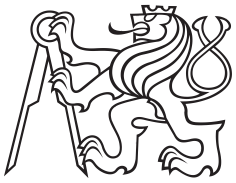
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra řízení

Platforma pro správu a sdílení projektů

Daniel Sklář

Vedoucí: Ing. Martin Ledvinka
Obor: Kybernetika a Robotika
Studijní program: Systémy a řízení
Květen 2017

Poděkování

Děkuji vedoucímu mé práce, panu Ing. Martinu Ledvinkovi, za odborné konzultace a rady, které mi při vytváření této bakalářské práce poskytnul. Také bych rád poděkoval své rodině a přátelům, kteří mi byli vždy oporou.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 26. května 2017

Abstrakt

Tato bakalářská práce pojednává o problému konzultační společnosti, kterým je nevhodné řešení správy a sdílení projektů. V práci je tato problematika analyzována, následně je navrženo řešení, jehož základ je implementován. Návrh spočívá ve flexibilním RESTovém serverovém aplikačním programovém rozhraní (API), které zprostředkovává komunikaci s různými klienty. Celá serverová část řešení je rozdělena do několika vrstev a je implementována v jazyce C# s frameworkem .NET. Práce se také zabývá zabezpečením serverového API a celkovou komunikací mezi klientem a serverem. Pro demonstrační účely a budoucí vývoj je společne s řešením navržen a implementován webový klient, na kterém se zobrazují data ze serveru. Práce staví platformu pro správu a sdílení projektů, která se bude v konzultační společnosti nasazovat a reálně používat.

Klíčová slova: API, REST, komunikace, zabezpečení, HTTP, klient/server, C#, .NET, platforma pro správa a sdílení projektů, konzultační společnost

Vedoucí: Ing. Martin Ledvinka

Abstract

This bachelor's thesis deals with problem of consulting company, which is inappropriate solution for managing and sharing projects. In the thesis, this issue is analyzed, then solution is designed and the basis of solution is implemented. The proposal lies in flexible RESTful serverside application programming interface (API) that communicates with different types of clients. The entire server part of the solution is divided into several layers and is implemented in C# with .NET framework. Work also deals with security of the serverside API and the overall communication between the client and the server. For demonstration purposes and future development, a web client together with solution is also designed and implemented to display data from the server. The work builds platform for managing and sharing projects that will be deployed and practically used in the consulting company.

Keywords: API, REST, communication, security, HTTP, client/server, C#, .NET, platform for managing and sharing projects, consulting company

Title translation: Platform for managing and sharing projects

Obsah

Obrázky

Tabulky

Kapitola 1

Úvod

Konzultační společnosti potřebují oslovovat potenciální zákazníky s nabídkou řešení jejich problémů. Jelikož je běžné, že společnost již dříve řešila podobný problém, může se na něj při kontaktu s klientem odkazovat. Prezentace výsledků vlastních zkušeností se ukázala jako efektivní způsob získávání nových nabídek. Společnost tím prokáže dostatečnou kvalifikaci danou problematiku řesit.

Doposud se pro prezentaci projektů používala individuální řešení. Způsob prezentace projektů byl na jednotlivci a jednalo se především o různou elektronickou formu či dokonce formu papírovou. Typická elektronická forma byla založena na získání dat z různých zdrojů a vytvoření prezentace, která se zaslala klientovi ve formě PDF. Tento způsob duplikoval data o projektech a zbytečně zahlcoval klienta novými soubory. Další ze způsobů byla příprava papírových prezentací. Tyto prezentace jsou náročné na přípravu a v dnešní době působí zastarale.

Další z problémů byla samotná správa projektů. Data o projektech se objevovala na více místech a spojovala se až při přípravě dané prezentace, což vytvářelo nejednotnost jednotlivých popisů projektů a působilo to zmatečně. Také sdílení mezi zaměstnanci probíhalo skrz různá řešení a působilo problematicky. Až spojením několika různých dat z různých aplikací vznikal celkový příběh projektu.

Kvůli těmto neefektivním postupům se hledalo nové řešení pro správu a sdílení projektů konzultační společnosti. Tato práce se danou problematikou zabývá. Jedním ze základních kritérií pro tuto práci je navrhnout a imple-

mentovat serverovou část řešení, která bude mít flexibilní rozhraní použitelné pro různé druhy klientských aplikací. Součást řešení je návrh a implementace webového klienta, který slouží k demonstraci možností serverové aplikace a jejímu otestování.

Flexibilní serverové rozhraní je ekonomicky výhodné, jelikož se jedná o univerzální řešení. Vysledek je takový, že při vývoji klientských aplikací již není třeba zabývat se implementací logických a datových vrstev kvůli přístupu k databázi a zpracování dat.

Součástí práce je také analýza současných řešení sdílení a správy projektů a také rozbor možné integrace s ostatními používanými systémy ve společnosti.

Kapitola 2

Motivace

Motivace pro tuto práci jsou částečně sděleny již v úvodu, ale jedná se především o důvody zadání práce. Konkrétní problémy a cíle uvádím zde, a to především s ohlédnutím na technické problémy, jelikož je tato práce hlavně o informačních technologiích a o jejich konkrétním využití v praxi.

2.1 Problémy

V této sekci zmiňuji problémy, které má práce řeší.

1. **Analýza**

   Potřebuji provést analýzu doposud používaných řešení ke sdílení a správě projektů v konzultační společnosti.

2. **Návrh serverové části aplikace**

   Musím vyřešit celkový návrh serverové části aplikace, včetně datových modelů, logických vrstev i propojení s externími klienty.

   a. **Kompatibilita**

      Je třeba vyřešit, jakým způsobem bude serverová část navržena, aby byla kompatibilní s více klientskými platformami.

   b. **Zabezpečení**

Jeden z komplexnějších problémů nastává při řešení zebezpečení. Zde je kladen důraz na vzdálenou komunikaci se serverem, při které je třeba uživatele prokazovat. Je tedy nutné vyřešit jakými prostředky bude uživatel přistupovat k datům poskytovaným serverem.

(i) **Auntentizace** - proces určující skutečnou identitu uživatele.

(ii) **Autorizace** - proces povolující uživateli určité akce.

(iii) **Role** - nejde pouze o autorizaci přihlášeného uživatele, ale také rozdělení těchto uživatelů do jednotlivých rolí.

c. **Datový model**

Je třeba navrhnout datový model, který bude splňovat základní zadání. Podle navrženého datového modelu se také budou v budoucnu integrovat data z ERP[1] systému SAP.

d. **Aplikační logika**

Problém spočívá v návrhu aplikační logiky řešení, která odpovídá za správu a sdílení projektů.

3. **Návrh webového klienta**

Webový klient slouží v práci především pro demonstrační účely možností serverového řešení a testování jeho funkčnosti. Je třeba jej tedy navrhnou a implementovat.

4. **Komunikace**

Komunikace mezi klientem a serverem je důležitá. Tato sekce je závislá na návrhu a implementaci serverové části aplikace, podle které bude vypadat také komunikace.

5. **Testování**

Musím vhodně otestovat funkčnost aplikace a demostrovat její možnosti pomocí webového klienta.

6. **Možné integrace**

Řeším zde problém možné integrace se stávajícími systémy. Jedná se o externí přihlašování skrz třetí aplikace a návrh integrace se systémem SAP, pro získávání dat.

7. **Dokumentace**

Dokumentace na úrovni programovacího jazyka, a také vhodně zvolená dokumentace pro externí systémy komunikující s mým serverovým řešením.

[1] Enterprise Resource Planning - plánování zdrojů v rámci podniku

2.2 Cíle

Hlavním cílem práce je navrhnou a implementovat balíček služeb, který se použije jako základ pro budoucí rozšiřovaný software na správu a sdílení projektů.

Součástí cíle je usnadnit konzultační společnosti a jejím zaměstnancům práci s klienty, a to především v rovině získávání nových projektů na základě prezentace stávajících.

Jedním z dílčích cílů je touto cestou unifikovat zápis projektů, zjednodušit práci s jejich propagací a ušetřit tím cenné zdroje, ať už se jedná o čas jednotlivých zaměstnanců a klientů, či o peníze, které z toho pramení.

Kapitola 3

Analýza

Jedná se o analýzu používaných řešení konzultační společnosti. Analýza obsahuje popis jednotlivých řešení, rozdělení těchto řešení do navzájem podobných celků a jejich zhodnocení. Analýza také obsahuje zvolené technologie použité v mé práci a zdůvodnění samotného postupu, a to na základě předešlých informací, získaných z komunikace s konzultační společností a také ze samotné analýzy problému.

3.1 Stávající řešení

V této části probírám popis používaných řešení pro správu a sdílení projektů v konzultační společnosti. Tyto řešení rozdělím na několik sekcí a uvedu pro to důvody.

Každá z konkrétních technologií řeší pouze část celkového problému. Pouze ve spojení těchto oblastní se obsáhne celý problém, který konzultační společnost řeší. Rozdělil jsem tedy problém do tří různých oblastí: **1. správa projektů**, **2. sdílení projektů** a **3. prezentace projektů**. V každé oblasti figuruje řada softwarových řešení a až jejím spojením dokážeme zaznamenat celkový příbeh projektu.

Oblasti tedy jsou:

1. **Správa projektů**

   Správou projektů se myslí především činnosti vytvářející či uchovávající data o projektech. Jedná se o řízení projektů, jejich dokumentaci a podobně.

   - **JIRA**
     je software pro práci s projekty. Ve společnosti se využívá především na monitorování a řízení projektů. V systému je možno používat týmové plánování, zadávání úloh, evidovat práci a řídit celkové procesy týkající se konkrétních projektů. Systém je tedy primárně navržen pro správu projektů.
   - **SAP**
     je ERP systém, česky tedy systém pro plánování zdrojů v podniku. Společnost za pomoci informačních technologií řídí a integruje všechny nebo většinu oblastí své činnosti, jako jsou plánování, zásoby, nákup, prodej, marketing, finance, personalistika a tak dále. Každý organizační útvar typicky potřebuje vlastní aplikace, což s ERP, tedy i se systémem SAP, dostane a jedna z hlavních výhod je, že tyto aplikace mezi sebou umí komunikovat a sdílet informace skrz celou organizaci. Systém SAP se ve společnosti využívá také pro uchovávání a správu dat týkajících se projektů.

2. **Sdílení projektů**

   Sdílením projektů je myšlena jejich distribuce mezi zaměstnanci. Především pro seznámení s projekty či celkovou komunikaci o projektech.

   - **SharePoint**
     je platforma sloužící primárně ke sdílení souborů. Ve společnosti se také využívá pro sdílení dat o projektech, které jsou zde uloženy. Data mohou přímo tady vznikat či být duplikována na základě dat ze systému SAP, potažmo JIRA. Ukládají se zde analýzy projektu, informační tabulky či konkrétní rešerše. Jsou to především informace o probíhajících nebo ukončených projektech.
   - **Outlook**
     je služba od společnosti Microsoft, která slouží k emailové komunikaci, vytváření kalendářů, sjednávání schůzek a posílaní menších souborů, jakožto příloh. Je to nejběžnější služba používaná ve větších firmách a korporacích, jelikož se přes ni vyřizuje veškerá formální komunikace (jak interní záležitosti, tak záležitosti externí s klienty a ostatními). Mezi zaměstnanci se tak sdílí data o projektech.
   - **Skype** pro podniky
     služba pro provozování internetových hovorů, videohovorů, instantního chatování a přenos souborů. Služba se využívá pro videohovory s klienty po celém světě a veškeré meetingy, které se z důvodu nemožnosti osobního kontaktu musí konat online. Sdílí se tak i data o projektech.

- **Slack**
  Slack je služba pro spolupráci mezi členy organizace, případně mezi účastníky stejného projektu. Jedná se o službu, kde lidé mohou komunikovat a veškeré příspěvky se zaznamenávají a jsou snadno dohledatelné. Pro práci v týmu je tato služba velice ceněná, díky své jednoduchosti a funkčnosti. Data o projektu touto cestou prochází.

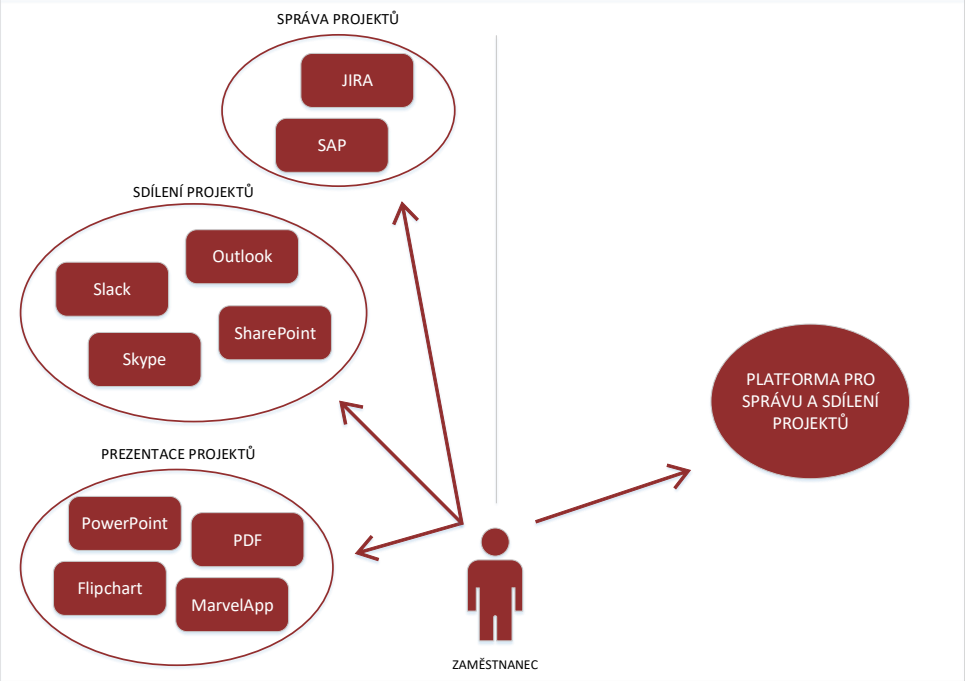
3. **Prezentace projektů**

   Technologií pro prezentaci samotných projektů používaných ve společnosti není mnoho. Prezentace probíhá spíše formou papírovou či zasíláním PDF.

   Zde se nachází zejména výčet forem prezentace projektů, než-li samotné technologie pro tuto činnost.

   - **MarvelApp**
     je služba na navrhování vzhledu projektů informačních technologií. Používá se ve společnosti především pro prezentaci budoucího vzhledu aplikací. Jedná se tedy pouze o projekty a aplikace v informačních technologiích.
   - **Papírová forma**
     je jednoduchá několikastránková, otáčecí forma prezentace, což je jeden ze způsobů používaných ve společnosti. Jedná se o byznysovou dokumentaci, která je předávána a představována potenciální klientele.
   - **PDF**
     soubor formátu PDF. Zde jde především o způsob prezentace projektu, než o samotnou technologii. Prezentuje se skrz přeposílání či představování PDF souborů, ve kterých jsou informace o projektech zpracovány.
   - **PowerPoint**
     program společnosti Microsoft pro tvorbu prezentací. I tímto způsobem mohou být prezentovány projekty klientům.

Problém je v nutnosti využívání několika odlišných řešení k dosažení cíle. Zaměstnanec je tedy zahlcen spoustou byrokracie, což způsobuje neefektivitu práce. Projekty jsou v systému uchovávány, zpracovávány a prezentovány na několika samostatných místech a až jejich spojením vzniká celkový příběh projektu. Základ řešení tohoto problému spočívá v mé práci. Při budoucím rozšíření platformy pro správu a sdílení projektů bude celý tento problém vyřešen. Tato platforma bude sjednocovat jednotlivé činnosti a uchovávat vše na jednom místě. Zaměstnanec nebude muset pracovat s několika různými technologiemi a bude moci udržovat informace o projektu pohromadě, což zefektivní jeho práci. Demonstrace momentálního rozdělení práce s projektem a jeho budoucí řešení demonstruje obrázek 3.1.



Obrázek 3.1: Zakladní rozdělení momentální práce s projekty versus budoucí řešení

3.2 Zvolené technologie

Výčet a popis technologií, které jsou pro práci použity. Většina technologií se shoduje s firemní kulturou. Jedná se především o vývojářské technologie, které rozděluji do více sekcí.

1. **Serverové technologie** - technologie používané především pro serverovou část řešení (jedná se přednostně o backend[2] celkového řešení)
2. **Klientské technologie** - technologie používané především pro klientskou část řešení (jedná se přednostně o frontend[3] celkového řešení)

Mezi technologiemi je určitý překryv. Rozdělil jsem je na následující sekce z více důvodů. Jedním z nich je potřeba popisu technologií a lepší orientace mezi nimi. Dalším důvodem je analýza firemní kultury a používání vývojových technologií ve společnosti. Posledním důvodem je rozčlenit pro práci podstatnější technologie od těch méně podstatných. Technologie použite v

[2] uživatelem neregistrovaná část softwaru
[3] uživatelem registrovaná část softwaru

serverové části řešení jsou pro mou práci významnější než technologie pro klientskou část. Jak jsem již zmiňoval, u použitých technologií je překryv, nicméně co se týče podílu jejich užití na dané sekci, tak následující rozdělení odpovídá. V serverové sekci jsou použity především serverové technologie a v klientské především klientské technologie, to však neznamená, že v serverové části nejsou použity klientské technologie a naopak.

1. **Serverové technologie**

- **C#**

  C# je objektově orientovaný programovací jazyk vyvinutý společností Microsoft. Je například významně podobný Javě, což je také objektově orientovaný jazyk.

  C# je ve společnosti upředňostňovaná technologie, což je částečně způsobeno upředňostňováním technologie i klientelou z finančního sektoru. Jeden z důvodů volby tohoto programovacího jazyka je tedy jeho upřednostňování v kultuře společnosti. S programovacím jazykem C# používám framework ASP.NET, který má detailní dokumentaci podporovanou Microsoftem. Dalším přínosem je soubor nástrojů LINQ pro dotazování se na data, což je technologie přímo integrovaná do C# jazyka a frameworku ASP.NET.[HT06]

- **ASP.NET**

  ASP.NET je open source webový framework[4] pro tvorbu moderních webových aplikací a služeb s .NET základem. ASP.NET je oblíbený framework pro tvorbu škálovatelných a interaktivních wevových stránek založených na HTML5, CSS a JavaScriptu. Používá kompilovaný kód psaný v CLR[5],jako je Visual Basic[6] nebo C#. ASP.NET je technologie od společnosti Microsoft, která má detailní podporu v dokumentaci.[Wal04]

- **Technologie spjaté s konkrétní oblastí vývoje**

  - **OAuth2**
    OAuth 2.0 je autorizační framework. Jedná se o autorizační standard dnes používaný společnostmi jako jsou Amazon, Facebook, Twitter a další.[Spa13]
  - **OWIN**
    Open Web Interface pro .NET definuje standardní rozhraní mezi .NET webovými servery a webovými aplikacemi. Cílem tohoto rozhraní je oddělit server od aplikace a podpořit tak vývoj jendoduchých modulů.
  - **OData**

[4] softwarová struktura sloužící, jako podpora při vývoji jiných softwarových projektů
[5] Common Language Runtime
[6] programovací jazyk

Open Data Protocol je webový protokol pro dotazování a aktualizování dat, který lze volně integrovat do aplikací pro přístup k datům.[Che12]

- **JSON**
  JSON je jednoduchý formát pro výměnu dat. JSON je způsob zápisu dat nezávislý na počítačové platformě. Tyto data mohou být organizována v polích nebo agregována v objektech.[Smi15]

2. Klientské technologie

- **CSS**

  Cascading Style Sheets je jazyk, který slouží jako nástroj pro ovlivňování vzhledu výsledných dokumentů, především HTML. V kaskádových stylech funguje dědičnost. CSS je tedy v zásadě neužitečné bez výsledného zobrazovacího obsahu.[Mey06]

- **Bootstrap**

  Bootstrap je framework pro rychlý a snadný webový vývoj s mobile-first[7] prioritou. Funguje na základě dvanácti sloupců, pomocí kterých definujete místo umístění elementů na stránce. Je to užitečný nástroj tvořící responzivní webdesign.

- **HTML**

  HyperText Markup Language je značkovací jazyk používaný pro tvorbu webových stránek, které jsou propojeny hypertextovými odkazy. Tímto jazykem popisujeme výslednou stránku, která se zobrazuje uživateli. Každá webová stránka vykresluje obsah pomocí HTML.

- **Razor**

  Razor je skryptovací syntax, který zjednodušuje způsob, jak vytvořit dynamický, daty poháněný web. Tato technologie je vyvinutá společností Microsoft. Razor je podobný jazyku HTML, ale mírně jej rozšiřuje. A přesto, že Razor tedy obsahuje některé nové symboly a klíčová slova, nejedná se o nový jazyk.[Cha11]

Zmíněné technologie jsou základní technologie použité v této práci. Korelují také s technologiemi využívanými v konzultační společnosti pro vývoj softwarových řešení. Jedná se tedy jak o analýzu vývojářských technologií používaných společností, tak o volbu technologií pro tuto práci.

[7] vzhled je předně uzpůsoben malým obrazovkám mobilních telefonů

3.3 Uživatelské příběhy

V této sekci popisuji uživatelské příběhy, které slouží k lepšímu definování zadání práce.

1. **Zvaní do aplikace**

   Jakožto novému uživateli, mi přichází na email pozvánka do aplikace formou časově omezené URL[8]. Po použití odkazu je nový uživatel přesměrován na server, kde si vyplní uživatelské údaje.

2. **Pozorovaný obsah**

   Jakožto uživatel s rolí klient vidím v aplikaci jiný obsah projektů než uživatel s rolí zaměstnanec, i když se mnou tyto projekty byly sdíleny.

3. **Sdílení projektu**

   Jakožto uživatel sdílím libovolný počet projektů s jiným uživatelem. Zadávám email uživatele, se kterým sdílím projekty a na základě jeho působení v aplikaci se buď uživatel dodatečně vytvoří, či se mu pouze přiřadí projekty.

4. **Vytváření uživatelů**

   Jakožto uživatel vytvářím jiného uživatele, přičemž vyplňuji novému uživateli dle potřeby jeho základní informace a volím jakou roli bude v aplikaci zastávat.

Toto jsou pouze vybrané uživatelské příběhy, které definují základní funkcionalitu aplikace.

[8] Uniform Resource Locator - řetězec znaků sloužící ke specifikaci umístění zdrojů informací

Kapitola 4

Návrh

V této sekci probírám návrh serverového řešení aplikace podle obecně uznávaných principů, dále tyto jednotlivé části návrhu popíši podrobněji a vydefinuji termíny používané později v implementační sekci 5. Také je zde obsažen návrh webové aplikace, která v projektu slouží především pro demonstraci komunikace se serverovým řešením a otestování funkčnosti. Části tohoto návrhu také procházím detailněji.

4.1 Serverové řešení

Návrh projektu je založen na třívrstvé architektuře. Jedná se o:

1. **Datovou vrstvu (DAL[9])** - zajišťuje práci s daty a komunikuje s databází (ukládání dat, výběry dat, aktualizace dat, ochrana dat a podobně)
2. **Byznysově logickou vrstvu (BLL[10])** - zajišťuje vnitřní logiku aplikace, tedy požadovanou transformaci vstupních dat na data výstupní (například výpočet průměrné ceny poskytovaných projektů v dané kategorii, transformace dat o projektech do jiných modelů a podobně)
3. **3. Aplikační programové rozhraní (API)** - vystavuje data z aplikace třetím stranám, zprostředkovává komunikaci mezi klientem a celým

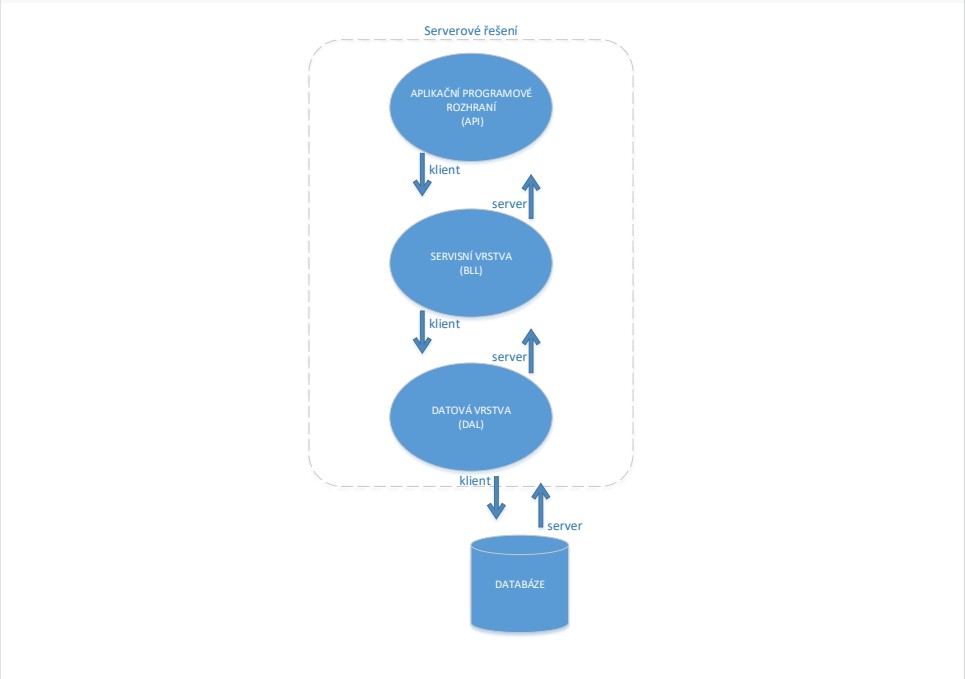
[9] Data Access Layer
[10] Business Logic Layer

serverovým řešením (komunikace s uživatelem ve zvolené formě - syrové data (JSON1), kontroluje transakce a koordinuje odpovědi aplikace)

Tyto vrstvy mezi sebou komunikují jako klient/server a využívají tedy současně tuto architekturu.

Klient/server architektura je taková, kde je aplikace rozdělena na dva nebo více kooperujících programů. Klientem je ten program, který vyžaduje provedení určité služby - funkcionality, zatímco serverem je ten, který danou službu vykoná a vrátí klientovi výsledek. Server však ve většině případů nevolá klienta, pouze odpovídá na požadavky.

Na obrázku 4.1 vidíte, jak tyto vrstvy společně komunikují a kdy a v jakých situacích jsou buďto klienty nebo servery.



Obrázek 4.1: Návrh vrstev serverového řešení

Aplikační programové rozhraní (API) komunikuje s BLL vrstvou. Podá požadavek a ten se vykoná v BLL vrstvě, která dle potřeb podává požadavek na ještě nižší úroveň, a tedy datovou DAL vrstvu. API nemá žádne reference na vrstvu datovou. Dalším důležitým faktorem je to, že v tomto rozhraní by mělo být implementováno co nejméně logiky. Logika totiž patří především do BLL vrstvy.[TJAk12]

Tyto postupy jsou zavedeny z mnoha pragmatických důvodů. Několik jich zmíním:

- Každou vrstvu lze rozvíjet a udržovat zcela samostatně. Architektura se tak stává lehce rozšiřitelná a je flexibilní změnám.

  A to především z důvodu jejich oddělené architektury. Nižší vrstva nemá žádnou referenci na vyšší vrstvu a odpovída pouze na požadavek. Jakoukoliv vrstvu můžeme libovolně nahradit za jiné funkční řešení a nemělo by to mít vliv na vrstvy ostatní a tedy celý chod projektu.

- Kód je přehlednější.

  Z této výhody mohou těžit zejména rozsáhlejší týmy.

- Je lehce testovatelná.

  Dá se snáz otestovat, jelikož má každá sekce kódu pouze jednu zodpovědnost, kterou vykonává.

Rozdělení na vícevrstvou architekturu je dnes běžná praxe, kterou doporučují přední odborníci na vývoj softwaru, jako je například Martin Fowler[11].[Fow12]

4.1.1 Datová vrstva - DAL

První z konkrétních vrstev, které popisuji je vrstva datová.

V datové vrstvě používám především dvě technologie, a to Entity Framework a Code First přístup.

- **Entity Framework**

  Jelikož je pojem Object-relation mapping, dále ORM, úzce spjat s technologií Entity Framework, definuji jej.

  Object-relational mapping je technika, při které dochází ke konverzi dat mezi nekompatibilním typovým systémem relačních databází a objektově orientovaných programovacích jazyků[Ges16]. Relační databáze je mapována na objekty. Pro každou mapovanou tabulku je vytvořena třída podle jejího schématu, která reprezentuje řádek v dané tabulce. Tato třída se označuje entita.

[11] Martin Fowler, autor mnoha knih na téma návrhu softwarových řešení

- **Výhody:** snížení nákladů na tvorbu DAL vrstvy.
- **Nevýhody:** časté a náročné dotazy do databáze, generované SQL[11] dotazy nemusí být efektivní, omezení možností databáze/SQL jazyka.

Entity Framework je ORM framework v ASP.NET1, který byl uveden v sprnu 2008. Nyní je dostupná verze 6.1.3.

- **Code First**

  Vytvářím třídy, které reprezentují objekty v doméně a z těch se vygenerují tabulky. Vlastnosti třídy definují jednotlivé sloupce tabulky a instance dané třídy jsou řádky tabulky. Je to přístup shora-dolů[12]. Nová databáze je vygenerována na základě vytvořených tříd:

  1. Vytvořím třídy reprezentující řádky tabulky (**entity**).
  2. Vytvořím kontextovou třídu reprezentující vlastní DbContext, dědící z **DbContext**4.1.1.
  3. Do vlastního DbContextu pro každou tabulku přidám vlastnost **DbSet<T>**.

  DbSet<T> je typ říkající Entity Frameworku, které entity mají být namapovány do databáze.[GWR+13]

  Generování databáze se řídí zabudovanými nebo vlastními konvencemi.

  - **Zabudované konvence:**
    - **Primary Key Convention**
      Entity Framework vyžaduje, aby tabulka měla primární klíč. Pokud se vlastnost entity jmenuje {ClassName}Id a je číselného typu nebo Guid, automaticky se primárním klíčem stane.
    - **Relationship Convention**
      Pokud v jedné entitě uvedu jako typ vlastnosti jinou třídu a přidám navigační vlastnost, automaticky se vytvoří cizí klíč.
    - **Foreign key Convention**
      Pokud vytvořím vlastnost stejného jména a typu jako je klíč v tabulce na kterou se odkazuji, automaticky se tato vlastnost použije jako ciží klíč.
    - **Type Discovery**
      Pokud v entitě uvedu jako vlastnost další třídu, pro kterou neexistuje DbSet<T> v rámci kontextu, sám se vytvoří.

[11] Structured Query Language - strukturovaný dotazovací jazyk pro práci s daty v relačních databázích

[12] vrchní vrstva je datová a spodní vrstva je samotná databáze

Také zde máme datové anotace, které lze použít k popisu metadat o vytvářených tabulkách. Používají se pokud nevyhovují zabudované konvence.

- Podporované datové anotace:
  - **[Key]**
    Určuje primární klíč.
  - **[ForeignKey()]**
    Určuje cizí klíč
  - **[Required]**
    Určuje zda je daná vlastnost vyžadována.
  - **[MaxLength()]**, **[StringLength()]**, **[MinLenght()]** a další
    Určijí maximální či minimální délku řetězce.
  - **[Index]**
    Vytvoří ukazatel se standardní hodnotou IX_<název vlastnosti> , když se databáze vytváří.
  - **[Table()]**,**[Column()]**
    Mění názvy tabulek a sloupců a další.[Mic16]

Code First podporuje v rámci vlastního iniciátoru databáze přepsání metody **Seed**, což je výhodné například pro doplnění inicializace dat.

- **DbContext**

  Je třída, jejíž hlavní zodpovědností je **správa instancí entit**. Což jsou:

  1. Vytvoření instancí entit při načtení z databáze.
  2. Zajištění správného instancování entit.
  3. Sledování změn provedených nad entitami.
  4. Uložení změn do databáze.

  Je vhodné, aby instance kontextu byla vždy uvolněna po každém dokončení transakce. V jiném případě by mohly nastat paměťové problémy. Nové instance kontextu vytvářím v konstruktoru používané třídy, či v konkrétní metodě.

- **Migrations**

  Migrations jsou důležité pro vývoj aplikace. Bez migrací by se dělaly změny v databázi obtížně a Code First přístup by byl komplikovanější.

  Code First Migrations je technologie, která umožňuje aktualizovat databázi na novou verzi a to standardně **bez ztráty dat**. Pokud by ke

ztrátě dat mělo dojít, systém mě na to upozorní. Případně mohu použít **-Force** příkaz, který umožní data v určených sloupcích smazat. Také si mohu zvolit, zda chci aktualizovat databázi ručně nebo automaticky. V projektu se vytváří výpisy verzí datového modelu, které vždy popisují migraci na vyšší i nižší verzi. Popis migrace mohu rozšířit o vlastní logiku pro vhodnou migraci dat. Použití:

1. Otevřu ve VisualStudiu[12] **Package Manager Console**[13] a zadám příkaz **enable-migrations**. Dojde k povolení migrací a vytvoření iniciálního stavu databáze/modelu.
2. Po provedení změn v modelu se ve stejné konzoli volá příkaz **add-migration "Název migrace"**.
3. Pro ruční aktualizaci databáze zavolám navíc příkaz **update-database**.

Datovou vrstvu tedy tvořím Code First přístupem s Entity Frameworkem. Hlavní výhody tohoto přístupu jsou:

1. Podporuje doménově řízený návrh (Domain Driven Design).
2. Omezuje nutnost čekání na explicitní tvorbu databáze a vyhýbá se složité implementaci přístupu k databázi.
3. Vzniká perzistenční vrstva, to znamená, že základní databáze může být změněna bez dopadu na modely.[Sin15]

Domain Driven Design je přístup k vývoji softwaru vývojáři, kteří pracují s objektovými technologiemi. V podstatě se jedná o způsob myšlení, při kterém se vývojáři snaží nastavit priority a kroky přispívající především k efektivnějšímu vývojovému procesu, a to i u vysoce komplikované domény. DDD se zaměřuje především na kvalitu popisu domény pomocí jednoznačného jazyka, který je společný vývojářům i doménovým expertům. Na základě tohoto popisu pak vzniká model domény v systému.[Eva04]

4.1.2 Byznysově logická vrstva - BLL

Byznysově logická vrstva je standardní v rámci vícevrstvé architektury. BLL vrstva je něco jako mozek celé aplikace. Všechny operace spjaté s logikou aplikace se dějí v této vrstvě. BLL umožňuje explicitní oddělení datové vrstvy

[12] programovací IDE
[13] konzole pro správu balíčků

od vrsty prezentační a nejenže vykonává všechny složité operace, ale také je mezi těmito vrstvami zprostředkovatelem toku dat.[SM07]

Ve vrstvě se nachází obvykle sada tříd, které pojmenováváme jako služby. Je dobrá konvence, aby se jednotlivé služby zabývaly pouze konkrétní oblasti práce. Pokud máme například službu pracující s projekty, měla by se dle konvencí pojmenovat *PojectService* a zabývat se pouze činností, která souvisí právě s projekty. Další z konvencí je vytvářet k těmto službám jejich vlastní rozhraní, které musí implementovat. Při daném postupu používám také **Dependency injection**[Nag16]

Dependency injection

je návrhový vzor pro předávání závislostí mezi jednotlivými komponentami aplikace tak, aby se komponenty mohly vzájemně používat.[Ars09] Používá se při tom prostředník zvaný DI kontejner, který se stará o propojení těchto komponent. Tyto komponenty se stávají nezávislými a je například možná jejich rekonfigurace bez nutnosti znovu projekt kompilovat.

Při návrhu také používám **SOLID** vzory, což jsou doporučované návrhy softwarového řešení.

- **SOLID**

  jsou základní principy tvorby objektově orientovaného jazyka[Hal14]. V názvu SOLID jich je obsaženo přesně pět:

  - **Single Responsibility Principle (SRP)**
    Každá třída má pouze jedinou zodpovědnost. Přesněji řečeno je zodpovědná pouze za řešení jednoho problému či úkolu. Pokud bude řešit problémů více, stane se tak nepřehledná.
  - **Open/Closed Principle (OCP)**
    Třída by měla být otevřená rozšířením, ale uzavřená modifikacím. Pokud tuto třídu začnou používat jiné části aplikace, neměla by se již modifikovat, ale pouze podle potřeby rozšiřovat.
  - **Liskov Substitution Principle (LSP)**
    Instance by měly být nahraditelné instancemi poděděných typů, aniž by to ovlivnilo vlastnosti programu.
  - **Interface Segregation Principle (IPS)**
    Více specificky definovaných rozhraní je lepší, než jedno obecné rozhraní. To znamená, že na každou funkci a jiné použití by mělo být implementováno rozhraní zvlášť.

- **Dependency Inversion Principle (DIP)**
  Závislost by měla být na abstrakcích, ne na implementacích. Tento princip popisujeme o pár řádku výše. Jedná se o to, aby k sobě nebyly například dvě třídy příliš svázány. V momentě, kdy se jedna třída v tomto případě změní, může se stát, že nastane chyba v třídě druhé. Tomuto se vyhneme, když místo nainstancováni konkrétni třídy nainstancujeme její rozhraní.

V momentě kdy se používá DI a předávají se závislosti skrz konstruktory, tak aby vše fungovalo, musí se třídy služeb a jejich rozhraní zaregistrovat. Pro registraci nám slouží **Unity**. Výhoda je, že třetí strana používající určitou službu nezná její konkrétní implementaci a programuje proti rozhraní. Nemusí se tedy starat o vnitřní strukturu konkrétních tříd. Používá pouze metody nadefinované v rozhraní.

Unity

Unity Application Block je jednoduchý, rozšiřitelný Dependency Injection kontejner[14], který podporuje Constructor Injection, Property Injection a Call Injection.[Mic08] Jedna ze základních výhod je ta, že zjednodušuje vytváření jiných objektů.Já Unity používám pro registraci mých služeb, abych mohl použít Constructor Injection a nainstancovat rozhraní mých služeb v konstruktorech tříd, které služby využívají, jako například API.

4.1.3 Aplikační programové rozhraní - API

Zadefinujii pojem aplikační programové rozhraní, dále v textu také API.[1]

API Aplikační programové rozhraní představuje logické rozhraní pro softwarovou komponentu a skrývá vnitřní detaily potřebné pro implementaci této komponenty. Nabízí vysokou úroveň abstrakce modulu a podporuje opakované použití kódu, což umožňuje sdílení stejných funkcí pro více aplikací, a to je jeden ze zásadních principů, proč se API vytváří. Je tedy jednotné pro více aplikací a nemusí se vytvářet pro každého klienta zvlášť řešení pracující se serverem.[Red11]

[14] objekt zajišťující interakci mezi ostatními komponentami
[1] Application user interface

Aplikační programové rozhraní je tedy jedna z vrstev našeho řešení. Je to ta vrstva, ke které přistupuje vnější okolí, tedy klienti. API zprostředkovává mimo jiné komunikaci mezi dvěma rozdílnými typy softwaru.[Uza16] Také funguje mezi komponentami, třídami a podobně. Zde především komunikaci mezi serverem, poskytujícím data a klientem, který je uživateli zobrazuje. Používáme RESTové API.

- **REST**

  Definuje sadu architektonických principů, podle kterých můžeme vytvářet datově orientovaná webová řešení. Dnes se jedná o standardní přístup při vývoji webových API.[Mac11] REST definuje jednotný a snadný přístup ke zdrojům na základě několika principů. REST popisuje jakým způsobem by měla probíhat komunikace mezi více oddělenými softwary.[FT00] Principy:

  - Doporučuje komunikační protokol HTTP.

    V dnešní době je HTTP protokol nejrozšířenější protokol pro komunikaci na webu. Toho využívá také RESTový přístup ke komunikaci.
  - Pro přístup a manipulaci se zdroji používá základní HTTP operace4.3, které reprezentují CRUD4.1.3 operace.
  - Zdroje přenáším ve formě XML[15], JSON[16] nebo v podobně uznávaném formátu.
  - Řešení by mělo být bezstavové.

    Každý požadavek musí obsahovat všechny informace nutné k jeho vykonání.
  - Každý zdroj má svou unikátní URL adresu.

  Toto jsou základní principy RESTové struktury.[WPR10]

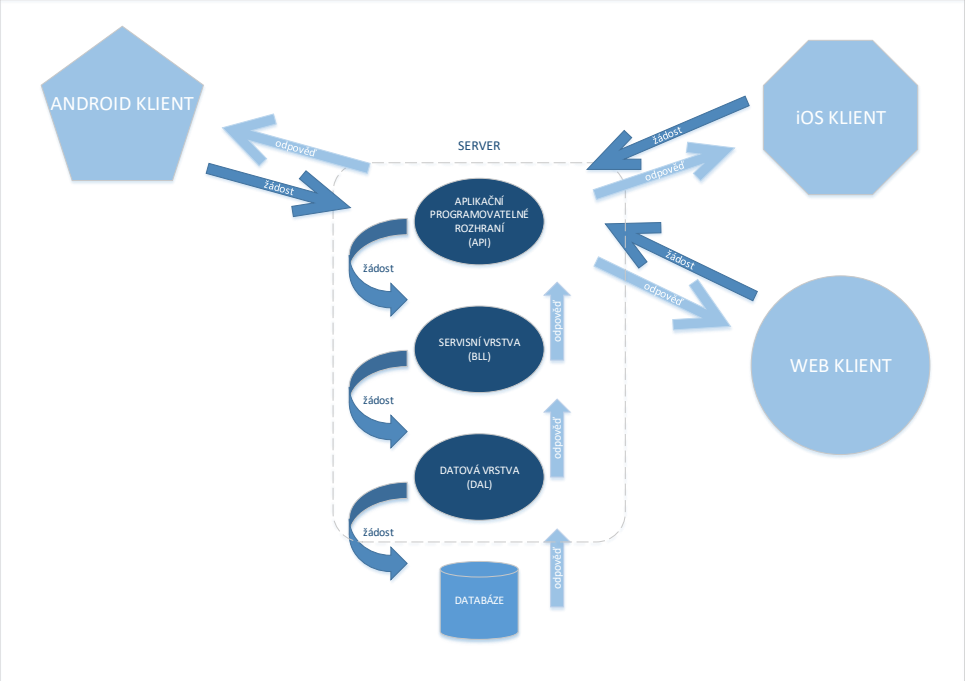
CRUD - je označení pro základní operace vykonávané při práci se zdroji. Jedná se o čtyři základní operace:

1. **Create** - vytváří data
2. **Read** - čte data
3. **Update** - aktualizuje data
4. **Delete** - maže data

S těmito čtyřmi operacemi si v základu vystačíme. [Bea09]

[15] značkovací jazyk
[16] formát zápisu dat

Na obrázku 4.2 je demonstrován základní smysl aplikačního programové rozhraní. Místo abych využíval více jednotlivých řešení pro přístup k datům, implementuji pouze jedno, které bude využíváno více klienty. Budu se tak řídit DRY[17] principem.



Obrázek 4.2: Demonstrace API a komunikace mezi vrstvami

- **Controller**

  V API se nachází především kontrolery. Kontrolery slouží pro vyřizování požadavků. Skrz kontrolery definuji možnosti, se kterými s námi může klient komunikovat. Slouží jako vnější viditelný obal aplikace, se kterým je možno komunikovat.[UZ13] Metody v kontroleru jsou takzvané akce a všechny známé metody mohu volat prostřednictvím adresy URL.

  V naší serverové části řešení používám tři různé kontrolery:

  1. API kontroler
  2. OData kontroler
  3. Běžný kontroler

- **API kontroler**

  Tento kontroler je RESTový, což znamená, že vrací surová data ve formě JSON, XML a podobně[KW14]. RESTové API ke každé odpovědi připojuje jasně definované statusové kódy.

[17] don't repeat yourself - neopakuj se

Operator	Bar	Example
Logic		
Eq	Equal	/Suppliers?$filter=Address/City eq 'Redmond'
Ne	Not equal	/Suppliers?$filter=Address/City ne 'London'
Gt	Greater than	/Products?$filter=Price gt 20
Ge	Greater than or equal	/Products?$filter=Price ge 10
Lt	Less than	/Products?$filter=Price lt 20
Le	Less than or equal	/Products?$filter=Price le 100
And	Logical and	/Products?$filter=Price le 200 and Price gt 3.5
Or	Logical or	/Products?$filter=Price le 3.5 or Price gt 200
Not	Logical negation	/Products?$filter=not endswith(Description,'milk')
Arithmetic		
Add	Addition	/Products?$filter=Price add 5 gt 10
Sub	Subtraction	/Products?$filter=Price sub 5 gt 10
Mul	Multiplication	/Products?$filter=Price mul 2 gt 2000
Div	Division	/Products?$filter=Price div 2 gt 4
Mod	Modulo	/Products?$filter=Price mod 2 eq 0
Grouping		
( )	Precedence grouping	/Products?$filter=(Price sub 5) gt 10

Tabulka 4.1: OData tabulka demonstrující možnosti filtru.

- **OData kontroler**

  OData kontroler je podobný API kontroleru v tom, že umožňuje vytvářet datové služby také založené na přístupu REST a také vrací surová data ve vhodně zvoleném formátu jako je XML či JSON. Rozdíl je však v tom, že OData kontroler má některé funkcionality navíc.[UZ13] Popisuji dvě, které jsou v aplikaci použity.

  1. **Filtrace**

     Tato filtrace funguje přímo na straně serveru, než se odpověď vrátí. Nezatěžuje tedy klienta filtrací celé množiny dat a usnadňuje mu tak práci. Příkaz k filtraci pošlu přímo v URL jako proměnnou **filter**. Filtrační příkazy fungují velice podobně jako SQL.[6] příkazy. Možnosti této filtrace lze vidět na tabulce:[ODa15]. Při použití v URL jsou mezery nahrazeny standardními znaky %20.

  2. **Nastavení**

     Jedná se o rozsáhlá nastavení, kterými OData konfiguruji. Zde například mohu v jednom místě nastavit, která data chci z určité entity v databázi ignorovat a nemusím vždy transformovat data do jiného objektu, abych se nepotřebných dat zbavili.[GHW+12]

```
public static class WebApiConfig
{
```

[6] jazyk pro manipulaci s uloženými daty v databázi

```
        public static void Register(HttpConfugiration config)
        {
              ODataConventionModelBuilder builder = new
              ODataConventionModelBuilder();
              builder.EntitySet<Project>(typeof(Project).Name)
              .EntityType.Ignore(x => x.Version);
        .
        .
        }
}
```

- Klasický kontroler

  Klasický kontroler nepoužívá API, ale webová aplikace4.1.4 stojící vedle API ano. Rozdíl od API kontroleru a OData kontroleru je takový, že klasický kontroler nevrací pouze surová data se statusovým kódem, ale vrací rovnou HTML, mnohdy s modelem, který budou vykreslovat.

4.1.4 Webová aplikace

Součástí mého serverového řešení je také webová aplikace. Tato webová aplikace slouží k zobrazování dat, které nechci zobrazovat na externím webovém klientovi, ale na straně serveru a tedy serverovém webovém klientovi. To jednak z důvodu zbytečné duplikace a pak také z důvodu nutného odkazování na URL při tvorbě uživatelů. Abych nebyl závislý na webovém klientu, na kterého bych se při tvorbě uživatele mohl odkazovat, navrhl jsem webovou aplikaci přímo na serverové straně. Webová aplikace taky slouží k dokumentaci samotného API.

Webová aplikace, která stojí vedle API využívá **Model-View-Controller** návrhový vzor.

- **MVC**

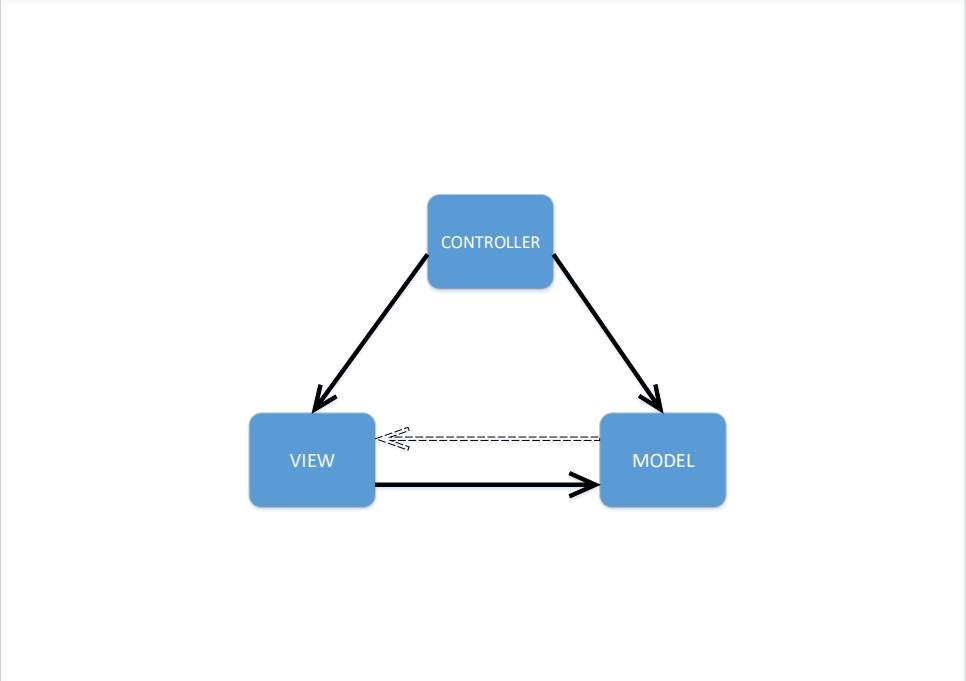
  Model-View-Controller je návrhový vzor, který rozděluje aplikaci na komponenci s oddělenými kompetencemi.[FS11] Nejedná se o několikavrstvou architekturu. Tento vzor může být použit v jedné vrstvě, tak jak je to v našem případě. Rozdělení je následující:

  - **Model**

    obsahuje nebo reprezentuje data, se kterými uživatel pracuje. Může se jednat o jednoduché zobrazovací modely, které pouze představují přenášená data mezi View a kontrolerem nebo se může jednat o modely transformací, operací či manupilací s daty. V tomto případě to lze chápat jako logiku na prezentační vrstvě.

- **View**
  je použito k vykreslení některých částí modelu.
- **Controller**
  je klasický kontroler, o kterém jsme se již bavili.4.1.3 Jedná se tedy o třídu, která zpracovává příchozí požadavky a rozhoduje co s němi bude dál. Většinou je předá modelu, který je zpracuje a podle změn se vykreslí View.

Pro demonstraci MVC interakce slouží obrázek4.3. Je na něm vidět komunikace mezi komponentami MVC návrhového vzoru. Kontroler přijme požadavek, zpracuje jej a předá ho modelu. Model většinou vykoná nějakou zadanou práci. View zaregistruje změnu v modelu a posléze data vykreslí uživateli.[Esp14]



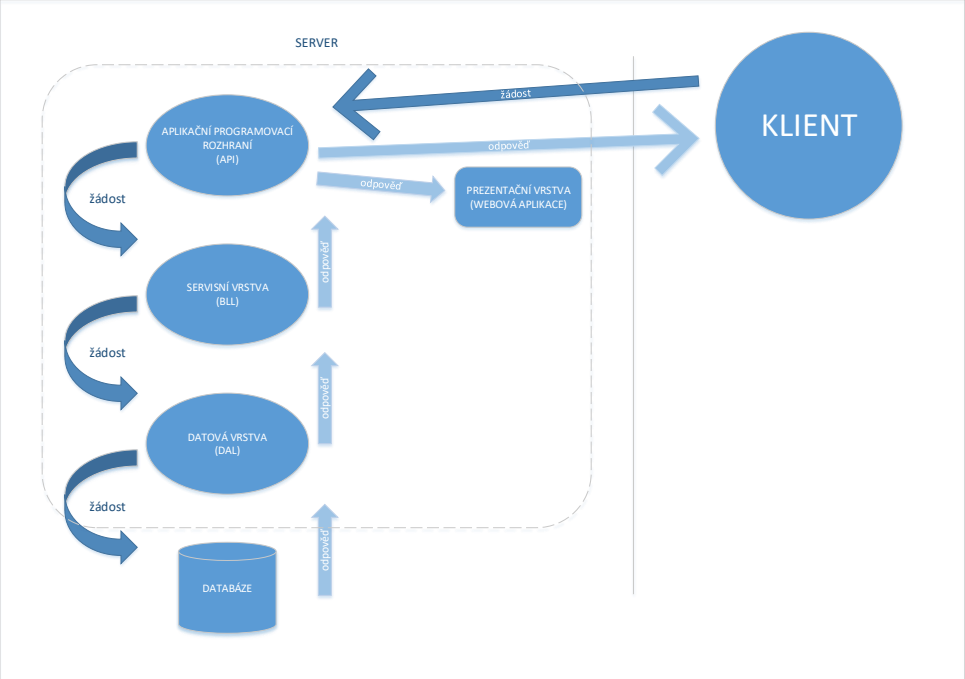
Obrázek 4.3: MVC interakce

Výhody tohoto návrhového vzoru jsou následující:

- Separovaný vývoj jednotlivých částí.
- Flexibilní změnám.
- Jednoduché testování.
- Přehlednost.

Na obrázku 4.4 je vidět jakým způsobem je serverová část řešení navržena. Je rozdělena do tří základních vrstev. **1. Datové vrstvy (DAL)**, která

zodpovídá za práci s daty přímo z databáze, poté **2. byznysově logická vrstva (BLL)**, která je zodpovědná za veškerou logiku serverového řešení a vyřizuje požadavky od třetí vrstvy. **3. Aplikační programové rozhraní (API)** slouží jako prostředník mezi požadavky třetích stran, tedy klientů a daty serveru. Součástí řešení je také webová aplikace, která slouží pro zobrazení dat.



Obrázek 4.4: Serverový návrh aplikace

4.2 Klientské řešení

V tomto konkrétním případě je klientské řešení webová aplikace, nicméně to není pravidlo a mohl bych použít například nativní aplikaci. Tato aplikace slouží především k demonstraci funkčnosti serverové části.

4.2.1 Webový klient

Tato webová aplikace je navržena stejně jako webová aplikace na straně serveru. Přesněji řečeno používá stejný návrhový vzor, a to konkrétně Model-View-Controller4.1.4. Jedna z technologií, které jsou použity jak ve webové aplikaci serverového řešení, tak ve webovém klientovi je Bootstrap.

- **Bootstrap**

  Jedná se o HTML, CSS a JS framework, který slouží pro formování vzhledu a animací aplikace. Systém používá columns, což jsou v češtině sloupce. Každá obrazovka se dělí na dvanáct sloupců a programátor může pomocí těchto sloupců navrhovat stránku. Volit, ve kterém sloupci se bude nacházet určitý element, jakým způsobem se bude zalamovat a také v jakém případě. Bootstrap upřednostňuje Mobile First[18] vzhled a celek je po používání Bootstrapu responzivní[19].[Spu13]

  Většina komponent je v Bootstrapu předem nadefinována, proto je technologie snadno použitelná a díky jejímu responzivnímu vzhledu také šetří vývojářův čas s rozdělováním vzhledu na určité velikosti obrazovek. V této verzi jsem ke knihovně Bootstrapu přidal CSS komponenty, kvůli požadavkům na specifický vzhled.

4.3 Komunikace

V této kapitole se také řeší návrh komunikace mezi klientem a serverovým API. Veškerá komunikace mezi API a klientem je zprostředkovávána pomocí protokolu HTTP.

- **HTTP**

  HTTP je protokol, který původně sloužil pro zobrazování HTML stránek. V dnešní době také slouží pro budování webových API pomocí HTTP požadavků,identifikátorů URI a hlaviček.[UZK13]

  URI

  Uniform Resource Identifier je jednotný identifikátor zdrojů. To znamená, že pomocí něj dokážete jasně identifikovat zdroj. Je to jeden ze základních principů RESTového API.

  - **HTTP požadavek**

    je forma, jakou žádáme od serveru data. Nejpoužívanější čtyři formy jsou:[GT02]

    1. **GET**

       Tento požadavek slouží k získání dat na určité URL. Jedná se o nejjednodušší HTTP žádost, která pouze vrací data. URL s GET požadavkem se dá zachytit a uložit. Což v určitých případech nechceme.

[18] při vývoji vzhledu je předně myšleno na mobilní zařízení
[19] vzhled stránky se mění v závislosti na velikosti obrazovky

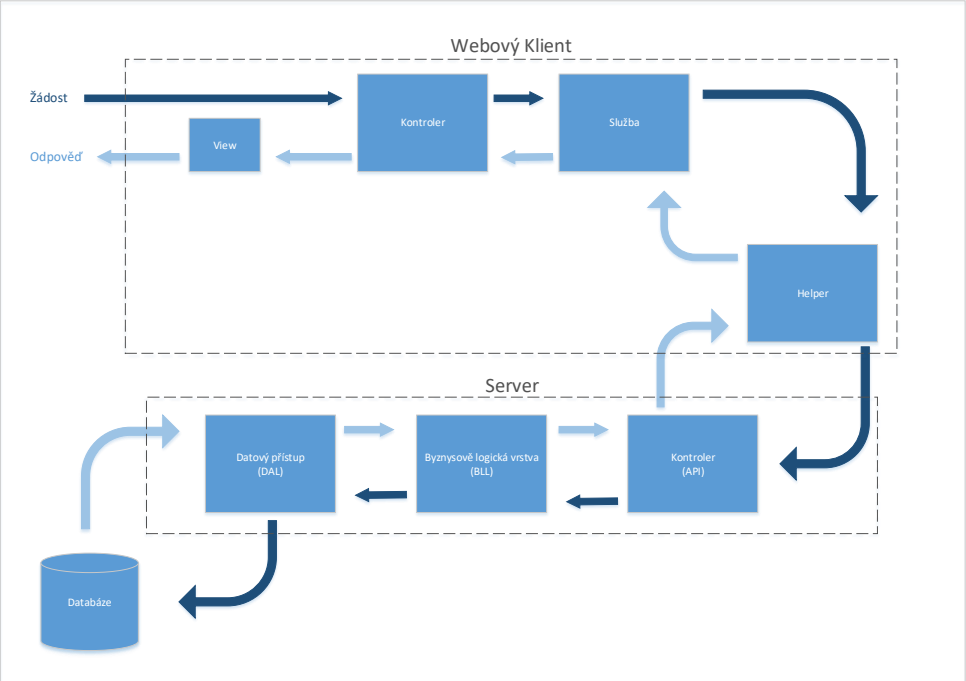
2. **POST**
   Narozdíl od GET požadavku požadavek POST data přenáší a to za účelem získání jiných dat. Tento požadavek se klasicky nedá uložit. Přes POST požadavek se většinou posílá formulář s daty, která mohou být citlivá.
3. **PUT**
   Tento požadavek slouží ke změně dat na serveru.
4. **DELETE**
   Slouží ke smazání zdrojů na serveru. Data se standardně nemažou, pouze se zneaktivní. Úspěšný statusový kód v odpovědi dorazí i v tomto případě.

Komunikace v mém případě vypadá následovně:

Uživatel pracuje s webovým klientem, což je pro mě prezentační vrstva. Celá vrstva je rozdělena do komponent a má tedy své části. Uživatel zadá HTTP požadavek, který převezme webový klient jeho kontrolerem a předá jej dle potřeby své službě. Služba požadavek převezme a předá jej Helperu4.3 a až ten teprve pošle HTTP požadavek na server. Na serveru jej API a jeho kontrolery převezmou. Zde probíhá obdobná činnost, pouze složitější, jelikož se zde již posunuji po vrstvách. Požadavek na data se dále posílá níže do BLL vrstvy, tam se zpracuje a posílá DAL vrstvě. Až ta poptá konkrétní data z databáze a celý proces se vrací jako odpověď. Odpověd zase prochází jednotlivými vrstvami, jelikož každé dvě vrstvy můžeme brát jako klient vůči serveru, kdy klient pošle požadavek a server mu pouze vrací odpověď. Odpověd postupně prochází vrstvami až do vrchních vrstev skrz DAL vrstvu, BLL vrstvu až po API, které vrací statusový kód včetně syrových dat ve formě JSONu. Tyto data převezme Helper ve webovém klientu, který jej vrátí svým službám, ty kontroleru a nakonec View zaregistruje změnu v datech modelu a vykreslí je uživateli. Tento postup demonstruje obrázek4.5.

- **Helper**
  Helper je mnou vytvořený balíček služeb, který pomáhá dvěma základním funkcionalitám ve webovém klientu. První je:
  - **Identita**
    Tato třída udržuje identitu o uživateli, kterou získá od serveru.
  - **Request**
    Tato třída pomáhá se zmiňovanými HTTP požadavky. Určuje jejich typ, přidává hlavičku dle potřeby a volá server.

  Helper je oddělen od služeb, jelikož jej využívá celá aplikace napříč.



Obrázek 4.5: Názorná ukázka komunikace, vrstva po vrstvě

Kapitola 5

Implementace

V této kapitole projednávám konkrétní implementace navrhované části. Rozebírám zde jednotlivé třídy vrstev a jejich funkce.

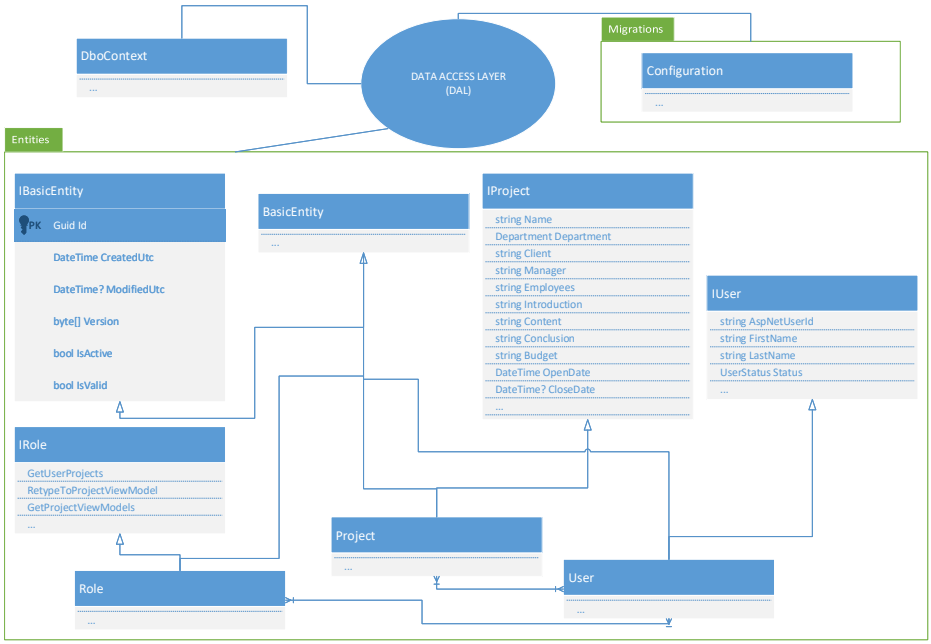
5.1 Serverová řešení

Konkrétní implementace vrstev serverového řešení.

5.1.1 Datová vrstva - DAL

Představuji hlavní balíčky a třídy, které se nacházejí v datové vrstvě serverové části řešení.

Na obrázku 5.1 je vidět hlavní obsah datové vrstvy. Jsou zde balíčky Migrations, Entities a zvlášť třída DboContext.



Obrázek 5.1: Hlavní balíčky, třídy a entity v datové vrstvě

Migrations

V balíčku Migrations se nachází třídy konkrétních migrací4.1.1 a třída Configuration.

- **Konkrétní migrace**

  Tyto migrace slouží k verzování databáze. Ukládají se zde informace o aktualizacích databáze pomocí Code First metodiky4.1.1.

- **Configuration**

  V této třídě se databáze konfiguruje. Je zde také **Seed** metoda, která slouží především k naplnění nové databáze daty, které jsou potřebné pro chod aplikace. Mohou to být například typy rolí a podobně. Jednoduše data, které aplikace nezíská od uživatelů.

Entities

V balíčku Entities definuji entity, podle kterých se databáze metodikou Code First4.1.1 vytvoří a které jsou základními objekty celé aplikace. Všechny

entity mají své vlastni rozhraní, které implemetují. Entity jsou:

- **BasicEntity**

  Základní entita od které dědí všechny ostatní. Entita definuje základní aplikační data pro údržbu a interní informace konkrétních objektů. Jedny ze základních dat jsou tyto:

  - Guid Id - jedná se o primární klíč každého objektu
  - DateTime CreatedUtc - drží informace o času vytvoření objektu
  - bool IsActive - údaj určuje, zda je objekt aktivní, jelikož žádná data nemažeme, pouze je zneaktivníme

- **Role**

  Entita Role definuje roli uživatele, buďto klientskou, či zaměstnaneckou. Dále definuje několik vlastností:

  - string Name - název role
  - RoleType Enum - jedná se o typ role
  - string Description - popis samotné role
  - ICollection<User> Users - vazba na uživatele.

- **Project**

  Entita Project definuje samotný projekt. Obsahuje veškeré informace o projektu včetně citlivých dat, jako je například rozpočet. Některé z nich vyjmenuji.

  - string Name - název projektu
  - Department Department - oddělení
  - string Employees - jména pracujících zaměstnanců na projektu
  - string Content - obsah projektu
  - string Budget - rozpočet projektu
  - DateTime OpenDate - začátek projektu
  - DateTime? CloseDate - konec projektu
  - ICollection<User> Users - vazba na uživatele

- **User**

  Tato entita definuje uživatele.

  - string AspNetUsersId - vazba na aplikačního uživatele.5.1.2
  - string FirstName - jméno
  - string LastName - přijmení
  - UserStatus Status - status uživatele
  - ICollection<Project> Projects - vazba na projekty
  - ICollection<Role> Roles - vazba na role

DboContext

DboContext je kontextová třída, dědící od DbContextu4.1.1. Konfiguruji zde konektivitu s naší databází. Také zde registruji mé entity pomocí DbSetu4.1.1.

```
public DbSet<User> Users { get; set; }
```

Vždy, když některá z vrstev pracuje s datovou, vytváří si DboContext. Po dokončení operace je třeba využívání DboContextu ukončit, jinak by mohlo dojít k nesprávnému chodu aplikace. DboContext využíváme tedy následovně:

```
using (var db = new DboContext())
{
        var dbUser = db.Users.Add(new User
        {
        ...
        });
        db.SaveChanges();
        return dbUser;
}
```

5.1.2 Byznysově logická vrstva - BLL

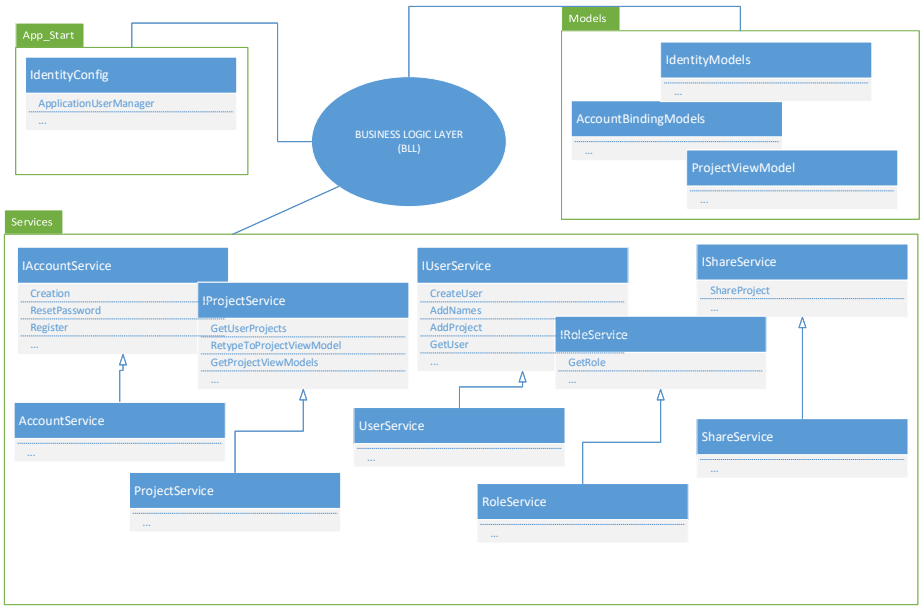
Implementace byznysově logické vrstvy se dělí do několika balíčků. Nachází se zde balíček App_Start se třídou *IdentityConfig*, dále balíček Models s jednotlivými modely a poslední důležitý balíček Services s jednotlivými službami. Graficky znázorněný obsah této vrstvy můžeme vidět na obrázku 5.2

App_Start

V tomto balíčku se nachází třída *IdentityConfig*, která definuje *ApplicationUserManager*, což je technologie společnosti Microsoft pro správu uživatelských identit.

- **ApplicationUserManager**

  Tato třída *ApplicationUserManager* je technologie společnosti Microsoft, sloužící jako správa aplikačních uživatelů. Jelikož v práci využívám zabezpečení vyvinuté společností Microsoft, pracuji také s těmito aplikačními uživateli.



Obrázek 5.2: Hlavní balíčky, třídy a metody v servisní vrstvě

Aplikační uživatel je uživatel vytvořený na základě ApplicationUserManager třídy. Jsou to defaultní uživatelé vytvoření technologií Entity Framework4.1.1. Tyto uživatele ukládám do tabulky AspNetUser a vážu je k mým uživatelům pomocí AspNetUserId. V této tabulce si držím pouze Email, UserName a PasswordHash, nicméně tabulka je připravena na ukládání více dat.

Také potřebuji třídu **UserManager** na správu **ApplicationUser** instancí. Abych to mohl udělat, musel jsem vytvořit třídu **IdentityConfig.cs** v App_Start, kterou držím v BLL vrstvě. Pokud budu chtít, mohu přidat kód do ApplicationUserManager třídy ke konfiguraci validační logiky pro uživatelské jméno – username a heslo. Mohu implementovat dvoufaktorového autentifikačního poskytovatele pro email a dokonce SMS. Také zde mohu připravit autorizaci na základě rolí. To vše mi ApplicationUserManager dovoluje.[Pen16]

Mohu zde nastavovat různé validátory. Nastavuji pravidla pro validaci uživatelských profilů, jako například vyžadování unikátních emailů nebo povolení pouze AlphanumericUserNames, což znamená, že součástí uživatelského jména musí být abecední znak i číslo. Co je však praktičtější, je možnost nastavení složitosti hesel. Technologie společnosti Microsoft vyžaduje kvůli bezpečnosti standardně složitá hesla. Nastavování těchto validátorů je snadné, například kód pro složitost hesel nastavím takto:

```
manager.PasswordValidator = new PasswordValidator
{
```

```
            RequiredLength = 6,
            RequireNonLetterOrDigit = true,
            RequireDigit = true,
            RequireLowcase = true,
            RequireUppercase = true,
};
```

Instance *ApplicationUserManager* poskytuje metody pro získání totožnosti uživatelských informací, změnu hesla, přidání externích přihlašovacích údajů pro aplikačního uživatelei, odstranění místního nebo externího přihlášení uživatele identity, registrace uživatele a přidání přihlašovacích údajů.[Gun15]

Instance *ApplicationUserManager* je k dispozici ostatním komponentám skrze Dependency Injection4.1.2 nebo z OWINContext1 v požadavku. Ukázka injektování třídy *ApplicationUserManager* ve třídě *AccountController*:

```
public class AccountController : ApiController
{
      public ApplicationUserManager UserManager
      {
            get
            {
                  return Request.GetOWINContext()
                  .GetUserManager<ApplicationUserManager>();
            }
       }
.
.
}
```

Models

V tomto balíčku se definují objekty, se kterými služby pracují. Data v modelech jsou službami buď vytvářeny či měněny a posléze se navracejí v odpovědi. Nicméně se nejedná o pravidlo a s modely a jejich daty mohou služby pouze pracovat a nevracet je v odpovědi. Hlavní jsou jmenovitě:

1. **IdentityModel**

   V *IdentityModelu* je definován *ApplicationUser*, tedy aplikační uživatel5.1.2. Bez *IdentityModelu* bychom nemohli používat třidu *ApplicationUserManager*5.1.2, protože bychom neměli tohoto aplikačního uživatele definovaného.

2. **AccountBindingModels**

AccountBindingModels jsou modely, které používám při práci s uživatelským účtem. Například *CreateBindingModel* při vytváření uživatele nebo *ResetPasswordViewModel* pro resetování hesla.

3. **ProjectViewModel**

   Tento model je podstatný pro práci s projekty. Do tohoto modelu transformuji data z databáze. Transformovaný Project poté posílám přes *ODataController* ve formě JSON.

Services

Zde se nachází hlavní logika celé aplikace. Každá ze služeb má své rozhraní, ve kterém je jasně definováno, které metody musí služba implementovat. Některé služby využívají jiné, jako například *ProjectService* využívá *UserService*, protože potřebuje dostat projekty určitého uživatele a toho uživatele si poptá přes *UserService*. Každá služba by se měla věnovat pouze svému specifickému úkolu.

1. **AccountService**

   Account service pracuje výhradně s metodami zabývajícími se uživatelským účtem. Jedná se tedy o metody jako jsou

   - **Creation** – metoda sloužící k vytvoření uživatele
   - **ResetPassword** – metoda pro resetování hesla
   - **Register** – metoda pro samotnou registraci uživatele

2. **ProjectService**

   Tato service pracuje opět pouze s projekty a jedny ze základních metod jsou:

   - **GetUserProjects** – metoda sloužící k získání projektů od určitého uživatele.
   - **RetypeToProjectModel** – tato metoda transformuje databázový objekt Project na objekt *ProjectModel*, se kterým se dále pracuje.
   - **GetProjectModels** – je metoda, která využívá obě dvě předešlé metody a vrací požadovaný výsledek, tedy projekty určitého uživatele ve formě *ProjectModels*.

3. **UserService**

   Je služba pracující pouze s uživateli. Tato služba kooperuje s nižší vrstvou, protože se dotazuje v databázi na uživatele, vytváří je, modifikuje a tak dále. Jedny z hlavních metod tedy jsou:

- **CreateUser** – vytvoří nového uživatele
- **AddNames** – přidá jméno a příjmení uživateli
- **AddProject** – přidělí vybrané projekty uživateli
- **GetUser** – vrátí požadovaného uživatele

4. **ShareService**

   Poslední služba, o které se zde zmiňuji pracuje se sdílením projektů. Zde by situace mohla být řešena tak, že místo ShareService se funkcionalita implementuje v ProjectService, protože se přece jen jedná o projekty, nicméně z důvodu dodržování SRP[18],je tato funkcionalita oddělena od zbytku.

   - **ShareProject** – zde je základní metoda, která sdílí projekt a v případě, že jej sdílí s novým uživatelem, musí být tento uživatel vytvořen a rovnou musí mít nasdílený projekt. Metoda je tedy zdánlivě triviální, nicméně musí pracovat i s jinými, poměrně složitými funkcemi, jako je vytváření uživatele a podobně.

5. **RoleService**

   Jedná se o službu zabývající se zjišťováním rolí. Jelikož bych bez její funkcionality nebyl schopen rozeznat klienty od zaměstnanců, mohlo by se stát, že klienti by se dostali k citlivým datům.

   - **GetRole** – na základě dotazu vrátí role daného uživatele

5.1.3 Aplikační programové rozhraní - API

V API je použit framework ASP.NET Web API, což je framework, který usnadňuje vytváření HTTP služeb, které spolupracují s širokou škálou klientů, včetně prohlížečů a mobilních zařízení. Je platformou pro vytváření RESTových aplikací na .NET framework.[Gun15]

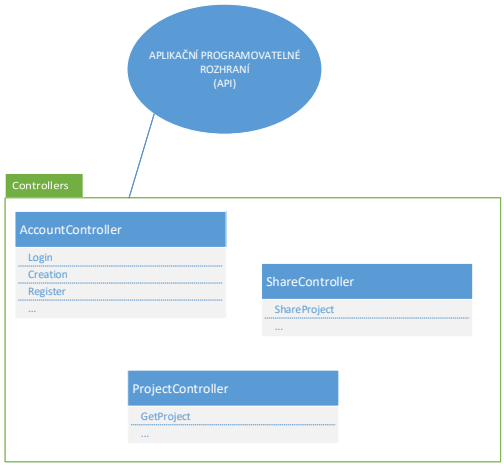
V této vrstvě se nachází především kontrolery, které jsou volány skrz HTTP požadavky4.3 a dále volají nějakou službu s určitým modelem.

Balíčky

- **Controllers**

  Ukážeme si zde základní kontrolery.

[18] Single Responsibility Principle - princip jedné odpovědnosti



Obrázek 5.3: Hlavní balíčky, třídy a metody v aplikačním programovém rozhraní

1. **AccountController**

   Jedná se o Controller dědící od ApiControlleru. Jeho akce jsou především ohledně uživatelských účtů a je na něj přímo napojen AccountService.

   - **Login** – volá službu, která přihlásí uživatele
   - **Creation** – volá službu, která vytvoří uživatele
   - **Register** – volá službu, která registruje uživatele

2. **ProjectController**

   Je OData controller. O výhodách si můžete přečíst zde4.1.3. Hlavní metoda je

   - **GetProject** – volá službu, která vrátí projekty uživatele s možností filtrace

3. **ShareController**

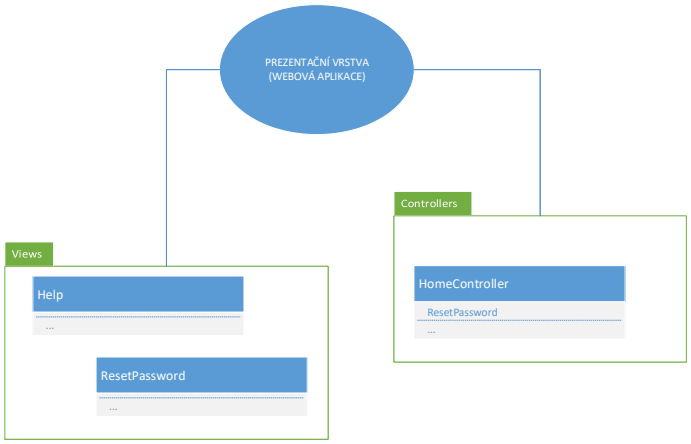
   je kontroler, který má jednu akci sloužící ke sdílení projektů. Tato akce není ve třídě *ProjectController*, protože se v tomto případě vyplatí pracovat s běžným kontrolerem oproti OData kontroleru.

   - **ShareProject** – volá službu, která sdílí projekt s uživatelem. V případě, že neexistuje vytvoří jej a pošle mu email.

Toto byly základní balíčky a metody našich vrstev API.

5.1.4 Webová aplikace

Zde se nachází prezentační vrstva jakožto součást serverového řešení. Na obrázku5.4 vidíte, že webové rozhraní má dva balíčky. První je Controllers a druhý Views.



Obrázek 5.4: Hlavní balíčky, třídy a metody v prezentační vrstvě serverového řešení.

- **Controllers**

  Zde se nachází základní kontroler pro práci s resetováním hesla.

  1. **HomeController**

     V tomto kontroleru je jedna z akcí *ResetPassword*. Proč tedy není v *AccountControlleru*, když se zabývá uživatelským účtem a změnou jeho hesla? Potřebuji totiž vlastnosti klasického kontroleru**??** a ne ApiControlleru, proto je nežádoucí mít tuto metodu v *AccountControlleru*. Jedná se o součást webové aplikace, která stojí vedle API.

     - **ResetPassword** – volá službu, která resetuje uživateli heslo

- **Views**

  Mám zde dvě základní Views, o kterých jsem se zmiňoval již dříve. Jedná se o views:

1. **Help** – modul pro dokumentaci API
2. **ResetPassword** – View pro resetování hesla

Praktické důvody webové aplikace přímo na serveru jsou dva.

1. **Potřeba zdokumentovat API.**

   Je běžné mít dokumentaci API dostupnou přímo na webových stránkách.

2. **Metoda vytváření uživatele.**

   Přihlášený uživatel vytváří nového uživatele. Při tvorbě komunikuje s webovým klientem a vyplňuje jeho základní údaje, roli a především email. Po dokončení odešle formulář a jeho práce je hotová, nicméně uživatel není stále dotvořen, jelikož mu chybí heslo, které si musí zadat sám. Při odeslání formuláře se na zadaný email také posílá pozvánka do aplikace skrz časově omezenou URL. Tato URL však neodkazuje na webového klienta, ale na webovou aplikaci na serveru, jelikož nechci být závislý na webovém klientu. V momentě, kdy by se webový klient zrušil a existovali by pouze nativní klienti, neměl bych možnost dodělat tuto základní operaci, jelikož uživatele chci zvát na zadání hesla na webové stránky. Tento průchod problému je vidět na diagramu5.5.

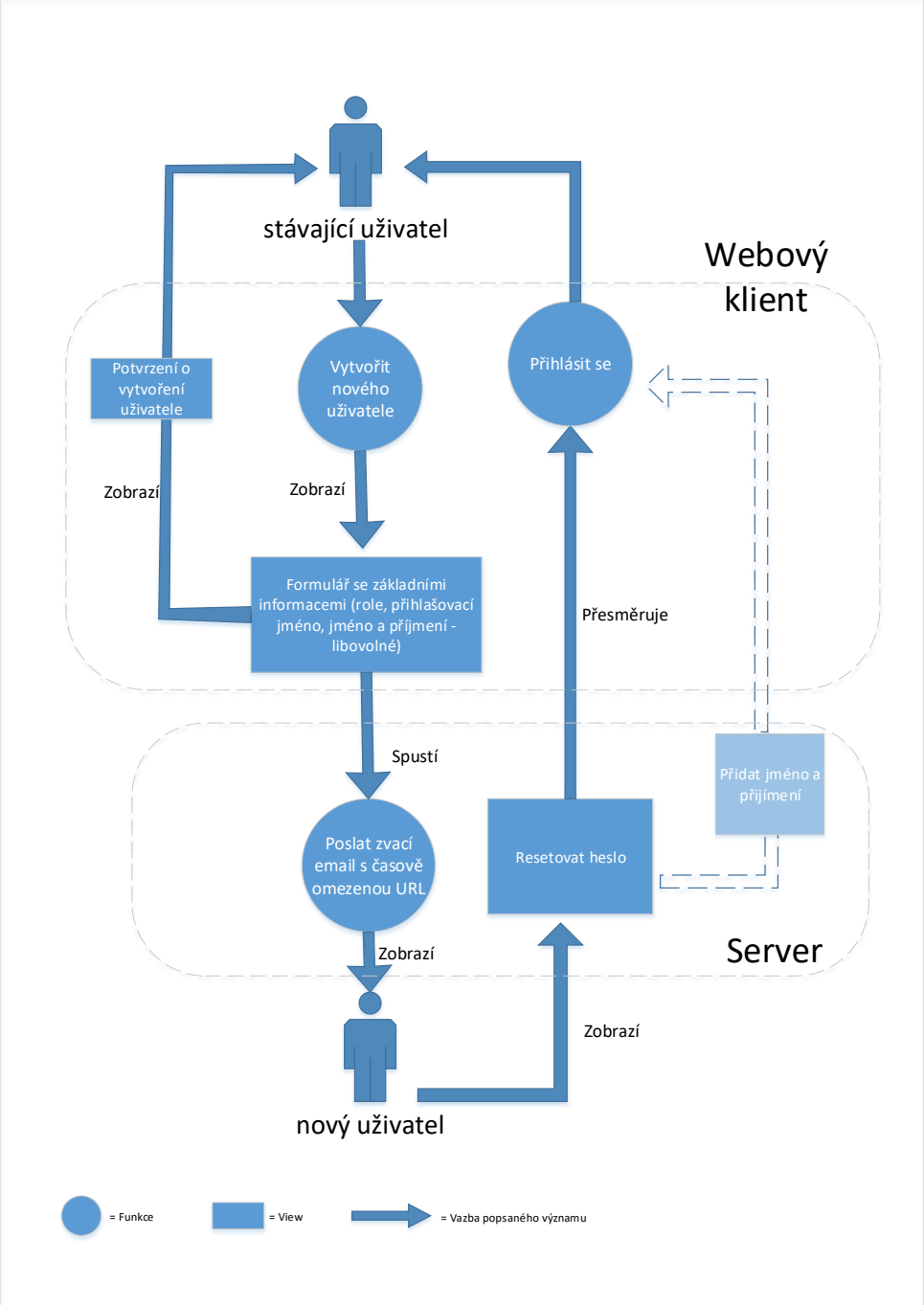
5.2 Zabezpečení

V této sekci se věnuji zabezpečení API, přistupu k samotnému účtu a další.

Pro základní zabezpečení identit používám ASP.NET Identity, což je framework společnosti Microstoft pro správu uživatelských identit v ASP.NET aplikacích. Používám jej pro registraci a správu uživatelských účtů, které jsou definovány ve třídě *IdentityUser*. Data pocházející z této technologie jsou uloženy v AspNetUser tabulce a namapovány na tabulku User pomocí AspNetUserId. Jelikož používám dvě zdánlivě oddělené identity uživatele, používám také dvě kontextové třídy.

- ApplicationDbContext

  ApplicationDbContext je podobně jako DbContext4.1.1 kontextová třída, která slouží pro práci s uživateli vytvořenými ASP.NET Identity frameworkem.



Obrázek 5.5: Metoda vytvoření uživatele s Views

5.2.1 Funkcionalita

V této sekci popisuji, jak ASP.NET Identity zabezpečení funguje.

Přihlášení dělíme na externí a interní.

1. **Interní přihlášení**

   Uživatel se registruje na webu a zadává uživatelské jméno a heslo. Aplikace uchovává hash[19] hesla v databázi. Při přihlášení uživatele ověřuje systém identifikace toto heslo.

2. **Externí přihlášení**

   Uživatel se přihlásí k externí službě jako je Facebook, Microsoft nebo Google. Aplikace stále vytváří položku pro uživatele v databázi, ale neukládá žádné autentikační data. Uživatel se ověřuje přihlášením do externí služby.

Pro oba typy přihlášení používá mé řešení technologii OAuth21, která slouží pro autentikaci požadavku. Mezi externím a interním přihlašováním se především liší tok přihlašovacích údajů a jejich rozdílné místo uložení.

Definice pojmů, které při popisování technologie přihlašování pomocí OAuth2 používám:

- Zdroje – jsou data, které lze chránit
- Zdrojový server – je takový server, který uchovává zdroje
- Vlastník zdroje – je entita, které může povolovat přístup ke zdroji
- Klient – aplikace, která chce přístup k datům
- Přístupový token – je token, který povoluje přístup ke zdroji
- Nosičský token (bearer token) – je typ přístupového tokenu, který je po vyžádání generovaný na straně autorizačního serveru a může být používán kýmkoliv. To znamená, že vlastník tokenu již nepotřebuje žádné informace navíc k jeho použití, jako je například kryptografický klíč, což některé jiné typy přihlašovacích tokenů vyžadují. Proto bývá token používán skrz HTTPS[20] a mívá relativně krátkou expirační dobu
- Autorizační server – server, který vydává přístupový token

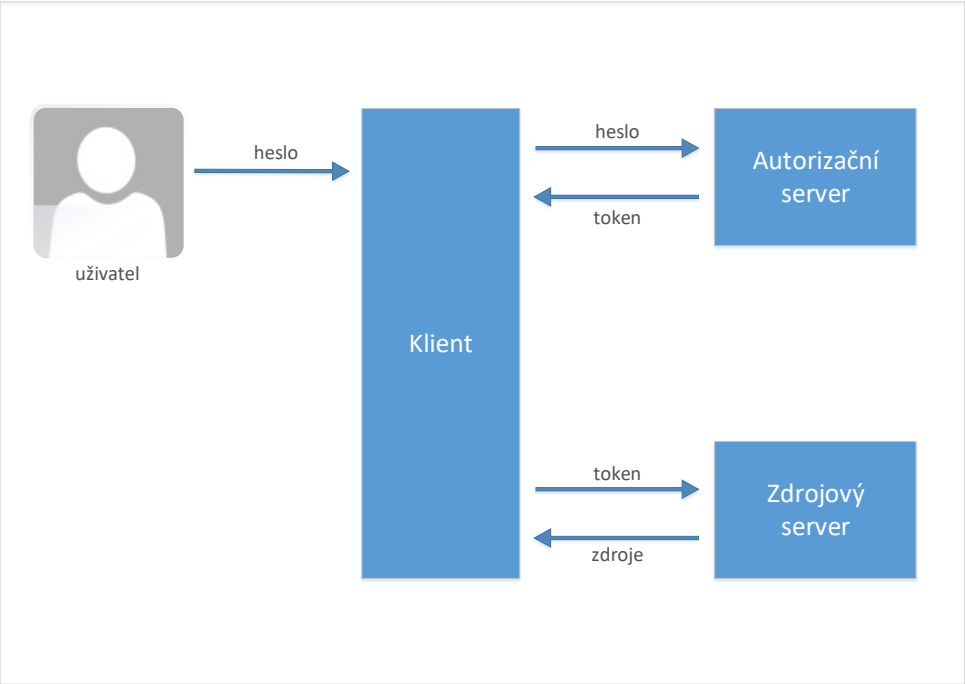
Aplikace se může chovat jako autorizační a zároveň jako zdrojový server. V našem případě tomu tak je.

[19] jednosměrný otisk vstupu
[20] šifrovaný protokol pro posílání dat na internetu

Základní tok dat

1. Uživatel zadává jméno a heslo klientovi.
2. Klient posílá přihlašovací údaje autorizačnímu serveru.
3. Autorizační server autentikuje přihlašovací údaje a vrátí přístupový token (v případě, že je klient autentikován).
4. K přístupu k zabezpečenému zdroji musí klient přidat přístupový token do hlavičky autorizačního HTTP požadavku.

Na obrázku5.6 je tok dat znázorněn, nicméně se jedná o verzi bez API komponent.

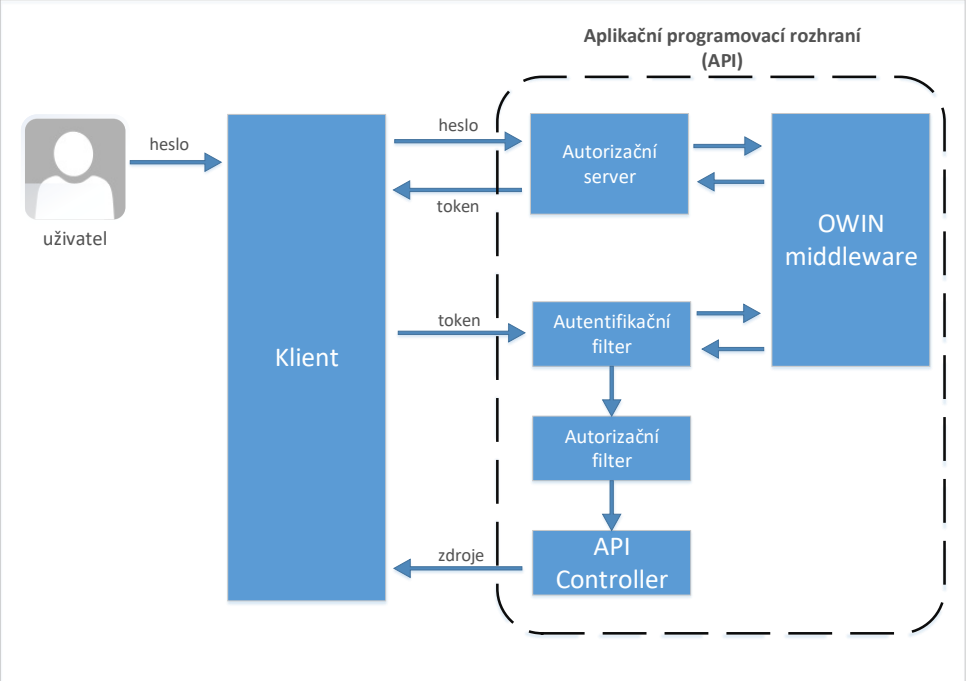


Obrázek 5.6: Tok dat při autentikaci uživatele bez API komponent

Na dalším obrázku5.7 API kontrolery pracují jako zdrojové servery. Autentikační filtr validuje přístupové tokeny a při použití atributu pro autorizaci - Authorize chráníme zdroj. V případě, že tento atribut použijeme, všechny požadavky, které směřují na tento kontroler musí být autentikovány. V případě že uživatel autentikací neprojde, vrátí mu server statusový kód 401 - Unauthorized error. Jak autorizační server, tak autentikační filtr volají OWIN1

middleware[21] komponentu, která pracuje s OAuth2.

Na obrázku5.7 je vidět autentikace uživatele s API komponentami.



Obrázek 5.7: Autentikace s API

V případě, že se uživatel neautorizuje, tedy nepřihlásí, nedrží u sebe žádný nosičský token a server nám pošle zpět již zmiňovaný statusový kód 401 - Unauthorized error. Průběh požadavku si demonstrujeme níže.

HTTP požadavek:

```
GET https://localhost:xyz/api/getproject HTTP/1.1
X-Requested-With: XMLHttpRequest
```

HTTP odpověď:

```
HTTP/1.1 401 Unauthorized
Content-Type: application/json; charset=utf-8
Server: Microsoft-IIS/8.0
WWW-Authenticate: Bearer
```

V odpovědi je vidět WWW-Authenticate, což je autentikační hlavička připravená s parametrem bearer. To znamená, že server token očekává.

[21] software, který propojuje softwarové komponenty a aplikace

Registrace

Při registraci uživatele dostanu jednoduchou odpověď 200 - OK i bez nosičského tokenu. Server jej neočekává a nevyžaduje, jelikož k této části kontroleru nepotřebuji žádnou autorizaci.

HTTP požadavek:

```
POST https://localhost:xyz/api/Account/Register HTTP/1.1
X-Requested-With: XMLHttpRequest
{"Email":"user@example.com","Password":"Password1!"
,"ConfirmPassword":"Password1!"}
```

HTTP odpověď:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/8.0
```

Login

Při přihlašování získám nosičský token. Zde se projeví OAuth2 autorizační server, který je o tento token uživatelem požádán. Po zadání emailu a hesla uživatel pošle stisknutím tlačítka login následující data:

- grand_type: „password“
- username: <uživatelův email>
- password: <uživatelovo heslo>

Autorizační server vyžaduje v objektu grand_type hodnotu „password“, aby věděl, že se jedná o požadavek o token. V případě úspěchu dostane uživatel od autorizačního serveru nosičský token v těle odpovědi. Já si tento token uložím do session1, kterou mám definovanou ve třídě Ident1 a při každém požadavku daného uživatele tento token přidám do autentikační hlavičky.

Na rozdíl od jiných forem autentikace, jako jsou například autentikace založené na cookies, si prohlížeč automaticky neukládá přístupový token do

následné žádosti sám o sobě. Musím to dělat explicitně, což je dobře, protože tím omezuji CSRF[22] útoky.

Názorná ukázka komunikace:

HTTP požadavek:

```
POST https://localhost:xyz/Token HTTP/1.1
Content-Type: application/x-www-form-urlencoded; charset
=UTF-8
X-Requested-With: XMLHttpRequest
{grant_type=password\&username=user\%40example.com
\&password=Password1!}
```

Pro zajištění bezpečnosti je nutné používat požadavek skrz HTTPS protokol. Posíláme citlivé údaje.

HTTP odpověď:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Server: Microsoft-IIS/8.0
{
  "access_token":"ismPtO98sfPrO8g7flqraY5EsNl8...",
  "token_type":"bearer",
  "expires_in":1906455,
  "userName":"user@example.com",
  ".issued":"Tue, 25 Apr 2017 23:24:49 GMT",
  ".expires":"Tue, 9 May 2017 23:24:49 GMT"
}
```

Tyto informace si ukládám do TokenModelu3 a dále je používám k autentikaci. OAuth2 definuje access_token, token_type a expires_in. Ostatní jsou pouze pro informační účel.

Autentikovaný požadavek

Nyní, když pošlu autentikovaný požadavek s nosičským tokenem v autorizační hlavičce, uspěji u autorizačního serveru i autentikačního filtru. Což demonstruje následující komunikace:

HTTP požadavek:

[22] Cross-Site Request Forgery - typ útoků

```
GET https://localhost:xyz/api/getproject HTTP/1.1
Host: localhost:xyz
Authorization: Bearer ismPtO98sfPrO8g7flqraY5EsNl8...
X-Requested-With: XMLHttpRequest
```

HTTP odpověď:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Server: Microsoft-IIS/8.0
```

Log Out

V momentě odhlášení uživatele je důležité pouze to, aby se vymazala session na klientské straně, držící si nosičský token uživatele. Vymažeme jej pomocí metody v třídě Ident Clear()1. Takto se stačí odhlásit na klientské straně aplikace, nicméně se také posílá požadavek na odhlášení serverovému API. Uživatelův nosičský token uložený u klienta se však vymaže v jakémkoliv případě.

Konfigurace autorizačního serveru

V StartupAuth třídě následující kód konfiguruje OAuth2 autorizační server.

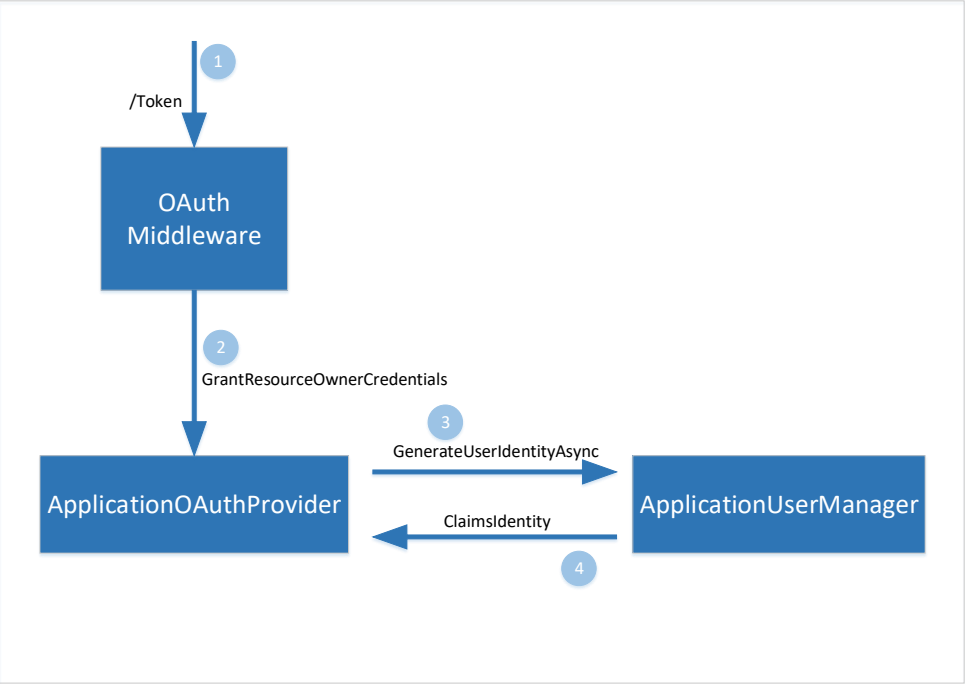
```
OAuthOptions = new OAuthAuthorizationServerOptions
{
    TokenEndpointPath = new PathString("/Token"),
    Provider = new ApplicationOAuthProvider(PublicClientId),
    AccessTokenExpireTimeSpan = TimeSpan.FromDays(14)
    .
    .
};
// Enable the application to use bearer tokensto authen-
-ticate users
app.UseOAuthBearerTokens(OAuthOptions);
```

TokenEndpointPath objekt určuje URL cestu ke koncovému bodu autorizačního serveru. To je URL, na které aplikace získávají nosičské tokeny. **Provider** je objekt specifikující poskytovatele, který je začleněn do OWIN middlewaru a do procesů vyvolaných middlewarem.

Detailní tok dat při požadavku o token

Tok dat při požadavku o token a na adresu ~/Token a ostatních přihlašovacích údajů, který lze vidět na obrázku5.8:

1. K získání přístupového tokenu pošle klient požadavek na adresu ~/Token.
2. OAuth middleware zavolá GrantResourceOwnerCredentials na poskytovateli.
3. Poskytovatel zavolá ApplicationUserManager třídu na zvalidování přihlašovacích údajů a vytvoření požadavku o identitu.
4. Pokud uspěje, ApplicationUserManager vrací nárok na identitu[24], který je použit na vygenerování tokenu.



Obrázek 5.8: Tok dat při požadavku o token

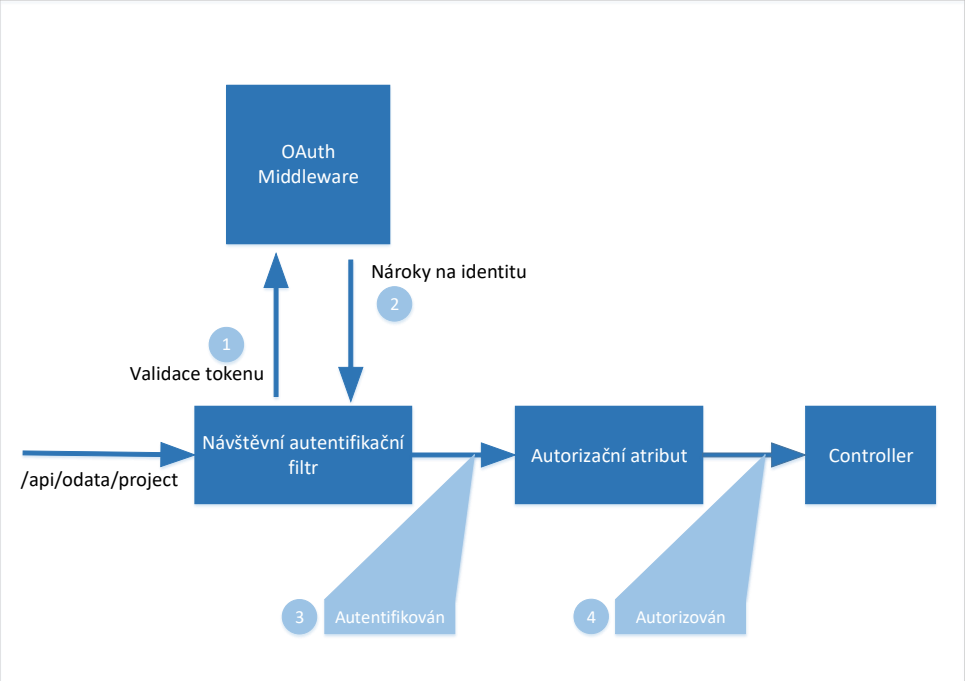
OAuth middleware neví nic o uživatelských účtech. Poskytovatel komunikuje mezi middlewarem a ASP.NET Identity.

[24] ClaimsIdentity

Autentikace versus autorizace

Rozdíl mezi autentikací a autorizací, který je také vidět na obrázku5.9:[WPD14]

1. HostAuthentication filtr zavolá OAuth middleware pro validaci tokenu.
2. Middleware konvertuje token na nárok na identitu.
3. V tomto bodě je požadavek autentikován, nikoli však autorizován.
4. Autorizační filtr zkoumá nárok na identitu. Pokud je nárok autorizován pro daný zdroj, požadavek je autorizován. Defaultně Authorize atribut autorizuje jakýkoliv požadavek, který je autentikovaný.
5. Pokud je předchozí krok úspěšný, kontroler vrátí chráněné zdroje. V jiném případě klient obdrží 401 - Unauthorized error.

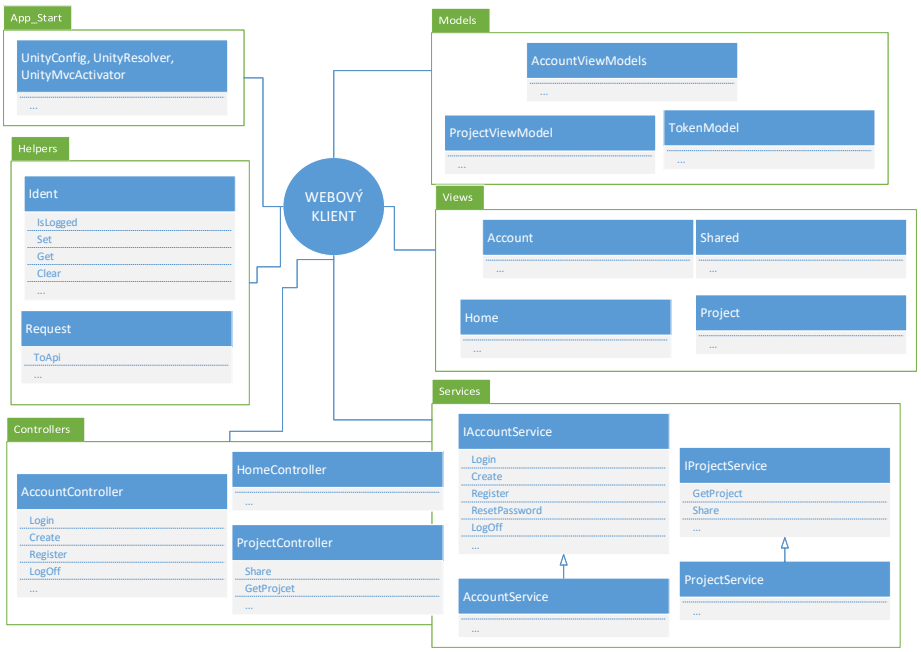


Obrázek 5.9: Autentikace versus autorizace

5.3 Webový klient

Pro ukázku funkcionality a jako jednu z platforem, která se bude pro správu a sdílení projektů používat, byl implementován webový klient. Klient, jak

jsem již zmínil, je navržen podle MVC4.1.4 vzoru. Základní balíčky, které klient obsahuje jsou: **App_Start**, **Models**, **Views**, **Controllers**, **Services** a **Helpers**. Všechny tyto balíčky a třídy jsou vidět na obrázku5.10.



Obrázek 5.10: Základní balíčky, třídy a metody ve webovém klientu.

- **App_Start**

  Balíček, ve kterém se nastavují základní informace o projektu.

  Tento balíček slouží k nastavení celého projektu, tedy celého webového klienta. Jedná se o nastavení URL konvencí, používání skriptů, kaskádových stylů a podobně. Zařadil jsem jej do základních balíčků, o kterých se v práci zmiňuji, jelikož se zde nachází *UnityConfig*, *UnityResolver* a *UnityMvcActivator*.[Dom13] Tyto třídy jsou podstatné pro možnost používání Dependency Injection4.1.2 skrz konstruktory.

  1. **UnityConfig**

     V této třídě se registrují typy, které se dále mohou používat v konstruktorech. Bez jejich registrace si řešení neví rady a nedokáže takový konstruktor vytvořit. To vede k nefunkčnosti nainstancovaných tříd a tedy nemožnosti jejich použití.

  2. **UnityResolver**

     Po registraci vazem v *UnityConfig*, musíte použít metodu Resolve v UnityResolver třídě, aby vyřešila libovolné závislosti. V této třídě se nachází také metoda Dispose, která tyto vazby přerušuje. Takto funguje celý cyklus registrování závislostí. Nejdříve je třeba

je zaregistrovat, poté použít metodu Resolve na jejich vyřešení a nakonec tyto vazby opět přerušit.

3. **UnityMvcActivator**
   Tato třída je zde pokud se používa ASP.NET MVC aplikace. Integruje Unity s naší aplikací.

- **Models**

  Zde se nacházejí modely, se kterými pracuji.

  1. **AccountViewModel**
     Modely zabývající se uživatelským účtem, jako jsou *CreateViewModel* na vytváření uživatele, či například *ResetPasswordViewModel* pro resetování hesla uživatelem.
  2. **ProjectViewModel**
     Model pro práci s projekty. Data obsažené v objektu se zobrazují ve View.
  3. **TokenModel**
     Tento model obsahuje informace o tokenu, který je zasílán kvůli autorizaci5.2. Vlastnosti TokenModelu jsou:
     - **access_token** - samotná hodnota tokenu
     - **token_type** - typ tokenu
     - **expires_in** - doba expirace tokenu
     - **userName** - uživatelské jméno autorizovaného uživatele
     - **error** - název chyby
     - **error_description** - popis chyby

- **Views**

  Zde se nacházejí Views4.1.4, což jsou soubory pro vykreslení uživateli.

  1. **Account**
     Zde se nachází Views pro zobrazení stránek související s uživatelským účtem.
  2. **Home**
     Zde se nachází základní stránka.
  3. **Project**
     Views související s vykraslením projektů.
  4. **Shared**
     Zde jsou sdílené Views, jako jsou Footer[25], Error[26] a podobně.

- **Controllers**

  Nachází se zde kontrolery4.1.3, které zpracovávají požadavky uživatele a dále je distribuují.

[25] zápatí stránky
[26] chyba

1. **AccountController**
   Tento kontroler se zabývá uživatelským účtem.
   - **Login**
     Akce zpracovávající přihlášení uživatele.
   - **Create**
     Akce na vytvoření uživatele.
   - **Register**
     Akce pro registraci uživatele.
   - **LogOff**
     Akce, která uživatele odhlašuje.
2. **HomeController**
   Tento kontroler slouží pouze pro základní stránku.
3. **ProjectController**
   ProjectController se zabývá projekty, jejich sdílením a zpracováváním.
   - **Share**
     Akce sloužící pro sdílení projektů.
   - **GetProject**
     Akce pro zobrazení a posílání projektů.

- **Services**

  Zde se nachází logika webového klienta, především jde o posílání požadavků na server a podobně.

  1. **IAccountService**
     Rozhraní, které definuje základní metody pro práci s uživatelskými účty.
     - **Login**
       Metoda pro přihlášení uživatele.
     - **Create**
       Metoda,která slouží k vytvoření uživatele
     - **Register**
       Metoda pro registraci uživatele.
     - **ResetPassword**
       Metoda pro resetování hesla uživatele. Slouží při vytváření uživatele, kdy je nový uživatel pozván skrz URL do aplikace a doplňuje si své heslo.
     - **LogOff**
       Metoda, která slouží k odhlášení uživatele. Používa Helper.Ident pro mazání informací o přihlašovacím tokenu a uživateli.
  2. **IProjectService**
     Rozhraní projektové služby. Toto rozhraní definuje základní práci s projekty.

- **GetProject**
  Tato metoda vrací projekty na základě uživatele a jeho role.
- **Share**
  Metoda pro sdílení projektů mezi uživateli.

3. **AccountSerice**
   Služba implementující rozhraní *IAccountService*.
4. **ProjectService**
   Služba implementující rozhraní *IProjectService*.

- **Helpers**

  Tento balíček obsahuje třídy, které slouží k pomoci při práci, kterou využívají všechny služby napříč aplikací.

  1. **Ident**
     Tato třída pracuje s identitou uživatele. Je zodpovědná za udržení uživatelovy identity a tedy jeho autorizačních informací, které vždy posílá v hlavičce na server při jakémkoliv autorizovaném požadavku. Tyto informace si udržuje aplikace v takzvané Session1.

     - **IsLogged**
       Hodnota, zda je uživatel přihlášen či nikoli.
     - **Set**
       Zde se nastavuje přihlašovací token uživatele a jeho informace.
     - **Get**
       Metoda pro vracení informací o přihlášeném uživateli.
     - **Clear**
       Metoda pro vymazání záznamu u přihlášeném uživateli. Používá se při jeho odhlášení, jelikož uživatel je ve webovém klientu příhlášen pouze způsobem uchovávání jeho přihlašovacího tokenu.

     Session je statusová informace, která je dostupná napříč celou aplikací.[MF10]
  2. **Request**
     Tato třída slouží k posílání HTTP požadavku, dle potřeby přidávání informací do hlavičky a rozhodování o typu požadavku.

     - **ToApi**
       Metoda je zodpovědná za posílání requestů serverovému API.

Kapitola 6

Integrace

V této části popisuji, jakým způsobem je možné integrovat mé řešení se službou SAP, a to konkrétně k získávání dat, a také možnost integrace s externími přihlašovacími systémy.

6.1 Data

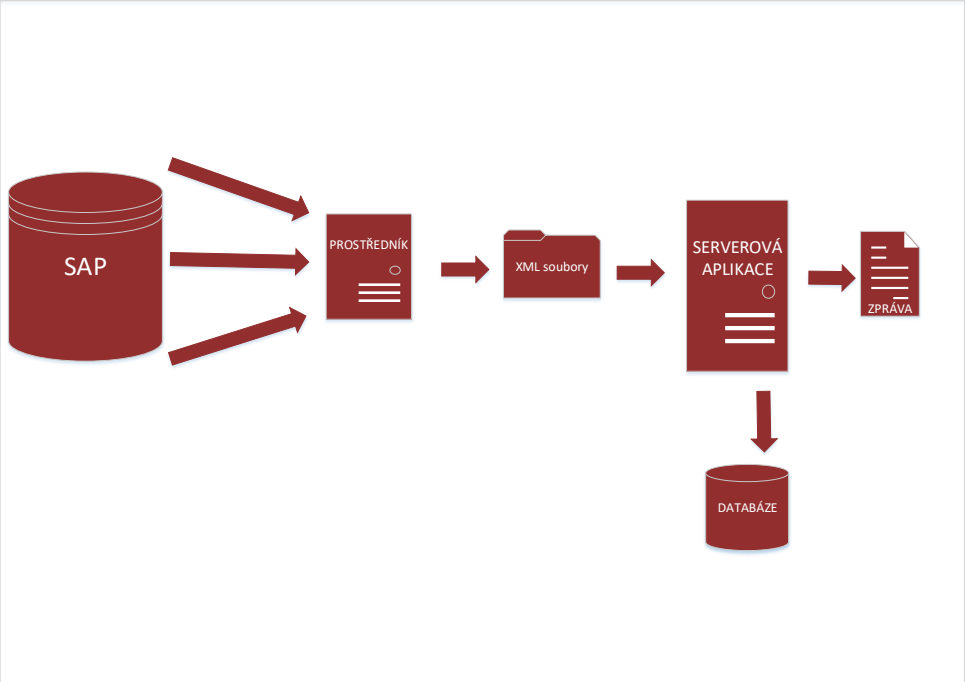
6.1.1 SAP

Již vzniklá data o projektech získám prostřednictvím integrace s prostředím SAP1. Jelikož se tyto data v systému SAP nacházejí, stačí je vybraným způsobem integrovat s mou aplikací. Možností, jak tyto data získat, je více, nicméně vybral jsem formu, která je nenáročná pro realizaci a je především bezpečná, jelikož nebudeme muset přímo propojovat službu SAP s aplikací třetí strany. Tento postup by byl nebezpečný, jelikož by třetí strana mohla mít přímý přístup k datům. Obvyklejší je tedy zvolit postup, který nenutí přímou integraci se systémem, ale pouze nutí systém data v určité formě poslat.

Datová struktura systému SAP s nejvyšší pravděpodobností není totožná s potřebami mé aplikace. Proto je třeba data zpracovat a poslat v takové formě, v jaké je požaduje má aplikace. Ta je vyžaduje podle své již navržené

a implementované datové struktury. Typicky vhodné řešení je navrhnou a implementovat prostředníka mezi systémem SAP a mou aplikací, který by přijímal od systému SAP data a konvertoval je na požadovaný způsob. To přímo nezatěžuje SAP k transformaci dat na požadovanou formu a zároveň dostane mé řešení definované data. Tento způsob je jednoduchý pro přípravu i samotnou implementaci. Soubory vyžadované mou aplikací mouhou být přijímány ve formě XML, JSON a podobně, nicméně já v tomto případě upřednostňuji soubory XML, jelikož se dle mého s těmito soubory dobře pracuje a také to koreluje s firemní kulturou společnosti, co se využívaní technologií týče. Nicméně pokud někdo upřednostňuje například JSON není v tom výrazný problém. Má aplikace již takto připravené soubory příjme a podle svých vnitřně implementovaných konvertorů těmito daty naplní databázi.[Aze14]

Na obrázku 6.1 je demonstrován celý postup.



Obrázek 6.1: Navrhovaný posup při přenosu dat ze systému SAP.

1. SAP shromáždí potřebná data ze svých datových struktur a zašle je prostředníkovi.
2. Prostředník převezme tyto data a překonvertuje je do předem nadefinované podoby, například XML.
3. Překonvertované soubory XML prostředník odešle serverové aplikaci.
4. Serverová aplikace přijme data, zpracuje je a vygeneruje informační zprávu.

5. Po zpracování je serverový systém náležitě uloží do databáze.

Návrh XML struktur pro příjem dat o projektech je následující:

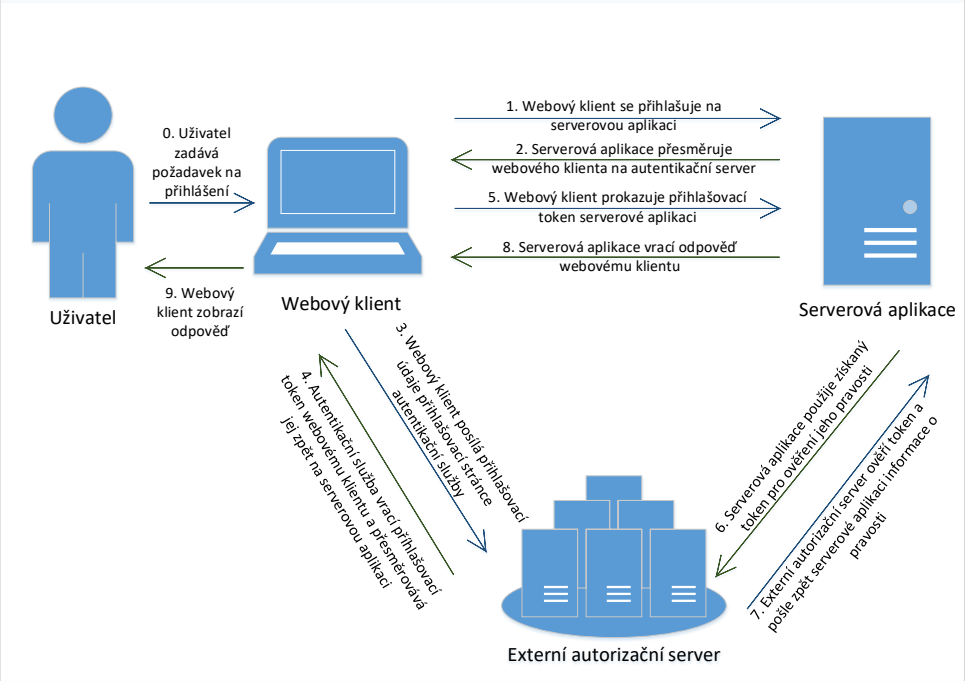
```
<xml>
<?xml version="1.0" encoding="UTF-8"?>
<MASTER_DATA>
        <item>
        <RECORD_TYPE>PROJECT</RECORD_TYPE>
        <NAME>DPSP</NAME>
        <DEPARTNMENT>CONSULTING</DEPARTMENT>
        <CLIENT>AADAMOVA@DPSP.COM</CLIENT>
        <MANAGER>SOUKUP@DPSP.COM</MANAGER>
        <EMPLOYEES>ONDERKA@DPSP.COM</EMPLOYEES>
        <INTRODUCTION>INTRODUCTION</INTRODUCTION>
        <CONTENT>CONTENT</CONTENT>
        <CONCLUSION>CONCLUSION</CONCLUSION>
        <BUDGET>150000</BUDGET>
        <OPEN_DATEP>2016-01-10</OPEN_DATE>
        <CLOSE_DATE>2017-05-01</CLOSE_DATE>
        </item>
        .
        .
</MASTER_DATA>
```

6.2 Externí přihlášení

V této sekci popisuji integraci jiných způsobů přihlašování, než-li lokálních5.2. Nejedná se tedy o interní, ale externí zabezpečovací systém. Těchto externích poskytovatelů přihlášení je několik. Rozhodl jsem se rozebrat možnost integrace s takzvaným sociálním loginem. Sociální login je typ přihlášení přes sociální sítě. Vzhledem k tomu, že jsou dnes běžné a masově používané služby společnosti Facebook, či Google, dá se téměř s jistotou říci, že většina uživatelů mé aplikace bude tyto služby také používat. V čemž vidím potenciál z hlediska integrace. Pokud bych se pokusil integrovat jiný systém, bylo by velice pravděpodobné, že bych musel nové uživatele navíc registrovat i do této externí služby, která by jako jeden z požadavků měla usnadnit přístup do aplikace. Což by se v takovém případě tedy nestalo. Sociální login je proto dle mého rozumné řešení.

Průběh standardní komunikace s externím autorizačním serverem je popsán

na obrázku 6.2.



Obrázek 6.2: Bežná komunikace s externím poskytovatelem autorizačních služeb.

6.2.1 Sociální přihlášení

Možnosti přihlášení skrz sociální login není pouze moderní trend, ale také pomáhá vývojářům ulehčit práci. Vývojář nemusí implementovat vlastní řešení a může použít ověřené a populární sociální přihlášení. Má to i další výhodu. Na sociálních sítích se náchází velká část uživatelů internetu, proto je velice pravděpodobné, že i uživatelé mé aplikace budou sociální sítě používat. A aby si nemuseli pamatovat další přihlašovací údaje, postačí jim ty, které používají běžně.

Výhody jsou tedy následující:

1. Ušetření času vývojáře.
2. Ulehčení přístupu uživatelům.
3. Zajištěná funkčnost.

V minulosti měli vývojáři dvě možnosti, jak tuto problematiku řešit:

1. Navrhnou a implementovat vlastní autentikační řešení.

   To je především spojeno s nevýhodou ztráty drahocených zdrojů, tedy především času samotných vývojářů.

2. Naučit se integrovat externí autentikační službu do aplikace.

   Dnes exituje několik způsobů, jak si tuto cestu ulehčit. O jednom z nich píši i já.

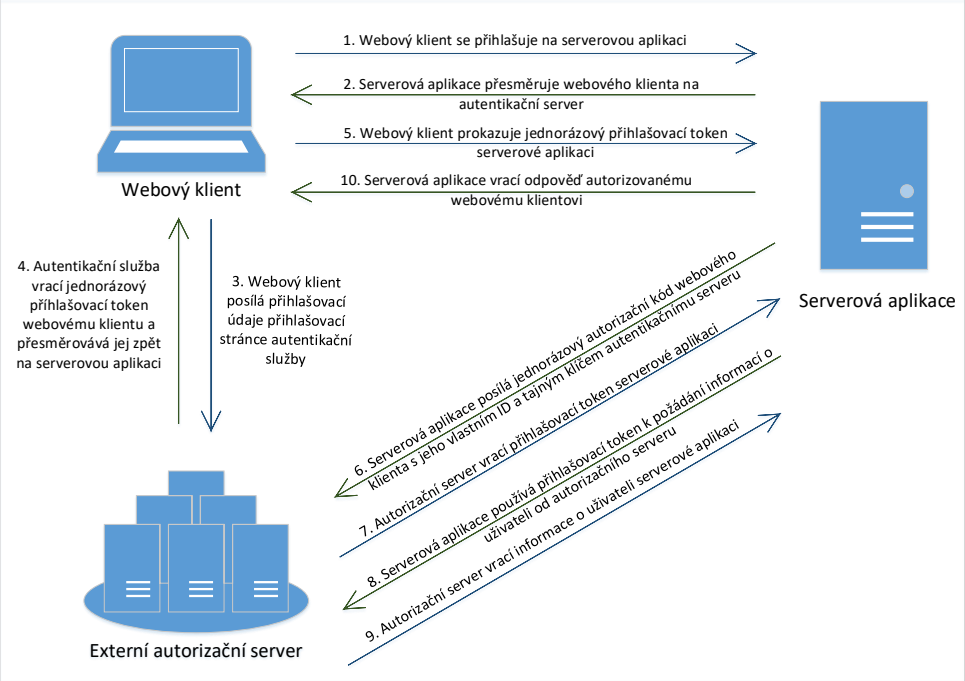
Díky technologiím společnosti Microsoft je integrace s externími poskytovateli příhlášení snadná. Stačí si nainstalovat balíček služeb Microsoft.Owin .Security.Facebook, .Google či dokonce .Twitter nebo .MicrosoftAccount a poté implementovat pouze malou část kódu do aplikace a integrace je téměř hotová. Do konfigurační třídy Startup.Auth vložíme následující kód:[MPD13]

```
public partial class Startup
{
     public void ConfigureAuth(IAppBuilder app)
     {
          app.UseFacebookAuthentication(
               appId: '' '',
               appSecret: '' '');
               .
               .
               app.UseGoogleAuthentication(
               new GoogleOAuth2AuthenticationOptions()
               {
                    ClientId= '' '',
                    ClientSecret= '' ''
               });
     }
}
```

Pokud chci používat externí login společnosti Facebook, musím si požádat o vývojářský účet. Poté co jej dostanu, vygeneruji si appId[23] a appSecret6.2.1, což jsou několikaznakové hodnoty, které pouze vložím do výše zmíněného kódu a integruji tak sociální login do mé aplikace. V případě využívání loginu společnosti Google je postup ještě jednodušší. Vývojář nepotřebuje speciální účet a může získat hodnoty ClientId[24] a ClientSecret6.2.1 přes svůj běžný účet poskytovaný společností.[Inc16]

Na obrázku6.3 lze vidět komunikaci s autorizačním serverem typu Facebook či Google, který poskytne mimo autorizačního přístupu navíc teké data o uživateli.

[23] ověřovací hodnoty společnosti Facebook
[24] ověřovací hodnoty společnosti Google



Obrázek 6.3: Komunikace s poskytovatelem sociálního přihlášení a poptání o data

Integrace mé aplikace s jiným typem přihlašování než lokálním je tedy snadná. Stačí použít zmíněné balíčky služeb společnosti Microsoft a poptat si Id s tajným klíčem od společnosti, která službu poskytuje.

Není však vhodné, aby se do aplikace mohl přihlásit každý, kdo tyto externí služby využívá, proto bych zavedl jednoduchý ověřovací systém, který by fungoval po pokusu o přihlášení a ještě by explicitně kontroloval, zda je osoba na seznamu pozvaných lidí do aplikace. Tento seznam by se automaticky aktualizoval po každé zaslané pozvánce do aplikace a pravděpodobně by se ověřoval podle emailových adres, což je výhodné, jelikož se emailová adresa používá v mém interním přihlašovacím mechanismu jako uživatelské jméno (a tak samo ve výše zmíněných externích systémech). Proto by toto ověření bylo snadné na implementaci a zamezilo by tak nezvaným hostům.

Kapitola 7

Verifikace

Verifikace kontroluje funkčnost a použitelnost aplikace pomocí testů a dokumentace.

7.1 Testování

Testování ověřuje funkčnost aplikace skrz různé přístupy. Jedna z forem testování je naimplementovat unit testy. Další, kterou používám je otestovat řešení funkčními testy.

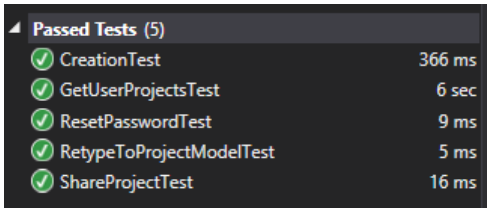
7.1.1 Unit testy

Unit test je automatický test k ověření fungování a korektnosti implementace systému. Test posílá data do metody třídy a ověřuje výsledné hodnoty, které metoda vrací, či se kterými pracuje. Unit testy jsou důležité pro ověření funkčnosti kódu a mohou být použity přednostně před vývojem aplikace.[Hor06] Tomuto postupu se říká Test Driven Development.

Test Driven Development je přístup k vývoji softwaru, který je řízený testy a vede ke zkvalitnění práce softwarových vývojářů. Prvním krokem

tohoto přístupu je definice funkcionality a následné napsání testu, který tuto funkcionalitu ověřuje.[Bec03]

V mém případě slouží hlavně k ověření základní funkcionality serverového řešení. Smyslem unit testů není pokrýt veškerý kód projektu, ale především hlavní a možná problémová místa v kódu. Na obrázku 7.1 jsou vidět metody pokryté unit testy. [Ham04]



Obrázek 7.1: Unit testy základních metod serverového řešení.

7.1.2 Funkční testy

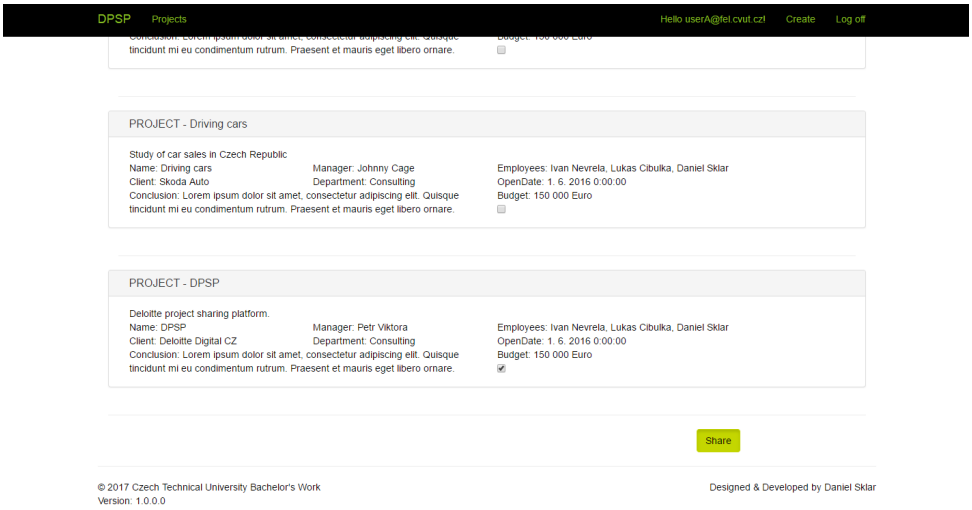
Funkční testy testují co by měl systém nebo komponenta dělat, což je typicky popsáno ve funkční specifikaci požadovaného softwaru. Tyto testy mohou být prováděny ze dvou základních perspektiv: s ohledem na **1. technické zadání** nebo **2. byznysové procesy**.[GVVE08] Vzhledem k tomu, že znám technické zadání i byznysové procesy aplikace, tak se mi funkční testy osvědčily. Tyto testy jsem prováděl pomocí navrženého a implementovaného webového klienta4.2.1.

Testování spočívá v průchodu různými částmi aplikace s očekávanými výsledky. Pro demonstraci připojuji jeden konkrétní funkční test.

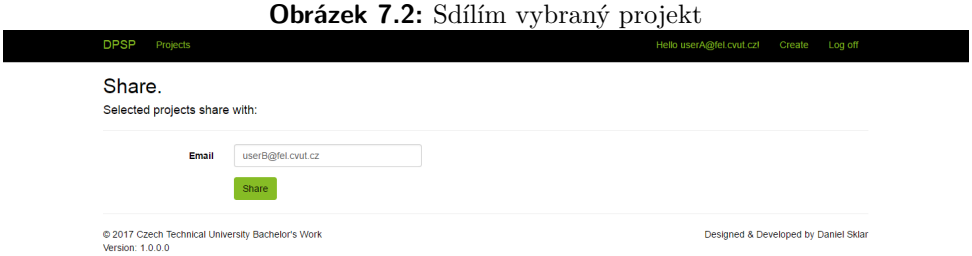
- **Zadání:** Jakožto uživatel A s rolí zaměstanec, budu sdílet projekt s uživatelem B s rolí klient, který původně žádný projekt nemá.
- **Očekávaný výstup:** Uživatel B po sdílení projektu vidí nový sdílený projekt bez citlivých údajů, tedy konkrétní částky rozpočtu.

1. Jakožto uživatel A vyberu projekt ke sdílení a stisknu tlačítko 'Share'.7.2
2. Vyberu, s kým chci projekt sdílet formou zadání emailu a sdílím stisknutím tlačítka 'Share'.7.3

3. Jakožto uživatel B zkontroluji sdílený projekt v záložce 'Projects', a to zejména citlivé údaje. 7.4



Obrázek 7.2: Sdílím vybraný projekt

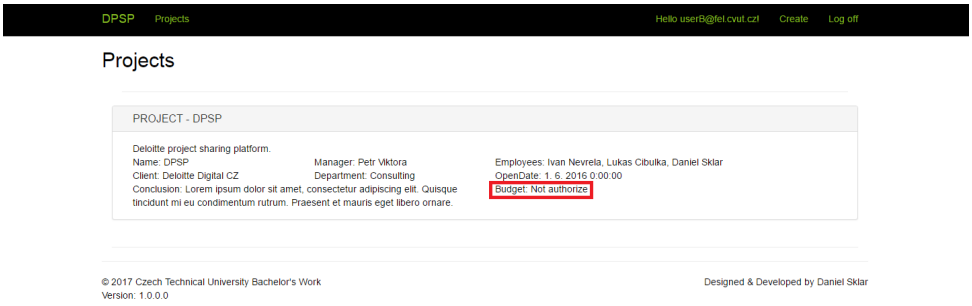


Obrázek 7.3: Vybírám uživatele, se kterým chci projekt sdílet

Funkční testy se ukázaly jako velice přínosné, protože se mi díky nim podařilo odhalit některé nedokonalosti, které jsem záhy opravil.

7.2 Dokumentace

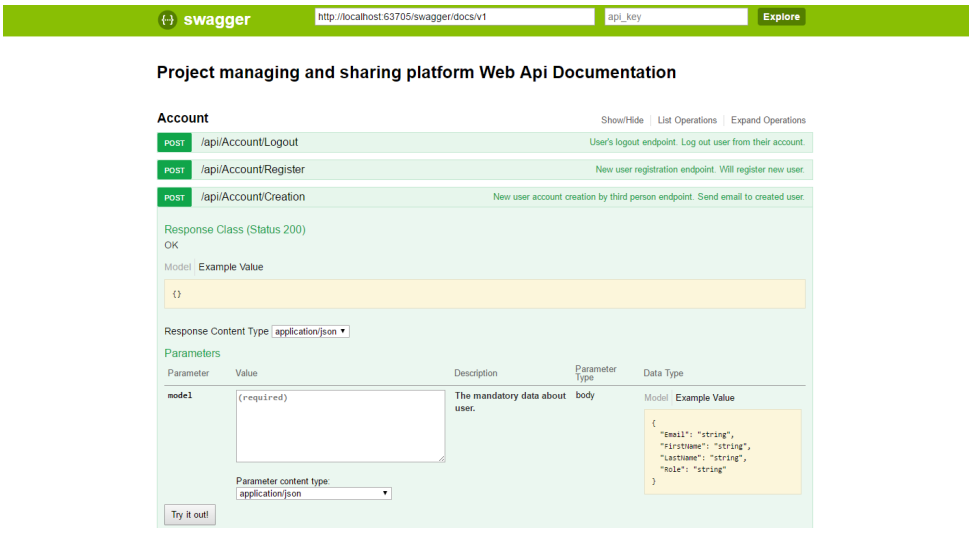
Dokumentace je připravena na úrovni programovacího jazyka a také dokumentace samotného API. Ta je vygenerováná pomocí frameworku Swagger.



Obrázek 7.4: Kontroluji sdílený projekt a uvedená citlivá data

Jedná se o framework pro dokumentaci projektů. Tento nástroj se snadno implementuje do projektu a zobrazuje mnou definovanou dokumentaci v uživatelsky přívětivé formě. Jedna z předních výhod, kterých jsem využil je automatické generování interaktivní dokumentace z anotací ve zdrojovém kódu.

Vygenerovaná dokumentace lze vidět na obrázku 7.5.



Obrázek 7.5: Dokumentace vygenerovaná frameworkem Swagger.

z

Kapitola 8

Závěr

V bakalářské práci jsem řešil problém konzultační společnosti. Tento problém byl především v používání různých technologií a přístupů ke správě, sdílení a prezentaci projektů napříč společností, což vyustilo ke zhoršení efektivity práce s projekty. Celkový příběh projektu se tak rozdělil do několika částí a různých technologií. To vedlo k neunifikované formě ukládání, sdílení a také výsledné prezentaci těchto projektů potenciálním klientům. Pro vyřešení tohoto problému bylo potřeba udělat analýzu dosud používaných technologií ve společnosti, navrhnout řešení a základ tohoto řešení implementovat. Také bylo nutné řešení vhodným způsobem otestovat, zdokumentovat a rozmyslet možnosti budoucí integrace s jinými systémy, zejména pro možnost získávání dat z ERP systému SAP a popřípadě navrhnout integraci s externím způsobem přihlašování uživatelů.

V první části bakalářské práce se věnuji analýze problému.3 Závěr analýzy je zjištění, že konzultační společnost používá příliš mnoho různých technologií a je třeba proces práce s projekty sjednotit. Tato část také zahrnuje analýzu firemní kultury, zejména co se týče implementačních technologií.

Dále se zabývám návhrem řešení této problematiky4, což vedlo v návrh serverového API, které skrz několik vrstev komunikoje s databází a vrací požadované data o projektech. Toto řešení se ukázalo vhodné především díky flexibilitě použití pro různé druhy klientských aplikací. Je tedy ekonomicky výhodné navrhnout a implementovat hlavní logickou část řešení pouze jednou a vzdáleně k ní přistupovat z různých prezentačních platforem.

Návrh řešení implementuji a rozebírám v sekci Implementace.5 Jedná se

zejména o implementaci několika vrstev serverové části a také webového klienta, který slouží hlavně pro demonstraci funkčnosti API a také jeho otestování. Tato implementace měla za úkol vytvořit základ pro budoucí rozšíření, jak serverové části, tak části klientské.

Po implementační fázi rozebírám integrační možnosti aplikace6 se zaměřením na získání dat ze systémem SAP a s usnadněním přístupu uživatelům pomocí externích autentikačních systémů. Nakonec řešení vhodně testuji a dokumentuji.

Podařilo se mi vyřešit problém, který tížil konzultační společnost pomocí analýzy, návrhu, implementace a dalších rozborů problematiky, čímž jsem také splnil zadání bakalářské práce a připravil platformu, která bude v blízké době společností reálně využívána.

Budoucí kroky leží v rozšíření funkcionality platformy s ohledem na složitost požadavků, konkrétní integraci s jinými systémy a také reálné nasazení do provozu. Platformu budou využívat zaměstnanci konzultační společnosti pro správu a sdílení projektů, včetně samotné prezentace projektů potenciálním zákazníkům.

Příloha A

Literatura

[Ars09] D. Arsenovski, *Professional Refactoring in C# & ASP.NET*, Wrox Professional Guides, Wiley, 2009.

[Aze14] A. Azevedo, *Integration of Data Mining in Business Intelligence Systems*, Advances in Business Strategy and Competitive Advantage:, IGI Global, 2014.

[Bea09] J. Bean, *SOA and Web Services Interface Design: Principles, Techniques, and Standards*, The MK/OMG Press, Elsevier Science, 2009.

[Bec03] K. Beck, *Test-driven Development: By Example*, Kent Beck signature book, Addison-Wesley, 2003.

[Cha11] J. Chadwick, *Programming Razor: Tools for Templates in ASP.NET MVC or WebMatrix*, O'Reilly Media, 2011.

[Che12] S. Cheng, *Odata Programming Cookbook for . Net Developers*, Enterprise : professional expertise distilled, Packt Pub., 2012.

[Dom13] Dominic Betts, Grigori Melnik, Fernando Simonazzi, Mani Subramanian, *Developer's guide to dependency injection using unity*, MSDN (2013), 14.

[Esp14] D. Esposito, *Programming Microsoft ASP.NET MVC*, Developer Reference, Pearson Education, 2014.

[Eva04] E. Evans, *Domain-driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley, 2004.

[Fow12] M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley Signature Series (Fowler), Pearson Education, 2012.

[FS11] A. Freeman and S. Sanderson, *Pro ASP.NET MVC 3 Framework*, Apresspod Series, Apress, 2011.

[FT00] Roy T Fielding and Richard N Taylor, *Architectural styles and the design of network-based software architectures*, University of California, Irvine Doctoral dissertation, 2000.

[Ges16] Mgr. David Gesvindr, *Úvod do Entity Frameworku*, PDF, March 2016.

[GHW+12] J. Galloway, P. Haack, B. Wilson, K.S. Allen, and S. Hanselman, *Professional ASP.NET MVC 4*, ITPro collection, Wiley, 2012.

[GT02] D. Gourley and B. Totty, *HTTP: The Definitive Guide*, Definitive Guides, O'Reilly Media, Incorporated, 2002.

[Gun15] R. Gunasundaram, *ASP.NET Web API Security Essentials*, Packt Publishing, 2015.

[GVVE08] D. Graham, E. Van Veenendaal, and I. Evans, *Foundations of Software Testing: ISTQB Certification*, Cengage Learning, 2008.

[GWR+13] J.N. Gaylord, C. Wenz, P. Rastogi, T. Miranda, S. Hanselman, and S. Hunter, *Professional ASP.NET 4.5 in C# and VB*, Wiley, 2013.

[Hal14] G.M.L. Hall, *Adaptive Code via C#: Agile coding with design patterns and SOLID principles*, Developer Reference, Pearson Education, 2014.

[Ham04] P. Hamill, *Unit Test Frameworks: Tools for High-Quality Software Development*, O'Reilly Media, 2004.

[Hor06] M. Horner, *Pro .NET 2.0 Code and Design Standards in C#*, Books for professionals by professionals, Apress, 2006.

[HT06] J. Hilyard and S. Teilhet, *C# Cookbook*, Cookbooks (o'Reilly) Series, O'Reilly Media, 2006.

[Inc16] Alphabet Inc., *Using OAuth 2.0 to Access Google APIs*, Tech. report, Google, Mountain View, Kalifornie, USA, November 2016.

[KW14] J. Kurtz and B. Wortman, *ASP.NET Web API 2: Building a REST Service from Start to Finish*, Expert's voice in ASP.NET, Apress, 2014.

[Mac11] M. MacDonald, *Pro Silverlight 4 in C#*, Books for professionals by professionals, Apress, 2011.

[Mey06] E.A. Meyer, *CSS: The Definitive Guide: The Definitive Guide*, O'Reilly Media, 2006.

[MF10] M. MacDonald and A. Freeman, *Pro ASP.NET 4 in C# 2010*, Apresspod Series, Apress, 2010.

[Mic08] Microsoft, *Introduction to Unity*, Tech. report, Microsoft, Redmont, USA, 2008.

[Mic16] ———, *Entity Framework Code First Data Annotations*, software, Microsoft, Redmont, USA, October 2016.

[MPD13] Robert McMurray, Andy Pasic, and Tom Dykstra, *External Authentication Services with ASP.NET Web API (C#)*, Microsoft Docs (2013).

[Nag16] C. Nagel, *Professional C# 6 and .NET Core 1.0*, Wiley, 2016.

[ODa15] OData, *URI Conventions (OData Version 2.0)*, Tech. report, OData, 2015.

[Pen16] W. Penberthy, *Beginning ASP.NET for Visual Studio 2015*, Programmer to programmer, Wiley, 2016.

[Red11] M. Reddy, *API Design for C++*, Elsevier Science, 2011.

[Sin15] R.R. Singh, *Mastering Entity Framework*, EBL-Schweitzer, Packt Publishing, 2015.

[SM07] M. Szpuszta and M. MacDonald, *Pro ASP.NET 2.0 in C# 2005, Special Edition*, Apress, 2007.

[Smi15] B. Smith, *Beginning JSON*, Expert's voice in Web development, Apress, 2015.

[Spa13] M. Spasovski, *OAuth 2.0 Identity and Access Management Patterns*, Community experience distilled, Packt Publishing, 2013.

[Spu13] J. Spurlock, *Bootstrap: Responsive Web Development*, O'Reilly Media, 2013.

[TJAk12] B. Tomáš, V. Jiří, B. Alena, and a kolektiv, *Tvorba informačních systémů: Principy, metodiky, architektury*, Grada Publishing a.s., 2012.

[UZ13] T. Ugurlu and A. Zeitler, *Pro ASP.NET Web API: HTTP Web Services in ASP.NET*, The expert's Voice in .NET, Apress, 2013.

[Uza16] S. Uzayr, *Learning WordPress REST API*, Packt Publishing, 2016.

[UZK13] A. Uurlu, A. Zeitler, and A. Kheyrollahi, *Pro ASP.NET Web API: HTTP Web Services in ASP.NET*, Expert's voice in .NET, Apress, 2013.

[Wal04] S. Walther, *ASP.NET Unleashed*, Unleashed Series, Sams, 2004.

[WPD14] Mike Wasson, Andy Pasic, and Tom Dykstra, *Secure a Web API with Individual Accounts and Local Login in ASP.NET Web API 2.2*, docs.microsoft (2014).

[WPR10] J. Webber, S. Parastatidis, and I. Robinson, *REST in Practice: Hypermedia and Systems Architecture*, Theory in practice series, O'Reilly Media, 2010.

DPSP_navrhTextu3České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Sklář Daniel**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Platforma pro správu a sdílení projektů**

Pokyny pro vypracování:

1. Analyzujte požadavky na systém pro správu projektů, včetně analýzy dosud používaných aplikací v organizaci.
2. Navrhněte a implementujte platformu pro správu projektů s ohledem na provedenou analýzu.
3. Ověřte správnou funkčnost platformy pomocí testů a vhodně zdokumentujte její API.
4. Prozkoumejte a popište možnosti integrace vytvořené platformy se stávajícími enterprise systémy (SAP, LDAP či SSO).

Seznam odborné literatury:

[1] Evans Eric - Domain-Driven Design: Tackling Complexity in the Heart of Software - Addison-Wesley Professional; 1 edition (August 30, 2003)
[2] Erl Thomas - Service-Oriented Architecture (SOA): Concepts, Technology, and Design

Vedoucí: Ing. Martin Ledvinka

Platnost zadání: do konce letního semestru 2017/2018

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 21. 2. 2017