení parametrů dokumentu (včetně validování jejich správnosti) a následné upravení příslušného datového XML souboru.



Obrázek 10: pohled seznam dokumentů v aplikaci Dokumenty

Dokument je po načtení v programu reprezentován třídou *TosDocument*. V této třídě jsou všechna metadata k dokumentu (jméno dokumentu, reference na objekt uživatele, který dokument přidal, datum přidání, aj.) Náhled dokumentu je reprezentován třídou *TosDocumentThumbmail*, na instanci této třídy má referenci odpovídající instance dokumentu *TosDocument*. Po načtení dokumentu (inicializování třídy *TosDocument*) se provede kontrola, jestli má dokument vygenerovaný obrazový náhled v datovém úložišti aplikace. Pokud ne, tak se provede vygenerování náhledu z PDF dokumentu (pomocí použité knihovny na zpracování PDF souborů), výsledný obrázek se uloží do PNG souboru, který je pojmenovaný stejně, jako samotný dokument. *TosDocumentThumbmail* má k tomuto souboru uloženou cestu.

Skupina dokumentů je v programu reprezentována třídou *TosDocumentGroup*. Tato třída v sobě udržuje kolekci všech dokumentů, které do skupiny patří. Obsahuje také kolekce *viewGroups* a *editGroups*, které v sobě obsahují reference na skupiny uživatelů, objekty *TosUserGroup*, které mají na tuto *TosDocumentGroup* oprávnění. Pokud je nějaká *TosUserGroup* obsažená v kolekci *viewGroups*, pak mají všichni uživatelé patřící do této skupiny právo na zobrazení všech dokumentů, které do dané *TosDocumentGroup* patří.

Pokud je nějaká *TosUserGroup* obsažená v kolekci *editGroups*, pak mají její uživatelé právo na upravování všech dokumentů této skupiny. Třída *TosDocumentDatabase* v sobě udržuje kolekci všech *TosDocumentGroup*, které se v aplikaci načetly a v momentě, kdy se má vykreslit seznam dokumentů v uživatelském rozhraní, tak se kontroluje, jestli daná instance *TosDocumentGroup* obsahuje v kolekci *viewGroups* (resp. *editGroups*) referenci na skupinu *TosUserGroup* aktuálně přihlášeného uživatele a pokud ano, pak bude mít uživatel právo zobrazit (resp. upravit) všechny dokumenty této skupiny. Aby měl uživatel právo upravit dokument, musí mít i právo dokument zobrazit.

5.2.3 Zobrazení dokumentu

Realizace zobrazení obsahu jednoho dokumentu je řešena ve třídě *FoxitRenderer*, která je, podobně jako třída *ViewDocuments*, odvozená od WPF třídy *Page*. Rozložení ovládacích prvků v této třídě je řešeno pomocí komponenty *Grid*, která rozděluje stránku na dva řádky. První řádek zabírá 60px na výšku, druhý řádek pak zabírá zbytek výšky stránky. V prvním řádku se nachází *StackPanel*, který své prvky řádí horizontálně vedle sebe. Uvnitř něj jsou tlačítka, která usnadňují manipulaci se zobrazeným dokumentem:

- tlačítko na přechod zpět do seznamu dokumentů. Po kliknutí se uvolní z paměti načtený dokument a dojde k přechodu zpět na seznam dokumentů.
- Tlačítka, která umožní rychlý přechod na předchozí a následující stránku dokumentu.
- Informace o číslu stránky, na které se uživatel v dokumentu aktuálně nachází a celkový počet stránek v dokumentu. To je řešené editovatelnou komponentou *TextBox*, uživateli je umožněno aktuální stránku přepsat na číslo požadované stránky, čímž dojde k přechodu na tuto stránku.
- Tlačítka pro ovládání přiblížení: „Přiblížit“ (zvýší aktuální zvětšení dokumentu o 20%), „Oddálit“ (sníží zvětšení o 20%), „Optimální přiblížení“ (upraví přiblížení tak, aby dokument maximálně vyplňoval šířku okna aplikace), „Zobrazit celou stránku“ (upraví přiblížení tak, aby se celý dokument vešel všemi svými rozměry do aplikačního okna – bude na obrazovce celý viditelný).

Jednotlivé ovládací prvky mají implementované eventy *OnClick*, které po stisknutí tlačítka zajistí zavolání odpovídajících metod v PDF knihovně, která je použitá pro vykreslení dokumentu.

V druhém řádku hlavní mřížky třídy *FoxitRenderer* se nachází obsah samotného dokumentu. Pro renderování dokumentu jsou v aplikaci připraveny dva funkční způsoby, každý využívá jinou knihovnu – první způsob využívá knihovnu Foxit PDF SDK [26], druhý využívá knihovnu Foxit PDF Viewer for .NET SDK [29]. Defaultně se aktuálně používá pouze druhý způsob (zdůvodněné to je v kapitole 4) a je proto v této kapitole detailně popsán, zmíněny jsou ale i základní principy prvního způsobu.

Zobrazení dokumentu pomocí knihovny Foxit PDF Viewer for .NET SDK:

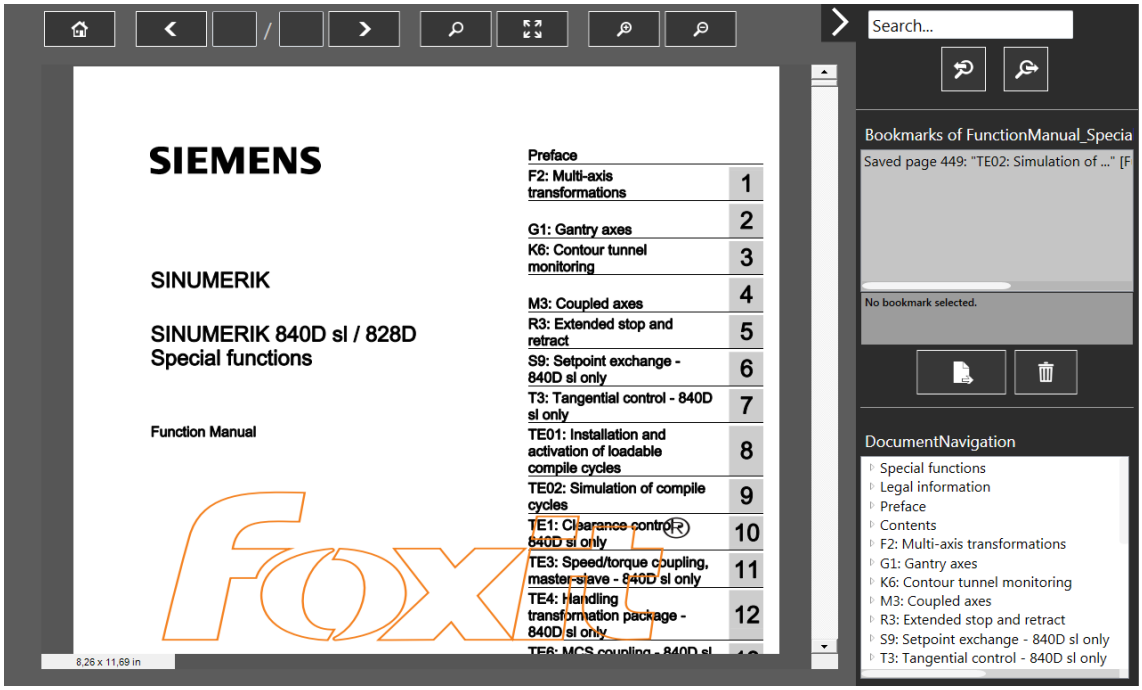
- PDF dokument je reprezentován třídou *PdfDocument* (Foxit.PDF.Viewer), kterému se cesta k PDF souboru předá jako parametr v konstruktoru. Celé to pak funguje na principu komponenty *PdfViewer* (Foxit.PDF.Viewer), která dědí od WinForms třídy *Form*. Její instanci se předá informace o PDF dokumentu, který se má zobrazit,

zavoláním metody *PdfViewer.Load(PdfDocument documentToLoad)*. Tím dojde k vykreslení obsahu dokumentu v této komponentě. Jelikož jsou aplikace systému TOSControl vytvořeny pomocí WPF, bylo nutné komponentu *PdfViewer* zobrazit pomocí WPF komponenty *WindowsFormsHost*. Ta má parametr *Child*, kterému se jako hodnota předá instance *PdfViewer* a dojde tak k vykreslení i uvnitř WPF aplikačního okna. *PdfViewer* také obsahuje vlastní *scrollbar* pro vertikální i horizontální posun v dokumentu.

- *PdfViewer* defaultně obsahuje i nástrojovou lištu, která obsahuje rychlý přístup k akcím, jako je otevření nového PDF dokumentu, tisk dokumentu, přechod na další či předchozí stránku, úprava chování kurzoru (dva režimy: výběr textu a posun stránky pomocí kurzoru), otočení dokumentu, přiblížení, aj. Tato lišta je ale v aplikaci Dokumenty nahrazena vlastním nezávislým panelem, který byl popsán na začátku této kapitoly a je vzhledově upravený (původní panel by šel z důvodu integrace do WinForms komponenty upravit velmi složitě), přizpůsobuje velikost tlačítek pro dotykové ovládání a obsahuje pouze ta tlačítka, která jsou pro aplikaci důležitá. Neobsahuje například otočení dokumentu, tisk dokumentu nebo změnu chování kurzoru myši. Tlačítka tohoto panelu volají stejné metody, které jsou volané odpovídajícími tlačítky nepoužité nástrojové lišty komponenty *PdfViewer*.
- *PdfViewer* obsahuje parametry, které umožní upravit vzhled a chování této komponenty. V aplikaci se přepisují zejména hodnoty parametrů *ShowToolbar* (zakazuje nástrojovou lištu), *CursorMode* (zajistí, že se uživatel bude schopný pohybovat v dokumentu pomocí tahů prstem), *PageColor* (úprava barvy pozadí za dokumentem). *PdfViewer* zobrazuje po kliknutí pravého tlačítka uvnitř komponenty vlastní kontextové menu, které obsahuje podobné funkce, jako jeho nástrojová lišta, včetně některých, které jsou v aplikaci Dokumenty nepotřebné, jako například „Tisk dokumentu“. Pro jejich odstranění je nutné při spuštění aplikace Dokumenty aplikovat následující postup: *foreach* cyklem projít všechny prvky kontextového menu (objekty typu *ToolStripItem*), podle jejich jména identifikovat prvky, které je třeba smazat (např. „Print“) a ty pak z této kolekce odstranit. Do této kolekce se poté ještě přidá nová instance *ToolStripItem* reprezentující funkci „Add bookmark“, tato funkce na aktuální stránku otevřeného dokumentu umístí pro aktuálně přihlášeného uživatele záložku.
- Jelikož aplikaci půjde ovládat pouze dotyky prstem, bylo nutné implementovat vlastní způsob aktivování kontextového menu. Využily se pro to eventy *OnMouseDown* a *OnMouseUp*: v implementaci eventu *OnMouseDown* dojde k uložení aktuálního času a souřadnic myši (resp. prstu) na obrazovce a uvnitř implementace *OnMouseUp* dojde k porovnání předtím uložených souřadnic a času. Pokud jsou souřadnice od sebe vzdálené méně než 35 pixelů a zároveň je rozdíl obou časů větší, než 500ms, pak dojde k otevření kontextového menu. V praxi to znamená, že uživatel musí podržet prst na stejném místě po dobu delší, než 0,5s.
- Aby se v horním ovládacím panelu zobrazovalo správně číslo aktuálně zobrazené stránky, bylo nutné vytvořit vlastní implementaci eventu *ValueChanged* pro objekt *VScrollBar*, který komponenta *PdfViewer* používá. Během tohoto eventu dojde k získání čísla aktuálně zobrazené stránky dokumentu a tato hodnota se přepíše i do horního ovládacího panelu.

Zobrazení dokumentu pomocí knihovny Foxit PDF SDK:

- V této knihovně je PDF dokument reprezentován třídou *Document* (*Foxit.PDF.Document)*. Načtení dokumentu se provádí vytvořením instance této třídy, u které se pak zavolá metoda *LoadFromFilePath(string)*, které se jako parametr předá cesta k PDF souboru. Stránky dokumentu jsou pomocí funkcí této knihovny převedeny na bitmapy (*Page Document.LoadPage(int pageNumber)*, *Page.RenderPage(PixelSource, Matrix, uint, Pause)*), které se v aplikaci vykreslí do komponenty, které se dá možnost „scrollování“, vertikálně pod sebe a tím dojde k vykreslení obsahu dokumentu v aplikaci. V případě změny aktuálního zvětšení dokumentu v aplikaci uživatelem je třeba stránky dokumentu znovu převést na nové bitmapy.



Obrázek 11: zobrazení dokumentu pomocí Foxit PDF Viewer for .NET SDK

Součástí aplikace Dokumenty je fulltextové vyhledávání v otevřeném dokumentu. Způsob implementace opět závisí na využité knihovně.

Pomocí knihovny Foxit PDF Viewer for .NET SDK:

- Realizace vyhledávání je v případě použití této knihovny poměrně snadná. Její komponenta *PdfViewer* totiž funkci vyhledávání v sobě obsahuje. Používají se pro to instanční metody *PdfViewer.SearchForward(string textToSearch, MatchOptions)* (resp. *PdfViewer.SearchBackward(string, MatchOptions)*). Tato komponenta se po zavolání těchto metod sama postará o vyhledání následujícího (resp. předchozího) výrazu v dokumentu a nastavení zobrazení tak, aby byl výraz v dokumentu umístěn na středu obrazovky. Parametr *MatchOptions* (enum) pak umožní definovat, jakým způsobem se budou hledat výsledky: hodnota *CaseSensitive* bere v potaz velká a malá písmena, *WholeWordOnly* – pro nalezení je nutné, aby se shodovalo celé slovo, ne pouze část; hodnota *All* na velikost písmen nehledí.

V aplikaci se používá vždy režim hodnoty *All*, neboť se zatím neidentifikovala nutnost dávat uživateli možnost tento režim změnit.

- Nevýhodou řešení pomocí této knihovny je to, že vyhledávací algoritmus nelze změnit. Je nastavený tak, aby zvýraznil vždy pouze jeden výsledek vyhledávání. Ideální scénář implementace, tedy spustit vyhledávání na pozadí, co nejdříve zobrazit první nalezený výsledek a poté postupně podsvěcovat další nalezené výsledky, tak možný pomocí této knihovny není. Dále také, pokud vyhledávání trvá dlouho, zobrazí se dialogové okno informující o stavu probíhajícího vyhledávání, vzhled tohoto okna ale není možné přizpůsobit. Z otevřeného dokumentu není možné pomocí komponenty *PdfViewer* extrahovat text, pokud by se tedy do budoucna naplánovala např. funkcionalita fulltextového vyhledávání nad celým seznamem dokumentů, muselo by se vymyslet alternativní řešení, knihovna Foxit PDF Viewer by v tomto případě využitelná nebyla.

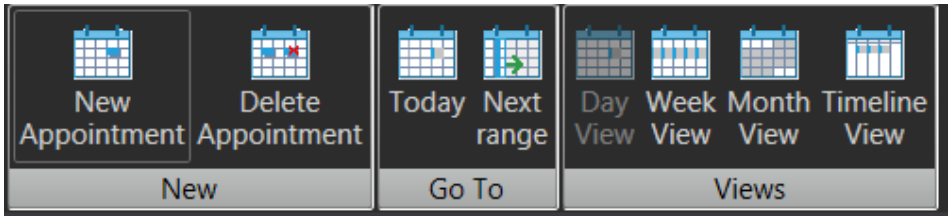
Pomocí knihovny Foxit PDF SDK:

- Výhodou knihovny Foxit PDF SDK je možnost extrakce velkého množství různých informací z dokumentu, včetně textu. Tím je umožněno implementovat vlastní způsob prohledávání dokumentu, který využívá podpůrnou třídu *TextSearch*, která se vytvoří zavoláním *Page.StartSearch(string pattern, uint searchFlags, int startIndex)*. *TextSearch* obsahuje metody *FindNext()* a *FindPrev()*, které vracejí další výsledky vyhledávání v dokumentu pro daný výraz. Proces vyhledávání pomocí této knihovny lze přesunout do vlákna na pozadí, čímž se zajistí lepší responzivnost aplikace uživateli v případě dlouhého vyhledávání v rozsáhlém dokumentu.

5.3 Aplikace Kalendář

Hlavní okno této aplikace je navržené ve třídě *CalendarMainWindow (*TOSControl_02.MyApps.CalendarApp), která je odvozená od WPF třídy *Window*. Obsah je strukturován pomocí WPF komponenty *Grid*, která aplikační okno rozděluje na dva řádky: první v sobě ukrývá navigační lištu, druhý slouží k samotnému zobrazení obsahu aplikace Kalendář. Druhý řádek je pomocí další komponenty *Grid* rozdělen na dva sloupce, které ho rozdělují v poměru 1:3. První sloupec slouží jako pomocný boční ovládací panel, pomocí kterého lze rychle vybrat požadované datum. V druhém sloupci jsou pak vykresleny události načtené z databáze aplikace.

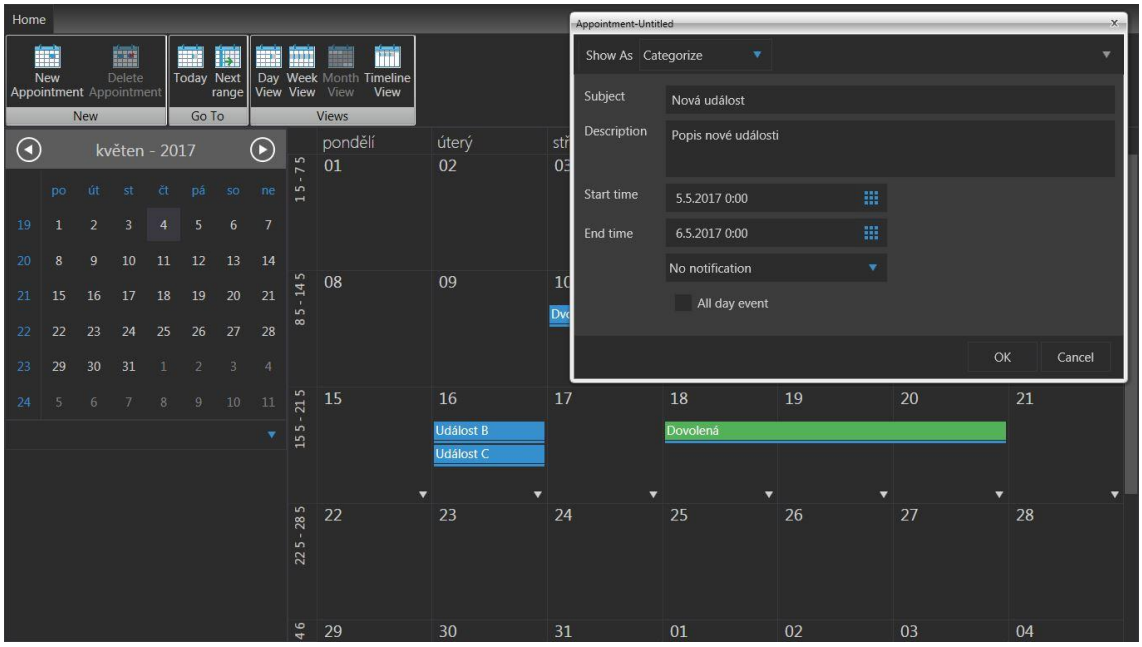
Ovládací panel je vystavěn ve stylu Ribbon [43], k jeho realizaci se využila komponenta *RadRibbonView* z knihovny Telerik UI for WPF. Obsahuje rychlý přístup pomocí tlačítek k vytváření a mazání událostí, přechod na další časové období, přechod na aktuální datum a změnu časového pohledu.



Obrázek 12: ovládací lišta ve stylu Ribbon v aplikaci Kalendář

V levém sloupci se nachází uživatelský prvek pro rychlé zvolení data sloužící k navigaci v kalendáři. Ten je realizovaný pomocí komponenty *RadCalendar*. Komponenta po kliknutí na aktuální měsíc (ve své horní části) zobrazí místo dnů jednoho měsíce všechny měsíce v roce, po opětovném kliknutí zobrazí všechny roky v desetiletí, atd. Díky tomu je navigování v kalendářním čase rychlé a uživatelsky přívětivé.

Zbylý prostor okna vyplňuje pohled na samotné události v Kalendáři. Ten je graficky realizován pomocí komponenty *RadScheduleView*, která na vstupu přijímá a vykresluje kolekci událostí, kde každá událost je reprezentována pomocí instance třídy *Appointment*. Tato třída obsahuje informace společné všem obecným typům událostí (datum, místo, jméno, popis, časové pásmo), které jsou potřebné k tomu, aby se událost mohla v mřížce kalendáře správně vykreslit. V aplikaci Kalendář se pak dále implementuje třída *TosAppointment*, která *Appointment* rozšiřuje a přidává k událostem specifické atributy pro aplikaci Dokumenty, zejména informaci o tom, jak dlouho před začátkem události musí být uživatel o události notifikován. Výběr vykreslených událostí je dán aktuálně zvoleným pohledem Kalendáře, součástí funkcionality komponenty *RadScheduleView* jsou celkem tři pohledy: denní, týdenní, měsíční.



Obrázek 13: prostředí aplikace Kalendář, včetně formuláře pro vytvoření události

Každá událost aplikace Kalendář má následující atributy: jméno, datum začátku události, datum konec události, popis události, kategorii, autora události (ID uživatele, který událost vytvořil), datum poslední notifikace (má hodnotu NULL, pokud notifikace nebyla provedena), notifikační interval (identifikuje dobu před začátkem události, do které musí být provedena notifikace uživateli). Kategorie události definuje, kteří uživatelé mají právo událost zobrazit či editovat – při načtení událostí z databáze se načtou jen ty, které má aktuálně přihlášený uživatel právo vidět. Databázová struktura pro ukládání dat je popsaná v kapitole 4.3.1. Pro vytváření nových i upravování stávajících událostí se využívá formulář, který je poskytnut jako součást komponenty *RadScheduleView*, byl pouze rozšířen o možnost editace nových atributů z třídy *TosAppointment* (zdrojové kódy komponent a stylů jsou přístupné k úpravám). Vyvolání formuláře se provede kliknutím na tlačítko „New *Appointment*“ v ovládacím panelu nebo kliknutím na existující událost. Při

potvrzení tohoto formuláře dojde v databázi dle účelu formuláře buď k uložení změn stávající události, nebo k vytvoření nové události.

Události kalendáře jsou načítané z databáze třídou *CalendarContentService*. Ta pomocí metody *GetCalendarAppointments(TosUser user)* provede načtení všech událostí, které má přihlášený uživatel právo zobrazit. Načtené události jsou reprezentovány objektem, jehož třída *Event* byla vygenerována pomocí Entity Framework a je obrazem databázové tabulky *Event*. Je nutné tyto objekty převést do objektů typu *TosAppointment*, které se můžou předat k vykreslení komponentě *RadScheduleView*. K tomu slouží třída *TosAppointmentMapper*, která umožňuje dvojicí přetížených metod *Map* oboustranné mapování tříd *Event* a *TosAppointment*. Obě třídy umožňují zapouzdření stejných informací o jedné události, avšak odpovídající si proměnné jsou v nich různě pojmenované (*TosAppointment* je odvozená třída od *Appointment*, která je implementovaná přímo v knihovně Telerik UI for WPF a většina těchto proměnných je ve třídě *Appointment*). Metody *Map* se postarají o zkopírování hodnot odpovídajících si proměnných mezi těmito třídami. *IEnumerable<TosAppointment> Map(IEnumerable<Event>)* se používá, pokud je nutné událost z uživatelského rozhraní (např. po editaci či přidání uživatelem) uložit do databáze. *IEnumerable<Event> Map(IEnumerable<TosAppointment>)* se využívá při načítání událostí z databáze a následném převodu do třídy *TosAppointment* pro vykreslení v uživatelském rozhraní.

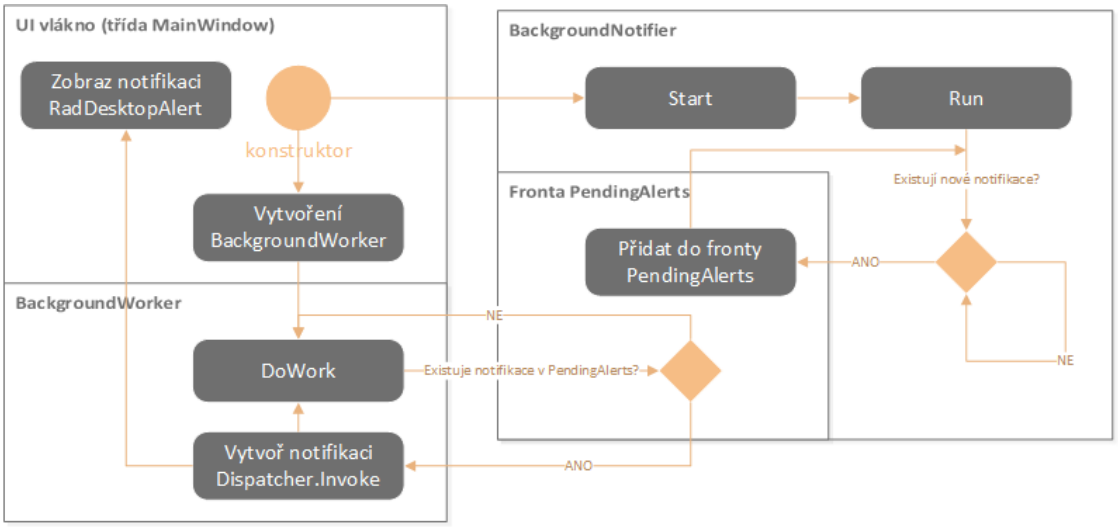
```
public static ObservableCollection<TosAppointment> GetCalendarAppointments(TosUser
user)
{
	var result = new ObservableCollection<TosAppointment>();
	var mapper = new TosAppointmentMapper();

	using (var context = new TosEntities2())
	{
		var entities = context.Events.Include(e =>
e.Category1).ToList().Where(c => c.Category1 == null || CanView(user,
c.Category1));
		IEnumerable<TosAppointment> models = mapper.Map(entities);
		result.AddRange(models);
	}

	return result;
}

public static bool CanView(TosUser user, Database.Category category)
{
	if (category.CategoryRights == null) // kategorie nema prava definovana
		return false;

	foreach (CategoryRight categoryRight in category.CategoryRights)
	{
		if (categoryRight.UserGroup == user.Group.Id)
			return categoryRight.CanView;
	}

	return false;
}
```

Zdrojový kód 8: načítání událostí z databáze pro konkrétního uživatele

5.3.1 Notifikace o nadcházejících událostech

Aplikace Kalendář musí umožňovat vytváření notifikací o nadcházejících událostech, i pokud není zrovna spuštěná. Byl tedy implementován modul, který se automaticky spustí při načtení systému TOSControl a zůstane běžet na pozadí. Jeho implementace se nachází ve třídě *BackgroundNotifier*, která obsahuje dvojici metod *Start()* a *Stop()*, ty notifikační službu spustí a zastaví. Po spuštění systému TOSControl dojde v hlavní třídě programu *MainWindow* k vytvoření instance *BackgroundNotifier* a zavolání jeho metody *Start()*. V metodě *Start()* dojde k vytvoření vlákna, které běží na pozadí aplikace a cyklicky kontroluje, jestli existuje událost, o které by měla aplikaci dát uživateli vědět formou notifikace. V rámci každého cyklu si *BackgroundNotifier* vždy stáhne nejčerstvější data událostí Kalendáře z databáze.

Grafická podoba notifikace je vyřešena pomocí komponenty *RadDesktopAlert* z knihovny Telerik UI for WPF, která zobrazí malé okénko v pravém dolním rohu obrazovky. Tato operace se musí provádět z vlákna aplikace, které provádí obsluhu UI (dále jen UI vlákno) bylo tedy nutné vymyslet způsob, jakým notifikační služba upozorní UI vlákno, aby vytvořilo notifikaci. K tomu je využitá třída *Dispatcher* [51], která poskytuje metodu *Invoke(Delegate)*, které se jako parametr předá delegát, který reprezentuje určitý spustitelný kus kódu. Ve třídě *MainWindow* je k dispozici instance objektu *Dispatcher*, která delegované akce spouští na aktuálním UI vlákně. Bylo tedy nutné vymyslet způsob, jakým instance *BackgroundNotifier* předá objektu *Dispatcher* ve třídě *MainWindow* informaci o tom, že je potřeba vytvořit notifikaci. Modul *BackgroundNotifier* by měl být (z hlediska architektonického návrhu) na *MainWindow* nezávislý a neměl by si na ni (resp. na její instanci *Dispatcher*) držet žádnou referenci. Ve třídě *BackgroundNotifier* se tak vytvořila veřejná fronta typu *ConcurrentQueue<PendingAlert>*, do které se přidávají všechny požadavky na nové notifikace (*PendingAlert* je jednoduchá třída, která obsahuje potřebné informace pro vznik notifikace, zejména jméno a text). V *MainWindow* je pak implementovaný proces, který periodicky kontroluje, jestli v této frontě existují nové požadavky na vytvoření notifikace a pokud ano, vytvoří na ně prostřednictvím UI vlákna notifikaci. Aby se touto kontrolou nemuselo zatěžovat UI vlákno, využívá se *BackgroundWorker* [52], který práci deleguje na určité vlákno na pozadí aplikace a volá metodu *Dispatcher.Invoke* pro vytvoření notifikace na UI vlákně aplikace.



Obrázek 14: diagram zobrazující princip procesu notifikací událostí Kalendáře

```
public void Start() // spustí hlavní proces BackgroundNotifier
{
	isRunning = true;
	// spustí metodu Loop na pozadí ve vlákně ThreadPoolu
	Task.Factory.StartNew(() => Loop(this), TaskCreationOptions.LongRunning);
}

private void Loop(BackgroundNotifier notifier)
{
	while (notifier.isRunning)
	{
		if (notifier.events != null)
		{
			// projít události, vytvořit požadavky na notifikace
			// umístí do fronty PendingAlerts
			NotifyAppointments();
		}

		const int interval = 30;
		// stáhnout nová data z databáze každých 30 sekund
		if (DateTime.Now.AddSeconds(-interval) > lastDataUpdate)
		{
			lastDataUpdate = DateTime.Now;

			// stáhnout nová data z databáze
			FetchData();
		}

		Thread.Sleep(2000);
	}
}

public void Stop()
{
	this.isRunning = false;
}

private void FetchData()
{
	lock (this)
	{
		this.events = ContentService.GetCalendarUnnotifiedAppointments();
	}
}
```

Zdrojový kód 9: základní logika notifikační služby BackgroundNotifier

5.4 Technologická kalkulačka - zobrazení technologického procesu

Ve jmenném prostoru TOScontrol_02.MyApps.IPcamApp se nachází hlavní třídy této aplikace. Tato aplikace je v projektu rozdělena na dvě části. První je samostatný aplikační modul, který se spustí kliknutím na příslušnou ikonu v přehledu aplikací Hlavní obrazovky. Tato část je popsaná v kapitole 5.4.1. Druhá je malé okénko, které je spustitelné tlačítkem z bočního panelu a poskytuje rychlý pohled na živý obraz IP kamery. Tato část je popsaná v kapitole 5.4.2. Každá část je řešená jiným způsobem.

5.4.1 Samostatný aplikační modul IP kamery

Řešení je realizované ve třídě *IPcamWindow*. Na stroj byl nainstalován model kamery FS-WD652-6105. Tato kamera nabízí na adrese http://192.168.214.188/ webové rozhraní, ve kterém se zobrazuje obraz kamery, dá se kamerou otáčet a upravovat parametry video přenosu (jas, kontrast, aj.). Tento aplikační modul funguje v podstatě jako webový prohlížeč, který je vložený do samostatného aplikačního okna TOSControl, který se na tuto adresu připojí a tím umožní práci s IP kamerou. Webové rozhraní ale vyžaduje, aby se uživatel nejprve přihlásil, bylo tedy nutné vymyslet řešení, které přihlášení zautomatizuje.

Ve třídě *IPcamWindow* je WPF komponenta *WebBrowser*. Po otevření hlavního okna této aplikace z úvodní obrazovky dojde k zavolání metody *WebBrowser.Navigate(new Uri(„http://192.168.214.188/login.html“)*, která načte webovou stránku, na které se nachází přihlašovací formulář webového rozhraní. Po dokončení načítání se vyvolá event *LoadCompleted*, jehož obslužná metoda ve třídě *IPcamWindow* provede akce potřebné k přihlášení: dohledání a vyplnění *TextBoxů* „username“ a „password“ přihlašovacího formuláře, po krátkém čekání pak nasimuluje stisk přihlašovacího tlačítka.

Aby uživatel tento proces po spuštění aplikace na obrazovce neviděl, je vytvořeno druhé okno *IPcamInitialize*. To se stará o zobrazení hlášky „IP camera is loading…“, která překryje celou obrazovku a je viditelná, dokud obrazovka *IpcamWindow* nedokončí přihlášení a načtení webového rozhraní. Poté je okno *IPcamInitialize* skryto a uživatel může začít pracovat s webovým rozhraním aplikace.

Obrázek 15: vykreslení obrazu IP kamery v samostatném aplikačním modulu

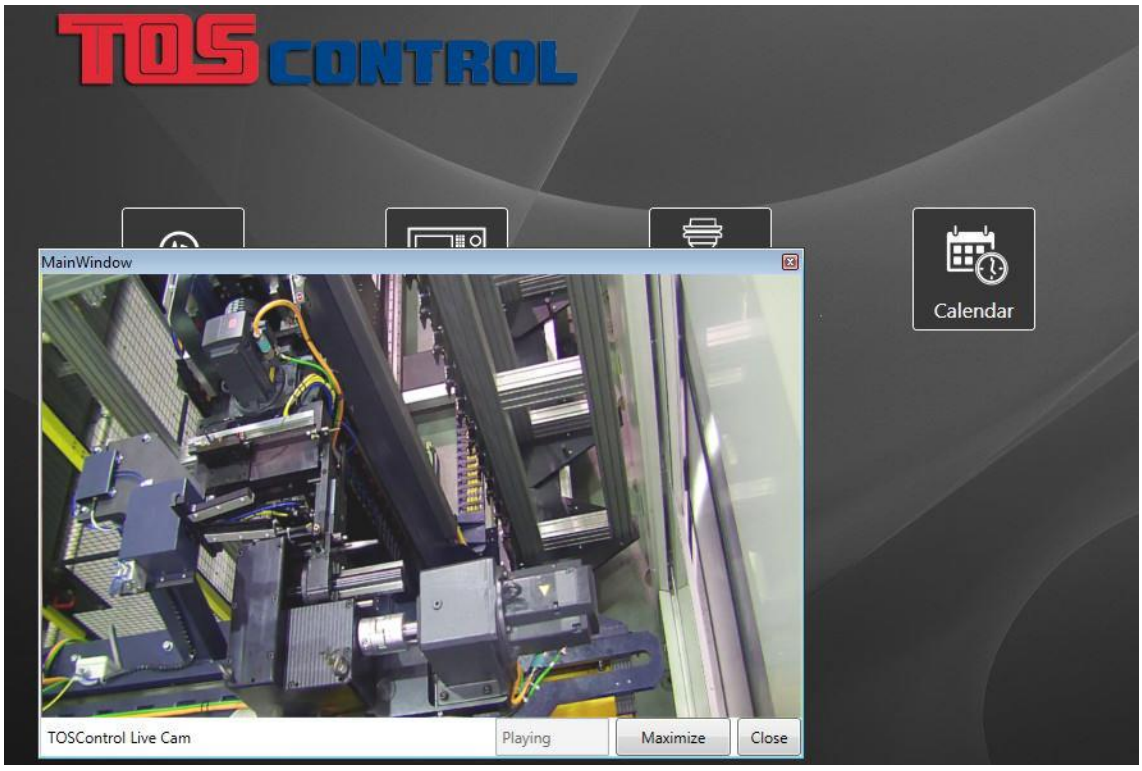
```
public void webOutput_LoadCompleted(object sender, NavigationEventArgs e)
{
    document = (HTMLDocument)webOutput.Document;

    // vyplnění uživatelského jména a hesla, pokud je adresa login.html
    if (document.url.ToString() == "http://192.168.214.188/login.html")
    {
        // zobrazit inicializační okno ("Camera is loading...")
        ipcamInitializeWindow = new IPcamInitialize();
        ipcamInitializeWindow.Owner = this;

        // nastavit toto okno aktivní a skrýt aktuální okno
        ipcamInitializeWindow.Show();
        this.Hide();

        // automatické prihlášeni - vyplnit uživ. údaje
        this.FillLoginDetails();

        // chvíli počkat, poté potvrdit (simulace zmáčknutí login tlačítka)
        await this.ConfirmLoginAsync();
    }
}

public Task ConfirmLoginAsync()
{
    // spustit akci asynchronně na pozadí, aby neblokovalo UI vlákno
    return Task.Run(() =>
    {
        // počkat 800ms
        Thread.Sleep(800);

        IHTMLElementCollection elementCollection = document.all;
        // najit element tlačítka
        foreach (IHTMLElement element1 in elementCollection)
        {
            if (element1.tagName=="BUTTON" &&
element1.className.Contains("login"))
            {
                element1.click();
            }
        }

        // zavolat na UI vlákně - skrýt inicializační okno, zobrazit hlavní okno
        this.Dispatcher.Invoke((Action)(() =>
        {
            ipcamWindow.Show();
            ipcamInitializeWindow.Close();
        }));
    });
}

public void FillLoginDetails()
{
    document.getElementById("username").setAttribute("value", "admin");
    document.getElementById("password"). setAttribute("value", "admin");
}
```

Zdrojový kód 10: kód pro zobrazení webového rozhraní IP kamery v TOSControl

5.4.2 Přenosné okno náhledu na obraz IP kamery

Účelem tohoto okna je pouze zobrazit obraz z IP kamery, není třeba využívat žádné ovládací prvky kamery. Pokud bude uživatel chtít kameru natočit, přepne se do samostatné aplikace, která toto ovládání umožňuje (viz kapitola 5.4.1). Pro realizaci tohoto okna je možné využít toho, že video stream z kamery je přístupný online pomocí protokolu RTSP na adrese *rtsp://192.168.214.188/channel1/1*. Je tedy nutné vymyslet, jak v aplikaci data z tohoto protokolu přijmout a zobrazit.

Pro realizaci se využila volně dostupná knihovna Stream Player control [53], která je distribuovaná pod licencí CPOL [54]. Tato knihovna poskytuje WPF a WinForms komponenty, které umožňují přehrávat video pomocí RTSP a RTMP protokol. Tuto knihovnu bylo nutné zkompilovat pomocí .NET Framework verze 4, pokud kompilace proběhla ve stejné verzi .NET Framework, ve které je vystavěn zbytek projektu TOSControl (4.5), tak byla knihovna nefunkční. Bylo tedy rozhodnuto, že se tento modul vytvoří a přemístí do separátního projektu, ve kterém se zkompiluje do spustitelné WPF aplikace, která bude mít své aplikační okno. V samotném TOSControl se k ní pak bude přistupovat podobně jako ke spustitelnému procesu – získá se cesta k EXE souboru, ten se spustí, následně se bude udržovat reference na nově vzniklou instanci třídy *Process* [55], která proces náhledového okna kamery reprezentuje. Pokud bude TOSControl muset okno kamery zavřít, využije se k tomu metoda *Process.Close()*, případně *Process.Kill()*.



Obrázek 16: otevřené okno pro náhled na obraz IP kamery v prostředí TOSControl

6 Závěr

Všechny stanovené cíle bakalářské práce byly úspěšně splněny. Podařilo se navrhnout a implementovat čtyři dílčí aplikace systému TOSControl, které splňují stanovené požadavky zadavatele práce, čímž se položil základní kámen, na kterém se může při budoucím vývoji nových aplikačních modulů systému stavět. Aplikace jsou vystavěny pomocí moderních technologií a knihoven, které jsou v současné době u desktopových aplikací často využívané, zejména Microsoft .NET Framework 4.5, WPF a Microsoft SQL Server 2013. Aktuální popularita těchto technologií představuje určitý příslib do budoucna, zejména z toho pohledu, že se jejich vývoj udrží a budou vycházet nové aktualizace. S využitím systému TOSControl totiž společnost TOS VARNSDORF a.s. počítá dlouhodobě i do budoucna, neboť by v případě pozitivních referencí od zákazníků mohl sloužit jako primární informační systém i novým strojům, které v dílnách této společnosti v budoucnu vzniknou.

Systém TOSControl byl úspěšně nasazený na PCU jednotce prototypu vyvíjeného stroje v sídle společnosti TOS VARNSDORF a.s. a v nejbližších měsících se začne využívat interně pracovníky této společnosti, dojde tak o otestování funkčnosti v praxi reálnými uživateli. Další vývoj systému v blízké době tedy bude věnován zejména řešení případných problémů a chyb, které z tohoto testování vyplynou.

Na závěr je třeba zmínit, že v této bakalářské práci byly vytvořeny poměrně základní aplikace, které jsou však pro informační systém tohoto typu důležité. Do budoucna pak existuje velký potenciál pro další vývoj ze dvou pohledů: implementace jednotného datového úložiště informačního systému pro skupinu strojů – princip IoT „Internet of Things“ (lze využít faktu, že již současná verze TOSControl využívá k ukládání dat databázový systém, který půjde sdílet mezi více strojů) a nové aplikační moduly: například systém pro kontrolu kolizí jednotlivých částí v reálném čase stroje by mohl být v budoucnu námětem na další akademickou práci.

Seznam použitých zdrojů

[1] Software | HYUNDAI WIA Machine Tools. *HYUNDAI WIA Machine Tools* [online]. b.r. [cit. 2017-05-08]. Dostupné z: http://machine.hyundai-wia.com/en/product/software_01.asp

[2] CELOS® from DMG MORI - From the idea to the finished product. *DMG MORI* [online]. b.r. [cit. 2017-05-08]. Dostupné z: http://en.dmgmori.com/products/celos

[3] OSP-P300. *Okume Europe GmbH* [online]. b.r. [cit. 2017-05-08]. Dostupné z: https://www.okuma.eu/en/products/cnc-control/osp-p300/

[4] Tosvarnsdorf.cz. *WHT 110 C* [online]. b.r. [cit. 2017-04-23]. Dostupné z: http://www.tosvarnsdorf.cz/cz/produkty/horizontalni-obrabeci-centra/wht-110-c/

[5] *Product details: HD IP High Speed Dome. SHENZHEN FSAN INTELLIGENT TECHNOLOGY CO.,LTD.* [online]. b.r. [cit. 2017-04-25]. Dostupné z: http://www.fsan.cn/goods/show-1379.html

[6] LINQ: .NET Language Integrated Query. *MSDN* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://msdn.microsoft.com/en-us/library/bb308959.aspx

[7] IEnumerable(T) Interface (System.Collections.Generic). *MSDN* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://msdn.microsoft.com/en-us/library/9eekhta0(v=vs.110).aspx

[8] *Paralelní LINQ (PLINQ). MSDN* [online]. b.r. [cit. 2017-04-28]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/dd460688(v=vs.110).aspx

[9] Asynchronous Programming with Async and Await (C# and Visual Basic). *MSDN* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://msdn.microsoft.com/library/hh191443(vs.110).aspx

[10] *NuGet* [online]. b.r. [cit. 2017-04-25]. Dostupné z: https://www.nuget.org/

[11] *C# - Stack Overflow - Windows GUI: WPF or WinRT (2015+)* [online]. b.r. [cit. 2017-04-25]. Dostupné z: http://stackoverflow.com/questions/28586642/windows-gui-wpf-or-winrt-2015

[12] *WPF vs. WinForms. WPF tutorial* [online]. b.r. [cit. 2017-04-25]. Dostupné z: http://www.wpf-tutorial.com/about-wpf/wpf-vs-winforms/

[13] *WindowsFormsHost Class. MSDN* [online]. b.r. [cit. 2017-04-25]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.windows.forms.integration.windowsformshost(v=vs.110).aspx

[14] Windows Presentation Foundation. *MSDN* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx

[15] Přehled XAML (WPF). *MSDN* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/ms752059(v=vs.110).aspx

[16] Style and template overview. *MSDN* [online]. b.r. [cit. 2017-05-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/cc295273.aspx

[17] *Telerik WPF Controls* [online]. b.r. [cit. 2017-04-25]. Dostupné z: http://www.telerik.com/products/wpf/overview.aspx

[18] *Devexpress WPF Controls* [online]. b.r. [cit. 2017-04-25]. Dostupné z: https://www.devexpress.com/products/net/controls/wpf/

[19] *Syncfusion Essential Studio for WPF* [online]. b.r. [cit. 2017-04-25]. Dostupné z: https://www.syncfusion.com/products/wpf

[20] *Introduction | UI for WPF Documentation by Progress* [online]. b.r. [cit. 2017-04-28]. Dostupné z: http://docs.telerik.com/devtools/wpf/introduction

[21] What Is JavaFX? | JavaFX 2 Tutorials and Documentation. *Java Documentation* [online]. b.r. [cit. 2017-05-09]. Dostupné z: http://docs.oracle.com/javafx/2/overview/jfxpub-overview.htm

[22] DB-Engines Ranking - popularity ranking of database management systems. *DB-Engines* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://db-engines.com/en/ranking

[23] Entity Framework. *MSDN* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://msdn.microsoft.com/en-us/library/gg696172(v=vs.103).aspx

[24] Portable Document Format. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001 [cit. 2017-04-26]. Dostupné z: https://en.wikipedia.org/wiki/Portable_Document_Format

[25] *PDF Reference (sixth edition). Adobe.com* [online]. b.r. [cit. 2017-04-26]. Dostupné z: http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/pdf_reference_1-7.pdf

[26] *PDF SDK | Foxit SDK | PDF Technology* [online]. b.r. [cit. 2017-04-27]. Dostupné z: http://www.foxitsdk.com/products/general-pdf-sdk/

[27] *Adobe PDF Library SDK | Adobe Developer Connection* [online]. b.r. [cit. 2017-04-28]. Dostupné z: http://www.adobe.com/devnet/pdf/library.html

[28] *Quick PDF Library | Powerful PDF SDK* [online]. b.r. [cit. 2017-04-28]. Dostupné z: http://www.debenu.com/products/development/debenu-pdf-library/

[29] *PDF Viewer for .NET SDK* [online]. b.r. [cit. 2017-04-27]. Dostupné z: https://www.foxitsoftware.com/de/products/sdk/viewer/

[30] *Pvginkel/PdfiumViewer: PDF viewer based on Google's PDFium* [online]. b.r. [cit. 2017-04-28]. Dostupné z: https://github.com/pvginkel/PdfiumViewer

[31] *Foxit PDF Technology Chosen for Google Open-Source* [online]. b.r. [cit. 2017-04-28]. Dostupné z: https://www.foxitsoftware.com/blog/foxit-pdf-technology-chosen-for-google-open-source/

[32] *Window class. MSDN* [online]. b.r. [cit. 2017-04-26]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.windows.window(v=vs.110).aspx

[33] *Grid Class (System.Windows.Controls). MSDN* [online]. b.r. [cit. 2017-04-28]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.windows.controls.grid(v=vs.110).aspx

[34] *Třída TextBox (System.Windows.Controls). MSDN* [online]. b.r. [cit. 2017-04-28]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.windows.controls.textbox(v=vs.110).aspx

[35] ComboBox Class (System.Windows.Controls). *MSDN* [online]. b.r. [cit. 2017-05-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.windows.controls.combobox(v=vs.110).aspx

[36] *Button Class (System.Windows.Controls). MSDN* [online]. b.r. [cit. 2017-04-28]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.windows.controls.button(v=vs.110).aspx

[37] *Page class. MSDN* [online]. b.r. [cit. 2017-04-26]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.windows.controls.page(v=vs.110).aspx

[38] *Frame Class (System.Windows.Controls). MSDN* [online]. b.r. [cit. 2017-04-28]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.windows.controls.frame(v=vs.110).aspx

[39] RadComboBox. *Overview | UI for WPF Documentation by Progress* [online]. b.r. [cit. 2017-05-08]. Dostupné z: http://docs.telerik.com/devtools/wpf/controls/radcombobox/overview

[40] RadMaskedInput. *Overview | UI for WPF Documentation by Progress* [online]. b.r. [cit. 2017-05-08]. Dostupné z: http://docs.telerik.com/devtools/wpf/controls/radmaskedinput/overview

[41] RadPasswordBox. *Overview | UI for WPF Documentation by Progress* [online]. b.r. [cit. 2017-05-08]. Dostupné z: http://docs.telerik.com/devtools/wpf/controls/radpasswordbox/overview

[42] RadRibbonView. *Overview | UI for WPF Documentation by Progress* [online]. b.r. [cit. 2017-05-08]. Dostupné z: http://docs.telerik.com/devtools/wpf/controls/radribbonview/overview

[43] Ribbons (Windows). *MSDN* [online]. b.r. [cit. 2017-05-08]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/dn742393(v=vs.85).aspx

[44] RadCalendar. *Overview | UI for WPF Documentation by Progress* [online]. b.r. [cit. 2017-05-08]. Dostupné z: http://docs.telerik.com/devtools/wpf/controls/radcalendar/overview

[45] RadScheduleView. *Overview | UI for WPF Documentation by Progress* [online]. b.r. [cit. 2017-05-08]. Dostupné z: http://docs.telerik.com/devtools/wpf/controls/radscheduleview/overview

[46] Singleton. *Algoritmy.net* [online]. b.r. [cit. 2017-05-08]. Dostupné z: https://www.algoritmy.net/article/1326/Singleton

[47] Modern UI Icons. *Modern UI Icons* [online]. b.r. [cit. 2017-05-08]. Dostupné z: http://modernuiicons.com/

[48] *Entity Framework Database First* [online]. b.r. [cit. 2017-05-03]. Dostupné z: https://msdn.microsoft.com/en-us/library/jj206878(v=vs.113).aspx

[49] DbContext Class (System.Data.Entity). *MSDN* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.data.entity.dbcontext(v=vs.113).aspx

[50] DbSet(TEntity) Class (System.Data.Entity). *MSDN* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://msdn.microsoft.com/en-us/library/gg696460(v=vs.113).aspx

[51] Třída Dispatcher. *MSDN* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/system.windows.threading.dispatcher(v=vs.110).aspx

[52] BackgroundWorker Component Overview. *MSDN* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://msdn.microsoft.com/en-us/library/8xs8549b(v=vs.110).aspx

[53] *Stream Player control - CodeProject* [online]. b.r. [cit. 2017-04-30]. Dostupné z: https://www.codeproject.com/Articles/885869/Stream-Player-control

[54] *CPOL: Code Project Open License - CodeProject* [online]. b.r. [cit. 2017-04-30]. Dostupné z: https://www.codeproject.com/info/cpol10.aspx

[55] Process Class. *MSDN* [online]. b.r. [cit. 2017-05-09]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.diagnostics.process(v=vs.80).aspx

Použitá literatura

[1] BLEWETT, Richard a Andrew CLYMER. *Pro asynchronous programming with .NET*. 1st edition. Apress, 2013. Expert's voice in .NET. ISBN 9781430259206.

[2] AGARWAL, Vidya Vrat a James HUDDLESTON. Databáze v C# 2008: průvodce programátora. Brno: Computer Press, 2009. ISBN 9788025123096.