UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Korporátní publikační systém se zaměřením na správu, výrobu a distribuci reklamních tiskovin

Bc. Vojtěch Pešl

Diplomová práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Vojtěch Pešl**
Osobní číslo:	**I11403**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Korporátní publikační systém se zaměřením na správu, výrobu a distribuci reklamních tiskovin**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

V úvodní části práce je nutné provést analýzu navrhovaného řešení, která bude obsahovat mimo základních prvků (sběr požadavků, případy a scénáře užití, ...) také popis použitých technologií a návrh aplikačního řešení. Analýza bude obsahovat diagramy využívající UML 2.0.
Součástí práce bude rešerše stávajících SW řešení vhodných pro správu a přípravu reklamních tiskovin. Rešerši je nutné doplnit o porovnání s výsledky analyzovaného navrhovaného systému.
Primárním cílem diplomové práce je návrh a implementace aplikace umožňující pohodlnou správu, sestavení a následnou distribuci reklamních tiskovin.
Aplikace by měla umožňovat:
- Tvorbu a správu (import, export, ...) šablon pro reklamní tiskoviny.
- Využití šablon pro návrh konkrétních reklamních tiskovin.
- Správu kontaktů - XML, CSV import a export kontaktů na klienty a spolupracující firmy. A udržování informací o jaký typ (kategorii) reklamních tiskovin má konkrétní klient zájem.
- Odeslat navržené tiskoviny v elektronické podobě (nejlépe PDF) do tiskárny a v podobě newsletteru též klientům, kteří mají o odběr novinek tohoto typu zájem.
- Aplikace by měla umožňovat publikování určitého typu reklamních tiskovin též v podobě JPEG nebo PNG obrázků na sociální síti Twitter či Facebook.

Aplikace bude napsaná v jazyku C ♯ na platformě .NET a bude dodržovat konvence návrhu aplikací pro tuto platformu.
Výsledná aplikace by měla být otestována v praktickém používání.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

[1] NAGEL CH. et al. C 2008. Programujeme profesionálně. Brno, 2009. ISBN 978-80-251-2407-7.
[2] VIRIUS, M. Od C+ k C. České Budějovice: KOPP, 2002. ISBN 80-7232-176-5.
[3] The C Language Specification 4th edition dostupné online:http://www.ecma-international.org/publications/standards/Ecma-334.htm

Vedoucí diplomové práce: **Ing. Jan Hřídel**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2012

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 19. 8. 2013 Bc. Vojtěch Pešl

Poděkování

Zde bych chtěl poděkovat Ing. Janu Hřídelovi za pomoc a cenné informace při vypracování diplomové práce.

Anotace

Tato práce se zaměřuje na problematiku návrhu a implementace aplikace umožňující správu, sestavení a následnou distribuci reklamních tiskovin. V úvodní části je vytvořena analýza stávajících publikačních systémů na přípravu tiskovin. Další část popisuje vývoj a implementaci systému včetně podrobného popisu použitých technologií.

Klíčová slova

Reklamní tiskovina, C#, .NET, analýza, návrh, implementace

Title

Corporate publishing system with a focus on management, production and distribution of printed advertising materials

Annotation

This work focuses on the design and implementation of the application for management, composition and subsequent distribution of printed advertising materials. The first part consists of analysis of existing publishing systems for the preparation of printed materials. The second part describes development and implementation of the system including detailed description of the technology.

Keywords

Printed advertising material, C#, .NET, analysis, design, implementation

Obsah

Seznam zkratek

UML	Unified Modeling Language
CASE	Computer-aided Software Engineering
UP	Unified Process
CASE	Computer Aided Software Engineering
C#	C Sharp
PDF	Portable Document Format
GNU GPL	GNU General Public License
XPS	XML Paper Specification
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
GIF	Graphics Interchange Format
BMP	Windows Bitmap
TIFF	Tagged Image File Format
HTML	HyperText Markup Language
VB	Visual Basic
GUI	Graphical User Interface
DTP	Desktop publishing
CIL	Common Intermediate Language
CLR	Common Language Runtime
SQL	Structured Query Language
SQL CE	Microsoft SQL Server Compact
GDI	Graphics Device Interface
API	Application Programming Interface
DPI	Dots per inch
MIME	Multipurpose Internet Mail Extensions
XML	Extensible Markup Language
W3C	World Wide Web Consortium
CSV	Comma-separated values
SOAP	Simple Object Access Protoco
SMTP	Simple Mail Transfer Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

Seznam obrázků

Seznam tabulek

1 Úvod

Tato práce se zabývá tvorbou informačního systému. Cílem práce bylo vytvořit návrh a implementaci aplikace, která umožňuje pohodlnou tvorbu a následnou distribuci reklamních tiskovin.

První část této práce se zabývá popisem již existujících informačních systémů sloužících pro správu a přípravu reklamních tiskovin. V závěru této části jsou zkoumané informační systémy srovnány dle jejich funkčnosti.

V následující části je popsán komplexní návrh dle metodiky UP a implementace navrženého systému. V této části je popsán sběr požadavků definovaný zadavatelem. Na základě získaných požadavků jsou modelovány případy užití, dále je provedena analýza systému a návrh struktury celého systému včetně návrhu databáze.

V kapitole, která se věnuje implementaci, je popsaná platforma .NET a použité technologie a jejich způsob implementace.

2 Publikační systém na výrobu reklamních tiskovin

Pod pojmem „publikační systém na výrobu reklamních tiskovin“ si můžeme představit systém, který nám jednak zjednodušuje přípravu tiskovin, ale také snižuje náklady na jejich výrobu. Cílem tohoto systému je zjednodušit práci s tiskovinami, které se opakují, jako například akční letáky, ceníky, technické a produktové listy. Dále by nám tento systém měl zjednodušit práci při publikování tiskovin, kdy nám umožní jednoduché zaslání na e-mail či zveřejnění na sociální síť. [1]

2.1 Analýza stávajících publikačních systémů na výrobu reklamních tiskovin

2.1.1 Servis Informačních Technologií, s.r.o.

Jedním z mála systémů na výrobu a publikování tiskovin je systém firmy „Servis Informačních Technologií, s.r.o.“. Tento systém je zaměřený pouze na generování tiskovin. Software nám umožňuje přípravu šablony, která nám zjednodušuje vkládat text a fotografie. Systém nám také umožňuje danou tiskovinu vygenerovat do PDF. Licence s tímto systémem není omezená na počet uživatelů, proto se na předtiskové přípravě může podílet i více pracovníků. [1]

2.1.2 Scribus

Scribus je program, zaměřený především pro počítačovou sazbu. Umožňuje nám rychlou tvorbu elektronických dokumentů prostřednictvím mnoha šablon, které lze i bezplatně stáhnout z webových stránek programu. Do navrhované publikace lze importovat nejenom obrázky, text, ale i vektorovou grafiku. Scribus nám neumožňuje odeslat navrhovanou publikaci e-mailem či publikovat na sociální síť, ale umožňuje export do mnoha formátů.

Program Scribus nedisponuje tolika funkcemi jako profesionální programy pro počítačovou sazbu, proto je vhodný pro tvorbu menších dokumentů, jako jsou vizitky, pozvánky, letáky či prezentace. Jeho hlavní výhodou je, že je šířen pod licencí GNU GPL, a proto je zcela zdarma.

2.1.3 Microsoft Publisher

„Microsoft Publisher je nástroj na tvorbu publikací v prostředí balíku MS Office, se kterým Publisher sdílí celou řadu funkcí včetně korektury pravopisu, formátování textů či manipulace s grafickými objekty. Na rozdíl od MS Wordu je Publisher koncipovaný přímo na sestavování různých druhů publikací, bookletů, brožur, zpravodajů a marketingových materiálů, které lze v layout editoru Publisheru rychle skládat prostřednictvím mnoha desítek offline i online dostupných šablon.“ [2]

Dokumenty Publisheru lze publikovat a rozeslat pak danou publikaci celému seznamu zákazníků ve formátech PDF, XPS či jako bitmapový náhled například ve formátech JPEG či PNG, pokud máme nainstalovaný a nastavený jako výchozí e-mailový klient Microsoft Outlook.

Publisher lze popsat jako užitečnou a celkově zjednodušenou „kancelářskou“ DTP aplikaci, která vhodnou formou nabízí většinu funkcí pro tvorbu či spíše sestavování publikací pomocí šablon a předdefinovaných objektových prvků. Publisher se nachází pouze v nejvyšší edici kancelářského balíku Microsoft Office. [2]

2.1.4 Adobe InDesign

InDesign je profesionální program pro počítačovou sazbu. V InDesignu lze vytvářet, jak vícestránkové tiskoviny, jako jsou magazíny, noviny a knihy, tak menší dokumenty, jako jsou vizitky, plakáty či letáky. Mezi samozřejmosti programu patří jednoduchý návrh pomocí šablony, export do různých formátů a rozsáhlé možnosti návrhu publikace. Program lze zakoupit samostatně nebo v rámci softwarových balíků, které obsahují kolekci grafických programů od společnosti Adobe, jež spolu vzájemně komunikují.

2.2 Shrnutí

V následující tabulce můžeme vidět srovnání analyzovaných systémů. Mezi analyzované programy jsou zahrnuty profesionální programy, které jsou primárně určeny pro počítačovou sazbu. Tyto programy slouží nejen na výrobu tiskovin, ale také elektronických publikací.

Tabulka 1 – Srovnání systému pro správu a přípravů reklamních tiskovin. Zdroj: vlastní

Název systému nebo firmy	Návrh tiskoviny	Použití šablony	Generování PDF	Odeslání tiskoviny na e-mail	Správa kontaktů	Publikování na sociální síť
Servis Informačních Technologií, s.r.o.	ANO	ANO	ANO	NE	NE	NE
Adobe InDesign	ANO	ANO	ANO	NE	NE	NE
Scribus	ANO	ANO	ANO	NE	NE	NE
Microsoft Publisher	ANO	ANO	ANO	ANO	ANO	NE

3 Vývoj a implementace systému

3.1 UML

Jazyk UML je univerzální jazyk pro vizuální modelování systému. Nejčastěji je spojován s návrhem softwarového systému, ale má i mnohem širší možnosti použití. Nabízí nám nejenom vizualizaci, navrhování, dokumentaci softwarových systémů, ale také tvorbu business procesů, systémové funkce, databázová schémata a mnoho dalšího.

Jazyk UML je navržen tak, aby spojil nejlepší postupy modelovacích technik. Je navržen takovým způsobem, aby jej mohly implementovat všechny nástroje CASE. Diagramy vytvořené v UML jazyce jsou srozumitelné pro lidi a navíc je mohou interpretovat i programy CASE. Nenabízí žádný druh metodiky modelování, poskytují pouze vizuální syntaxi, kterou můžeme využít při sestavování svých modelů.

Metodika Unified Process sděluje jaké činnosti a pracovníky musíme využít a jaké produkty musíme vyrobit. Využívá jazyk UML jako vlastní syntaxi pro vizuální modelování, protože je pro tento jazyk nejlépe adaptovatelná. [3]

3.1.1 Historie

V roce 1995 začal vznikat jazyk UML sjednocením různých metod a syntaxí pro modelování za záštitou firmy Rational. Výsledkem tohoto snažení bylo v roce 1997 vytvoření první verze modelovacího jazyka 1.1, který přijala jako standart standardizační skupina Object Management Group. Tento souhrn metod se stal průmyslovým standardem a postupně se vyvíjel až do aktuální verze 2.0.

Verze UML 2.0 přináší mnoho změn, došlo ke změnám v metamodelu. Největší změnou je uvedení nových typů diagramů. Diagramy objektů a seskupení se sice používaly, ale nebyly oficiálním typem diagramů. Dále došlo ke změnám v názvu diagramů objektové spolupráce na komunikační diagramy. Vznikly nové typy diagramů, například diagram přehledu interakce, diagram načasování a další. [3] [4]

3.1.2 Objekty UML

Základním předpokladem jazyka UML je, že umožňuje modelování systému jako kolekce spolupracujících objektů. Na první pohled se zdá, že tento jazyk slouží zejména na návrh softwarových systémů, ale funguje stejně spolehlivě v obchodních a podnikatelských procesech.

- Statická struktura – popisuje jaké typy objektů jsou pro modelování daného systému důležité a jak spolu tyto systémy souvisejí.
- Dynamické chování – popisuje životní cyklus zmiňovaných objektů a způsob jejich vzájemné spolupráce s cílem dosáhnout požadované funkce navrhovaného systému. [3]

3.1.3 Stavební bloky jazyka UML

Jazyk UML se skládá ze třech stavebních bloků:

- Předměty – představují samotné prvky modelu.
- Vztahy (relace) – určují jak spolu dva nebo více předmětů souvisejí.
- Diagramy – jsou pohledy na modely UML. Představují to, co bude systém dělat (analytické diagramy) a to, jak to bude dělat (návrhové třídy). [3]

Předměty

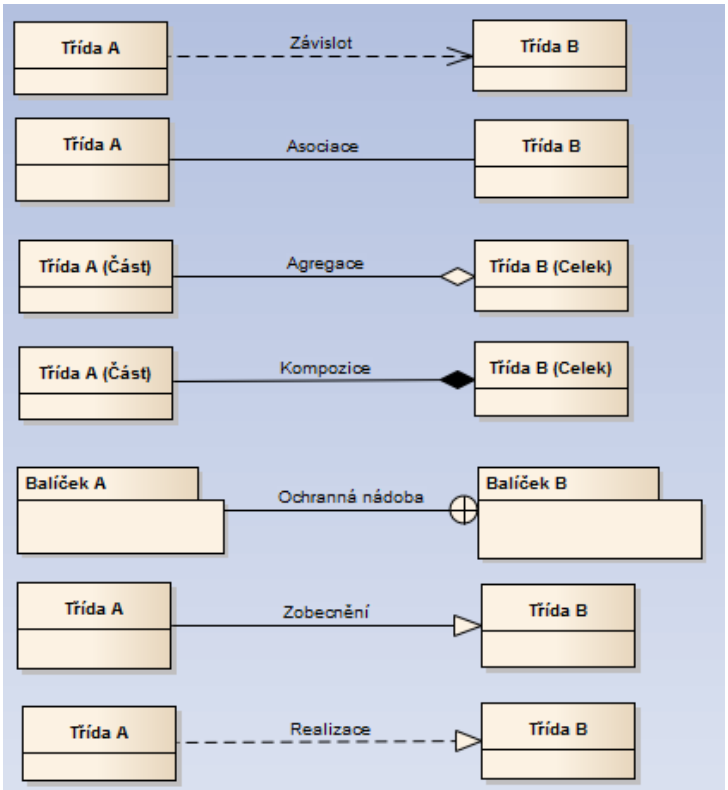
Předměty dělíme v jazyku UML:

- Strukturní abstrakce – třídy, rozhraní, spolupráce, případy užití, komponenty
- Chování – interakce, stav
- Seskupení – balíčky seskupující spolu související prvky modelu
- Poznámky – anotace, které lze k modelu připojit s úmyslem zachytit informaci sestavenou jen k tomuto modelu [3]

Relace

Relace zachycují vztah mezi dvěma předměty. V jazyce UML existuje několik druhů relací.

- Závislost – změna v jednom předmětu ovlivňuje význam v druhém předmětu.
- Asociace – volný vztah mezi dvěma nebo více třídami, představující jejich komunikaci.
- Agregace – vazba mezi celkem a součástí. Součást může fungovat bez celku.
- Kompozice – speciální typ agregace. Součást nemůže fungovat bez celku.
- Ochranná nádoba – zdrojový prvek obsahuje cílový prvek.
- Zobecnění – jeden prvek je specializací jiného prvku. Pomocí zobecnění realizujeme dědičnost.
- Realizace – vztah mezi rozhraním a třídou. [3]



Obrázek 1 – Relace. Zdroj: vlastní

Diagramy

Diagramy jsou pohledy na modely UML. Diagramy nejsou modely. Předměty a relace lze z diagramu odstranit, ale v modelu mohou dále existovat. V UML existuje třináct typů diagramů, dělí se do dvou základních skupin: diagramy struktury a diagramy chování.

- Diagramy struktury - modelují statickou strukturu systému. Zachycuje předměty a asociace mezi předměty.
  - Diagram tříd
  - Diagram složené struktury
  - Diagram komponent
  - Diagram nasazení
  - Objektový diagram
  - Diagram balíčků
- Diagramy chování - zachycují způsob, jak na sebe jednotlivé předměty navzájem působí, aby bylo dosaženo požadovaného chování.
  - Diagram aktivit
  - Diagram interakce
  - Diagram případu užití
  - Diagram stavového automatu [3]

3.1.4 Obecná mechanika jazyka UML

Jazyk UML obsahuje čtyři obecné mechanismy, které se používají v celém jazyku konzistentně. [3]

Specifikace

Specifikace jsou textovým popisem sémantiky jednotlivých prvků. [3]

Ornamenty

Pomocí ornamentů zdobíme jednotlivé prvky v diagramech UML proto, abychom zvýraznili nebo zdůraznili určité důležité detaily. [3]

Podskupiny

V jazyku UML rozlišujeme dvě podskupiny:

- Skupina klasifikátorů a instancí – klasifikátor je abstraktním vyjádřením typu předmětu, instance je naproti tomu konkrétním výskytem typu předmětu.
- Skupina rozhraní a implementací – rozhraní specifikuje chování předmětu, implementace specifikuje podrobnosti tohoto chování. [3]

Mechanismy rozšíření

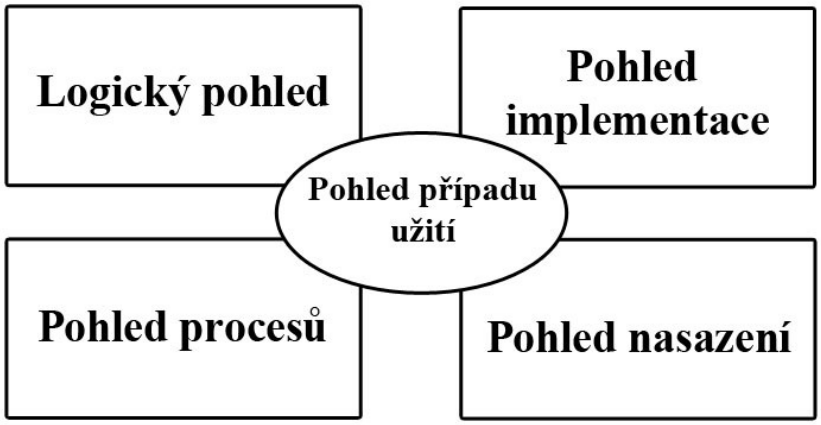
UML zahrnuje tři jednoduché mechanismy umožňující jeho rozšíření:

- Omezení – omezující podmínky umožňují přidávat nová pravidla k prvkům modelu.
- Stereotypy – stereotyp umožňuje definovat nový prvek modelu UML, jenž je založen na existujícím prvku.
- Označené hodnoty – označené hodnoty umožňují rozšířit specifikaci prvku tím, že k takovému prvku přidáme informaci sestavenou jen k tomuto účelu. [3]

3.1.5 Architektura

Jazyk UML zachycuje architekturu jako organizační strukturu systému, která je rozložená do součástí, které jsou vzájemně propojeny. Architektura „4+1“ zachycuje strategické aspekty systému.

- Logický pohled – zachycuje slovník oblasti problému jako množinu tříd a objektů.
- Pohled procesů – modeluje spustitelná vlákna a procesy jako třídy.
- Pohled implementace – modeluje soubory a komponenty, které tvoří hotový kód systému.
- Pohled nasazení – modeluje fyzické nasazení komponent na množinu fyzických uzlů.
- Pohled případů užití – všechny pohledy jsou sloučeny v pohledu případů užití, který popisuje požadavky uživatele. [3]



Obrázek 2 – Architektura 4+1. Zdroj: vlastní

3.2 Unified Process

Unified Process je proces vývoje softwaru, který je zkráceně nazýván UP. Kořeny metodiky UP sahají až do roku 1967, jehož pojetí vycházelo z faktů, že složité systémy se mají modelovat jako vzájemně propojené bloky. Jeho vývoj postupně pokračoval, až v roce 1995 dostal podobu architektury „4+1“. Tento model je dodnes základem pojetí UP. V roce 1998 vznikla komerční metodika RUP, která rozšiřuje metodiku UP. Tato metodika je úplnější a podrobnější.

Metodika UP se snaží rozdělit projekt na řadu menších projektů. Každý menší projekt se nazývá iterace. V každé iteraci existuje pět základních postupů.

1. Požadavky – zachycují co by měl systém dělat.
2. Analýza – upřesňuje strukturu a požadavky.
3. Návrh – realizace požadavků v architektuře systému.
4. Implementace – vytváření softwaru.
5. Testování – ověření funkčnosti implementace.

Rozdělením projektu do iterací dosáhneme dynamického plánování. Iterace mohou být realizovány souběžně, tím se zkrátí doba celého projektu a také dosáhneme lepšího využití vývojového týmu.

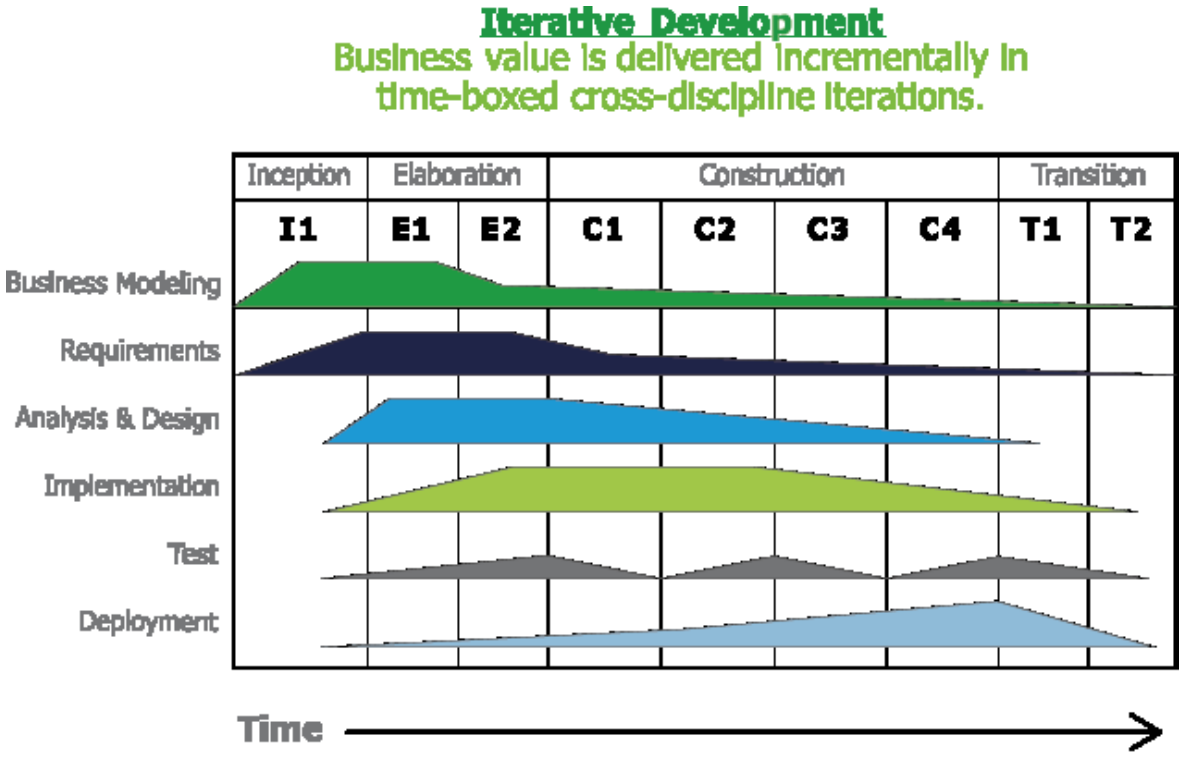
Životní cyklus projektu je rozdělen do čtyř fází. Jakákoliv fáze může obsahovat jednu nebo více iterací. V každé iteraci můžeme realizovat pět základních postupů. Počet iterací v jednotlivých fázích není přesně stanoven, záleží na velikosti projektu.

UP se skládá ze čtyř po sobě následujících fází. Každá fáze má určitý cíl a je zakončena milníkem, kterého musíme dosáhnout, abychom ji dokončili.

1. Zahájení (Inception) – Cílem této fáze je zahájení projektu. Hlavní důraz v této fázi je kladen na zachycení podstatných požadavků, jež definují rozsah vznikajícího projektu a jejich analýzu. Milníkem jsou záměry životního cyklu jako představení

hlavních požadavků na projekt, realizace rizik projektu a zda je projekt realizovatelný z technického hlediska a jeho financování.

2. Rozpracování (Elaboration) – Tato fáze je zaměřena na tvorbu částečně spustitelného systému. Jedná se o spustitelný architektonický základ, který bude v dalších fázích rozšiřován. V této fázi také vylepšujeme odhad rizik a případy užití. Milníkem je architektura životního cyklu programového díla.
3. Konstrukce (Construction) – Je to fáze, v níž je vytvářen software. Konstrukce spustitelného architektonického základu je postupně modifikována v kompletní funkční systém. Milníkem je počáteční funkční varianta, která je připravena pro testování na počítačích uživatele.
4. Zavedení (Transition) – Ve fázi zavedení je nasazen software do uživatelského prostředí. Tato fáze začíná v okamžiku, kdy je dokončeno testování a konečné nasazení systému. Milníkem je nasazení produktu do užívání na pracoviště uživatele. [3]



Obrázek 3 – Fáze podle metodiky UP. Zdroj: dostupný na www [5]

3.3 Case nástroje

Case nástroje představují nástroje pro podporu vývoje informačního systému v celém průběhu jeho vývoje. Vycházejí z modelovacího jazyka UML. Plnohodnotné CASE nástroje umožňují propojení jednotlivých technik UML a sdílení modulů mezi členy týmu a tedy týmovou práci.

Tyto nástroje zpravidla obsahují možnosti generování a synchronizaci kódů v objektových prostředích, včetně generování skriptů pro vytvoření databáze, reverzního engineeringu

a správy datových modelů. Většina dnešních nástrojů CASE umožňuje do jisté míry i procesní modelování. Produkty CASE jsou ideálním nástrojem pro návrh software, umožňují řízení softwarových projektů. Při tvorbě systému byl použit nástroj Enterprise Architect. [4]

Enterprise Architect

Enterprise Architect je profesionální CASE nástroj od firmy Sparx Systems. Jedná se komplexní nástroj pro analýzu a návrh softwaru. Poskytuje plnou podporu vývoje software v celém jeho životním cyklu. Enterprise Architect poskytuje mnoho nástrojů, jako je například: tvorba softwarového designu, generování kódu, testování, verzování modelů, vytváření business modelů, modelování databází a mnoho dalšího.

Tento CASE nástroj jde také integrovat do vývojového prostředí Eclipse nebo Visual Studia. Zde poskytuje přímý přístup k modelům, generování kódu a mnoho různých funkcionalit. Pomocí tohoto nástroje lze také generovat komplexní dokumentaci s detailními informacemi. Jedná se tedy o velmi rozsáhlý CASE nástroj, který je zejména vhodný pro nasazení v rozsáhlých týmech. [6]

3.4 Požadavky

Pod pojmem požadavek si můžeme představit jisté funkce nebo vlastnosti, které by měly být implementovány ve vyvíjeném systému. Požadavky můžeme rozdělit na dva základní typy: funkční a nefunkční. [3]

3.4.1 Sběr požadavků

Proces získávání požadavků je velmi důležitá část celého projektu. Nedostatečné specifikování požadavků a nedostatečné zapojení uživatelů do procesů správy požadavků může být hlavní příčina neúspěšného projektu. [4]

Získávání požadavků tohoto projektu probíhalo jak konzultací se zadavatelem, tak navštívením firmy, kterou zadavatel využívá pro velkoplošný tisk tiskovin. Dále k získání požadavků bylo využito zadání diplomové práce.

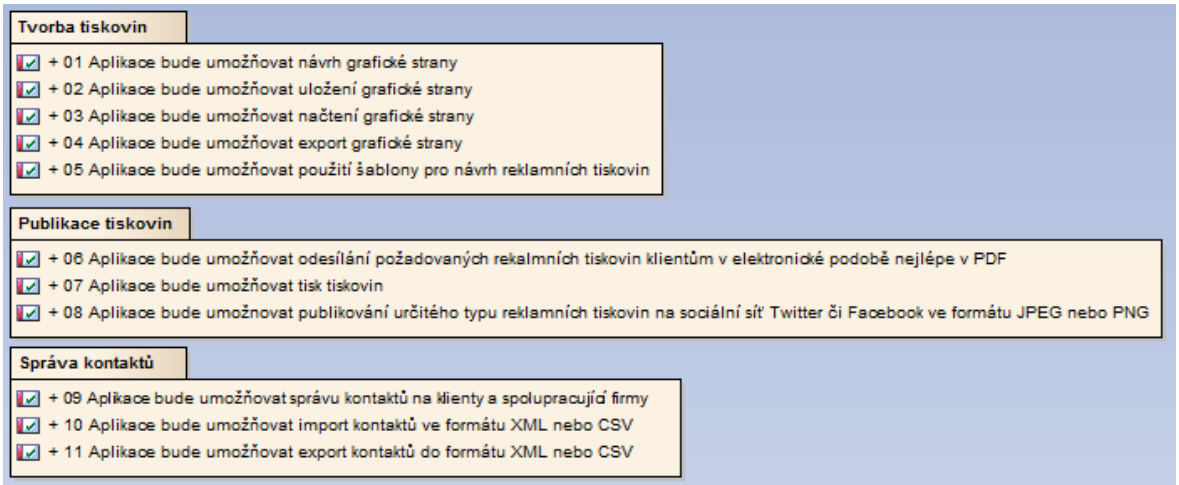
3.4.2 Funkční požadavky

Funkční požadavky nám specifikují požadavky na funkčnost systému. Jaké chování bude systém nabízet.

Jazyk UML neposkytuje žádné doporučení pro zápis požadavků a proto byl využit zápis, který byl doporučen v literatuře [4]

<id> <systém> bude <funkce>

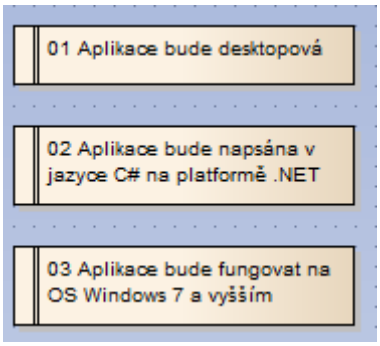
Na obrázku č. 4 jsou zobrazeny funkční požadavky aplikace, které jsou rozděleny do kategorií Tvorba tiskovin, Publikace tiskovin, Správa kontaktů.



Obrázek 4 – Funkční požadavky aplikace. Zdroj: vlastní

3.4.3 Nefunkční požadavky

Nefunkční požadavky nám specifikují vlastnosti systému, případně omezující podmínky. Na následujícím obrázku jsou zobrazeny nefunkční požadavky aplikace.



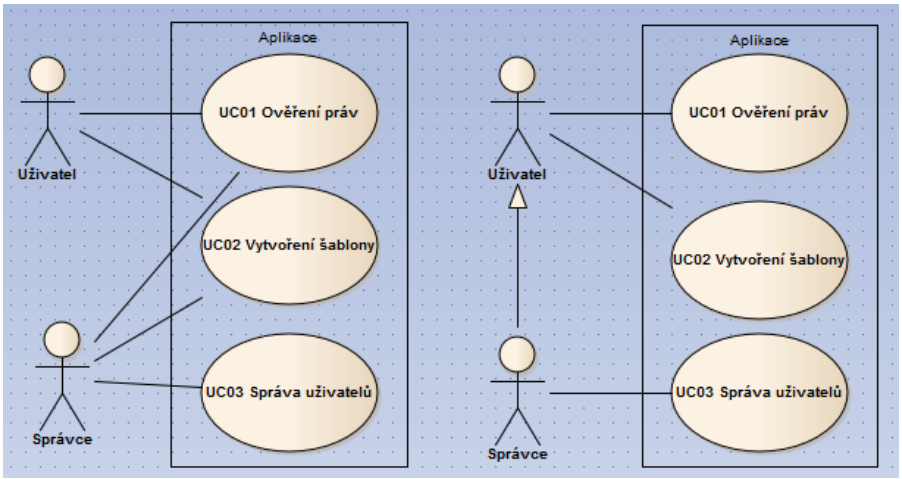
Obrázek 5 – Nefunkční požadavky aplikace. Zdroj: vlastní

3.5 Případy užití

Případy užití zachycují funkce, které systém poskytuje k užití jednoho nebo více aktérů. Každý případ užití popisuje jeden způsob použití systému, jednu požadovanou funkčnost. [3]

3.5.1 Aktéři

Aktér nebo také účastník vyjadřuje roli, ve které komunikuje s daným systémem. V roli aktéra může vystupovat uživatel systému, externí systém nebo také čas. Aktéři spouští jednotlivé případy užití a ty jsou psány z pohledu aktéra. Při hledání aktérů může vzniknout situace, kdy najdeme dva aktéry, kteří se od sebe liší pouze tím, že jeden aktér spouští navíc jiné případy užití. V tomto případě je vhodné využít generalizaci aktérů. Na následujícím obrázku můžeme vidět ukázku generalizace aktérů. Z obrázku je jasné, že při použití generalizace se graficky zjednodušil diagram. [3]



Obrázek 6 – Generalizace aktérů. Zdroj: vlastní

Před modelováním případů užití aplikace bylo nutné nejprve najít aktéry, kteří komunikují se systémem. S navrhovaným systémem komunikuje pouze jeden aktér, který se nazývá *Uživatel*.

3.5.2 Relace mezi případy užití

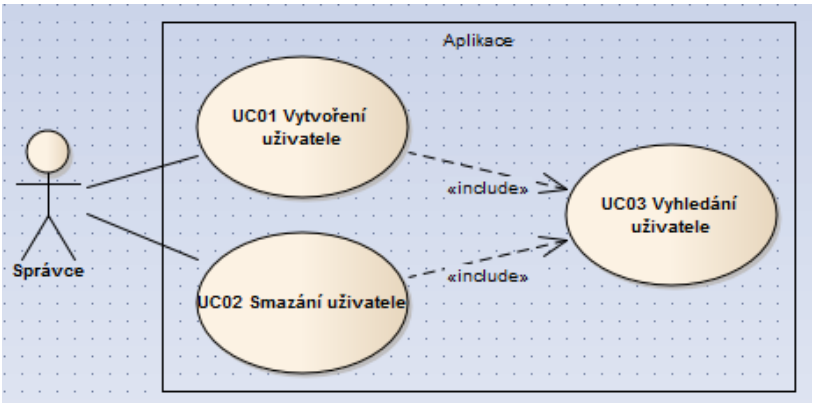
Mezi případy užití mohou také vznikat vazby:

- Relace <<include>>
- Relace <<extend>>
- Generalizace případů užití

Relace <<include>> se používá:

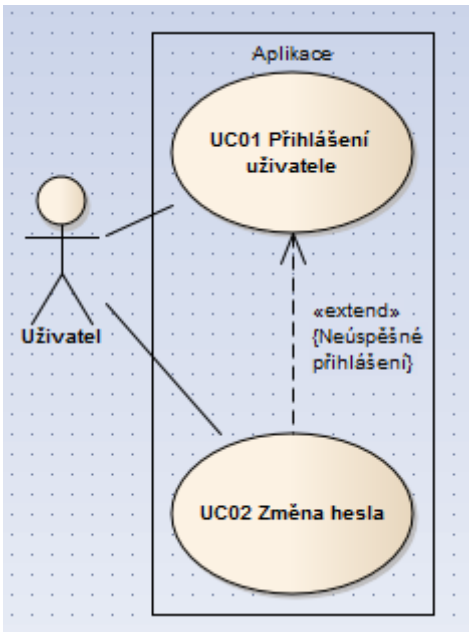
1. Pokud má více případů užití stejnou část scénáře. Jedná se tedy o vyčlenění společného chování několika scénářů.
2. Pro zpřehlednění scénáře. Pokud máme jeden dlouhý scénář nebo mnoho větvení ve scénáři. [3]

Na obrázku č. 7 můžeme vidět použití relace <<include>>. Abychom neustále nepopisovali postup vyhledání uživatele, vytvoříme samostatný případ užití *Vyhledání uživatele.* Případy užití, které zahrnují chování *Vyhledání uživatele,* s ním budou propojeny pomocí relace <<include>>.



Obrázek 7 – Relace <<include>> mezi případy užití. Zdroj: vlastní

Relace <<extend>> rozšiřuje případ užití o nové chování z jiného případu užití. Na rozdíl od relace <<include>> je bázový případ užití zcela soběstačný. Bázový případ užití má připravená místa rozšíření, které nejsou součástí scénáře, ale ukazují místo ve scénáři, kde může být funkčnost dále rozšířena. Rozšiřující případy užití mohou mít také vstupní a výstupní podmínky. Na obrázku č. 8 je znázorněn příklad vazby <<extend>> mezi případy užití *Přihlášení uživatele* a *Změna hesla*. Rozšiřující případ užití *Změna hesla* je proveden, pokud je splněna vstupní podmínka „Neúspěšné přihlášení“, nebo ho může uživatel spustit samostatně. [3]



Obrázek 8 – Relace <<extend>> mezi případy užití. Zdroj: vlastní

Generalizace případu užití znamená vyjmutí společného chování dvou a více případů užití do rodičovského případu užití. [3]

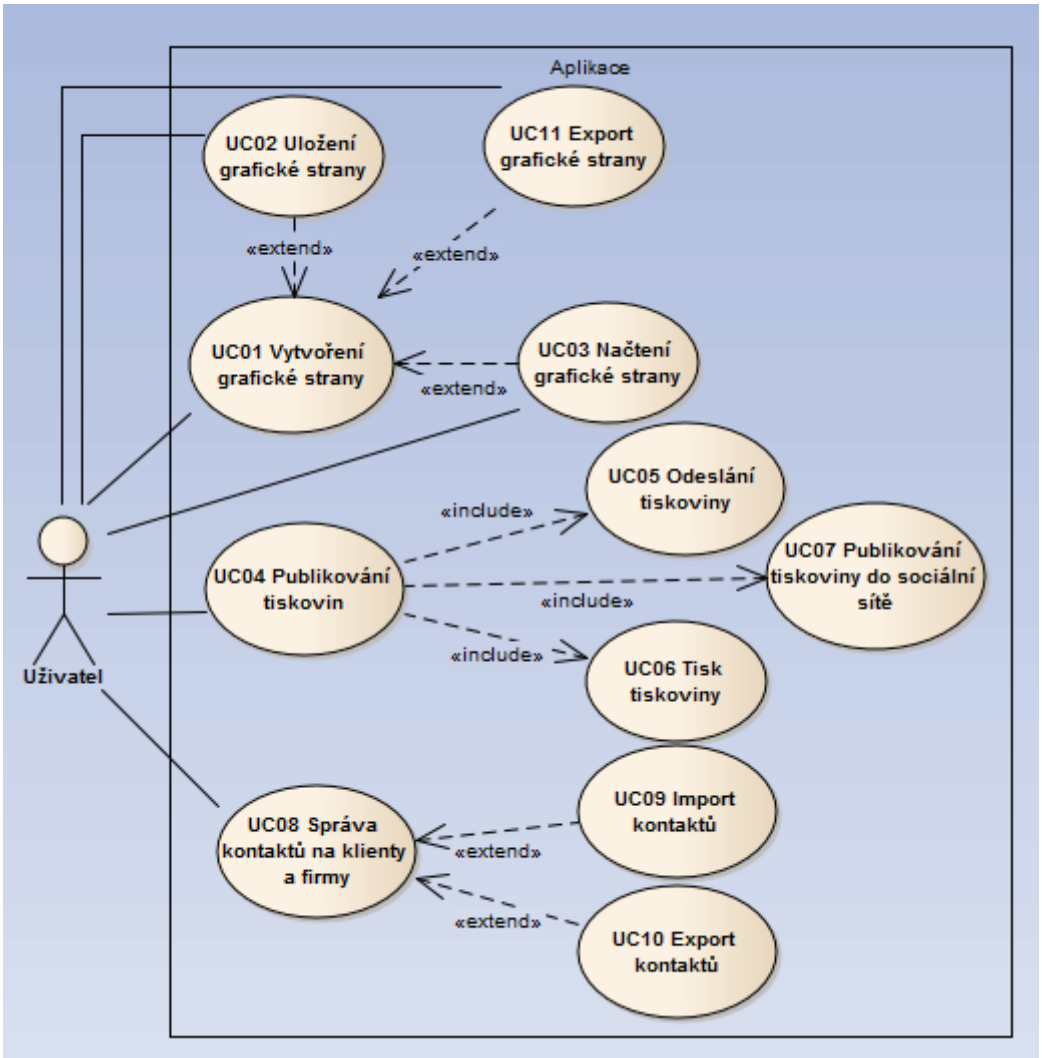
3.5.3 Případy užití navrhovaného systému

Na obrázku č. 9 jsou zobrazeny případy užití navrhovaného systému.

Případ užití *vytvoření grafické strany* nám reprezentuje jednu z hlavních funkcí systému. Zachycuje jednotlivé kroky pro zhotovení tiskoviny. Je rozšířen pomocí případů užití *načtení, uložení a export grafické strany*.

Případ užití *Publikování tiskovin* nám reprezentuje veškeré možnosti zveřejnění tiskoviny. Pro zpřehlednění scénáře byla využita relace include na případy užití *odeslání tiskoviny, publikování tiskoviny do sociální sítě, tisk tiskoviny*.

Případ užití *správa kontaktů* na klienty firmy zahrnuje vytvoření, smazání a editování kontaktů a je rozšířen pomocí případů užití *import a export kontaktů.*



Obrázek 9 – Případy užití navrhovaného systému. Zdroj: vlastní

3.5.4 Scénáře případu užití

Scénář popisuje jednotlivé kroky případu užití, které znázorňují interakci mezi aktérem a systémem. Při návrhu systému byly použity strukturované scénáře. Na obrázku č. 10 je znázorněn hlavní scénář případu užití *Publikování tiskovin do sociální sítě* navrhovaného systému.

Step	Action	Uses	Results	State
1	Uživatel zvolí možnost publikování tiskoviny na sociální síť v GUI menu		Zobrazení formuláře	
2	Uživatel vyplní přihlašovací údaje pro přihlášení na sociální síť a potvrdí podmínky			
3	Autorizace aplikace sociální sítě		Zobrazení formuláře	
4	Vyplnění formuláře			
5	Odeslání formuláře pomocí tlačítka "Odeslat"			
6	Úspěšné publikování tiskoviny na vybranou sociální síť			

Entry Points | Context References | Constraints

Step	Path Name	Type	Join
0	Publikování do sociální sítě	Basic Path	-
6a	Neúspěšné publikování	Alternate	6

Obrázek 10 – Hlavní scénář případu užití. Zdroj: vlastní

3.6 Analýza

Modelování analýzy je strategickou aktivitou, neboť se během ní snažíme o modelování základního chování systému. Záměrem analýzy je tvorba analytického modelu, který zachycuje podstatné požadavky a charakteristické rysy požadovaného systému. Výstupem analýzy jsou analytické třídy a realizace případu užití. Realizace případů užití znázorňují vzájemnou komunikaci analytických tříd s cílem realizovat chování systému, které je specifikováno v případech užití. [4]

3.6.1 Analytické třídy

Analytické třídy by měly obsahovat klíčové atributy a množinu hlavních operací. Tyto atributy budou obsahovat i návrhové třídy. Operace v analytické třídě mohou být rozděleny na více samostatných metod v návrhové třídě.

Dle Kanisové, Mülera [4] by analytická třída měla obsahovat tyto součásti:

- Název třídy, který je povinný.
- Název atributu je také povinný, ale typ atributu nemusí být specifikován.
- Operace vyjadřují obecnou odpovědnost třídy. Operace má povinný název, ale argumenty, stejně jako návratové typy, jsou uvedeny pouze tam, kde jsou nutné k pochopení modelu.
- Typ viditelnosti nemusí být specifikován.

- Stereotypy mohou být zobrazeny, pokud vylepšují model.
- Označené hodnoty mohou být také specifikovány, pokud vylepšují model.

Název dobré analytické třídy by měl odrážet její účel. Analytická třída by měla být velmi soudržná, měla by obsahovat malou, ale správně definovanou množinu operací. Také by měla obsahovat minimum vazeb na ostatní třídy. [4]

3.6.2 Hledání analytických tříd

Hlavním úkolem analýzy je nalezení analytických tříd. Pro jejich dohledání existuje mnoho spolehlivých technik, mezi nejznámější patří hledání tříd pomocí metody podstatných jmen a sloves a hledání tříd pomocí metody CRC štítků.

Metoda podstatných jmen a sloves je technika hledání analytických tříd, která jednoduchým způsobem analyzuje všechny důležité textové zdroje informací navrhovaného systému, ve kterém chceme najít atributy a operace. Podstatná jména a fráze tvořené podstatnými jmény vyjadřují třídy nebo atributy a slovesné fráze vyjadřují operaci třídy.

Metoda CRC štítků využívá při hledání analytických tříd samolepící lístečky, které jsou rozděleny na tři oddíly:

- Classes – obsahuje název třídy
- Responsibilities – obsahuje seznam odpovědností
- Collaborators – obsahuje seznam dalších spolupracujících tříd

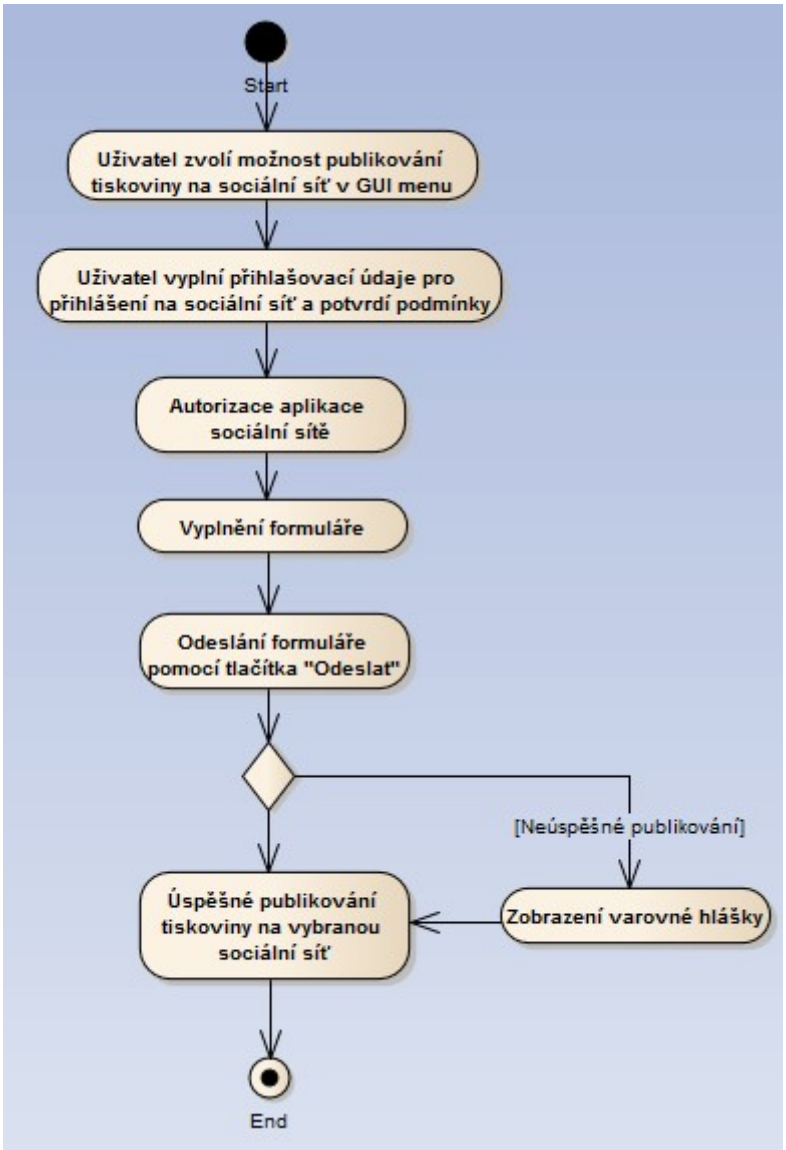
Do aktivity hledání analytických tříd se zapojují také ostatní členové týmu. V první fázi zapíší členové týmu důležité předměty problémové domény na lísteček, uvedou odpovědnosti a označí třídy, které by mohly pracovat společně. V druhé fázi členové týmu rozhodnou, zda je lísteček atribut či třída. [4]

3.6.3 Aktivity diagram

Digram aktivit modelují procesy jako kolekce aktivit a přechodů mezi nimi. Aktivity podporují jak sekvenční, tak paralelní chování. Diagramy aktivit jsou určeny především pro komunikaci s lidmi, proto by diagramy neměly být složité a nepřehledné. [3]

Diagramy aktivit se nejčastěji používají během analýzy pro grafické modelování scénářů případů užití nebo pro modelování cest mezi případy užití. Aktivity diagramy mohou být také použity během návrhu např. pro modelování podrobností operací nebo pro modelování detailů algoritmů. Také mohou být použity během modelování organizace pro modelování obchodního procesu. [4]

Na obrázku č. 11 je graficky znázorněn scénář případu užití *publikování tiskovin do sociální sítě* pomocí aktivity diagramu z navrhovaného systému.



Obrázek 11 – Aktivity diagram scénáře případu užití. Zdroj: vlastní

3.7 Návrh

Analytický model je zaměřen hlavně na tvorbu logického modelu připravovaného systému, který zachycuje funkce, které musí tento systém poskytovat, aby uspokojil požadavky uživatelů. Cílem návrhového modelu je přesná specifikace způsobu jak takové funkce implementovat. Rozšiřuje analytický model tříd o implementační třídy a detaily. [4]

3.7.1 Návrhové třídy

Návrhové třídy jsou specifikovány na takové úrovni, že je lze snadno implementovat. Návrhové třídy lze získat ze dvou zdrojů:

- Z problémové domény postupným upřesňováním analytických tříd.
- Z domény řešení z knihoven existujících užitkových tříd, znovupoužitelných komponent, knihoven grafického uživatelského rozhraní nebo implementačních detailů.

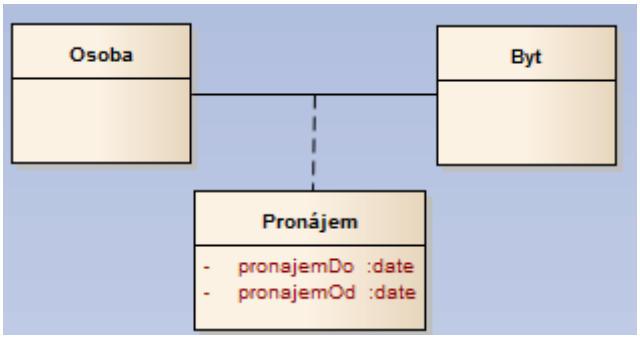
Návrh spočívá v modelování způsobu, jakým má být toto chování implementováno. Návrhová třída oproti analytické třídě musí obsahovat:

- Kompletní sadu atributů a jejich detailní specifikace včetně názvu, typu, viditelnosti a nepovinně implicitní hodnoty.
- Převod operací specifikovaných v analytické třídě na úplnou sadu metod.
- Konstruktory a destruktory.
- Get a Set metody.

Pokud nastane případ, že návrhová třída je příliš velká, je vhodné rozdělit třídu na dvě nebo více menších tříd. Správně formulovaná návrhová třída by měla obsahovat tyto vlastnosti:

- Jednoduchost – obsahuje jednoduché nedělitelné metody.
- Úplnost a dostatečnost – obsahuje vše pro implementaci, její metody, odráží její účel.
- Soudržnost – vnitřní soudržnost a minimum vazeb na jiné třídy.
- Dobré pojmenování – název třídy odráží její účel.
- Veřejné metody správně definují funkce poskytované ostatním klientům.

Při přechodu z analytických tříd k návrhovým je také důležité doplnit rozhraní, šablony, vnořené třídy a stereotypy. Doplnit chybějící relace a upřesnit analytické relace. Vztahy určené jako asociace upřesníme do relací typu agregace nebo kompozice. Obousměrné asociace je nutné rozdělit na dvě jednosměrné. Relace typu M:N také rozdělíme pomocí asociační třídy. Obrázek č. 12 znázorňuje použití asociační třídy.



Obrázek 12 – Asociační třída. Zdroj: vlastní

Pokud jsou návrhové třídy detailně specifikovány, lze pomocí vhodného CASE nástroje vygenerovat zdrojový kód. [4]

3.7.2 Návrhové vzory

Návrhový vzor je obecný postup, podle kterého tvoříme návrh systému. Návrhový vzor není knihovna nebo část zdrojového kódu, který by se dal vložit přímo do programu. Jedná se o popis řešení problému nebo šablony, která může být použita v různých situacích.

Návrhové vzory můžeme rozdělit na základní typy:

- Creational patterns (vytvářející návrhové vzory) – Řeší problémy související s vytvářením objektů v systému. Snahou těchto návrhových vzorů je popsat postup výběru třídy nového objektu a zajištění správného počtu těchto objektů. Většinou se jedná o dynamická rozhodnutí učiněná za běhu programu.

- Structural patterns (strukturální návrhové vzory) – Představují skupinu návrhových vzorů zaměřujících se na možnosti uspořádání jednotlivých tříd nebo komponent v systému. Snahou je zpřehlednit systém a využít možnosti strukturalizace kódu.

- Behavioral Patterns (návrhové vzory chování) – Zajímají se o chování systému. Mohou být založeny na třídách nebo objektech. U tříd využívají při návrhu řešení především principu dědičnosti. V druhém přístupu je řešena spolupráce mezi objekty a skupinami objektů, která zajišťuje dosažení požadovaného výsledku. [7]

Při realizaci diplomové práce byl využit návrhový vzor *Singleton*, který patří do skupiny vytvářejících návrhových vzorů. Už z názvu vyplývá, že se jedná o jednu instanci, která je sdílená mezi třídami, které ji používají. Nejčastější využití tohoto návrhového vzoru je databázové připojení, kdy celý program pracuje s jedním připojením.

Vzor je tvořen třídou, která se stará o to, aby její instance existovala jenom jednou. V první řadě vytvoříme privátní konstruktor, tím zakážeme vytvoření instance. Dále vytvoříme privátní statický atribut, který uchovává instanci, kterou sdílíme. V poslední řadě vytvoříme statickou metodu, která je pojmenovaná *GetInstanc*. Metoda je veřejná a stará se o vytvoření instance třídy. Pokud instance třídy již existuje, vrátí již existující instanci třídy. Třída tedy vytvoří instanci sebe sama a tu uloží do statické proměnné. Instanci třídy máme tedy zcela pod kontrolou, nezíská ji žádná jinačí třída, protože ji nemůže vytvořit.

Ukázka návrhového vzoru *Singleton*:

```
class DBConnection
{
    private static DBConnection instance;
    private SqlCeConnection spojeni = new SqlCeConnection(@"Data Source
    =|DataDirectory|\PublikDb.sdf; Password ='*****'");

    private DBConnection()
    {

    }

    public static DBConnection GetInstanc()
    {
        if (instance == null)
        {
            instance = new DBConnection();
        }
        return instance;
    }
}
```

3.8 Datové modelování

Cílem datového modelování je navrhnout datový model pro ukládání dat, který bude využívat informační systém.

Při datovém modelování rozlišujeme dva druhy datových modelů:

- Logický datový model, který je nezávislý na druhu použité relační databáze.
- Fyzický datový model, jež zohledňuje použitou relační databázi. [4]

3.8.1 Mapování diagramu tříd

Jakmile máme vytvořený diagram tříd a potřebujeme navrhnout datový model, můžeme využít mapování diagramu tříd na tabulky. Mezi modelem tříd a relačním databázovým modelem můžeme vidět určitou podobnost. Přesto ne všechny využité konstrukty na modelu tříd lze použít na datový model.

Atributy z diagramu tříd se stanou sloupcem tabulky, instance objektů potom odpovídají řádkům tabulky. Mapování atributů musí odpovídat konverzi datových typů mezi navrženými atributy tříd a formáty sloupců tabulek fyzické databáze. Tomuto kroku je nezbytné věnovat patřičnou pozornost, protože datové typy nebo jejich názvy se mohou lišit dle druhu použité relační databáze. Při tvorbě diplomové práce byla použita databáze od firmy Microsoft. [4]

Tabulka 2 – Mapování atributů. Zdroj: dostupný na www [8]

Datový typ Microsoft SQL Server	Type v System.Data.SqlTypes nebo Microsoft.SqlServer.Types	Datový typ v .NET Framework
bigint	SqlInt64	Int64, Nullable<Int64>
binary	SqlBytes, SqlBinary	Byte[]
bit	SqlBoolean	Boolean, Nullable<Boolean>
char	None	None
date	SqlDateTime	DateTime, Nullable<DateTime>
datetime	SqlDateTime	DateTime, Nullable<DateTime>
int	SqlInt32	Int32, Nullable<Int32>
nvarchar	SqlChars, SqlString	String, Char[]
tinyint	SqlByte	Byte, Nullable<Byte>

Při mapování asociací se můžeme setkat s asociacemi typu 1:1, 1:N a M:N. Asociace typu 1:1 vedou ke vzniku pouze jedné tabulky. Asociace typu 1:N znamenají, že musíme vytvořit dvě tabulky a pomocí cizího klíče odkazujeme na nadřazenou tabulku. Obě tabulky musí mít primární klíč. Asociace typu M:N mezi třídami vedou na vytvoření vazební tabulky. Mapování agregací se modeluje stejným způsobem jako asociace.

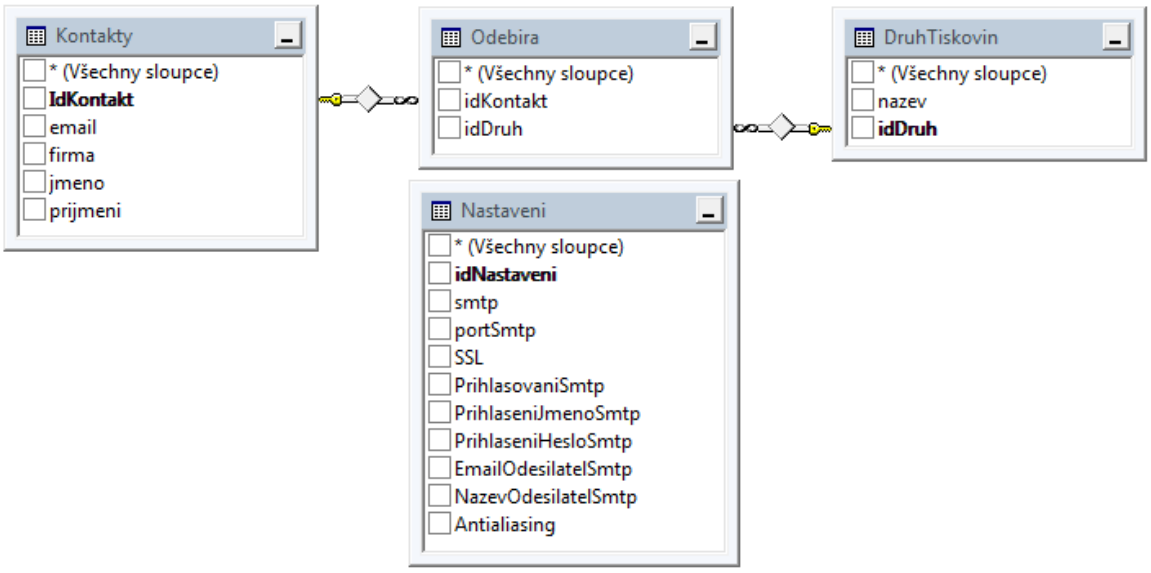
Mapování dědičnosti je z hlediska mapování složitější. Rozlišujeme tři druhy mapování dědičnosti:

- Mapování dědičnosti 1:1 – Všechny třídy se mapují do samostatné tabulky a všechny tabulky mají stejný primární klíč.

- Mapování dědičnosti zahrnutím do nadtřídy – Tento způsob spočívá v tom, že atributy podtříd jsou zahrnuty do nadtřídy.
- Mapování dědičnosti rozpuštěním do podtříd – V tomto případě jsou přeneseny všechny atributy nadtřídy do tabulek pro všechny neabstraktní podtřídy. [4]

Datový model navrhovaného systému je zobrazen na obrázku č. 13. Tabulka *Kontakty* obsahuje sloupec *idKontakt*, který je jednoznačný identifikátor záznamů v této tabulce. Do této tabulky se ukládají informace o kontaktu. Tabulka *Odebira* je vazební tabulka, která nám rozděluje vazbu M:N. Obsahuje dva sloupce cizích klíčů, které odkazují na tabulky *Kontakty* a *DruhTiskovin*. Tabulka *DruhTiskovin* je tvořena dvěma sloupci *idDruh* a *nazev*. Sloupec *idDruh* je jednoznačným identifikátorem této tabulky.

Tabulka *Nastaveni* ukládá informace o nastavení informačního systému, obsahuje sloupec *idNastaveni*, což je primární klíč tabulky.



Obrázek 13 – Datový model navrhovaného systému. Zdroj: vlastní

3.9 Implementace

V této kapitole bude popsán komplexní postup implementace navrhovaného systému, použité technologie i jakým způsobem probíhal výběr použitých technik a způsob implementace.

3.9.1 Microsoft Visual Studio

Při implementaci navrhovaného systému bylo použito vývojové prostředí Visual Studio 2010. Tento nástroj slouží k vývoji softwarových aplikací pro desktopové i dotykové prostředí Windows, pro web/HTML 5, SharePoint, mobilní zařízení i cloud prostředí. Existuje mnoho edicí tohoto softwaru, bezplatná licence Expres, Test Professional pro testery a manažery a mnoho dalších. Pro vývoj tohoto systému byla použita edice Professional.

Visual Studio podporuje jazyky prostřednictvím jazykových služeb, což umožňuje, aby editor kódu a debugger podporoval jakýkoliv programovací jazyk. Jazyky C#, VB.NET, C/C+ jsou v nástroji integrovány. Ostatní jazyky mohou být přidány pomocí jazykových služeb.

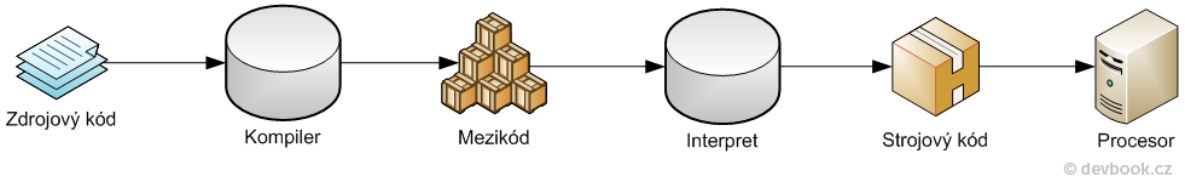
Visual Studio obsahuje editor kódu podporující IntelliSense a refaktorování. Integrovaný debugger pracuje jak na úrovni kódu, tak na úrovni stroje. Další vestavěné nástroje zahrnují designer formulářů pro tvorbu aplikací s GUI, designer webu, tříd a databázových schémat. Debugger, který pracuje jak se spravovaným kódem, tak se strojovým kódem může být použit pro debugování aplikací psaných v jakémkoliv jazyce podporovaném Visual Studiem. [9]

3.9.2 Programovací jazyk C#

„Programovací jazyk C# je jednoduchý, výkonný, objektově orientovaný programovací jazyk určený k vývoji aplikaci na platformě .NET. C# je následníkem jazyka C++ a je podobný jazyku Java. Všechny tyto jazyky mají základ v jazyce C a díky tomu je značně usnadněn přechod mezi nimi. C# disponuje automatickou alokací paměti a absencí ukazatelů. Díky uživatelské přívětivosti a kvalitnímu vývojovému prostředí Visual Studia je tento jazyk čím dál tím více oblíbenější.“ [10]

Překlad jazyka na platformě .NET

Zdrojový kód je nejprve přeložen do tzv. mezikódu CIL. Jedná se v podstatě o strojový kód, který má ale o poznání jednodušší instrukční sadu a přímo podporuje objektové programování. Tento mezikód je potom díky jednoduchosti relativně rychle interpretovatelný virtuálním strojem tedy interpretem. V případě platformy .NET je to Common Language Runtime. Výsledkem je strojový kód pro náš procesor. [11]



Obrázek 14 – Překlad jazyka na platformě .NET. Zdroj: dostupný na www [11]

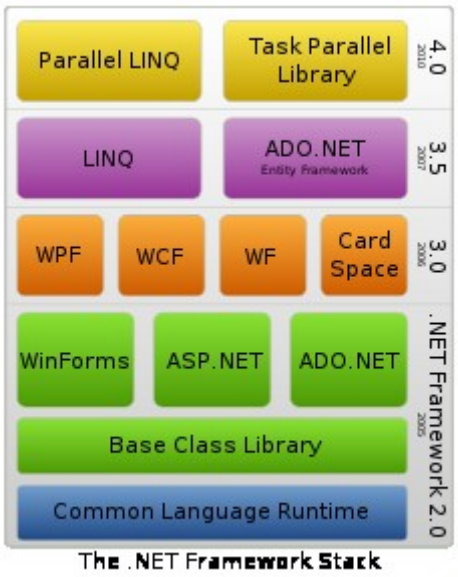
Výhody platformy .NET:

- Odhalení chyb ve zdrojovém kódu díky kompilaci do CIL.
- Stabilita – Interpret nás zastaví před vykonáním nebezpečné operace.
- Jednoduchý vývoj – Možnost použití datových struktur a knihoven. Správu paměti provádí garbage collector.
- Málo zranitelný kód – Zdrojový kód se šíří v CIL, není jednoduše lidsky čitelný.
- Přenositelnost – Na jednom projektu může dělat více lidí v různých jazycích, zdrojový kód se vždy přeloží do CIL. [11]

3.9.3 .NET Framework

.NET Framework je rozsáhlá platforma, která je vhodná pro vývoj mnoha různých druhů aplikací. Díky ní lze vyvíjet nejenom klasické aplikace pro Windows, ale mimo jiné i webové aplikace a služby nebo aplikace pro mobilní zařízení.

Framework obsahuje kromě již zmíněného CLR také velkou sadu knihoven a funkcí. Tyto knihovny disponují řadou struktur a komponent, které nám zajišťují, že nemusíme psát věci, které se používají často. Jako například práce s konzolí, databází nebo formulářovými prvky. Pro běh aplikace je nutné, aby koncové zařízení mělo nainstalovanou stejnou verzi .NET Frameworku, ve které byla vyvinuta aplikace. Na následujícím obrázku je znázorněna struktura .NET Frameworku. [11]



Obrázek 15 – Struktura .NET Frameworku. Zdroj: dostupný na www [11]

Na obrázku je vidět co obsahují jednotlivé verze .NET, verze 2.0 obsahuje již zmíněný CLR a základní knihovnu Base Class Library. Verze 3.0 přinesla nový směr vývoje formulářových aplikací a procesů. Verze 3.5 obsahuje dotazovací jazyk LINQ a komponentu ADO .NET. Při implementaci navrhovaného systému byla použita verze 4.0. S touto verzí frameworku se dá pracovat ve vývojovém prostředí Visual Studia 2010. Nově nám umožňuje efektivně provozovat LINQ na vícejádrových procesorech a použití dynamických da-

tových typů. Nejnovější verze je 4.5, která je součástí operačního systému Windows 8 a pracovat s touto verzí frameworku se dá v prostředí Visual Studia 2012. [11]

3.9.4 Databáze

Před zahájením implementace navrhovaného systému bylo nutné vybrat vhodnou databázi. Jelikož systém měl být implementován na platformě .NET v jazyce C#, který je propagován firmou Microsoft. Bylo vhodné pro zachování co největší kontability použít databázi od stejné firmy. Dále bylo nutné vybrat bezplatnou edici, do úvahy tedy připadala edice Express nebo Compact.

Edice Express je vhodná pro nenáročné webové a malé serverové aplikace. Edice Compact je určena pro použití jednodušších aplikací pro Windows a malá zařízení, kde se počet záznamů nevyšplhá do milionových hodnot. Proto tato edice byla vhodnější pro navrhovaný systém, který obsahuje jedinou větší tabulku kontaktů, která nebude obsahovat velký počet záznamů.

Jedná se o jednoduchou relační databázi. Záznamy jsou ukládány do jediného databázového souboru s příponou SDF. Velký rozdíl oproti ostatním edicím spočívá v tom, že neběží jako systémová služba, ale je součástí dané aplikace, tedy v rámci jejího procesu a paměťového prostoru. Databáze vyžaduje 1,7 MB prostoru na disku a přibližně 5MB prostoru v operační paměti.

Databáze SQL CE nepodporuje uložené funkce, procedury, pohledy nebo triggery a maximální velikost databáze je omezena na 4GB. Přístup do databáze může být chráněný heslem a zabezpečení dat na disku můžeme docílit pomocí šifrování databáze. Databáze nepodporuje nastavení přístupových rolí. [12]

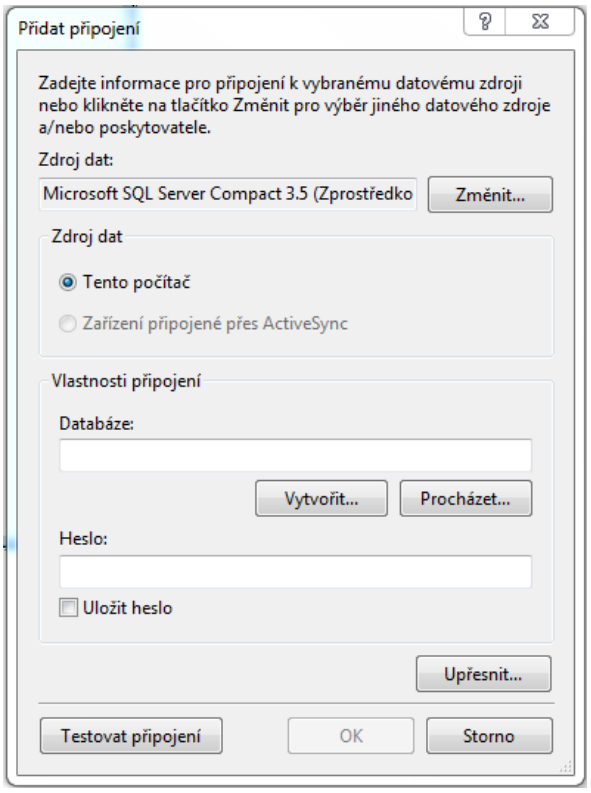
Aktuální verze SQL CE databáze je 4.0, která neposkytuje veškerou funkcionalitu verze 3.5, nepodporuje LINQ To SQL, není podporována v mobilních zařízeních a neobsahuje několik dalších funkcionalit. Na jednom systému může byt nainstalováno více verzí databází SQL CE. [13]

Tabulka 3 – Srovnání databází. Zdroj: dostupný na www [12]

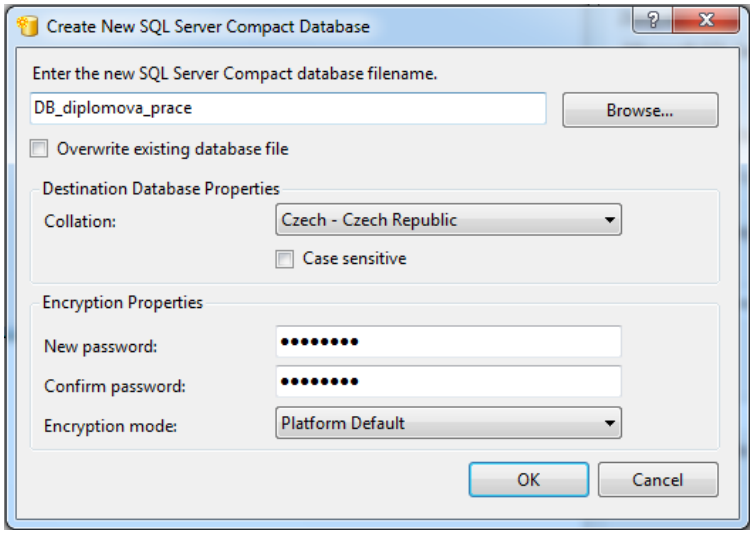
Vlastnost	SQL CE 3.5	SQL CE 4.0	EXPRESS 2008 R2
Velikost instalace	2,5 MB instalátor 18 MB na disku	2,5 MB instalátor 18 MB na disku	2,5 MB instalátor 18 MB na disku
Instalace pomoci MSI	ANO	ANO	ANO
Instalace bez admin. oprávnění	ANO	ANO	NE
Použití na ASP.NET	NE	ANO	ANO
Použití na Windows Phone	ANO	NE	NE
Běh v procesu s aplikací	ANO	ANO	NE
Běh jako služba	NE	NE	ANO
Podpora 64-bit	ANO	ANO	ANO
Souborový formát	Jeden soubor	Jeden soubor	Více souborů
Maximální velikost DB	4GB	4GB	10GB
Binární (BLOB) úložiště	ANO (jako image)	ANO (jako image)	ANO
Transact-SQL	ANO	ANO	ANO

Procedural T-SQL	NE	NE	ANO
LINQ to SQL	ANO	NE	ANO
Transakce	ANO	ANO	ANO
XQuery / XPath	NE	NE	ANO
Uložené procedury, pohledy, triggery	NE	NE	ANO
Počet současných připojení	256	256	Neomezeně

Pro tvorbu schématu databáze můžeme využít SQL Server Managment Studio pouze do verze 2008 nebo integrovanou podporu ve vývojovém prostředí Visual Studia. Při implementaci navrhovaného systému bylo použito prostředí Visual Sdtudio 2010, které plně integruje použitou databázi SQL Compact Edition 3.5. Pomocí panelu „Průzkumník serveru“ můžeme pohodlně přidat nový soubor s databází, následně vytvářet tabulky, definovat jejich schéma a vytvářet vztahy mezi tabulkami. Pomocí volby „Přidat připojení“ vytvoříme novou databázi, kde v dialogu stačí vybrat druh databáze a vyplnit požadovaná nastavení databáze.[12]



Obrázek 16 – Dialog přidaní DB pomocí Visual Studia 2010. Zdroj: vlastní



Obrázek 17 – Dialog nastavení DB pomocí Visual Studia 2010. Zdroj: vlastní

Pro použití databáze v aplikaci stačí do *connection string* přidat atribut *Data Source* a cestu k databázovému souboru. Vlastnost *DataDirectory* představuje adresář, ve kterém je aplikace umístěna. Pokud je databáze zabezpečena, je nutné také přidat atribut *Password*. [12]

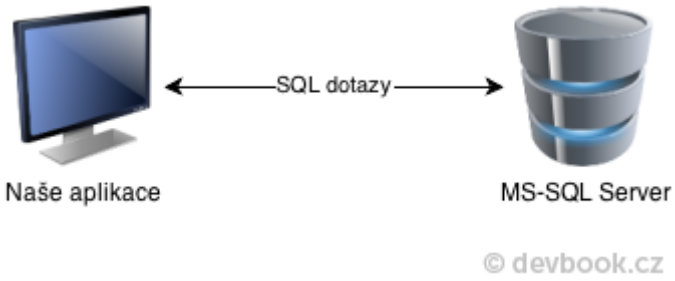
```
SqlCeConnection(@"Data Source =|DataDirectory|\PublikDb.sdf; Password ='*****");
```

Databáze SQL CE implementuje použití transakcí pomocí instance třídy *SqlCeTransaction*. V následujícím zkráceném kódu je zobrazen způsob použití transakce.

```
connection.Connect();
SqlCeTransaction transaction = connection.BeginTransaction();
try
{
	//SqlCeCommand...
	transaction.Commit();
}
catch (SqlCeException)
{
	transaction.Rollback();
}
finally
{
	connection.Disconnect();
	transaction = null;
}
```

3.9.5 Možnosti připojení k databázi

Na platformě .NET máme několik možností použití databáze. Jedna z možností je připojená aplikace. Tento přístup se využívá pokud v reálném čase potřebujeme číst nebo měnit data. Pomocí tříd *DataReader*, *Command* a *Connection* posíláme databázi příkazy v jazyce SQL a dostáváme výsledky nebo provádíme změny. [14]

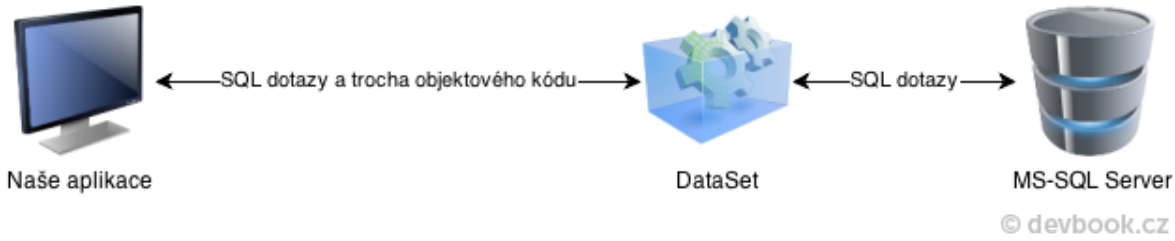


Obrázek 18 – Připojená aplikace. Zdroj: dostupný na www [14]

Odpojená aplikace uchovává v operační paměti *DataSet*, který v sobě obsahuje data z databáze. Aplikace pracuje pouze s daty z DataSetu. Jen v některých případech se sesynchronizuje se serverovou databází. Za cenu méně aktuálních dat získáme rychlejší přístup a pohodlnější práci. Tento přístup zahrnuje komponenta ADO.NET, která je součásti .NET frameworku.

Tabulka uchovávána v DataSetu je objekt, do kterého můžeme přidávat a upravovat řádky bez psaní SQL kódu. Pokud chceme spustit na databázi nějaký příkaz, použijeme *DatAdapter*, pomocí kterého naplníme *DataSet* daty. Příkaz již musíme psát v jazyce SQL dané databáze.

Pomocí DataSetu vznikne abstrakce. S tabulkami pracujeme objektově, ale data jsou pouze sloupce tabulky, ne instance objektů. [14]



Obrázek 19 – Odpojená aplikace. Zdroj: dostupný na www [14]

Další možností připojení databáze je *LINQ To SQL*, který poskytuje objektovou abstrakci nad databází. S databází pracujeme jako kdyby byla např. list objektů a vůbec neřešíme SQL dotazy. Neznáme informace o tabulkách ani sloupcích, vše se děje automaticky na pozadí. Dotaz nám vrací rovnou objekty. Nevýhoda této technologie je špatná optimalizace dotazů.

Při použití této technologie využíváme objektovou strukturu databáze *DataClasses*, která obsahuje jednotlivé tabulky. Sloupce jsou vlastnosti jednotlivých tříd. Aplikace komunikuje pouze s *DataClasses*, *LINQ To SQL* generuje na pozadí SQL dotazy, pracujeme tedy pouze s objektovou strukturou databáze v operační paměti. [14]

Obrázek 20 – LINQ To SQL. Zdroj: dostupný na www [14]

Poslední popisovaný způsob pro připojení k databázi se nazývá *Entity Framework*. Tato technologie je složitější na použití a zachází ještě dále než *LINQ To SQL*. [14]

Při implementaci navrhovaného systému byla využita technologie odpojené aplikace. V následujícím kódu je zobrazena třída *Kontakty*, která je typu *„Model“* a třídy tohoto typu představují práci s daty. Tato třída se stará o veškerou manipulaci s daty týkajících se kontaktů. Pro připojení k databázi využívá instanci třídy *DBConection*. Data načtená z databáze uchovává v instanci třídy typu *DataSet*.

```
class Kontakty
{
    private DBConnection connection = DBConnection.GetInstanc();
    private static Kontakty instance;
    private DataSet dataSet = new DataSet();
    private SqlCeDataAdapter dataAdapterKontakty = new SqlCeDataAdapter();
    private SqlCeDataAdapter dataAdapterOdebira = new SqlCeDataAdapter();
}
```

V konstruktoru této třídy využíváme *DataAdapter*, kterému nastavíme patřičný SQL dotaz a připojení k databázi.

```
private Kontakty()
{
    try
    {
       dataAdapterKontakty.SelectCommand = new SqlCeCommand("SELECT [IdKontakt],
       [email], [firma], [jmeno], [prijmeni] FROM [Kontakty]",
       connection.GetDBConnection());
       dataAdapterOdebira.SelectCommand = new SqlCeCommand("SELECT [idKontakt],
       [nazev] FROM [Odebira], [DruhTiskovin] WHERE
       [Odebira].[idDruh]=[DruhTiskovin].[idDruh]", connection.GetDBConnection());
       RefreshDataSet();
    }
    catch (SqlCeException ex)
    {
        Hlaska.HlaskaError("Chyba DB: " + ex.Message);
    }
}
```

V metodě *RefreshDataSet*, pomocí metody *Fill*, naplníme daty existující tabulky a nastavíme relace mezi jednotlivými tabulkami.

```
public void RefreshDataSet()
{
    try
    {
        dataSet.Relations.Clear();
        if (dataSet.Tables.Contains("Odebira"))
            dataSet.Tables["Odebira"].Clear();

        if (dataSet.Tables.Contains("Kontakty"))
        dataSet.Tables["Kontakty"].Clear();

        dataAdapterKontakty.FillSchema(dataSet, SchemaType.Source, "Kontakty");
        dataAdapterKontakty.Fill(dataSet, "Kontakty");
        dataAdapterOdebira.FillSchema(dataSet, SchemaType.Source, "Odebira");
        dataAdapterOdebira.Fill(dataSet, "Odebira");

        dataSet.Relations.Add("KontaktyOdebiraRelace",
        dataSet.Tables["Kontakty"].Columns["idKontakt"],
        dataSet.Tables["Odebira"].Columns["idKontakt"]);
     }
     catch (Exception ex)
     {
        Hlaska.HlaskaError("Nastala chyba: " + ex.Message);
     }
}
```

K jednotlivým tabulkám třídy *DataSet*, které jsou typu *DataTable*, můžeme přistupovat pomocí indexeru. Jako argument dosadíme pořadí tabulky nebo řetězec představující název tabulky, pod kterým byla uložena do instance třídy *DataSet*.

```
DataTable table = dataSet.Tables["Kontakty"];
```

Pokud chceme provést příkaz SELECT na datech v tabulce, použijeme metodu *Select* dané instance. Tato metoda vrátí vybrané řádky jako pole typu *DataRow*. Do metody SELECT můžeme dosadit jako argument podmínku, jež nahrazuje klauzuli WHERE v dotazu SELECT.

```
DataRow[] row = dataSet.Tables["Kontakty"].Select("email='" + email + "'");
```

3.9.6 Grafická knihovna GDI +

Pro vykreslování grafiky byla použita grafická knihovna GDI+, která zahrnuje základní třídy platformy .NET pro řízení kreslení. Tyto třídy odesílají instrukce ovladačům grafických zařízení, aby bylo možné umístit správný výstup na obrazovku nebo vytisknout na tiskárně. [15]

Rozšíření GDI

Rozhraní GDI poskytuje úrovně abstrakce, a proto nemusíme rozlišovat rozdíly mezi různými druhy grafických karet. Zavoláme pouze rozhraní Windows API pro příslušný úkol a rozhraní GDI zjistí, jak vykreslit libovolnou operaci pomocí konkrétní grafické karty. Rozhraní GDI poskytuje sice vysokoúrovňové API, ale stále se jedná o API, které je

založené na starém rozhraní Windows API s funkcemi ve stylu jazyka C. Rozhraní GDI + tvoří vrstvu umístěnou mezi rozhraním GDI a tvořenou aplikací. Poskytuje intuitivnější objektový model s podporou dědičnosti. Rozhraní GDI+ poskytuje nové funkce a do jisté míry také zlepší výkon starších funkcí GDI. [15]

V následující tabulce jsou zobrazeny hlavní jmenné prostory GDI+.

Tabulka 4 – Jmenné prostory GDI+. Zdroj: [15]

Jmenný prostor	Popis
System.Drawing	Obsahuje většinu tříd, struktur, výčtů a delegátů souvisejících se základními kreslícími funkcemi.
System.Drawing.Drawing2D	Poskytuje převážnou část podpory pokročilého dvourozměrného a vektorového kreslení, včetně vyhlazování, geometrických transformací a grafických cest.
System.Drawing.Imaging	Zahrnuje různé třídy, které pomáhají při manipulaci s obrázky.
System.Drawing.Printing	Do této třídy patří třídy, které usnadňují konkrétní výstup, pokud jako „výstupní zařízení“ funguje tiskárna nebo okno s náhledem tisku.
System.Drawing.Design	Zde se nacházejí některá předdefinovaná dialogová okna, listy, vlastnosti a další prvky uživatelského rozhraní, které souvisejí s rozšířením uživatelského rozhraní ve fázi návrhu.
System.Drawing.Text	Obsahuje třídy, které zajišťují pokročilejší manipulaci s písmy a skupinami písem.

Kontexty zařízení a třída Graphics

Kontext zařízení obsahuje informace o konkrétním zařízení a překládá funkce volané GDI API na libovolné instrukce, které je nutné libovolnému zařízení odeslat. Mezi tyto zařízení patří například obrazovka nebo tiskárna. Kontext zařízení nepracuje jenom s hardwarovými zařízeními. Funguje také jako most k systému Windows a dokáže zohlednit všechny požadavky nebo omezení, které systém na kreslení klade.

V technologii GDI+ je kontext zařízení zabalen do základní třídy platformy .NET *System.Drawing.Graphics*. Většina operací kreslení se realizuje voláním metod v instanci třídy *Graphics*. [15]

3.9.7 Double Buffering

Jedná se o techniku, která nám zabraňuje blikání obrazu. Blikání obrazu je zapříčiněno postupným překreslováním obrazu, kdy se postupně vykreslují jednotlivé objekty obrazu. V následujícím kódu můžete vidět třídu *DoubleBuffPanel*, která je potomkem třídy *Panel*. Tato komponenta využívá techniky double bufferingu a je použita pro vykreslování grafiky. V konstruktoru třídy je nastaven *DoubleBuffer*, který aktivuje vykreslování do bufferu. Dále je nastaven *AllPaintingInWmPaint*, který nám redukuje problikávání. Nastavení *UserPaint* zapříčiní, že ovládací prvek je vykreslován sám místo operačním systémem. Pro správné použití double bufferingu musí být zapnuté všechny tyto styly. [16]

```
class DoubleBuffPanel : Panel
{
    public DoubleBuffPanel()
    {
        SetStyle(ControlStyles.DoubleBuffer | ControlStyles.UserPaint |
                 ControlStyles.AllPaintingInWmPaint, true);
        UpdateStyles();
    }
}
```

Dále tato komponenta obsahuje funkce pro kreslení pomocí Win API funkcí, kdy v režimu XOR vykreslujeme objekty při přesouvání nebo při vytváření. V následujícím kódu je zobrazena funkce pro vykreslení čáry, která využívá funkce Win API. [16]

```
public void XorDrawLine(int x1, int y1, int x2, int y2)
{
       Graphics aGraphics = this.CreateGraphics();
       IntPtr HDC = aGraphics.GetHdc();
       IntPtr AktPen = DoubleBuffPanel.CreatePen((int)PenStyles.PS_SOLID, 2,
       DoubleBuffPanel.RGBColor(0, 255, 255));
       IntPtr OldPen = DoubleBuffPanel.SelectObject(HDC, AktPen);
       IntPtr OldBrush = DoubleBuffPanel.SelectObject(HDC,
       DoubleBuffPanel.GetStockObject((int)StockObjects.NULL_BRUSH));
       int OldMode = DoubleBuffPanel.SetROP2(HDC, (int)RasterOps.R2_NOTXORPEN);
       DoubleBuffPanel.MoveToEx(HDC, x1, y1, IntPtr.Zero);
       DoubleBuffPanel.LineTo(HDC, x2, y2);
       DoubleBuffPanel.SelectObject(HDC, OldPen);
       DoubleBuffPanel.DeleteObject(AktPen);
       DoubleBuffPanel.SelectObject(HDC, OldBrush);
       DoubleBuffPanel.SetROP2(HDC, OldMode);
       aGraphics.ReleaseHdc(HDC);
       aGraphics.Dispose();
}
```

3.9.8 Vykreslování grafických objektů

Aby navrhovaný systém v případě potřeby překreslil obsah okna je snadné. Systém Windows vyvolá událost *Paint*, aby aplikaci oznámil, že je nutné překreslení. [15]

Pomocí okna událostí u komponenty *DoubleBuffPanel* byla vytvořena obslužná metoda *kresleniDoubleBuffPanel_Paint*, která automaticky vytvoří *eventHandler* na událost *Paint*. Tato metoda využije parametr *PaintEventArgs*. Metoda má vlastnost *Graphics* a je optimalizována na kreslení požadované části okna. Do metody *Vykreslovani* tedy předáme odkaz na objekt *Graphics* a pomocí této metody vykreslíme grafické objekty. Způsob vykreslování je závislý na nastavení vykreslování grafiky v aplikaci.

```
private void kresleniDoubleBuffPanel_Paint(object sender, PaintEventArgs e)
{
       if (inastaveni.isAntialiasing())
         e.Graphics.SmoothingMode = SmoothingMode.AntiAlias;
       else
         e.Graphics.SmoothingMode = SmoothingMode.None;
         igrafika.Vykreslovani(e.Graphics);
}

this.kresleniDoubleBuffPanel.Paint += new
System.Windows.Forms.PaintEventHandler(this.kresleniDoubleBuffPanel_Paint);
```

Metoda Invalidate

Metoda *Invalidate* je složkou *System.Windows.Forms.Form*. Pokud tuto metodu zavoláme, znamená to, že oblast klientského okna již není platná a vyžaduje tedy překreslení. Tato metoda vyvolá událost *Paint*. Překreslení lze docílit i zavoláním metody *OnPaint* nebo některou jinou metodou pro vykreslení, ale to se považuje za nesprávný programátorský postup.

Operace kreslení patří v aplikaci GDI+ k nejnáročnějším z hlediska zatížení procesoru. Když kreslení probíhá současně s jinými operacemi, bude tyto další operace zdržovat. Pokud bychom zavolali metodu pro kreslení, tak by nemohly probíhat jiné operace, dokud by nebyla kreslící operace dokončena. Jestliže zavoláme metodu *Invalidate*, jednoduše zajistíme, že systém Windows vyvolá událost *Paint*. Události, které čekají na zpracování, jsou reprezentovány frontou zpráv. Systém frontu pravidelně kontroluje, pokud najde nějaké události na zpracování, vyjme je a zavolá odpovídající obsluhu. Ve složitějších aplikacích mohou existovat události, které by mohly dostat přednost před událostí *Paint*. [15]

Metodu *Invalidate* v navrhovaném systému voláme v mnoha případech, například při vložení nebo změně vlastností grafického objektu.

```
kresleniDoubleBuffPanel.Invalidate();
```

3.9.9 Grafické objekty

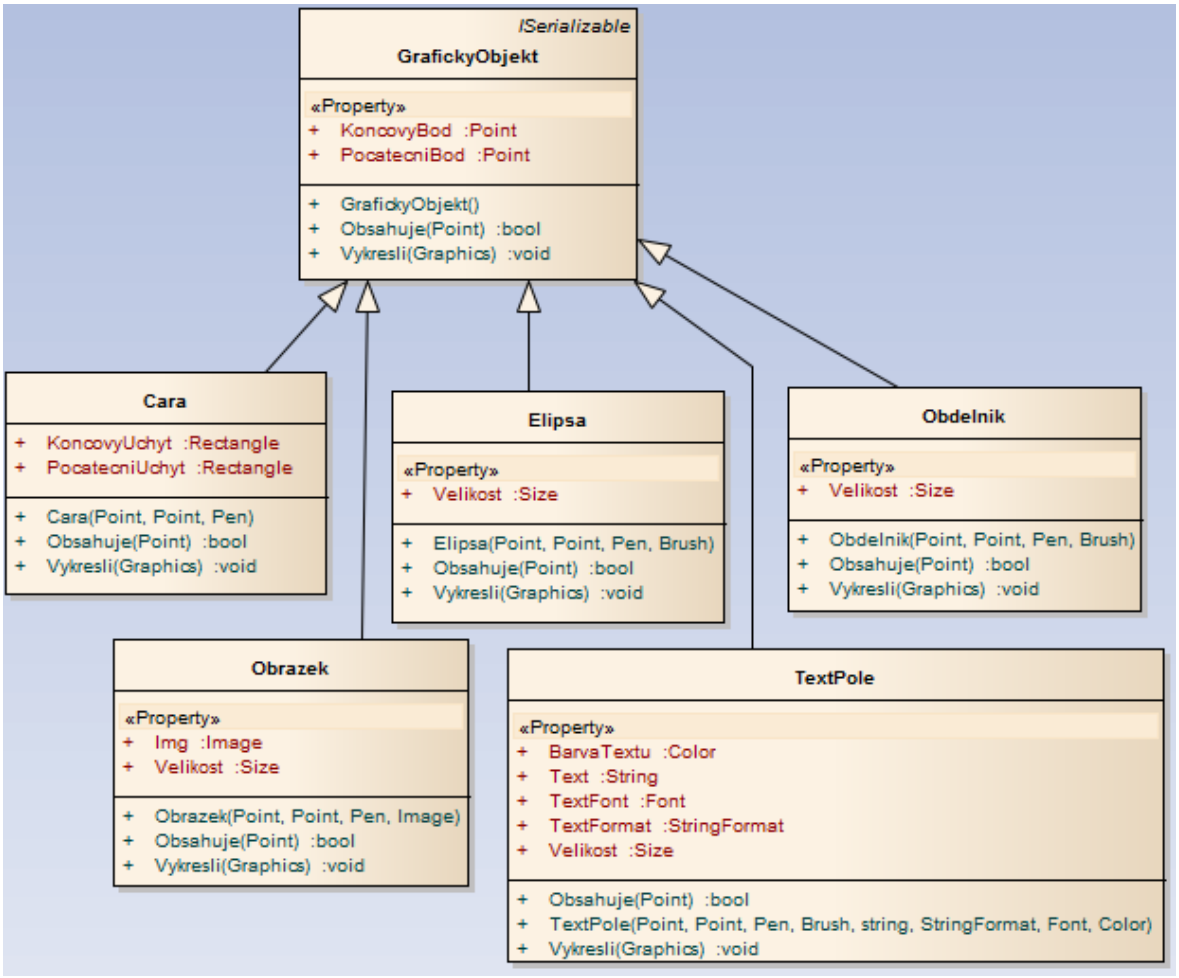
Pro uchování informací o grafických objektech využíváme třídy *Obrazek, TextovePole, Obdelnik, Elipsa, Cara*. Tyto třídy jsou potomky třídy *GrafickyObjekt*. Třída *GrafickyObjekt* uchovává základní informace o grafickém objektu, jako je například jeho počáteční a koncový bod, štětec a pero, které jsou využity při vykreslení grafického objektu. Pro vykreslení objektu je použita metoda *Vykresli*, pro zjištění, zda grafický objekt obsahuje konkrétní bod, využíváme metodu *Obsahuje*. Tyto metody jsou předefinovány v odvozených třídách, proto jsou označeny jako virtuální. Tento způsob nám zajišťuje různé implementace stejné metody.

```
public virtual void Vykresli(Graphics g){}
public virtual bool Obsahuje(Point point){ return false;}
```

V odvozených třídách pomocí klíčového slova *override* deklarujeme jinou implementaci téže metody. V následujícím kódu jsou zobrazeny metody *Vykresli* a *Obsahuje* třídy *Obdelnik*.

```
public override bool Obsahuje(Point point)
{
	Rectangle obdelnik = new Rectangle(PocatecniBod, Velikost);
	return obdelnik.Contains(point);
}
```

```
public override void Vykresli(Graphics g)
{
        if (this.Stetec != null)
          g.FillRectangle(this.Stetec, new Rectangle(PocatecniBod, Velikost));
        if (this.Pero != null)
          g.DrawRectangle(this.Pero, new Rectangle(PocatecniBod, Velikost));
        if (this.Oznacen)
          vykresleniUchytu(g);
}
```



Obrázek 21 – Diagram tříd grafických objektů. Zdroj: vlastní

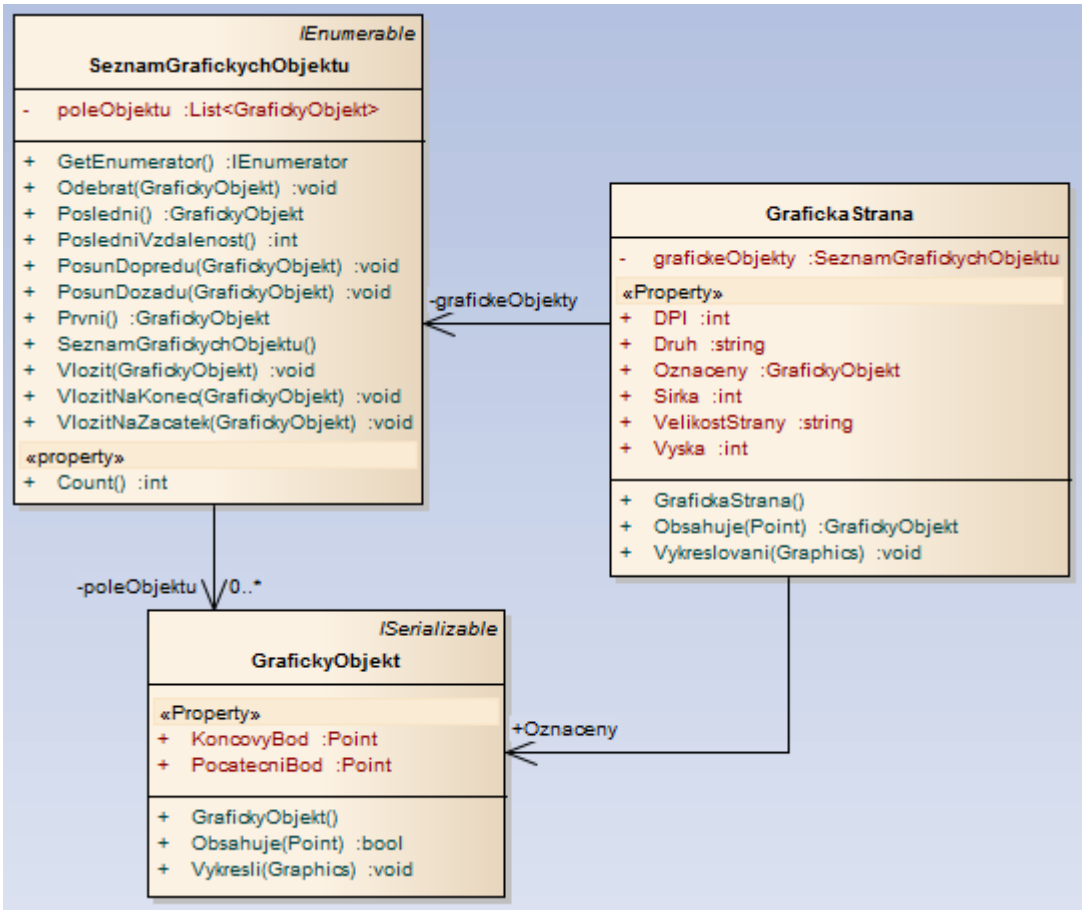
Při vykreslování grafických objektů byly použity kreslící metody třídy *Graphics*, které umožňují kreslit různé čáry, obrysy a plné tvary. Tato třída má velké množství metod, které umožňují kreslit. Proto jsou v tabulce č. 5 zobrazeny pouze metody, které byly použity pro vykreslování grafických objektů.

Tabulka 5 – Použité metody třídy Graphics pro kreslení. Zdroj [15]

Metoda	Výsledek kreslení
DrawLine	Jednoduchá čára
DrawRectangle	Obrys obdélníka
FillRectangle	Plný obdélník
DrawElipse	Obrys elipsy
FillElipse	Plná elipsa
DrawString	Text
DrawImage	Obrázek

Vytvořené grafické objekty jsou uchovávány ve třídě *SeznamGrafickychObjektu*, která uchovává seznam těchto objektů a metody pro práci s tímto seznamem. Jednotlivé grafické objekty jsou uchovávány v pořadí, v jakém budou vykreslovány. Třída *GrafickaStrana* uchovává informace o vytvořené grafické straně, jako je seznam vytvořených grafických objektů, velikost strany v pixelech, DPI a o jaký druh tiskoviny se jedná. Při vytvoření grafické strany se z velikosti formátu strany a DPI vypočítá její velikost v pixelech.

```
double sirkaPixel = Math.Round(sirka / 25.4 * DPI, 0);
double vyskaPixel = Math.Round(vyska / 25.4 * DPI, 0);
```



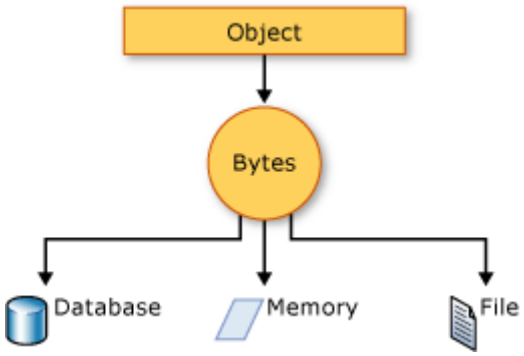
Obrázek 22 – Diagram tříd pracujících s grafickými objekty. Zdroj: vlastní

Pro vykreslení všech grafických objektů, které obsahuje grafická strana, slouží metoda *Vykreslovani*. Ta má vstupní parametr typu *Graphics,* do kterého se provede vykreslení. Pomocí příkazu *foreach* projdeme kolekci grafických objektů a vykreslíme je.

```
public void Vykreslovani(Graphics g)
{
  foreach (GrafickyObjekt item in grafickeObjekty)
  {
    item.Vykresli(g);
  }
}
```

3.9.10 Serializace a deserializace grafické strany

Pro uložení grafické strany byla využita serializace a pro načtení deserializace. Serializace je proces, pomocí kterého uložíme objekty, včetně vazeb, na další objekty do datového proudu. Pomocí serializace je možné také přenést objekty do databáze nebo operační paměti. Jeho účelem je uložit stav objektu tak, aby jej bylo možné znovu vytvořit pomocí opačného procesu deserializace. [17]



Obrázek 23 – Serializace. Zdroj: dostupný na www [17]

Ukládat objekty pomocí serializace jde do těchto formátů:

- Binární
- XML
- SOAP

Existují dva způsoby serializace, základní a vlastní. Pomocí základní serializace se automaticky serializují objekty pomocí rozhraní .NET. Při provádění základní serializace musí být třídy, které se mají serializovat, označeny atributem *Serializable* a všechny její datové členy musí být serializovatelné. Datové členy, které nechceme serializovat, označíme atributem *Nonserialized*. [17]

Při implementaci navrhovaného systému byla použita vlastní serializace, protože některé datové členy grafických objektů nebyli serializovatelné. Mezi tyto datové členy patřily třídy typu *Pen*, *Brush*, *StringFormat*. Pro použití vlastní serializace musí být třída označena atributem *Serializable* a implementovat rozhraní *ISerializable*. Rozhraní definuje metodu *GetObjectData*, která má vstupní parametry typu *SerializationInfo*

a *StreamingContext*. Data potřebná k serializaci a následně i deserializaci instance, přenášíme pomocí prvního parametru typu *SerializationInfo*. [18] [19]

```
public virtual void GetObjectData(SerializationInfo info, StreamingContext context)
```

Parametr *info* musíme naplnit kolekcí dvojic název, hodnota, pomocí přetížené metody *AddValue*. V následujícím kódu je zobrazeno uložení datového členu třídy typu *System.Drawing.Brush*, který nelze serializovat. Proto ukládáme název typu použitého štětce a dále vlastnosti použité pro konkrétní typ štětce. [18] [19]

```
if (Stetec==null)
{
  info.AddValue("StetecStyle", "null");
}
else if (Stetec.GetType().Name == "SolidBrush")
{
  info.AddValue("StetecStyle", Stetec.GetType().Name);
  info.AddValue("StetecSolidColor", ((SolidBrush)Stetec).Color);
}
else if (Stetec.GetType().Name == "HatchBrush")
{
  info.AddValue("StetecStyle", Stetec.GetType().Name);
  info.AddValue("HatchBrushHatchStyle", ((HatchBrush)Stetec).HatchStyle);
  info.AddValue("HatchBrushHatchForegroundColor",
((HatchBrush)Stetec).ForegroundColor);
  info.AddValue("HatchBrushHatchBackgroundColor",
((HatchBrush)Stetec).BackgroundColor);
}
```

Deserializace je definovaná pomocí speciálního chráněného deserializačního konstruktoru.

```
protected GrafickyObjekt(SerializationInfo info, StreamingContext context)
```

Tak jak jsme v metodě specifikovali data k serializaci, tak v deserializačním konstruktoru, který je při deserializaci vždy volán z parametru *SerializationInfo,* tato data opět vyzvedneme a provedeme potřebné navázání instančních členů pomocí metody *GetValue*. První parametr musí obsahovat název, pod kterým jsme datový člen serializovali a druhý parametr musí předat informace o skutečném typu objektu, který metoda vrací. V následujícím kódu je zobrazena deserializace datového člena třídy typu *System.Drawing.Brush*. [18] [19]

```
switch (info.GetString("StetecStyle"))
{
  case "null":
    Stetec = null;
  break;
  case "SolidBrush":
    Stetec = new SolidBrush((Color)info.GetValue("StetecSolidColor",typeof(Color)));
  break;
  case "HatchBrush":
    Stetec = new HatchBrush((HatchStyle)info.GetValue("HatchBrushHatchStyle",
typeof(HatchStyle)),
(Color)info.GetValue("HatchBrushHatchForegroundColor", typeof(Color)),
(Color)info.GetValue("HatchBrushHatchBackgroundColor", typeof(Color)));
  break;
}
```

Instanci třídy *GrafickaStrana* serializujeme pomocí metody *Serializuj*, vytvoříme datový proud *System.IO.FileStream* k souboru, do kterého hodláme serializovanou třídu uložit. Instance třídy je uložena do binárního formátu, proto je pro serializaci použita třída *Serialization.Formatters.Binary.BinaryFormatter*. Metoda *Serializuj* má vstupní parametr cestu k souboru, do kterého má být provedena serializace. [10]

```
public void Serializuj(string soubor)
{
  if (grafStrana != null)
  {
    BinaryFormatter bf = new BinaryFormatter();
    using (FileStream fs = File.Create(soubor))
    {
      bf.Serialize(fs, grafStrana);
    }
    ulozit = false;
  }
}
```

Pro deserializaci instance této třídy ze souboru je použita metoda *Deserializuj*.

3.9.11 Import a export kontaktů

Import kontaktů měl být proveden dle požadavků zadavatele do formátů XML, CSV.

Formát XML

Formát XML je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Je to velmi univerzální formát, který je dobře čitelný. Tento jazyk je především určen pro výměnu dat mezi aplikacemi a pro publikování dokumentů. Zpracování tohoto jazyka je podporováno mnoha nástroji a programovacími jazyky. XML je v podstatě založen na jednoduchém textu a lze jej zpracovat pomocí libovolného textového editoru. Pomocí XML značek (tagů) vyznačujeme v dokumentu význam jednotlivých částí textu. [20]

Pro export a import do formátů XML byla využita serializace. Pro serializaci objektů do formátu XML nám slouží třída *XmlSerializer*. Třídy, které se mají serializovat ve formátu XML, musí být veřejné a musí obsahovat konstruktor bez parametrů. Datové složky a vlastnosti, které se nemají serializovat musí být označeny atributem *XmlIgnore*. Datové složky a vlastnosti, které mají být serializované, musí být deklarovány jako veřejné. [19]

```
public class Kontakt
{
  [XmlIgnore]
  public int IdKontakt { get; set; }
  public string Email { get; set; }
  public string Firma { get; set; }
  public string Jmeno { get; set; }
  public string Prijmeni { get; set; }
  public List<string> Odebira { get; set; }
  public Kontakt()
  {Odebira = new List<string>();}
}
```

Po připravení třídy, která se má serializovat, byla vytvořena metoda *SerializaceXML*. Tato metoda ukládá objekty do datového proudu. V této metodě vytvoříme instanci třídy *XmlSerializer* pomocí konstruktoru, který má vstupní parametr typ objektu, jež se má serializovat. Kontroluje, zda tento typ splňuje podmínky XML serializace. Následně vytvoříme datový proud a pomocí metody *Serializace* zapíšeme do tohoto datového proudu serializované objekty. [19]

```
public void SerializaceXML(string nazevSouboru)
{
    List<Kontakt> kontakty = modelKontakty.GetKontakty();
    XmlSerializer xs = new XmlSerializer(typeof(List<Kontakt>));
    using (FileStream fs = new FileStream(nazevSouboru, FileMode.Create))
    {
      xs.Serialize(fs, kontakty);
    }
}
```

V následujícím zdrojovém kódu je zobrazena ukázka vytvořeného XML souboru.

```
<?xml version="1.0"?>
<ArrayOfKontakt xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <Kontakt>
    <Email>david@seznam.cz</Email>
    <Firma>Seznam.cz</Firma>
    <Jmeno>David</Jmeno>
    <Prijmeni>Procházka</Prijmeni>
    <Odebira>
      <string>Akční leták</string>
      <string>Produktový list</string>
    </Odebira>
  </Kontakt>
  <Kontakt>
    <Email>vojta@seznam.cz</Email>
    <Jmeno>Vojta</Jmeno>
    <Prijmeni>Pešl</Prijmeni>
    <Odebira>
      <string>Produktový list</string>
    </Odebira>
  </Kontakt>
</ArrayOfKontakt>
```

Deserializaci provedeme pomocí metody *Deserializace*, která vrací objekt, který načte z datového proudu.

```
List<Kontakt> kontakty = new List<Kontakt>();
XmlSerializer xd = new XmlSerializer(typeof(List<Kontakt>));
using (FileStream fs = new FileStream(nazevSoboru, FileMode.Open))
{
  kontakty = (List<Kontakt>)xd.Deserialize(fs);
}
```

Formát CSV

CSV je jednoduchý souborový formát určený pro výměnu tabulkových dat. Soubor CSV se skládá z řádků, v kterých jsou jednotlivé položky odděleny znakem čárkou (,). Tento formát je jednoduchý, nenáročný a dobře čitelný. Používá se pro výměnu informací mezi různými systémy. Slouží ke stejnému účelu jako formát XML. [21]

V navrhovaném systému je export do formátu CSV realizován pomocí metody *ExportCSV*. V této metodě provádíme zápis textového řetězce do souboru a k tomu nám slouží třída *StreamWriter*, která leží ve jmenném prostoru *System.IO*. Tato třída obsahuje metodu *WriteLine* sloužící pro zápis hodnot základních datových typů. Do konstruktoru třídy *StreamWriter,* zadáme cestu k souboru. *P*omocí výčtového typu *FileMode* nastavíme způsob otevření, pomocí *FileAccess* nastavíme možnosti přístupu k souboru. Jelikož texty a řetězce v jazyce C# jsou v paměti uloženy v určitém kódování, je nastaveno defaultní kódování. [19]

```
StreamWriter sw = new StreamWriter(File.Open(nazevSouboru, FileMode.Create, FileAccess.Write),Encoding.Default))
```

V následujících tabulkách jsou popsány konstanty výčtových typů, které specifikují režim otevření souborů.

Tabulka 6 – Výčtový typ FileMode. Zdroj: [19]

Konstanta	Popis
Read	Povoluje čtení..
Truncate	Otevření existujícího souboru a vymazání jeho obsahu.
Create	Vytvoření nového souboru. Pokud jíž existuje, bude přemazán
CreateNew	Vytvoření nového souboru. Pokud již existuje, vyvolá se výjimka *System.IO.IOException.*
OpenOrCreate	Otevření existujícího souboru. Vytvoří se nový, pokud ten neexistuje.
Append	Otevření souboru a přesunutí ukazatele na jeho konec.

Tabulka 7 – Výčtový typ FileAcces. Zdroj: [19]

Konstanta	Popis
Read	Povoluje čtení ze souboru.
Write	Povoluje zápis do souboru.
ReadWrite	Povoluje čtení i zápis z/do souboru.

Dále je v této metodě procházena kolekce všech kontaktů a každý kontakt reprezentuje jeden řádek souboru, který je následně zapsán do souboru pomocí metody *WriteLine*.

```
sw.WriteLine(radek);
```

Import z formátu CSV realizuje metoda *ImportCSV*, která má vstupní parametr cestu k souboru.

```
public string ImportCSV(string nazevSouboru)
```

V této metodě používáme pro načtení obsahu textového souboru třídu *StreamReader*, která také leží ve jmenném prostoru *System.IO*. Do konstruktoru této třídy zadáme cestu k textovému souboru, jehož obsah chceme získat.

```
StreamReader sr = new StreamReader(File.OpenRead(nazevSouboru), Encoding.Default)
```

Pro načtení jednoho řádku z datového proudu využijeme metodu *ReadLine*. Ze získaného řádku pomocí metody *Split* oddělíme jednotlivé prvky, pomocí kterých následně vytvoříme kontakt a uložíme. Textový soubor procházíme pomocí cyklu while, dokud neprojdeme celý soubor.

```
while (!sr.EndOfStream)
{
  line = sr.ReadLine();
  row = line.Split(',');
  ......
}
```

3.9.12 Export grafické strany

Export grafické strany je realizován ve třídě *SpravaGrafiky*, která je typu „Controler“ a reprezentuje řídící logiku pro práci s grafickou stranou. Metoda umožňující export grafické strany se nazývá *Exportuj*. Tato metoda nám umožňuje uložit grafickou stranu do formátů: JPEG, PNG, GIF, BMP, TIFF, PDF. Export do grafického formátu je realizován překreslením grafické strany do bitmapy, která má stejné rozměry a DPI.

```
Bitmap bmp = new Bitmap(grafStrana.Sirka, grafStrana.Vyska);
bmp.SetResolution(grafStrana.DPI, grafStrana.DPI);
using (Graphics g = Graphics.FromImage(bmp))
{
  if (inastaveni.isAntialiasing())
    g.SmoothingMode = SmoothingMode.AntiAlias;
  else
    g.SmoothingMode = SmoothingMode.None;
  g.Clear(Color.White);
  Vykreslovani(g);
}
```

Po vykreslení grafické strany využijeme metodu *Save*, pomocí které uložíme vytvořenou bitmapu do požadovaného formátu.

```
bmp.Save(fileName, ImageFormat.Jpeg);
```

Pro exportování do formátu PDF byla použita knihovna *PDFsharp*. Tato knihovna slouží pro zpracování PDF souborů. Tvorba PDF souboru funguje pomocí vykreslovacích metod, které jsou podobné jako v knihovně GDI+. Nejprve je nutné vytvořit PDF dokument, kterému můžeme nastavit informace jako je titulek, klíčová slova vlastník a mnoho dalších.

```
PdfDocument document = new PdfDocument();
document.Info.Title = "Vytvořeno Korporatním publikačním systém";
```

V následujícím kroku vytvoříme stranu PDF dokumentu, které nastavíme stejnou velikost jako velikost grafické strany. Vytvoříme objekt *XGraphics*, do kterého vykreslíme pomocí metody *DrawImage* image z vytvořené bitmapy a následně uložíme pomocí metody *Save*.

```
PdfPage page = document.AddPage();
GetPageSize(grafStrana.VelikostStrany, ref page);
XGraphics gfx = XGraphics.FromPdfPage(page);
XImage img = bmp;
gfx.DrawImage(img, new Point(0, 0));
document.Save(fileName);
```

3.9.13 Sociální síť

Sociální síť je služba na Internetu, která registrovaným členům umožňuje vytvářet osobní nebo firemní profily, na kterých mohou sdílet informace, fotografie, videa, komunikovat a mnoho dalších aktivit. Komunikace může probíhat pomocí soukromých zpráv nebo komunikací se skupinou uživatelů. Hlavní nevýhoda sociálních sítí je sdílení a možnost zneužití citlivých dat, důvěryhodnost sdělení a důvěryhodnost uživatele.

Za první sociální síť lze považovat projekt Sixdegrees, který vznikl v roce 1997. Tento projekt nebyl úspěšný a v roce 2000 zanikl. Mezi nejznámější sociální sítě roku 2013 patří Facebook, Twitter, Google +. [22]

Facebook

Facebook je rozsáhlý webový systém sloužící převážně ke komunikaci mezi uživateli, sdílení multimediálních dat a zábavě. V roce 2012 měl miliardu aktivních uživatelů a patřil mezi jednu z největších společenských sítí na světě.

Facebook byl založen v roce 2004 bývalým studentem Markem Zuckerbergem. Původně byla tato sociální síť omezena jen pro studenty univerzit, ale postupem času byla zpřístupněna všem uživatelům. Hlavní zisk společnosti Facebook je na reklamách, které se zobrazují na stránkách. Tyto reklamy jsou přesně cíleny podle zájmů uživatele a jeho chování na stránkách samotného Facebooku. [23]

Twitter

Twitter je poskytovatel sociální sítě a mikroblogu, který umožňuje uživatelům posílat a číst příspěvky zaslané jinými uživateli. Tyto zprávy mohou být dlouhé maximálně 140 znaků a zobrazují se na uživatelově profilové stránce a na stránkách jeho odběratelů. V roce 2011 měl 200 milionů uživatelů. Sociální síť Twitter byla založen v roce 2006 Jackem Dorsey. [24]

Využití sociálních sítí firmami

Přestože sociální sítě byly původně založeny aby sloužili lidem, řada firem je využívá jako komunikační kanál se svými zákazníky. Firmy přistupují k sociálním sítím různými způ-

soby. Mohou se věnovat komunikaci na sociálních sítích se svými zákazníky či potencionálními zaměstnanci. Sociální sítě dávají do rukou zákazníků firem silný nástroj na projevení svých názorů, a proto stále více firem se přinejmenším snaží citlivě monitorovat dění na sociálních sítích, které se týká jejich značky.

Další možností je použít sociální síť jako marketingový nástroj. Firmy mohou využít cílené reklamy pro své potencionální zákazníky, ale také vytvářet reklamní multimediální zprávy na firemním profilu. V dnešní době, kdy mnoho uživatelů využívá sociální sítě, je velmi důležité se zaměřit na tento silný marketingový nástroj. [22]

3.9.14 Publikování grafické strany

Navrhovaný systém umožňuje tři způsoby publikování a to prostřednictvím:

- e-mailové zprávy
- sociální sítě Facebook
- sociální sítě Twitter

O publikování se stará třída typu „Controler“ *SpravaPublikovani*, která obsahuje veškeré metody řídící publikování.

3.9.15 Publikování pomocí e-mailové zprávy

Odesílání e-mailu všem kontaktům, které mají o daný druh tiskoviny zájem, je realizováno pomocí metody *OdesliEmail*. Tato metoda má vstupní parametry předmět a text e-mailu, název přílohy a datový proud pracující s operační pamětí.

```
public void OdesliEmail(string predmet, string textEmailu, string NazevPrilohy,
String typTiskoviny, MemoryStream stream)
```

E-mailové zprávy jsou odesílány pomocí protokolu *SMTP*. Jeho úkolem je přenést poštu mezi jednotlivými počítači. Platforma .NET obsahuje pro práci s tímto protokolem třídu *SmtpClient*. Pomocí této třídy metodou *Send* lze odeslat e-mailovou zprávu. [10]

Nejprve musíme vytvořit novou zprávu pomocí třídy *MailMessage*, které nastavíme odesilatele a také příjemce pomocí třídy *MailAddress*. Konstruktor této třídy má dva vstupní parametry, e-mailovou adresu a zobrazované jméno.

```
MailAddress odesilatel = new System.Net.Mail.MailAddress(nastaveni.EmailOdesilatel,
nastaveni.NazevOdesilatel);
MailMessage message = new MailMessage();
message.From = odesilatel;
```

Všechny příjemce e-mailové zprávy získáme ze seznamu kontaktů pomocí metody *GetEmailAdress*. Procházením kolekce přidáme kontakty do instance třídy *MailAddres.*

```
List<System.Net.Mail.MailAddress> seznamEmailAdres =
kontakty.GetEmailAdress(typTiskoviny);
foreach (System.Net.Mail.MailAddress item in seznamEmailAdres)
{
  message.To.Add(item);
}
```

Následně nastavíme e-mailové zprávě předmět a tělo zprávy. Přílohu vytvoříme pomocí instance třídy *Attachment*. Do konstruktoru třídy nastavíme jako vstupní parametr datový proud, název přílohy a MIME typ přílohy.

```
message.Subject = predmet;
message.Body = textEmailu;
Attachment atach = new Attachment(stream, NazevPrilohy,
System.Net.Mime.MediaTypeNames.Application.Pdf);
```

Jako poslední krok vytvoříme instanci již zmiňované třídy *SmtpClient*, které nastavíme údaje uložené v nastavení aplikace. Jedná se o název SMTP serveru, port, povolení nebo zakázání zabezpečeného odesílání a přihlašovací údaje k SMTP serveru.

```
smtp.Host = nastaveni.SMTP;
smtp.Port = int.Parse(nastaveni.Port);
smtp.EnableSsl = nastaveni.SSL;
smtp.DeliveryMethod = SmtpDeliveryMethod.Network;
smtp.Credentials = new NetworkCredential(nastaveni.PrihlaseniJmeno,
nastaveni.PrihlaseniHeslo);
smtp.Timeout = 20000;
smtp.Send(message);
```

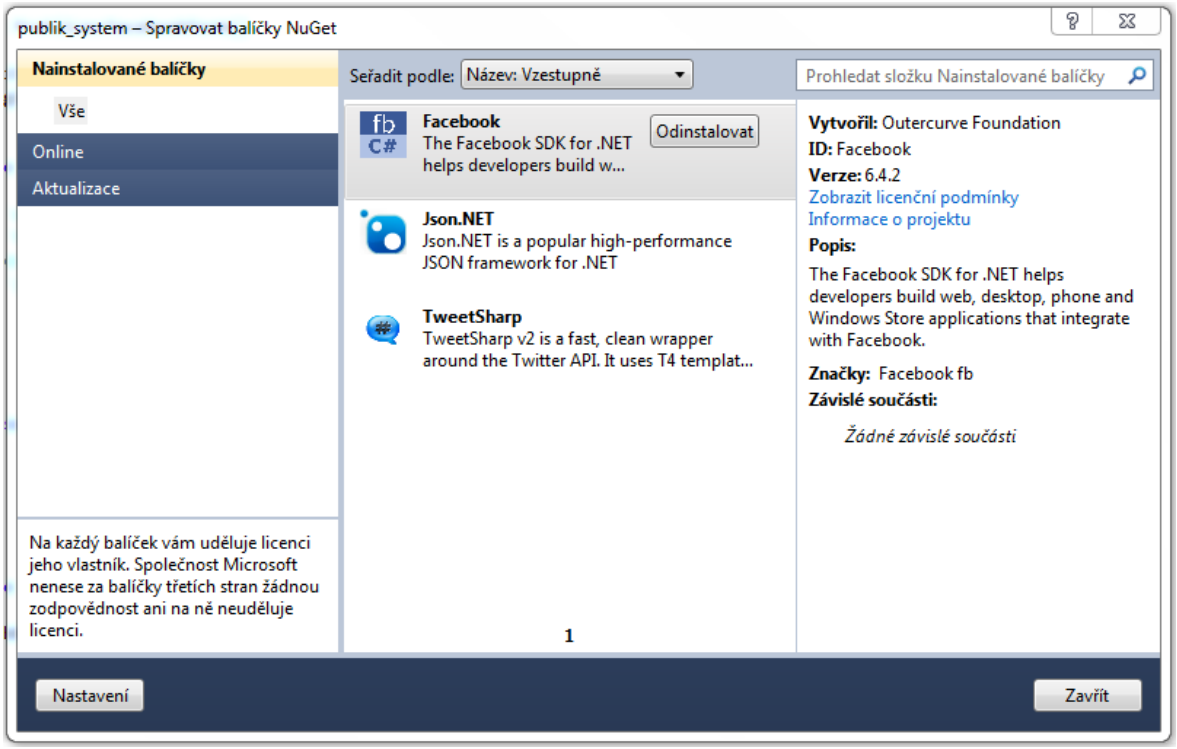
Pro uložení grafické tiskoviny do datového proudu operační paměti slouží metoda *MemoryPDF*. Metoda funguje stejným způsobem jak výše zmiňovaná metoda *Export* grafické strany, pouze s tím rozdílem, že uložíme vytvořený PDF dokument do datového proudu, který je výstupem této metody.

```
MemoryStream stream = new MemoryStream();
document.Save(stream, false);
```

3.9.16 Publikování pomocí sociální sítě Facebook

Při implementaci publikování na sociální síť Facebook byl použit balíček *Facebook SDK for .NET* verze 6.4.2. Facebook SDK for .NET umožňuje vývojářům vytvářet nejenom webové, desktopové, mobilní aplikace, ale také aplikace pro Windows 8. Projekt *Facebook SDK for .NET* je open source a využívá autorizační framework *OAuth2* a *Graph API*. [25]

Instalace balíčku *Facebook SDK for .NET* jde pouze přes nástroj *Nuget*. Nuget je nástroj rozšiřující Visual Studio. Umožňuje nám rychle a jednoduše stahovat a spravovat knihovny a nástroje v projektu Visual Studia. Pokud instalujeme balíček pomocí tohoto nástroje, automaticky zkopíruje soubory do vašeho řešení a dělá potřebné změny, jako je přidávání odkazů a změna nastavení v *app.config* nebo *Web.config*. Veškeré potřebné soubory k instalaci knihovny nebo nástroje jsou zabaleny v balíku „.nupkg“.



Obrázek 24 – Nástroj Nuget. Zdroj: vlastní

Na začátku implementace publikování na sociální síť Facebook bylo nutné vytvořit facebookovou aplikaci. Podrobný návod vytvoření aplikace je popsán v příloze.

Autentizace

V navrhovaném systému provedeme přihlášení uživatele k Facebooku pomocí *OAuth 2.0* protokolu. Tím získáme přístupový token a pomocí něj získáme dle nastavených práv další informace a možnost publikování. Postup autentizace se skládá z následujících kroků:

1. Vytvoření URL adresy pro přihlášení
2. Autorizace aplikace
3. Načtení přístupového tokenu [25]

Vytvoření URL adresy pro přihlášení

Pro vytvoření adresy pro přihlášení nám slouží metoda *FacebookGenerateLoginUrl*, která má návratovou hodnotu URI. Pomocí metody *GetLoginUrl* objektu *FacebookClient* a zadaných parametrů vytvoříme URL adresu.

Popis parametrů:

- client_id – Tímto parametrem nastavíme id aplikace. ID aplikace získáme v nastavení vytvořené facebookové aplikace.
- redirect_uri – URL adresa, kam se po přihlášení provede přesměrování.
- response_type – Pomocí tohoto parametru nastavíme jakým způsobem získáme přístupový token. Při nastavení *token* je vrácen přístupový token v URL v části frag-

ment. Toto nastavení je vhodné v desktopových aplikacích, protože přihlášení probíhá v hostovaném okně komponenty WebBrowser, kde je část fragment načtena.

- display – Určuje, jak vykreslit dialog. Přihlášení je realizováno v jiném okně, proto je uvedena hodnota *popup*.
- scope – Tímto parametrem nastavíme práva, která požadujeme, aby uživatel poskytl aplikaci. V navrhovaném systému jsou použity práva *user_about_me, publish_stream* pro možnost přidání příspěvku na facebookovou stránku a získání informací o přihlášeném uživateli. [26][25]

```
public Uri FacebookGenerateLoginUrl()
{
  dynamic parameters = new ExpandoObject();
  parameters.client_id = appId;
  parameters.redirect_uri = "https://www.facebook.com/connect/login_success.html";
  parameters.response_type = "token";
  parameters.display = "popup";
  if (!string.IsNullOrWhiteSpace(extendedPermissions))
    parameters.scope = extendedPermissions;
  return fb.GetLoginUrl(parameters);
}
```

Seznam skupin práv, které můžeme použít:

- Email Permissions – umožní načíst e-mail uživatele.
- Extended Permissions – poskytne přístup k citlivým informacím, pomocí kterých máme možnost publikovat a mazat data.
- Extended Profile Properties – poskytne přístup k informacím o uživateli a jeho přátelích.
- Open Graph Permissions – povolí přístup facebookové aplikaci publikovat pomocí API Opean Graph.
- Page Permissions – skupina oprávnění vztahující se na zprávu Facebookových stránek.
- Public Profile and Friend List – poskytne přístup k veřejným informacím o uživateli a jeho seznam přátel.

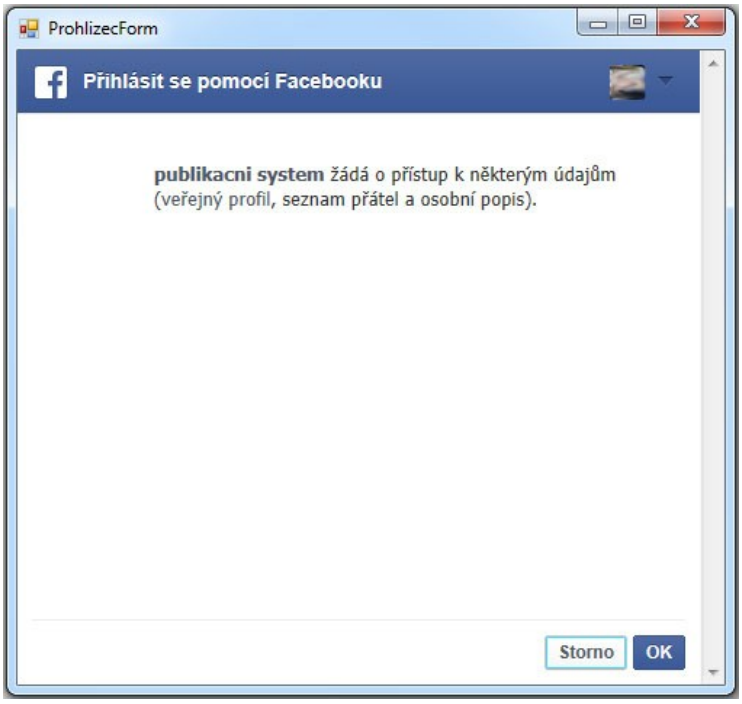
Po vytvoření URL adresy tuto adresu načteme do komponenty WebBrowser.

```
webBrowser.Navigate(loginUrl.AbsoluteUri);
```

Autorizace aplikace

Po zavolání vytvořené adresy je uživatel vyzván k přihlášení na Facebook a poté je vyzván k souhlasu s právy pomocí *OAuth 2.0* dialogu, které aplikace požaduje. Tento proces se nazývá autorizace aplikace. Pokud uživatel aplikaci schválil již v minulosti, je tento krok přeskočen. [25]

Obrázek 25 – Ověření uživatele - přihlášení. Zdroj: vlastní



Obrázek 26 – Autorizace aplikace. Zdroj: vlastní

Načtení přístupového tokenu

Při každé změně komponenty WebBrowseru kontrolujeme, zda neproběhl autorizační proces do konce. Analyzujeme vrácenou URL adresu pomocí metody *TryParseOAuthCallbackUrl*. Pokud proběhne v pořádku autentizace pomocí *OAuth 2.0*, tak vrací tato metoda hodnotu *true* a získáme přístupový token. [26]

```
FacebookOAuthResult oauthResult;
if (fb.TryParseOAuthCallbackUrl(e.Url, out oauthResult))
{
  FacebookOAuthResult = oauthResult;
  DialogResult = FacebookOAuthResult.IsSuccess ? DialogResult.OK : DialogResult.No;
}
else
{
  FacebookOAuthResult = null;
}
```

Přidání příspěvku

Získaný přístupový token je unikátní pro každé navázané spojení a je časově omezený. Pomocí objektu *FacebookClient*, kterému nastavíme získaný token a metody *Post,* můžeme odeslat příspěvek na zeď přihlášeného uživatele.

```
dynamic result = fb.Post("/me/photos", new
{
  message = zprava,
  file = new FacebookMediaStream
  {
    ContentType = "image/jpeg",
    FileName = nazevImg
  }.SetValue(imgStream)
});
```

Tuto logiku zastřešuje metoda *FacebookPostImage*, pomocí které odešleme příspěvek s obrázkem. Obrázek tvoří navržená grafická tiskovina, kterou získáme z datového proudu.

```
public bool FacebookPostImage(string zprava, string nazevImg, Stream imgStream)
```

Pro uložení grafické tiskoviny do datového proudu operační paměti slouží metoda *GetImageStream*. Metoda funguje stejným způsobem jak výše zmiňovaná metoda *Export* grafické strany, pouze s tím rozdílem, že uložíme vytvořenou bitmapu do datového proudu, který je výstupem této metody.

```
Stream stream = new MemoryStream();
bmp.Save(stream, ImageFormat.Jpeg);
```

3.9.17 Publikování pomocí sociální sítě Twitter

Při implementaci publikování na sociální síť Twitter byl použit balíček *TweetSharp* verze 2.3.1. *TweetSharp* je knihovna pracující s *Twitter API*, která zjednodušuje vytvářet desktopové, webové a mobilní aplikace, které pracují se sociální sítí Twitter. Druhá verze této knihovny je navržena tak, aby byla rychlejší a intuitivnější než originální API. Celý projekt *TweetSharp* je pod licencí Open Source.

Knihovna byla nainstalována do prostředí Visual Studia pomocí již zmiňovaného nástroje *Nuget*. Prvním krokem k použití *Twitter API* bylo nutné vytvořit aplikaci na sociální síti Twitter. Podrobný návod vytvoření aplikace je popsán v příloze. [27][28]

Autentizace

Twitter využívá autorizační framework *OAuth 2.0* k poskytnutí oprávnění používat jeho API. Twiter API v1.1 využívá dva modely autentizace:

1. Application-user authentication – Aplikace získá pomocí uživatelského a bezpečnostního klíče podepsaný request, který identifikuje aplikaci. Kromě identifikace aplikace musí koncový uživatel potvrdit oprávnění aplikace, abychom získali přístupový token. Pomocí získaného tokenu můžeme využívat API.
2. Application-only authentication – Tato forma autentizace spočívá v tom, že aplikace bez koncového uživatele může využívat API. [28]

Při implementaci byla použita metoda autentizace „Application-user authentication“. V následujících krocích je popsán postup této metody.

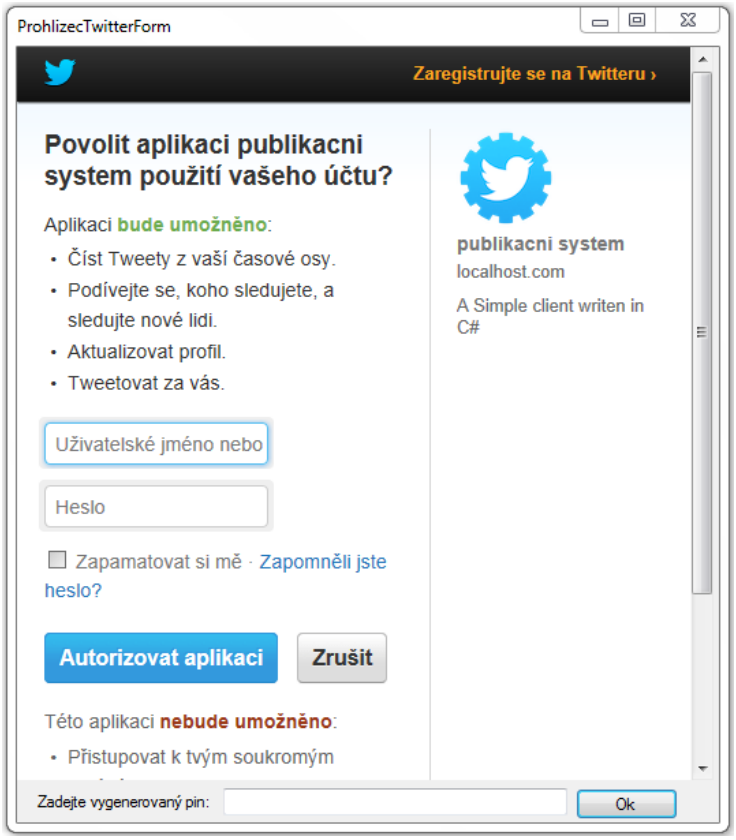
Z vytvořené aplikace získáme uživatelský klíč „*Consumer Key*“ a bezpečnostní klíč „*Consumer Secret*“. Pomocí těchto klíčů vytvoříme objekt *TwitterService*.

```
twitterClientInfo.ConsumerKey = "oPrYU16Yhg9NrlvuV0ZQ8w";
twitterClientInfo.ConsumerSecret = "ic8kaSvGok7PUfrDDJ7gvKazs30H3cgUUNUCwFTKtva";
twitterService = new TwitterService(twitterClientInfo);
```

Nyní pomocí metody *GetRequestToken* získáme podepsaný request a pomocí něj vytvoříme URL adresu, kterou předáme komponentě WebBrowser.

```
requestToken = twitterService.GetRequestToken();
return twitterService.GetAuthorizationUri(requestToken);
```

Pomocí komponenty WebBrowser a svých přihlašovacích údajů se uživatel přihlásí, přepíše vygenerovaný pin. Tímto krokem autorizuje aplikaci. [27]



Obrázek 27 – Autorizace aplikace – přihlášení. Zdroj: vlastní

Obrázek 28 – Autorizace aplikace – zadání pinu. Zdroj: vlastní

Za použití získaného pinu a metody *GetAccessToken* objektu *TwitterService* získáme přístupový token, pomocí kterého provedeme autentizaci tohoto objektu. Tuto logiku obsahuje metoda *TwitterExchangeRequestTokenForAnAccesToken*, která má jeden vstupní parametr a ten představuje získaný pin. [27]

```
public void TwitterExchangeRequestTokenForAnAccesToken(string pin)
{
  OAuthAccessToken access = twitterService.GetAccessToken(requestToken, pin);
  if (access.Token != "?" || access.Token != "?")
  {
    twitterService.AuthenticateWith(access.Token, access.TokenSecret);
    userAuthenticated = true;
  }
  else
    Hlaska.HlaskaInformation("Aplikace nebyla autorizována!");
}
```

Přidání příspěvku

Pokud proběhne autorizace aplikace v pořádku, můžeme pomocí metody *SendTweetWithMedia* odeslat příspěvek na sociální síť Twitter. Odeslání příspěvku s obrázkem nám umožňuje metoda *TwitterPostImageMessage*, která využívá výše zmíněnou metodu a obrázek uložený v datovém proudu.

```
TwitterPostImageMessage(string message, string nazevImg, Stream imageStream)
```

V následujícím kódu je zobrazeno vytvoření a následné odeslání zprávy pomocí metody *SendTweetWithMedia*.

```
SendTweetWithMediaOptions msgMedia = new SendTweetWithMediaOptions();
msgMedia.Status = message;
Dictionary<string, Stream> imageDict = new Dictionary<string, Stream> { { nazevImg,
imageStream } };
msgMedia.Images = imageDict;
twitterService.SendTweetWithMedia(msgMedia);
```

3.9.18 Popis a tvorba uživatelských formulářů

Platforma .NET Framework umožňuje vytvářet inteligentní klientské aplikace pomocí knihovny Windows Forms nebo také Presntation Foundation. Klientské aplikace pro Windows lze vyvíjet rychle a efektivně a nabízejí uživatelům bohaté možnosti.

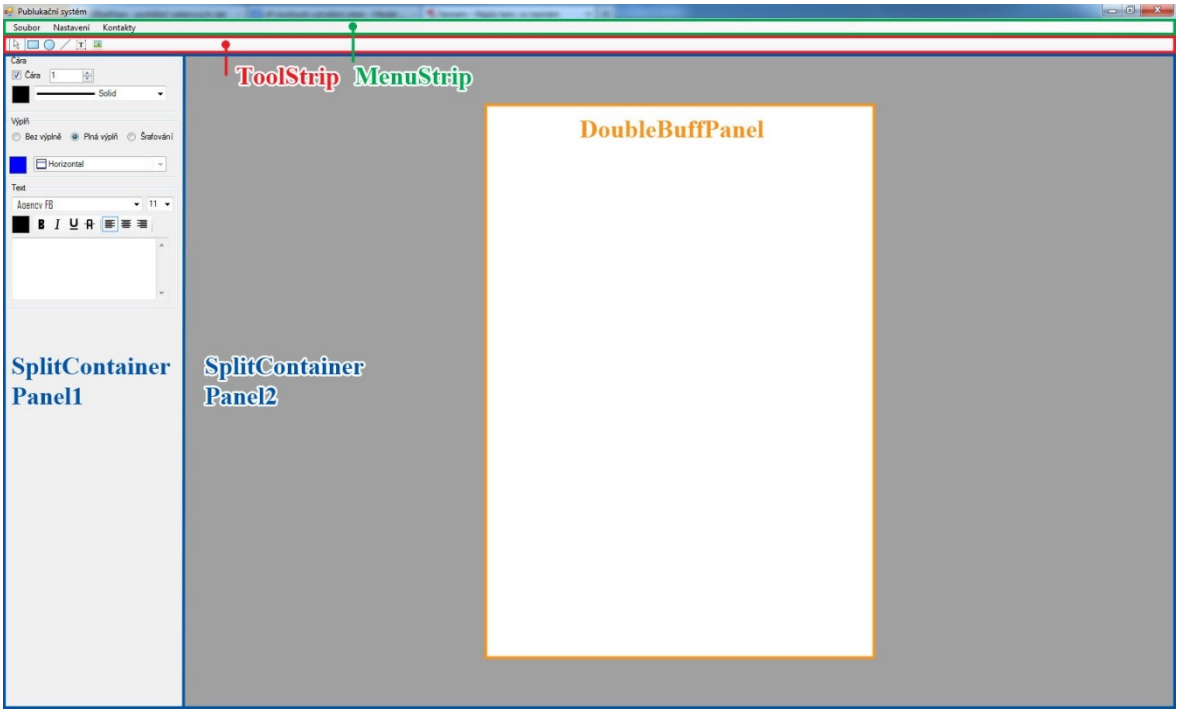
Navrhovaný systém obsahuje třídy typu „View“, které představují uživatelské grafické rozhraní mezi uživatelem a aplikací. Tyto třídy jsou odvozeny od třídy *System.Windows.Form*. [15]

Pomocí metody *InitializaceComponent*, inicializujeme všechny ovládací prvky, které byly přidány do formuláře pomocí návrháře. Tato metoda se nachází v souboru „Název formuláře“.Designer.cs. Pokud provedeme nějaké změny v okně vizuálního návrhu, změny se odrazí v metodě *InitializaceComponent*. Změnami vlastností ovládacích prvků nebo komponent v okně „Vlastnosti“ se změní také kód této metody.

Při tvorbě uživatelského rozhraní byly použity standardní ovládací prvky a komponenty. Mezi nejdůležitější patří:

- Button – reprezentuje jednoduché příkazové tlačítko.
- CheckBox – ovládací prvek, který nám umožňuje, aby uživatel zvolil mezi dvěma nebo třemi stavy.
- RadiButton – přepínač se obvykle používá ve skupinách. Pomocí přepínačů může uživatel zvolit jednu z několika možností.
- ComboBox, ListBox, CheckedList – tyto ovládací prvky jsou odvozeny od třídy ListControl. Poskytují základní funkce pro správu seznamů a práci s nimi.
- TextBox – je komponenta reprezentující textové pole. Text v této komponentě můžeme zobrazit ve více řádcích pomocí vlastnosti *MultiLine*.
- Panel – seskupuje prvky a usnadňuje nám jejich správu.
- SplitContainer – zahrnuje funkčnost tří ovládacích prvků. Obsahuje dva panely a příčku, která je odděluje.
- ToolStrip – kontejner slouží k vytváření panelů nástrojů, struktur, nabídek a stavových řádků.
- MenuStrip – ovládací prvek, který slouží jako kontejner pro strukturu nabídky aplikace.
- ContextMenuStrip – slouží k zobrazení kontextové nabídky neboli nabídky vyvolané klepnutím pravým tlačítkem myši.
- DataGridView – slouží k zobrazení dat a vytvoření vazby na třídy Array, DataTable, DataView či DataSet. [15]

Hlavní třída typu *Form* se jmenuje *MainForm* a přestavuje hlavní formulář aplikace. Uživateli se zobrazí ihned po spuštění aplikace. Skládá se z komponenty *SplitContainer*, která rozděluje formulář na dvě části. Pomocí panelu, který se nachází vlevo, můžeme nastavit vlastnosti vytvořených grafických objektů. Pravý panel slouží, jako pozadí pro vytvořenou komponentu *DoubleBuffPanel*, kterou využíváme pro vykreslování grafických objektů. Ve vrchní části formuláře se nachází ovládací prvek *MenuStrip*, který tvoří hlavní menu aplikace. Pod tímto prvkem se nachází panel nástrojů *ToolStrip*, který obsahuje jednotlivé nástroje pro kreslení grafických objektů.



Obrázek 29 – Hlavní formulář aplikace. Zdroj: vlastní

Navrhovaný systém obsahuje další vedlejší formuláře, které nám slouží k spravování dat v systému. Tyto formuláře v navrhovaném systému otevíráme jako dialogová okna. Rozlišujeme dva druhy dialogových oken, modální a nemodální.

Modální dialogová okna neumožňují dále pracovat s aplikací, dokud uživatel dialogové okno nepotvrdí nebo neuzavře.

Nemodální dialogové okno průběžně vrací řízení aplikaci a není nutné toto okno uzavřít, pokud uživatel potřebuje pracovat v jiném dialogovém okně.

V následujícím kódu je zobrazeno vytvoření dialogu modálně. Vytvoření je realizováno v hlavním formuláři aplikace, ve třídě *MainForm*.

```
private void přidatKontaktToolStripMenuItem_Click(object sender, EventArgs e)
{
  try
  {
    using (KontaktForm kontaktForm = new KontaktForm())
    {
      if (kontaktForm.ShowDialog() == DialogResult.OK)
        ikontakt.AddKontakt(kontaktForm.kontakt);
    }
   }
   catch (Exception exp)
   {
     Hlaska.HlaskaError(exp.Message);
   }
}
```

Ovládací prvek DataGridView

Ovládací prvek *DataGrid* byl součástí od první verze platformy .NET. Byl sice funkční, ale z mnoha hledisek nebyl použitelný. Neumožňoval například zobrazovat obrázky či rozevírací ovládací prvky nebo uzamknout sloupce.

Frameworku .NET verze 2.0 zavedl novou komponentu *DataGridView*, která řeší mnoho nedostatků původního ovládacího prvku a doplňuje mnoho dalších funkcí, které dříve poskytovaly pouze doplňkové produkty. Tento ovládací prvek nám umožňuje vytvářet vazbu na třídy *Array*, *DataTable*, *DataView* či *DataSet*.

V navrhovaném systému využíváme tuto komponentu k zobrazování dat uložených v tabulkách *Kontakty* a *Odebira*. Pomocí objektu typu *BindingSource* propojíme prvky uživatelského rozhraní s datovou vrstvou. Komponenta *DataGridView* získá datový zdroj z datové vrstvy a následně jej zobrazí. Mezi těmito dvěma tabulkami je vytvořená relace, které využijeme při tvorbě objektu *BindingSource*.[15]

```
dataSet.Relations.Add("KontaktyOdebiraRelace",
dataSet.Tables["Kontakty"].Columns["idKontakt"],
dataSet.Tables["Odebira"].Columns["idKontakt"]);
```

Vytvoříme jeden objekt *BindingSource* jako „Master“, který bude vázaný na tabulku Kontakty. Druhý objekt *BindingSource* bude vázaný na první vytvořený objekt a do vlastnosti *DataMember* mu bude nastavena vytvořená relace.

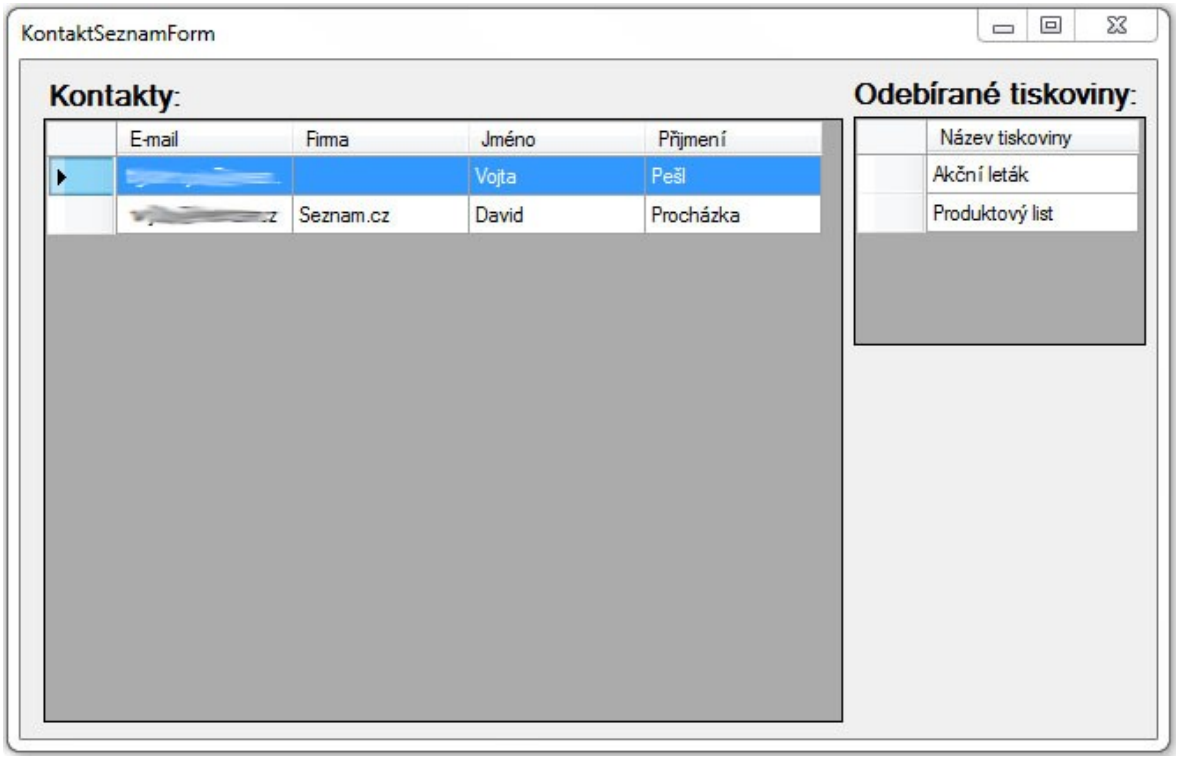
```
BindingSource bindingSourceKontakty = new BindingSource();
bindingSourceKontakty.DataSource = dataSet;
bindingSourceKontakty.DataMember = "Kontakty";

BindingSource bindingSourceOdebira = new BindingSource();
bindingSourceOdebira.DataSource = bindingSourceKontakty;
bindingSourceOdebira.DataMember = "KontaktyOdebiraRelace";
```

To zapříčiní, že komponenta *DataGridView*, která zobrazuje odebírané tiskoviny, zobrazí vždy odebírané tiskoviny aktuálně označeného řádku v komponentě *DataGridView*, která zobrazuje „Master“ tabulku Kontaktů.

Formulář, pomocí kterého zobrazujeme kontakty, obsahuje funkci, která se jmenuje *AktualizujDataGridView*. Tato funkce je volána v konstruktoru této třídy a při každé změně ve formuláři. Komponentám *DataGridView* nastavíme objekty *BindingSource*, které představují zdroj dat. Dále je nutné, abychom nezobrazovali id kontaktů a odebíraných tiskovin, proto nejsou tyto sloupce viditelné. Jsou zde také nastavené hlavičky, formátování jednotlivých sloupců a velikost buněk.

```
private void AktualizujDataGridView()
{
  BindingSource[] poleBS=ikontakt.RefreshDataSet();
  tiskovinySeznamDataGridView.DataSource = poleBS[1];
  kontaktySeznamDataGridView.DataSource=poleBS[0];
  kontaktySeznamDataGridView.Columns[0].Visible = false;
  tiskovinySeznamDataGridView.Columns[0].Visible = false;
  kontaktySeznamDataGridView.Columns[1].HeaderText = "E-mail";
  kontaktySeznamDataGridView.Columns[1].DefaultCellStyle.Alignment =
  DataGridViewContentAlignment.MiddleCenter;
  kontaktySeznamDataGridView.Columns[2].HeaderText = "Firma";
  kontaktySeznamDataGridView.Columns[3].HeaderText = "Jméno";
  kontaktySeznamDataGridView.Columns[4].HeaderText = "Přijmení";
  tiskovinySeznamDataGridView.Columns[1].HeaderText = "Název tiskoviny";
  tiskovinySeznamDataGridView.Columns[1].Width = 120;
}
```

Na následujícím obrázku je zobrazen formulář zobrazující kontakty na klienty a spolupracující firmy.



Obrázek 30 – Komponenta DataGridView. Zdroj: vlastní

Ovládání formuláře při kreslení grafických objektů

Navrhovaný systém je zaměřený na návrh grafických tiskovin, proto bylo nutné, aby reagoval na uživatelské vstupy pro kreslení na obrazovku. V navrhovaném systému provádíme kreslení na komponentu *DoubleBuffPanel*, proto překryjeme metody této třídy, které se volají z odpovídající obsluhy událostí. V následujících metodách reagujeme na situace, kdy uživatel stiskne, uvolní tlačítko myši a zaznamenáváme pohyb myši.

```
private void kresleniDoubleBuffPanel_MouseDown(object sender, MouseEventArgs e)
private void kresleniDoubleBuffPanel_MouseMove(object sender, MouseEventArgs e)
private void kresleniDoubleBuffPanel_MouseUp(object sender, MouseEventArgs e)
private void kresleniDoubleBuffPanel_MouseClick(object sender, MouseEventArgs e)
```

V těchto funkcích také musíme rozlišovat jaké tlačítko myši je stisknuto a jaký typ kreslení. V navrhovaném systému rozlišujeme tyto typy kreslení:

- Výběr – označení, posouvání, roztahování grafického objektu.
- Obdélník – kreslení obdélníků.
- Elipsa – kreslení elipsy a kruhu.
- Čára – kreslení čáry.
- Text – kreslení textu.
- Obrázek – vkládání obrázku ze souboru.

Pokud je nastaven typ kreslení „vyber“, musíme tak rozlišovat, zda grafický objekt posouváme nebo byl označen úchyt tohoto objektu a je roztahován. Proto uchováváme tyto typy ovládání grafického objektu:

- Roztahování
- Posouvání
- Nic - Režim při, kterém není ovládán grafický objekt.

Vycentrování komponenty DoubleBuffPanel

Velikost komponenty *DoubleBuffPanel* se dynamicky mění dle velikosti navrhované tiskoviny. Proto bylo nutné vytvořit metodu, která bude řídit její umístění. Pomocí metody *nastavVelikostKresleniPanel* nastavíme této komponentě velikost a nastavíme její umístění. Umístění této komponenty je vždy uprostřed panelu komponenty *SplitContainer*. Pokud velikost DoubleBuffPanelu přesáhne velikost panelu, který této komponentě tvoří pozadí, je mu nastavena velikost minimálního posouvání.

```
splitContainer1.Panel2.AutoScrollMinSize = new Size(width + 80, height + 80);

if ((splitContainer1.Panel2.Width - kresleniDoubleBuffPanel.Width) / 2 < 40)
  kresleniDoubleBuffPanel.Left = 40;
else
  kresleniDoubleBuffPanel.Left = (splitContainer1.Panel2.Width -
kresleniDoubleBuffPanel.Width) / 2;

if ((splitContainer1.Panel2.Height - kresleniDoubleBuffPanel.Height) / 2 < 40)
  kresleniDoubleBuffPanel.Top = 40;
else
  kresleniDoubleBuffPanel.Top = (splitContainer1.Panel2.Height -
kresleniDoubleBuffPanel.Height) / 2;
```

3.9.19 Tisk

Navrhovaný systém nám umožňuje tisk navržené grafické tiskoviny. Tisk je realizován pomocí třídy *Printing*. Je vytvořena instance třídy *PrintDocument*, která představuje dokument, jenž má být vytisknut. Následně povolíme obsluhu události *PrintPage* třídy *PrintDocument*. [10]

```
private void NastavitTisk()
{
  printDocument = new PrintDocument();
  printDocument.DocumentName = "publikacni_system_dokument";
  printDocument.PrintPage += new PrintPageEventHandler(printDocument_PrintPage);
}
```

Grafickou stranu vytiskneme pomocí metody *Print* instance třídy *PrintDocument*.

```
printDocument.Print();
```

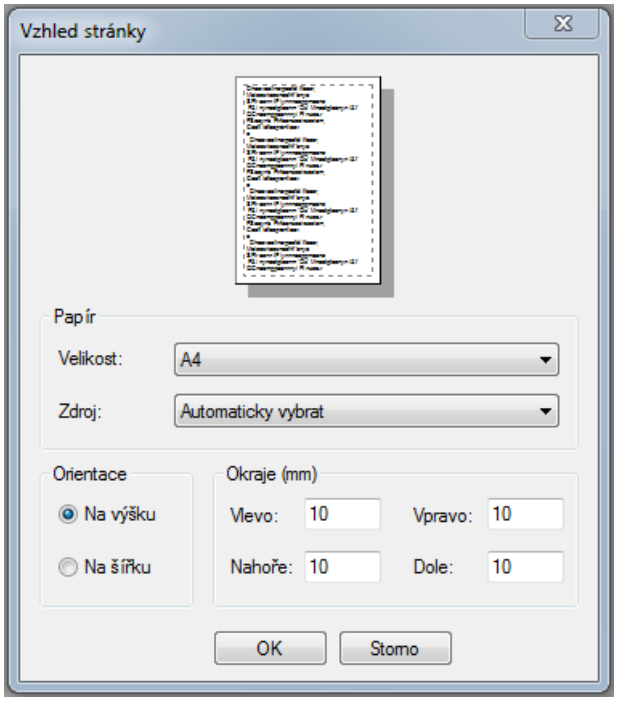
Pomocí metody *printDocument_PrintPage* definujeme grafické objekty, jež mají být vytisknuty.

Prostřednictvím třídy *PrintPreviewDialog* vytváříme dialog, který je určený pro zobrazení náhledu tisknuté strany. Po zobrazení náhledu tisknuté strany vytvoříme novou instanci třídy *PrintPreviewDialog*, která slouží jako náhled na tisknutou stranu. Do vlastnosti *Document* přiřadíme instanci třídy *PrintDocument* a zobrazíme dialog. [10]

```
using (PrintPreviewDialog dialog = new PrintPreviewDialog())
{
  dialog.Document = printDocument;
  dialog.ShowDialog();
}
```

Nastavení vzhledu stránky vyvoláme pomocí dialogu *PageSetupDialog*. Tímto dialogem můžeme nastavit vlastnosti tisknutého dokumentu, který je představován třídou *PrintDocument*. Instanci této třídy přiřadíme do vlastnosti *Document*. Dialog následně zobrazíme metodou *ShowDialog*, která vrací položku výčtového typu *DialogResult* a definuje, kterým tlačítkem uživatel dialog uzavřel. Pokud byla vrácena položka „OK“, můžeme dokument vytisknout metodou *Print*. Dialog *PageSetupDialog* je zobrazen na obrázku č. 31. [10]

```
using (PageSetupDialog dialog = new PageSetupDialog())
{
  dialog.Document = printDocument;
  DialogResult result = dialog.ShowDialog();
  if (result == DialogResult.OK)
  {
     printDocument = dialog.Document;
  }
}
```



Obrázek 31 – Zobrazení dialogu PageSetupDialog. Zdroj: vlastní

4 Testování

Testování je velmi důležitá část vývoje navrhovaného systému. K testování navrženého systému by mělo dojít nejenom při vývoji nového systému, ale i při úpravách již existujícího systému.

Testování systému probíhalo ve dvou fázích. Nejprve jsem testoval osobně a po absolvování této části přišla na řadu druhá část, a to testování aplikace v prostředí firmy se zadavatelem.

Aplikace byla spuštěna na firemním počítači, který splňoval tyto požadavky:

- Operační systém – Windows 7
- Nainstalovaný .NET Framework 4
- Internetové připojení

Testování probíhalo podle schválených scénářů, které ověřovaly, zda jsou požadované funkcionality v systému obsaženy. Při testování nebyly shledány žádné problémy s navrženým systémem. Dále byl předán systém do ověřovacího provozu, kde budou nadále testovat jeho reálné nasazení.

Závěr

Cílem diplomové práce bylo vytvořit rešerši stávajících informačních systémů vhodných pro správu a přípravu reklamních tiskovin. Dále návrh a implementaci aplikace umožňující pohodlnou správu, sestavení a následnou distribuci reklamních tiskovin.

Na navrhovaný systém byly kladeny podmínky, aby umožňoval publikovat navržené tiskoviny na sociální sítě, ale také pomocí e-mailu. Při realizaci tiskoviny měly být využity šablony, které nám zrychlí jejich návrh. V poslední řadě měl systém uchovávat kontakty na klienty a spolupracující firmy.

Při vývoji navrhovaného systému se nejprve vypracoval komplexní návrh dle metodiky UP. V ní se provedl sběr požadavků na základě konzultací se zadavatelem a firmou, kterou zadavatel využívá pro velkoplošný tisk tiskovin. Dále byly vytvořeny případy užití, které zachycují funkce, jež systém poskytuje. Také byly vytvořeny analytické a návrhové třídy a v poslední řadě byl vytvořen návrh databáze.

Na základě všech požadavků definovaných zadavatelem byl vytvořen systém, který splňuje všechny požadavky. Informační systémy pro správu a přípravu reklamních tiskovin disponují rozsáhlými možnostmi návrhu grafické tiskoviny, ale neobsahují mnoho možností distribuování tiskoviny. Navržený systém oproti ostatním systémům pro správu a přípravu reklamních tiskovin obsahuje rozsáhlé možnosti distribuování tiskovin, ale nedosahuje takových kvalit v návrhu grafické tiskoviny. Do budoucna by bylo vhodné rozšířit systém o další funkcionality pro práci s grafickými objekty a celkovým návrhem grafické tiskoviny.

Tvorba diplomové práce byla pro mě velkým přínosem. Rozšířil jsem si znalosti v oblasti platformy .NET a seznámil jsem se s novými technologiemi, jako jsou například knihovny TweetSharp a Facebook SDK for .NET, které nám zjednodušují práci se sociálními sítěmi Twitter a Facebook. Při implementaci praktické části jsem se obohatil o mnoho programátorských znalostí.

Literatura

1. Generování tiskovin. *Servis IT.* [Online] 2011. [Citace: 13. Březen 2013.] http://www.systems-it.cz/cz.

2. Microsoft Publisher 2010: snadná tvorba publikací v prostředí MS Office. *Grafika: vše o počítačové grafice.* [Online] 19. Leden 2011. [Citace: 18. Březen 2013.] http://www.grafika.cz/rubriky/software/microsoft-publisher-2010-snadna-tvorba-publikaci-v-prostredi-ms-office-138269cz.

3. **Arlow, Jim a Neustadt, Ila.** *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* [překl.] Bogdan Kiszka. Vyd. 1. Brno : Computer Press, 2007. str. 567. ISBN 978-80-251-1503-9.

4. **Kanisová, Hana a Müler, Miroslav.** *UML srozumitelně.* 2. aktualiz. vyd. Brno : Computer Press, 2012. str. 176. ISBN 978-80-251-1083-6.

5. How We Do It. *Exto Solutions: Web Applications development company.* [Online] © 2005-2012 . [Citace: 14. Srpen 2013.] http://www.extosolutions.com/how_we_do_it.php.

6. Popis aplikace Enterprise Architect. *Dataprojekt.* [Online] Dataprojekt s.r.o. [Citace: 14. Srpen 2013.] http://www.dataprojekt.cz/content/popis-aplikace-enterprise-architect.

7. Návrhový vzor. *Wikipedia: the free encyclopedia.* [Online] San Francisco (CA): Wikimedia Foundation, 2001-2013. [Citace: 29. Červenec 2013.] http://cs.wikipedia.org/wiki/N%C3%A1vrhov%C3%BD_vzor.

8. Mapping CLR Parameter Data. *MSDN: the Microsoft Developer Network.* [Online] Microsoft, © 2013. [Citace: 01. Srpen 2013.] http://msdn.microsoft.com/en-us/library/ms131092.aspx.

9. Microsoft Visual Studio. *Wikipedia: the free encyclopedia.* [Online] San Francisco (CA): Wikimedia Foundation, 2001-2013. [Citace: 10. Srpen 2013.] http://cs.wikipedia.org/wiki/Microsoft_Visual_Studio.

10. **Mareš, Amadeo.** *1001 tipů a triků pro C# 2010.* Brno : Computer Press, 2011. str. 416. ISBN 978-80-251-3250-0.

11. **sdraco.** Úvod do C# a .NET frameworku. *Devbook.* [Online] © 2013. [Citace: 01. Srpen 2013.] http://www.devbook.cz/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework.

12. **Puš, Petr.** SQL Server Compact Edition a .NET 3.5. *Vyvojar.* [Online] Devmasters s.r.o., 27. Prosinec 2007. [Citace: 31. Červenec 2013.] http://www.vyvojar.cz/Articles/576-sql-server-compact-edition-a-net-3-5.aspx.

13. **Nixon, Jerry.** SQL Express v LocalDB v SQL Compact Edition. *MSDN Blogs.* [Online] 26. Únor 2012. [Citace: 31. Červenec 2013.] http://blogs.msdn.com/b/jerrynixon/archive/2012/02/26/sql-express-v-localdb-v-sql-compact-edition.aspx.

14. **Sdraco.** Přístupy pro práci s relačními databázemi v .NET. *Devbook.* [Online] © 2013. [Citace: 30. Srpen 2013.] http://www.devbook.cz/c-sharp-tutorial-pristupy-pro-praci-s-relacni-databazi.

15. **Nagel, Christian.** *C# 2008: programujeme profesionálně.* Vyd. 1. Brno : Computer Press, 2009. str. 772. ISBN 978-80-251-2401-7.

16. **Veselý, Petr.** *Počítačová grafika 2D.* [přednáška] Pardubice : UPCE, 8.8.2013.

17. Serialization (C# and Visual Basic). *MSDN: the Microsoft Developer Network.* [Online] Microsoft, © 2013. [Citace: 05. Srpen 2013.] http://msdn.microsoft.com/cs-cz/library/ms233843.aspx.

18. **Puš, Petr.** Poznáváme C# a Microsoft.NET 41. díl: pokročilé využití serializace. *Zive.* [Online] Mladá fronta a. s., © 2013. [Citace: 05. Srpen 2013.] http://www.zive.cz/clanky/poznavame-c-a-microsoftnet-41-dil---pokrocile-vyuziti-serializace/sc-3-a-126639/default.aspx.

19. **Hřídel, Jan.** *Pokročilé techniky programování.* [přenáška] Pardubice : UPCE, 11. Srpen 2013.

20. Extensible Markup Language. *Wikipedia: the free encyclopedia.* [Online] San Francisco (CA): Wikimedia Foundation, 2001-2013. [Citace: 10. Srpen 2013.] http://cs.wikipedia.org/wiki/Extensible_Markup_Language.

21. CSV. *Wikipedia: the free encyclopedia.* [Online] San Francisco (CA): Wikimedia Foundation, 2001-2013. [Citace: 10. Srpen 2013.] http://cs.wikipedia.org/wiki/CSV.

22. Sociální síť. *Wikipedia: the free encyclopedia.* [Online] San Francisco (CA): Wikimedia Foundation, 2001-2013. [Citace: 14. Srpen 2013.] http://cs.wikipedia.org/wiki/Soci%C3%A1ln%C3%AD_s%C3%AD%C5%A5.

23. Facebook. *Wikipedia: the free encyclopedia.* [Online] San Francisco (CA): Wikimedia Foundation, 2001-2013. [Citace: 14. Srpen 2013.] http://cs.wikipedia.org/wiki/Facebook.

24. Twitter. *Wikipedia: the free encyclopedia.* [Online] San Francisco (CA): Wikimedia Foundation, 2001-2013. [Citace: 14. Srpen 2013.] http://cs.wikipedia.org/wiki/Twitter.

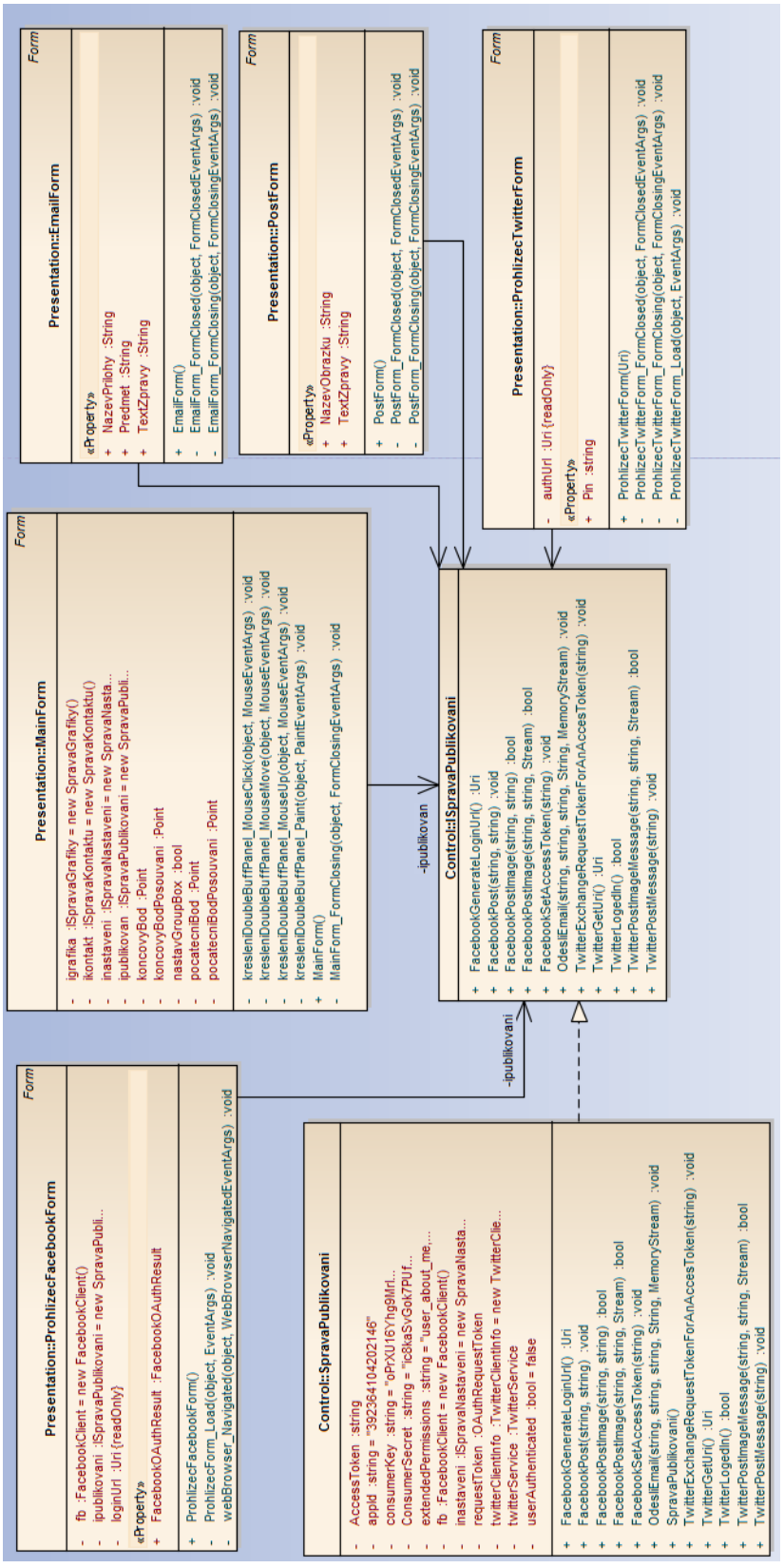
25. **Holan, Jan.** Autentizace pomoci Facebook Server Flow. *Blog: komunikace na Internetu.* [Online] IMP spol. s r.o., 09. Květen 2012. [Citace: 07. Srpen 2013.] http://blog.imp.cz/post/2012/05/09/Autentizace-pomoci-Facebook-Server-Flow.

26. **Shrestha, Prabir.** Facebook C# SDK - Writing your First Facebook Application (v6). *Prabir's Blog.* [Online] © 2011. [Citace: 07. Srpen 2013.] http://blog.prabir.me/post/Facebook-CSharp-SDK-Writing-your-First-Facebook-Application-v6.aspx.

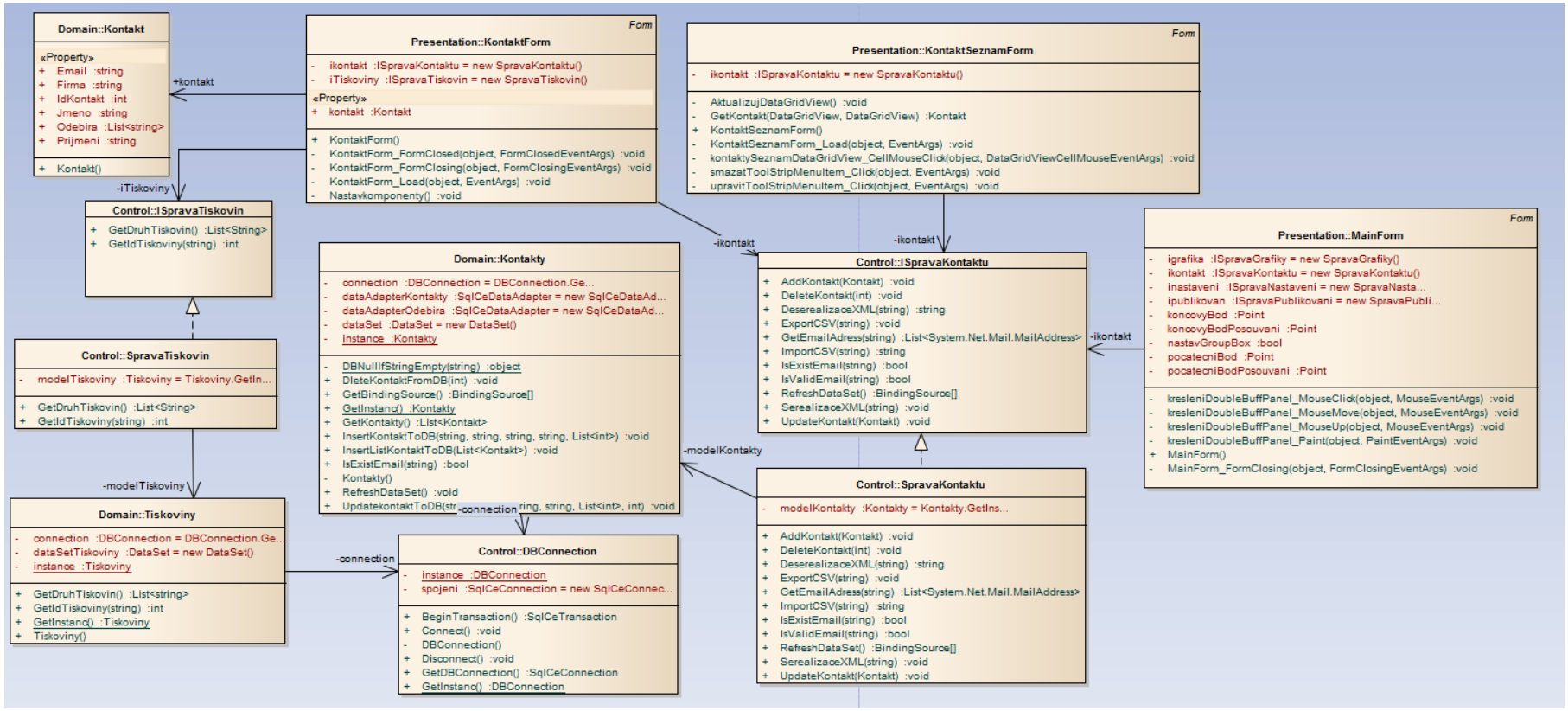
27. **Crenna, Daniel.** Tweetsharp. *GitHub.* [Online] 2013. [Citace: 08. Srpen 2013.] https://github.com/danielcrenna/tweetsharp.

28. Send secure authorized requests to the Twitter API. *Twitter Developers.* [Online] 11. Březen 2013. [Citace: 08. Srpen 2013.] https://dev.twitter.com/docs/auth/oauth.

Příloha A – Návrhové třídy

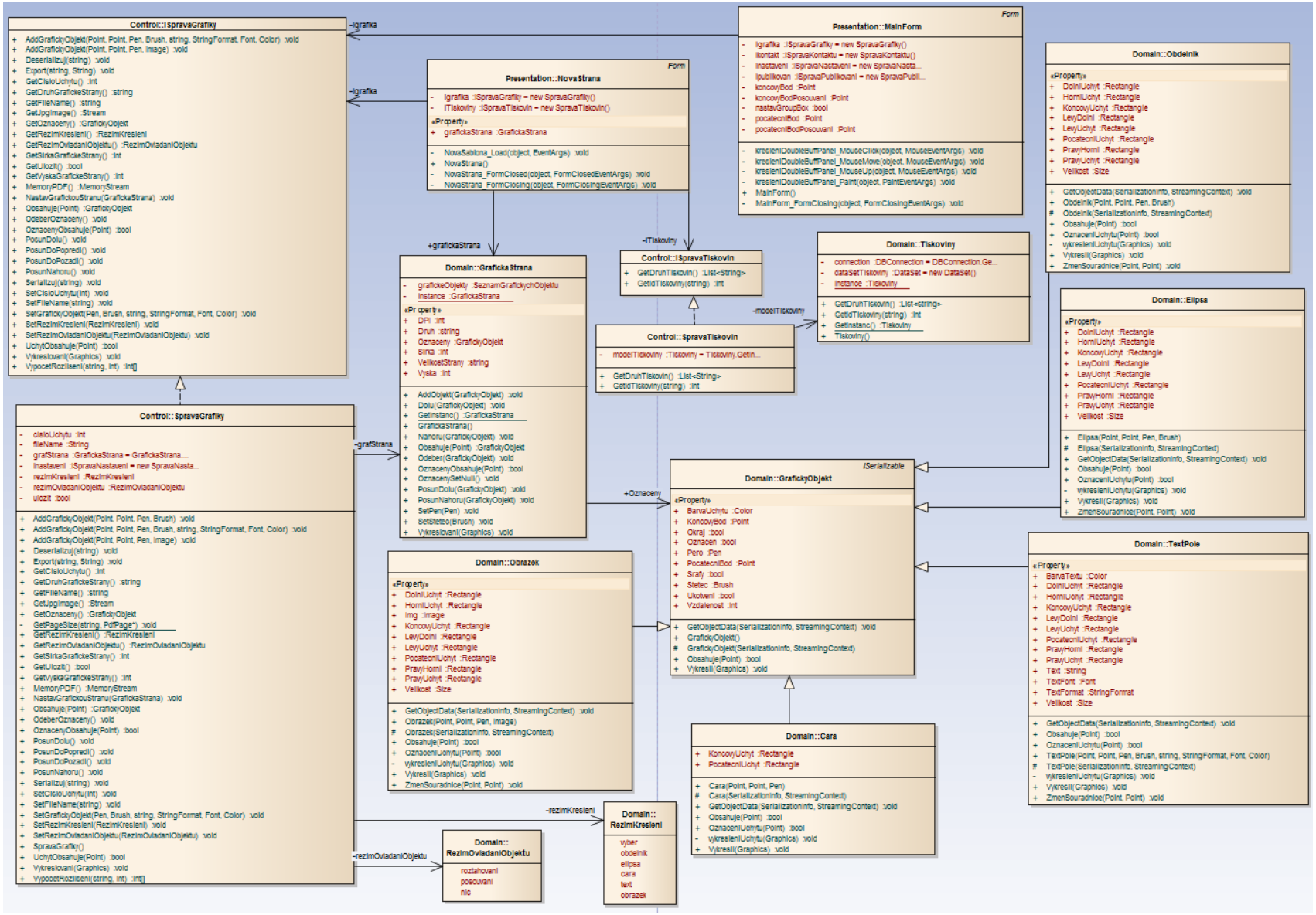


Obrázek 32 – Návrhové třídy – Publikování tiskovin. Zdroj: vlastní



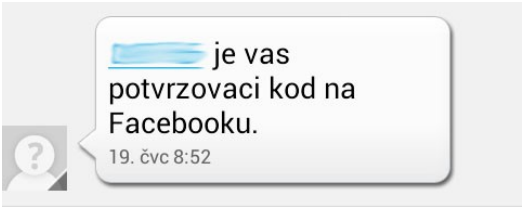
Obrázek 33 – Návrhové třídy – Správa kontaktů na klienty a firmy. Zdroj: vlastní

Obrázek 34 – Návrhové třídy – Vytvoření grafické strany. Zdroj: vlastní



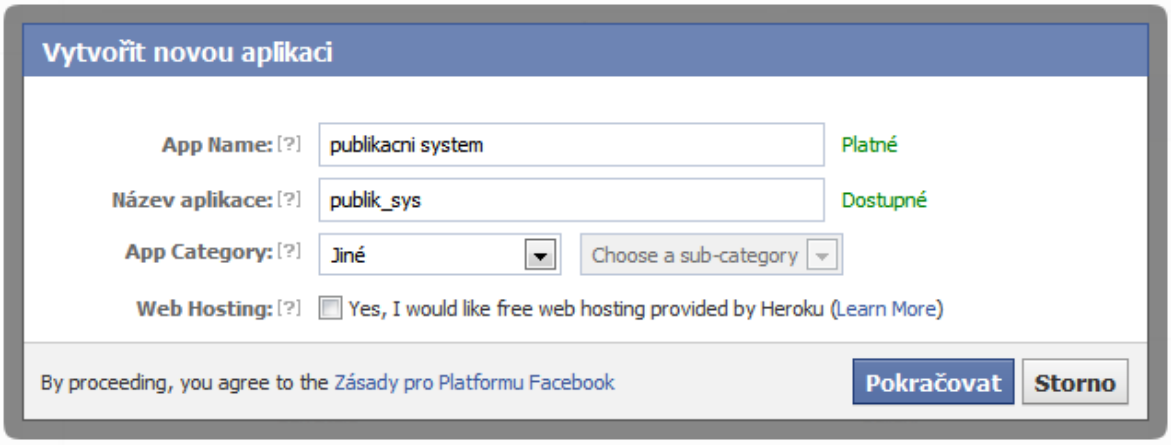
Příloha B – Postup registrace aplikace v prostředí Facebook

1. Facebookovou aplikaci lze zaregistrovat na webové stránce: https://developers.facebook.com/apps, kde se musíme přihlásit nebo zaregistrovat jako vývojář. Při registraci musí uživatel vyplnit číslo mobilního telefonu, na který mu přijde ověřovací sms.



Obrázek 35 – Ověřovací sms. Zdroj: vlastní

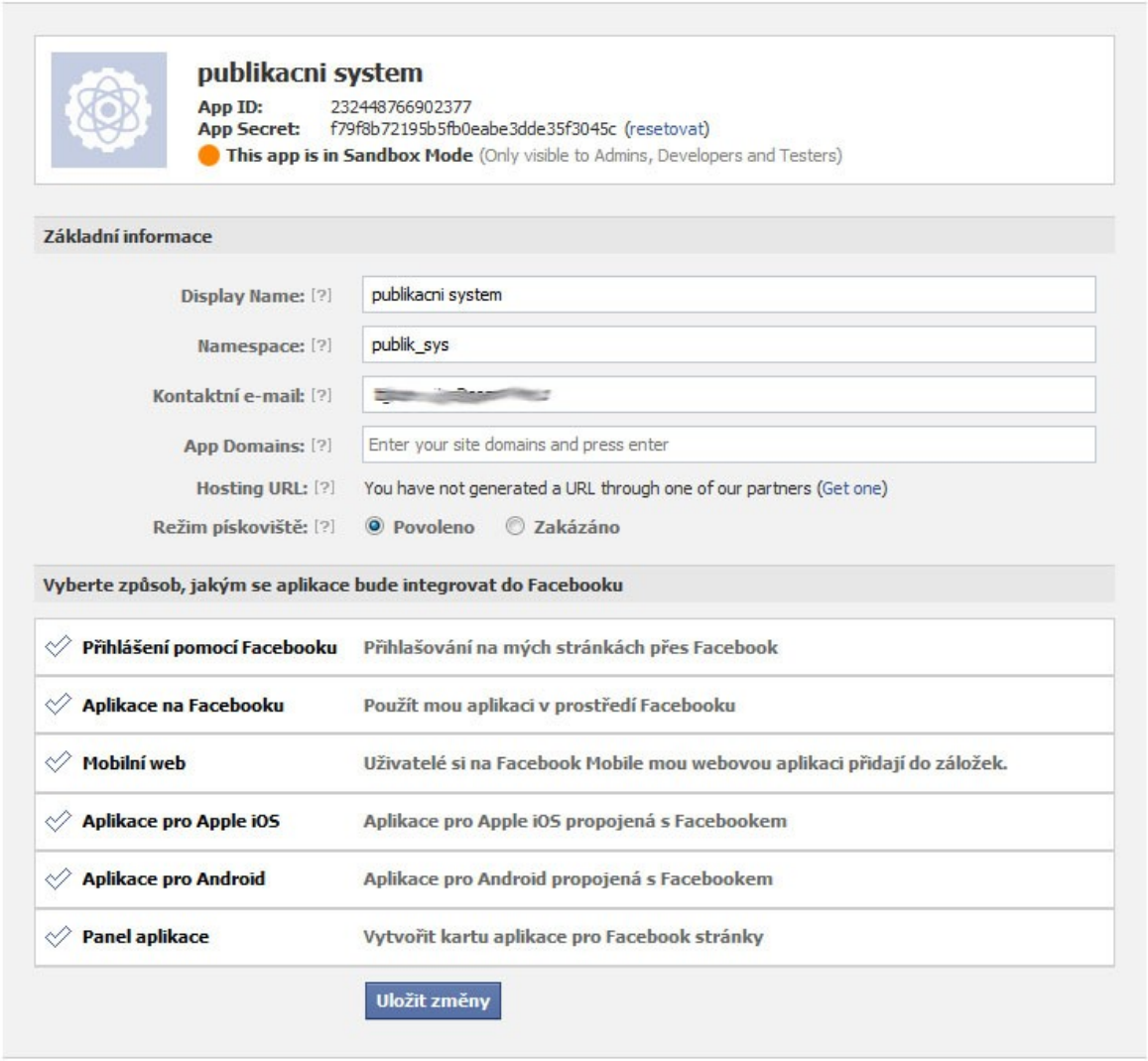
2. Nyní pomocí tlačítka „Vytvořit novou aplikaci“, vytvoříme novou aplikaci a vyplníme její název, kategorii.



Obrázek 36 – Vytvoření facebookové aplikace. Zdroj: vlastní

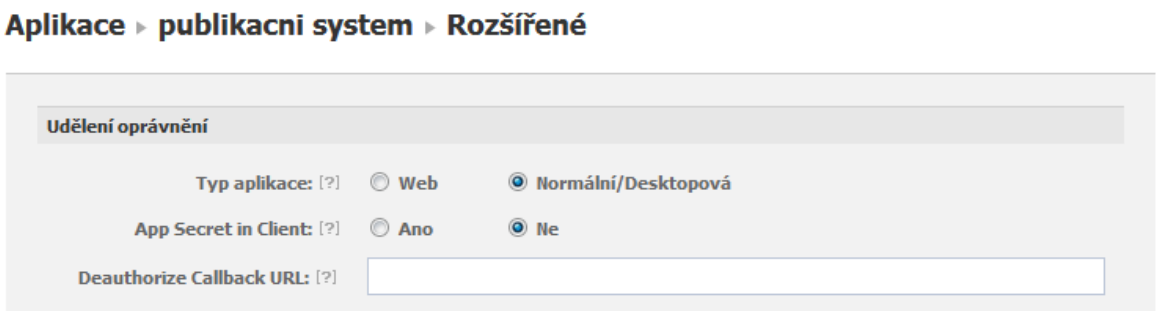
3. V dalším kroku vytvoření musíme projit bezpečností kontrolou Captcha.

4. Nyní jsme se dostali do základního nastavení aplikace. V horní části jsou zobrazeny základní informace o naši aplikaci. Velmi důležité jsou údaje ID aplikace a App Secret. Tyto údaje potřebujeme k autentizaci aplikace. Pomocí režimu pískoviště nastavíme, zda je aplikace přístupná pouze vývojářům aplikace nebo všem uživatelům. Vývojáře můžeme do vytvořené aplikace libovolně přidávat.



Obrázek 37 – Údaje o facebookové aplikace. Zdroj: vlastní

5. V rozšířeném nastavení aplikace musíme nastavit, že se jedná o desktopovou aplikaci.



Obrázek 38 – Nastavení facebokové aplikace. Zdroj: vlastní

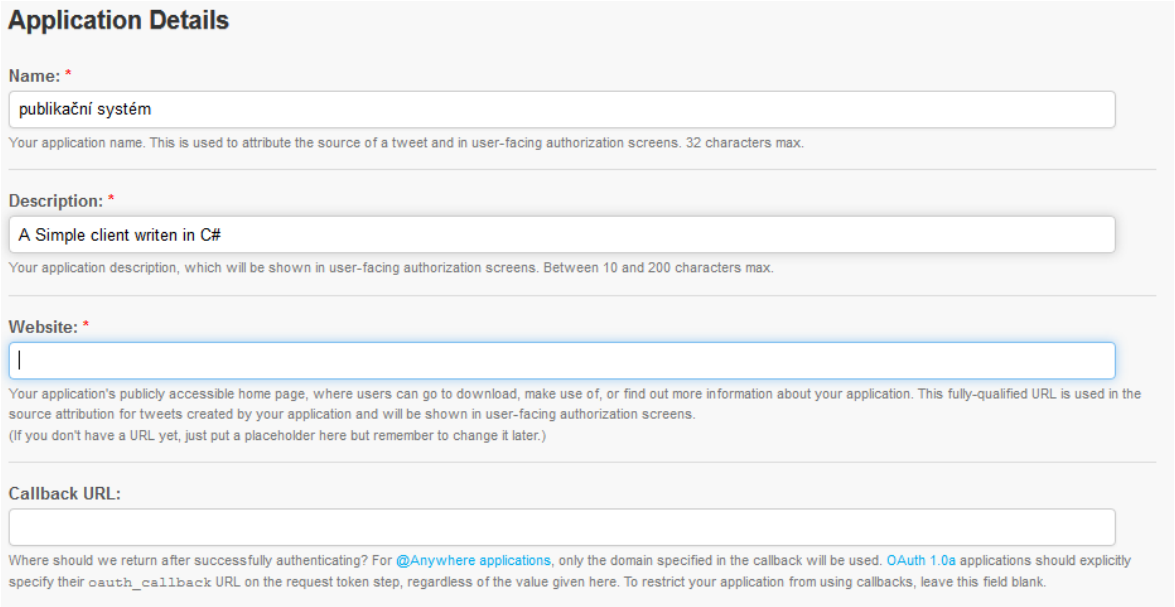
6. Nyní máme všechny potřebné věci pro fungování aplikace nastaveny a můžeme vyplnit detaily aplikace jako je popis, kategorie aplikace, ikonu aplikace, e-mail na podporu a mnoho dalších věcí.

Příloha C – Postup registrace aplikace v prostředí Twitter

1. Prvním krokem při registraci aplikace v prostředí Twitter je přejít na webové stránky http://dev.twitter.com. Zde se uživatel přihlásí pomocí účtu, který má vytvořený na sociální síti Twitter, nebo účet vytvoří.

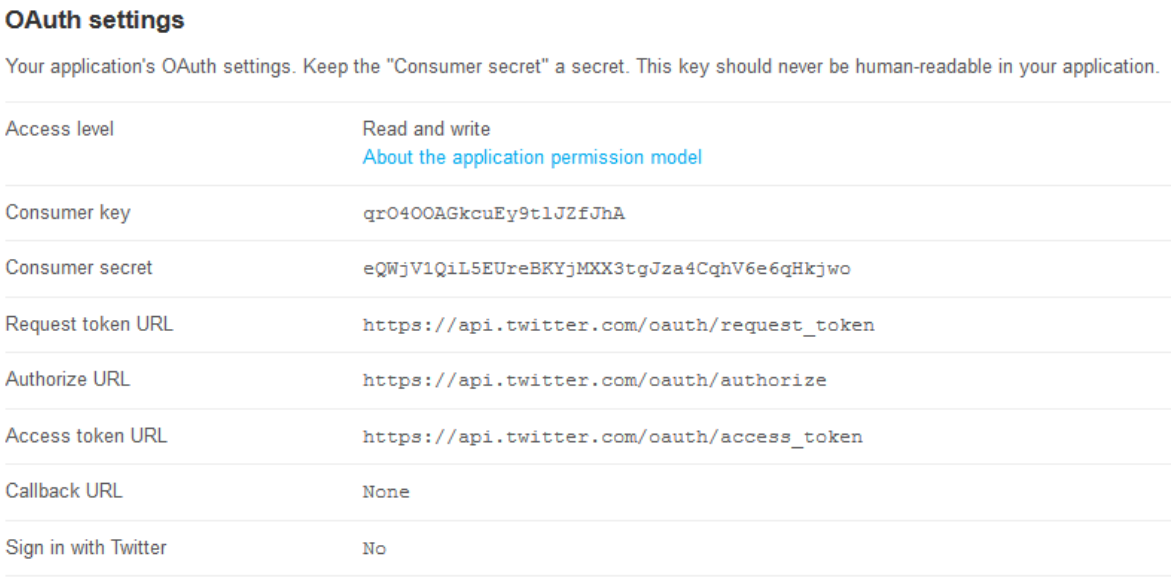
2. Po přihlášení přejdem do sekce „My applications“ a pomocí tlačítka „Create a new application“ zahájíme registraci aplikace.

3. V následujícím kroku vyplníme údaje o naší aplikaci jako je Název, popis a webové stránky naší aplikace. Potvrdíme podmínky pro používání aplikace a projdeme bezpečnostní kontrolou Captcha.



Obrázek 39 – Vytvoření aplikace na sociální síti Twitter. Zdroj: vlastní

4. Nyní se nám zobrazí základní informace o aplikaci, kde jsou velmi důležité informace Consumer key a Consumer secret, které potřebujeme pro identifikaci aplikace. Pokud vývojář potřebuje autorizovat aplikaci pouze vlastním účtem, může si pomocí tlačítka „Create my access token“ vygenerovat přístupový token.



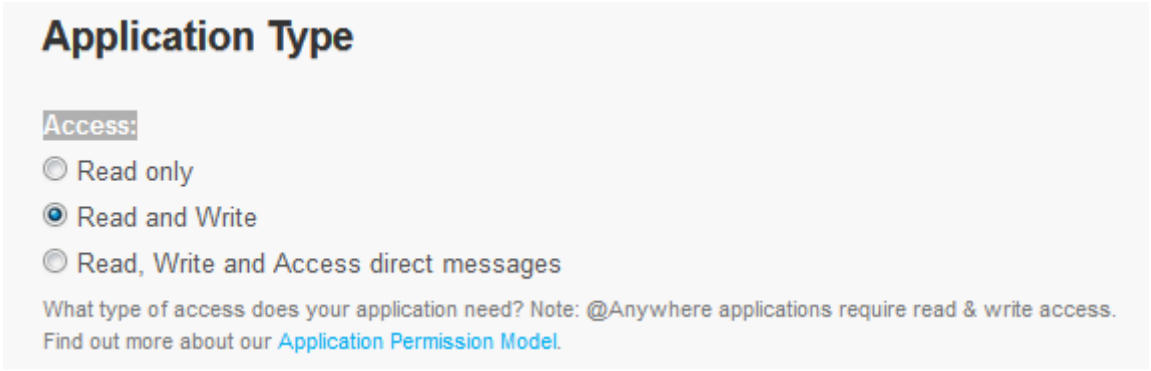
OAuth settings

Your application's OAuth settings. Keep the "Consumer secret" a secret. This key should never be human-readable in your application.

Access level	Read and write About the application permission model
Consumer key	qrO4OOAGkcuEy9tlJZfJhA
Consumer secret	eQWjV1QiL5EUreBKYjMXX3tgJza4CqhV6e6qHkjwo
Request token URL	https://api.twitter.com/oauth/request_token
Authorize URL	https://api.twitter.com/oauth/authorize
Access token URL	https://api.twitter.com/oauth/access_token
Callback URL	None
Sign in with Twitter	No

Obrázek 40 – Informace o aplikaci. Zdroj: vlastní

5. Dále je nutné v nastavení aplikace nastavit přístup, který vyžaduje navrhovaný systém. V našem případě je nutné nastavit přistup na čtení i zápis.



Application Type

Access:

- ○ Read only
- ◉ Read and Write
- ○ Read, Write and Access direct messages

What type of access does your application need? Note: @Anywhere applications require read & write access.
Find out more about our Application Permission Model.

Obrázek 41 – Nastavení aplikace. Zdroj: vlastní

6. Nyní máme všechny potřebné věci pro fungování aplikace nastavené a můžeme vyplnit detail aplikace, jako je název a webové stránky organizace, ikonu aplikace a mnoho dalšího.