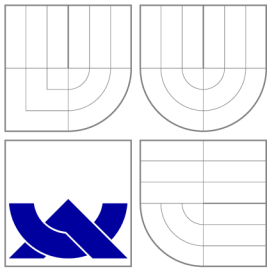
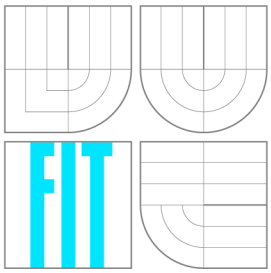


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PROTOTYP INFORMAČNÍHO SYSTÉMU PRO EXEKUTORSKÉ ÚŘADY

A PROTOTYPE OF INFORMATION SYSTEM FOR BAILIFF OFFICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. DUŠAN BEZDĚK

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. ALEŠ SMRČKA, Ph.D.

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav inteligentních systémů Akademický rok 2015/2016

Zadání diplomové práce

Řešitel: **Bezděk Dušan, Bc.**
Obor: Informační systémy
Téma: **Prototyp informačního systému pro exekutorské úřady**
A Prototype of Information System for Executor Offices
Kategorie: Informační systémy

Pokyny:

1. Nastudujte metodiky vývoje informačních systémů s webovým rozhraním (např. TDD/BDD, HTML5, CSS). Zaměřte se na vývoj v prostředí Microsoft .NET v jazyce C#. Nastudujte tvorbu moderních webových aplikací.
2. Analyzujte požadavky exekutorských úřadů na informační systém (IS). Navrhněte IS pro exekutorské úřady. Informační systém bude zahrnovat správu spisů ex. úřadu, evidenci stran (osob) ve spisu, správu dokumentů, implementaci workflow spisů, evidenci třetích stran a součinností. Při návrhu dávejte důraz na přívětivé webové uživatelské rozhraní.
3. Implementujte prototyp vámi navrženého systému v prostředí MS .NET.
4. Ověřte prototypovou implementaci pomocí automatizovaných jednotkových testů pokrývající všechny klíčové části systému. Proveďte systémové nebo akceptační testy informačního systému.

Literatura:

1. Solis, C.; Xiaofeng Wang; A Study of the Characteristics of Behaviour Driven Development. In Proc. of SEAA 2011. 383-387. doi: 10.1109/SEAA.2011.76
2. Testování GUI v MS Visual Studio 2012. Online URL: https://msdn.microsoft.com/en-us/library/dd286726(v=vs.110).aspx#VerifyingCodeUsingCUITCreate

Při obhajobě semestrální části projektu je požadováno:

- První bod zadání a část návrhu systému.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrčka Aleš, Ing., Ph.D.**, UITS FIT VUT
Konzultant: Najvárek Jan, Ing., Ph.D., ARTIN
Datum zadání: 1. listopadu 2015
Datum odevzdání: 25. května 2016

doc. Dr. Ing. Petr Hanáček
vedoucí ústavu

Abstrakt

Tato práce se zabývá analýzou, návrhem a implementací informačního systému pro exekutorské úřady. Úkolem tohoto systému je správa exekučních spisů, jejich subjektů, příslušejících dokumentů, finančních náležitostí a evidence součinností. Systém dále umožňuje komunikaci s datovými schránkami pro odesílání a přijímání pošty a provádí lustraci subjektů v administrativním registru ekonomických subjektů. Navíc implementuje možnost multitenance, díky které je aplikace schopna poskytovat služby více úřadům. Informační systém je realizován jako webová aplikace pod technologií ASP.NET MVC v jazyce C#. Klientskou stranu aplikace tvoří AngularJS a Bootstrap framework.

Abstract

This master's thesis deals with analysis, design and implementation of information system for bailiff office. The system provides functionality for managing bailiff's files, their subjects, appropriate documents, financial requirements and cooperation with third parties. Moreover, the system enables communication with data boxes for receiving and sending messages and performing vetting in registers of economic subjects. The application implements also multitenant architecture, which means the system can serve multiple offices. The information system is realised as a web application in ASP.NET MVC technology and written in C# language. The client side of the application is composed of AngularJS and Bootstrap frameworks.

Klíčová slova

Informační systém, exekutorský úřad, datové schránky, lustrace, MS.NET, ASP.NET MVC, C#, AngularJS, Bootstrap framework, multitenantní aplikace, testy řízený vývoj, jednotkové testy.

Keywords

Information system, bailiff office, data boxes, vetting, MS.NET, ASP.NET MVC, C#, AngularJS, Boostrap framework, multitenant application, test-driven development, unit tests.

Citace

BEZDĚK, Dušan. *Prototyp informačního systému pro exekutorské úřady.* Brno, 2016. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Smrčka Aleš.

Prototyp informačního systému pro exekutorské úřady

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Aleše Smrčky, Ph.D. Další informace mi poskytli pan Bc. Lukáš Landsmann a pan Ing. Jan Najvárek, Ph.D. Uvedl jsem zde všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Dušan Bezděk
24. května 2016

Poděkování

Chtěl bych tímto poděkovat externímu zadavateli této práce, společnosti ARTIN, spol. s.r.o, že mi nabídla a umožnila pracovat na této práci. Především pak panu Bc. Lukáši Landsmannovi za jeho odbornou pomoc a vedení práce. Poděkování si zaslouží také vedoucí této práce pan Ing. Aleš Smrčka, Ph.D. za poskytnutí spousty cenných rad během její tvorby. Nemůžu však zapomenout poděkovat také svým rodičům za podporu během studia a své přítelkyni za velkou dávku trpělivosti.

Obsah

Kapitola 1

Úvod

Cílem této práce je vytvoření informačního systému pro exekutorské úřady. Jedná se o prototypovou implementaci webové aplikace, která nabídne řešení pro agendu soudních exekutorů. Tento software je zaměřen na menší exekutorské úřady o několika málo zaměstnancích. Jeho primárním úkolem je zjednodušit agendu úřadu a umožnit elektronickou komunikaci s dalšími institucemi. Aplikace umožňuje provádět správu exekučních spisů, jejich subjektů, příslušejících úředních dokumentů, finančních náležitostí a evidovat součinnost. Navíc nabízí možnost komunikovat s datovými schránkami pro odesílání a přijímání pošty a také provádět lustraci v administrativním registru ekonomických subjektů. Samozřejmostí je také autentizace a řízení přístupu pro jednotlivé uživatele.

Hlavním přínosem aplikace je integrace stanovených potřeb exekutorského úřadu do jednoho systému. Důsledkem toho je snížení potřebných administrativních činností, usnadnění dohledávání souvisejících údajů a také částečná automatizace úkonů. S tím souvisí také zrychlení provedení práce. Další důležitou vlastností je poskytnutí příjemného a efektivního prostředí, které bude na první pohled pochopitelné a nebude vyžadovat rozsáhlou uživatelskou příručku. Aplikace přináší výhodu také poskytovateli služeb, který díky implementaci multitenance může v rámci jedné nasazené aplikace poskytovat služby více úřadům.

Tato práce postupně seznámí čtenáře s vývojem popsané aplikace. Nejprve je v práci nastíněna kapitola teorie shrnující především technologie využité pro tvorbu aplikace. Je zde také popsána charakteristika obecného informačního systému, metodika testy řízeného vývoje a stručně shrnuta problematika exekutorského úřadu. Další kapitola se zabývá analýzou požadavků informačního systému pro exekutorské úřady, kde jsou pokryty potřebné části pro vytvoření aplikace a sepsány případy užití systému. Následuje kapitola návrhu aplikace, jenž seznamuje s částmi navrženého systému, návrhem schématu databáze a návrhem grafického uživatelského rozhraní. Kapitola implementace navrženého systému rozebírá použitou vícevrstvou architekturu a strukturu serverové a klientské části aplikace. Zároveň se také věnuje implementačním detailům důležitých částí aplikace, prvkům zabezpečení systému a vytvořenému testovatelnému kódu. Následující kapitola obsahuje popis testování vytvořeného systému pomocí automatizovaných a manuálních testů a závěrečná kapitola shrnuje dosažené výsledky této práce.

Kapitola 2

Teorie pro vývoj informačního systému

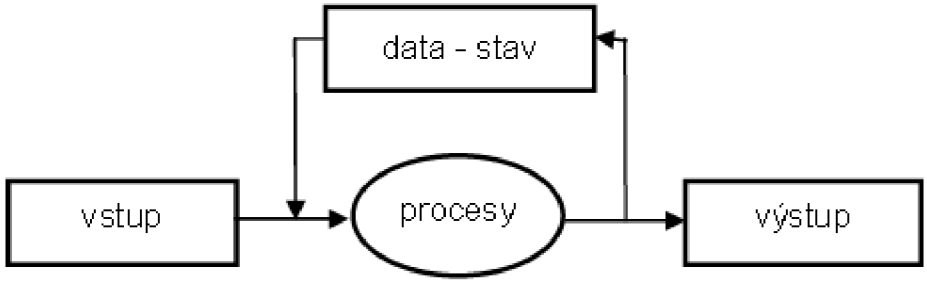
Tato kapitola popisuje pojmy a technologie použité v rámci vytvářené práce. Je zde nastíněna charakteristika informačního systému, platformy Microsoft .NET a tvorba webových aplikací pro ASP.NET framework. Kapitola se věnuje také architektuře ASP.NET MVC a tzv. jednostránkovým aplikacím. Dále jsou zde umístěny sekce, které popisují klientské části aplikace, tedy AngularJS framework a Bootstrap framework. Následují podkapitoly věnující se pojmu multitenantní aplikace a metodice testy řízeného vývoje. Kapitola popisuje také základní úrovně testování softwaru a testovací dvojníky použitých při jednotkovém testování. Na závěr je shrnuta problematika exekutorského úřadu a existující řešení informačních systému pro exekutorský úřad.

2.1 Charakteristika informačního systému

Samotný pojem informační systém lze chápat jako systém pracující s informacemi [5]. Skládá se ze vstupní části, kde vstupují data do systému a výstupní části, která slouží k zobrazení dat, jež uživatel interpretuje jako informace. Mezi těmito body vstupu a výstupu probíhají transformace nad daty, typicky pomocí různých algoritmů. Probíhají zde procesy, a je tedy zapotřebí zabývat se způsobem definice procesů a jejich vzájemnou interakcí. Důležitou součástí je také zpětná vazba, která využívá uloženého stavu systému. Výsledky a výstupy procesů nemusí záviset na jednotlivých vstupech, ale typicky na stavu systému. Důležitými aspekty informačního systému jsou tedy jeho stav, vyjádřený pomocí uchovávaných dat, a procesy, které často ve formě transakcí realizují transformace nad daty. Tento popis je zobrazen pomocí schématu na obrázku 2.1.

Informační systém je provozován s vazbami na své okolí, jedná se tedy o systém otevřený. Znamená to, že se mu dostává dat na vstupu a zároveň jsou jeho data odebírána na výstupu. Informační systém obvykle modeluje nějaký fyzický systém, tzn. je virtuálním obrazem nějakého skutečného systému. Jelikož není možné modelovat veškeré procesy a zdroje fyzického systému, je informační systém jistou abstrakcí svého skutečného vzoru. Vybírají se pouze takové aspekty, které jsou podstatné pro úroveň řízení, pro kterou systém budujeme.

Pro modelování skutečné činnosti fyzického systému a pro svou otevřenost, musí informační systém obsahovat dvě základní části, a to modelovací a komunikační prostředí. Modelovací prostředí slouží k modelování dat a procesů. Nejběžnějším modelovacím prostře-



Obrázek 2.1: Schéma informačního systému [5].

dím stavu modelu je databáze, neboť poskytuje persistentní (v čase přetrvávající) prostředí pro udržování stavu systému a zároveň dostatečnou kapacitu. Takovým typem systému pro řízení báze dat je nejčastěji relační model databáze. Naopak modelovací prostředí pro procesy bývá kompilovaný nebo interpretovaný programovací jazyk. Důležitými aspekty při modelování procesů jsou udržování konzistentního stavu systému, transakční zpracování a vzájemné ovlivňování paralelně běžících procesů. Otevřenost systému vůči okolí je zajištěna pomocí komunikačního prostředí přes kontaktní body. Takovými kontaktními body mohou být různá čidla, měřící zařízení, automatické ovladače (tzv. automatické kontaktní body) nebo různé typy displejů a jiných kontaktních zařízení ovládaných lidmi (tzv. manuální kontaktní body). Důležitým bodem je také prezentace dat, a to jakým způsobem je uživateli předán stav modelu. Kvalitní a přehledné zobrazení složité informace je velmi užitečné pro koncového uživatele. Samozřejmostí je i fakt, že kontaktní body jsou obvykle připojeny k počítačové síti a proto je nutné řešit také problém týkající se bezpečnosti.

2.1.1 Komponenty informačního systému

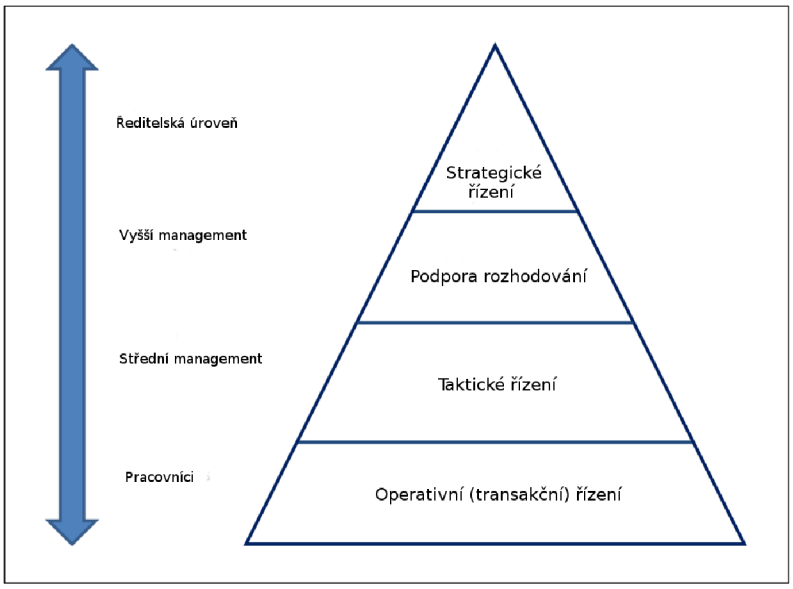
Pokud budeme popisovat informační systém na nižší úrovni abstrakce, bude se systém skládat z následujících částí [14].

- **Hardware:** zařízení jako monitor, procesor, tiskárna, klávesnice, z nichž všechny se podílejí na příjmu, zpracování a zobrazení informací.
- **Software:** veškeré programy, jenž povolují hardwaru zpracovat data.
- **Databáze:** persistentní uchování souborů či tabulek obsahující související data.
- **Síť:** umožňuje počítačům distribuovat své zdroje.
- **Procesy:** popisují, jak jsou data zpracována a analyzována k dosažení cíle, ke kterému byl informační systém navržen.

První čtyři komponenty vytváří platformu informační technologie. Tyto komponenty mohou být použity pro vytvoření různých systémů. Naopak procesy jsou specifické pro určité informace a pevně spojeny se systémem, kterému náleží.

2.1.2 Typy informačních systémů

V současnosti existuje nepřeberné množství typů informačních systémů, se kterými se můžeme setkat. Některé systémy slouží pro podporu částí organizace, jiné naopak tvoří základ pro celou organizaci a další dokonce sdružují skupiny organizací. Základní rozdělení systémů však vychází z tzv. pyramidy systémů 2.2, která odráží hierarchii rozhodování v organizaci. Na dně pyramidy se nachází systémy pro transakční zpracování, nad nimiž se vyskytují systémy pro podporu řízení, rozhodování a exekutivu. Pojmem informační systémy pro podporu řízení lze chápat obecně skupinu systémů pro efektivní řízení organizace, tedy k automatizaci a podpoře rozhodování (v pyramidě vrchní tři vrstvy). Od doby, kdy byl pyramidový model formulován, však byla vytvořena spousta nových technologií a kategorií informačních systémů, z nichž některé nelze jednoduše do původního pyramidového modelu přiřadit.



Obrázek 2.2: Pyramidový model informačních systémů [14].

2.2 ASP.NET framework

ASP.NET je serverový webový aplikační framework určený pro vývoj dynamických webových stránek, webových aplikací a webových služeb [7]. Je vytvořen firmou Microsoft a jeho zdrojový kód je volně dostupný (open source). ASP.NET je vystavěn na platformě .NET framework, takže veškeré funkce .NET framework jsou dostupné také v ASP.NET aplikacích. Tyto aplikace mohou být tedy napsány v programovacím jazyce, který je kompatibilní s Common Language Runtime (CLR) prostředím, včetně jazyků Visual Basic a C#. ASP.NET nabízí další aplikační rámce pro vývoj webových aplikací:

- **Web Forms:** využívá model řízený událostmi, kde webová aplikace je tvořena ovládacími prvky a jejich událostmi. Odděluje HTML od serverového kódu. Vývoj je podobný jako pro platformu Windows Forms.
- **Web Pages:** kombinuje serverový kód s HTML pro vytvoření dynamického webového obsahu.

- **MVC:** respektuje návrhový vzor MVC (model-view-controller), funkcionalita je tedy náležitě rozprostřena mezi komponenty modelu, pohledu a ovladače.
- **Single Pages:** moderní technika, česky zvaná jako jednostránková aplikace, využívá HTML5 a AJAX pro pohotově reagující aplikaci bez neustálého znovunačítání celých stránek. Většina práce leží na klientské straně v JavaScriptu. Serverová strana je implementována pomocí Web API.
- **Web API:** implementuje REST rozhraní pro poskytování a zpracování dat ve formátu XML nebo JSON. Ideální pro dotazování z klienta pomocí JavaScriptu.
- **SignalR:** slouží ke komunikaci v reálném čase a poskytuje obousměrnou komunikaci mezi klientem a serverem. Server je tedy schopný poslat obsah připojenému klientovi, jakmile se klient stává dostupným.

Aplikace vytvořená v této práci kombinuje přístupy architektury MVC (použitým aplikačním rámcem bude tedy ASP.NET MVC) s jednostránkovou technologií. Obě tyto technologie budou více popsány dále.

2.2.1 Platforma .NET

Jak již bylo výše zmíněno, ASP.NET platforma pro webové aplikace je postavena na platformě .NET. Tato platforma stojí na dvou pilířích, jimiž jsou běhové prostředí pro vykonávání kódu, tzv. Common Language Runtime (CLR) a knihovna tříd zvaná Framework Class Library (FCL), obsahující knihovny jako Windows Forms, WPF, ASP.NET, ADO.NET a mnoho dalších [4].

CLR je aplikační virtuální stroj (obdoba Java VM pro .NET), která mimo jiné poskytuje služby jako správu paměti, načítání knihoven, bezpečnostní služby a zachytávání výjimek. Poskytuje také jazykovou interoperabilitu, tzn. podporu vývoje napříč několika programovacími jazyky (C#, VB, Visual C++, Delphi .NET, F#).

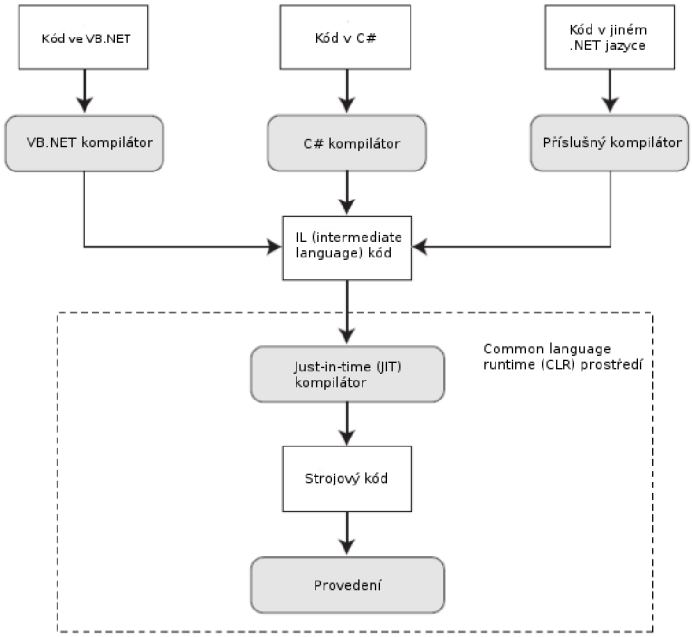
Co se týče samotné kompilace C# kódu, ten je nejprve kompilován do tzv. managed kódu, jenž je zabalen do assembly, která je dvou typů. Buď je to spustitelný kód s příponou .exe, nebo knihovny s příponou .dll. Intermediate language (IL) reprezentuje managed kód a při čtení CLR je kód z assembly konvertován do nativního kódu stroje (x86). Tento proces transformace realizuje Just-in-Time (JIT) kompilátor. Pro lepší představu je kompilace kódu zobrazena na obrázku 2.3.

Platforma .NET je primárně navržena pro běh na prostředí Windows, nicméně existují multiplatformní projekty postavené na tomto aplikačním rámci jako např. projekt Mono, který nepoužívá běhové prostředí CLR, ale má vlastní kompilátor. Další náznaky multiplatformnosti přináší modulární verze .NET rámce zvaná .NET Core, poskytující podporu pro Windows, Linux a Mac OsX, která má navíc otevřený zdrojový kód.

2.2.2 ASP.NET MVC

ASP.NET MVC [8] je framework obsažený uvnitř balíku ASP.NET, který implementuje architektonický vzor MVC (model-view-controller), jenž se skládá z následujících komponent:

- **Model:** objekty uvnitř modelu jsou části aplikace, které implementují logiku pro aplikační datovou doménu. Obvykle se tyto objekty starají o obdržení a úpravu dat z databáze.



Obrázek 2.3: CLR kompilace [4].

- **Pohled:** pohledy jsou části které zobrazují uživatelské rozhraní. Typicky je pohled vytvořen na základě datového modelu.
- **Ovladač:** ovladače jsou komponenty, které řeší uživatelské vstupy, pracují s modelem a vybírají pohled, který se uživateli zobrazí.

Návrhový vzor MVC pomáhá vytvořit aplikaci, která odděluje odlišné aspekty aplikace (vstupní logiku, funkční logiku a UI logiku) a zároveň poskytuje volné propojení těchto elementů. Vzor také vymezuje, kam každý druh logiky v aplikaci patří. Vstupní logika náleží ovladači, aplikační logika patří modelu a UI logika je spjata s pohledem. Takové oddělení pomáhá vývoji rozsáhlých projektů, protože umožňuje soustředit se na konkrétní aspekt během implementace a zároveň napomáhá vývoji v týmu, jelikož každý člen týmu může pracovat na své komponentě. Další výhodou tohoto přístupu je možnost testování jednotlivých částí aplikace. Díky odlučitelnosti komponent a častému užití rozhraní je možné testovat elementy izolovaně od ostatních částí aplikačního rámce.

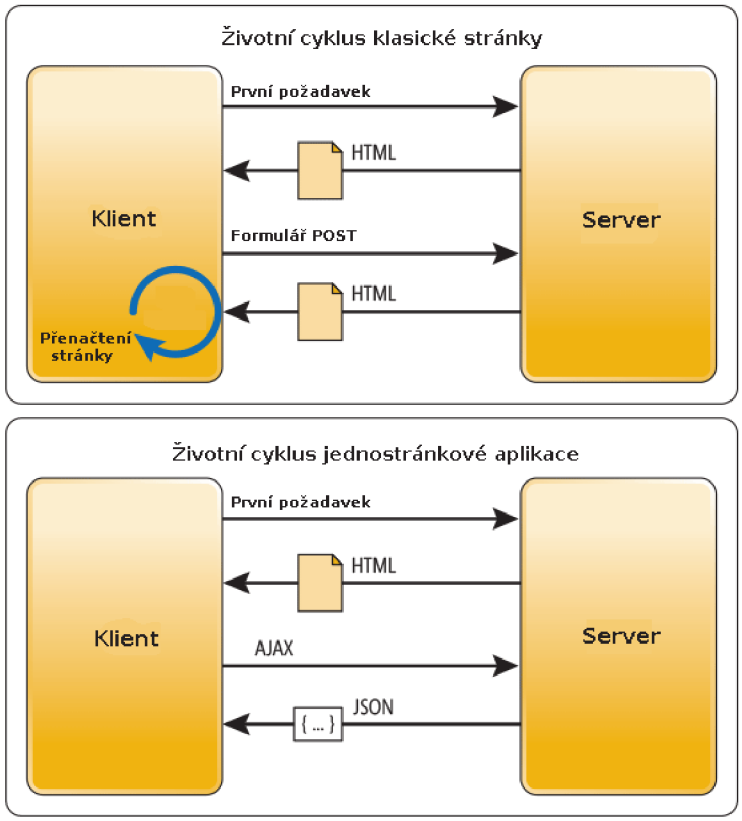
2.2.3 Jednostránková aplikace

Pojem jednostránková aplikace (single page application - SPA) [16] označuje webovou aplikaci, která zobrazí jedinou HTML stránku a dynamicky ji upravuje na základě uživatelské interakce. Základním stavebním kamenem aplikace se stává klientský JavaScript, který pomocí zasílání asynchronních dotazů (asynchronous JavaScript and XML - AJAX) získává data ze serveru a upravuje stránku pouze tam, kde se přijatá data změnila.

U běžné webové aplikace se při každém volání serveru vyrenderuje na straně serveru nová HTML stránka, a to způsobí obnovení celé stránky na straně klienta. U jednostránkové aplikace se stránka načte pouze jednou a poté je veškerá komunikace se serverem prováděna pomocí volání AJAX zpráv. Výsledkem volání jsou pouze data ve formátu JSON nebo

XML (nikoliv celá stránka), které slouží k dynamické úpravě části stránky bez jejího celého překreslení.

Výhoda takového přístupu je zřejmá. Aplikace pohotověji reaguje bez efektu obnovení celé stránky. Další výhodou může být oddělení prezentační (HTML) a aplikační logiky (AJAX dotazy a JSON odpovědi). To vede ke stejným možnostem popsaným u MVC architektury, a to úpravě konkrétní části bez dotčení se jiné komponenty aplikace. Server se v tomto případě chová jako služba, která pouze poskytuje data. Ideální pro tento přistup je vytvoření Web API aplikace na straně serveru. Klient poté musí znát pouze formát HTTP dotazů pro server.



Obrázek 2.4: Srovnání životních cyklů běžné a single page webové aplikace [16].

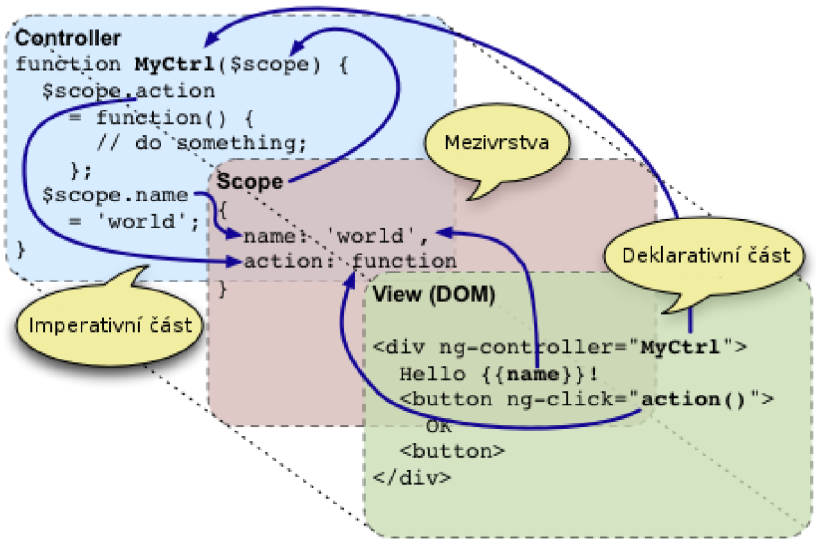
2.3 AngularJS framework

AngularJS je otevřený aplikační framework postavený na programovacím jazyce JavaScript, jenž je z velké části vyvíjen nejen společností Google, ale i otevřenou vývojářskou komunitou [1]. Cílem aplikačního rámce je zjednodušení vývoje a testování klientské části webových aplikací. Zaměřuje se především na jednostránkové aplikace.

Po mnoho let byl AngularJS blíže k architektuře MVC (na klientské straně), ale časem se díky mnoha úpravám a vylepšením přiblížil spíše vzoru MVVM (Model-View-ViewModel). Vzhledem k tomu, že oba architektonické prvky jsou v aplikačním rámci zastoupeny, bylo přistoupeno k označení vzoru MVW (Model-View-Whatever), kde slovíčko whatever zastupuje myšlenku co se vám zde hodí.

Základní myšlenkou aplikačního rámce je oddělení prezentační logiky od logiky aplikační. Tvrdí, že uživatelské rozhraní a jeho spojení s dalšími komponentami by mělo být tvořeno deklarativním programováním, zatímco aplikační logika programováním imperativním. Framework rozšířil HTML stránku o tzv. direktivy, které slouží k zobrazení dynamického obsahu za pomoci dvoucestné synchronizace dat (two-way data-binding), který umožňuje automatickou synchronizaci mezi modely a pohledy. AngularJS implementuje vzor MVC k oddělení prezentace, dat a logiky a s použitím vložení závislosti přináší tradiční serverové služby na stranu klienta. Následkem toho může být redukce zátěže na straně serveru.

Zajímavou novinkou je již zmíněná dvoucestná synchronizace dat. Díky ní je v HTML vyrenderována šablona podle dat obsažených v komponentě zvané rámec (scope). Tato část detekuje změny v modelové části a okamžitě modifikuje HTML pohled přes ovladač. Tento proces funguje i opačným směrem, tzn. jakákoliv změna na stránce je propagována do modelu. Tato technika zabraňuje nutnosti neustálé manipulaci s DOM objektem a vede k rychlému vývoji webových aplikací.



Obrázek 2.5: Struktura frameworku AngularJS [17].

Tento framework tvoří společně s aplikačním rámcem Bootstrap (popsaným níže) základ pro klientskou stranu vytvořené aplikace. Díky těmto částem je aplikace ze strany klienta platformně nezávislá, neboť obě tyto technologie jsou dostupné u většiny používaných internetových prohlížečů.

2.4 Bootstrap framework

Bootstrap je otevřený framework určený pro klientskou část aplikací, který je vytvořen týmem společnosti Twitter [2]. Jedná se o kombinaci HTML, CSS a JavaScript kódu vytvořenou pro rychlý vývoj uživatelského rozhraní a jeho komponent.

Bootstrap je kompatibilní s nejnovějšími verzemi prohlížečů, ačkoliv některé nejsou podporovány na všech platformách. Další vlastností aplikačního rámce je jeho podpora pro interaktivní webový design. Znamená to, že rozvržení webových stránek se mění dynamicky

v závislosti na použitém zařízení a velikosti obrazovky.

Bootstrap je modulární a skládá se z kolekce tzv. LESS stylových souborů (stylesheets), které implementují různé komponenty aplikačního rámce. Díky tomu mají vývojáři možnost si přizpůsobit tyto komponenty podle svých požadavků a podle využití ve svých aplikacích. Bootstrap poskytuje soubor stylových dokumentů, jenž obsahují definici stylů pro nejrůznější HTML komponenty, díky kterým je dosaženo sjednoceného a moderního vzhledu pro formátování textu, tabulek a dalších HTML elementů. Tímto způsobem se z jednotlivých částí vzhledu stávají znovupoužitelné komponenty. Navíc je takové elementy možno sdružovat do skupin, navigačních lišt, navigačních seznamů a podobně. Zároveň je v aplikačním rámci obsaženo také několik komponent z knihovny jQuery, které buďto obohacují množinu použitelných elementů, nebo rozšiřují jejich funkcionalitu.

Tento framework tvoří základ grafického uživatelského rozhraní naší aplikace. Díky němu jsou vytvořeny veškeré grafické komponenty, se kterými systém disponuje a zároveň umožňuje změnu rozvržení vzhledu v závislosti na velikosti obrazovky.

2.5 Multitenantní aplikace

Pojem multitenance [6] souvisí s cloud službami, který patří mezi základní technologie modelu zvaného cloud computing. Multitenantní je taková architektura, ve které jedna instance softwarové aplikace slouží více zákazníkům, přičemž každý zákazník se označuje jako uživatel (tenant). Jednotlivým uživatelům je možné nabídnout přizpůsobení si některé části aplikace, jako schéma barev grafického uživatelského rozhraní, ale nemohou si přizpůsobit funkčnost aplikace.

V technologii cloud computing se význam pojmu multitenance rozšířil díky využívání virtualizace a vzdáleného přístupu v rámci nových modelů služeb. Poskytovatel softwaru (označení software as a service - SaaS) nechá běžet pouze jedinou instanci webové aplikace a poskytne přístup více zákazníkům, přičemž data každého uživatele musí být izolovaná a musí být viditelná pouze těm, kterým tato data náleží. Existují tři základní přístupy, jak spravovat data v rámci multitenantní aplikace:

- **Oddělené databáze:** každý uživatel má vlastní data, která jsou izolovaná od dat náležících jiným uživatelům. Metadata spojují každou databázi s příslušným uživatelem a oddělení databází ochraňuje uživatele od úmyslného či neúmyslného dosažení cizích dat. Oddělení databází navíc umožňuje snadnější rozšíření datového modelu k dosažení zákazníkových potřeb a snadnější obnovení dat ze zálohy. Zřízení vlastní databáze pro data každého uživatele však stojí větší hardwarové nároky, proto je tento přístup realizován pro zákazníky s vyššími nároky na bezpečnost dat.

- **Sdílená databáze, oddělená schémata:** další přistup zahrnuje hostování více zákazníků v rámci jedné databáze, kde každý zákazník má svou množinu tabulek, jež jsou shlukovány do schématu vytvořeného speciálně pro zákazníka. Stejně jako izolovaný přistup, je snadnější ho implementovat a rozšířit podle zákazníkových požadavků. Navíc je vhodnější pro vyšší počet zákazníků než u oddělených databází. Tento přístup však izoluje data uživatelů na logické úrovni a velký problém vzniká u obnovení zálohy dat.

- **Sdílená databáze i schémata:** poslední přístup zahrnuje sdílení databáze i schématu, každá tabulka tedy obsahuje záznamy různých uživatelů a tedy každý záznam je svázán se svým uživatelem pomocí identifikátoru uživatele. Tento přístup má nejnižší

nároky na hardware a zálohu, ale naopak vyžaduje úsilí navíc v otázce implementace bezpečnosti k zamezení přístupu k cizím datům, především v případě neočekávaných chyb nebo útoku. Přístup je vhodný pro velké množství uživatelů, kteří upřednostní nižší cenu za službu na úkor bezpečnosti.

Každý přístup má své klady a zápory, záleží tedy především na počtu a potřebách uživatelů, kteří budou aplikaci využívat. Multitenance může poskytnout ekonomické výhody, neboť vývoj softwaru a cena údržby jsou sdílené, tzn. poskytovatel služby aplikuje záplatu či vylepšení pouze na jedinou instanci softwaru. Poskytováním takových služeb se však od poskytovatele očekává adekvátní úroveň bezpečnosti, robustnosti a výkonu.

Informační systém vytvářený v rámci této práce implementuje, vzhledem k citlivosti ukládaných dat, přístup oddělených databází pro každý exekutorský úřad. Podrobnější informace budou popsány v kapitole návrhu řešení.

2.6 Testy řízený vývoj

Mezi moderní metodiky vývoje softwaru se řadí také metodika testy řízeného vývoje (test-driven development - TDD). Tato technika patří do větve agilních metodik, které vychází z techniky zvané extrémní programování. Často je však využívána také jinými agilními metodikami, jako například metodikou Scrum. Při tvorbě této práce, především během fáze implementace, bude snaha vyzkoušet si a aplikovat metodiku testy řízeného vývoje.

Základem této techniky je krátký vývojový cyklus založený na automatizovaném jednotkovém testování. V každé iteraci napíše vývojář nejprve testy, které definují požadovanou funkcionalitu a až poté se věnuje samotné tvorbě kódu implementující dané vlastnosti. Poté, kdy testy skončí úspěchem, dochází k refaktorizaci kódu, aby kód vyhovoval přijatelným standardům. Takový proces vede především k jednoduchému návrhu. Technika vychází z agilní vývojové metodologie extrémního programování.

Jednotlivé etapy testy řízeného vývoje jsou následující [15]:

- **Přidat test:** každá nová funkcionalita začíná napsáním testu. Aby byl vývojář test vůbec napsat, musí být plně seznámen s se specifikací a požadavky na funkcionalitu.
- **Spustit testy a ověřit, že nový test neprošel:** tento krok ujišťuje, že napsaný test není náhodou splněn, což by znehodnotilo vytvoření takového testu.
- **Napsat kód:** v této fázi je nutné napsat kód, který úspěšně projde napsaným testem. Tento kód nemusí být perfektní ani konečný, neboť bude nadále upraven v následujících krocích. Cílem fáze není napsat čistý kód, nýbrž kód, jenž projde testem.
- **Spustit testy:** pokud v této fázi projdou veškeré testy, vývojář si může být jist, že kód splňuje specifikaci, testovací požadavky a negativně neovlivnil již dříve napsanou funkčnost. Do další fáze se lze přesunout pouze pokud testy úspěšně projdou.
- **Refaktorovat kód:** poslední etapa slouží k vyčištění a vylepšení napsaného kódu. Spuštěním testů se programátor ujistí, že refaktorizace kódu nijak neporušila fungující implementaci.
- **Opakovat:** pro každý nový test se celý cyklus opakuje.

Psaním testů před samotnou implementací funkčnosti garantuje okamžitou zpětnou vazbu po každé změně kódu. Dosažení požadovaného výsledku v každém kroku vede ke zvýšení produktivity a ke zkrácení doby vývoje produktu.

2.7 Testování softwaru

Před tím, než je softwarový produkt vypuštěn do světa, prochází procesem testování pro ověření, že se vytvořený produkt chová zamýšleným způsobem, splňuje potřebné požadavky z pohledu zákazníka nebo návrhu aplikace, zda je použitelný, obsahuje co nejméně chyb atd.

2.7.1 Základní úrovně testování

Existuje spousta typů testů a strategií, jak softwarový produkt otestovat, avšak v dalším textu budou zmíněny pouze čtyři základní úrovně testování [12]:

- **Jednotkové testování:** jednotkové testy ověřují správnou funkcionalitu určité sekce kódu. Tyto testy jsou obvykle tvořeny vývojáři při psaní kódu k ujištění se, že právě vytvořená část funguje správně. Pro jednu funkci může být napsáno více testů, které ověřují správnost jednotlivých větví uvnitř, případně chování v mezních případech. Velkou výhodou těchto testů je jejich rychlost provedení, tedy po každé změně kódu poskytnou testy okamžitou zpětnou vazbu, zda upravený blok funguje správně.

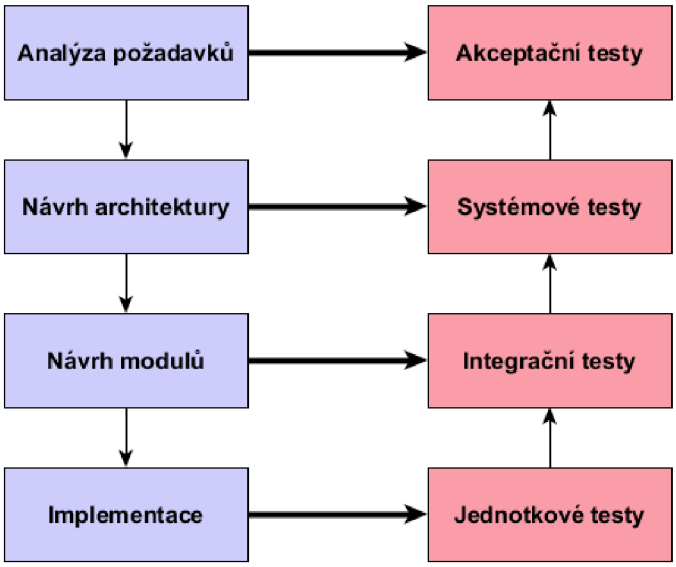
- **Integrační testování:** integrační testy netestují pouze jeden fragment kódu, jako je tomu při jednotkovém testování, ale vzájemnou spolupráci mezi komponentami systému. Cílem tohoto typu testování je odhalit chyby rozhraní modulů systému a zjistit, jak dobře fungují moduly společně. Postupně jsou mezi sebou integrovány a testovány větší a větší skupiny komponent, dokud software nepracuje jako kompletní systém.

- **Systémové testování:** systémové testování je první fází testování, ve kterém je ověřována celá aplikace jako celek. Cílem systémového testování je zhodnocení, zda systém splňuje specifikaci požadavků. Tyto testy zároveň odhalují chyby a nedostatky v návrhu aplikace. V rámci systémového testování se mohou hodnotit funkční i nefunkční požadavky na systém. Funkčními požadavky se rozumí funkcionalita výrobku, kterou očekává zákazník, tedy například provedení operací definovaných v rámci případů užití. Nefunkčními požadavky na systém se rozumí různé metriky napsaného kódu, kvalita kódu, dokumentace, robustnost aplikace, doba odezvy atd. Systémové testy jsou prováděny na straně dodavatele produktu před jeho předáním zákazníkovi.

- **Akceptační testování:** akceptační testování je poslední etapou testování, jehož cílem je určit, zda je produkt připraven pro přijetí zákazníkem. Akceptační testování je prováděno koncovým uživatelem systému nebo samotným zákazníkem. Zákazník poté sám určí, zda systém splňuje jeho požadavky. Po úspěšném provedení akceptačních testů je produkt předán zákazníkovi.

K těmto čtyřem fázím testování je dobré zmínit navíc i regresní testování, které není samostatnou vrstvou testování, protože je možné jej provádět na kterékoliv úrovni testování. Podstatou regresního testování je ověření, zda nová verze softwaru nepřináší do produktu chyby nové. Jde tedy o zjištění, zda změny či implementace nových funkcí nemělo žádný dopad na stávající funkce a vlastnosti, které nebyly nikterak změněny.

Na obrázku 2.6 jsou tyto úrovně testování zobrazeny společně s fázemi vývoje softwaru, ke kterým náleží. K etapě analýzy požadavků náleží akceptační testování a s návrhem architektury se spojují systémové testy. K ověření návrhu modulů slouží integrační testy a ke kontrole implementace a vytvořených částí kódu se užívají jednotkové testy.



Obrázek 2.6: Úrovně testování softwaru společně s fázemi vývoje softwaru.

Pro aplikaci vytvořenou v této práci byly vytvořeny automatizované jednotkové a integrační testy a pro kontrolu splnění definovaných požadavků byly provedeny akceptační testy.

2.7.2 Testovací dvojníci

Jak již bylo zmíněno výše, jednotkové testy jsou určeny ke kontrole jedné určité jednotky kódu. Problém ovšem nastává tehdy, pokud v testované části jsou použity a volány jiné jednotky, které patří jiným objektům či vrstvám v rámci aplikace. Tyto závislosti testované jednotky testovat nechceme, jelikož přesahují rámec jednotkového testu. Nemluvě o tom, že by v takovém případě bylo potřeba všechny tyto závislosti instanciovat, čímž by se dramaticky zvýšila obtížnost testu. Z tohoto důvodu se zavádí pojem tzv. testovacího dvojníka, jehož cílem je nahradit původní objekt, na němž je testovaná jednotka závislá. Mezi takové dvojníky se podle Martina Fowlera řadí [3]:

- **Dummy:** dummy objekty nic neprovádějí a obvykle jsou použity pouze k předávání parametrů.
- **Fake:** fake objekt poskytuje funkční implementaci, která je značně odlehčená.
- **Stub:** stub objekty poskytují předpřipravené odpovědi pro jejich volané funkce uvnitř testu. Tyto objekty mohou také zachycovat stav objektu, tedy například stub pro vkládání záznamů do databáze uchovává kolekci záznamů k uložení, případně pouze počet vkládaných záznamů.
- **Mock:** mock objekt implementuje stejné rozhraní jako objekt, na kterém testovaný kód závisí. Pro očekávaná volání funkcí poskytují připravený výstup a zároveň ověřuje, zda očekávané funkce byly skutečně zavolány.

Mezi nejčastěji použité dvojníky patří objekty stub a mock, které jsou často zaměňovány. Základní rozdíl tkví v tom, že stub ověřuje výsledný stav testovaného kódu a jeho

závislostí po provedení testů. To nazýváme verifikací stavu. Na rozdíl od stub objektu, mock objekt užívá kontrolu chování. Objektu je řečeno, jaké vstupy a volání funkcí očekávat a po provedení testů se zkontroluje, zda očekávané chování bylo splněno. Dalším rozdílem může být fakt, že stub nemůže způsobit pád testu, zatímco mock ano. Samotný mock totiž určí, zda test uspěl či nikoliv.

Důkazem toho, že dvojníků existuje více, může být i skutečnost, že Microsoft definuje své vlastní dvojníky, které nazývá souhrnně Microsoft fakes [9]:

- **Stub:** stub nahrazuje třídu malou substitucí, která implementuje stejné rozhraní. K dosažení testovatelnosti pomocí tohoto přístupu je nutné, aby komponenta závisela pouze na rozhraních a nikoliv na konkrétních třídách. Tato technika se nijak neliší od obecné definice stub objektů.
- **Shim:** shim modifikuje kompilovaný kód za běhu a místo volání původních metod volá shim metody definované v testu. Může být použit v případě, kdy je zapotřebí volat uvnitř kódu již sestavené knihovny (např. .NET knihovny).

Ve výčtu těchto dvojníků vystupuje namísto mock objektů technika shim. Tato technika ovšem může vést ke zneužití psaní testovatelného kódu. Jedna z výhod tvorby jednotkových testů je tendence vést programátora k psaní testovatelného a čistého kódu, jelikož kód bez těchto znaků je velice obtížné testovat. To už tolik neplatí v případě, kdy je jednotka testována pomocí techniky shim. Části kódu testované tímto přístupem nevyžadují použití vnoření závislostí (dependency injection) a volné vazby (použití rozhraní namísto konkrétních objektů), tedy typických znaků pro testovatelný kód a tím nenutí programátora ke správným programovacím praktikám. Proto v rámci testování této aplikace budou použity techniky mock objektů, přičemž detailní tvorba testů bude popsána dále.

2.8 Problematika exekutorského úřadu a existující řešení

Soudním exekutorem je fyzická osoba, kterou stát pověřil exekutorským úřadem [13]. Je to svobodné povolání (podobně jako notář nebo advokát), přičemž soudní exekutor má postavení veřejného činitele. Všichni soudní exekutoři jsou organizování v samosprávné exekutorské komoře. Soudní exekutor navíc může zaměstnávat ve svém úřadu zaměstnance, kterými mohou být exekutorský kandidát, exekutorský koncipient, vykonavatel, nebo další zaměstnanci, kteří mohou být pověření exekutorem k vykonání jednoduchých úkonů související s exekuční činností.

Činností soudního exekutora, který vede svůj exekutorský úřad, je provádění exekucí na základě pověření soudu. Samotná exekuce většinou spočívá ve vymožení peněžité částky od povinného (dlužníka) pro oprávněného (věřitele), případně v donucení ke splnění jiné povinnosti. Kromě exekuce může exekutorský úřad sepisovat exekutorské zápisy, poskytovat právní pomoc v souvislosti s činností exekuce či přijímat peníze, listiny a jiné movité věci do úschovy. Další jeho činností mohou být dražby, autorizovaná konverze dokumentů, doručování písemností soudu nebo zjišťování a spravování majetku v trestním řízení.

V současnosti se na trhu vyskytují produkty, které nabízí komplexní řešení pro exekutorské úřady. Tato řešení kromě nezbytného základu pro funkci exekutorského úřadu obsahují také řadu funkcí a modulů, které nejsou nutné pro činnost úřadu. Takovým příkladem mohou být např. moduly manažerských statistik, zasílání sms zpráv, automatického vyplácení výplat, exportu spisů do HTML atd., případně jeden takový systém obsahuje funkcionalitu

pro více typů úřadů, nikoliv pouze pro ten exekutorský. Každá taková funkce navíc se poté projeví na ceně systému. Ano, tyto moduly a funkce mohou být užitečné, nicméně takto rozsáhlé systémy si můžou dovolit pouze větší exekutorské úřady. Zároveň jsou tyto systémy složitější pro ovládání. Pro nového uživatele není jednoduché se v systému vyznat a velmi často je nutné jej zaškolit. Systém vytvářený v rámci této práce je určen pro exekutorské úřady, které mají menší počet zaměstnanců a obsahuje vše nezbytné pro činnost úřadu. Mezi tyto nezbytné vlastnosti patří správa exekučních spisů a subjektů disponujících ve spisech, správa dokumentů, možnost komunikace s datovou schránkou, provádění lustrací a evidovat veškeré úkony do auditu. Dalším aspektem vytvářeného systému je jeho jednoduchost, která vede k příjemnému a efektivnímu uživatelskému rozhraní. Takové rozhraní je pro uživatele na první pohled pochopitelné a nevyžaduje rozsáhlou uživatelskou příručku. A právě takový systém je v této práci vytvářen.

Kapitola 3

Analýza požadavků informačního systému pro exekutorské úřady

V této kapitole je nastíněna analýza požadavků na informační systém pro exekutorský úřad. Jsou zde uvedeny a popsány části, které musí implementovat výsledná aplikace. Těmito částmi jsou agenda spisu společně s jeho workflow, subjekty a dokumenty, lustrace, audit a komunikace pomocí datové schránky. Kapitola je uzavřena sekcemi popisující případy užití systému a rizika projektu související s lustracemi.

Jak již bylo zmíněno v předchozí kapitole, vytvářená aplikace je určena pro menší exekutorské úřady, jejíž primárním úkolem je zjednodušení agendy úřadu soudních exekutorů, integrace stanovených potřeb do jednoho systému a poskytnutí elektronické komunikace s dalšími institucemi. Tento systém je implementován jako webová aplikace, která musí navíc brát ohled na povahu citlivých dat, která budou v databázi uložena. Povinné součásti informačního systému pro exekutorský úřad tvoří:

- **Agenda spisu:** propojení spisu s příslušejícími subjekty, správa příslušných dokumentů a celkový workflow spisu.
- **Agenda subjektů:** ke spisům náleží subjekty, jimiž jsou povinný a oprávněný. Volitelně mohou být ke spisu přiřazeny i subjekty typu banka, pojišťovna, právní zástupce, obecně subjekty třetích stran.
- **Správa dokumentů:** možnost spravovat dokumenty příslušející ke spisům.
- **Lustrace:** kontrola údajů v různých evidencích a rejstřících.
- **Audit:** evidence úkonů ve formě auditu, který zajistí dohledatelnost změn.
- **Datová schránka:** příjem a odesílání pošty a dokumentů prostřednictvím datové schránky.

3.1 Agenda spisu

Exekuční spis je povinnou dokumentací exekučního řízení, která obsahuje veškeré informace o exekuci, přiřazených subjektech a náležících dokumentech. Vzhledem k tomu, že hlavní požadavek na systém exekutorského úřadu je správa spisů, je tato část popsána podrobněji. Agendu spisu můžeme rozdělit na následující části:

- **Základní informace o spisu:** obsahují obecné informace o spisu, kterými jsou číslo spisu, datum přijetí návrhu, datum vytvoření návrhu, spisová značka, typ předmětu exekuce (např. vyklizení, provedení práce a výkonu, odebrání věcí, peněžní plnění, rozdělení společné věci), poznámky nebo umístění spisu.

- **Subjekty:** subjekty, související se spisem, mohou být typu povinný, oprávněný, právní zástupce, opatrovník, banka, obecný zástupce, místně příslušný soud, zaměstnavatel či partner. Ke každému subjektu je podle jeho typu přiřazeno jméno, příjmení, RČ, IČ, název organizace a adresa. Další možností dělení subjektů je podle jeho stavu, tzn. v konkurzu, návrh na osobní konkurz, bez závad, pobyt neznámý, osobní konkurz, zemřel.

- **Exekuční titul:** exekuční titul je rozhodnutí nebo jiná listina, vydaná v našem případě soudem, sloužící jako podklad pro exekuci. Váže se k subjektu a obsahuje informace týkající se data vykonatelnosti, data vykonání, názvu titulu, čísla titulu a částky původní pohledávky.

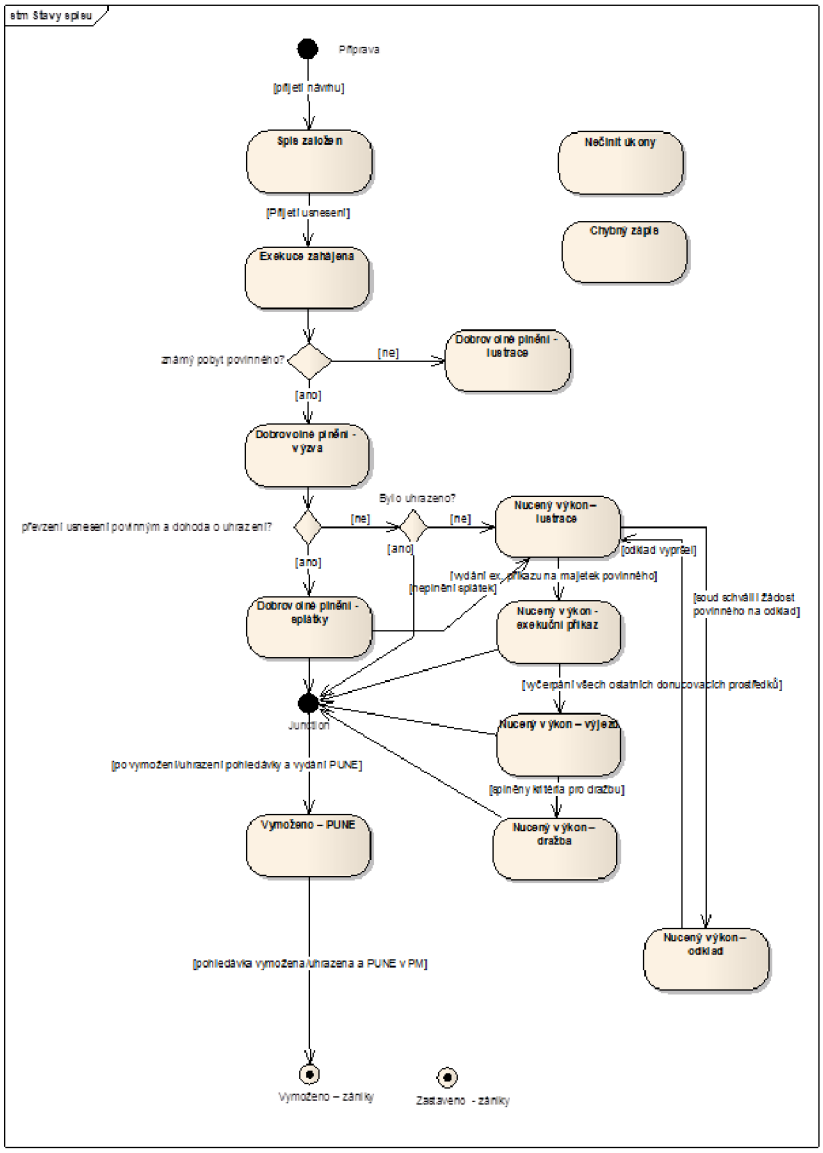
- **Finanční náležitosti:** částí spisu jsou také jeho finanční náležitosti, které si za dané činnosti nárokuje exekutorský úřad obsahující jistiny, úroky, náklady a příslušenství.

- **Dokumenty:** nedílnou součástí spisů jsou veškeré dokumenty, exekuční příkazy, oznámení, povolení od soudu atd.

- **Evidence součinností:** v rámci exekučního spisu se zaznamenává veškerá součinnost. Touto součinností je myšlena veškerá korespondence, ať už pomocí datových schránek či klasickou poštovní obálkou a provedené lustrace.

Každý spis prochází stavy, které popisuje workflow spisu na obrázku 3.1. Jakákoliv exekuce začíná podáním návrhu ze strany oprávněného soudnímu exekutorovi. Ten jej předá exekučnímu soudu (okresnímu soudu), který leží v obvodu místa trvalého bydliště povinného. Jestliže jsou splněny zákonné požadavky, pověří exekuční soud provedením exekuce daného soudního exekutora. Proti pověření se nelze odvolat, lze jen podat návrh na zastavení zahájené exekuce. Po pověření zasílá exekutor povinnému do vlastních rukou vyrozumění o zahájení exekuce a výzvu ke splnění vymáhané povinnosti. Pokud povinný do 30 dnů dobrovolně splní vymáhaný nárok včetně nákladů exekuce, exekuce tím zaniká. V opačném případě je provedena exekuce. Exekuce může probíhat více způsoby zároveň, přičemž pro každý z nich musí být vydán zvláštní exekuční příkaz. Primárně však musí být upřednostněno přikázání pohledávky z bankovního účtu, až poté prodej movitých či nemovitých věcí. Exekuce ukládající zaplacení peněžité částky může být provedena srážkami ze mzdy a jiných příjmů, přikázáním pohledávky nebo prodejem movitých či nemovitých věcí. Exekuci, která neukládá povinnost uhrazení peněžité částky, lze provést odebráním věci, vyklizením, rozdělením společné věci nebo provedením prací a výkonů. Celková částka nákladů exekuce je zapsána v samostatném příkazu k úhradě nákladů exekuce (PUNE), která se rovněž vymáhá v rámci nařízené exekuce. Exekuce končí vymožením pohledávky oprávněného a nákladů exekuce.

Pro systém je důležité uchovávat si u spisu jeho stav, který indikuje, jak je, případně není možné se spisem manipulovat. Zároveň umožňuje automatický přechod do následujícího stavu přiřazením, doručením nebo odesláním určitého dokumentu. Samozřejmostí je umožnění uživateli provádět celkovou správu nad spisy, jejich vytváření, úpravu, filtrování a zobrazování. Důležitým aspektem systému související se spisy je také možnost tvorby a



Obrázek 3.1: Stavový diagram procesu exekučního spisu [11].

přiřazení subjektů ke spisu a správa dokumentů, které se k danému spisu vážou. Systém bude navíc umožňovat zaznamenání finančních náležitostí, součinností a zobrazení seznamu úkolů, náležících ke spisu.

Co se týče subjektů, systém bude implementovat funkcionalitu správy subjektů, jejich filtrování a vyhledávání. Pod pojmem subjekt zde nebudou vystupovat pouze povinní a oprávnění uvnitř spisu, ale také osoby třetích stran, jako jsou banky, soudy, pojišťovny atd. Systém tedy nerozlišuje správu subjektů a třetích osob a obě tyto skupiny jsou souhrnně označeny jako správa subjektů.

3.2 Lustrace

Lustrací se rozumí kontrola údajů v různých evidencích a rejstřících. V rámci systému se jedná o provedení lustrace pro určitý subjekt, přičemž si je možné vybrat, ve kterých rejstřících se má lustrace vykonat. Mezi možné varianty lustrace patří lustrace exekučního titulu, lustrace živnostenského úřadu, centrální evidence osob (CEO), nemovitostí, banky, VZP, ostatních zdravotních pojišťoven, české správy sociálního zabezpečení, rejstříku motorových vozidel, úřadu práce, státní sociální podpory a nebo penzijního připojištění.

Protože se vývojem systému nezabývá osoba exekučního úřadu, nýbrž vývojář, je pro něj většina z těchto rejstříků buďto zpoplatněná nebo úplně nepřístupná. Proto bude systém umožňovat pouze lustraci ve veřejných evidencích, které nejsou zpoplatněny, případně na jejich testovacích prostředích (pokud existují). Mezi takové kandidáty se řadí insolvenční rejstřík, rejstřík evidence úpadců, rejstřík katastru nemovitostí nebo administrativní registr ekonomických subjektů (ARES). Jelikož ARES zpřístupňuje ve svých výsledcích i údaje z jiných registrů (mezi nimž jsou i insolvenční rejstřík a rejstřík evidence úpadců), lustrace bude probíhat pouze s rejstříkem ARES.

ARES je informační systém spravován ministerstvem financí ČR, který umožňuje dotazovat se na ekonomické subjekty registrované v ČR. Data čerpá z jednotlivých registrů státní správy a souhrnně je zprostředkovává k zobrazení informací pro daný ekonomický subjekt. Má ovšem omezení pro veřejnost, které neumožňuje vyhledávat informace podle rodného čísla (to je určeno pouze pro státní správu). V naší aplikaci bude tedy možné pouze lustrovat právnickou osobu podle IČ. Všechny provedené lustrace se opět zaznamenávají do evidence součinností.

3.3 Audit

Pro exekuční úřad vyplývá ze zákona povinnost vytvářet si evidenci změn. Informační systém tedy musí zaznamenávat a především vypisovat do žurnálu veškeré činnosti a změny provedené v systému daným uživatelem. Záznam auditu musí obsahovat datum a čas provedení úkonu, uživatele který provedl danou činnost, typ auditu, stručný popis a údaje týkající se spisu nebo subjektu, ke kterému se tato činnost váže. Veškeré takto uložené informace musí být zpětně dohledatelné.

3.4 Datová schránka

Datová schránka je komunikační nástroj garantovaný státem [18], který slouží především ke komunikaci s orgány veřejné moci. Veškeré úřady mají povinnost komunikovat prostřednictvím datových stránek s každým, kdo ji má zřízenou. Všechny orgány veřejné moci, právnické osoby zapsané v obchodním rejstříku, advokáti, daňoví poradci a insolvenční správci mají povinně ze zákona zřízenou datovou schránku. Ostatní si o ni mohou dobrovolně zažádat. Orgány veřejné moci jsou povinny posílat dokumenty adresátům přednostně do datové schránky, pokud ji mají zřízenou. V opačném případě dochází ke komunikaci klasickým způsobem v listinné podobě.

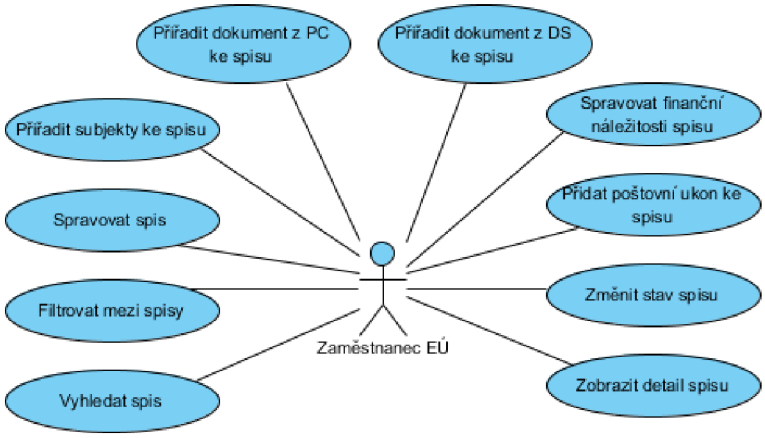
Z uživatelského pohledu funguje jako běžná e-mailová schránka, technické řešení je však odlišné kvůli zajištění důvěryhodnosti doručování datových zpráv. Na rozdíl od klasického e-mailu nemá datová zpráva část zvanou tělo zprávy, ale obsah zprávy má formu příloh. Navíc je zpráva obalena elektronickou a časovou značkou (razítkem).

Vzhledem k tomu, že exekutorský úřad patří mezi orgány veřejné moci, musí mít ze zákona zřízenou datovou schránku. Pro komunikaci s ostatními úřady či soudem používá datové zprávy a proto je nutností systému, aby obsahoval funkcionalitu umožňující takové zprávy zasílat a přijímat, respektive zobrazovat doručené zprávy v datové schránce. Přiřazení dokumentu ke spisu bude ponecháno na uživateli, který si příchozí dokument manuálně přiřadí ke správnému spisu. Veškerá komunikace vedena datovými schránkami se eviduje jako součinnost. Tím, že informační systém implementuje rozhraní pro komunikaci s datovými schránkami, zjednodušuje a urychluje práci uživatele, který provádí potřebné operace v rámci jednoho systému. Pokud by tato funkcionalita nebyla implementována, nutilo by to uživatele k užívání jiného systému pro zobrazení zpráv a poté přepisování důležitých dat a dokumentů z jednoho systému na druhý.

3.5 Případy užití systému

Případy užití se používají k popisu chování systému z hlediska uživatele a zachycují, kteří uživatelé se systémem pracují a jaké činnosti se v rámci systému vykonávají. V rámci informačního systému pro exekutorské úřady vystupují tři uživatelské role, a to role zaměstnance úřadu, exekutora a administrátora.

Role zaměstnance je určena pro veškeré zaměstnance exekutorského úřadu kromě samotného soudního exekutora. Tato role umožňuje provádět běžné administrativní úkony uvnitř systému. Mezi nejdůležitější činnosti systému patří správa exekučních spisů. Uživatel systému bude schopen vytvářet a editovat tyto spisy a především přiřazovat jim veškeré náležitosti, jakou jsou finance, poštovní úkony, dokumenty atd. Výčet všech případů užití pro kategorii exekučního spisu pro zaměstnance exekutorského úřadu jsou uvedeny na obrázku 3.2. Pod pojmem spravovat spis se rozumí jeho vytvoření a jeho editace. Mazání spisů není v systému umožněno kvůli nutnosti zpětné dohledatelnosti spisu.



Obrázek 3.2: Diagram případů užití systému pro agendu spisu.

Dalšími důležitým aspektem pro systém je správa subjektů a možnost komunikovat přes datové schránky. Zaměstnanec bude schopen vyhledávat, filtrovat a zobrazovat detaily subjektů a provádět lustraci pro právnické osoby. Správou subjektů se opět rozumí pouze vytvoření a editace, kde mazání subjektů je vzhledem k možnosti dohledat subjekt znemožněno. Rozhraní systému bude poskytovat funkcionalitu pro komunikaci s datovou schránkou, tedy

především zobrazení doručených, odeslaných zpráv a vytvoření zprávy nové. Každou zprávu a její dokumenty bude možné přiřadit ke spisu a označit ji jako zpracovanou. Tyto i další případy užití jsou uvedeny na obrázku 3.3.

Obrázek 3.3: Diagram případů užití systému pro správu subjektů (vlevo) a komunikaci s datovými schránkami (vpravo).

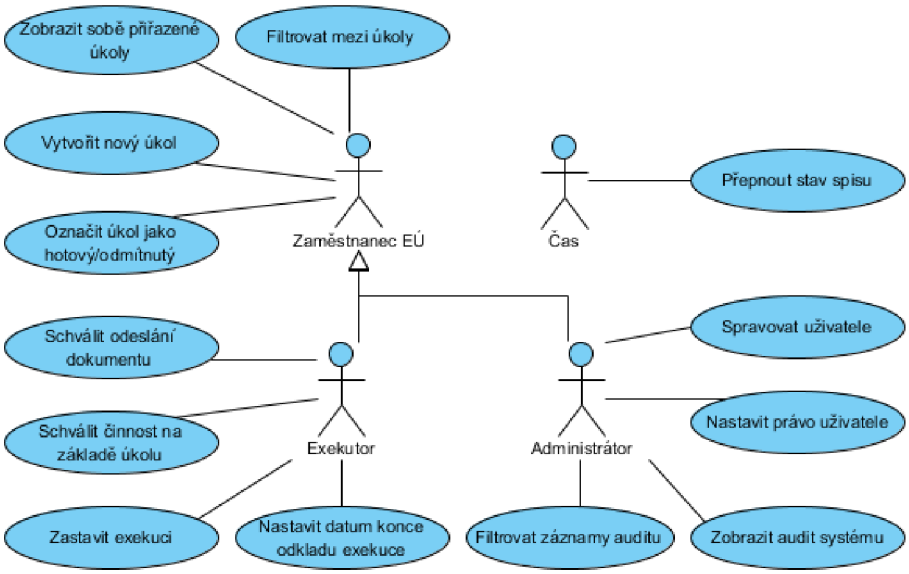
Posledními případy užití systému pro zaměstnance úřadu jsou případy týkající se modulu úkolů. Úkoly jsou do systému zavedeny z toho důvodu, aby exekutor mohl pomocí nich schválit činnost zaměstnance úřadu. Typickým příkladem je odeslání pošty pro soud. Samotný zaměstnanec úřadu má sice možnost odeslat zprávy z datové schránky, nicméně odeslání některých dokumentů vyžaduje schválení ze strany exekutora. V takovém případě se vytvoří úkol pro exekutora, týkající se schválení odeslání, a dokument je poslán až po potvrzení exekutorem. Podobně se můžou řešit i další případy, které jsou pouze v kompetenci samotného soudního exekutora.

Dosud popsané případy užití se týkaly pouze zaměstnance úřadu. Nesmí však zapomenout ani na roli soudního exekutora a administrátora. Role exekutora nabízí stejné možnosti jako role zaměstnance, a navíc ji rozšiřuje o schvalování potřebných úkonů, na které má právo pouze samotný exekutor. Exekutor má poté jako jediný právo schválit odeslání dokumentu soudu, zastavit exekuci a nastavit datum, které ukončí odklad exekuce. Poslední rolí, je role administrátora, která také rozšiřuje možnosti zaměstnance, a navíc spravuje uživatele systému. V jeho kompetenci je nastavení rolí uživatelů v rámci systému a také vyhledávání v auditu. Detaily případů užití systému právě zmíněných uživatelských rolí doplněno o případy užití v části úkolů je graficky znázorněno na obrázku 3.4.

Tento diagram obsahuje také nepřímého aktéra, kterým je čas. V rámci workflow spisu se vyskytují případy, ve kterých dochází ke změně stavu spisu po uplynutí určité lhůty. Konkrétně je to situace, kdy je povinnému zaslána výzva k uhrazení dlužné částky. Pokud do dané lhůty 30 dnů nepřistoupí dlužník k dobrovolnému plnění, je stav spisu přepnut do stavu nuceného výkonu. Podobně je řešen případ, kdy dochází k ukončení lhůty, po kterou byl nucený výkon odložen. Systém musí být v obou případech schopen zjistit, zda došlo k ukončení dané lhůty a případně přepnout spis do příslušného stavu.

3.6 Rizika projektu

Rizika projektu popsané v této sekci se týkají součinností systému s třetími stranami. Jak již bylo výše řečeno, exekutorský úřad využívá lustrační rejstříky pro vyhledání informací o subjektech. Většina rejstříků je však zpoplatněná nebo vůbec nepřístupná pro veřejnost. Rejstříky je možné využít pouze v případě, kdy uživatel disponuje exekutorským titulem. Tento fakt zamezuje možnosti využití rejstříků uvnitř vyvíjené aplikace. V jeho důsledku



Obrázek 3.4: Diagram případů užití systému pro další role včetně případů užití pro úkoly.

musí uživatel využít jiné aplikace či klienty k tomu, aby získal informace o hledaném subjektu. Vyvíjená aplikace tedy předpokládá, že k lustraci je využita jiná aplikace a získaná data jsou vložena do systému uživatelem.

Součinnost nemusí být v rámci exekutorského úřadu uplatněna pouze pro evidence a rejstříky, ale také pro banky, zdravotní pojišťovny, poštu, telekomunikační úřady apod. Tyto instituce dokonce spravují své webové služby pro elektronickou spolupráci, nicméně opět se jedná o přísně střežené služby, které jsou otevřeny pouze státní správě.

Elektronické součinnosti v této aplikaci jsou prováděny pomocí datových schránek a lustrací uvnitř rejstříku ARES, přičemž do budoucna by bylo žádoucí získat také přístup k dalším evidencím pro jejich automatizované využití exekutorským úřadem.

Kapitola 4

Návrh řešení informačního systému pro exekutorské úřady

V kapitole návrhu řešení vytvářeného informačního systému pro exekutorské úřady je popsán návrh struktury systému a jeho rozdělení do modulů. Následně je nastíněna architektura pro dosažení multitenantní aplikace a podkapitoly věnující se autentizaci a řízení přístupu, návrhu databázového schématu a rozdělení obrazovek v rámci aplikace.

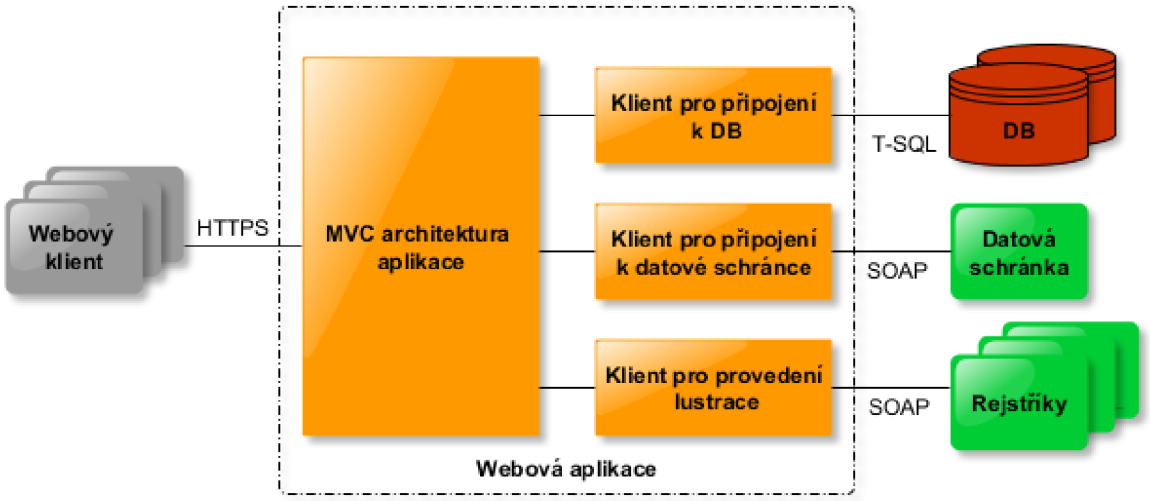
Aplikace je realizována v technologii ASP.NET MVC v jazyce C#. Jedná se tedy o aplikaci, jejíž serverové řešení není platformně nezávislé, a pro své nasazení potřebuje prostředí operačního systému Windows. To však neplatí pro klientskou stranu, které použitá platforma nevadí, neboť využívá framework AngularJS a framework Bootstrap. Použitým databázovým serverem je Microsoft SQL Server.

4.1 Struktura systému

Vzhledem k analýze požadavků na systém lze aplikaci rozdělit do následujících částí. Jádro aplikace bude tvořit modul s MVC architekturou. Tato část bude zodpovědná za zobrazování stránek klientům, obsluhování žádostí od klienta a celkovou distribuci práce. Jádro bude komunikovat s klienty pro připojení k databázi, klientem pro připojení a obsluhu datových schránek a s klientem pro provedení lustrace v rejstříku ARES. Všechny tyto části poté budou v rámci webové aplikace nasazeny na aplikační server. Popisované komponenty znázorňuje obrázek 4.1.

Následuje podrobnější popis jednotlivých součástí systému:

- **Webový klient:** webový klient, respektive klienti, přistupují k aplikaci v rámci webového prohlížeče. Komunikace mezi klienty a serverem je zprostředkována pomocí zabezpečeného protokolu HTTPS.
- **Jádro aplikace (MVC architektura aplikace):** jádrem aplikace se rozumí modul, který zpracovává požadavky a distribuuje celkovou práci uvnitř systému. Tento modul bude realizován pomocí MVC architektury (odtud označení modulu na obrázku), kde logika ovladače bude mít na starosti obsluhu událostí od klienta, tok událostí v aplikaci a obecně aplikační logiku. Část označená jako pohled bude poskytovat uživatelské rozhraní a model bude reprezentovat data a funkční logiku. Část modelu bude rozdělena na další části pomocí vícevrstvé architektury pro vytvoření znovupoužitelného a testovatelného kódu. Součástí modulu aplikace bude tedy i část



Obrázek 4.1: Struktura systému.

zodpovědná za správu veškerých spisů, subjektů a dokumentů exekučního úřadu. Tato část bude rovněž zodpovědná za vedení auditu uvnitř aplikace. Pro připojení k databázi využívá databázového klienta, pro propojení s datovou schránkou využívá klienta pro práci se schránkami a pro provedení lustrace klienta implementující tuto činnost.

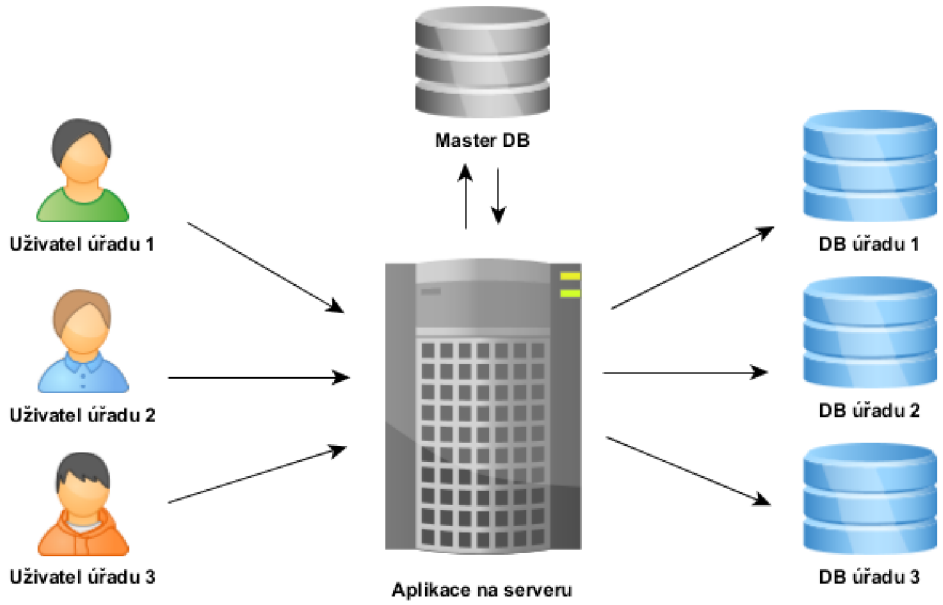
- **Klient pro připojení k databázi:** tento modul bude zodpovědný za veškerou interakci s databází a jejím serverem. Databázovým serverem bude Microsoft SQL Server, kde data budou uložena v tradičním relačním databázovém modelu. Pro dotazy do databáze bude tedy použita odnož jazyku SQL, a to T-SQL (Transact-SQL). Vzhledem k dosažení vlastnosti multitenance, může naráz existovat více instancí klienta pro různé databáze.
- **Klient pro připojení k datové schránce:** úkolem této komponenty je zajištění spolupráce s datovými schránkami. Ta bude uskutečněna pomocí volání veřejného API informačního systému datových schránek přes protokol SOAP. SOAP je protokol pro výměnu zpráv založených na XML, který bude volán přes HTTPS. API datových schránek pokrývá veškeré funkce týkající se správy a odesílání zpráv uvnitř datové schránky.
- **Klient pro provedení lustrace:** další částí je modul umožňující vyhledávání subjektů v rejstříku ARES. Tento rejstřík umožňuje komunikaci přes protokol SOAP nebo přes REST rozhraní pomocí HTTP metody *get*.

4.2 Zajištění multitenance

Zajímavým požadavkem na systém je zajištění multitenance. Cílem aplikace je vytvoření architektury, která umožňuje, aby na serveru běžela pouze jedna instance softwaru, která by poskytovala služby více zákazníkům. V případě této aplikace by se jednalo o poskytování služeb více exekutorským úřadům. Vzhledem ke skutečnosti, že data exekutorského úřadu jsou velice citlivá, měla by aplikace disponovat patřičnou úrovní zabezpečení těchto dat.

Proto bude použit model oddělených databází pro každý úřad, který ochraňuje uživatele od úmyslného či neúmyslného dosažení cizích dat. Oddělení databází navíc umožňuje snadnější rozšíření datového modelu k dosažení specifických zákazníkových potřeb a snadnější obnovení dat ze zálohy na úkor vyššího nároku na hardware.

K dosažení takové architektury je zapotřebí uchovávat metadata, která obsahují informace o spojení uživatele s příslušným úřadem a především informaci, ke které databázi se smí uživatel připojit. Tuto funkci bude vykonávat hlavní (master) databáze. Tato databáze bude obsahovat nejen spojení uživatelů k příslušejícím databázím, ale také autentizační informace. Po zaslání dotazu uživatele o přihlášení se do systému je nejprve porovnáno uživatelské jméno a heslo uvnitř hlavní databáze, a pokud je autentizace úspěšná, je vytvořeno spojení do databáze úřadu, ke kterému uživatel patří. Nyní má už uživatel k dispozici data svého exekutorského úřadu. Schéma tohoto principu demonstruje obrázek 4.2.



Obrázek 4.2: Schéma multitenantní aplikace.

Zvolená architektura má však i své slabé stránky. Tou největší je fakt, že mezi databázemi nelze zajistit referenční integritu. Omezení pro náš návrh to přináší v případě, kdy uvnitř databáze úřadu chceme využít data o uživateli, která jsou v hlavní databázi. Řešením je vytvoření nové tabulky v databázi úřadu, která obsahuje replikovaná data z hlavní databáze. Data budou replikována pouze jedním směrem (z hlavní databáze do příslušné databáze úřadu), čímž předejdeme problému obousměrné synchronizace mezi databázemi. V rámci našeho řešení bude každá změna uvnitř tabulky uživatelů hlavní databáze replikována do tabulky uživatelů uvnitř databáze úřadu, přičemž nebude docházet k žádným změnám replikovaných dat v databázi úřadu.

4.3 Autentizace a řízení přístupu

Základem bezpečnosti systému bude ověření identity uživatele před vstupem do aplikace. Tato autentizace bude probíhat pomocí zadání vstupních údajů, tzn. uživatelského jména

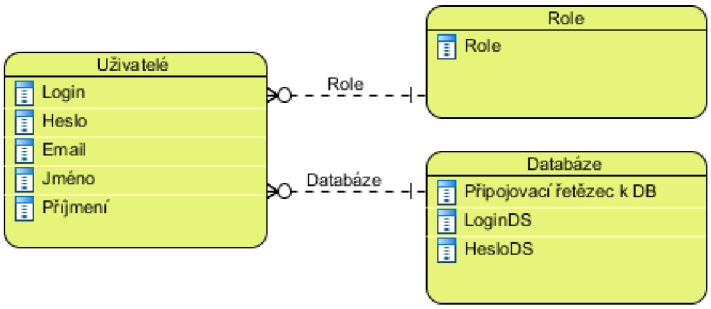
a hesla. Jak již bylo nastíněno v předchozím odstavci, autentizační údaje se uchovávají v hlavní (master) databázi, přičemž heslo je navíc zašifrováno. Neautentizovanému uživateli bude přístup do aplikace odmítnut.

Samozřejmostí je také implementace procesu řízení přístupu. V rámci něj bude ověřována míra oprávnění a uživatelských práv k přístupu ke zdrojům. V systému budou nadefinovány uživatelské role související s pravomocemi uvnitř exekutorského úřadu, kterými jsou role exekutora, zaměstnance a administrátora. Uživatel bude moci zadávat veškeré informace ke spisům a subjektům, ovšem nebude mít právo odesílat některé typy dokumentů. Exekutor bude rozšiřovat práva uživatele v podobě schopnosti schvalovat odesílání dokumentů, kde je to potřeba. Role, přiřazená konkrétnímu uživateli, je uložena rovněž v hlavní databázi a je uživateli přiřazena po úspěšné autentizaci. Aplikace následně povoluje provést pouze ty činnosti, které jsou pro danou roli povoleny.

4.4 Návrh schématu databáze

Velmi důležitou částí této fáze je rovněž návrh databázového schématu. V rámci návrhu databázového schématu byly vytvořeny konceptuální ER diagramy. Vzhledem k rozsahu vytvořeného databázového schématu a vytvoření multitenantní aplikace budou ER diagramy rozděleny do několika celků.

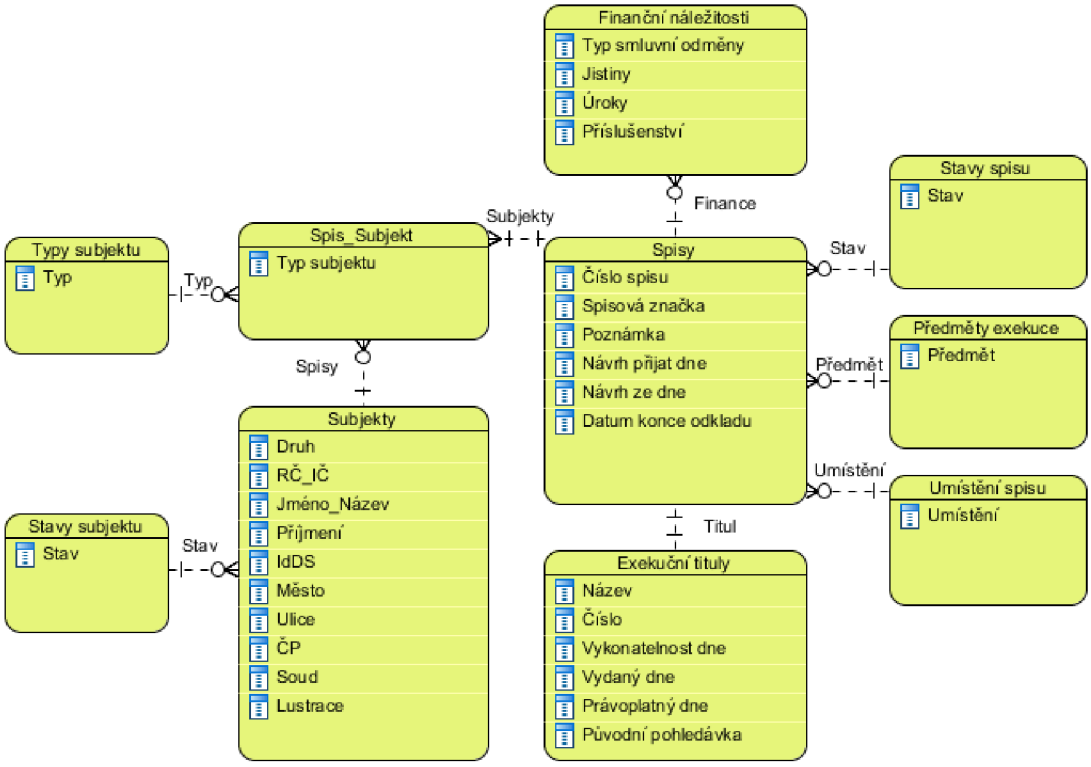
Za účelem umožnění multitenance musí být vytvořena nejprve databáze, která obsahuje data o jednotlivých uživatelech, databázích a uživatelských rolích. Nejdůležitějšími údaji z hlediska multitenance jsou připojovací řetězce k databázi, jenž jsou obsaženy v tabulce databází. Tyto řetězce poté definují potřebné informace pro připojení k uživatelské (exekutorské) databázi. Součástí této tabulky jsou i údaje potřebné pro připojení se k datovým schránkám, jelikož celý exekutorský úřad sdílí jednu konkrétní datovou schránku. Samozřejmostí je také sloupec pro přihlašovací jméno a heslo uvnitř tabulky uživatelů, které slouží k autentizaci uživatelů. Toto schéma je vykresleno v obrázku 4.3.



Obrázek 4.3: Konceptuální návrh hlavní (master) databáze.

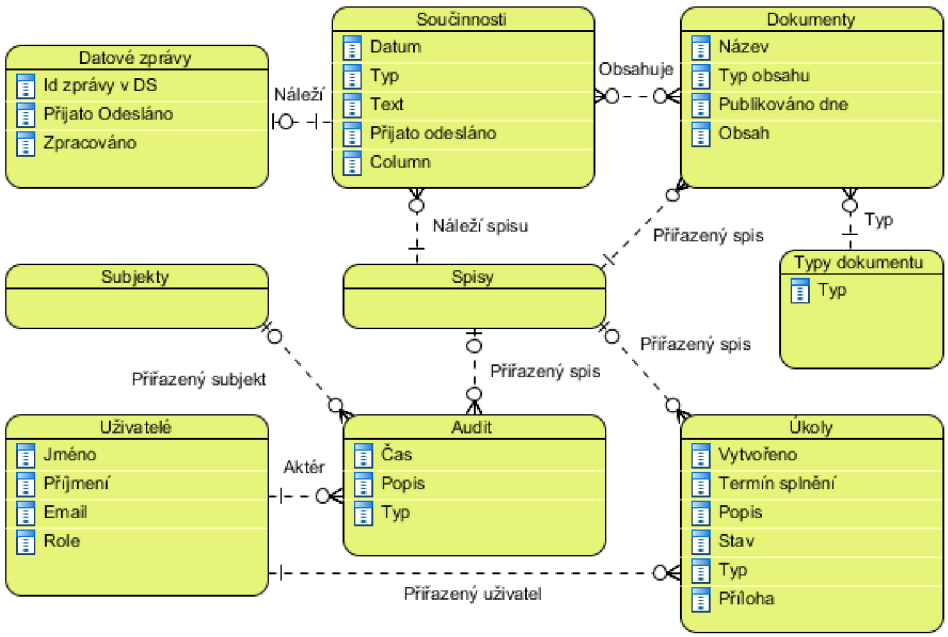
Od hlavní databáze se nyní přesuneme k databázi exekutorského úřadu a rozdělíme ji na dvě části. Kostrou databázového modelu jsou tabulky spisů a subjektů, které jsou spojeny vazební entitní množinou. Ta je velmi důležitá, protože obsahuje atribut typu subjektu. Subjekt poté může vystupovat v jednom spise v roli povinného a v jiném naopak v roli oprávněného subjektu. Stejným způsobem se ke spisu přiřazují i subjekty třetích stran. Subjekt navíc nemusí vystupovat v žádném spise, zatímco spis musí obsahovat alespoň dva subjekty (povinného a oprávněného). Odtud plynou použité typy vazeb v návrhu. Ke spisům

náleží také finanční náležitosti a každý spis má přiřazen exekuční titul. Ostatní tabulky, kterými jsou stavy spisů a subjektů, typy subjektů, předměty exekuce a umístění spisu, plní funkci číselníků, které obsahují seznam všech možných hodnot v rámci dané kategorie. Popisované tabulky jsou graficky znázorněny na obrázku 4.4.



Obrázek 4.4: Konceptuální návrh vztahů mezi spisem a subjekty.

Druhá část databázového schématu zachycuje vztahy mezi spisy, evidencí součinností a dokumenty. Každému spisu mohou náležet dokumenty, které musí mít přiřazený typ dokumentu. Na základě takového typu se poté může měnit stav příslušného spisu. Tabulka typu dokumentu obsahuje možné hodnoty ve formě číselníku. Dokument se ale může vázat také k záznamu součinnosti. V takovém případě je dokument přiřazen ke korespondenci, která může obsahovat více dokumentů a naopak dokument může vystupovat ve více korespondencích. Vztah mezi dokumentem a součinností má tedy vazbu M:N. Jednotlivé záznamy součinností se spojují s příslušným spisem. Pro indikaci, zda byla zpráva z datové schránky zpracována či nikoliv, je vytvořena ve schématu tabulka datových zpráv, která se spojuje se záznamem součinnosti. Pro podporu dohledatelnosti je v databázi nadefinována tabulka auditu. Záznamy v této tabulce se vážou buďto ke spisu, nebo k subjektu a zároveň k uživateli, který změnu provedl. Tabulka uživatelů je kromě vazby s tabulkou auditu spojená také s tabulkou úkolů. Každý úkol je pochopitelně přiřazen řešiteli a proto je tato vazba nezbytná. Některé úkoly souvisejí s určitým spisem a je tedy možné je přiřadit rovněž k tabulce spisu. Pro lepší ilustraci je vše vyobrazeno na obrázku 4.5. Tabulky subjektů a spisů pro jednoduchost neobsahují příslušné sloupce, jelikož byly zobrazeny v předchozím obrázku.



Obrázek 4.5: Konceptuální návrh vztahů mezi spisem a součinnostmi.

4.5 Grafické uživatelské rozhraní

Součástí návrhu aplikace jsou také obrazovky, kterými bude výsledný systém disponovat. Každá obrazovka bude přizpůsobena datům, které bude zobrazovat, a při jejím návrhu bude snaha o vytvoření jednoduchého a intuitivního ovládání. Rozdělení jednotlivých obrazovek aplikace bude vypadat následovně:

- **Obrazovka pro přehled spisů:** obrazovka přehledu spisů poskytuje uživateli přehled o veškerých spisech vedených daným exekučním úřadem. Obsahuje také vstupní pole pro možnost filtrování zobrazených záznamů a tlačítko pro vytvoření spisu. Součástí každého záznamu je odkaz pro zobrazení detailu vybraného spisu.
- **Obrazovka pro detail spisu:** tato obrazovka zobrazí veškeré údaje týkající se exekučního spisu, tedy jeho základní informace, informace o povinném a oprávněném a finanční přehled. V dolní části obrazovky jsou vykresleny přepínatelné záložky, které dávají uživateli přehled o veškeré korespondenci v rámci spisu a lustraci přiřazených subjektů (souhrnně součinnosti), připojených dokumentech, exekučním titulu, finančních záznamech a úkolech, které se spisem souvisí. Nesmíme však zapomenout ani na přehled přiřazených subjektech a jejich typech. V okně se navíc nachází tlačítka pro úpravu spisu a subjektů a vysunovací lišta poskytující operace přidání poštovní korespondence, dokumentu z PC, finanční náležitosti a přepnutí stavu spisu. Pro roli exekutora je tu navíc možnost zastavení exekuce a vložení data, které ukončuje odklad exekuce.
- **Obrazovka pro přehled subjektů:** okno pro přehled subjektů vypisuje uživateli záznamy všech subjektů spravovaných úřadem. Stejně jako obrazovka přehledu spisu nabízí vstupy pro filtrování jednotlivých záznamů. V rámci okna je k dispozici také tlačítko pro vytvoření nového subjektu.

- **Obrazovka pro detail subjektu:** tato obrazovka nabízí uživateli detailní informace týkající se subjektu a také výpis všech spisů, ve kterých daný subjekt figuruje včetně jeho typu uvnitř spisu. Pokud je zobrazeným subjektem právnická osoba s přiřazeným IČ, je možné ji nechat lustrovat v systému ARES a výsledky dotazu jsou zobrazeny ve spodní části obrazovky. Kromě možnosti lustrace nabízí obrazovka také volbu pro editaci subjektu.

- **Obrazovka pro poštu:** cílem obrazovky pro poštu je vyobrazení veškeré odeslané a přijaté pošty prostřednictvím datové schránky. Poštu bude možné filtrovat a každá poštovní zpráva bude obsahovat navíc informaci, zda je již zpracována či nikoliv. Pro každou příchozí poštu je možné zobrazit detail zprávy a přiřadit ji ke spisu včetně dokumentů uvnitř zprávy. Okno obsahuje také volbu vytvoření nové zprávy, která má rovněž možnost přiřazení odesílané zprávy ke spisu.

- **Obrazovka pro úkoly:** tato obrazovka vykreslí veškeré úkoly přiřazené přihlášenému uživateli a jejich detaily. Úkoly je opět možné filtrovat a označit je jako hotové, či odmítnuté. Okno nabízí také volbu vytvoření nového úkolu.

- **Obrazovka pro administraci:** okno pro administraci je přístupné pouze uživateli s rolí administrátora, který v rámci této obrazovky bude moci zobrazit a upravit data uživatelů úřadu. Ve spodní části okna může administrátor vidět přehled provedených operací v systému, které se ukládají ve formě auditu.

- **Obrazovka pro přihlášení:** obrazovka pro přihlášení se zobrazí vždy před vstupem do systému a nabídne vstupní pole pro přihlašovací údaje uživatele. Pokud se uživateli nepodaří přihlásit do systému, zobrazí se mu varování o neúspěšném přihlášení.

- **Modální okno pro rychlé vyhledávání:** není to čistě obrazovka jako v předchozích případech, nicméně je možné jej zobrazit z navigační lišty aplikace a proto je zařazeno mezi obrazovky aplikace. Obsahuje vstupní pole pro rychlé vyhledání spisu nebo subjektu a po úspěšném vyhledání záznamů zobrazí výsledky hledání. Výsledek hledání obsahuje také odkaz, kterým se uživatel dostane na detail nalezeného záznamu.

Všechny výše zmíněné obrazovky využívají pro zobrazení formulářů modální okna. Pokud tedy dochází k vytvoření záznamu, jeho editaci či zobrazení detailu, jsou tyto informace a formuláře umístěny v modálních oknech. Výhodou takového přístupu je skutečnost, že pro zobrazení modálního okna není nutné dotazovat se serveru pro novou obrazovku a uživateli je modální okno zobrazeno ihned. Navíc je formulář čistě odstíněn od původní obrazovky, která je k vidění na pozadí. Takovým případem je i již zmíněné modální okno pro rychlé vyhledávání.

Součástí každé obrazovky (kromě té přihlašovací) je navigační lišta, která umožňuje snadnou orientaci uvnitř aplikace. Tato lišta obsahuje ve své levé části odkazy na obrazovky přehledu spisů, subjektů, pošty, úkolů a odkaz pro zobrazení modálního okna pro rychlé vyhledávání. Pokud je přihlášeným uživatelem administrátor, uvidí v nabídce lišty také odkaz na stránku administrace. V pravé části navigační nabídky je zobrazeno jméno přihlášeného uživatele společně s rozbalovacím menu umožňující změnu hesla a odhlášení z aplikace.

Grafické nákresy jednotlivých obrazovek v podobě drátěného modelu jsou obsaženy v sekci příloh této práce.

Kapitola 5

Implementace navrženého systému

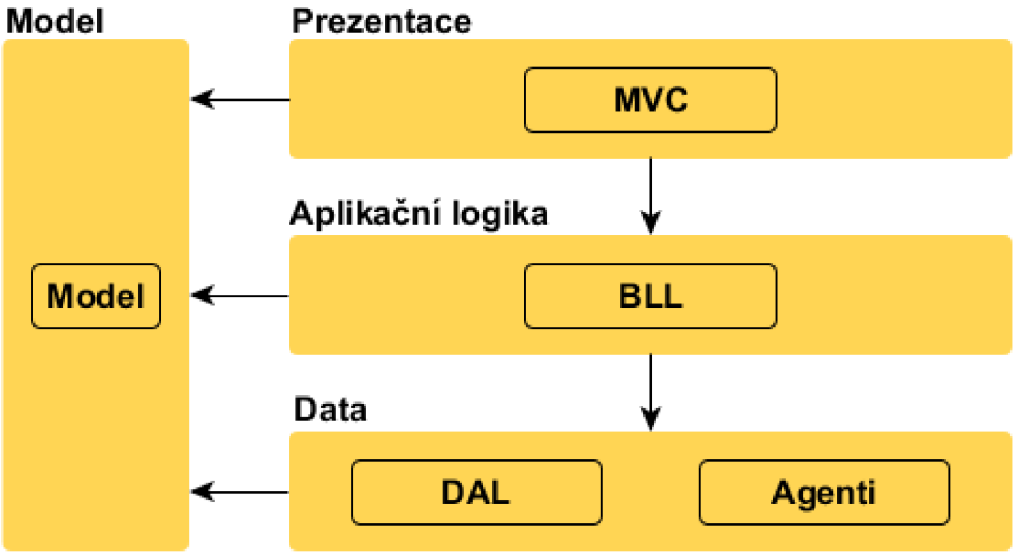
Kapitola implementace popisuje implementační detaily vytvořeného informačního systému. Nejprve je sepsána podkapitola o struktuře serverové části aplikace s popisem jednotlivých vrstev. Následují sekce věnující se struktuře klientské části aplikace, komunikaci mezi klientem a serverem a podrobnému popisu některých součástí systému. Kapitola je doplněna úsekem o použitých aspektech zabezpečení a uzavřena podkapitolou o tvorbě testovatelného kódu.

5.1 Struktura serverové části aplikace

V kapitole analýzy a návrhu je zmíněna struktura aplikace a součásti, které jsou obsaženy ve výsledném systému. Tyto komponenty jsou rozmístěny do implementační struktury systému, která vychází z vícevrstvé architektury. Taková architektura se vyznačuje tím, že spojuje související funkcionalitu uvnitř aplikace do jednotlivých vrstev, které mezi sebou komunikují pomocí nadefinovaného rozhraní. Nižší vrstvy architektury poskytují obecnější funkcionalitu a naopak vyšší vrstvy disponují aplikačně specifičtějšími službami. Vazba je poté vedena směrem od vyšší vrstvy k nižší. Taková architektura přináší následující výhody:

- **Oddělení zodpovědnosti:** jednotlivé části aplikace, které spolu souvisí, jsou rozděleny do vrstev, čímž usnadňují návrh a umožňují paralelní vývoj na odlišných částech aplikace.
- **Abstrakce:** jednodušší pohled na kompletní aplikaci s porozuměním rolí a zodpovědností jednotlivých vrstev a vztahů mezi nimi. Každou část aplikace je navíc možné analyzovat v izolaci.
- **Testovatelnost:** vícevrstvá architektura umožňuje testovat každý oddíl zvlášť, přičemž je zde méně závislostí mezi vrstvami pro tvorbu jednotkových testů.
- **Nahraditelnost:** tento přístup dovoluje také nahrazení vrstev aniž by byla ovlivněna jiná vrstva.
- **Znovupoužitelnost:** již vytvořené vrstvy je možné využít i v jiných aplikacích.

Aplikace implementovaná v rámci této práce se skládá z datové vrstvy, vrstvy aplikační logiky, prezentační vrstvy a modelu. Pro lepší ilustraci je tato architektura zobrazena na obrázku 5.1.



Obrázek 5.1: Vícevrstvá architektura aplikace.

5.1.1 Datová vrstva

Datová vrstva je nejnižší vrstva architektury, která je zodpovědná za získání dat pro vrstvu aplikační logiky. Vzhledem k tomu, že aplikace neoperuje pouze s daty z databáze, ale také s daty z datových schránek a rejstříku, je vrstva rozdělena do dvou sekcí.

Sekce DAL

Sekce DAL (data access layer) je zodpovědná za práci s databází. Umožňuje vkládat, číst, modifikovat a mazat data, přičemž tyto operace se souhrnně označují jako CRUD (create, read, update, delete) operace. Jádrem této části je ORM (object-relational mapping) framework nesoucí název Entity Framework[1], který umožňuje pracovat s relačními daty z databáze jako s doménovými objekty. Generování příslušných databázových dotazů je ponecháno knihovně a je úplně odstíněno od vyšších vrstev. Díky tomu je odděleno databázové schéma od doménových objektů aplikace, což přispívá udržovatelnosti a rozšiřitelnosti této vrstvy. Entity Framework má další zajímavou vlastnost a tou je tvorba databázového schématu z namodelovaných tříd (tzv. code first). Tato funkce se v aplikaci hodně využívá, neboť při každé změně kódu týkající se databázových tříd umí framework aktualizovat databázové schéma.

Důležitou roli uvnitř datové sekce hraje návrhový vzor repozitář. Repozitář je zodpovědný za obalení kódu pro dosažení dat a zprostředkovává komunikaci mezi aplikační a datovou vrstvou. Pro každou entitu v databázi je vytvořen repozitář, který se chová jako kolekce doménových objektů. Aplikační logika poté volá pouze rozhraní repozitáře a tím je oddělena od detailní implementace připojení k databázi, případně jinému úložišti. Úkolem repozitáře je naopak provést požadované konkrétní operace vůči databázi. V rámci aplikace je vytvořen také generický repozitář, který volá metody, jenž implementuje použitý Entity

[1]Dostupné z `https://entityframework.codeplex.com/`.

Framework. Repozitáře pro jednotlivé entity tedy volají generický repozitář, který je zodpovědný za provedení příslušných operací v databázi. Výhodou generického repozitáře je možnost jeho použití ve všech repozitářích.

Aby byl výčet funkcionality datové sekce úplný, je třeba také zmínit mapování mezi doménovými objekty a objekty použitými v datové části. Objekty použité uvnitř datové části obsahují také vlastnosti, které jsou použity pouze v rámci ORM frameworku a nikde dále v aplikaci nejsou potřeba. Proto jsou objekty získané z databáze mapovány na objekty domény, které obsahují pouze informace použité uvnitř dalších vrstev aplikace. Analogicky mapování probíhá i opačným směrem, z modelů aplikace do databázových objektů.

Sekce datových agentů

Sekcí datových agentů (v implementaci nese název *Services*) se označují komponenty, které zpřístupňují data z nedatabázových zdrojů. V tomto případě se jedná o data z datových schránek a data z administrativního rejstříku ekonomických subjektů. Tato část tedy obsahuje klienty pro připojení se k daným webovým službám a také komponenty, které mapují objekty služeb do objektů používaných uvnitř aplikace. Těmto klientům se budou více věnovat kapitoly v sekci implementace částí systému.

5.1.2 Vrstva aplikační logiky

Vrstva aplikační logiky (v aplikaci modul s názvem *BLL*) je mozkem celé aplikace. Obsahuje veškerou aplikační logiku, která je použita v rámci aplikace. Veškeré dopočítávání a získávání potřebných informací, změny stavů spisů, výpočty finančních náležitostí, schválení pošty pro soud, zápisy do auditu či vytváření úkolů je prováděno právě zde.

Metody této vrstvy jsou volány obvykle z řadičů uvnitř prezentační vrstvy, přičemž každému řadiči náleží jeden objekt aplikační vrstvy. Tyto objekty poté volají metody repozitářů nebo klientů pro splnění požadované funkcionality. Tvoří mezistupeň mezi prezentační a datovou vrstvou.

5.1.3 Prezentační vrstva

Prezentační vrstva (v implementaci má název *MVC*) je část systému, ve které dochází k interakci s uživatelem. Tato část systému tedy zpracovává požadavky klienta, deleguje je do dalších vrstev a zobrazuje výsledky dotazů v grafické formě. V tomto případě bude vrstva realizována pomocí návrhového vzoru MVC, kde za obsluhu událostí od klienta budou zodpovědné ovladače. Pro každou obrazovku aplikace je vytvořen jeden řadič, který volá příslušné objekty aplikační vrstvy. Po provedení operací nižších vrstev předává výsledek k zobrazení pohledu, který předaná data zobrazí. Klasický model z návrhového vzoru MVC je ve vytvářeném systému nahrazen ostatními vrstvami vícevrstvé architektury.

Vzhledem ke skutečnosti, že ovladače nemusí vracet pouze celé HTML stránky, ale také JSON data, je možné tuto vrstvu částečně označit i jako servisní vrstvu. Metody vracející JSON objekty jsou volány právě tehdy, když klient posílá AJAX požadavky na server.

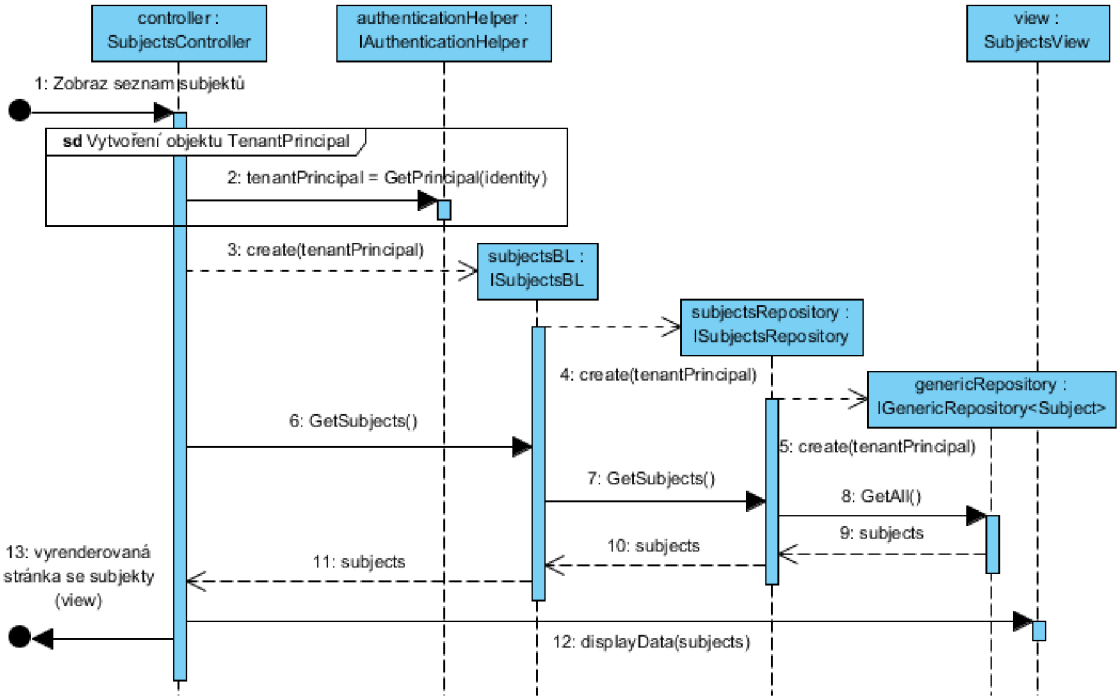
5.1.4 Model

Poslední vrstva, která je zobrazena na obrázku architektury aplikace a o které zatím nebyla řeč, je vrstva modelu. Tato část obsahuje veškeré objekty, které jsou používány uvnitř aplikace a které jsou předávány mezi vrstvami. Tyto objekty můžeme označit pojmem

DTO (data transfer object), jelikož neobsahují žádnou logiku a slouží pouze k přenosu informací. Obecně by měla prezentační vrstva používat vlastní objekty pro zobrazení a příjem dat z grafického uživatelského rozhraní a stejně tak datové vrstvy by měly používat vlastní objekty pro poskytnutí nebo příjem dat z jiných koncových bodů. Tyto objekty jednotlivých vrstev by měly být mapovány do doménových objektů obsažených v modelu. V aplikaci je tento přístup realizován pouze u datové vrstvy, kde jsou databázové modely mapovány do modelových objektů a naopak. Pro zobrazení a příjem dat z uživatelského rozhraní slouží modely domény. Výsledkem předávání objektů modelu mezi vrstvami je skutečnost, že všechny vrstvy obsahují závislosti do sekce modelu.

5.1.5 Komunikace mezi vrstvami

Typickým požadavkem ze strany klienta je provedení databázových operací a zobrazení databázových dat. Protože je to nejčastěji prováděná činnost v rámci systému, bude popsána detailněji na požadavku pro zobrazení seznamu subjektů v databázi úřadu. Sekvenční diagram komunikace mezi třídami odpovídající popisované operaci je uveden na obrázku 5.2.

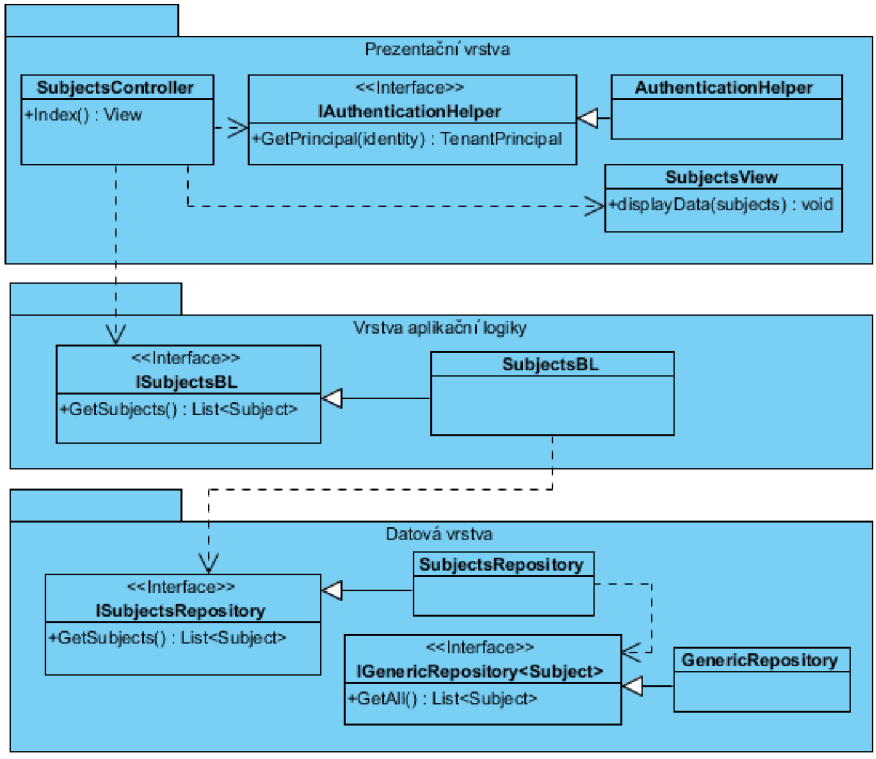


Obrázek 5.2: Sekvenční diagram pro operaci zobrazení seznamu subjektů.

Požadavek odeslaný ze strany klienta je přijat příslušným řadičem, v popisovaném případě je to řadič *SubjectsController*. Tento řadič nejprve ověří, zda je požadavek autentizovaný pomocí *AuthenticationHelper* třídy. Pokud ano, je navracen objekt *TenantPrincipal*, který obsahuje připojovací řetězec (tzv. connection string) potřebný pro připojení databázového klienta ke správné databázi. Třída *TenantPrincipal* je klíčová pro implementaci multitenance, je tedy detailněji popsána dále v podkapitole multitenance. Diagramu rámce vytvoření objektu TenantPrincipal odpovídá diagram 5.4. Pro nás je důležité, že vzhledem k multitenanci známe připojovací řetězec až za běhu aplikace, a tak jsou objekty nižších vrstev

vytvářeny až po získání připojovacího řetězce. Na obrázku můžeme vidět, že řadič vytváří objekt logické vrstvy *SubjectsBL* a ten poté dále iniciuje repozitář pro subjekty *SubjectsRepository*. Všechny repozitáře využívají generického repozitáře *GenericRepository*, který volá metody použité knihovny Entity Framework. Všem vytvářeným objektům je v konstruktoru předán objekt *TenantPrincipal*. Následně dochází v řadiči k zavolání metody *GetSubjects*, která získá seznam subjektů z databáze pomocí právě vytvořených objektů nižších vrstev. Po získání seznamu subjektů jsou poslány objektu *SubjectsView*, který je zodpovědný za vykreslení dat do obrazovky. Klientovi se poté vrací HTML stránka zobrazující seznam subjektů. Pokud by klient požadoval pouze data bez HTML stránky, vynechá se poslední krok předání dat obrazu, data jsou serializována do JSON formátu a poslána na výstup. Záměrně jsou na obrázku uvedeny rozhraní namísto tříd objektů, jelikož kvůli testovatelnosti jsou třídy odkazovány právě pomocí rozhraní.

Pro představu, do kterých vrstev popsané komunikující objekty náleží, je zde uveden i druhý obrázek, který obsahuje diagram tříd použitých v popsaném sekvenčním diagramu 5.3.



Obrázek 5.3: Diagram tříd zúčastněných při operaci zobrazení seznamu subjektů

Pro jednoduchost zobrazeného diagramu tříd neobsahuje diagram veškeré metody objektů a rozhraní, ale pouze metody, které byly použity při požadavku pro zobrazení všech subjektů. Navíc objekty, jenž implementují rozhraní, samozřejmě obsahují metody rozhraní, což také není v obrázku ze stejného důvodu vykresleno.

Z diagramu tříd vidíme, že prezentační vrstva obsahuje objekty *SubjectsController*, který má závislosti na rozhraních *IAuthenticationHelper*, *ISubjectsBL* a na obrazovce *SubjectsView*. Díky závislosti na rozhraní *ISubjectsBL* je vrstva prezentační logiky závislá na vrstvě

aplikační logiky. Rozhraní *ISubjectsBL* je implementováno třídou *SubjectsBL*, která má závislost na rozhraní *ISubjectsRepository*, která je obsažena uvnitř datové vrstvy. Díky tomu obsahuje vrstva aplikační logiky závislost na datové vrstvě. Objekt *SubjectsRepository* následně využívá rozhraní *IGenericRepository*. V obrázku není obsažena sekce modelu, která obsahuje objekt *TenantPrincipal*, díky čemuž je každá vrstva závislá také na modelové vrstvě.

Na podobném principu je vystavěna celá struktura aplikace. Do prezentačních vrstev náleží ovladače obrazovek, které závisí na aplikačních vrstvách. Objekty aplikačních vrstev poté rozhodují, které repozitáře z datové vrstvy budou využívat a případně zda využijí sekci agentů. Demonstrovaný příklad využíval pouze repozitáře subjektů, ale v implementaci programu používá každá třída aplikační vrstvy repozitářů více.

5.2 Struktura klientské části aplikace

Vedle hlavní části aplikace, která je umístěna na straně serveru, obsahuje aplikace také klientskou část, která je prováděna v prohlížeči uživatele systému. Ta kromě definice vzhledu obsahuje také jednoduchou logiku pro zobrazování tlačítek, rozbalování lišt, kontrolu vstupních údajů a především pro asynchronní dotazy na server ve formě JavaScriptu využívajícího AngularJS framework. Tento framework má však vlastní strukturu, kterou musí projekt dodržovat.

Každá obrazovka vytvořená v rámci aplikace má kromě serverového řadiče přidělen i řadič na straně klienta. Takový řadič obsahuje veškerou funkcionalitu využitou uvnitř HTML stránky, která obsahuje především obslužné funkce pro prvky uživatelského rozhraní. Řadiče obsahují velmi často také kolekce dat, které jsou renderovány do výsledného HTML kódu. Tyto kolekce jsou využity především při zobrazení dat získaných z asynchronních dotazů na server. Po aktualizaci kolekce uvnitř řadiče dochází k okamžitému zobrazení dat do obrazu a toho je využíváno při získání dat ze serveru bez přenačtení celé stránky. Pro možnost provádění asynchronních dotazů se dostáváme k další komponentě, kterou je tzv. služba (service v terminologii AngularJS). Tuto službu využívají všechny klientské řadiče, které potřebují asynchronně komunikovat se serverem. Velmi důležitým rysem řadičů je možnost vyvolat modální okna. Tato schopnost je v aplikaci velmi často využívána, jelikož veškeré formuláře jsou zobrazovány právě pomocí modálních oken. Každé modální okno má také vytvořen svůj vlastní řadič, který se opět stará o obsluhu tlačítek.

Souhrnně, pro klientskou stranu jsou vytvořeny řadiče obrazovek, řadiče modálních oken, služba pro tvorbu asynchronních HTTP dotazů a modul, který instanciuje námi vytvořené komponenty. Všechny tyto soubory jsou umístěny ve složce *Scripts* v prezentační vrstvě aplikace.

5.2.1 Použité moduly

V klientské části aplikace byly využity následující AngularJS moduly:

- **UI Bootstrap**[2]**:** tento modul nabízí komponenty uživatelského rozhraní, kterým definuje chování AngularJS a vzhled Bootstrap framework. Modul nabízí širokou škálu komponent od nejrůznějších tlačítek, vysunovacích lišt, záložek, rozevíracích nabídek až po stránkovače apod. Na těchto komponentách je postaven vzhled implementované aplikace, přičemž mezi použité komponenty patří modální okna, tlačítka, lišta záložek

a rozbalovací nabídka.

- **Smart table**[3]**:** tabulky v uživatelském rozhraní jsou vytvořeny pomocí modulu s názvem Smart Table. Tato komponenta nabízí řešení pro tvorbu tabulek poskytující funkce vyhledání, filtrování, výběru záznamů, stránkování a rozhraní pro tvorbu vlastních funkcí. Pro zobrazení dat se tabulce předá pouze kolekce dat. Tabulky použité v aplikaci obsahují funkce filtrování, řazení, stránkování a výběr počtu zobrazených dat na stránku. Všechny tyto operace jsou prováděny na straně klienta.

- **ngMessages**[4]**:** tento modul slouží pro zobrazení a ukrytí informačních zpráv, které souvisí se stavem objektu, kterému naslouchá. V aplikaci je modul použit při kontrole vstupních údajů uživatele, kdy při chybném vstupu se u daného pole zobrazí informační zpráva.

- **ngFileUpload**[5]**:** modul využitý pro nahrání souboru z klienta na server.

5.3 Komunikace mezi klientem a serverem

Klient při komunikaci se serverem kombinuje klasické synchronní HTTP požadavky s asynchronními AJAX požadavky. Při přepnutí na novou stránku dochází ke klasické žádosti o celou HTML stránku. Tato stránka je sestavena na straně serveru a následně poslána klientovi, který jej zobrazí. Výsledkem je doba odezvy, při které se uživateli přenačítá celá obrazovka a uživatel není schopen s aplikací pracovat. Při snaze zefektivnit práci s aplikací jsou v některých částech nahrazeny synchronní požadavky na server asynchronními. Těmito částmi jsou modální okna. Kdykoliv, když uživatel zasílá vyplněný formulář z modálního okna, ať už pro vytvoření záznamu, jeho úpravu nebo vyhledání, vytváří se asynchronní dotaz, jehož výsledek nenutí překreslovat celou stránku. Asynchronní požadavky probíhají také při získání dat pro formulář. Při zobrazení detailu např. úkolu je proveden požadavek, který vrátí požadovaná data a zobrazí je ve formuláři.

Jak již bylo popsáno výše, pro tvorbu klientských asynchronních dotazů slouží komponenta s označením služba (z terminologie AngularJS). Tato komponenta zasílá *get*, *post* a *delete* HTTP požadavky na server a následně je také zpracovává. Veškerá data v rámci této komunikace jsou ve formátu JSON. Na straně serveru ovšem musí být implementováno rozhraní, které umožňuje tyto žádosti obsloužit. Takové metody ovšem nevrací jako odpověď vyrenderovanou HTML stránku, ale JSON objekty. Ovladače tedy kombinují standardní MVC rozhraní s rozhraním podobající se REST API, které implementuje požadované chování pro HTTP *get*, *post*, případně *delete* požadavky. Pokud se tedy jedná o získání dat, klient volá metodu *get*, pro uložení dat volá metodu *post* a pro smazání dat metodu *delete*. Vynechána je metoda *put*, která slouží pro editaci záznamů. Naše řešení provádí editaci záznamů metodou *post*, kdy k rozlišení operace dochází až na úrovni repozitáře.

Zajímavé řešení je použito při zobrazení stránky pro detail spisu. Vzhledem k tomu, že zobrazovaných dat v této obrazovce je spousta, dochází k postupnému načítání těchto dat. Nejprve je ze serveru získána obrazovka s hlavními údaji o spise, přičemž další data týkající se souvisejících dokumentů, finančních náležitostí, úkolů a součinností se načítají později

[2]Dostupné z `https://angular-ui.github.io/bootstrap/`.
[3]Dostupné z `http://lorenzofox3.github.io/smart-table-website/`.
[4]Dostupné z `https://docs.angularjs.org/api/ngMessages/directive/ngMessages`.
[5]Dostupné z `https://github.com/danialfarid/ng-file-upload`.

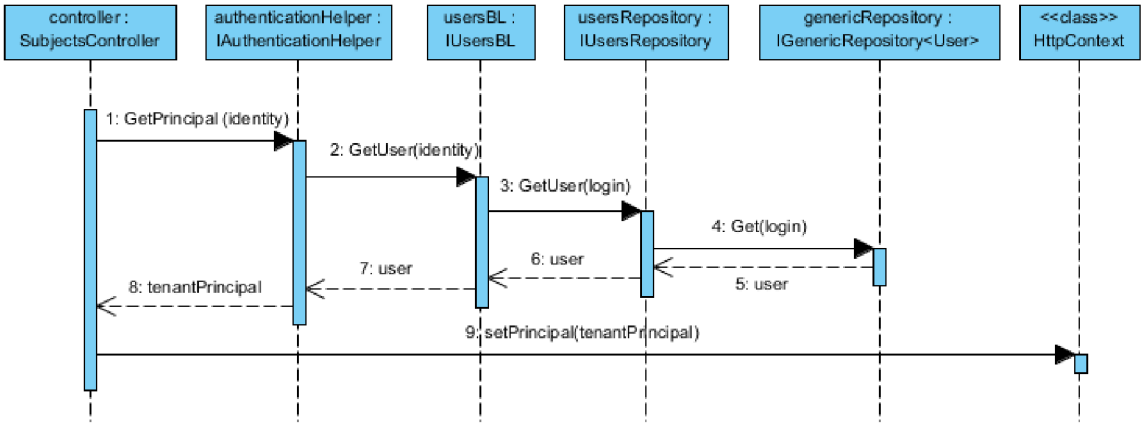
při provádění asynchronních volání. Výhodou řešení je fakt, že se uživateli zobrazí stránka dříve a ten poté má pocit, že aplikace s ním lépe spolupracuje.

5.4 Implementace částí systému

V této podkapitole jsou uvedeny implementační detaily částí, které se týkají multitenance, workflow spisu, správy dokumentů, datových schránek, administrativního registru ekonomických subjektů a auditu.

5.4.1 Multitenance

Pro zajištění funkcionality multitenance musí být upravena činnost při příchodu serverového požadavku tak, aby se sekce pro získání dat byla schopna připojit k požadované databázi. Po příchodu autentizovaného požadavku na server je v hlavní databázi vyhledán záznam uživatele, který odpovídá uživateli autentizovaného požadavku. Tento záznam z databáze obsahuje tzv. připojovací řetězec (connection string), který je klíčový pro připojení vrstvy DAL k příslušné databázi úřadu a obsahuje údaje potřebné k připojení do datové schránky. Tyto informace jsou předány objektu *TenantPrincipal*, který je přiřazen objektu *HttpContext* obsahující informace o aktuálním serverovém požadavku. Díky tomu je vytvořený *TenantPrincipal* dostupný v celém webovém kontejneru a je tak možné uvnitř jednotlivých řadičů získat jeho data. Celý tento proces znázorňuje sekvenční diagram na obrázku 5.4.



Obrázek 5.4: Sekvenční diagram pro získání objektu *TenantPrincipal*.

Za tvorbu objektu *TenantPrincipal* je zodpovědný objekt *AuthenticationHelper*, který se dotazuje databáze na přihlášeného uživatele. Vidíme, že využitý princip je podobný jako při provedení požadavku na databázová data z obrázku 5.2, ale za aplikační vrstvu zde vystupuje objekt *UsersBL*. Ovšem je zde i jeden rozdíl. Při dotazu do databáze úřadu je nutné získat nejprve připojovací řetězec a až poté jsou vytvářeny objekty aplikační a datové vrstvy. V tomto případě jsou tyto objekty již vytvořené, jelikož repozitáře se připojují do hlavní databáze, která je specifikována v konfiguračním souboru.

Každý řadič, který získává data z databáze úřadu, předává připojovací řetězec dále do aplikační vrstvy, jenž ho opět předává repozitářům pro připojení ke konkrétní databázi. Tímto způsobem je docíleno schopnosti aplikace přepínat připojení mezi požadovanými

databázemi úřadu. Připojovací řetězec je předán nižším vrstvám pomocí rozhraní *IContextUser*, které implementuje třída *TenantPrincipal*. Toto rozhraní poskytuje také informace o identifikátoru přihlášeného uživatele a roli přihlášeného uživatele, což je užitečné v případě, kdy aplikační logika určuje operace, které jsou dostupné roli přihlášeného uživatele.

Kromě již zmíněného připojovacího řetězce obsahuje třída *TenantPrincipal* také informace k potřebné k připojení do datové schránky. Každý úřad využívá svoji datovou schránku a tak je nutné kromě připojovacího řetězce předávat i údaje k připojení do datové schránky. V řadiči, který zprostředkovává komunikaci s datovými schránkami dochází opět k předání těchto údajů do aplikační vrstvy, která se postará o vytvoření agenta se správnými připojovacími informacemi. Údaje jsou předány nižším vrstvám pomocí rozhraní *IContextMailBoxData*, které je implementováno třídou *TenantPrincipal*, jež poskytuje přihlašovací jméno a heslo k připojení.

Popsaný cyklus získání údajů z hlavní databáze pro přihlášeného uživatele probíhá při každém požadavku na server a tím se zvyšuje čas obsluhy každého požadavku. Tuto skutečnost je dobré mít na paměti v případě, pokud by bylo nutné optimalizovat proces rychlosti získání odpovědi. Pro optimalizaci by bylo možné využít mezipaměť, která by obsahovala informace o naposledy přihlášených uživatelích a tím by ušetřila čas při opakovaném dotazování stejného uživatele. V našem případě však není tato optimalizace nutná, neboť čas získání odpovědi je postačující, i vzhledem k tomu, že se nepředpokládá vysoká frekvence dotazů na server.

5.4.2 Workflow spisu

Workflow je implementován pomocí přepínání stavů exekučního spisu, které korespondují s obrázkem 3.1. K přepnutí stavu dojde přímou změnou stavu uživatelem, po přiřazení dokumentu určitého typu ke spisu, nebo po vypršení termínu daného stavu plánovanou úlohou. Na přímé změně uživatelem není nic až tak zvláštního, pouze dojde k přepsání stavu příslušného spisu v databázi. Přiřazení dokumentu ke spisu už je malinko zajímavější, neboť při přiřazení dokumentu ke spisu dochází k porovnání typu dokumentu se stavem spisu. Pokud je tedy například spis ve stavu *spis založen* a přiřazený dokument typu *přijetí usnesení*, přepne se stav spisu do stavu *exekuce zahájena*. Všechna tato činnost se odehrává ve třídě *FilesBL*.

Součástí workflow jsou i přechody, které závisí na vypršení časové lhůty. Cílem tedy je, aby docházelo každý den k pravidelné kontrole, zda daný termín již nevypršel a zda není potřeba přepnout stav spisu. Problémem ovšem je, že webová aplikace není určena k tomu, aby prováděla a starala se o spouštění pravidelného procesu na pozadí. Z toho důvodu je vytvořena konzolová aplikace, která obstarává potřebnou obsluhu přepnutí stavu. Tato aplikace je spouštěna pravidelně každý den plánovačem úloh v systému Windows, který se stará o její životní cyklus.

Popsaná konzolová aplikace se nachází v rámci projektu ve složce *ScheduledTasks* a má název *CheckFileDates*. Tento proces si po spuštění vytvoří instance třídy *FilesBL* pro každou databázi úřadu, ve kterých volá metody pro kontrolu dat uvnitř spisů každé databáze. K přepnutí spisu následně dochází, pokud již vypršela lhůta pro dobrovolné plnění nebo pokud byl překročen termín konce odkladu exekuce.

5.4.3 Správa dokumentů

Pro možnost skladování dokumentů se nabízely dvě možnosti. Ukládání nestrukturovaných dat v souborovém systému s podporou databáze (v prostředí MS SQL databáze se tato

volba nazývá filestream) nebo klasické uložení dokumentů do relační databáze. Zadavatel práce předpokládá, že do systému budou vkládány dokumenty v řádu jednotek MB. Pro tyto velikosti jsou časy pro vložení a získání dokumentu z databáze srovnatelné u obou srovnávaných typů úložišť. Pokud jsou však dokumenty větší, výhoda je na straně souborového systému, naopak pokud jsou dokumenty menší, je výhodnější uložení dokumentů v databázi. Problém ovšem nastává u použité knihovny pro přístup k databázi. Entity Framework nepodporuje souborový systém (filestream) při použité technice vývoje mapování z definovaných tříd do databázových objektů (code first). Zde použitá knihovna poskytuje podporu pouze pro uložení dokumentu přímo v databázi. Další faktor, který hraje v neprospěch uložení souboru v souborovém systému je složitější konfigurace podpory v databázi. Z toho důvodu jsou dokumenty v rámci aplikace skladovány přímo v relační databázi.

Pro skladování dokumentů v tabulce je využit sloupec s datovým typem *varbinary(max)*, který si alokuje potřebnou délku místa v závislosti na velikosti souboru. Pro správnou interpretaci uloženého souboru je nutné v databázovém záznamu uchovávat také název souboru a jeho popisovač určující typ dokumentu. Pro získání souboru z databáze je tedy nutné získat všechny tyto informace, aby byl uživateli dodán soubor se správným názvem a typem dokumentu.

5.4.4 Datové schránky

Za účelem práce s datovými schránkami byla vytvořena testovací datová schránka, jenž je určena pro dodavatele třetích aplikací. Tato testovací datová schránka poskytuje stejné aplikační rozhraní jako běžná datová schránka a je tedy na ní možné ověřit interakci s datovými schránkami před nasazením do skutečného systému.

Webové služby datových schránek[6] poskytují čtyři základní rozhraní, přičemž první z nich popisuje služby související s přístupem do datových schránek (rozhraní je popsáno souborem *db_acess.wsdl*), druhé slouží pro vyhledávání datových schránek (rozhraní je popsáno souborem *db_search.wsdl*), třetí a čtvrté slouží pro manipulaci s datovými schránkami (popsáno soubory *dm_info.wsdl* a *dm_operations.wsdl*). Pro nás jsou nejdůležitější poslední dvě rozhraní popisující manipulaci se schránkou a obsahující např. operace pro získání zpráv, získání detailu zprávy, odeslání zprávy atd. Všechny tyto služby umožňují komunikaci pomocí protokolu SOAP nad HTTPS, data jsou tedy přenášena ve formátu XML.

Implementovaná aplikace obsahuje SOAP klienty pro všechny tyto služby, které jsou umístěny v sekci agentů uvnitř DAL vrstvy. Aplikace však používá pouze metody pro odeslání zprávy *CreateMessage*, získání seznamu obdržených zpráv *GetListOfReceivedMessages*, získání seznamu odeslaných zpráv *GetListOfSentMessages* a pro stažení celé zprávy společně s přiloženými soubory *MessageDownload*. Poslední použitou metodou je metoda *GetOwnerInfoFromLogin*, ta je však pouze použita pro ověření, zda je klient schopen připojit se ke službám datových schránek.

Jelikož je pro odeslání požadavku službě nutné vytvořit správné parametry, je řešení v rámci aplikace rozděleno do dvou vrstev. Nižší vrstvu obstarává třída *MailBoxClient*, která se stará o správu připojení, tedy nejprve iniciuje připojení ke službě, poté odesílá požadavek, přijímá odpověď a ukončuje spojení. Vyšší vrstvu tvoří třída *MailBoxHelper*, která mapuje parametry volání do objektů, které využívá služba a naopak výsledky volání služby do objektů, které jsou dále využity v aplikaci.

[6] Dostupné z `https://www.datoveschranky.info/dulezite-informace/provozni-rad-isds`.

5.4.5 Administrativní registr ekonomických subjektů

Administrativní registr ekonomických subjektů (ARES) je další forma součinnosti realizována systémem. Webová služba rejstříku ARES[7]nabízí nejen SOAP rozhraní pro metodu POST, ale také REST rozhraní pro metodu GET. Vytvořená aplikace využívá REST rozhraní, kde pro dotazy není nutnost digitálního podpisu. Služba nabízí výpisy z více rejstříků, ovšem v rámci vytvořené aplikace se využívá standardní výpis, který obsahuje vše důležité ze všech rejstříků. Zasílaný dotaz vypadá následovně:

```
http://wwwinfo.mfcr.cz/cgi-bin/ares/darv_std.cgi?ico=27074358
```

K adrese služby je připojen url parametr *ico*, který definuje IČ lustrovaného subjektu. Rozhraní podporuje také vyhledávání pomocí názvu firmy, my však využijeme pouze vyhledání podle IČ. Odpověď služby je zasílána ve formátu XML.

Klient pro připojení ke službě je umístěn v sekci agentů ve třídě *AresClient*. Pro implementaci volání byla využita knihovna RestSharp[8], která nabízí jednoduché rozhraní pro vytvoření a použití REST klienta.

5.4.6 Audit a záznam událostí

Veškeré databázové změny týkající se spisu a subjektů jsou zaznamenávány do auditu. V našem případě se jedná o databázovou tabulku obsahující informace o datu provedení, typu operace, detailní výpis změny, uživateli a případně přiřazení ke spisu či subjektu. Vytvoření spisu, subjektu, finance, dokumentu, titulu, to vše je potřeba zaznamenat. Veškerá logika rozhodující o zaznamenání do auditu by podle vzoru implementace ostatních částí logiky měla náležet aplikační vrstvě, jenomže úseky rozhodující o tvorbě záznamů do auditu by byly rozprostřeny přes všechny části aplikační vrstvy. Proto bylo využito návrhového vzoru vydavatel-odběratel (observer). Každý repozitář, jenž implementuje změny v databázi, implementuje událost změny záznamu, kterému předá upravený záznam. Těmto událostem naslouchá třída *AuditBL*, která obsahuje náležitou obsluhu pro události repozitářů. Obslužné metody této třídy implementují vložení záznamu auditu do databáze. Úkolem aplikační vrstvy je pouze zaregistrovat obsluhu událostí ze třídy *AuditBL* požadovaným repozitářům, jejichž operace mají být zaznamenány v auditu. Tímto způsobem se části aplikační logiky elegantně zbavily zodpovědnosti za vytváření záznamů do auditu.

Zápis do žurnálu naopak slouží pro osoby, které se starají o správu aplikace. Do žurnálu jsou zaznamenávány informace o přihlášených a odhlášených uživatelích provádějící serverové požadavky a především o chybových stavech, které se vyskytnou při běhu aplikace. Primárním cílem takového žurnálu je tedy poskytnutí detailní informace o chybě v systému. Zápis do logu je prováděn pomocí knihovny log4net[9]. Velkou výhodou použité knihovny je možnost nastavení způsobu záznamu událostí v konfiguračním souboru. Další zajímavou vlastností je možnost zvolit typ zapisované informace, zda se jedná pouze o informativní zápis, ladící výpis, varování, chyba, fatální chyba atd. Následně je možné nastavit, zda chci do žurnálu vypsat pouze chyby, fatální chyby, varování nebo všechny možné výpisy.

[7]Dostupné z `http://wwwinfo.mfcr.cz/ares/`.
[8]Dostupné z `http://restsharp.org/`.
[9]Dostupné z `https://logging.apache.org/log4net/`.

5.5 Zabezpečení

Součástí implementace aplikace jsou také aspekty zabezpečení systému, kterými jsou autentizace a autorizace, použití HTTPS protokolu a validace vstupních dat.

5.5.1 Autentizace a autorizace

Autentizace je realizována pomocí modulu ASP.NET forms authentication. Tento modul se stará o to, aby se k dalšímu zpracování dostal pouze autentizovaný požadavek a zároveň napomáhá s udržováním a ověřováním autentizačního žetonu uvnitř HTTP požadavku. Pokud žádá o zdroj (ať už o webovou stránku, případně data) neautentizovaný uživatel, modul ho přesměruje na stránku k zadání přihlašovacích údajů. Aplikace neumožňuje práci se systémem, pokud není uživatel autentizován. Poslední využitou funkcí modulu je odhlášení uživatele po určité době nečinnosti (v našem případě 30 minut).

Při verifikaci uživatelského jména a hesla dochází nejprve k vyhledání uživatelského jména v databázi. Pokud je vyhledáno, dochází k verifikaci hesla pomocí knihovny Crypto[10]. Protože hesla v databázi jsou generována pomocí této knihovny algoritmem popsaným v RFC 2898, je pro jejich verifikaci nutné stejným způsobem vytvořit hash uživatelem zadaného hesla, aby bylo možné zadané hesla porovnat. Knihovna poskytuje metodu pro takové porovnání hesel a je tedy použita pro verifikaci zadaného hesla.

Pokud bylo zadané heslo správné, je vytvořen autentizační žeton. Tento žeton je udržován jako cookie v klientském prohlížeči. Další následná komunikace ze strany klienta obsahuje tento žeton v HTTP požadavku a tím umožňuje webové aplikaci identifikovat uživatele po jeho přihlášení.

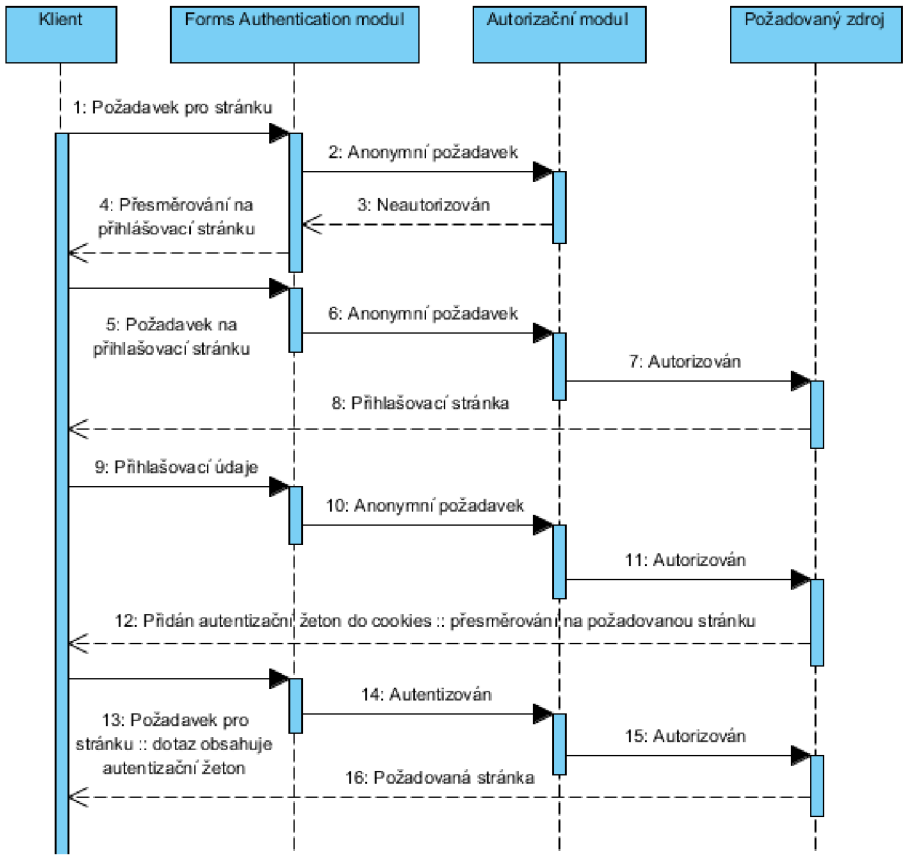
Obrázek 5.5 znázorňuje funkci modulu Forms Authentication ve formě sekvenčního diagramu. Pokud přijde neautorizovaný požadavek na stránku, modul ho přesměruje na přihlašovací stránku. Pro stránku přihlášení není nutná autentizace ani autorizace, je tedy dostupná anonymnímu uživateli. Uživatel vyplní přihlašovací údaje a pokud jsou správně, je do komunikace přidán autentizační žeton a uživatel je přesměrován na požadovanou stránku. Požadavek s autentizovaným žetonem je již autentizován a pokud je také pro zdroj autorizován, dochází ke stažení požadovaného zdroje.

Autorizace probíhá po ověření autentizace uživatele a určuje, zda má přihlášený uživatel právo pro získání zdroje. Jak již bylo popsáno v kapitole o multitenanci, po autentizaci dochází k vytvoření objektu *TenantPrincipal*, který kromě údajů potřebných pro multitenanci disponuje také údaji pro ověření role přihlášeného uživatele, které mu jsou předány ze záznamu uživatele z databáze. Tento objekt implementuje rozhraní *IContextUser*, které obsahuje predikátové metody *IsAdmin* a *IsExecutor*. Voláním těchto metod aplikace zjišťuje, zda má přihlášený uživatel právo na prováděné operace. Každý autentizovaný uživatel disponuje minimálně rolí zaměstnance úřadu, a tedy kontrola autorizace je aplikována pouze pro zdroje exekutora či administrátora systému.

5.5.2 Další aspekty zabezpečení

Komunikace mezi klientem a serverem probíhá pomocí protokolu HTTPS, který zabezpečuje spojení, chrání před odposloucháváním, podvržením dat a umožňuje ověřit identitu druhé strany. Přenášená data jsou šifrována pomocí protokolu SSL. Pro vynucení podpory

[10] Dostupné z `https://msdn.microsoft.com/en-us/library/system.web.helpers.crypto.aspx`.



Obrázek 5.5: Ověření autentizace pomocí modulu Forms Authentication [10].

komunikace přes protokol HTTPS je použit atribut *RequireHttpsAttribute*, který vynutí nezabezpečený HTTP požadavek přeposlat pomocí HTTPS. K možnosti použití protokolu SSL je nutné, aby server měl zřízený certifikát, který pomůže klientovi ověřit autentičnost serveru. Pro účely vývoje a testování aplikace byl použit certifikát podepsaný sám sebou, který poskytl vývojový server IIS. Nicméně pro nasazení aplikace do produkce bude nutné si zřídit nový certifikát podepsaný certifikační autoritou.

Kromě použitého protokolu byly nastaveny bezpečnostní vlastnosti také pro použité autentizační cookies. Jednou z vlastností je použití příznaku *requireSSL*, který zaručí, že cookies jsou přiloženy pouze k zabezpečenému připojení. Pokud by se tedy náhodou stalo, že by požadavky byly vyslány nezabezpečeným protokolem, cookies nebudou přiloženy a není je tedy možné získat třetí osobou. Druhou vlastností je příznak *httpOnly*, díky které není možné přečíst cookies lokálně v jazyce Javascript. To může být užitečné při eliminaci XSS útoku.

Pro ověření vstupních dat je implementována validace vstupu na straně klienta. Před každým odesláním formuláře je ověřeno, zda data jsou očekávaného typu, zda datum splňuje očekávaný formát, zda povinná pole jsou vyplněná apod. Pokud nejsou tyto podmínky splněny, je uživatel vyzván k úpravě vstupních dat a formulář není odeslán.

Dalším projevem zabezpečení aplikace je prevence SQL injection útoku. Díky tomu, že databázové dotazy jsou sestavovány pomocí lambda výrazů, každý parametr je do vy-

tvořeného databázového dotazu vkládán jako parametr dotazu a nikoliv jako řetězec dotazu. O tuto výhodu se tedy postaralo využití lambda výrazů pro dotazy generované pomocí použité knihovny Entity Framework.

5.6 Testovatelný kód

Součástí zadání práce je vytvořit jednotkové testy pro klíčové části programu. Ovšem aby bylo možné dané části kódu testovat, musely být napsány tak, aby byly testovatelné. Dobrým začátkem pro testovatelnost kódu je jeho rozdělení do vrstev se společnou zodpovědností. Tato podmínka je díky vícevrstvé architektuře splněna. Nicméně k dosažení testovatelného kódu je potřeba navíc použít následující techniky:

- **Vložení závislostí (dependency injection):** vložení závislostí je technika, která vkládá závislost (komponentu) do objektu, který jej bude používat. Díky tomu daný objekt nezodpovídá za životní cyklus vkládané komponenty a je tedy odstíněn od jeho správy. Výhodou tohoto přístupu je skutečnost, že objektu je možno předat několik různých komponent, která splňují očekávání objektu, přičemž každá komponenta může poskytovat odlišné služby. Závislost je možné do objektu vložit buďto konstruktorem, nastavovací metodou (tzv. setter) nebo případně pomocí rozhraní.

- **Volná vazba (loose coupling):** pokud není použita volná vazba, objekt má přímou závislost na konkrétní třídě, která poskytuje požadované chování. Tato závislost však nemůže být nahrazena aniž by tato konkrétní třída nebyla změněna. Volná vazba se vyznačuje tím, že namísto odkazování konkrétní třídy odkazuje na rozhraní, které může být poté implementováno mnoha třídami. Jakákoliv třída implementující dané rozhraní může být použita jako závislost třídy, aniž by bylo potřeba tuto třídu měnit.

V přiloženém úryvku kódu 5.6 je ukázka testovatelného kódu pro třídu *SubjectsBL*, která má na starost aplikační logiku subjektů. Tato ukázka je dosti zjednodušená, jelikož ve skutečnosti obsahuje více závislostí a metod, nicméně pro ilustraci testovatelného kódu je plně dostačující.

V tomto úryvku najdeme obě popisované vlastnosti. Vložení závislostí je zprostředkováno pomocí konstruktoru třídy *SubjectsBL*, do kterého je parametrem vložena závislost rozhraní repozitáře pro subjekty *ISubjectsRepository*. Tato závislost je následně předána do atributu třídy. Tím, že atributem třídy *SubjectsBL* je rozhraní, a nikoliv konkrétní třída, se dostáváme k volné vazbě. Díky této vlastnosti jsme třídu schopni testovat, protože ji můžeme předat testovacího dvojníka v podobě mock, případně stub objektu. Aby bylo možné testovat i třídy, které mají referenci na tuto třídu subjektů, je nutné vytvořit rozhraní, které tato třída bude implementovat. Tím bude docíleno možnosti vytvářet testovací dvojníky pro třídu subjektů.

Testovací dvojníky je možné vytvořit pouze pro třídy implementující rozhraní. Pokud chceme testovat třídu, která volá statickou třídu (případně třídu, která neimplementuje rozhraní), nastane problém. Řeší se to tak, že pro statickou třídu je vytvořena třída navíc, která obaluje používané metody této statické třídy a implementuje nové rozhraní. Testovaná třída poté nevolá metody statické třídy přímo, ale volá metody rozhraní nově vytvořené třídy. Díky tomu jsme již schopni třídu otestovat pomocí testovacích dvojníků.

V rámci vývoje celé aplikace byla snaha o vytvoření testovatelného kódu, která používá praktiky vložení závislostí a volnou vazbu. Této možnosti napomáhá také použitá vrstvená

```
public class SubjectsBL
{
    /* zavislost */
    private ISubjectsRepository _subjectsRepository;

    /* konstruktor s vlozenim zavislosti */
    public SubjectsBL(ISubjectsRepository subjectsRepository)
    {
      _subjectsRepository = subjectsRepository;
    }

    /* metoda volajici rozhrani zavislosti */
    public Subject GetSubjectById(int id)
    {
      return _subjectsRepository.GetSubjectById(id);
    }
}
```

Obrázek 5.6: Ukázka testovatelného kódu.

architektura aplikace. Díky těmto technikám bylo možné vytvořit automatizované jednotkové testy pro klíčové části aplikace. Navíc, z důvodu vytváření testovacích dvojníků, byla při implementaci aplikace dána přednost tvorbě instančních tříd před těmi statickými. Jediný případ, kdy se používají statické třídy uvnitř aplikace, jsou statické třídy třetích stran a .NET knihoven. Pro každou takovou statickou třídu je vytvořena třída, jenž ji obaluje a umožňuje tak testování.

Kapitola 6

Testování vytvořeného systému

Kapitola testování vytvořeného produktu se věnuje popisu automatizovaných a manuálních testů, které byly prováděny pro ověření správné funkčnosti systému. Podkapitola automatizovaného testování popisuje vytvořené jednotkové a integrační testy a podkapitola manuálních testů obsahuje přehled provedených akceptačních testů.

Vytvořená aplikace byla testována automatickými i manuálními testy. Vzhledem k tomu, že nevíme, zda implementovaný produkt bude mít delší životnost, u kterého by vzniklo více verzí, nemělo pro nás význam vytvářet automatizované systémové testy (např. v podobě UI testů). Proto byly vytvořeny pouze automatizované jednotkové a integrační testy, které nám pomohly už při fázi vývoje aplikace. Těmito testy byly pokryty důležité části aplikace, které jsou zodpovědné za aplikační logiku a také oddíly, u kterých hrozí výskyt vážných chyb.

Ostatní testy byly prováděny manuálně. Sloužily k ověření, že systém pracuje správně jako celek a že aplikace splňuje funkční požadavky. V rámci testování aplikace byl kladen důraz především na akceptační testy, které byly provedeny po ukončení implementační fáze. Těmito testy bylo ověřeno, že vytvořený produkt splňuje stanovené požadavky a zároveň dokládá výpis funkcí, které byly aplikací implementovány. V průběhu vývoje byly prováděny také funkční systémové a integrační testy, které sloužily pro kontrolu správnosti implementace.

6.1 Automatizované testování

V rámci vytvářené aplikace byly vytvořeny automatizované jednotkové a integrační testy. Tyto testy pomohly nejen při fázi vývoje a úprav kódu, ale také ve fázi zpětného ověření funkčnosti. Obecně je snaha o pokrytí co nejvíce segmentů kódu, nicméně v této práci jsme se zaměřili pouze na klíčové části systému, u kterých hrozí výskyt závažných chyb. Tyto testy byly vytvářeny v průběhu vývoje produktu, přičemž při vývoji byla snaha o dodržování techniky testy řízeného vývoje. Ovšem ne vždy se to povedlo, a tak některé testy vznikaly zpětně po vytvoření funkcionality uvnitř aplikace.

Klíčovými částmi se v rámci vyvíjené aplikace rozumí především vrstvy aplikační logiky a sekce DAL a agentů uvnitř datové vrstvy. Pro vrstvu aplikační logiky byly vytvořeny testy především pro funkce související s exekučními spisy, tedy např. pro logiku změny stavu spisů, vypočítání přehledu finančních náležitostí, přiřazení subjektů ke spisům, ale i pro schválení pošty, vytvoření úkolů atd. V sekci DAL byl kladen důraz na otestování repozitářů pro spisy a subjekty a na ověření správnosti generického repozitáře. Tato část obsahuje

spousty mapovacích funkcí pro mapování mezi databázovými a modelovými objekty, pro které nejsou vytvořeny žádné testy, jelikož pro nás nejsou tolik důležité. V rámci sekce agentů byly napsány testy pro zpracování a odeslání zpráv prostřednictvím datové schránky a také pro příchozí informace z lustračního rejstříku. Prezentační vrstva je testována pouze okrajově v místech, kde se objevuje funkcionalita přihlášení uživatele do systému a také jeho odhlášení. Jednotkové testy byly také vytvořeny pro funkce umožňující multitenanci, tedy zda přihlášený uživatel se skutečně připojí do příslušné databáze svého úřadu. Jelikož ve vrstvě modelu se vyskytují pouze používané objekty, není tato vrstva nikterak důležitá pro jednotkové testování. Pokrytí kódu automatizovanými testy je souhrnně znázorněno v tabulce 6.1.

Sekce	Nepokryto bloků	Nepokryto bloků (%)	Pokryto bloků	Pokryto bloků (%)
MVC	756	91,08 %	74	8,91 %
BLL	441	40,99 %	635	59,01 %
DAL	2514	68,93 %	1133	31,07 %
Agenti	33	9,07 %	331	90,93 %
Model	291	46,04 %	341	53,96 %
Celkem	4035	61,61 %	2514	38,39 %

Tabulka 6.1: Tabulka pokrytí kódu automatizovanými testy.

V tabulce pokrytí kódu se pod pojmem blok rozumí fragment kódu, který má vstupní a výstupní bod. Pokud tok programu prochází daným blokem, blok je označen jako pokrytý. Při tvorbě testů byla snaha o pokrytí kódu se složitější logikou nebo více větvemi. Nemělo tedy význam vytvářet testy pro metody, které pouze předávaly data z jiných komponent, případně metody, které pouze volaly knihovny .NET frameworku. U takových knihoven se již počítá s tím, že jsou otestovány jejich tvůrcem.

6.1.1 Jednotkové testy

Pro tvorbu testů byla vytvořena samostatná knihovna, která obsahuje automatizované testy oddělené podle vrstev, které testují. V sekci agentů jsou vytvořeny testy pro klienty připojující se k rejtříku ARES a pro klienta datových schránek. Pro vrstvu repozitářů jsou implementovány jednotkové testy pro repozitář souborů a subjektů. Aplikační logiku pokrývají testy pro logiku auditu, spisů, datových schránek, subjektů a úkolů. Pro prezentační vrstvu jsou udělány testy pro autentizační část a pro funkcionalitu přihlášení a odhlášení se ze systému. Jednotkové testy nejsou použity pouze v jednom případě automatizace, kterým je generický repozitář (ten je testován integračními testy).

Ke tvorbě testů byl zvolen nejrozšířenější framework pro tvorbu jednotkových testů pod platformou .NET NUnit[1], který obsahuje bohatou škálu možností a podpory pro provádění jednotkových testů. Testovacím třídám a testovacím metodám stačí přiřadit atribut, který rozpozná NUnit framework, a tím umožní metodám být prováděny jako testy.

Veškeré závislosti testovaných metod jsou řešeny pomocí mock objektů. K tomuto účelu byla využita knihovna Moq[2], která poskytuje podporu pro lambda výrazy a tím usnadňuje

[1]Dostupné z `http://www.nunit.org/`.

tvorbu testovacích dvojic. V současnosti patří mezi nejpoužívanější knihovny pro vytváření mock objektů na platformě .NET, pyšnící se velmi dobrou dokumentací.

Jelikož jednu metodu může ověřovat více testů, kde každý test testuje jiný stav a výstup metody, je snaha o dodržování konvence názvů testů. Pokud daný test neproběhne úspěšně, mělo by být hned jasné, jaký stav je testován. Proto název každého testu obsahuje název testované metody, stav nebo podmínku která se v této části očekává, a očekávané chování. Každý vytvořený test je strukturován pomocí tzv. AAA syntaxe. Základní myšlenkou je vytvoření všech závislostí, které vyžaduje testovaná komponenta (arrange), spuštění testované metody (act) a ověření, že podmínky testu byly splněny (assert). Pro lepší ilustraci je na obrázku 6.1 přiložen vzorový test.

```
[Test]
public void ScreenSubject_SubjectIsNotLegalPerson_ReturnsNull()
{
  int searchedIndex = 1;
  Subject returnedSubject = new Subject { Type =
     SubjectType.NaturalPerson };

  /* vytvoreni mock objektu */
  Mock<ISubjectsRepository> mockSubjectsRepository =
     new Mock<ISubjectsRepository>();

  /* definice chovani pro metodu GetSubject */
  mockSubjectsRepository.Setup(x => x.GetSubject(It.IsAny<int>()))
     .Returns(returnedSubject);

  /* vlozeni mock objektu do testovane tridy */
  SubjectsBL subjectsBL = new SubjectsBL(mockSubjectsRepository.Object);

  /* spusteni testovane metody */
  Subject result = subjectsBL.ScreenSubject(searchedIndex);

  /* kontrola vysledku */
  Assert.IsNull(result);
}
```

Obrázek 6.1: Ukázka jednotkového testu s použitím knihovny Moq.

Přiložená metoda dokumentuje test pro metodu lustrace subjektu uvnitř třídy *SubjectBL*. Tato metoda nejprve získá z repozitáře subjekt a pokud je daný subjekt právnická osoba, provede lustraci. V tomto případě však repozitář vrátí fyzickou osobu, nedojde tedy k žádné lustraci a návratovou hodnotou metody je *null*. Nejprve jsou v rámci testu nadefinovány potřebné proměnné. Poté dochází k vytvoření mock objektu pomocí knihovny Moq. V tomto objektu je nadefinováno chování pro metodu *GetSubject*. Vidíme, že pro vstup jakéhokoliv čísla vrátí nadefinovanou fyzickou osobu. Následně je vytvořena instance třídy *SubjectsBL*, které je v konstruktoru předán mock objekt. Nakonec dojde ke spuštění testované metody a kontrole výsledku.

Všechny vytvořené testy následují popsaný vzor a kontrolují výsledky testovaných metod pomocí tvrzení *Assert*. V případě, kdy testovaná metoda nemá žádnou návratovou

[2]Dostupné z `https://github.com/Moq/moq4/wiki/Quickstart`.

hodnotu, je v rámci testů kontrolováno, zda nadefinované metody mock objektu byly skutečně zavolány, případně s jakými parametry. Této kontroly je dosaženo pomocí knihovny Moq, která umí tuto událost ověřit.

6.1.2 Integrační testy

Pro ověření správnosti fungování generického repozitáře uvnitř vrstvy DAL byly vytvořeny integrační testy. Tento generický repozitář je užíván v dalších repozitářích, je tedy velmi důležité otestovat jeho správnou funkčnost. Vytvořené integrační testy jsou ve formě testů databáze v paměti (in-memory database tests). Tyto testy přináší možnost ověření, že vrstva komunikující s databází funguje opravdu správně. Oproti připojení ke klasické databázi mají především tu výhodu, že jsou rychlé a mohou proběhnout bez připojení k síti. Klasické testy proti skutečné databázi zahrnují také čas pro připojení a přenos po síti, a mohou tedy trvat déle.

Pro vytvoření testů databáze v paměti byla použita knihovna Effort[3]. Tato knihovna umožňuje vytvoření dočasné databáze v paměti pro aplikace, respektive pro klienta, který využívá knihovnu Entity Framework. Díky tomu je možné volat standardní metody knihovny Enitity Framework a volané metody se budou provádět v paměťové databázi. Není tedy zapotřebí měnit kód, pouze je knihovně předána třída *DbContext*, která se postará o připojení k paměťové databázi.

Vytvořené testy vypadají následovně. Nejprve je provedena metoda pro přípravu prostředí pro test (NUnit tuto metodu označuje jako setup), ve které je vytvořena dočasná databáze a připojení k této databázi je předáno do testovaného generického repozitáře. Uvnitř testovací metody poté dochází k naplnění databázových dat, spuštění testované metody repozitáře a následná kontrola, zda repozitář funguje správně. Pokud je tedy například testována metoda pro vložení záznamu do databáze, je po provedení metody repozitáře kontrolováno, zda se záznam skutečně v paměťové databázi nachází. Po provedení testu dochází ke spuštění metody, která po testu uklidí (NUnit tuto metodu označuje jako teardown). V této metodě dojde k uvolnění použité databáze. V rámci každého testu se spouští metoda pro přípravu a metoda pro úklid testu, což přináší výhodu v případě, kdy by bylo potřeba testy paralelizovat.

6.2 Manuální testování

Kromě automatizovaných testů byla práce testována také manuálně. V rámci manuálního testování šlo především o to, aby se systém odzkoušel jako celek. Podle specifikace případů užití se procházely jednotlivé scénáře a kontrolovalo se, zda se aplikace chová tak jak má. Oproti automatizovanému testování přináší manuální testování výhodu ověření chování systému z pohledu uživatele. To přichází vhod také v našem případě, kdy je cílem kromě dosažení jisté funkčnosti také přehledné uživatelské rozhraní a přívětivé chování vůči uživateli. Takové vlastnosti systému bylo možné ověřit pouze při provádění manuálních testů.

Ovšem manuální testování nebylo použito pouze pro otestování výsledného produktu. Souvisle během vývoje docházelo také k manuálnímu testování nově implementovaných částí, které sloužilo ke kontrole správnosti implementace. Manuální testování obvykle probíhalo přes grafické uživatelské rozhraní, kdy se pro patřičný vstup zobrazoval požadovaný výstup, případně přes kontrolu dat v databázi. Pokud bylo potřeba testovat jakoukoliv

[3]Dostupné z `http://effort.codeplex.com/`.

část aplikace, musela být propojena s grafickým rozhraním, aby ji bylo možné spouštět nebo zobrazovat její výsledky. Tento přístup je však vhodný v případě, kdy testování dané části neprobíhá příliš často. Pokud by se v budoucnu dále na tomto produktu pracovalo a bylo by nutné opakovaně kontrolovat funkčnost, stálo by za uvážení vytvoření dalších automatizovaných testů.

6.2.1 Akceptační testy

Závěrečnou etapou testování vytvořené aplikace bylo akceptační testování. V této fázi byl již vývoj aplikace ukončen a čekal na schválení zadavatelem. Přehled a výsledky prováděných akceptačních testů shrnují následující tabulky, které jsou rozděleny na části týkající se přihlášení do systému, práce se spisy, subjekty, datovou schránkou a úkoly.

Testovaný scénář	**Očekávaný výsledek**	**Test**
Přihlášení do systému s neplatným uživatelským účtem	Uživatel není přihlášen do aplikace	OK
Přihlášení do systému s platným uživatelským účtem	Uživatel je přihlášen do aplikace	OK
Přihlášení do systému s rolí zaměstnance	Uživateli nejsou umožněny funkce a okna pro role exekutora a administrátora	OK
Přihlášení do systému s rolí exekutora	Uživateli jsou umožněny funkce pro roli exekutora a zároveň odepřeny funkce administrátora	OK
Přihlášení do systému s rolí administrátora	Uživateli jsou umožněny funkce pro roli administrátora a zároveň odepřeny funkce pro roli exekutora	OK
Přihlášení do systému uživatele, který náleží danému úřadu (test multitenance)	Uživateli jsou zobrazeny data z databáze daného úřadu	OK
Změna hesla uživatele	Uživateli je změněno přihlašovací heslo	OK
Odhlášení se	Uživatel je odhlášen z aplikace	OK

Tabulka 6.2: Tabulka akceptačních testů pro přihlášení a role systému.

Tabulka akceptačních testů pro přihlášení a role systému 6.2 testuje funkčnost přihlášení a odhlášení uživatele ze systému. S tím souvisí také role přihlášeného uživatele, kterému jsou dostupné požadované funkce systému. Test s označením test multitenance ověřuje správnou funkcionalitu multitenance, tedy zda uživatel náležící k úřadu uvidí po přihlášení data z databáze daného úřadu.

Další tabulka 6.3 obsahuje testy operací související se spisem. Kromě posledních dvou funkcí jsou všechny tyto funkce dostupné pro každého uživatele. Poslední dva testy z tabulky lze provést pouze pod rolí exekutora. V tabulce nejsou uvedeny scénáře přepnutí stavu na základě vypršení časového intervalu, které obstarává plánovaná úloha, jelikož tento scénář nenáleží uživateli systému.

Testovaný scénář	**Očekávaný výsledek**	**Test**
Vytvoření spisu	Zadaný spis je vytvořen	OK
Upravení spisu	Zadaný spis je upraven	OK
Vyhledání spisu	Spis se zadaným klíčem je vyhledán	OK
Filtrování spisu	Spisy se zadaným klíčem jsou vyfiltrovány a zobrazeny	OK
Přiřazení subjektu ke spisu	Ke spisu je přiřazen zadaný subjekt	OK
Přiřazení dokumentu z PC	Ke spisu je přiřazen vybraný dokument	OK
Přiřazení dokumentu z datové schránky	Ke spisu je přiřazen vybraný dokument z datové schránky	OK
Vytvoření finanční náležitosti	Vytvořená finanční náležitost je přiřazena ke spisu a finanční přehled je přepočítán.	OK
Smazání finanční náležitosti	Smazaná finanční náležitost nefiguruje mezi finančními náležitostmi spisu, finanční přehled je přepočítán.	OK
Vytvoření poštovního úkonu	Zadaný poštovní úkon je přiřazen ke spisu	OK
Změna stavu spisu	Spis je v novém stavu	OK
Zastavení exekuce (role exekutora)	Exekuční spis je ve stavu zastaveno	OK
Nastavení termínu konce odkladu exekuce (role exekutora)	Termín je přiřazen ke spisu a zobrazen v detailu spisu	OK

Tabulka 6.3: Tabulka akceptačních testů pro spisy.

Tabulka 6.4 zobrazuje provedené testy pro subjekty. Zde pouze dodám, že lustrace v rejstříku ARES je možná pouze pro právnické osoby.

Seznam testů v tabulce 6.5 shrnuje funkcionalitu systému týkající se datových schránek. Pokud uživatel systému, který není v roli exekutora, odesílá poštu soudu, prochází vytvořená zpráva kontrolou exekutora. Daná zpráva není tedy odeslána, ale je vytvořen úkol pro exekutora. Exekutor poté zobrazí zprávu přiloženou k vytvořenému úkolu a pokud ji schválí, je zpráva skutečně odeslána.

Posledními tabulkami akceptačních testů jsou tabulka testů pro úkoly 6.6 a tabulka testů pro administrátorskou sekci 6.7. Administrátorská sekce je dostupná pouze pro roli administrátora a jelikož veškeré zmíněné testy lze provést pouze v této roli, není nutné vypsat do tabulky údaj o administrátorské roli uživatele.

Při vytváření a editaci spisů, subjektů, finančních náležitostí a obecně údajů v databází, dochází k vytvoření auditních záznamů. Tvorba auditních záznamů nebyla v tabulkách zmíněna, jelikož uživatel systému nemá informace o těchto záznamech (pokud není administrátor) a ani možnost, jak vytvořená záznam do auditu ověřit.

Testovaný scénář	**Očekávaný výsledek**	**Test**
Vytvoření subjektu	Zadaná data jsou vložena do databáze.	OK
Upravení subjektu	Zadaná data jsou upravena v databázi	OK
Vyhledání subjektu	Subjekt se zadaným klíčem je vyhledán	OK
Filtrování subjektů	Subjekty se zadaným klíčem jsou vyfiltrovány a zobrazeny	OK
Zobrazení spisů, ve kterých subjekt figuruje	Požadované spisy jsou zobrazeny	OK
Lustrace pro právnickou osobu v rejstříku ARES	Provedena lustrace a výsledky jsou zobrazeny	OK

Tabulka 6.4: Tabulka akceptačních testů pro subjekty.

Testovaný scénář	**Očekávaný výsledek**	**Test**
Odeslání nové zprávy	Nová zpráva je odeslána.	OK
Vytvoření nové zprávy pro soud v roli zaměstnance	Zpráva není odeslána, je vytvořen úkol pro uživatele s rolí exekutora.	OK
Vytvoření nové zprávy pro soud v roli exekutora	Zpráva je odeslána.	OK
Schválení odeslání pošty (role exekutor)	Zpráva je odeslána	OK
Zobrazení doručené pošty	Doručená pošta je zobrazena	OK
Zobrazení odeslané pošty	Odeslaná pošta je zobrazena	OK
Filtrování pošty	Zprávy se zadaným klíčem jsou vyfiltrovány a zobrazeny	OK
Přiřazení příchozí pošty ke spisu	Zpráva je přiřazena ke spisu a figuruje v evidenci součinnosti.	OK
Přiřazení dokumentu z příchozí pošty ke spisu	Dokument je přiřazen ke spisu.	OK
Označení zprávy jako zpracované	Zpráva je zobrazena mezi zpracovanými zprávami.	OK

Tabulka 6.5: Tabulka akceptačních testů pro poštu datové schránky.

Testovaný scénář	Očekávaný výsledek	Test
Vytvoření nového úkolu	Nový úkol je vytvořen	OK
Zobrazení úkolů, které jsou přiřazeny přihlášenému uživateli	Úkoly přiřazené přihlášenému uživateli jsou zobrazeny	OK
Filtrování úkolů	Úkoly se zadaným klíčem jsou vyfiltrovány.	OK
Označení úkolu jako hotový	Úkol je zobrazen jako hotový	OK
Označení úkolu jako odmítnutý	Úkol je zobrazen jako odmítnutý	OK

Tabulka 6.6: Tabulka akceptačních testů pro úkoly.

Testovaný scénář	Očekávaný výsledek	Test
Vytvoření nového uživatele systému	Nový uživatel je vytvořen	OK
Upravení údajů uživatele systému	Nové údaje jsou upraveny a zobrazeny	OK
Změnit roli uživatele systému	Uživateli je přiřazena nová role	OK
Zobrazení auditu	Informace o auditu jsou zob razeny	OK
Filtrování záznamů auditu	Záznamy auditu se zadaným klíčem jsou vyfiltrovány a zobrazeny	OK

Tabulka 6.7: Tabulka akceptačních testů pro administrátorské akce.

6.3 Použité testovací prostředí

Testování aplikace probíhalo pod webovým serverem IIS Express, který je určen pro vývoj a testování webových aplikací na platformě .NET. Tento server je dodáván společně s vývojovým prostředím Visual Studio. Použitý server je navržen tak, aby emuloval produkční server IIS (Internet Information Services), který bude hostit aplikaci v provozu. Pro skladování dat byl použit MS SQL Server, což je systém pro správu relační databáze. Veškeré použité databáze, tedy hlavní (master) databáze i uživatelské (tenant) databáze byly umístěny na tomto serveru.

Popsaný webový i databázový server se při testování nacházely na lokálním stroji, nezanáší tedy do testování časovou prodlevu způsobenou síťovou komunikací. Primárním testovacím nástrojem pro klientskou část byl použit prohlížeč Mozilla Firefox, sekundárním prohlížeč Google Chrome. Detaily použitého testovacího prostředí shrnuje následující tabulka.

Komponenta	**Model**
Webový server	IIS Express 10.0 (10.0.10557.1000)
Databázový server	MS SQL Server 2014 (2014.120.2000.8)
Primární webový prohlížeč	Mozilla Firefox 46.0.1
Sekundární webový prohlížeč	Google Chrome 50.0.2661.102 m
Procesor testovacího stroje	Intel Core i7-5500U
RAM paměť testovacího stroje	8GB DDR3
Operační systém testovacího stroje	Windows 10 Home, 64bit, verze 1511

Tabulka 6.8: Tabulka detailů testovacího prostředí.

Kapitola 7

Závěr

V této práci byl vytvořen prototyp informačního systému, který je určen pro menší exekutorské úřady. Cílem vytvořeného prototypu bylo sjednotit nejnutnější potřeby exekutorského úřadu do jednoho systému, který by ulehčil a částečně zautomatizoval práci jeho uživatelů a poskytl jednoduché a přehledné uživatelské rozhraní. Vytvořeným prototypem je webová aplikace, která umožňuje provádět správu exekučních spisů, jejich subjektů, příslušejících dokumentů, finančních náležitostí a evidovat součinnost. Součástí agendy spisu je také implementace jeho workflow. Mimoto poskytuje aplikace funkcionalitu komunikace s datovými schránkami, provádění lustrace právnických osob v administrativním registru ekonomických subjektů a zaznamenávání událostí do auditu. Navíc implementuje možnost multitenance, díky které je aplikace schopna poskytovat služby více úřadům. Pro implementovanou aplikaci byly vytvořeny automatizované jednotkové testy, které pomáhaly při fázi vývoje i během zpětného ověřování funkčnosti. Tyto testy pokrývají 38,39% zdrojového kódu, přičemž pokrytými částmi jsou především části aplikační logiky. Správnost dosažené implementace byla také ověřena pomocí akceptačních testů.

Ve srovnání s produkty na trhu obsahuje vytvořený prototyp pouze nejnutnější části pro agendu úřadu soudního exekutora. To může být výhoda právě pro malé exekutorské úřady, které nejsou ochotny zaplatit za komplexní systémy, jejichž veškerou funkcionalitu nevyužijí. Další výhodou vytvořené aplikace je jeho jednoduchost, která oproti komplexním systémům vede k příjemnému a jednoduchému uživatelskému rozhraní. Na druhou stranu vytvořený systém postrádá možnost automatické součinnosti s lustračními rejstříky, které jsou pro veřejnost zpoplatněny nebo vůbec nepřístupné.

Implementovaný prototyp byl předán zadavateli a je pouze na něm, zda bude projekt pokračovat či nikoliv. Pro budoucí práci na aplikaci by určitě stála za zvážení automatická součinnost s lustračními rejstříky, které exekutor denně využívá. Díky jejich nepřístupnosti pro veřejnost by však takové rozšíření nebylo pouze o implementaci, ale také o administrativním povolení jednotlivých institucí, které rejstříky spravují. Zajímavým rozšířením by byla také automatická tvorba dokumentů, která by na základě šablon úředních dokumentů vytvořila dokumenty nové. Tato činnost je také velmi častá u zaměstnanců exekutorského úřadu, byla by tedy určitě využita. Z pohledu automatizovaného testování by stálo za úvahu vytvoření automatizovaných systémových testů, které by ulehčily a urychlily opakované testování v případě, pokud by měla aplikace delší životnost a více verzí. Návrhů na rozšíření aplikace může být celá řada a záleží pouze na výsledku, kterého chceme dosáhnout.

Literatura

[1] AngularJS community: AngularJS - Superheroic JavaScript MVW Framework [online]. https://angularjs.org/, 2015 [cit. 2015-11-28].

[2] Bootstrap community: Bootstrap: The world's most popular mobile-first and responsive front-end framework [online]. http://getbootstrap.com/, 2015 [cit. 2015-11-29].

[3] Fowler, M.: Mocks aren't stubs [online]. http://martinfowler.com/articles/mocksArentStubs.html, 2007 [cit. 2016-05-14].

[4] Gajdušek, R.: Úvod do platformy .NET, slajdy do předmětu IW5 [online]. http://www.fit.vutbr.cz/study/courses/IW1/public/info/doku.php?id=iw5, 2015 [cit. 2015-11-16].

[5] Hruška, T.; Křivka, Z.: Studijní opora do předmětu IIS a PIS, Pojem informačního systému, Data, Procesy, Transakce [online]. https://www.fit.vutbr.cz/study/courses/WAP/private/opory/.cs, 2012 [cit. 2015-11-15].

[6] Microsoft: Multi-Tenant Data Architecture [online]. https://msdn.microsoft.com/en-us/library/aa479086.aspx#mlttntda_sdss, 2006 [cit. 2015-12-19].

[7] Microsoft: ASP.NET | The ASP.NET Site [online]. http://www.asp.net/, 2015 [cit. 2015-11-16].

[8] Microsoft: ASP.NET MVC Overview [online]. https://msdn.microsoft.com/en-us/library/dd381412%28v=vs.108%29.aspx, 2015 [cit. 2015-11-17].

[9] Microsoft: Izolace testovaného kódu pomocí zástupného rozhraní Microsoft [online]. http://martinfowler.com/articles/mocksArentStubs.html, 2016 [cit. 2016-05-14].

[10] Mitchell, S.: Security basics and ASP.NET Support. http://www.asp.net/web-forms/overview/older-versions-security/introduction/security-basics-and-asp-net-support-cs, 2008 [cit. 2016-05-18].

[11] Najvarek, J.; Radkovič, P.; Černocký, P.: Projektová dokumentace exekutor. 2009.

[12] Pearson, L.: The four levels of software testing [online]. http://www.seguetech.com/blog/2013/07/31/four-levels-software-testing, 2015 [cit. 2016-05-12].

[13] Slavata, M.: Kdo je soudní exekutor [online]. http://www.exekutormlboleslav.cz/kdo-je-soudni-exekutor.html, 2015 [cit. 2015-12-28].

[14] Wikipedia, the free encyclopedia: Information system [online]. https://en.wikipedia.org/wiki/Information_system, 2015 [cit. 2015-11-15].

[15] Wikipedia, the free encyclopedia: Test-driven development [online]. https://en.wikipedia.org/wiki/Test-driven_development, 2015 [cit. 2015-12-20].

[16] WWW stránky: ASP.NET - Single Page Applications: Build Modern, Responsive Web Apps with ASP.NET [online]. https://msdn.microsoft.com/en-us/magazine/dn463786.aspx, 2013 [cit. 2015-11-17].

[17] WWW stránky: Fun with AngularJS [online]. http://devgirl.org/2013/03/21/fun-with-angularjs/, 2013 [cit. 2015-11-28].

[18] WWW stránky: Co jsou Datové schránky [online]. https://www.datoveschranky.info/o-datovych-schrankach/co-jsou-datove-schranky, 2015 [cit. 2015-12-28].

Přílohy

Seznam příloh

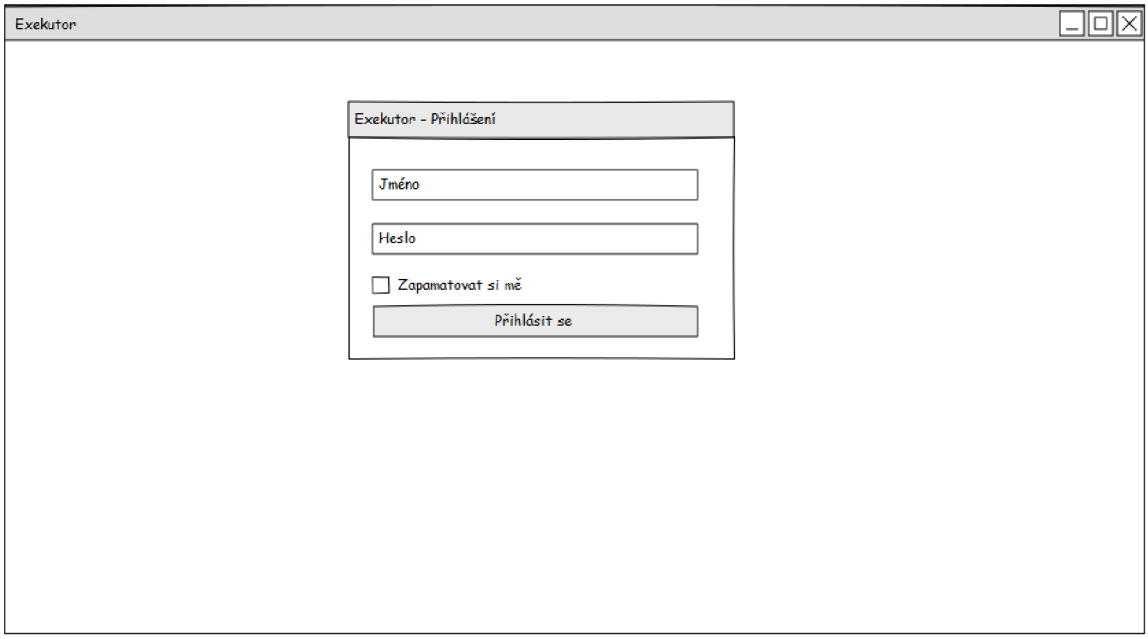
Příloha A

Obsah DVD

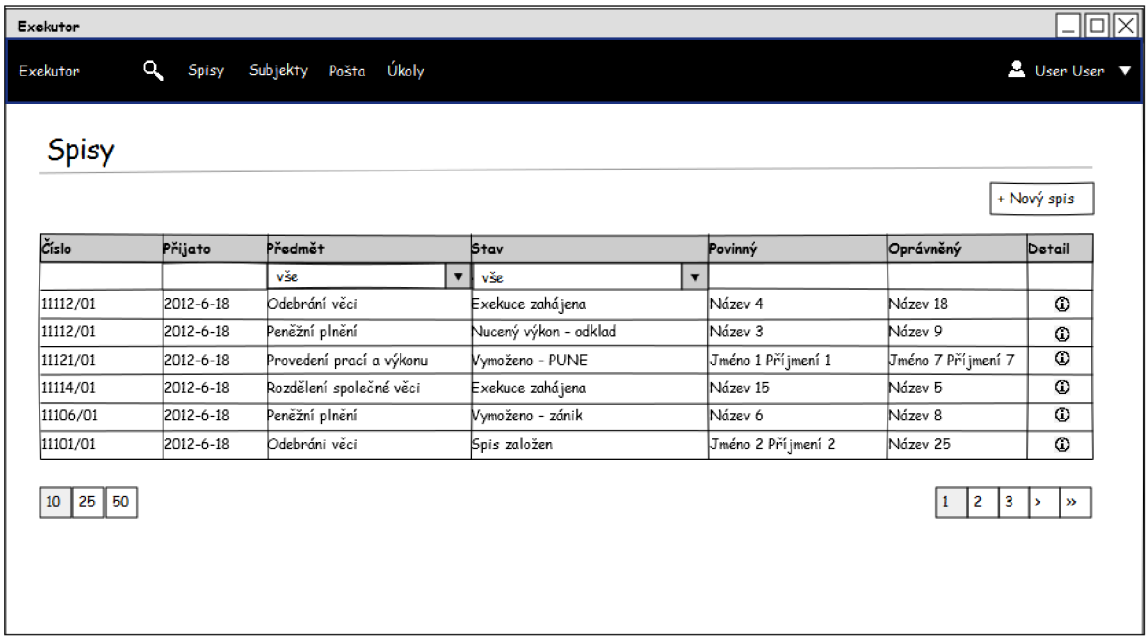
- **src** - zdrojové soubory aplikace
- **bin** - binární soubory aplikace
- **screens** - snímky obrazovek aplikace
- **wireframes** - drátový model obrazovek
- **sql** - databázové skripty
- **latex** - zdrojové soubory dokumentace
- **doc** - výsledná dokumentace

Příloha B

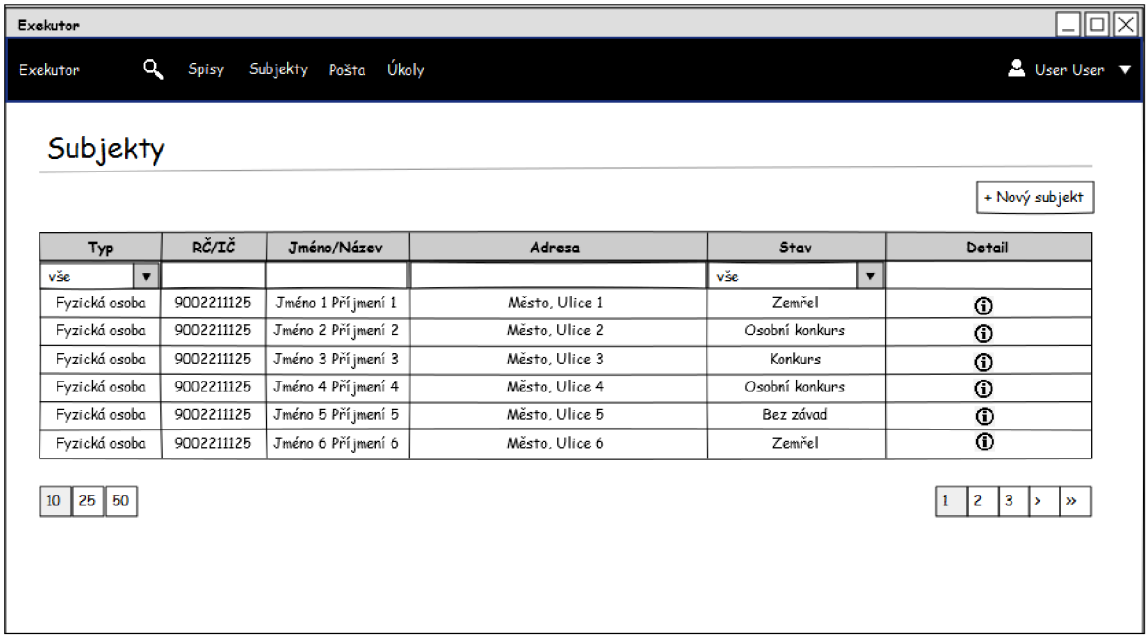
Drátěný model obrazovek



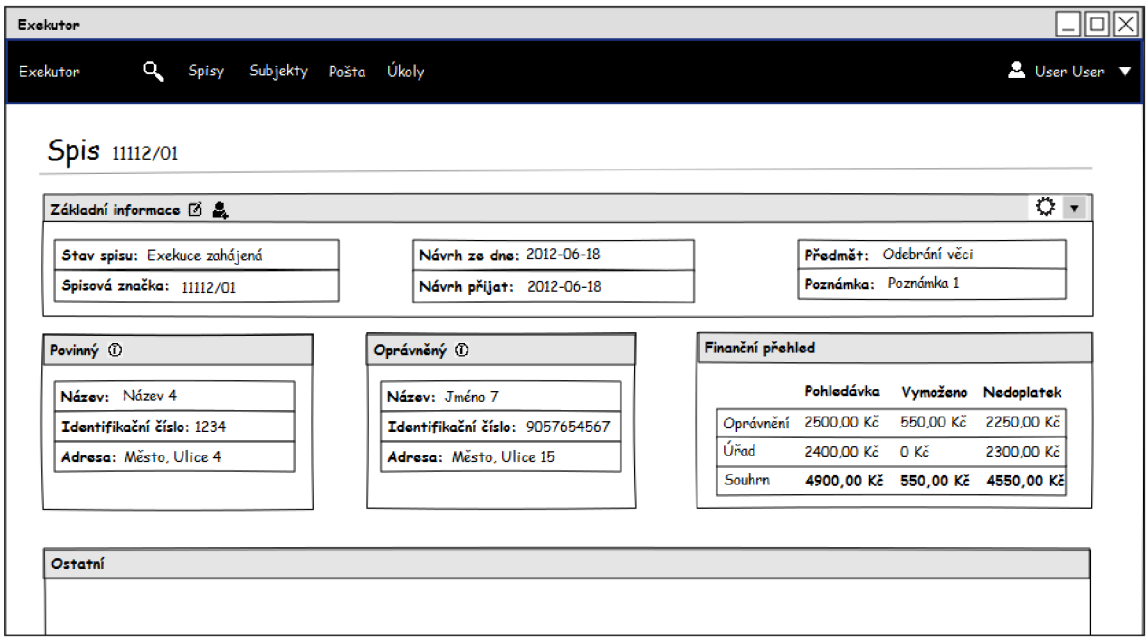
Obrázek B.1: Model obrazovky pro přihlášení.



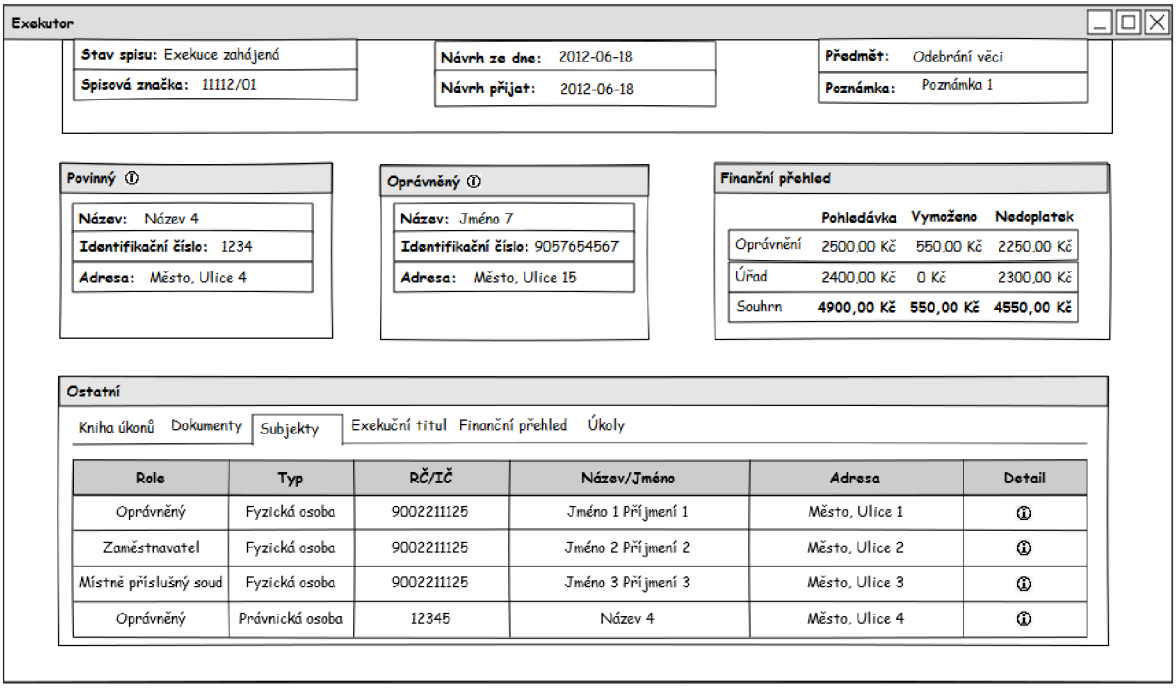
Obrázek B.2: Model obrazovky pro přehled spisů.



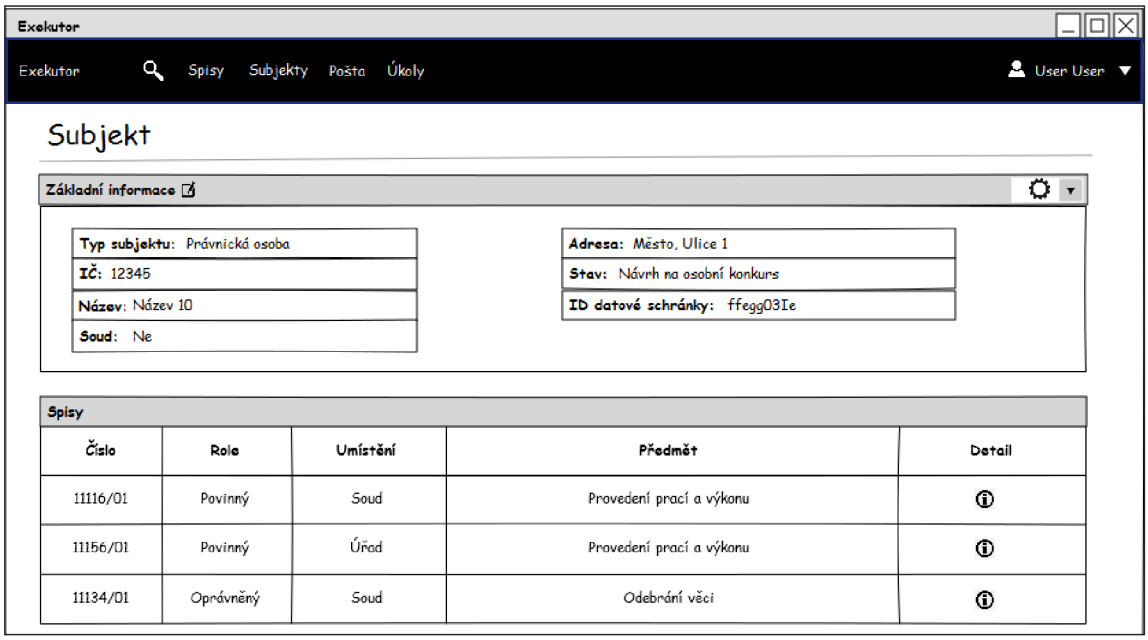
Obrázek B.3: Model obrazovky pro přehled subjektů.



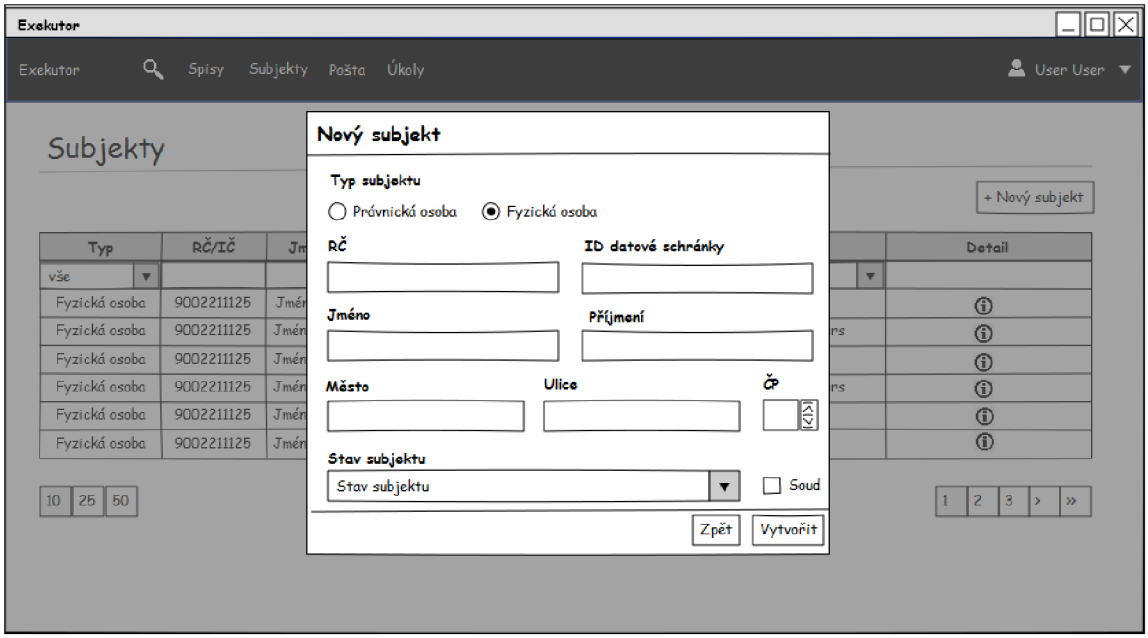
Obrázek B.4: Model vrchní části obrazovky pro detail spisu.



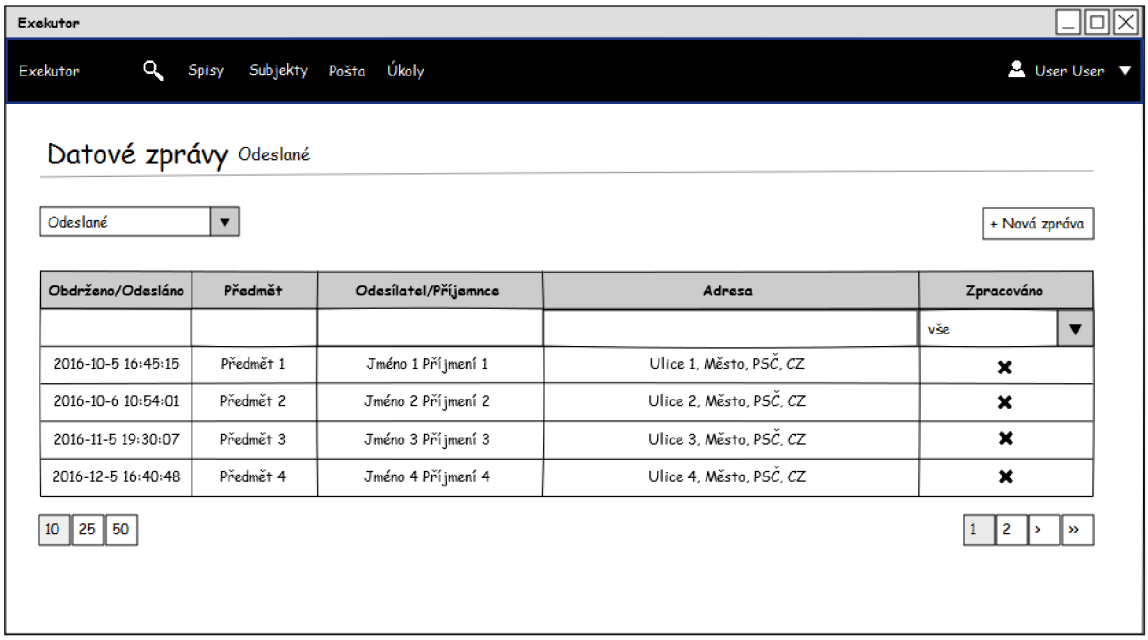
Obrázek B.5: Model spodní části obrazovky pro detail spisu.



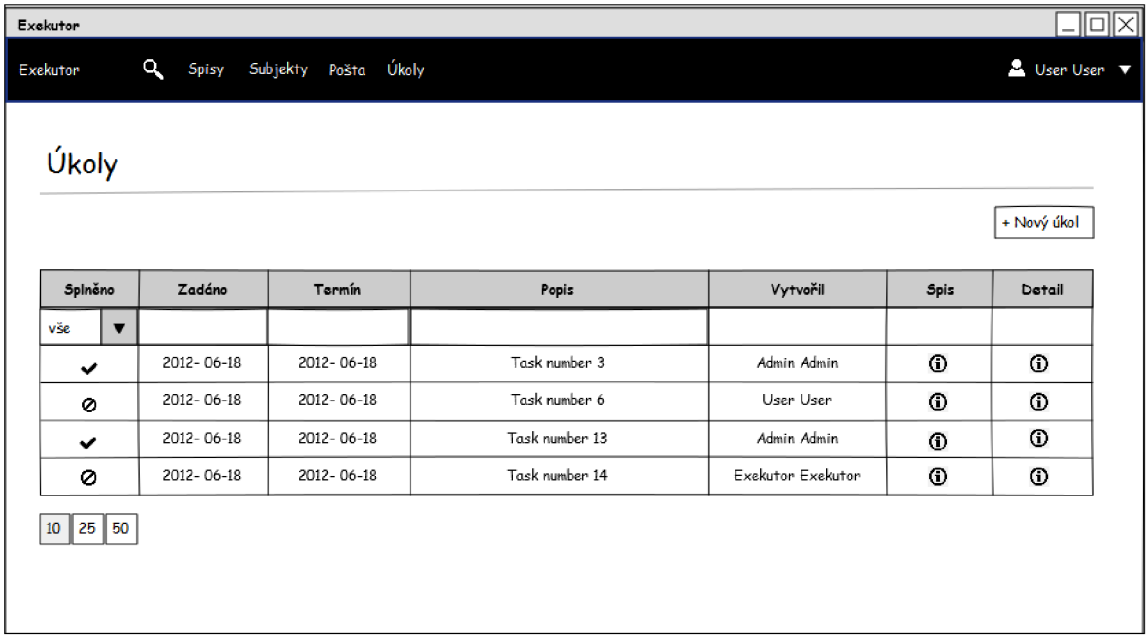
Obrázek B.6: Model obrazovky pro detail subjektu.



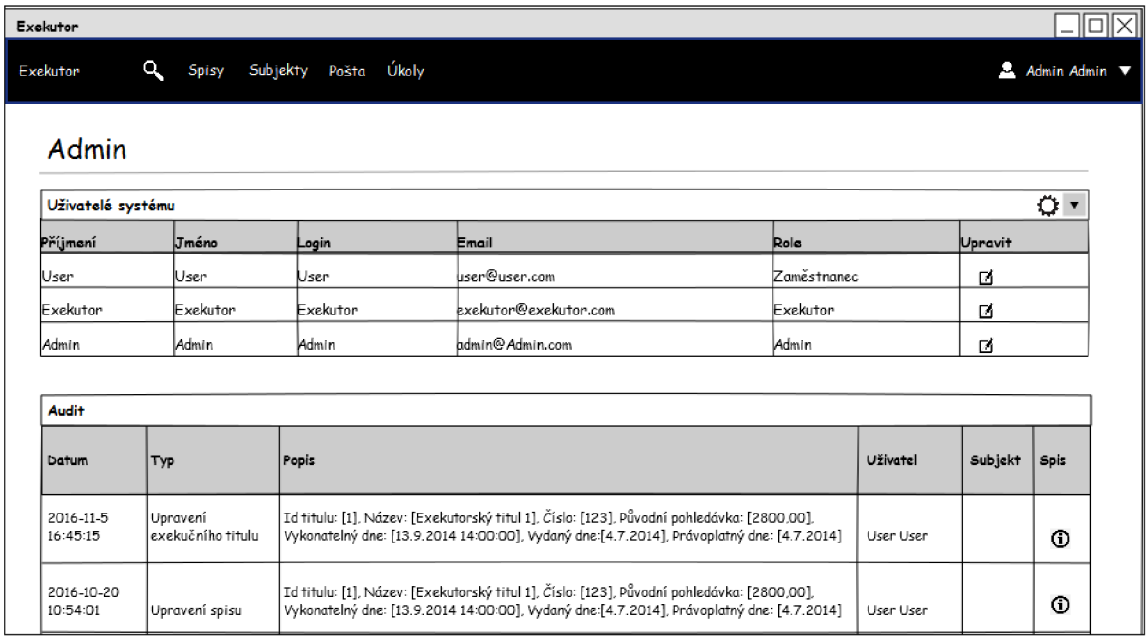
Obrázek B.7: Model modálního okna pro vytvoření subjektu.



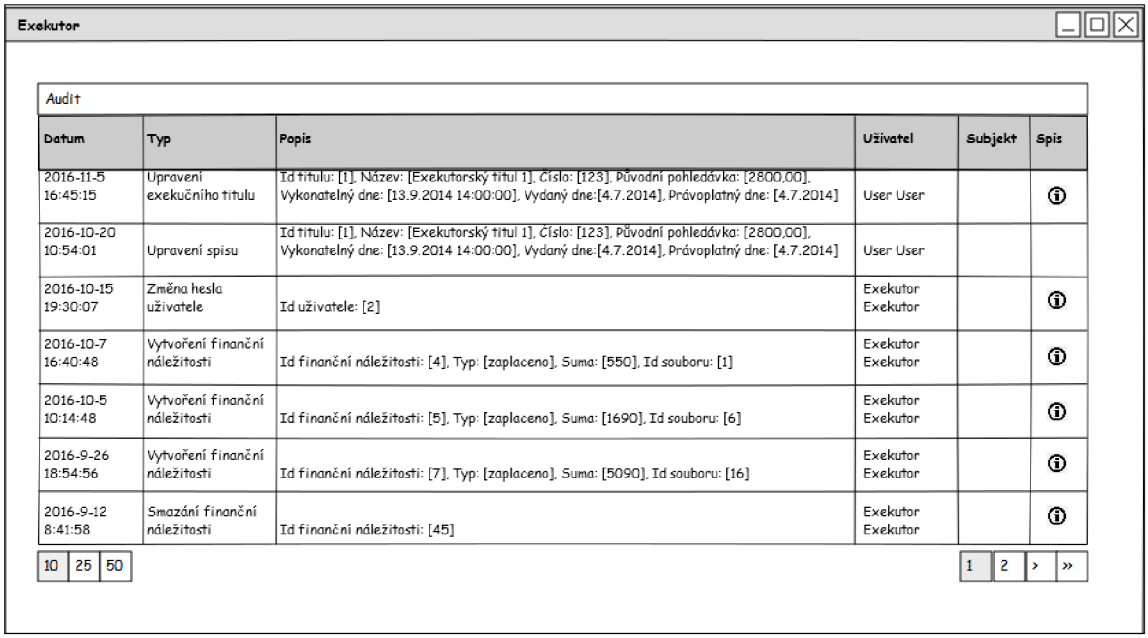
Obrázek B.8: Model obrazovky pro poštu.



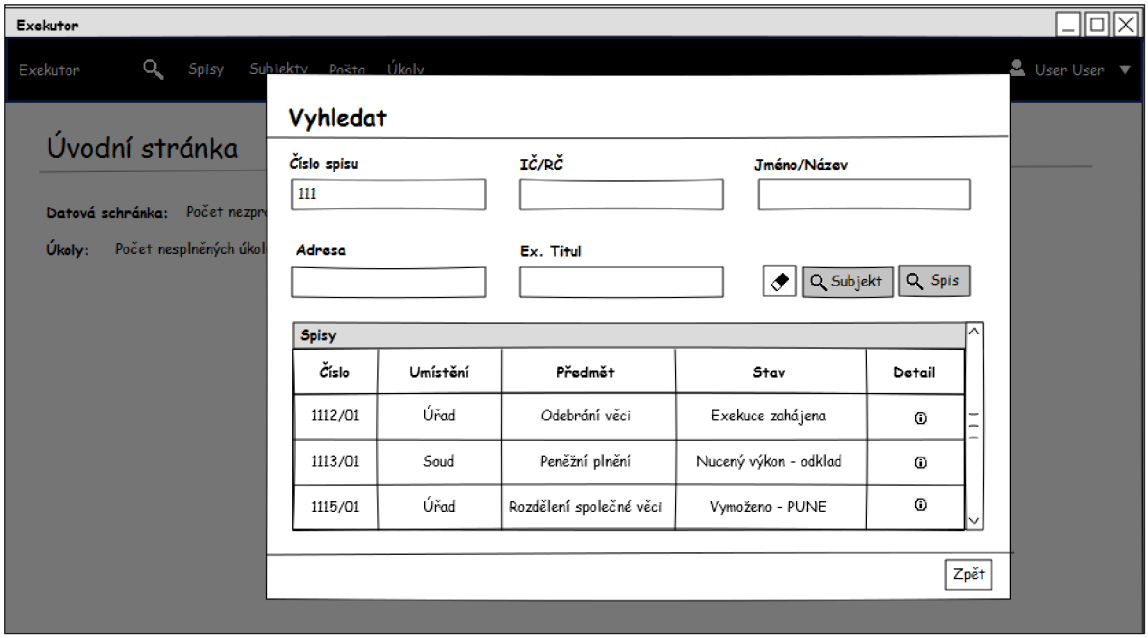
Obrázek B.9: Model obrazovky pro úkoly.



Obrázek B.10: Model vrchní části obrazovky pro administrátora.



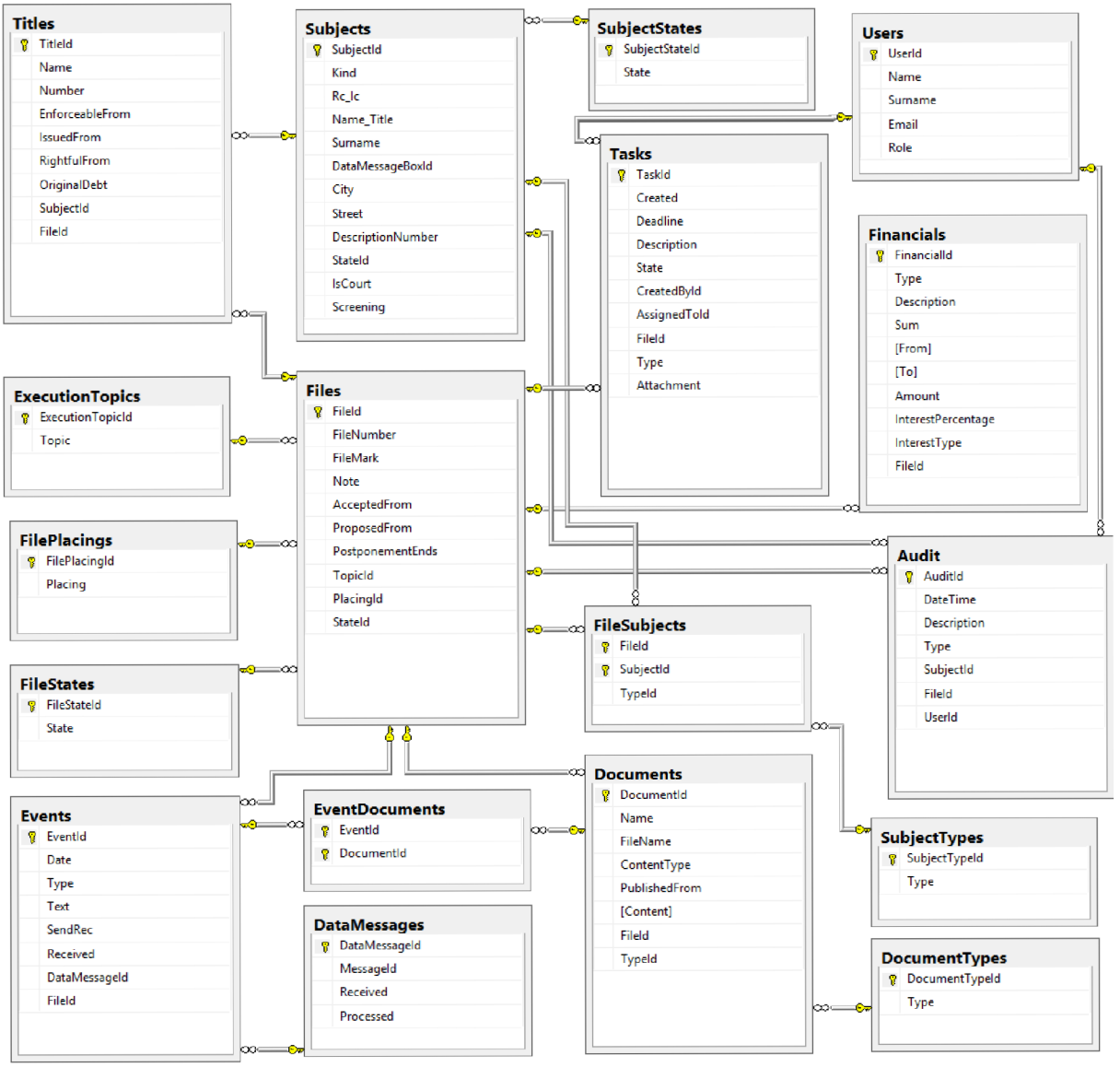
Obrázek B.11: Model spodní části obrazovky pro administrátora.



Obrázek B.12: Model modálního okna pro vyhledávání.

Příloha C

Kompletní ER diagram úřadu



Obrázek C.1: ER diagram databázového schématu úřadu.