Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Informační systémy využívající klient-server architektury

Tomáš Markacz

Vedoucí práce: Ing. Vojtěch Jirkovský

11. května 2015

Poděkování

V prvé řadě bych chtěl poděkovat svým rodičům, za podporu při studiu, bez kterých bych v Praze nemohl začít studovat. Dále bych chtěl poděkovat své přítelkyni za podporu při psaní této práce. Samozřejmě bych chtěl také poděkovat panu Ing. Vojtěchu Jirkovskému za cenné podněty a zpětnou vazbu při vedení mé bakalářské práce. V neposlední řadě bych také rád poděkoval svým kolegům a nadřízeným za poskytnutí pracovní flexibility v průběhu dokončování této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 11. května 2015 .....

Odkaz na tuto práci

Markacz, Tomáš. *Informační systémy využívající klient-server architektury.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Cílem této bakalářské práce je implementace webového informačního systému, který je založen na klient-server architektuře. V této práci jsou popsány nejen výhody tohoto řešení, ale také úskalí, která se v průběhu implementace naskytla. Informační systém byl vytvořen na základě provedené analýzy a návrhu. Následně byla provedena implementace a testování řešení. Výsledkem je funkční informační systém pro advokátní kancelář podporující zpracování objednávek právnických služeb. V závěru práce je provedeno srovnání klíčových procesů bez a s podporou tohoto informačního systému.

Klíčová slova informační systém, klient-server architektura, analýza, návrh, implementace, REST API, Nette, Doctrine, AngularJS, Behat

Abstract

The objective of this bachelor thesis is an implementaion of a web-based information system using the client-server architecture. In this work, there are described not only advantages of this solution, but also pitfalls encountered while performing the implementation. The information system was created based on analysis and design, followed by implementation and testing. The result is a functional information system for a law office that supports processing of law services orders. A comparison of the key processes with and without the support of this information system was done in the very end of this work.

Keywords information system, client-server architecture, analysis, design, implementation, REST API, Nette, Doctrine, AngularJS, Behat

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V současné době již většina firem využívá nějakou formou informační systémy. Tyto systémy zvyšují produktivitu, omezují chybovost a pomáhají se složitými rozhodnutími. Jejich teorií se zabývá mnoho oborů, institucí i firem, proto bych se ve své bakalářské práci chtěl zaměřit na jejich praktickou a implementační stránku.

V posledních několika letech se stále více budují dekomponované aplikace sestavené z více částí. Jedním z možných přístupů k dekompozici je architektura klient-server. Důvodem, proč píši bakalářskou práci na toto téma, je vytvoření uceleného pohledu na problematiku týkající se zmiňované architektury ve spojení s informačními systémy.

Cílem této práce je seznámit se s problematikou informačních systémů a klient-server architektury. Pro podpoření teorie praxí provádím analýzu uživatelských požadavků a procesů v modelové firmě. Jedná se o fiktivní firmu, jejíž požadavky však vycházejí z reálných požadavků na podobný informační systém, který jsem v minulosti pomáhal implementovat. Na základě této analýzy vypracovávám návrh vlastního řešení. Návrh implementuji jako webovou aplikaci za využití technologií popsaných v kapitole 3 a řádně otestuji. V závěru práce provádím ekonomicko-manažerské zhodnocení přínosů navrženého informačního systému.

Kapitola 1

Teorie

V této kapitole provádím rešerši týkající se klient-server architektury, druhů webových aplikací a možných způsobů komunikace mezi klientem a serverem.

1.1 Klient server architektura

Klient-server architektura umožňuje oddělit klienta, kterým je často aplikace s grafickým uživatelským rozhraním (GUI), a server, který obsahuje business logiku systému a zajišťuje uložení dat. Klient a server spolu komunikují pomocí počítačové sítě.

Mezi charakteristické vlastnosti klienta patří [1]:

- Je aktivní.
- Posílá žádosti serveru.
- Čeká na odpovědi, které obdrží od serveru.
- Obvykle se připojuje k jednomu serveru.
- Komunikuje přímo s koncovým uživatelem pomocí GUI.

Naproti tomu mezi charakteristiky serveru patří:

- Je pasivní.
- Naslouchá na síti a zachycuje požadavky připojených klientů.
- Přijaté požadavky zpracovává a odpovídá na ně.
- Obvykle se stará o řízení přístupu k datům a jejich přenos ke klientovi.
- Může zajišťovat nepřímou komunikaci dvou a více klientů.

Pokud se budeme dále zabývat touto architekturou v kontextu informačních systémů, můžeme definovat další vlastnosti klienta a serveru [2].

Úkolem klienta je přijímat požadavky a data ve formě vstupů od uživatele, zvalidovat je a delegovat jejich zpracování na serverovou část aplikace. Data přijatá od serveru vhodně prezentuje uživateli pomocí grafického rozhraní. Server přijaté požadavky a data zpracovává a ukládá do datového úložiště. Serverová část v tomto případě obsahuje klíčovou business logiku a řízení procesů, které mají být informačním systémem podporovány.

Klienti se při komunikaci se serverem autentizují. Autentizace je proces ověření identity klienta či uživatele. Důležitou vlastností serveru je možnost řídit přístup k datům, k čemuž využívá autorizace (udělení souhlasu s provedením nějaké operace nad daty) [3]. Postavení serveru jako centrálního bodu pro všechny klienty umožňuje zaručit důvěryhodnost dat – známe jejich původ a víme, že jsou pod kontrolou serveru. Tato vlastnost je pro informační systémy velmi důležitá.

1.1.1 Role klienta a serveru

Protože klient je typicky navržen pro interakci s koncovým uživatelem, je na ní jeho funkcionalita a implementace zaměřena [1]. Hlavní funkcí, kterou klient zajišťuje, je prezentace dat a interakce s uživatelem.

Klient také může obsahovat business a validační logiku, takového klienta nazýváme tlustým klientem [4]. Aplikace tlustého klienta je více samostatná a pro svou činnost potřebuje méně kooperovat se serverem, protože podstatná část business logiky je umístěna právě v ní. I při využití tlustého klienta však nelze business a validační logiku ze serveru zcela odstranit. Server nemůže důvěřovat datům, která od klienta přijímá. Musí je opět validovat a ověřit si jejich správnost v aktuálním kontextu business logiky. V praxi se tak nedá zabránit duplicitám v kódu klientské a serverové části aplikace.

Pokud však klient žádnou logiku neobsahuje, je validace dat a vyhodnocení business logiky delegováno čistě na server. Klient tak v odpovědích přijatých od serveru pouze kontroluje správnost svých požadavků a v případě jejich nesprávnosti se o korekci může pokusit sám nebo o to požádá uživatele systému. Takového klienta nazýváme tenkým klientem.

V praxi se často tyto přístupy vhodně kombinují s cílem minimalizovat složitost aplikace.

1.1.2 Klient

Možné podoby aplikace klienta informačního systému mohou být:

- desktopová aplikace,
- mobilní aplikace a

- webová aplikace.

Desktopová aplikace je druh aplikace, která běží na stolním počítači nebo notebooku [5]. Ke svému běhu potřebuje operační systém a systémové knihovny, případně další software.

Mobilní aplikace je naproti desktopové navržena pro běh na chytrých telefonech, tabletech a dalších mobilních zařízeních [6]. Mezi hlavní rozdíly oproti desktopovým aplikacím patří nižší hardwarová náročnost na zařízení a přizpůsobení uživatelského rozhraní pro dotykové ovládání a omezenou velikost obrazovky.

Webová aplikace ke svému běhu vyžaduje webový prohlížeč, z tohoto důvodu musí být napsána v programovacím jazyce, který bude prohlížeč schopný interpretovat. Grafické uživatelské rozhraní je řízeno a vykreslováno prohlížečem [7]. Webové aplikace jsou populární díky širokému rozšíření webových prohlížečů na všech typech zařízení a platformách.

Výhodou webové aplikace je její dostupnost, menší náklady na vývoj a možnost jejího běhu jak na počítačích, tak i na přenosných zařízeních. Aplikace použitelná na různých velikostech obrazovky musí své grafické rozhraní přizpůsobit cílovému zařízení. Přizpůsobení lze docílit pomocí responzivního designu nebo adaptivního designu [8].

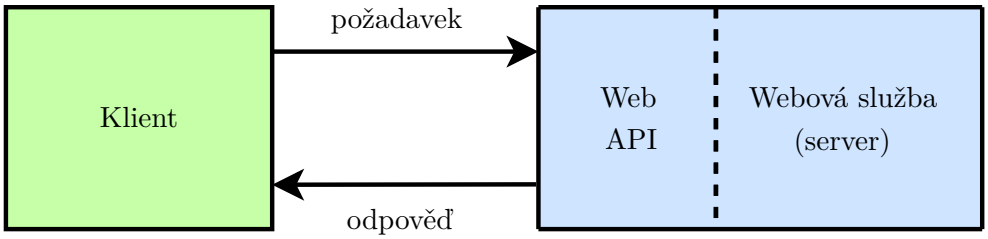
Při výběru vhodného druhu aplikace je dobrou volbou webová aplikace, pokud není potřeba žádného z následujících [9]:

- Aplikace potřebuje vysoký procesorový a grafický výkon.
- Aplikace vyžaduje přístup k hardwaru (kamera, mikrofon).
- Aplikace potřebuje přístup k uživatelským datům (adresář kontaktů, knihovna médií).
- Aplikace potřebuje přístup distribučním službám a on-line platbám (Google Play, App Store, Windows Store).
- Aplikace potřebuje vytvářet push notifikace.
- Aplikace potřebuje být spuštěna na pozadí operačního systému jako systémová služba.

V mé bakalářské práci se dalé zaměřuji pouze na webové aplikace.

1.1.3 Web services

Webové služby (web services) jsou speciální webové servery, jejichž účelem je podporovat procesy a potřeby webových stránek nebo aplikací [10]. Ke komunikaci s webovými službami používají klientské aplikace API (application programming interface), které jim server poskytuje. API obecně zpřístupňuje



Obrázek 1.1: Web API

sadu dat a funkcí pro interakci mezi počítačovými programy, což jim umožňuje výměnu informací [11]. Na obrázku 1.1 je pomocí diagramu znázorněna podoba Web API, které je vlastně fasádou webové služby naslouchající a odpovídající klientům.

Některé způsoby, jakými spolu klientská aplikace a webová služba mohou komunikovat, a datové formáty, které při komunikaci mohou využívat, jsou popsány v následujících sekcích.

1.2 Druhy webových aplikací

V této sekci se zaměřuji na dva pohledy na moderní webovou aplikaci. Principy těchto přístupů jsem využil později při návrhu a implementaci klientské aplikace.

1.2.1 RIA

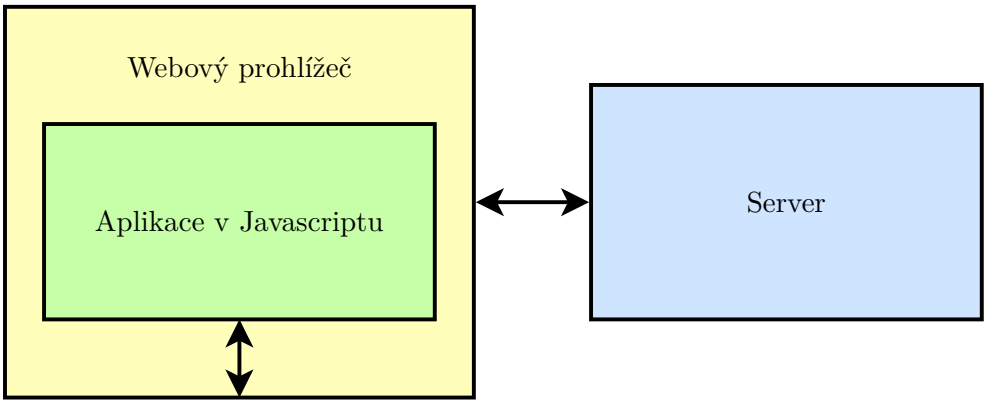
Bohatá internetová aplikace, v angličtině rich internet application, zkráceně RIA, je webová aplikace, která se chová a vypadá spíše jako desktopová aplikace, avšak běží ve webovém prohlížeči. Je spouštěna jako zásuvný modul, javascriptový kód nebo virtuální stroj [12].

V současné době jsou platformy využívající zásuvné moduly nebo virtuální stroje, jako je například Adobe Flash, JavaFX či Microsoft Silverlight, vytlačovány řešením založeném na HTML 5 a Javascriptu [13].

Strukturu RIA aplikace běžící v prohlížeči popisuje diagram na obrázku 1.2.

1.2.2 SPA

Nevýhodou klasického přístupu k tvorbě webových aplikací je to, že se soustřeďují pouze na vygenerování stránky pro prohlížeč, která je do něj poté pomocí sítě posílána. Pokud uživatel klikne na odkaz na webové stránce, musí se vygenerovat, přenést a vykreslit zcela nová stránka, vše od záhlaví, navigace, textu až po patičku, i když se na webové stránce oproti předchozí paradoxně



Obrázek 1.2: Schéma javascriptové RIA aplikace

změní například pouze text. Tento proces, který se zbytečně opakuje, vytěžuje server i síť.

Z tohoto důvodu vznikly jednostránkové aplikace, v angličtině single page application (SPA), které tyto nevýhody eliminují [14]. Principem SPA je běh aplikace v prohlížeči, která se načte ze serveru pouze napoprvé. Tato aplikace se sama stará o to, co se stane, pokud uživatel například klikne na odkaz. Zabrání prohlížeči přejít na prokliknutý odkaz, a změnu stránky si tak může provést sama. Není tedy nutné znovu načítat další stránku. Aplikace si ze serveru vyžádá například pouze změněný text, který ve vhodném formátu obdrží a ten původní jím ve stránce nahradí.

SPA musí být napsaná v programovacím jazyce, který dokáže interpretovat přímo uživatelův prohlížeč nebo zásuvný modul, který si uživatel v případě absence doinstaluje. Server i síť jsou v tomto případě tedy mnohem méně zatěžovány a aplikace dokáže uživateli rychleji servírovat obsah. Nevýhodou tohoto řešení je vyšší výpočetní náročnost na uživatelův počítač i prohlížeč. Tato nevýhoda však v dnešní době pozbývá významu s rostoucím výkonem osobních počítačů i chytrých telefonů.

1.3 Komunikace klienta a serveru

Pro komunikaci klienta a serveru je možné použít různé způsoby komunikace a typy datových formátů. V následující kapitole stručně shrnu především ty způsoby a typy, které se používají v prostředí webových aplikací.

1.3.1 Datové formáty

Aby spolu klient a server mohli komunikovat, je nutné definovat formát dat, ve kterém si budou posílat informace.

1.3.1.1 JSON

JavaScript Object Notation, zkráceně JSON, je jednoduchý formát pro výměnu dat. Jedná se o textový formát čitelný pro člověka a vhodný pro strojové zpracování. Je založen na podmnožině programovacího jazyka JavaScript. Formát JSON je definován od října roku 2013 standardem ECMA-404 [15]. Výhodou tohoto formátu je nezávislost na programovacím jazyce, avšak využívá dobře známých datových struktur v běžně používaných programovacích jazycích, což z něj dělá vhodný formát pro výměnu dat.

Data v tomto formátu mají vysokou datovou efektivitu, která se negativně projevuje na volné struktuře dat. Nemožnost validovat jejich strukturu lze eliminovat například pomocí nástroje JSON Schema [16], který však není součástí formátu.

1.3.1.2 XML

Extensible markup language, zkráceně XML, lze do češtiny přeložit jako rozšiřitelný značkovací jazyk. XML je obecným značkovacím jazykem určeným pro výměnu dat. Jeho účelem je umožnit tvorbu konkrétních značkovacích jazyků s cílem popisovat data.

Jazyk XML byl vyvinut organizací W3C, která jej 10. února 1998 standardizovala. Tento standard slouží jako specifikace jazyka [17].

XML dokument je stromová struktura založená na značkách, které označují elementy dokumentu. Každý element představuje uzel stromu a může obsahovat další elementy. XML dokument musí vždy obsahovat právě jeden kořenový element, který je kořenem stromu.

Výhodou tohoto jazyka je strukturovatelnost dat a jejich sémantický popis pomocí značek. Značky je navíc možné obohatit pomocí atributů o metadata (data o datech). Takto sestavená struktura podpořená validací pomocí definice DTD nebo XML Schema dělá z XML velmi robustní formát pro výměnu dat mezi různorodými systémy.

1.3.2 Způsoby komunikace

Existuje několik způsobů, jakými spolu mohou klient a server komunikovat. Každý z těchto způsobů je vhodný pro jiný druh aplikací. Já se zaměřím na dva způsoby komunikace, jeden procedurálně orientovaný a druhý datově orientovaný.

1.3.2.1 RPC

Remote procedure call, zkráceně RPC, můžeme do češtiny přeložit jako volání vzdálené procedury [18]. Jedná se o způsob komunikace, který umožňuje spustit proceduru v jiné aplikaci, která nemusí běžet na stejném stroji. Progra-

mátor se nemusí starat o podrobnosti tohoto spojení. Může spouštět vzdálené procedury stejným způsobem, jako procedury lokální.

Pomocí RPC je možné zavolat proceduru, která je identifikována jménem a předat jí argumenty. Tato procedura je vzdáleně spuštěna a její návratová hodnota je vrácena volajícímu. Tento přístup umožňuje definovat pouze rozhraní procedur ve vlastním kódu, jejich spuštění probíhá jinde a programátorovi může být zcela skryto.

1.3.2.2 XML-RPC

XML-RPC je RPC protokol využívající XML formátu pro popis volání procedury a HTTP protokolu pro jeho přenos [18].

Volání vzdálené procedury začíná odesláním HTTP požadavku na server. Klientem je v tomto případě program, který chce spustit jednu z procedur vzdáleného serveru. Vzdálené proceduře může být předáno několik vstupních parametrů, procedura následně vrací jednu návratovou hodnotu. Pomocí kolekcí a seznamů, které je do sebe možné vnořovat, lze přenášet větší datové struktury. Díky tomu je možné pomocí XML-RPC přenášet objekty či struktury jak ve vstupních parametrech, tak i v návratové hodnotě.

Pro autentizaci klienta je nutné použít metody, které nabízí přenosový protokol. Pomocí Basic authentication (v překladu jednoduché ověření) protokolu HTTP je možné identifikovat klienta podle uživatelského jména a hesla. Pro komplexnější identifikaci klienta je možné využít klientského certifikátu [19] protokolu HTTPS, který navíc zajišťuje šifrování přenosu.

XML-RPC definuje několik datových typů, které je možné přenášet.

1.3.2.3 JSON-RPC

JSON-RPC je RPC protokol využívající JSON formátu [20]. Je jednoduchý a velmi podobný XML-RPC. Nedefinuje datové typy jako XML-RPC, využívá jen typů definovaných v JSON. JSON-RPC umožňuje posílat upozornění (data jsou odeslána na server, ale není vyžadována odpověď) a spouštět několik souběžných procedur, které mohou být zpracovány paralelně a na které nemusí být odpovězeno ve stejném pořadí, v jakém byly spuštěny.

JSON-RPC existuje ve verzích 1.0 a 2.0. Volání procedury probíhá stejně, jako v případě XML-RPC. Od verze 2.0 lze mimo HTTP, jako protokolu pro přenos, použít také TCP/IP socket.

Požadavek na spuštění vzdálené procedury obsahuje objekt v JSON formátu, který obsahuje položky popisující volání a předávané parametry. Server spustí požadovanou proceduru a poté odpovídá na požadavek JSON objektem, který klienta informuje o výsledku.

Velmi často není nutné na odeslaný požadavek odpovídat (procedura nemá co vrátit), proto JSON-RPC umožňuje odeslat požadavek, na který server

nebude odpovídat. Tento druh volání definuje JSON-RPC jako notification, což můžeme do češtiny přeložit jako upozornění.

1.3.2.4 REST

Representational state transfer (REST) je rozhraní umožňující přístup k datům na serveru pomocí HTTP volání.

REST byl navržen a popsán Royem Fieldingem v rámci disertační práce „Architectural Styles and the Design of Network-based Software Architectures“ v roce 2000 [21]. Principy REST popisuje v kapitole 5. Toto rozhraní se používá k přístupu ke zdrojům (resources), které mohou představovat jak data, tak stavy aplikace. Zdroje jsou identifikovány pomocí URI (Uniform Resource Identifier) a pracuje se s nimi pomocí čtyř základních metod HTTP. Na rozdíl od procedurálně orientovaných RPC či SOAP je REST orientován datově.

Zdroje jsou vzájemně provázány pomocí odkazů, které se nacházejí v odpovědi od serveru. Tento princip se nazývá HATEOAS (Hypermedia as the Engine of Application State) [22]. Server vrací reprezentaci zdroje, která může mít různé podoby (XML, JSON, PDF...). K manipulaci se zdrojem se využívá čtyř CRUD (Create-Read-Update-Delete) operací.

Mezi vlastnosti a výhody REST patří:

- oddělení klienta a serveru,
- bezstavovost,
- snadná rozšířitelnost,
- využití mezipaměti (cache),
- a code-on-demand (server dodá kód rozšiřující funkčnost klienta).

Kapitola 2

Analýza

Tato kapitola se zabývá analýzou potřeb a procesů modelové advokátní kanceláře. Na základě této analýzy jsem navrhl informační systém, který jsem následně podle tohoto návrhu implementoval a řádně otestoval.

2.1 Současný stav

Advokátní kancelář zabývající se právními službami a poradenstvím nabízí své služby ve formě produktů, které je možné si objednat.

Objednávky klientů je nutné evidovat. Zpracování objednávky může trvat i několik týdnů v závislosti na součinnosti klienta a uskutečnění objednaných služeb. Z tohoto důvodu je nutné, aby měli zaměstnanci kanceláře přehled o stavu zpracovávaných objednávek.

V současné době využívá advokátní kancelář pro správu objednávek následující nástroje:

Evidence objednávek
: Sdílený dokument tabulkového procesoru.

Komunikace s klientem
: Telefon a e-mail, případně osobní schůzka. Klíčové body komunikace se zapisují do dokumentu objednávky.

Tvorba právních dokumentů
: Vyplnění připravené šablony v textovém procesoru, vygenerování PDF

Sdílení dokumentů s klientem
: Dokumenty se při komunikaci s klientem přikládají do emailu a také se ukládají do sdíleného adresáře.

Při vyřizování objednávky je potřeba dodržet postup stanovený interním předpisem. Tento postup je ovšem náchylný k chybám, které používané nástroje neeliminují. Například všechna odeslaná i přijatá komunikace s klientem

musí být evidována u patřičné objednávky. Pokud ji ovšem zaměstnanec kanceláře zapomene zaevidovat nebo ji zaeviduje špatně, nebude k dohledání a při případném řešení sporů bude advokátní kanceláři chybět. Rovněž při ručním přepisování informací o klientovi a případu do právních dokumentů může dojít k chybám či překlepům.

Současný postup se tak díky narůstajícímu počtu objednávek ukázal jako nedostatečný. Po konzultaci s poradenskou společností se advokátní kancelář rozhodla pro pořízení informačního systému.

2.2 Procesy

Proces je sada uspořádaných aktivit. Tyto aktivity transformují vstupy na výstup [23]. Hlavním procesem, který bude navrhovaný informační systém podporovat, je vyřízení objednávky. Do tohoto procesu vstupují údaje o klientovi a jeho požadavku ve formě objednávky, které jsou transformovány na právní služby a dokumenty. Tento proces se skládá z několika dílčích procesů, které jsem v rámci analýzy namodeloval.

Pomocí analýzy procesů, které probíhají v advokátní kanceláři při vyřizování objednávek, jsem:

- lépe pochopil činnost zaměstnanců,
- byl schopen přesněji specifikovat požadavky na výsledný systém,
- lépe tyto procesy v systému podporovat
- a identifikovat problémová místa.

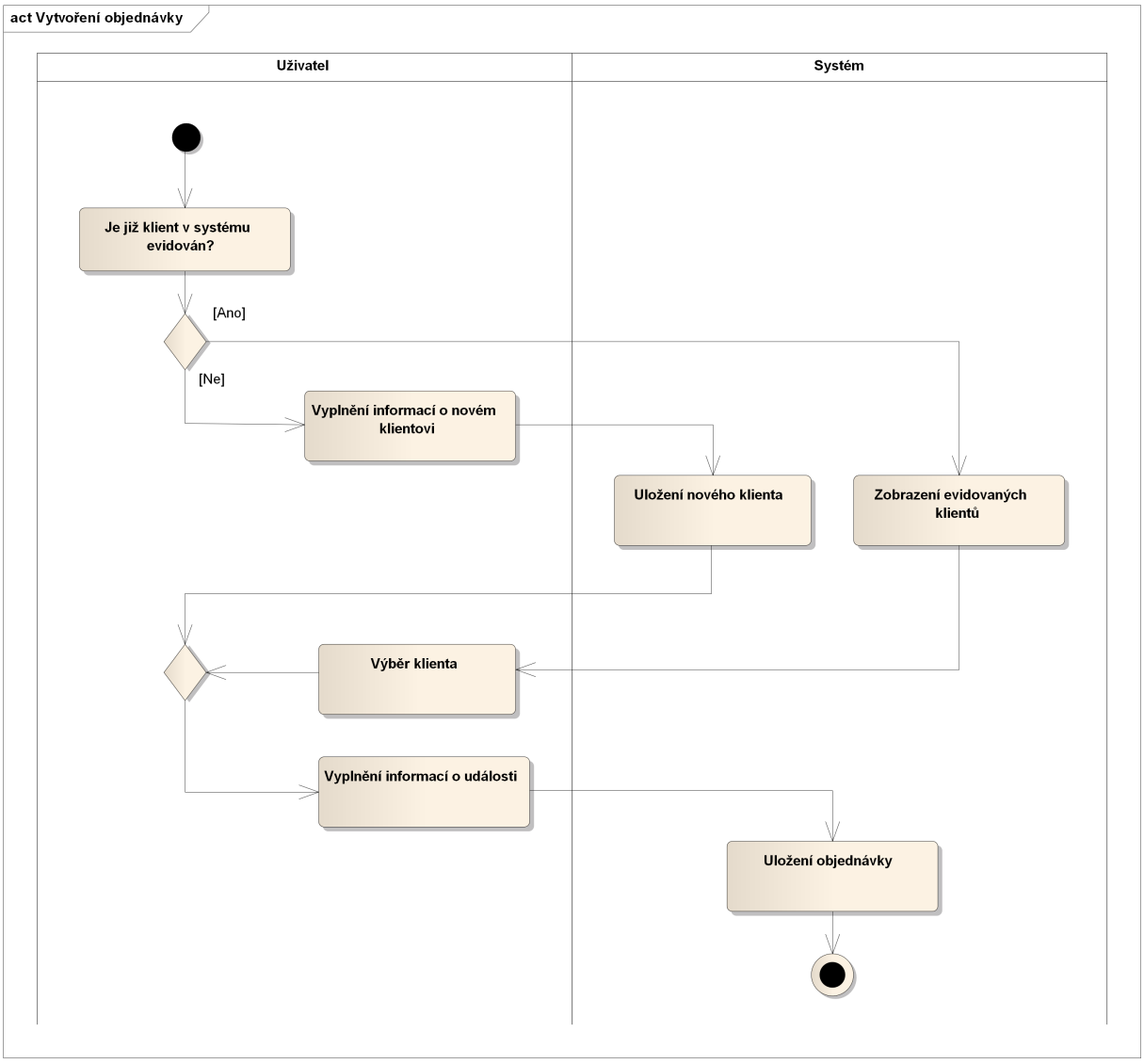
Tyto procesy, tak jak by vypadaly s podporou navrhovaného informačního systému, jsem zachytil ve formě diagramu aktivit pomocí grafického jazyka UML.

2.2.1 Vytvoření objednávky

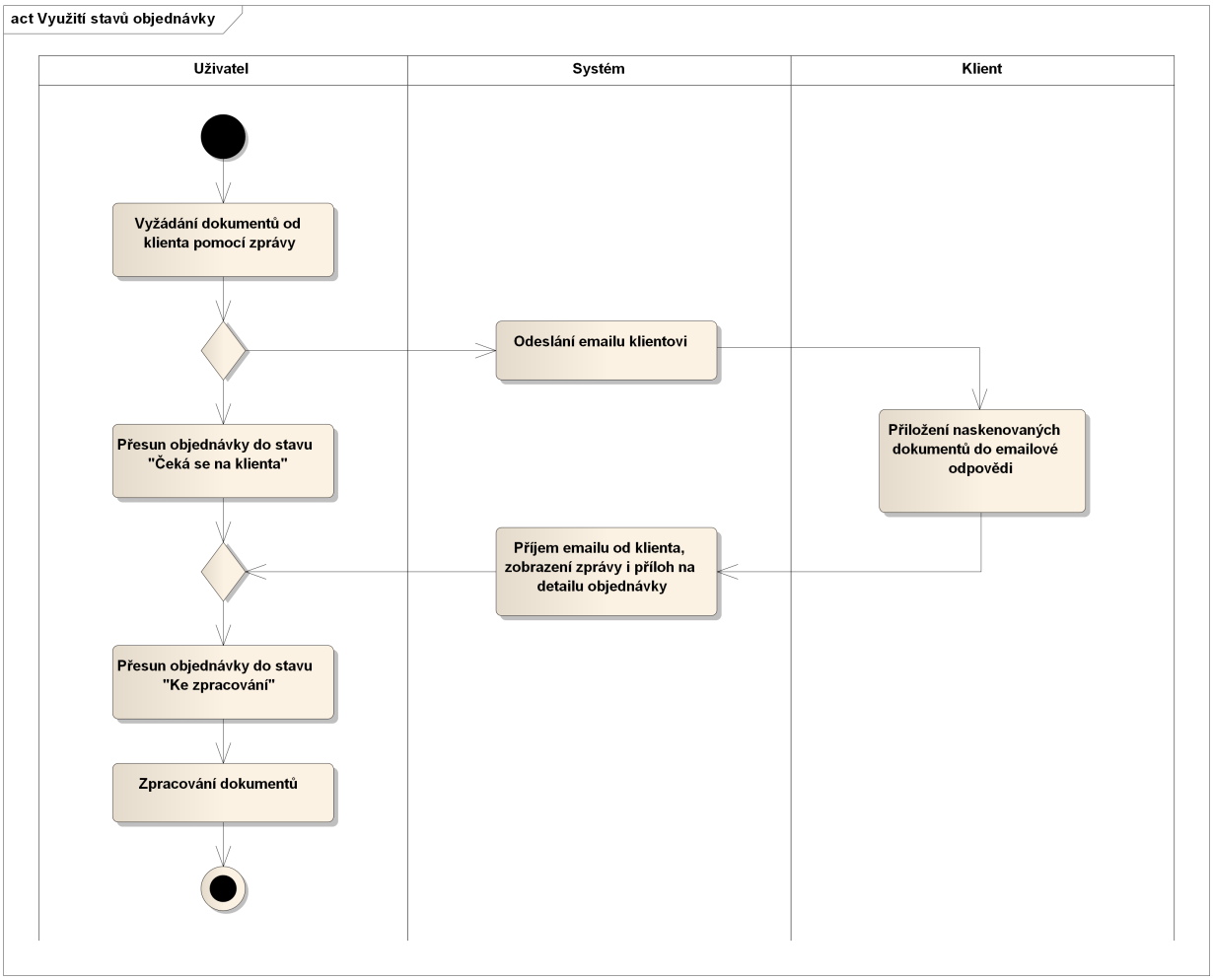
Diagram procesu vytvoření objednávky na obrázku 2.1 popisuje vytvoření objednávky nového nebo již evidovaného klienta kanceláře. Hlavní aktivitou v tomto procesu je vyplnění potřebných údajů.

2.2.2 Využití stavů objednávky

Diagram na obrázku 2.2 ukazuje, jak by navrhovaný informační systém podporoval změny stavu objednávky při vyžádání podkladů od klienta nutných k vyřízení objednávky.



Obrázek 2.1: Vytvoření objednávky



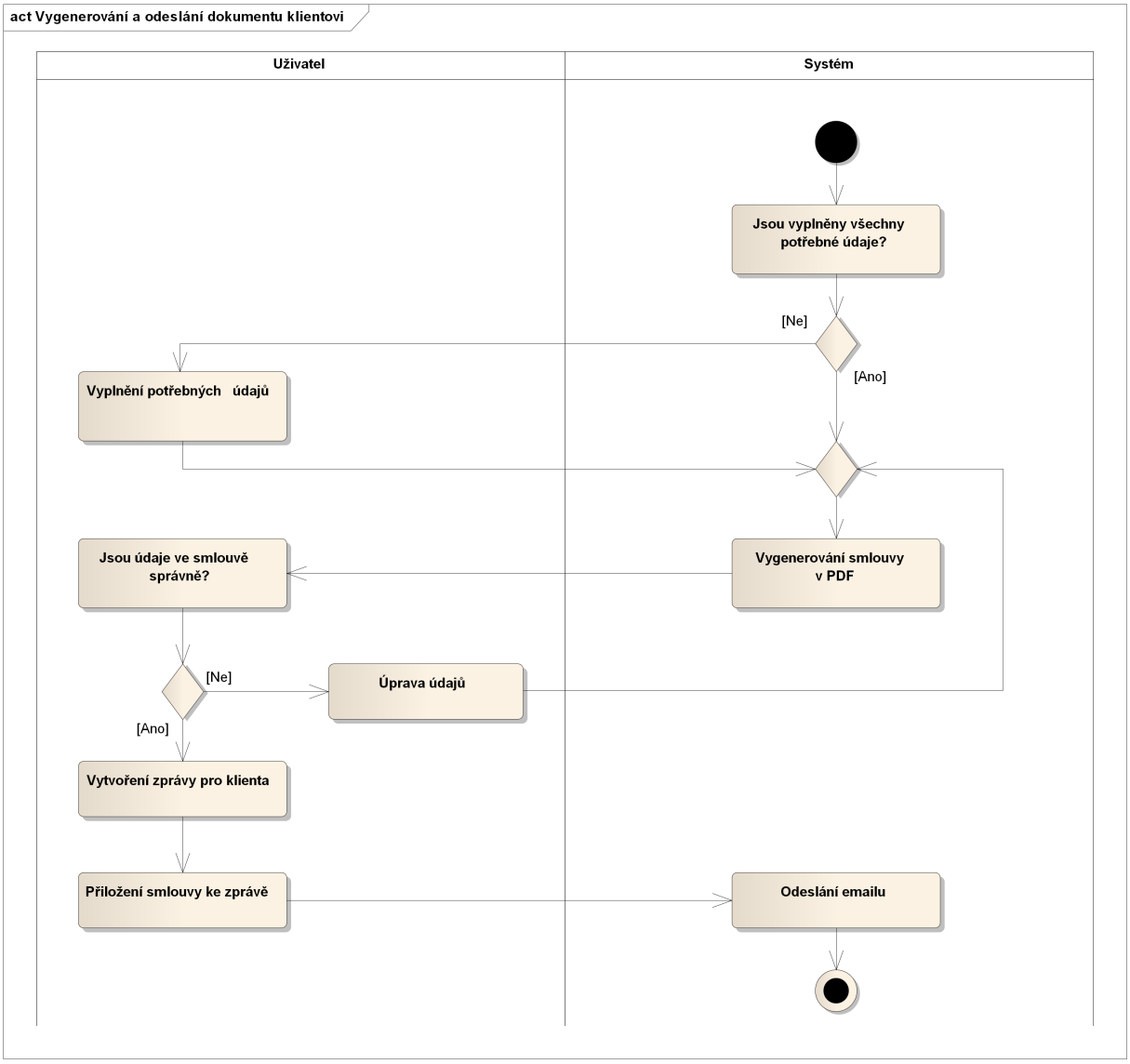
Obrázek 2.2: Využití stavů objednávky

2.2.3 Vygenerování a odeslání právního dokumentu

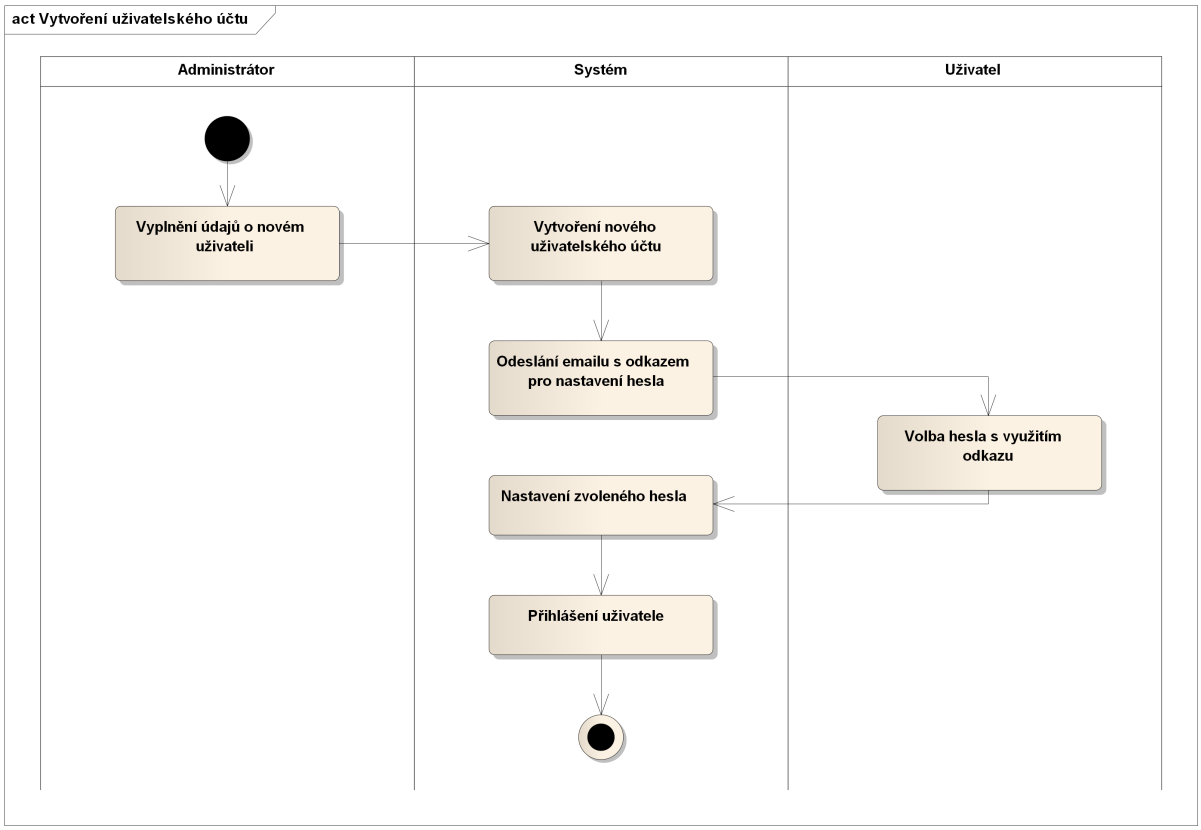
Vygenerování právního dokumentu a jeho následné odeslání klientovi v příloze e-mailu popisuje diagram na obrázku 2.3. Systém před vygenerováním ověřuje, zda jsou vyplněny všechny údaje, které budou do smlouvy potřeba. Advokát systémem vygenerovaný dokument zkontroluje. Tento dokument poté spolu se zprávou o jeho vytvoření odesílá prostřednictvím e-mailu klientovi.

2.2.4 Vytvoření uživatelského účtu

Proces vytvoření uživatelského účtu je zachycen diagramem na obrázku 2.4. Administrátor v systému založí nový uživatelský účet. Na e-mail, který administrátor při zakládání uvedl, bude zaslán e-mail s odkazem pro nastavení



Obrázek 2.3: Vygenerování a odeslání právního dokumentu

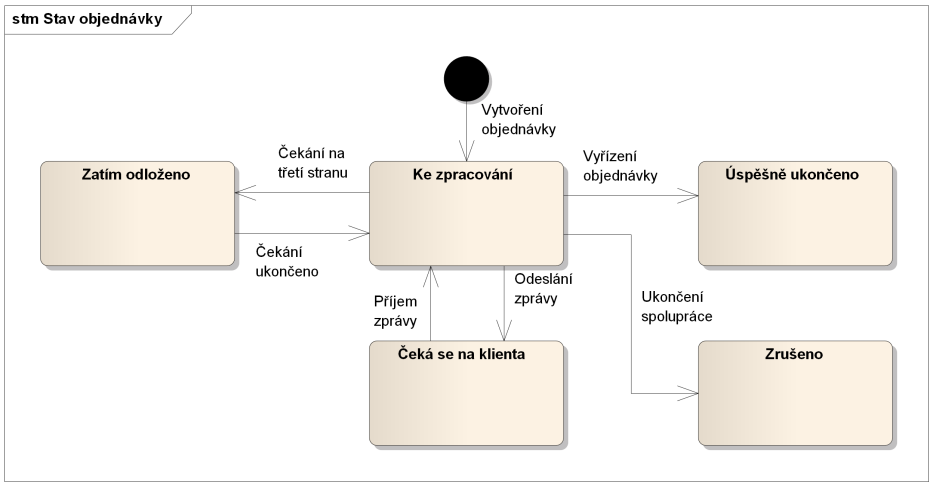


Obrázek 2.4: Vytvoření uživatelského účtu

hesla. Na odkazované stránce si nový uživatel nastaví své heslo pro vstup do systému. Nyní může uživatel v systému začít pracovat.

2.2.5 Pozice objednávky v procesu

Aktuální pozice objednávky v procesu jejího vyřizování bude v navrhovaném systému zachycena pomocí stavu. Stavy odpovídají předpisu advokátní kanceláře a umožňují lépe řídit zpracování objednávky. Možné stavy objednávky a přechody mezi nimi popisuje diagram na obrázku 2.5. Důležitý je stav „Čeká se na klienta“. Pokud klient odešle advokátovi e-mail a jeho objednávka se nachází v tomto stavu, systém automaticky změní stav objednávky na „Ke zpracování“. Advokáta tedy zajímají primárně objednávky v tomto stavu, protože ty potřebují být vyřízeny. Pokud objednané služby čekají na součinnost třetí strany, je možné přesunout objednávku do stavu „Zatím odloženo“.



Obrázek 2.5: Stavy objednávky

2.3 Požadavky

Po provedení analýzy procesů probíhajících při zpracování objednávky byl sestaven následující výčet funkčních požadavků na informační systém:

Řízení přístupu
: Přístup do systému bude možný s použitím firemní e-mailové adresy zaměstnance kanceláře spolu s heslem. V systému budou zavedeny dvě uživatelské role: role pro zaměstnance a role pro administrátora. Uživatelský účet s přiřazenou zaměstnaneckou rolí bude moci spravovat klienty a objednávky. Účet v roli administrátora bude mít navíc možnost spravovat uživatele systému.

Vytváření objednávek a klientů
: Objednávky budou zakládány zaměstnancem kanceláře při prvním kontaktu s klientem. Klienti budou v systému evidováni i po dokončení objednávky, díky čemuž bude možné jejich údaje znovu použít, pokud se klient rozhodne využít služeb advokátní kanceláře znovu. Každá objednávka a klient budou mít definované položky, které se budou v systému evidovat. Některé položky budou systémem vyžadovány (povinné položky), jiné budou volitelné (nepovinné položky). Systém bude zadané vstupy validovat (tvar e-mailu, správnost telefonního čísla...).

Stav objednávky
: Každá objednávka bude mít v určitý okamžik přiřazen právě jeden stav. Stavy budou pevně dané a budou vycházet z postupu, jakým zaměstnanci advokátní kanceláře objednávky zpracovávají. Mezi stavy půjde

přecházet podle stanovených pravidel. Stavy budou sloužit k získání přehledu v objednávkách a půjde podle nich filtrovat.

Odpovědná osoba
Ke každé objednávce bude možné přiřadit uživatele, který je odpovědný za její vyřízení. Uživatelé si na výpisu objednávek budou moci zobrazit pouze objednávky, ke kterým jsou přiřazeni.

Komunikace s klientem
Pomocí informačního systému bude možné s klientem komunikovat přímo z detailu objednávky. Takto vytvořená komunikace tak bude automaticky evidována u objednávky. Primárně se z informačního systému budou odesílat e-maily na adresu evidovaného klienta a příchozí odpovědi opět přijímat systémem. Pokud to ovšem bude nutné, je možné se s klientem domluvit na telefonním hovoru či osobní schůzce. Informace o průběhu hovoru nebo schůzky pak musí advokát vyplnit k objednávce sám.

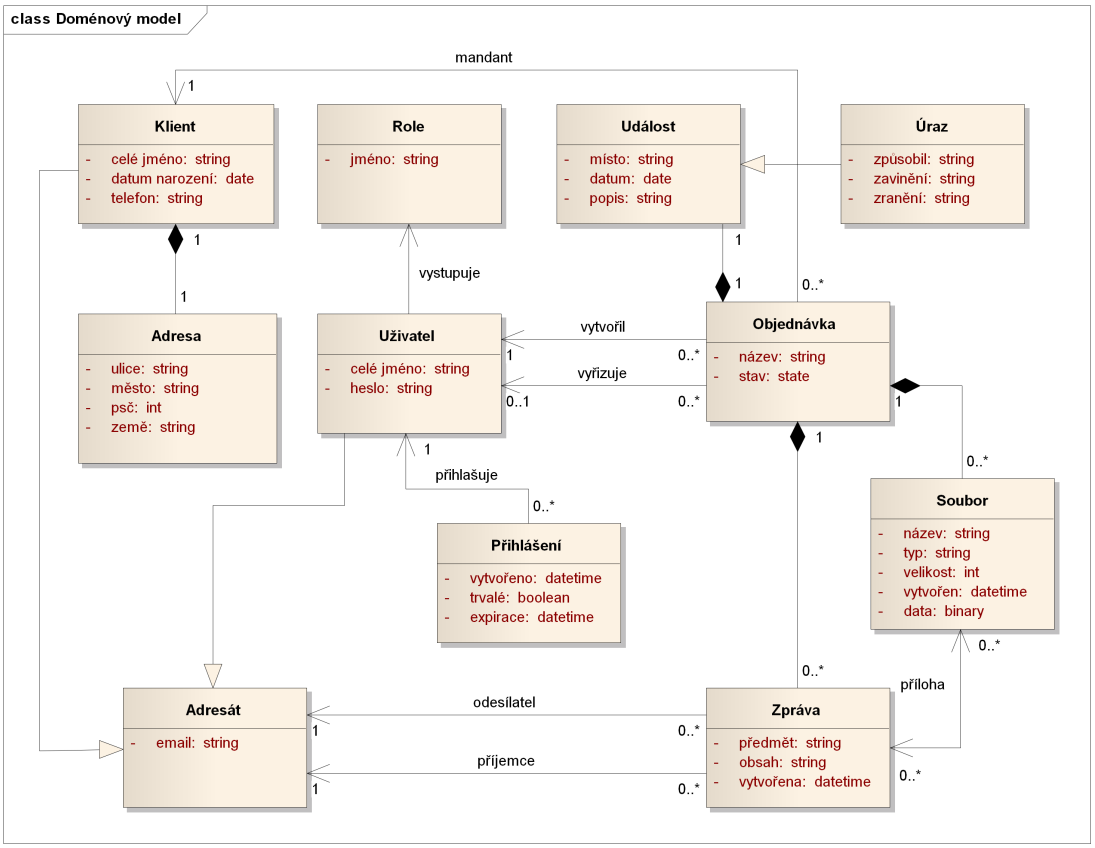
Tvorba právních dokumentů
Ke každé objednávce bude možné vygenerovat právní dokument podle implementovaných šablon. Do těchto dokumentů se automaticky doplní údaje objednávky a klienta. Výstupem bude soubor PDF, který advokát pouze zkontroluje.

Sdílení dokumentů s klientem
U každé objednávky bude možné nahrávat do systému dokumenty, které půjdou přiložit k e-mailu pro klienta. Soubory příložené k přijatým e-mailů od klienta se budou automaticky nahrávat do systému a bude možné je dohledat na detailu objednávky.

2.4 Doménový model

Na základě předcházející analýzy jsem sestavil doménový model. Cílem doménového modelu je [24]:

- popsat data,
- popsat vazby mezi entitami,
- identifikovat jejich stavy,
- zachytit atributy
- a být základem pro návrh.



Obrázek 2.6: Doménový model

Navržený model umožňuje splnit všechny sepsané požadavky na navrhovaný informační systém. Podoba většiny entit vyplynula z analýzy procesů a požadavků, které jsem popisoval v předcházejících sekcích 2.2 a 2.3. Některé entity jsem navíc dekomponoval.

Zpráva je vždy směrována od advokáta ke klientovi či obráceně. Je vázaná na jejich e-mailové adresy, proto jsem tento atribut z obou entit vyčlenil do entity rodičovské, která představuje adresáta (nebo odesílatele) zprávy.

Navíc jsem do doménového modelu přidal entitu přihlášení, která do něj přidává informaci o přihlašování uživatelů.

Takto sestavený doménový model jsem zachytil pomocí diagramu tříd v UML na obrázku 2.6. V modelu je zatím zachycen pouze jeden druh události (úraz). Další druhy by byly průběžně doplňovány za provozu informačního systému podle potřeb advokátní kanceláře.

Po úspěšném provedení analýzy jsem postoupil k návrhu informačního systému.

Kapitola 3

Návrh

V kapitole 2 jsem se věnoval analýze informačního systému, jejíž cílem bylo pochopit procesy probíhající v advokátní kanceláři a definovat vlastnosti navrhovaného informačního systému. Cílem této kapitoly je popsat, jakým způsobem budou naplněny požadavky, které vzešly z provedené analýzy.

3.1 Výběr technologií

Ještě před samotným návrhem architektury jsem se rozhodl vybrat technologie, které k realizaci použiji. Návrh architektury je v tomto případě silně vázán na zvolený framework, protože většina frameworků přichází se svou vlastní architekturou.

3.1.1 Klientská aplikace

Klientskou aplikaci jsem se rozhodl napsat v Javascriptu z důvodu jeho podpory v každém současném prohlížeči. Použití jiných programovacích jazyků, pro které by bylo potřeba doinstalovávat zásuvné moduly nebo využívat kompilátoru do Javascriptu, jsem v tomto případě zavrhl.

3.1.1.1 AngularJS

Pro urychlení a zjednodušení vývoje vybuduji aplikaci nad javascriptovým frameworkem určeným pro SPA aplikace. Rozhodl jsem se použít AngularJS, který se již delší dobu chci naučit. S tímto frameworkem nemám téměř žádné zkušenosti, proto pro mě bude tvorba aplikace postavené nad tímto frameworkem výzvou. AngularJS je open source framework vyvíjený společností Google [25]. Má velmi rozsáhlou dokumentaci, mnoho rozšíření a hotových komponent od velkých společností i jednotlivců. Řeší problémy, na které vývojář často naráží při vývoji SPA aplikací, čímž urychluje jejich vývoj. Jeho cílem

je zjednodušit vývoj a testování webových aplikací poskytnutím frameworku pro MVC architekturu spolu s komponentami používanými v RIA aplikacích.

Mezi klíčové vlastnosti AngularJS frameworku patří [26]:

- two-way data binding (oboucestné datové provázání),
- dependency injection (vkládání závislostí),
- oddělení konceptů,
- testovatelnost,
- a vysoká úroveň abstrakce.

Oddělení konceptů je v AngularJS realizováno pomocí rozdělení aplikace do komponent: kontrolery, šablony, direktivy, služby, filtry a další.

Angular nejdříve projde HTML stránku, do které byly umístěny speciální HTML značky nebo atributy. Tyto značky a atributy představují direktivy Angularu, které jsou schopné manipulovat s DOM dokumentu [27]. AngularJS obsahuje velké množství standardních direktiv (provázání dat s input elementem, kliknutí na tlačítko, odeslání formuláře, podmíněný výpis, iterování nad elementem...) Programátor si také může definovat vlastní direktivy, které jsou vhodné obzvláště pro znovupoužitelné komponenty.

Do HTML šablon je také možné zapisovat jednoduché javascriptové výrazy obohacené o speciální konstrukty a filtry [28]. Tyto výrazy jsou vyhodnocovány v kontextu aktuálního scope, což je javascriptový objekt sloužící k propojení HTML šablony a javascriptového kódu. Jakákoliv změna v datech umístěných ve scope, ať už na straně HTML šablony nebo Javascriptu, je automaticky promítána také do scope objektu na druhé straně. Tato vlastnost se nazývá two-way data binding.

Myšlenkou Angularu je, že pro stavbu uživatelských rozhraní a provázání softwarových komponent je vhodné využívat deklarativního programování a imperativní programování by pak mělo být použito jen pro business logiku aplikace [29]

Pro automatické načítání javascriptových souborů použiji RequireJS. Jedná se o jednoduchý asynchronní loader skriptů, který umožňuje definovat závislosti mezi soubory [30].

3.1.1.2 HTML a CSS

Grafické rozhraní aplikace bude vytvořeno za pomoci HTML a CSS. Pro urychlení vývoje použiji HTML 5 Boilerplate, což je open-source frontendová šablona zvěřejněná pod MIT licencí [31]. Boilerplate obsahuje:

- responzivní HTML šablonu,

- optimalizovaný Google Analytics kód,
- připravené všechny druhy ikon pro dotyková zařízení,
- CSS reset pro HTML 5,
- základní tiskové styly,
- polyfill knihovna Modernizr,
- odladěný konfigurační soubor pro server.

Modernizr je javascriptová knihovna poskytující a emulující chybějící technologie a rozhraní v prohlížečích, které částečně nebo zcela nepodporují HTML 5. Kód, který toto provádí, se nazývá polyfill [32]. Programátor tak může předpokládat, že uživatelův prohlížeč podporuje všechny HTML 5 technologie a plně s nimi pracovat. Vyššího výkonu výsledné aplikace je docíleno odladěným konfiguračním souborem .htaccess pro webový server Apache.

Pro rychlejší vytvoření prototypu informačního systému jsem se rozhodl použít CSS framework. CSS frameworky poskytují styly pro HTML elementy, nabízejí speciální komponenty a zjednodušují tvorbu responzivních layoutů.

Při návrhu informačního systému jsem vybíral z frameworků: Blueprint, Bootstrap, Foundation, Semantic UI a Pure.

Nakonec jsem se rozhodl pro Bootstrap vyvinutý zaměstnanci společnosti Twitter [33]. S tímto frameworkem mám nejvíce zkušeností a pravidelně se s ním setkávám na různých projektech, na kterých spolupracuji. Jeho výhodou je početná komunita a podrobná dokumentace. Navíc existuje sada UI komponent pro AngularJS stylovaná pomocí Bootstrap frameworku, které bych chtěl v informačním systému využít [34]. V neposlední řadě také obsahuje poměrně velký set vektorových ikon Glyphicons, které lze velmi jednoduše používat např. v odkazech a na tlačítkách pro zrychlení orientace uživatele v grafickém rozhraní.

3.1.2 Serverová aplikace

Serverovou část informačního systému jsem se rozhodl napsat v PHP, které poběží jako modul webového serveru Apache. PHP je široce používaný jazyk pro tvorbu webových aplikací. Výhodou je jeho dostupnost v nabídce tuzemských hostingových služeb. Stejná výhoda platí i pro webový server Apache.

Serverovou aplikaci vytvořím za pomocí Nette frameworku. Nette framework je sada 21 nezávislých komponent, které ulehčují vývoj webových aplikací v PHP [35]. Tyto komponenty je možné využívat jako celek (framework) nebo použít jen ty, které bude aplikace potřebovat. Sestavení závislostí aplikace se děje pomocí balíčkovacího nástroje Composer. Pro zjednodušení vývoje využiji celý framework, namísto ručního sestavení konfigurace vybraných komponent.

Mezi vlastnosti Nette frameworku patří:

- automatická eliminace bezpečnostních děr (XSS, CSRF, session hijacking, session fixation atd.)
- ladící nástroje,
- rychlost,
- aktivní komunita (v Česku)
- architektura,
- událostmi řízené programování,
- rozšířitelnost.

V Nette frameworku tvořím webové stránky a aplikace již 4 roky, s jinými PHP frameworky mám jen málo zkušenosti (Symfony) nebo žádné (Zend, Laravel), proto jsme volil právě tento framework. V kombinaci s Nette frameworkem také využiji některé komponenty z projektu Kdyby, což je soubor rozšiřujících komponent pro Nette framework.

Pro generování dokumentů bude použita knihovna mPDF [36]. Tato knihovna napsaná v PHP dokáže vygenerovat z HTML dokumentu PDF soubor.

3.1.3 Uložení dat

Aplikace bude data ukládat do relační databáze s pomocí ORM knihovny Doctrine 2 [37]. ORM je objektově relační mapování, které zajišťuje konverzi mezi daty uloženými v relační databázi a objekty v aplikaci. Doctrine odstiňuje programátora od konkrétního databázového stroje, z tohoto důvodu je možné zvolit libovolný z podporovaných. Z čistě praktických důvodů jsem zvolil MySQL, které je na mém hostingu k dispozici.

3.2 Architektura

Informační systém bude rozdělen na dvě oddělené aplikace, které spolu budou komunikovat pomocí REST API. Data si budou vyměňovat ve formátu JSON, který byl popsán v sekci 1.3 věnující se teorii komunikace klienta a serveru. Každá z aplikací bude napsána v programovacím jazyce, který je možné použít v prostředí, ve kterém aplikace poběží.

3.2.1 Klientská aplikace

Klientská aplikace se skládá ze tří hlavních částí: z HTML šablon, kontrolerů a zdrojů (resources).

Šablony jsou klasické HTML dokumenty doplněné o speciální značky a atributy.

Úkolem kontroleru je řídit tok dat od uživatele k serveru a naopak. Kontroler je implementován jako konstrukční funkce, která ve svých argumentech obdrží požadované závislosti.

Cílem zdroje je poskytnout základní metody pro CRUD operace nad položkou a kolekcí, které vrátí instanci odpovědi. V této odpovědi půjdou nastavit obslužné funkce pro úspěšné zpracování požadavku, chyby a doručení odpovědi (success, error a finally). Pokud tyto funkce nebudou nastaveny, použijí se funkce výchozí, které lze definovat přímo na objektu konstrukční funkce odpovědi (statická metoda).

3.2.2 Serverová aplikace

Serverová aplikace se skládá z presenterů a modelu.

Každý presenter představuje jeden kontrétní zdroj (resource). Jeho úkolem je přijmout požadavek, zkontrolovat jeho správnost, zpracovat jej a odeslat odpověď. Všechny presentery v aplikaci dědí od API presenteru, který poskytuje metody pro získání dat z požadavku, odesílání odpovědí a chybových zpráv v jednotném tvaru. Pokud je resource přístupný pouze pro přihlášeného uživatele, nedědí presenter přímo od API presenteru, ale od zabezpečeného presenteru, který ověří sezení.

Modelová část se skládá z entit a služeb. Podoba entit vychází z doménového diagramu na obrázku 2.6 v kapitole 2 věnované návrhu. Pro práci s entitami se využívá ORM knihovny Doctrine 2. Složitější procesy, nezávislá funkčnost a případné duplicity v logice jsou z presenterů vyčleněny do služeb.

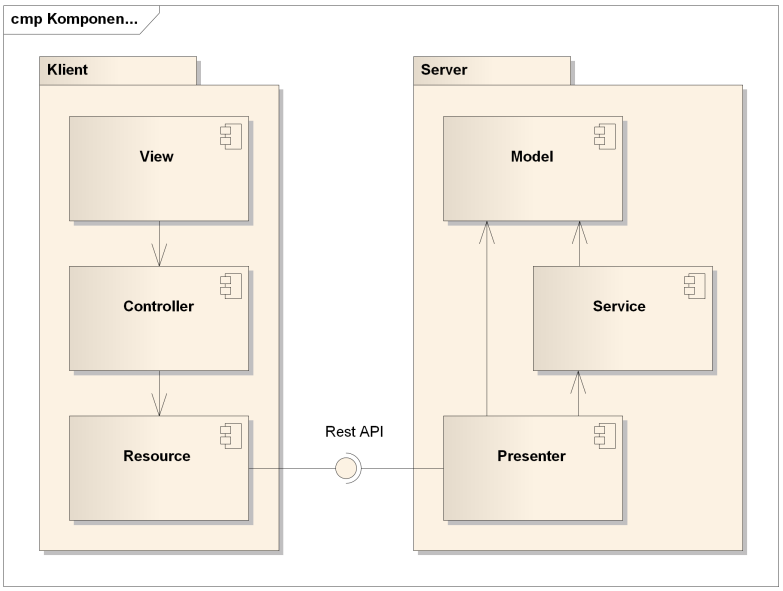
Mimo presentry a model se v serverové aplikaci ještě nacházejí třídy zodpovědné za sestavení e-mailu a jeho odeslání. Tyto třídy využívají pro sestavení těla e-mailu šablonovací jazyk Latte.

Směrování požadavků ke správným presenterům obstarává směrovač (router), který je vytvořen a nakonfigurován továrnou (factory). Ke směrování požadavků i podle HTTP metody, pomocí které byl odeslán, slouží rozšířená třída ApiRoute.

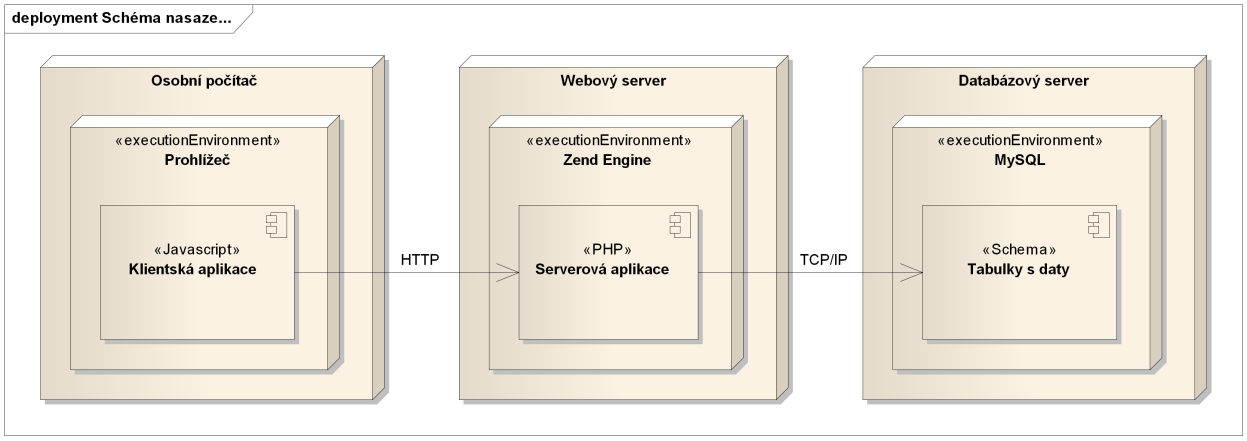
Jednotlivé komponenty vystupující v celém systému a směr volání popisuje diagram na obrázku 3.1.

3.3 Model nasazení

Podobu aplikace z pohledu nasazení v běhovém prostředí zachycuje diagram na obrázku 3.2. Aplikace jako celek je konstruována pro běh na třech stro-



Obrázek 3.1: Diagram komponent



Obrázek 3.2: Model nasazení

jích. Databázová část však může běžet na stejném stroji jako serverová část aplikace.

Kapitola 4

Implementace a testování

Informační systém jsem implementoval na základně návrhu, který byl popsán v předchozí kapitole. Systém se skládá ze dvou oddělených aplikací, které bylo potřeba vytvořit. Aplikace serveru se nachází v adresáři `app` a aplikace klienta v adresáři `www/app`. Adresář `www` je document root pro webový server Apache. Pro případy, kdy by nebyl možný posun document rootu (např. sdílený hosting), je zajištěn jeho posun pomocí konfiguračního souboru `.htaccess`.

4.1 Klientská aplikace

Klientská aplikace je rozdělena na moduly dle konvencí použitého frameworku AngularJS. Kód pro každý z modulů je umístěn v samostatném souboru, který načítá knihovna RequireJS.

Aplikace klienta je inicializovaná v souboru `www/app/init.js`, kde se načítají všechny moduly obsahující routovací pravidla a spouští framework Angular. Kód konfigurující načítání souborů pomocí RequireJS a spouštějící tuto inicializaci je umístěn v souboru `www/js/main.js`. V tomto souboru je navíc ještě přidána všem instancím Date metoda `toString`, která je převede na řetězec dle normy ISO 8601. Datum v této podobě je v aplikaci používáno při přenosu data na server pomocí formátu JSON. Pouze tento soubor je nutné spolu s použitými knihovnami pomocí script tagu vložit do HTML stránky klienta.

Mimo moduly, které představují konkrétní užitou hodnotu pro uživatele, se aplikace skládá ze znovupoužitelných modulů. Mezi tyto moduly patří:

- Modul GUI, starající se o sběr a vykreslování zpráv informujících uživatele o výsledku jeho akce.
- Modul REST, sloužící k práci s REST API a ke zpracování chyb.
- Modul sekcí aplikace, který obsahuje logiku obnovy sezení (popsáno v sekci 4.4) a načítání šablon layoutů.

4.2 Serverová aplikace

Kód serverové aplikace postavené nad Nette frameworkem je rozdělen do pěti částí:

- **Modelová část** – Tato část aplikace pracuje s daty. Skládá se z entit, služeb a SQL filtrů. Entity zapouzdřují data a operace nad nimi. Služby slouží k provádění náročnějších operací nad entitami, nebo vznikly vyčleněním opakujícího se kódu z presenterů. Obsahují také datové operace, které nepatří k žádné entitě.

- **Prezentační část** – Obsahuje presentery, třídy zodpovědné za zpracování požadavků na REST API. Tyto třídy pracují s entitami a službami, mohou také obsahovat vlastní logiku. Jejich cílem je vykonávat příkazy od klienta a vracet požadovaná data.

- **Routovací část** – Provádí routování (směrování) HTTP požadavků na konkrétní akce presenterů.

- **E-mailová část** – Úkolem této části je vytvářet a odesílat e-maily. Obsahuje třídy představující jednotlivé druhy e-mailů. Těla e-mailů jsou vytvořena za pomocí šablon šablonovacího jazyka Latte.

- **Dokumentová část** – Zodpovídá za generování dokumentů z údajů uvedených v objednávce a u klienta. Výstupem jsou dokumenty ve formátu PDF.

4.3 Autentizace

Autentizace uživatele je v informačním systému prováděna dvěma způsoby, které nyní popíši. Důležitý je také způsob uložení dat, pomocí kterých se uživatel autentizuje.

4.3.1 Uložení dat

Uživatelé systému se autentizují pomocí e-mailové adresy a hesla. Tyto údaje jsou uložené v databázi. Heslo ovšem není uloženo přímo v čitelné podobě, je uložen pouze jeho hash („otisk“). Hash je na serveru vytvářen pomocí PHP funkce `password_hash`, která ve výchozím režimu využívá algoritmu Bcrypt [38]. Bcrypt je adaptivní kryptografická hashovací funkce, která z tajného hesla vytvoří otisk doplněný o informace identifikující použitý algoritmus a parametry, s jakými byl vytvořen [39]. Bcrypt před začátkem hashování vygeneruje náhodou „sůl“, která bude součástí procesu hashování, a která zaručí, že pro stejný vstup (heslo) funkce vygeneruje rozdílné výstupy. Správnost

všech vygenerovaných klíčů lze ověřit vůči tajnému vstupnímu heslu. Algoritmus Bcrypt navíc provádí odvozování v iteracích, jejichž počet lze nastavit, a řídit tak časovou náročnost celé operace.

Tyto vlastnosti velmi znesnadňují případnému útočníkovi zjistit hesla uživatelů při úniku dat z databáze [40]:

- Díky použití „solených“ hashů nemůže útočník použít metodu založenou na jejich porovnávání. Tato metoda spočívá v tom, že si útočník zvolí nějaké heslo, u kterého předpokládá, že jej nějaký uživatel bude používat. Spočítá otisk hesla pomocí hashovací funkce, která pro stejné vstupy vrací stejný hash, a tento hash vyhledá v databázi uživatelů. Pokud jej nalezne, ví, že uživatel zvolené heslo používá.

- Protože lze u algoritmu Bcrypt řídit časovou složitost, lze ji nastavit tak, aby případný útočník nebyl schopen použít útok hrubou silou. Například při nastavení složitosti odvození na 200 ms, které při přihlášení do systému uživatel nepostřehne, bude útočník schopen na stejném hardware spočítat jen 5 hashů za sekundu. S rostoucím výkonem hardwaru v čase, lze tuto složitost navíc zvyšovat (například každých několik let).

4.3.2 Proces přihlášení

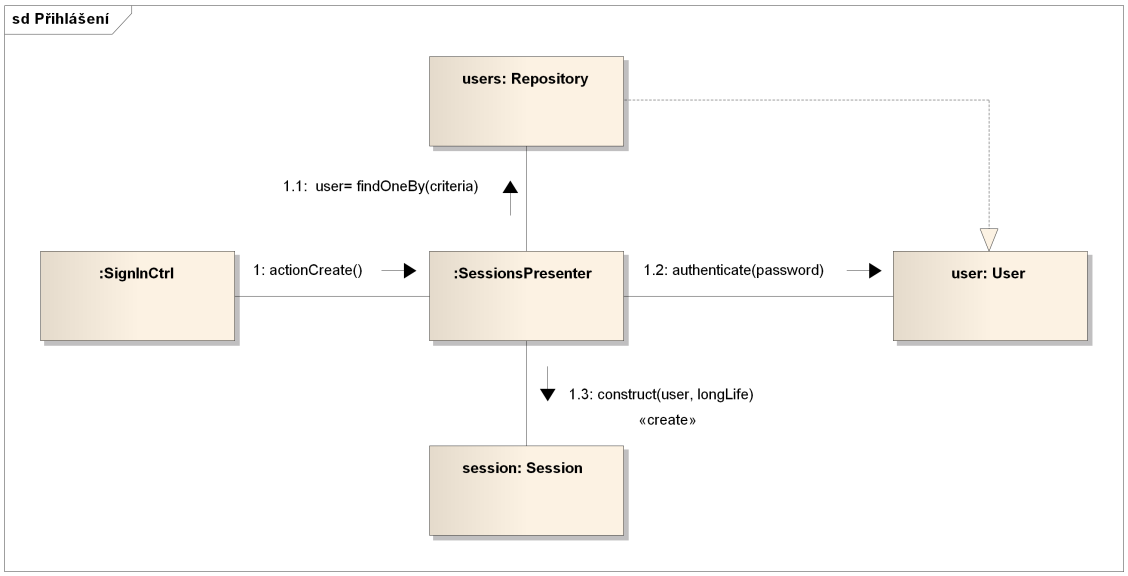
Přihlášení iniciuje klientská aplikace (dále jen klient), která si od uživatele vyžádá e-mail a heslo. Klient poté tyto údaje přes REST API odešle serverové aplikaci (dále jen server). Server pomocí e-mailové adresy získá z databáze entitu uživatele, která ověří správnost hesla. Pokud bylo heslo ověřeno, server vytvoří entitu sezení (session), jejíž cílem je identifikovat přihlášeného uživatele. Sezení je časově omezené – má nastaveno, kdy expiruje. Entita sezení při svém vytvoření vygeneruje náhodný token, který server odešle klientovi. Klient poté do všech odesílaných požadavků vkládá hlavičku `X-Session-Token` obsahující tento token. Pomocí entity sezení, jejíž token server přijal v hlavičce požadavku, ověří, zdali je sezení uživatele aktivní (neexpirovalo) a prodlouží jeho platnost. Server má k dispozici informaci o přihlášeném uživateli prostřednictvím entity sezení.

Celý tento proces zachycuje také diagram komunikace na obrázku 4.1.

4.3.3 Jednorázové tokeny

Pro autentizaci při provádění akcí, které jsou spouštěny mimo sezení, slouží jednorázové tokeny. Jedná se o entitu, která se váže na určitou akci a ke konkrétnímu uživateli. Token má časově omezenou platnost. Při svém vytvoření vygeneruje náhodný klíč, který slouží k potvrzení akce, pro kterou byl vytvořen.

Tokeny jsou uloženy v databázi, ze které jsou expirované tokeny průběžně odstraňovány, aby nedocházelo k jejich kumulaci. Navíc je pro čtení tokenů



Obrázek 4.1: Přihlášení uživatele

z databáze nastaven v Doctrine ORM filtr, který odfiltruje všechny neaktivní tokeny. Nemůže se tedy stát, že by programátor načetl entitu neplatného tokenu.

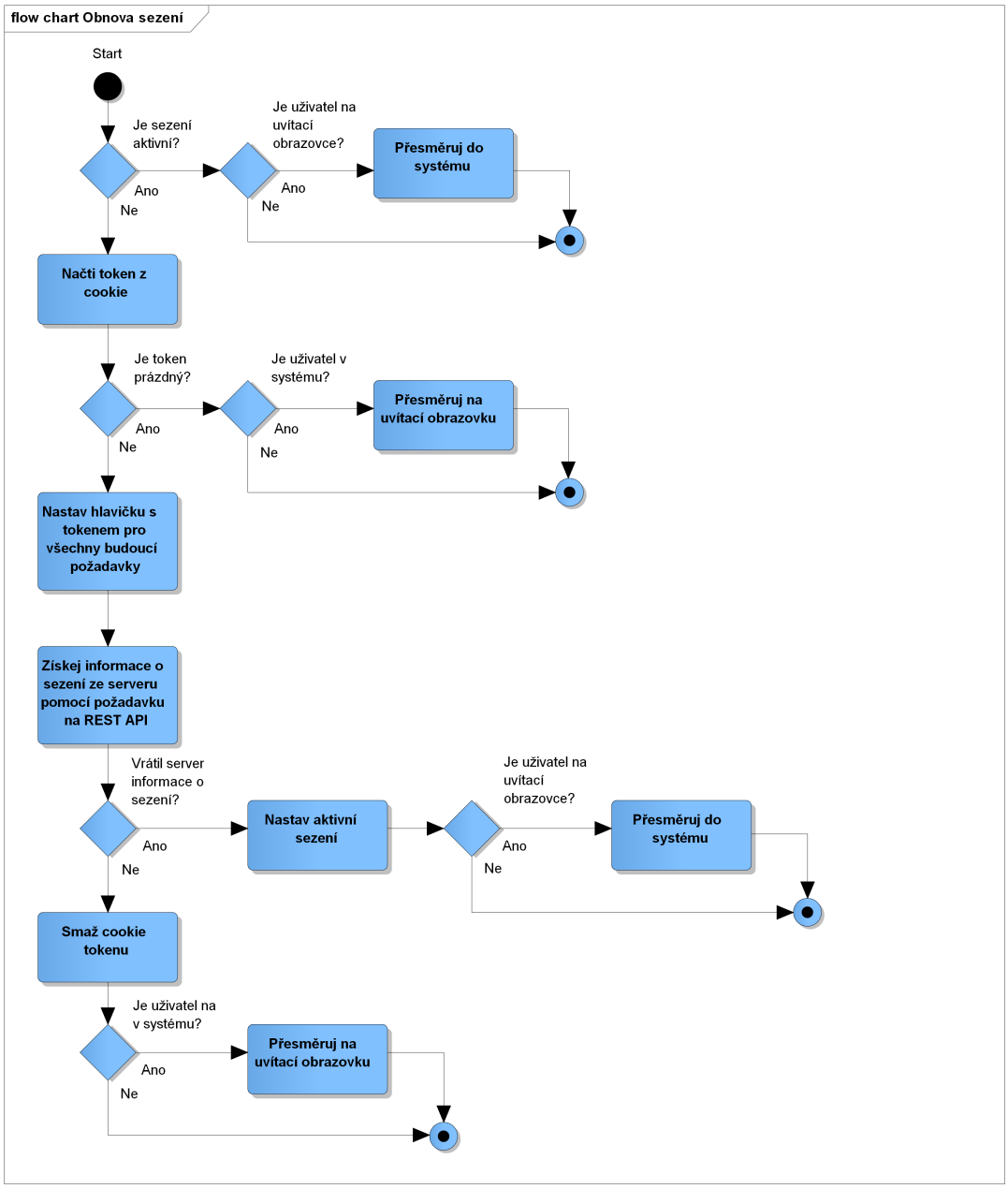
Při provádění akce musí být předán klíč tokenu, pomocí kterého se načte entita tokenu, a ověří se prováděná akce. Poté je možné provést akci, kterou nemohl spustit někdo jiný, protože klíč tokenu má jen oprávněná osoba.

Jednorázové tokeny jsou použity například pro nastavení hesla při prvním přihlášení do systému nebo při jeho ztrátě. Uživatel pomocí odkazu v e-mailu otevře URL s klíčem tokenu, na které se nachází stránka s formulářem pro nastavení hesla. Zvolené heslo je spolu s klíčem tokenu z URL odesláno na server, který tak může získat entitu uživatele, pomocí které nastaví heslo.

4.4 Obnovení sezení

Pokud by přihlášený uživatel v prohlížeči obnovil stránku, došlo by k novému spuštění klientské aplikace, která by tak ztratila informaci o tom, že byl uživatel přihlášen, a zobrazila mu přihlašovací formulář. Z tohoto důvodu klient po úspěšném přihlášení uloží token sezení do souboru cookie. Pokud tak přihlášený uživatel načte stránku, klient získá token sezení z cookie. Musí však nejdříve ověřit platnost tohoto sezení, a proto si vyžádá informace o sezení od serveru, který mu je v případě platnosti vrátí. Teprve na základě této komunikace se klient rozhodne, jakou stránku uživateli zobrazí.

Logiku klienta při obnovení sezení zachycuje vývojový diagram na obrázku 4.2.



Obrázek 4.2: Obnova sezení v klientské aplikaci

4.5 Stavy objednávky

Stav objednávky je vyčleněn do samostatné třídy `OrderState`. Tato třída obsahuje výčet možných stavů a povolené přechody mezi nimi. Prakticky se tak jedná o stavový automat.

4.6 Napojení na e-mailovou schránku

Napojení zpráv na e-mailovou schránku advokátní kanceláře je zajištěno pomocí protokolů SMTP a IMAP. Pokud uživatel odešle v aplikaci zprávu pro klienta, je mu tato zpráva doručena formou e-mailu. O odeslání e-mailu se stará server, který jej pomocí SMTP protokolu odešle skrze e-mailovou schránku advokátní kanceláře. Pokud uživatel ke zprávě přiloží soubory, jsou automaticky přiloženy k e-mailu jako příloha. Po úspěšném odeslání e-mailu je zpráva zaevidována v systému u patřičné objednávky.

Jakmile klient na zaslaný e-mail odpoví, dorazí tento e-mail do schránky advokátní kanceláře. Systém, při každém požadavku na výpis zpráv, stahuje ze schránky všechny e-maily pomocí protokolu IMAP. S e-maily stahuje i jejich přílohy, pro které pak v systému zakládá entity dokumentů. Aplikace podle e-mailové adresy odesílatele dohledá klienta v systému. Pokud má klient otevřenou pouze jednu objednávku, je tento e-mail i s přiloženými dokumenty u objednávky automaticky zaevidován jako zpráva od klienta. Pokud klient v systému neexistuje, nebo pokud má více otevřených objednávek (a systém tak nemůže jednoznačně přiřadit zprávu k objednávce), přesune e-mail ze složky příchozích zpráv do složky s názvem „Odmítnuto systémem“. Pro zpracované e-maily pak slouží složka s názvem „Přijato systémem“, kam aplikace přesouvá všechny e-maily, které byly úspěšně zpracovány.

Z důvodu stahování e-mailů do systému až když je to potřeba (on demand) načítá klientská část aplikace zprávy u objednávky dvěma požadavky. Nejdříve si vyžádá seznam všech zpráv u objednávky, které jsou uložené v databázi. Tyto e-maily uživateli vykreslí, uživatel se tak ke zprávám v systému dostane okamžitě. Poté si znovu vyžádá seznam zpráv, tentokrát i se stažením e-mailů ze schránky. Tato operace může trvat i několik sekund v závislosti na počtu zpráv, velikosti příloh a rychlosti připojení. Server stáhne e-mailové zprávy, uloží je do databáze a vrátí je klientské aplikaci. Klient poté dodatečně doplní stažené zprávy k již vykresleným. O průběhu stahování je uživatel informován pomocí GUI, pokud tedy čeká novou zprávu, vyčká si na její stažení, v opačném případě na dokončení stahování čekat nemusí.

4.7 Stahování dokumentů

Stahování dokumentů z informačního systému není triviální, jak se na první pohled může zdát. Stažení souboru nelze provést asynchronně. Klientská apli-

kace tedy musí přesměrovat prohlížeč na URL, na které dojde ke stažení souboru. V tomto případě však požadavek odesílá sám prohlížeč, a klient tak není schopen nastavit hlavičku s tokenem sezení. Z tohoto důvodu musí být autentizace uživatele provedena jiným způsobem. Rozhodl jsem se pro využití již implementovaných jednorázových tokenů, které umožňují provést akci uživatele, který je autentizovaný pomocí klíče. Pokud chce tedy uživatel stáhnout soubor, aplikace si nejdříve pomocí REST API vyžádá od serveru jednorázový token pro jeho stažení. Server vytvoří token, který uloží do databáze a jeho klíč odešle zpátky klientovi. Klient teprve nyní přesměruje prohlížeč na URL určenou pro stažení souboru, do které doplnil klíč tokenu. Server je na této URL schopen autentizovat uživatele pomocí tokenu v databázi, jehož klíč obdržel od klienta. Soubor byl tedy vyžádán přihlášeným uživatelem a server jej může odeslat.

4.8 Testování

Funkčnost aplikace jako celku a splnění požadavků na aplikaci je ověřeno za pomoci frameworku Behat. Behat je framework podporující programování řízené chováním (v angličtině behaviour-driven development) [41]. Požadavky na aplikaci se zapisují ve formě vět do souboru s příponou `.feature`. Požadavky se skládají ze scénářů. Jednotlivé scénáře se pak skládají z kroků, které mohou být trojího druhu:

- Given (Dáno) – cílem tohoto kroku je uvést systém do známého stavu, definuje předpoklady scénáře.
- When (Když) – tento krok popisuje akci, kterou provádí uživatel.
- Then (Tak) – účelem tohoto kroku je definovat výstup, který je v nějaké formě dostupný uživateli; neměl by sledovat vnitřní stavy aplikace, volání apod.

Více opakujících se kroků stejného typu lze spojovat pomocí spojek „and“ a „but“ tak, aby se jednotlivé kroky lépe četly. Ukázka scénáře popisující přihlášení a odhlášení uživatele v systému je vidět na výpisu 1.

Tyto scénáře jsou vykonávány v kontextech. Kontext je třída implementující rozhraní pro kontexty, která řídí běh scénáře a vykonává kroky v nich uvedené. Kroky jsou rozpoznány s využitím regulárních výrazů, které se zapisují pomocí anotace k metodám kontextové třídy, které je mají vykonat.

Pro testování celého systému, tj. jak klientské aplikace, tak i serverové, využívám Behatu s rozšířením Mink. Mink slouží k ovládání nebo emulaci prohlížeče pro testování webových aplikací [42]. Poskytuje kontext, který umožňuje spouštět kroky scénáře ovládající prohlížeč. Mink jako takový pouze abstrahuje od konkrétního použitého ovladače a poskytuje jednotné API. Jako ovla-

Výpis 1 Scénář přihlášení a odhlášení uživatele

```
Feature: Sign in/out
  In order to use the information system, user must be signed in.

  Scenario: Unknown email address
    Given I am on the welcome screen
    When I fill in "email" with "foo@bar.com"
    And I fill in "password" with "12345"
    And I press "signIn"
    Then I should see "Uživatel se zadaným e-mailem v systému neexistuje."

  Scenario: Wrong password
    Given I am on the welcome screen
    When I fill in "email" with "mark.mcdonald@example.net"
    And I fill in "password" with "ipsum"
    And I press "signIn"
    Then I should see "Zadáno chybné heslo k uživatelskému účtu."

  Scenario: Successful sing in
    Given I am on the welcome screen
    When I fill in "email" with "mark.mcdonald@example.net"
    And I fill in "password" with "lorem"
    And I press "signIn"
    Then I should see "Objednávky"

  Scenario: Sign out
    When I follow "signOut"
    Then I should see "Odhlášení proběhlo úspěšně."
```

dač lze použít jakýkoliv „headless“ prohlížeč (emulovaný prohlížeč bez grafického rozhraní běžící pouze v paměti) nebo skutečný prohlížeč řízený programem. Protože je potřeba testovat i klientskou aplikaci napsanou v Javascriptu, je potřeba použít skutečný prohlížeč. Emulované prohlížeče jej bohužel neumí spustit. Nejdříve jsem se rozhodl pro použití Sahi, jakožto ovladače pro prohlížeč.

Sahi je multiplatformní sada automatizačních nástrojů pro testování webových aplikací [43]. Je napsaná v Javě a Javascriptu. Sahi je k dispozici v open-source a proprietární variantě. Dokáže spustit prohlížeč, který poté ovládá pomocí javascriptových volání a simuluje tak chování skutečného uživatele.

S narůstajícím počtem scénářů jsem s ním však začal mít problémy. Sahi začalo po nějakém čase vždy některé požadavky na webový server chybně zpracovávat. Tento problém se projevoval jako zobrazování chybové stránky Sahi v prohlížeči a chybovým návratových kódem při asynchronním volání z klientské aplikace.

Z tohoto důvodu jsem se rozhodl nahradit Sahi nástrojem Selenium.

Selenium je stejně jako Sahi sada nástrojů pro automatizované testovaní webových aplikací [44]. Je také napsané v Javě a Javascriptu, avšak je vývíjeno

pouze jako open source. Z nástrojů, které Selenium nabízí, využívám pouze Selenium Standalone Server, který je určen k ovládání prohlížeče. Selenium není, jako v případě Sahi, potřeba instalovat, stačí pouze spustit stažený Java archiv.

S pomocí Behatu, jeho rozšíření Mink a nástroje Selenium jsem tak schopen spustit vytvořené scénáře a ověřit, zdali systém tyto scénáře splňuje. Tento způsob testování je kombinací systémového testování (testuje se celý systém a kooperace jeho částí) a akceptačního testování (testuje se výsledná funkčnost popsaná požadavky) [45]. Chyba na jakékoliv úrovni (jednotka, komponenta, systém) je tak s nejvyšší pravděpodobností těmito testy odhalena. Nevýhodou těchto testů je jejich časová a hardwarová náročnost. Je nutné spustit prohlížeč, interakce s ním je pomalá, a Selenium napsané v Javě spolu s prohlížečem vyžadují poměrně mnoho paměti, zvláště pokud testy spouštíme paralelně. Pokud je nutné testy spouštět na integračním serveru, musí být patřičně výkonný a je nutné počítat s delším průběhem testů, než při použití například jednotkových testů.

Testovací data (fixtures) vytvářím za pomoci Doctrine knihovny Data Fixtures. Tato data jsou definovaná vedle scénářů, které nad nimi provádí operace a vyhodnocují výstupy. V kontextu, ve kterém spouštím testy, vždy před začátkem scénáře nahraji testovací data do databáze a po jeho skončení jsou data zahozena.

Scénáře, které jsem pro potřeby otestování aplikace napsal, vychází z funkčních požadavků definovaných v kapitole 2 věnující se analýze.

Některé funkce bohužel nebylo možné jednoduše automatizovaně testovat. Například stahování dokumentů vyžaduje složitější interakci s prohlížečem, kterou Mink v kombinaci se Seleniem neumí. Tuto funkci by bylo možné otestovat pomocí headless prohlížeče, který umí zachytit odpověď serveru a porovnat ji s očekávanou odpovědí. Také komunikace s klientem pomocí e-mailů není testována. Simulování SMTP a IMAP připojení je příliš složité v porovnání s „ručním“ testováním. Tyto složitější funkce tedy musí testovat člověk.

Kapitola 5

Zhodnocení

Informační systém pro advokátní kancelář jsem úspěšně implementoval a otestoval. Systém splňuje požadavky na funkčnost definované v kapitole 2 zabývající se analýzou.

Dalším krokem by v případě reálné advokátní kanceláře bylo testovaní systému v pilotním provozu přímo zaměstnanci kanceláře. Připomínky a požadavky vzešlé z pilotního provozu by byly zapracovány a systém by poté byl připraven pro produkční nasazení.

Cílem této kapitoly je provést srovnání procesů vyřízení objednávky před a po nasazení informačního systému a zhodnotit přínosy výsledného systému z ekonomického pohledu.

5.1 Srovnání procesů

Porovnal jsem čtyři klíčové procesy probíhající při vyřizování objednávek advokáty.

5.1.1 Evidence objednávek

Evidence objednávek bez podpory informačního systému probíhala za pomoci sdílených dokumentů. Tyto dokumenty byly sdíleny v cloudovém úložišti a všichni zaměstnanci k nim měli přístup. Pro každý případ byl vytvořen nový dokument, do kterého byly advokátem vyplněny všechny potřebné údaje. Do tohoto úložiště se také nahrávaly soubory od klienta nutné pro vyřízení objednávky. Vyhledávání mezi těmito dokumenty bylo značně omezené, všechny dokumenty musely být správně zaevidovány, aby byly jednoduše dohledatelné. Advokát si musel v externím systému vést poznámky s aktuálně otevřenými objednávkami a jejich stavem.

Tento proces je s podporou informačního systému zefektivněn. Případy a vyplňování údajů k případu probíhá pomocí grafického rozhraní přímo v systému. Výhodou je také centrální evidence všech klientů v systému. Klienta

i konkrétní objednávku je možné vyhledat pomocí jednoho či více filtrů na výpisu klientů nebo objednávek. Každá objednávka se nachází v konkrétním stavu, mezi kterými lze přecházet podle nastavených pravidel. K objednávce je možné přiřadit osobu zodpovědnou za její vyřízení a jednoduše tak rozdělit zpracování více objednávek mezi jednotlivé advokáty. Každý advokát si tak může na začátku své pracovní doby nechat pomocí filtrů vypsat všechny objednávky, které je nutné zpracovat a ke kterým byl přiřazen. Toto řešení velmi zjednodušuje řízení zpracování objednávek.

5.1.2 Komunikace s klientem

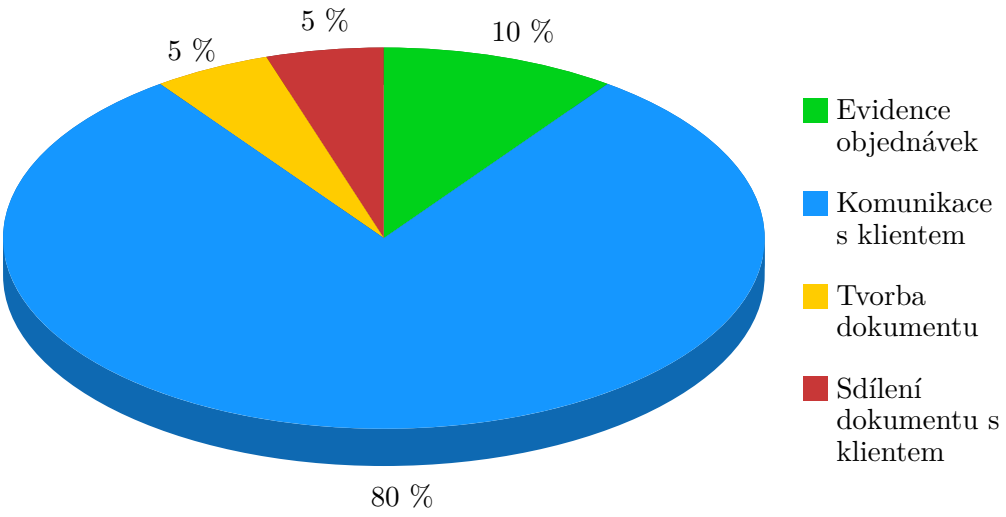
Komunikace s klientem probíhá i nadále pomocí e-mailů. Hlavním rozdílem při využití informačního systému je však jejich automatická evidence u objednávky. Advokát odpovídá klientovi na e-maily přímo z detailu případu. Před nasazením informačního systému bylo nutné e-maily odesílat z desktopové aplikace a texty v nich obsažené kopírovat do sdílených dokumentů objednávek. Tato evidence se nyní děje zcela automaticky. Advokát má na detailu objednávky přehled o proběhlé komunikaci a snadno se tak rychle vrátí zpět do kontextu případu, protože intervaly mezi jednotlivými odpověďmi mohou být v řádu několika dní. Další výhodou je také automatická změna stavu objednávky, pokud od klienta dorazí zpráva.

5.1.3 Tvorba dokumentů

Tvorba dokumentů bez podpory informačního systému probíhala ručním přepsáním údajů případu do předem připravené šablony. Tyto dokumenty se poté exportovaly do formátu PDF a zasílaly klientovi k podpisu. Tyto šablony jsou nyní implementovány v informačním systému a jejich naplnění daty se děje automaticky z položek objednávky a případu. Systém sám vygeneruje dokument ve formátu PDF a na advokátovi tak zůstává jen jejich kontrola před odesláním klientovi. Toto je proces, který byl nejvíce zefektivněn.

5.1.4 Sdílení dokumentů s klientem

Všechny klientovi zaslané a od klienta přijaté dokumenty byly součástí e-mailových zpráv. Dokumenty přiložené k e-mailům bylo nutné stáhnout a nahrát do cloudového úložiště advokátní kanceláře. S použitím implementovaného informačního systému budou nyní všechny dokumenty automaticky evidovány u patřičné objednávky a komunikace. Všechny dokumenty v e-mailech přijatých systémem jsou automaticky staženy a zaevidovány k případu. Pokud chce advokát odeslat klientovi nějaký dokument, nahraje jej do systému a poté přiloží k odesílané zprávě. Soubory je možné kdykoliv stáhnout.



Obrázek 5.1: Rozdělení dílčích procesů v procesu zpracování objednávky

5.2 Ekonomický pohled

Pro zjištění přibližného zefektivnění klíčových procesů, které jsou systémem podporovány, jsem třikrát zkušebně provedl každý z procesů. Bejdříve tak, jak probíhal v advokátní kanceláři bez informačního sytému a poté s implementovaným systémem. Tyto hodnoty jsem zprůměroval a zaokrouhlil (jedná se o hrubý odhad), nyní je použiji pro zjištění celkové časové úspory. Informační systém procesy v průměru urychlil, a to konkrétně o 15 % v případě evidence objednávek (založení nové objednávky, vyhledávání, agenda), o 10 % v případě komunikace s klientem, o 80 % v případě tvorby dokumentů a o 10 % v případě sdílení dokumentů s klientem.

Poměrové rozdělení těchto dílčích procesů v celém procesu zpracování objednávky advokátem jsem pouze odhadl, jelikož k reálným datům nemám přístup a nelze je ani nijak odvodit či změřit. Toto rozdělení jsem zanesl do grafu na obrázku 5.1.

Pokud by advokáti věnovali vyřizování objednávek 10 hodin týdně vedle jejich hlavní činnosti, lze dopočítat roční časovou úsporu. Celkový čas strávený ročně vyřizováním objednávek jednoho advokáta jsem spočítal pomocí vzorce $c = 10 \cdot 52 = 520\,\mathrm{h}$, kde 52 je průměrný počet týdnu v roce.

Původní roční čas strávený nad dílčím procesem jsem spočítal pomocí vzorce $p = z \cdot c$, kde z je procentuální zastoupení tohoto procesu při vyřizování objednávek z grafu na obrázku 5.1.

Roční čas strávený nad dílčím procesem při použití informačního sytému jsem spočítal pomocí vzorce $p' = (100\,\% - e) \cdot p$, kde e je úspora procesu v procentech.

Tabulka 5.1: Čas strávený nad procesy

Proces	**Původní čas [h]**	**Čas při použití IS [h]**	**Rozdíl [h]**
Evidence objednávek	52	44	8
Komunikace s klientem	416	374	42
Tvorba dokumentů	26	5	21
Sdílení dokumentů s klientem	26	23	3

Výsledné hodnoty vzorců pro jednotlivé dílčí procesy uvádím v tabulce 5.1. Z tabulky je patrné, že celková roční úspora času vyřizování objednávek je s pomocí informačního systému 74 hodin, což je o 14,2 % méně. Tato hodnota se může zdát malá, avšak je zapříčiněna tím, že advokát nejvíce svého času tráví komunikací s klientem. Ta probíhá stejným způsobem v desktopovém e-mailovém klientovi i informačním systému, rozdíl je pouze v automatické evidenci proběhlé komunikace v případě informačního systému. Další optimalizace tohoto procesu již není jednoduše proveditelná.

Průměrný hrubá mzda advokáta v České republice je 59 976 Kč měsíčně [46]. Při standardní pracovní době 40 hodin týdně stráví jednu čtvrtinu svého času vyřizováním objednávek. Na tuto činnost připadá tedy 14 994 Kč měsíčně. Pokud v advokátní kanceláři pracují čtyři advokáti, nasazením informačního systému ročně ušetří $12 \cdot 4 \cdot 14\,994 \cdot (100\,\% - 14,2\,\%) \approx 102\,200\,\text{Kč}$.

5.3 Další kroky

Systém je v této fázi připraven pro další rozšíření. Před započetím dalšího rozvoje systému by bylo vhodné zvážit provedení refaktorování současného kódu. Tato procedura by vyčistila a zjednodušila kód, který by tak byl lépe připraven pro další rozšíření. Například v aplikaci klienta je možné vyčlenit tabulkové výpisy s filtrací do vlastní komponenty nebo kontroleru.

Po pilotním testování systému je možné naimplementovat více případů (které se skládají z datových položek) a šablon právních dokumentů, do kterých se tyto položky budou propisovat. Pokud by vedení advokátní kanceláře projevilo také zájem o implementaci mobilní aplikace, je možné ji pomocí REST API napojit na současnou serverovou aplikaci.

V další etapě by mohlo být implementováno rozšíření současného API serveru pro vytvoření objednávky z vně systému. Díky tomu by bylo možné vytvořit novou objednávku skrze webovou stránku advokátní kanceláře za pomoci objednávkového formuláře. Tato funkce by měla zvýšit efektivitu procesu evidence objednávek, které tak ve většině případů nebude nutné zakládat ručně.

Závěr

Cílem této práce bylo vytvořit fungující informační systém využívající klient-server architektury, který podpoří probíhající procesy v advokátní kanceláři. Tento cíl se podařilo úspěšně naplnit, implementovaný informační systém zvyšuje efektivitu zaměstnanců a nepřímo tak snižuje náklady kanceláře. Vzniklý informační systém jsem zhodnotil z ekonomicko-manažerského pohledu.

Zjistil jsem, že i relativně jednoduchý informační systém, který podporuje klíčové procesy, dokáže výrazným způsobem ušetřit finanční prostředky. V případě komplikovanějších procesů by snížení nákladů mělo být mnohem výraznější. Důležité je správně tyto procesy identifikovat a popsat. Neméně důležité je mít k dispozici metriku, se kterou bude možné po dokončení systému analyzovat řešení a vyhodnotit skutečné přínosy jeho nasazení.

Správně zvolená architektura systému ulehčuje jeho další rozšiřitelnost. Dekompozice systému na menší oddělené části tuto rozšiřitelnost podporuje. Jednou z možností, jak dekompozici informačního systému provést, je rozdělit jej na klientskou a serverovou část, čímž dojde k oddělení prezentační a doménové logiky. Je ovšem důležité vytvořit pevné rozhraní pro komunikaci těchto částí, které omezí jejich provázanost.

Díky tomuto přístupu se údržba kódu zjednoduší. Výhody toho přístupu jsem si prakticky ověřil v průběhu implementace řešení.

Aplikaci klienta je možné v případě webového informačního systému vytvořit podle principů RIA a SPA, čímž dojde ke zvýšení uživatelského komfortu. Takto realizovaný informační systém se pak velmi podobá desktopové aplikaci. Navíc je přístupný odkudkoliv, pro běh vyžaduje pouze webový prohlížeč a internetové připojení.

Literatura

[1] Bernson, A.: *Client/Server Architecture*. New York: McGraw-Hill, druhé vydání, 1996, ISBN 0-07-005664-1.

[2] Stair, R.; Reynolds, G.: *Principles of Information Systems*. Cengage Learning, dvanácté vydání, 2015, ISBN 9781305482210.

[3] Doseděl, T.: *Počítačová bezpečnost a ochrana dat*. Computer Press, 2004, ISBN 9788025101063.

[4] Tang, X.; Xu, J.; Chanson, S. T.: *Web Content Delivery*. Springer Science & Business Media, 2006, ISBN 9780387277271.

[5] The Computer Language Company Inc.: *PC Magazine Encyclopedia: Desktop application [online]*. [cit. 2015-04-21]. Dostupné z: `http://www.pcmag.com/encyclopedia/term/41158/desktop-application`

[6] BBC: *BBC Webwise: What are mobile apps? [online]*. [cit. 2015-04-21]. Dostupné z: `http://www.bbc.co.uk/webwise/guides/mobile-applications`

[7] Nations, D.: *What is a Web Application? [online]*. [cit. 2015-04-21]. Dostupné z: `http://webtrends.about.com/od/webapplications/a/web_application.htm`

[8] Tilp, D.: *Responzivní a adaptivní design [online]*. 2013-11-07, [cit. 2015-04-21]. Dostupné z: `http://www.agile-ict.com/cs/blog/responzivni-a-adaptivni-design`

[9] McWherter, J.; Gowell, S.: *Professional Mobile Application Development*. ITPro collection, New York: John Wiley & Sons, 2012, ISBN 9781118240687.

[10] Masse, M.: *REST API Design Rulebook*. O'Reilly Media, Inc., 2011, ISBN 9781449319908.

[11] Daigneau, R.: *Service Design Patterns*. New Jersey: Addison-Wesley, 2012, ISBN 9780321544209.

[12] Roebuck, K.: *Rich Internet Application (RIA): High-impact Strategies*. Brisbane: Emereo Publishing, 2012, ISBN 9781743046876.

[13] Google Chrome Developers: *Chromium Blog: The Final Countdown for NPAPI [online]*. 2014, [cit. 2015-04-21]. Dostupné z: `http://blog.chromium.org/2014/11/the-final-countdown-for-npapi.html`

[14] Mikowski, M.; Powell, J.: *Single Page Web Applications: Javascript end-to-end*. New York: Manning Publications, 2014, ISBN 9781617290756.

[15] Ecma International, Geneva: *Standard ECMA-404: The JSON Data Interchange Format [online]*. První vydání, 2013. Dostupné z: `http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf`

[16] Galiegue, F.; Court, G.; Zyp, K.: *JSON Schema [online]*. 2013-01-30, [cit. 2015-04-22]. Dostupné z: `http://json-schema.org/latest/json-schema-core.html`

[17] W3C: *Extensible Markup Language (XML) 1.0 (Fifth Edition) [online]*. 2008, [cit. 2015-03-11]. Dostupné z: `http://www.w3.org/TR/REC-xml/`

[18] Laurent, S.; Johnston, J.; Dumbill, E.: *Programming Web Services with XML-RPC*. O'Reilly Media, Inc, 2001, ISBN 9780596001193.

[19] Oracle Corporation: *HTTPS Client Authentication [online]*. 2010, [cit. 2015-04-23]. Dostupné z: `https://docs.oracle.com/cd/E19226-01/820-7627/bncbs/index.html`

[20] Morley, M.: *JSON-RPC 2.0 Specification [online]*. 2013-01-04, [cit. 2015-05-05]. Dostupné z: `http://www.jsonrpc.org/specification`

[21] Fielding, R.: *Architectural Styles and the Design of Network-based Software Architectures*. Disertační práce, University of California, Irvine, 2000. Dostupné z: `http://jpkc.fudan.edu.cn/picture/article/216/35/4b/22598d594e3d93239700ce79bce1/7ed3ec2a-03c2-49cb-8bf8-5a90ea42f523.pdf`

[22] Thijssen, J.: *What is HATEOAS and why is it important? [online]*. [cit. 2015-04-23]. Dostupné z: `http://restcookbook.com/Basics/hateoas/`

[23] Mlejnek, J.: *Modelování obchodních procesů [přednáška]*. Praha: ČVUT, letní semestr 2014.

[24] Mlejnek, J.: *Analýza problémové domény [přednáška]*. Praha: ČVUT, letní semestr 2014.

[25] Google: *AngularJS – Superheroic JavaScript MVW Framework [online].* [cit. 2015-04-27]. Dostupné z: https://angularjs.org

[26] Gechev, M.: *AngularJS in Patterns [online].* 2014-05-08, [cit. 2015-04-27]. Dostupné z: http://blog.mgechev.com/2014/05/08/angularjs-in-patterns-part-1-overview-of-angularjs/

[27] Green, B.; Seshadri, S.: *AngularJS.* O'Reilly Media, Inc., 2013, ISBN 9781449344856.

[28] Menard, D.: *Instant AngularJS Starter.* Packt Publishing, 2013, ISBN 9781782166764.

[29] Google: *AngularJS: Developer Guide [online].* [cit. 2015-04-27]. Dostupné z: https://docs.angularjs.org/guide/introduction

[30] Burke, J.: *RequireJS [online].* [cit. 2015-04-27]. Dostupné z: http://requirejs.org/

[31] H5BP: *HTML5 Boilerplate: The web's most popular front-end template [online].* [cit. 2015-04-27]. Dostupné z: https://html5boilerplate.com/

[32] Sharp, R.: *What is Polyfill? [online].* 2010-10-08, [cit. 2015-04-27]. Dostupné z: https://remysharp.com/2010/10/08/what-is-a-polyfill

[33] Bootstrap organization: *Bootstrap, a sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development. [online].* [cit. 2015-04-27]. Dostupné z: http://getbootstrap.com/

[34] AngularUI Team: *Angular directives for Bootstrap [online].* [cit. 2015-04-27]. Dostupné z: https://angular-ui.github.io/bootstrap/

[35] Nette foundation: *Rychlý a pohodlný vývoj webových aplikací v PHP [online].* [cit. 2015-04-27]. Dostupné z: http://nette.org/

[36] Back, I.: *mPDF [online].* [cit. 2015-05-10]. Dostupné z: http://www.mpdf1.com/mpdf/index.php

[37] Doctrine Team: *Doctrine 2 ORM: Getting Started with Doctrine [online].* [cit. 2015-05-05]. Dostupné z: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/tutorials/getting-started.html

[38] The PHP Group: *PHP: password_hash [online].* [cit. 2015-05-02]. Dostupné z: https://php.net/manual/en/function.password-hash.php

[39] Provos, N.; Mazieres, D.: *A Future-Adaptable Password Scheme [online].* 1999-04-28, [cit. 2015-05-02]. Dostupné z: https://www.usenix.org/legacy/events/usenix99/provos/provos_html/node1.html

[40] Lockhart, J.: *Modern PHP: New Features and Good Practices*. O'Reilly Media, Inc., 2015, ISBN 9781491905180.

[41] Kudryashov, K.: *Behat Documentation [online]*. [cit. 2015-05-02]. Dostupné z: `http://docs.behat.org/en/v2.5/`

[42] Kudryashov, K.: *Mink 1.6 documetation [online]*. [cit. 2015-05-02]. Dostupné z: `http://mink.behat.org/en/latest/`

[43] Sahi Pro: *Automation Testing Tool For Web Applications [online]*. [cit. 2015-05-02]. Dostupné z: `http://sahipro.com/`

[44] Selenium Committers: *Selenium – Web Browser Automation [online]*. [cit. 2015-05-10]. Dostupné z: `http://www.seleniumhq.org/`

[45] Krátký, T.: *Software testing [přednáška]*. Praha: ČVUT, zimní semestr 2014.

[46] LMC, s.r.o.: *Průměrný plat – Advokát [online]*. [cit. 2015-05-04]. Dostupné z: `http://www.platy.cz/platy/pravo-a-legislativa/advokat`

Příloha A

Seznam použitých zkratek

API Application Programming Interface

CRUD Create, Read, Update, Delete

CSRF Cross-site Request Forgery

CSS Cascading Style Sheets

DOM Document Object Model

DTD Document Type Definition

ECMA European Computer Manufacturers Association

GUI Graphical User Interface

HATEOAS Hypermedia as the Engine of Application State

HTML Hypetext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IMAP Internet Message Access Protocol

IS Informační systém

ISO International Organization for Standardization

JSON JavaScript Object Notation

MIT Massachusetts Institute of Technology

MVC Model-view-controller

ORM Object-relational Mapping

PDF Portable Document Format

PHP PHP: Hypertext Preprocessor

REST Representational State Transfer

RIA Rich Internet Application

RPC Remote Procedure Call

SMTP Simple Mail Transfer Protocol

SOAP Simple Object Access protocol

SPA Single-page Application

SQL Structured Query Language

TCP/IP Transmission Control Protocol/Internet Protocol

UI User Interface

UML Unified Modeling Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

XML Extensible Markup Language

XSS Cross-site scripting

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src` ..... zdrojové soubory
  - `impl` ..... zdrojové kódy implementace
    - `readme.md` ..... pokyny pro spuštění
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF