VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

EVIDENČNÍ SYSTÉM SE ŠABLONAMI

REGISTRATION SYSTEM WITH TEMPLATES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **PAVEL ŠESTÁK**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Šesták Pavel**
Program: Informační technologie
Název: **Evidenční systém se šablonami**
Registration System with Templates
Kategorie: Informační systémy
Zadání:

1. Seznamte se se současnými technologiemi pro tvorbu webových aplikaci klient-server.
2. Prostudujte požadavky na systém pro evidenci různých typů položek na základě předdefinovaných šablon.
3. Po dohodě s vedoucím navrhněte architekturu evidenčního systému s důrazem na další rozšiřitelnost. Řešte rovněž možné kolize při souběžné editaci položek více uživateli.
4. Implementujte navržený systém pomocí vhodných technologií.
5. Proveďte testování vytvořeného systému.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Gutmans, A., Rethans, D., Bakken, S.: Mistrovství v PHP 5, Computer Press, 2012
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/
Vedoucí práce: **Burget Radek, doc. Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 11. října 2021

Abstrakt

Cílem této práce byla tvorba webového informačního systému, který slouží pro evidenci materiálů, dokumentů a generování výstupních dokumentů pro zákazníka. Aplikace byla rozdělena na klienta a server, kde server byl implementován pomocí C# ASP.NET Core 3.1 a klientská část pomocí frameworku React ve spojení s knihovnou Redux. Komunikace mezi klienty byla realizována pomocí websocketu knihovnou signalR. Serverová část aplikace byla následně testována automatickými testy. Klientská část byla testována podle diagramu případu užití. Při implementaci byl kladen důraz na využití generických tříd pro zjednodušení testovatelnosti a snazší možnosti rozšiřitelnosti celého systému. Informační systém je nasazen na firemním serveru v rámci IIS a testován na reálných uživatelích.

Abstract

The aim of this work was to create a web information system, which is used to record materials, documents and generate output documents for the customer. The application was divided into a client part and a server part. The server part was implemented using C# ASP.NET Core 3.1 and the client part using the React framework in conjunction with the Redux library. Communication between clients was realized using a websock library signalR. The server part of the application was subsequently tested by automatic tests. The client part was tested according to the use case diagram. During the implementation, emphasis was placed on the use of generic classes to simplify testability and make it easier to extend the entire system. The information system is deployed on a corporate server within IIS and tested on real users.

Klíčová slova

informační systém, webová aplikace, C#, ASP.NET, React, Redux, websocket, signalR, IIS, návrhové vzory, architektura informačního systému, generické třídy

Keywords

Information system, web application, C#. ASP.NET, React, Redux, websocket, signalR, IIS, design patterns, information systems architekture, generic classes

Citace

ŠESTÁK, Pavel. *Evidenční systém se šablonami.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

Evidenční systém se šablonami

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc.Ing. Radka Burgeta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

. . . .
Pavel Šesták
9. května 2022

Poděkování

Tímto bych chtěl poděkovat svému vedoucímu práce doc. Ing. Radku Burgetovi, Ph.D., za zastřešení dané práce a cenné rady v rámci psaní a strukturování této práce. Dále bych chtěl poděkovat Ing. Jitce Čapkové, Ph.D., MBA, a Mgr. Marku Hajnovi z Vojenského technického ústavu za námět pro tvorbu informačního systému formou bakalářské práce a pomoc při technické specifikaci požadavků na informační systém.

Obsah

1 Úvod **4**

2 Teoretický úvod **5**
2.1 Informační systém . . . 5
2.1.1 Data . . . 5
2.1.2 Informace . . . 5
2.1.3 Systém . . . 5
2.2 Webové aplikace . . . 6
2.2.1 Hypertext Transfer Protokol . . . 6
2.2.2 REST API . . . 7
2.2.3 Dostupné technologie . . . 8

3 Návrh systému **10**
3.1 Klíčové vlastnosti systému . . . 11
3.2 Modelování systému . . . 11
3.2.1 UML Entity relationship diagram . . . 12
3.2.2 UML diagram případů užití . . . 13
3.3 Návrh serverové části . . . 13
3.3.1 Architektura . . . 13
3.4 Návrh frontendu . . . 16
3.4.1 Architektura . . . 16

4 Implementace **19**
4.1 Implementace serverové části . . . 19
4.1.1 Autentizace . . . 19
4.1.2 Injektáž závislostí . . . 19
4.1.3 Data Access Layer . . . 20
4.1.4 Business Layer . . . 22
4.1.5 Presentation Layer . . . 24
4.2 Implementace frontendu . . . 26
4.2.1 Komunikace se serverovou částí aplikace . . . 26
4.2.2 Komunikace mezi jednotlivými klienty pomocí websocketu . . . 27
4.2.3 Centrální stav aplikace pomocí knihovny Redux . . . 27
4.2.4 Služby . . . 28
4.2.5 Utils . . . 28
4.2.6 Renderovací knihovny . . . 29
4.2.7 Komponenty . . . 30
4.2.8 Stránky . . . 30

4.2.9 Směrování v rámci single page application 35

5 Testování aplikace 36

5.1 Typy testů 36

5.2 Testování serverové části 36

5.2.1 Struktura testů 37

5.2.2 Integrační testy pro Data access layer 37

5.2.3 Unit testy pro Business layer 37

5.2.4 Integrační testy pro Business layer 38

5.2.5 Systémové testy serverové části aplikace 38

5.3 Testování frontendu 38

6 Nasazení aplikace 39

7 Závěr 40

Literatura 41

A Slovník použitých zkratek 43

Seznam obrázků

3.1 Diagram architektury informačního systému 11
3.2 UML Entity relationship diagram 12
3.3 UML diagram případů užití 13
3.4 Diagram architektury pro serverovou část systému 14
3.5 Diagram architektury pro klientskou část systému 16

4.1 Titulní strana generovaného dokumentu s informacemi o soupravě 25
4.2 Seznam předmětu v soupravě 25
4.3 Pohled na seznam předmětů v soupravě 31
4.4 Pohled na editační formulář předmětu 32
4.5 Pohled na detail předmětu s historií záznamu 33
4.6 Pohled pro nastavení parametrů generování seznamu předmětů 34
4.7 Pohled pro prezentaci kritických chyb 35

Kapitola 1

Úvod

Bakalářská práce pojednává o tvorbě webového informačního systému, který umožňuje spravovat a uchovávat kompletní informace o dodávkách a modernizacích techniky a systémů. Ke každému záznamu umožňuje uchovávat libovolné dokumenty a generovat výstupní dokumenty pro předání zákazníkovi.

Pracovní procesy se postupem času stávají složitější a složitější. I když nám může připadat, že v dnešní době již každá firma využívá pokročilé metody pro zjednodušení práce, opak je pravdou a často se můžeme setkat se starými systémy, které nemají vhodně navržené uživatelské rozhraní, nebo dokonce na danou práci žádný takový systém neexistuje a zaměstnanci si musejí vypomáhat různými vlastními metodami.

Toto zadání jsem si vybral, jelikož mě lákala zkušenost na rozsáhlejším projektu, na kterém už je nutné rozmyslet návrh a rozvržení zdrojových kódů tak, aby se kód nestal nepřehledným a byl snadno udržitelný. K tomuto účelu jsem začal studovat architektury systémů, návrhové vzory, vhodné využití objektového programování a jako nejzajímavější z této části mi přijdou generické třídy, jež jsou podobné šablonám v C++.

V této práci je popsán kompletní proces tvorby informačního systému od návrhu přes implementaci a testování až po nasazení systému. Jelikož je informační systém rozsahově již větší aplikace, velký důraz při implementaci je kladen na použití generických řešení, použití standardizovaných návrhových vzorů a celkovou minimalizaci množství kódu.

V rámci textu si postupně představíme webové informační systémy z teoretického hlediska a jaké jsou dostupné technologie pro implementaci webového informačního systému. Před samotnou implementací se podíváme na systém pomocí modelovacího jazyka UML z hlediska případů užití a na uchovávaná data a vazby mezi nimi. Po zvolení technologií pro implementaci a seznámení se s architekturou systému se text dále věnuje implementaci samotné, v níž jsou popsané jednotlivé části aplikace. Po implementaci se text věnuje implementaci automatizovaných testů pro serverovou část systému na více úrovních, a to od unit testů až po systémové testy. Klientská část aplikace je testována ručně za pomoci diagramu případů užití uživateli, kteří se systémem po nasazení budou reálně pracovat. Poslední část textu je věnována právě nasazení systému na server a veškerému nastavení, které je nutné pro běh systému.

Kapitola 2

Teoretický úvod

V této sekci bych rád čtenáře uvedl do problematiky informačních systémů a tvorby webových informačních systémů, které nyní dominují v tomto odvětví.

2.1 Informační systém

Podnikový informační systém je aplikace, jež umožňuje spravovat firemní data o projektech, z kterých tvoří informační a znalostní bázi. Dále pomáhá řídit podnikové procesy dané firmy.[23]

Informační systém je obrazem reálného systému na určité úrovni abstrakce, takže nám umožňuje sledovat stav reálného systému. Informační systém se skládá ze dvou separátních pojmů, které si dále představíme spolu s dalšími pojmy nutnými pro pochopení dané problematiky.

2.1.1 Data

Jedná se o způsob jak přenášet a uchovávat informace. Surová data jsou řetězec jedniček a nul. K tomu, abychom s těmito daty mohli pracovat, musíme vědět, jak je přečíst a interpretovat.

2.1.2 Informace

Informace jsou data, která je uživatel schopen přečíst a dále interpretovat. V rámci informačních technologií je informace definována jako interpretované kvantitativní vyjádření obsahu zprávy. V informatice je informace vyjádřena pomocí jednotky bit a ta nám slouží k rozhodnutí mezi dvěma hodnotami.

2.1.3 Systém

Systém v informatice je množina prvků a vazeb mezi nimi, jež jsou definované na nějakém nosiči. Nosiče se řadí do dvou kategorií. Jedná se o fyzické a konceptuální nosiče. Fyzické nosiče jsou stroje, materiál, lidské zdroje a mnoho dalšího. V rámci konceptuálního nosiče se bavíme pouze o informacích reprezentujících již existující fyzický systém.[8]

2.2 Webové aplikace

Jedná se o typ aplikace, kde se uživatelé připojují přes internet k webovému serveru. Uživatel komunikuje s aplikací pomocí webového prohlížeče, který slouží jako klient dané aplikace. Komunikace probíhá standardně pomocí protokolu HTTP. Řešení pomocí webové aplikace nelimituje uživatele od užívání různých operačních systémů a není nutné řešit instalace a aktualizace na klientských počítačích.[9]

2.2.1 Hypertext Transfer Protokol

Hypertext Transfer Protokol je webový protokol umožňující stahovat data z různých zdrojů. HTTP protokol je definován na aplikační vrstvě referenčního modelu ISO/OSI a je provozován standardně na TCP portu 80.[7]

HTTP protokol funguje v režimu požadavek - odpověď. Klient pošle požadavek na server. Server tento požadavek zpracuje a následně pošle odpověď. Odpověď obsahuje stavový kód, typ obsahu a obsah.

HTTP je textový protokol a první řádek specifikuje metodu, cestu, verzi protokolu a znak konce řádku. Následují hlavičky ve formátu: "název: hodnota". Každá tato hlavička je zakončena znakem konce řádku. Jakmile jsou všechny hlavičky specifikovány, tak se přidá další znak konce řádku, za nímž může být tělo dotazu, které nese data.

HTTP hlavičky umožňují oběma stranám komunikace přidat metadata ke každému požadavku či odpovědi. Každá hlavička obsahuje klíč, který je následovaný dvojtečkou. Po dvojtečce následuje hodnota. Hlavičky obvykle nesou informace o odesílateli, nastavení cachování, nápovědě jaký obsah je očekáván, specifikaci akceptované jazykové mutace a mnoho dalšího.

Rozdílné dotazovací metody slouží pro rozlišení požadavků v případě REST, a to bez nutnosti vlastní URL adresy pro každý požadavek. Některé metody se liší ve formě odesílání dat. Použití těchto metod bude v kapitole 2.2.2 REST API.

HTTP odpověď obsahuje stavový kód, který informuje volaného o stavu jeho požadavku. Stavové kódy pro HTTP jsou definované v intervalu <100, 600), kdy jsou kódy rozděleny do skupin a každé skupině náleží 100 různých po sobě jdoucích kódů. Skupiny stavových kódů jsou informační stavové kódy, kódy informující o úspěšném provedení požadavku, kódy pro přesměrování požadavku, kódy pro informování uživatele na jeho straně i kódy pro informování o chybě na straně serveru.

Ukázka komunikace

V ukázce níže je dotaz na index webové stránky s doménovým jménem info.cern.ch pomocí metody get. Server vrací návratový kód 304, který indikuje, že se od posledního dotazu nezměnil. Dále se můžeme například dočíst, že na serveru běží Apache.

Požadavek na server

```
GET / HTTP/1.1\r\n
Host: info.cern.ch\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
Upgrade-Insecure-Requests: 1\r\n
```

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/94.0.4606.71 Safari/537.36\r\n
Accept: text/html,application/xhtml+xml,
application/xml;q=0.9,image/avif,image/webp,
image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9\r\n
Referer: https://www.google.com/\r\n
Accept-Encoding: gzip, deflate\r\n
Accept-Language: cs-CZ,cs;q=0.9\r\n
If-None-Match: "286-4f1aadb3105c0"\r\n
If-Modified-Since: Wed, 05 Feb 2014 16:00:31 GMT\r\n
\r\n
```

Odpověď serveru

```
HTTP/1.1 304 Not Modified\r\n
Date: Sun, 03 Oct 2021 16:17:52 GMT\r\n
Server: Apache\r\n
Connection: close\r\n
ETag: "286-4f1aadb3105c0"\r\n
\r\n
```

[22]

2.2.2 REST API

REST API dnes výhradně pracuje nad HTTP protokolem. Pro každý poskytovaný zdroj API definuje vlastní endpoint. Například pro získání všech platných předmětů je to endpoint GET /api/components/getallvalid. Pro odlišení jednotlivých koncových bodů API můžeme použít jak cesty endpointů, tak i jednotlivé HTTP metody, které mají definovanou svoji sémantiku.

- GET - Získávání záznamů ze serveru.
- POST - Vytváření nových záznamů.
- PUT - Úprava celého záznamu (posílá se celý záznam).
- DELETE - Smazání záznamu ze serveru.
- PATCH - Částečná úprava záznamu.

K informování volajícího o stavu slouží návratové kódy, které jsme si více rozebrali v kapitole 2.2.1 Hypertext Transfer Protokol. V rámci naší implementace se nejčastěji setkáme se stavovým kódem 200 pro úspěšný dotaz a 201 pro úspěšné vytvoření. V případě chyby autentizace vrací serverová část kód 401. V případě, že daný endpoint na serverové části neexistuje, tak vrací chybu 404 a v případě, že dojde k chybě při vykonávání kódu na straně serveru, tak je vrácena chyba 500. Z prohlížeče je možné otestovat všechny GET koncové body, nicméně pro ostatní metody je nutné využít aplikaci Postman, umožňující poskládat libovolný požadavek, nebo Swagger, což je interaktivní dokumentace API dostupná na svém vlastním koncovém bodě serverové části aplikace.

2.2.3 Dostupné technologie

V této části si představíme jednotlivé technologie, které jsou dostupné pro implementaci serverové a klientské části aplikace. Většina frameworků lze využít více způsoby, a to buď pomocí architektury MVC, kde implementujeme celou aplikaci pomocí jednoho frameworku zastřešujícího více technologií, nebo serverový framework využijeme jen pro tvorbu API a pro klienta využijeme vlastní frontendový framework.

Přehled dostupných technologií pro serverovou část

Serverová část definuje logiku aplikace a dále řeší autentizaci, autorizaci uživatelů, mapuje endpointy na funkce, validuje data a komunikuje s databází systému.

Serverovou část aplikace je možné implementovat v mnoha jazycích s pomocí mnoha různých frameworků. Webové frameworky jsou velké balíky aplikací, které v sobě obsahují balíkovací systém, kontejner závislostí, přístup k databázi, testovací framework a mnoho dalšího. Největší rozdíly jsou v syntaxi a jazyku, nad kterým jsou postaveny, a množství přídavných balíků. Výhody využití frameworku jsou definovaná struktura projektu, bezpečnost, autentizace, autorizace a mnoho dalšího.

ASP.NET Core
ASP.NET Core je bezplatný open source webový framework v jazyce C#, který podporuje jak tvorbu API, tak tvorbu aplikací podle návrhového vzoru MVC. Návrhový vzor MVC rozděluje aplikaci na tři hlavní části: model, view a controller.[4] Model reprezentuje objekt s daty. View se stará o prezentaci dat uživateli. View má přístup k datům modelu a controller slouží pro spojení View s Modelem a řídí tok dat mezi nimi. Díky tomuto návrhovému vzoru jsme schopni oddělit logiku a prezentaci dat. V případě použití ASP.NET core jako REST API může controller místo pohledu vracet strukturovaná data ve formátu JSON. Jazyk C# je staticky typovaný, typově bezpečný a objektově orientovaný programovací jazyk neumožňující vícenásobnou dědičnost tříd. Vícenásobná dědičnost je zde částečně suplována možností vícenásobného dědění rozhraní.[15]

Django
Django je webový framework postavený na jazyce Python, který podporuje architekturu MVT.[12] Architektura MVT také rozděluje program do tří hlavních částí: model, view a template, nicméně zde mají pojmy jinou sémantiku. Model slouží jako vrstva pro práci s daty (DAL). View interaguje s modelem a slouží pro volání BL. Poslední část template slouží jako prezentační vrstva.[6] Pro tvorbu REST API je nutné využít verzi Django REST framework. Python je dynamicky typovaný, interpretovaný a více paradigmatický programovací jazyk. Obdobně jako C# obsahuje garbage collector.

Flask
Flask je modernější webový python framework. Oproti Djangu se jedná o micro webový framework, který je vhodnější pro menší a jednodušší projekty.[1]

Spring
Spring je open source robustní webový framework pro Javu. Java je v mnoha ohledech velmi podobná C#. Jedná se o jazyk přeložený do byte kódu, který pro spuštění vyžaduje interpret. [3]

Ruby on Rails
Ruby on Rails je webový framework napsaný v jazyce Ruby. Distribuovaný je pod licencí MIT a využívá návrhový vzor MVC. Ruby je interpretovaný jazyk pro obecné použití. Jazyk Ruby je dynamicky typovaný a využívá garbage collector a JIT kompilaci. [18]

Express.js
Express.js je open source webový framework pro Node.js. Node.js je open source běhové prostředí pro javascript umožňující spouštět javascript mimo webový prohlížeč. [13]

Laravel
Laravel je PHP webový framework založený na Symfony. Laravel je distribuovaný pod MIT licencí a také využívá návrhový vzor MVC. PHP je interpretovaný a dynamicky typovaný jazyk.

Přehled dostupných technologií pro frontend

Frontendová část aplikace slouží k interakci se serverovou částí aplikace a vhodnou prezentací dat uživateli s možností modifikace dat.

React
React je javascriptová knihovna vyvíjená firmou Facebook a slouží pouze pro tvorbu uživatelského rozhraní. React je distribuován pod licencí MIT. Slouží pro tvorbu jednostránkových aplikací, kde se každý pohled skládá z komponent, které umožňují komplexní pohledy dekomponovat na menší části. Každá komponenta si může uchovávat stavové proměnné, na jejichž základě se překresluje. Pro popis komponent se využívá formát JSX. JSX je rozšíření pro javascript, které umožňuje psát HTML s vlastními komponentami uvnitř javascriptového kódu. [17]

Angular
Angular je javascriptový framework, který využívá typescript. Typescript je programovací jazyk, vyvíjený společností Microsoft, a který rozšiřuje javascript o statické typování. Angular je vyvíjen firmou Google. Jedná se o kompletní řešení a má náročnější učící křivku. [10]

Svetle
Svetle na rozdíl od klasických reaktivních frameworků je kompilovaný. Provádí se překlad HTML šablon do kódu, který přímo manipuluje s DOM, což slouží k rychlejšímu běhu aplikace. Svetle je napsaný pomocí typescriptu a distribuovaný pod licencí MIT. [19]

Vue.js
Vue je další reaktivní framework, který na rozdíl od Reactu a Angularu není zaštítěn velkou korporací. Základní Vue slouží pouze pro zobrazování, pokročilou funkcionalitu je třeba přidat pomocí balíků. Ve srování s React a Angular se jedná o jednodušší framework. [20]

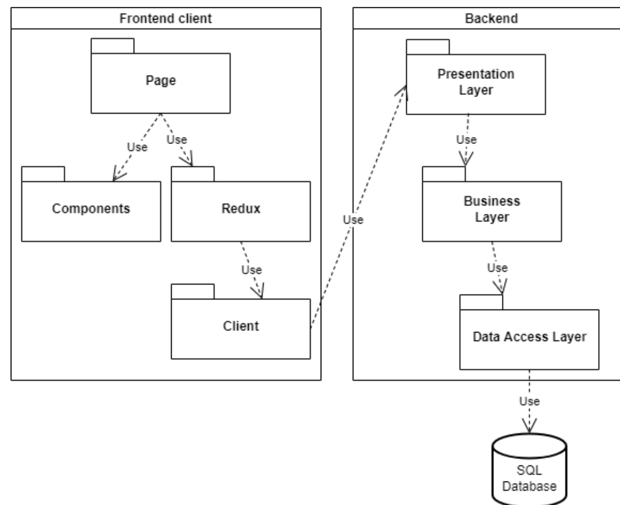
jQuery
Jedná se o starší javascriptovou knihovnu umožňující lépe oddělit javascript od HTML. Pomocí selektorů umožňuje modifikovat DOM a modifikovat obslužné rutiny událostí. JQuery dále obsahuje pomocné funkce pro javascript, AJAX pro asynchronní požadavky a další. [5]

Kapitola 3

Návrh systému

Cílem práce je návrh a následně implementace informačního systému firmy, která vyvíjí, vyrábí, nasazuje, renovuje a opravuje produkty. Při předání produktu je potřeba dodat soupis všech předmětů v soupravách a udělat seznam funkčních celků. Určitá skupina zaměstnanců plní seznamy předmětů, další lidé řeší servis, případně přiřazování funkčních celků k předmětům v soupravách. Každý předmět nebo systém může být v rámci servisu opraven, takže je nutné tento předmět vyhledat a přiložit k němu protokol o opravě. Se systémem bude současně pracovat více pracovníků, a proto je nutné řešit aktualizaci dat na klientech a souběžný přístup při editacích záznamů. Kvůli dosavadní zkušenosti všech zaměstnanců s touto prací v softwaru Microsoft Excel z balíku Office je hlavní požadavek, aby se výsledný informační systém dal ovládat podobným způsobem. Zaměstnanci doposud pracovali se svými lokálními verzemi dokumentů, ze kterých následně další zaměstnanci vytvářeli finální verze dokumentů. Jednotlivé dokumenty si verzovali pomocí kopií souborů a v případě jakékoliv chyby bylo nemožné dohledat u koho a proč k dané chybě došlo. Jako hlavní vyžadované funkce ze softwaru Microsoft Excel byly identifikované editace jednotlivých buněk a přeuspořádání jednotlivých záznamů. Jelikož tabulky bude používat mnoho lidí s různými preferencemi, tak je vyžadována možnost personifikace zobrazení tabulky. Další specifikace byla, že každá změna se bude uchovávat a bude možné zobrazit vývoj každého záznamu a autora změny, díky čemuž lze zobrazit konfiguraci jednotlivých souprav ke specifickému datu. Výsledné soupravy je nutné exportovat zpět do Excelu, nicméně neexistuje jednotný formát, ve kterém se sestavy předávají. Z tohoto důvodu je nutné mít možnost konfigurovat generování seznamů komponent a nastavit, jaké sloupce se budou generovat, v jakém pořadí, jaký budou mít název sloupce a podle rozhodnutí zda se jedná o předmět k likvidaci či předmět pro finální seznam.

Celá aplikace bude s uživatelem komunikovat pomocí javascriptového klienta. Data budou ukládána v SQL databázi, ke které bude klient přistupovat pomocí aplikačního rozhraní s byznys logikou. Diagram níže reprezentuje systém na nejvyšší úrovni abstrakce, přičemž jednotlivé bloky diagramu budeme rozebírat podrobněji dále v textu. Celá architektura je rozdělena na klientskou a serverovou část. V rámci serverové části je řešení rozděleno do tří vrstev, kde jedna řeší přístup k datům, další byznys logiku a poslední prezentaci dat. Prezentační vrstva nám implementuje REST rozhraní. Klientská část nemá architekturu tak přímočarou jako serverová, díky klientům stáhneme data ze serverové části, ty pomocí knihovny Redux, jež bude dále v textu podrobněji vysvětlena, uložíme do centrálního stavu aplikace a dále k nim budeme přistupovat z jednotlivých pohledů systému, které skládáme za pomoci dílčích komponent.



Obrázek 3.1: Diagram architektury informačního systému

3.1 Klíčové vlastnosti systému

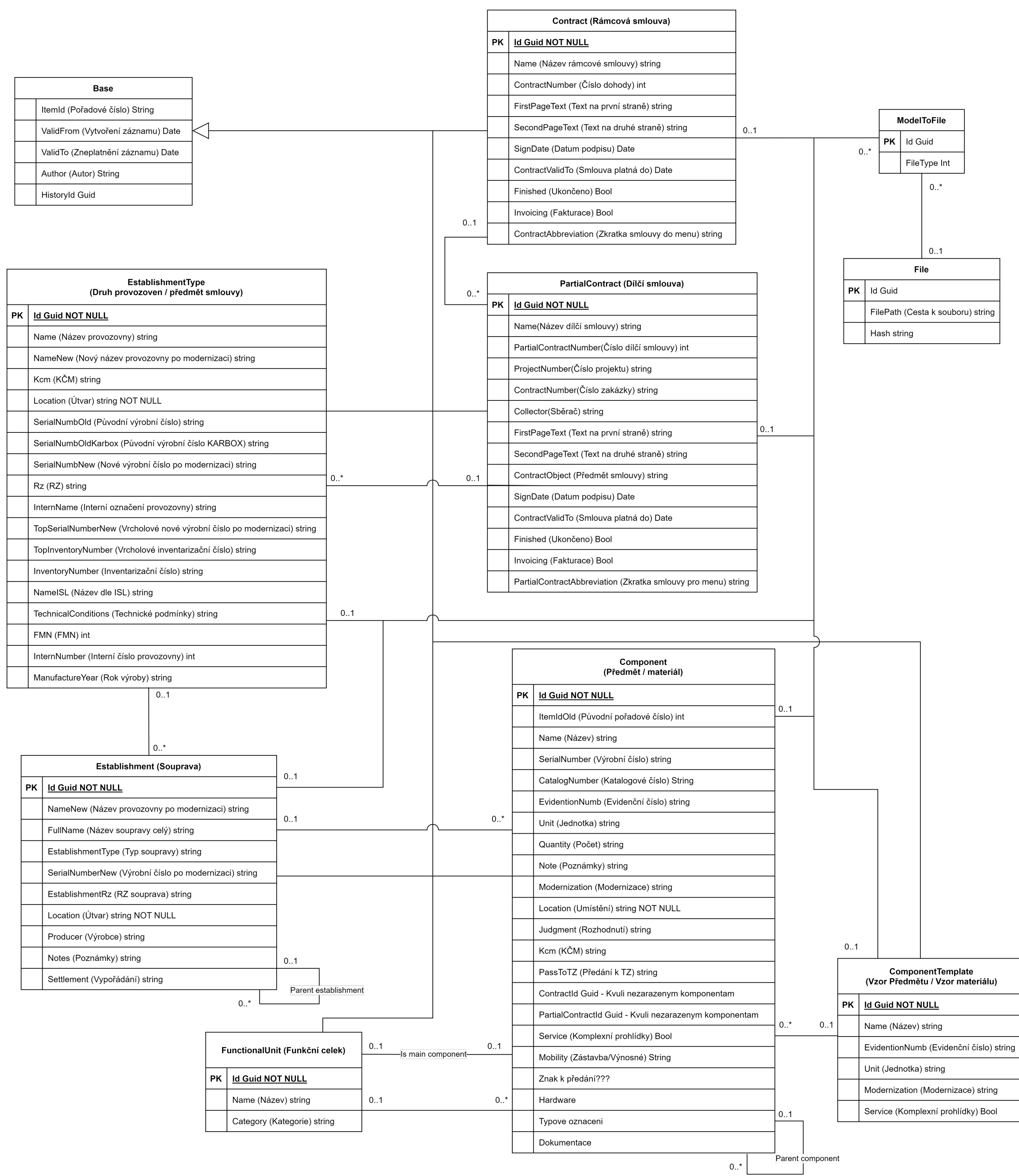
- Podpora dynamicky generovaných pohledů systému založených na předdefinovaných šablonách.
- Řešení souběžné editace záznamů více uživateli. Detekce úpravy daného záznamu jiným uživatelem pomocí websocket spojení. S informací o zneplatnění dojde i nový záznam, který lze dohledat podle historyId a uživatele přesměrovat na editační formulář s tímto záznamem. Rozpracované hodnoty jsou uloženy a nabídnuty u nově přesměrovaného záznamu.
- Dynamické generování výstupních souborů.
- Vedení historie jednotlivých úprav.
- Zabezpečení pomocí Windows Authenticate.

3.2 Modelování systému

V této sekci se podíváme na modelování informačního systému pomocí jazyka UML, a to konkrétně pomocí diagramu E-R a diagramu případů užití.

3.2.1 UML Entity relationship diagram

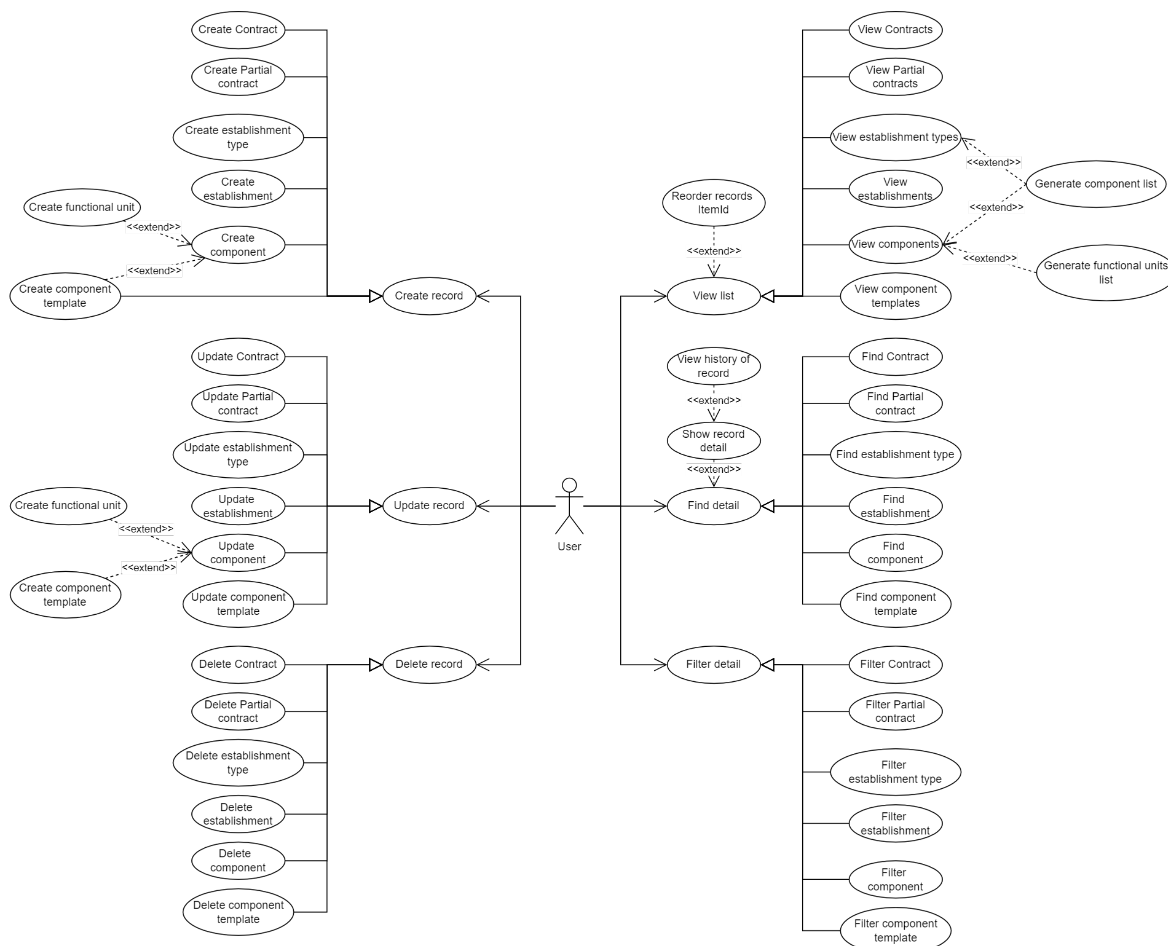
Entity relationship diagram slouží k modelování cílové domény, aby ji výsledný informační systém správně zobrazoval. Diagram cílovou doménu modeluje pomocí entit a jednotlivých vztahů mezi nimi. Na základě tohoto diagramu následně vznikne databáze a návrh entit v rámci serverové části. Pro zjednodušení jsou sloučené vztahy s bázovou entitou Base, která vytýká obecné vlastnosti všech entit. Další je vztah všech entit s ModelToFile, kde každá entita může mít libovolný počet záznamů o přílohách.



Obrázek 3.2: UML Entity relationship diagram

3.2.2 UML diagram případů užití

Diagram případů užití slouží pro analýzu, jaké operace bude uživatel skutečně se systémem vykonávat. Tento diagram je následně využit u testování výsledné aplikace.



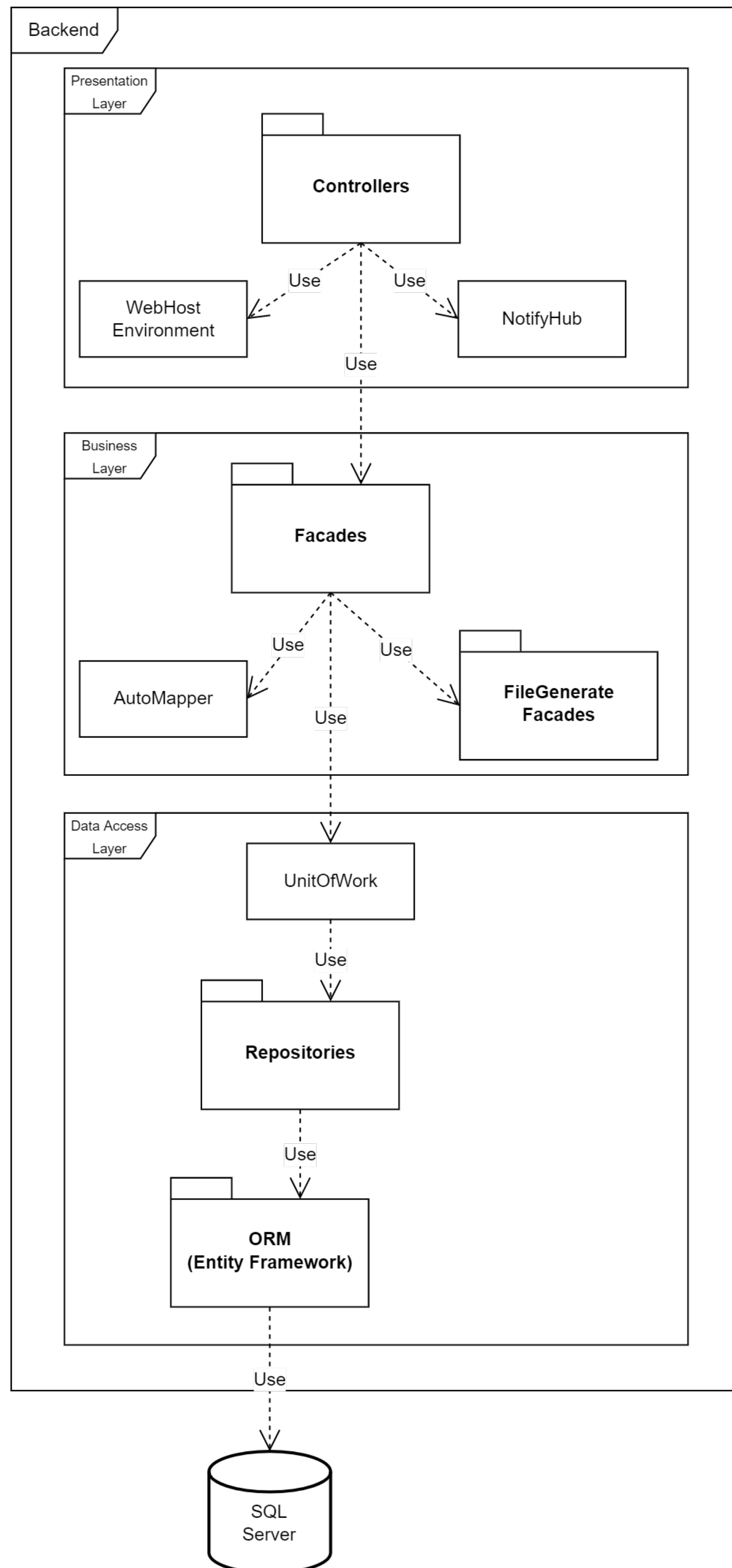
Obrázek 3.3: UML diagram případů užití

3.3 Návrh serverové části

Pro serverovou část byla zvolena technologie ASP.NET Core kvůli vlastnostem jazyka a osobním preferencím. Mezi hlavní aspekty výběru byla typová kontrola, kterou považuji u větších projektů za nutnou.

3.3.1 Architektura

V rámci serverové části byla zvolena standardní třívrstvá architektura, jež se skládá z vrstvy pro přístup k datům, vrstvy pro byznys logiku a prezentační vrstvy. Vícevrstvá architektura snižuje redundanci kódu a zjednodušuje celkovou testovatelnost a rozšiřitelnost aplikace.



Obrázek 3.4: Diagram architektury pro serverovou část systému

Databáze

Pro informační systém byla zvolena relační SQL databáze z důvodu pevné struktury dat. Návrh databáze vychází z ER diagramu, který byl představen v kapitole 3.2.1 UML Entity relationship diagram. Z důvodu požadavku na verzování jednotlivých záznamů byl přidán vyhledávací index v tabulkách na validitu, jelikož ve většině případů je potřeba oddělit platné záznamy od neplatných.

Data Access Layer

Tato vrstva slouží k přístupu k databázi. V rámci C# se využívá knihovna Entity Framework pro objektově relační mapování. Entity Framework nám poskytuje databázový kontext, s jehož pomocí pracujeme s databází. Repozitáře přistupují k databázovému kontextu a definují nám základní operace v systému, jako je například získání dat z databáze, úprava dat s ohledem na verzování, výpočet pořadových čísel a mnoho dalších. Jelikož náročnější operace v rámci byznys logiky budou vyžadovat práci i s více repozitáři v rámci jedné byznys transakce, vzniká abstraktní úroveň UnitOfWork, která nám umožní uložit změny do databáze až provedeme všechny požadované operace nad repozitáři, a tím ošetříme nechtěnou inkonzistenci databáze v případě jakékoliv chyby uprostřed transakce.

Business Layer

Tato vrstva slouží pro definování byznys operací. Fasády využívají UnitOfWork z vrstvy DAL pro přístup k databázi. Tato vrstva vrací záznamy z databáze namapované na entity, nicméně v rámci databáze ukládáme mnoho informací, které jsou vyžadovány pouze ve specifických pohledech. Z důvodu šetření času a datového toku mezi klientem a serverovou částí jsou zde entity mapovány na specifické modely, popřípadě jsou komplexnější modely skládány více dotazy nad UnitOfWork. Takto jsme schopni poslat jiná data pro tabulku, detailní pohled nebo pro dynamický výběrový box. V případě úpravy dat v databázi po dokončení transakce se provolá funkce Commit nad UnitOfWork a změny se zapíší do databáze. Pro mapování je využita knihovna AutoMapper, jež je distribuovaná pod MIT licencí. Separátně jsou oddělené fasády pro generování jednotlivých dokumentů z dat za pomoci knihovny ClosedXML.

Presentation Layer

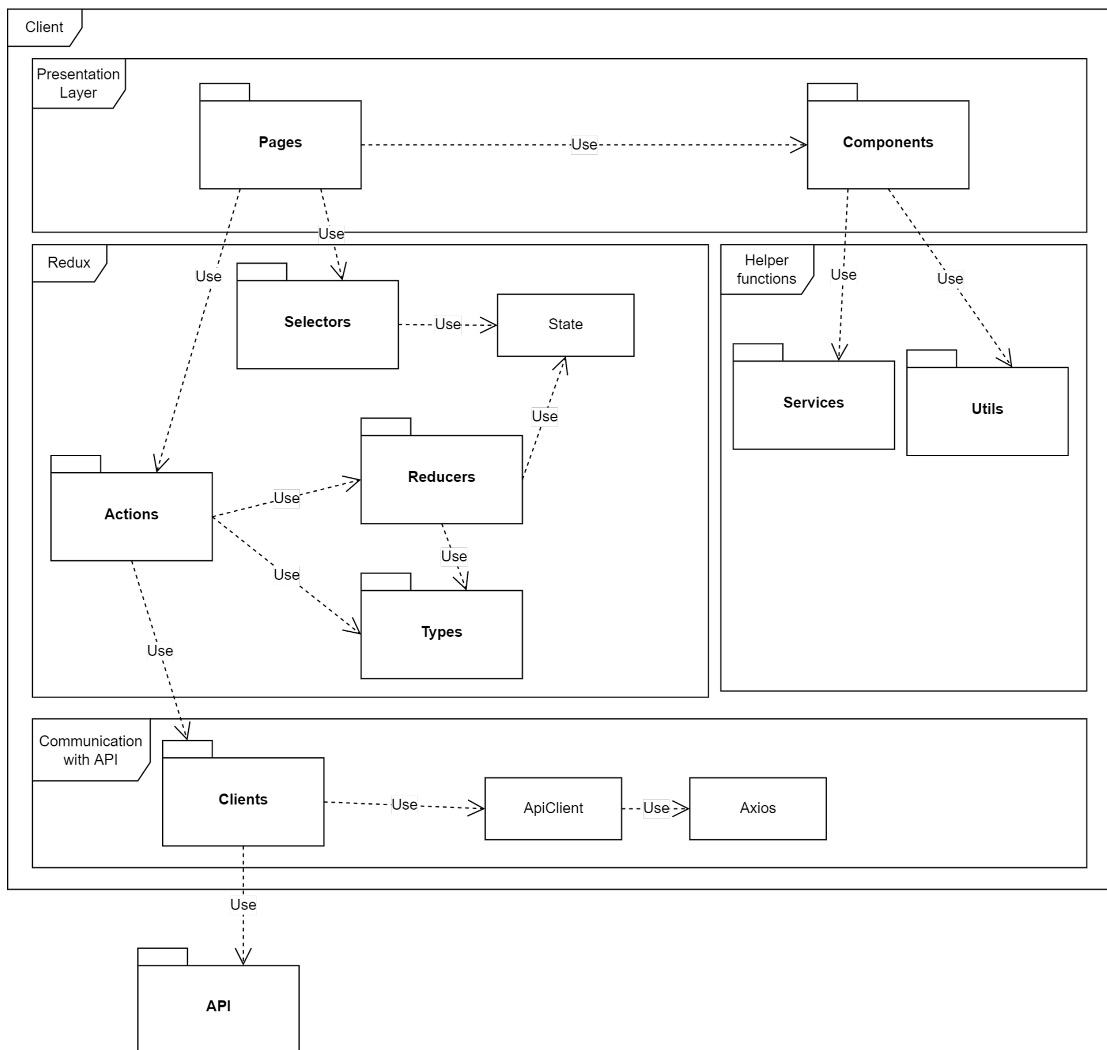
Prezentační vrstva slouží pro mapování HTTP endpointů na funkce fasád v rámci Business Layer a implementuje nám REST rozhraní. Kontrolery definují pro jednotlivé endpointy HTTP metodu a parametry. U parametrů specifikujeme, jak budou předány, zda se jedná o formulářová data nebo budou součástí query. Následně provolají fasádu. V případě úpravy dat v databázi provedou upozornění pomocí NotifyHubu a vrátí výsledek operace. NotifyHub reprezentuje otevřené spojení pomocí websocketu mezi serverem a klienty a slouží pro zasílání změn v datech ze strany serveru. Tato metoda je efektivnější než polling, kdy by se každý klient musel periodicky dotazovat a stahovat nová data. V rámci notifikace se posílá, jaké záznamy byly přidány a jaké byly zneplatněny. Díky dynamickému přečíslování seznamů může být jednou operací upraveno mnoho záznamů.

3.4 Návrh frontendu

Pro tvorbu frontendu byla zvolena technologie React bez použití typescriptu jakožto střední cesta mezi Angularem a Vue. Pro centrální správu stavu aplikace byla zvolena knihovna Redux.

3.4.1 Architektura

V rámci klientské části aplikace byla zvolena standardní architektura pro SPA, kdy jsou jednotlivé stránky směrovány pomocí knihovny react-router-dom. Každá stránka je React komponenta skládající se z dílčích komponent. Tato architektura umožňuje držet kód čistý a úhledný. Většina komponent v projektu jsou funkcionální komponenty, které využívají hooky pro správu stavu.



Obrázek 3.5: Diagram architektury pro klientskou část systému

Komunikace se serverovou částí aplikace

Ke komunikaci se serverovou částí byla zvolena knihovna Axios, jež umožňuje asynchronně posílat požadavky na server a zapouzdřuje javascriptové fetch API. Instance objektu pro práci s knihovnou Axios je nakonfigurována a vytvořena v rámci souboru client, který obsahuje metodu getClient. Třída ApiClient tento objekt zaobalí a definuje volání veškerých HTTP metod. S třídou ApiClient již pracují klienti entit, kteří implementují volání jednotlivých endpointů API.

Motivace k zavedení centrálního stavu aplikace

Každá komponenta má své závislosti, data která má zobrazovat. Tato data musí stahovat ze serverové části. V rámci jednoho pohledu se může stát, že dvě či více komponent budou potřebovat ze serverové části stejnou informaci. Existují tři cesty, jak tento problém řešit, první je, že všechny komponenty požádají serverovou část nezávisle o stejná data, což povede ke zbytečnému zahlcování serveru. Druhá varianta je najít všechny komponenty se stejnou závislostí a správu těchto dat vytknout do komponenty nad nimi. Toto řešení sice nebude vytvářet zbytečné dotazy na server, ale dostaneme správu dat do části aplikace, jež s danými daty nijak nepracuje, čímž vznikne nepřehlednost. Nejlepší by bylo, kdybychom mohli data stáhnout, někam uložit a následně k nim odkudkoliv přistoupit, tohle přesně dělají knihovny pro správu a centralizaci stavu. V rámci této architektury byla zvolena knihovna Redux.

Architektura knihovny Redux

Celá architektura Redux se snaží rozdělit práci s centrálním stavem aplikace a odstínit komponenty od přímého přístupu k objektu stavu. Architektura je rozdělena na akce, typy, reducery pro úpravu dat a selektory pro přístup k datům.

Akce

Akce slouží pro zavolání serverové části pomocí klienta a reakci na odpověď. V případě, že se vyskytne chyba, tak se nastaví příznak chyby a chyba je následně prezentována uživateli. V případě, že byla upravena nějaká data, tak zavolá speciální funkci knihovny Dispatch, které se předá typ zprávy a data od serverové části aplikace. Metoda Dispatch najde vhodný reducer pro tento typ zprávy.

Typy

Jedná se o prostý výčtový typ specifikující typy zpráv pro reducery. Každý název zprávy musí být v celém systému jedinečný, jinak by docházelo k nechtěným úpravám stavu ve více reducerech zároveň.

Reducery

Reducery mají přístup k centrálnímu stavu. V případě, že nějaká akce zavolá metodu Dispatch, tak knihovna Redux projde veškeré registrované reducery. Každý reducer dokáže obsloužit určitou podmnožinu všech typů zpráv v systému. V případě, že daný reducer rozumí typu zprávy, tak vykoná úpravu stavu na základě dat, která dostane od akce.

Selektory

Selektory využívají komponenty, jež chtějí přistoupit k nějaké hodnotě z centrálního stavu

aplikace. Jedná se o funkce mající přístup ke stavu pomocí hooku useSelector, který poskytuje knihovna Redux.

Služby

Služby implementují samostatné logické funkce, jež neupravují centrální stav aplikace, jako je například stažení a uložení souboru ze serverové části aplikace do počítače klienta nebo instanciace objektů z knihovny signalR pro připojení k serveru pomocí websocketu.

Utils

Jedná se o pomocné třídy a funkce. Jsou zde například definice výčtových typů, definice vlastních react hooků a definice filtrů. Nejdůležitější částí jsou zde renderovací skripty, které na základě definice entity pomocí JSON souboru dokáží vytvořit dynamické pohledy. Díky těmto skriptům je možné změnu struktury dat promítnout na klientovi, a to změnou pouze v rámci jednoho JSON souboru. [21]

Komponenty

Zde je definována většina vlastních komponent, jež se využívají napříč celou aplikací. Jedná se o vlastní prvky formulářů, jako je například rozbalovací menu, komponenta pro nahrávání mediálního obsahu s náhledem, notifikační banner a mnoho dalších.

Stránky

V rámci složky pages je definice všech pohledů v rámci systému. Jsou zde stránky pro řešení chybových stavů, jako je výpadek serveru či chyba autorizace. Dále jsou zde pohledy pro jednotlivé tabulky, formuláře a detailní pohledy, které vycházejí z generických bázových komponent pro snížení redundance kódu.

Směrování

Směrování v rámci aplikace je zajištěno knihovnou react-router-dom, která umožňuje směrování v rámci SPA. Definovány jsou dva typy cesty: PublicRoute a GuardedRoute. PublicRoute nevyžaduje autorizaci uživatele a jedná se tedy o veřejnou část systému. Ostatní endpointy, které jsou obalené GuardedRoute, před zobrazením obsahu zkontrolují, zda je uživatel úspěšně autorizován, jinak zobrazí chybovou hlášku. Každý endpoint má definovanou stránku a cestu.

Kapitola 4

Implementace

V této části textu se podíváme na samotnou implementaci navrženého systému a jak jsou jednotlivé vrstvy architektury řešeny a realizovány.

4.1 Implementace serverové části

Implementace serverové části je v rámci jednoho řešení rozdělena do osmi projektů. Hlavní část programu je rozdělena na projekt, který pracuje s databází. Další část projektu řeší byznys logiku a poslední mapuje url adresy na funkce. Každá třída má své závislosti, abychom nemuseli životní cyklus všech závislostí spravovat uvnitř třídy, využijeme kontejner, do něhož závislosti zaregistrujeme a budou za nás automaticky vloženy. Každý projekt má další projekt, kde je testován, a jako poslední je projekt Common, kde jsou obecné definice různých rozšíření tříd a výčtových typů.

4.1.1 Autentizace

Serverová část bude zpřístupňovat data obsahující utajované informace podle zákona č. 412/2005 Sb., o ochraně utajovaných informací a o bezpečnostní způsobilosti, ve znění pozdějších předpisů. Z tohoto důvodu bude serverová část adresovatelná pouze z lokální sítě. Veškeré endpointy poskytující tyto informace budou vyžadovat ověření pomocí Windows Authentication, uživatel musí mít účet v doméně, musí být uveden v konfiguračním souboru uživatelů a musí mít adekvátní roli pro daný endpoint.

4.1.2 Injektáž závislostí

Vkládání závislostí je technika objektově orientovaného programování pro dosažení inverze řízení. V případě, že komponenta potřebuje ke svému běhu jinou třídu, tak si ji sama neinstanciuje, ale požádá o ni v rámci konstruktoru, nemusí tedy řešit životní cyklus komponent, na nichž je závislá. Tyto komponenty se pak dosazují z kontejneru, do kterého musí být před použitím tato závislost přidána. Z důvodu možné záměny implementace jednotlivých tříd jsou většinou komponenty závislé na rozhraních a konkrétní implementace je řešena až při konfiguraci daného kontejneru. Každá vrstva serverové části má vlastní instalátor, který do kontejneru zavádí své závislosti. [14]

4.1.3 Data Access Layer

V této části bude podrobněji rozebrána implementace serverové části, která zapouzdřuje entity framework.

Entity

Entity jsou datové třídy reprezentující jednotlivé tabulky databáze. Entity krom samotných datových položek, jež jsou v rámci tabulek, obsahují i vazební položky sloužící pro uložení a práci s navázanými entitami. Entity v sobě dále mohou obsahovat komparátor využívající se u testování. Každá entita dědí z bázové třídy, která slučuje společné položky pro soft delete a verzování záznamů. Společné vlastnosti jsou datum začátku a konce platnosti záznamu, identifikátor pro vyhledání starších verzí záznamu a jméno autora dané verze záznamu. Všechny entity dědí rozhraní IEntity, které obsahuje společné vlastnosti entit a slouží pro omezení pomalé reflexe v rámci generického bázového repozitáře.

Abstraktní továrna entit

Abstraktní továrna slučuje více továren entit a je dále využita pro testování a pro instanciaci entit. Pro každou entitu existuje metoda, jež vytvoří objekt a naplní entitu testovacími daty. Data jednotlivých entit můžeme ovlivnit indexem, který předáme jako parametr, což nám umožní rozeznat jednotlivé záznamy v rámci testů. Pro případ testování mapování na modely je možné předat i konkrétní identifikátor entity.

Databázový kontext

V rámci knihovny entity framework je definována třída DbContext, která je použita jako bázová třída pro kontext naší databáze a slouží pro objektově relační mapování. V rámci databázového kontextu definujeme veřejné položky typu DbSet, přes které následně přistupujeme k jednotlivým tabulkám databáze. Přes kontext se můžeme dotazovat pomocí jazyka LINQ, který se před materializací přeloží na SQL příkaz. Vazby mezi jednotlivými entitami jsou definované v rámci konfigurací, kde každá entita má vyčleněn vlastní konfigurační soubor. Tyto konfigurace jsou v rámci kontextu aplikovány uvnitř metody OnModelCreating. DbContext nám umožňuje tvořit transakce, jelikož data do databáze zapíše až po zavolání metody SaveChanges. [16]

Továrny pro databázový kontext

Továrny slouží pro instanciaci databázového kontextu. Pro databázový kontext jsou definovány tři různé továrny, které mají své případy užití. Všechny tyto továrny mají společné rozhraní IDbContextFactory s metodou CreateDbContext. Toto rozhraní nám umožňuje použít repozitáře pro produkční kód i testování.

První továrna SqlServerDbContextFactory bere jako parametr řetězec pro připojení k databázi a slouží pro připojení k SQL databázi.

Další továrna InMemoryDbContextFactory slouží pro instanciaci kontextu v rámci testování, kde nechceme pracovat s pravou databází a ohrozit produkční data.

Továrna DesignTimeDbContextFactory slouží pro aplikování migrací databáze. Řetězec pro připojení k databázi je předán přes příkazovou řádku. Migrace slouží pro dynamickou úpravu schématu databáze pomocí skriptů, které definují změny oproti minulé verzi.

Repozitáře

Repozitáře tvoří abstraktní vrstvu nad databázovým kontextem. Pomocí injektáže závislostí dostává repozitář referenci na komparátor přirozených čísel, http kontext pro zjištění aktuálního uživatele a databázový kontext. Repozitáře obsahují implementaci základních operací, jež budou dále využity v byznys logice. V rámci bázové třídy RepositoryBase jsou implementovány soft delete a verzování jednotlivých záznamů. Dále se zde vypočítává hierarchické pořadové číslo a v případě úpravy se i přepočítávají ostatní pořadová čísla v seznamu. Z důvodu přepočítávání pořadových čísel je nutné vracet seznam přidaných záznamů a seznam identifikátorů záznamů, které mají být odstraněny. Z důvodu použití soft delete nelze použít nastavení entity framework pro mazání navázaných komponent a z důvodu verzování záznamů je nutné upravovat návaznosti i při editaci záznamu. V rámci bázové třídy jsou nadefinované funkce EditDependencies a DeleteDependencies, které jsou předefinované v každém konkrétním repozitáři. Pro úpravu konkrétní položky entity je funkce UpdateField, jež pomocí reflexe zjistí, zda existuje daná položka na entitě a nastaví požadovanou hodnotu, kterou dynamicky přetypuje na požadovaný typ. Dále si rozebereme zajímavější pasáže z jednotlivých repozitářů.

Výpočet pořadového čísla

V případě, že daná entita nemůže mít pod sebou stejnou entitu a nelze tvořit hierarchii, tak získání nového pořadového čísla probíhá následovně: z databáze se vyberou všechny validní záznamy, u nichž pořadové číslo odpovídá regulárnímu výrazu ^\d+$, který slouží pro filtrování celých čísel. Daný seznam se seřadí sestupně a vybere se první pořadové číslo. V případě, že záznam neexistuje, tak je přiřazeno pořadové číslo 1, v opačném případě se číslo inkrementuje o jedna a přiřadí k entitě.

V případě souprav a předmětů v soupravě je nutné dané funkci předat i informace o nadřazených záznamech. V případě, že záznam je podřazen záznamu a není tedy v hierarchii na první úrovni, tak se z databáze vyberou záznamy, které jsou podřazené stejnému záznamu. Opět se vyberou pouze platné záznamy a seřadí se podle pořadového čísla. V případě, že takový záznam neexistuje, tak se vyhledá nadřazený záznam a jeho pořadové číslo se konkatenuje s řetězcem ".1". V případě, že podřazený záznam existuje, tak se vezme poslední část pořadového čísla za tečkou a číslo se inkrementuje. V případě, že záznam není žádnému záznamu podřazen, se postupuje stejně jako v případě záznamů bez hierarchie.

Oprava struktury pořadových čísel

V rámci aplikace je požadavek na přeuspořádávání záznamů v tabulkách. Záznamy jsou řazeny podle pořadového čísla. To v praxi znamená, že v případě změny pořadového čísla z 50 na 20 je nutné všechny záznamy s pořadovým číslem 20 až 49 inkrementovat o jedna. V případě hierarchických záznamů je nutné adekvátně upravit i podřazené záznamy. Tato funkcionalita je v rámci bázového repozitáře definovaná v rámci virtuální metody FixItemIdStructure. Z databáze se vytáhnou platné záznamy a v případě předání další podmínky jsou vyfiltrovány. Rozšiřující podmínka je předána u hierarchických záznamů. V případě, že neexistuje kolize s daným pořadovým číslem, tak je oprava ukončena. Následně se vezmou pořadová čísla původního a nového záznamu a kontrolují se jednotlivé složky, kde se liší, jakmile se najde úroveň, ve které se liší, může se určit, zda se bude přičítat či odčítat u modifikovaných záznamů a na jaké úrovni dojde k posunu pořadových čísel. Nyní se iteruje přes vybrané záznamy, v případě, že záznam není tak hierarchicky zanořen jako úroveň, na níž se upravuje číslování nebo je mimo rozsah úprav, tak je záznam přeskočen. Ostatní

záznamy se upraví podle směru úprav a změny se uloží do databáze. Funkce vrací seznam upravených záznamů. Entity s hierarchií mají funkci rozšířenou o získání všech podřazených záznamů.

Abstraktní továrna repozitářů

Abstraktní továrna slučuje továrny všech repozitářů a je dále využita u testování pro instanciování repozitářů. Továrna v rámci parametrického konstruktoru bere závislosti pro repozitáře krom samotného databázového kontextu. Abstraktní továrně nemůže být předána továrna na databázový kontext, jelikož všechny repozitáře musí pracovat se stejnou instancí databázového kontextu, na kterou musí mít referenci i nadřazený unitOfWork pro uložení do databáze.

UnitOfWork

UnitOfWork je třída, přes níž přistupujeme k jednotlivým repozitářům. Pomocí injektáže závislostí dostává referenci na repozitáře a zpřístupňuje je pomocí veřejných položek třídy. Dále obsahuje generickou funkci GetRepository, která vrátí konkrétní repozitář na základě hodnoty z výčtového typu entit a je využita v bázové fasádě. Veškeré operace provedené nad repozitáři se ukládají pouze do kontextu, pokud chceme data zapsat a uložit do databáze, je nutné provolat funkci Commit v rámci unitOfWork. Tento mechanismus slouží jako prevence proti nekonzistentnímu stavu, kde u složitějších operací může dojít k chybě či vypnutí systému a operace by se neprovedla celá.

Továrna UnitOfWork

Továrna UnitOfWork slouží pro tvorbu instancí unitOfWork pro testování. Jako závislosti bere továrnu databázového kontextu a abstraktní továrnu repozitářů. V rámci metody CreateUnitOfWork vytvoří jeden databázový kontext, který předá všem repozitářům.

Instalátor

V rámci instalátoru pro Data access layer registrujeme databázový kontext. Dále pomocí knihovny Scrutor, která je publikována pod MIT licencí, skenuje danou assembly a hledá třídy implementující specifikované rozhraní. Tyto třídy automaticky registruje do závislostního kontejneru. Registrujeme tedy repozitáře, unitOfWork a databázový kontext.

4.1.4 Business Layer

V této části bude podrobněji rozebrána implementace business layer. Business layer pracuje nad unitOfWork, který jsme již definovali v kapitole 4.1.3 Data Access Layer.

Modely

Modely reprezentují již nějaký určitý pohled na data, která se posílají na klienta. Obsah dat se liší podle toho, zda se jedná o data do tabulky, detailní pohled nebo například data do dynamického select boxu. Vhodným mapováním entit na modely se šetří množství přenesených dat mezi klientem a serverovou částí. Mezi hlavní modely patří ListModely, které se využívají pro zobrazení v tabulkách a neobsahují informace o navázaných záznamech

a historii záznamu. Pro zobrazení kompletních informací o entitě slouží DetailModely obsahující i reference na nadřazené záznamy a kompletní historii záznamu. Pro úpravu dat využíváme FormModely, jež jsou nutné z důvodu předávání multimediálního obsahu do položek typu IFormFile. Při otevření formuláře jsou dotaženy informace pro select boxy a výchozí hodnoty, pro tyto účely slouží FormDefaultModely. SelectModely obsahují popisek a Id do select boxu. Postranní menu je předáváno pomocí SideBarMenuModelů.

Mapování

Pro mapování mezi entitami na modely se využívá knihovna Automapper, jež je distribuovaná pod MIT licencí. Mapování je nakonfigurované pomocí mapovacích profilů. Každá entita má svůj mapovací profil, který je odvozen od bázového mapovacího profilu. Přes generickou metodu CreateMap se vytvoří mapování mezi specifikovanými třídami. V případě komplexnějších mapování například pro select boxy, kde skládáme hodnotu pro select box z více hodnot entity, použijeme metodu ForMember a specifikujeme konkrétně pravidla pro mapování dané položky.

Fasády

Fasády mají generickou bázovou třídu FacadeBase, která implementuje rozhraní IFacade. Pro korektní fungování generické fasády musíme předat veškeré typy modelů dané entity, pro níž je fasáda určená. Hlavní úlohou fasád je aplikace mapování modelů na entity pro komunikaci s repozitáři. Dále zde probíhá propojení entit s jejich mediálním obsahem a jejich správa. Fasády vrací přemapované listy modelů a identifikátorů.

Obsluha mediálního obsahu

Každá entita může obsahovat náhledovou fotku a přílohy. Záznamy v soupravách se mohou opakovat napříč projekty, díky čemuž zde hrozí velká redundance dat. Jako prevence ukládání redundantních dat se počítá kontrolní součet daného souboru pomocí algoritmu SHA1, ten se následně vyhledá v databázi. V případě, že soubor s daným součtem existuje v databázi, tak entita dostane odkaz na tento soubor. V opačném případě se uloží na server a přidá se záznam do databáze s daným kontrolním součtem. Tento přístup má nevýhodu, jelikož vyžaduje identické soubory a v případě, že dojde k nepatrné změně (například ořez obrázku) se uloží jako nový soubor.

Fasády pro generování souborů

V rámci případů užití je potřeba generovat výstupní soubory ve formátu xlsx pro seznam komponent a funkční celky dané soupravy. Pro programovou tvorbu souborů Microsoft Excel využijeme knihovnu ClosedXML, která je distribuovaná pod MIT licencí. Knihovna ClosedXML nám umožní od zapisování na určitou buňku, formátování, nastavení rozložení pro tisk až po vkládání médií. Společné nastavení pro všechny dokumenty jako je nastavení metadat souboru, nastavení odsazení a funkce pro práci s obrázky je vytknuto v bázové třídě. Práce s obrázky je komplikovanější, jelikož můžeme vložit obrázek do konkrétní buňky, nicméně obrázek není buňce přizpůsoben. Zjistíme velikost buňky a v případě, že obrázek je větší než samotná buňka, tak přepočítáme nové rozměry a změníme velikost obrázku. Dále spočítáme odsazení od okraje tak, aby byl obrázek uprostřed buňky. Další problém nastává s jednotkami, kde soubor Excelu pracuje v palcích a obrázek máme v pixelech. Experimentálně bylo zjištěno, že při použití přepočtu jeden palec na 7 pixelů jsou výsledky

použitelné. Následně pro každý dokument nastavíme úvodní stránky a iterujeme přes požadovaná data. Data filtrujeme pomocí parametrů předaných od klienta. Každý řádek má nastavené formátování podle specifikace. Jelikož pořadí není pevně dané, tak využíváme reflexy pro nalezení hodnoty daného záznamu pro buňku.

Ukázka generovaného souboru

Instalátor

Business Layer také obsahuje instalátor, který registruje fasády pro entity a fasády pro generování souborů do kontejneru závislostí.

4.1.5 Presentation Layer

Jedná se o poslední vrstvu serverové části, jež nám implementuje REST rozhraní naší aplikace. Slouží pro zpřístupnění metod fasád a obsluhu websocketu pro notifikaci klientů.

Kontrolery

Kontrolery slouží pro mapování url adres na funkce. Jsou nadefinované dva bázové kontrolery pro hierarchické a nehierarchické entity. Hierarchické entity jsou předměty a soupravy, kde každý předmět může mít pod sebou další předměty a každá souprava může mít pod sebou další soupravy. Každý kontroler má závislost na konkrétní fasádě a kontext pro notifikační hub. V případě, že byla provedena operace, která způsobila změnu dat v databázi, tak kontroler pomocí notifikačního hubu posílá zprávu všem klientům o této změně. Každý kontroler obsahuje attribut, který říká verzi API, pro níž je určen, url cestu pro daný kontroler a specifikace autorizační politiky.

Notifikační hub

Notifikační hub je v rámci startupu namapován na určitý API endpoint a slouží pro obsluhu web socket spojení mezi serverem a klienty. Běžný scénář je, že klient žádá server o data, nicméně v tomto případě potřebujeme, aby server informoval o změně v datech ostatní klienty pomocí zprávy. Zpráva obsahuje typ upravené entity, seznam přidaných záznamů a seznam zneplatněných záznamů.

Startup konfigurace serverové části

V rámci startup konfigurace proběhne nastavení autentizace, cors politik a registrace závislostí do kontejneru. Nedílnou součástí je problematika předání connection stringu do databáze, což je citlivý údaj a v případě verzování na git není vhodné, aby byl umístěn v rámci zdrojového textu. V rámci C# ASP.NET Core se využívá Secret manager, který vytvoří na lokálním počítači JSON soubor, do nehož se ukládají citlivá data. C# projekt dostane identifikátor tohoto souboru do konfigurace pomocí položky UserSecretsId v rámci souboru csproj. Tímto jsou data oddělena od zdrojového kódu a nejsou verzovány nástroji, což vyžaduje, aby všichni vývojáři pracující na projektu dostali tyto citlivé informace jiným kanálem a vložili je do svého user secret souboru.

SEZNAM PŘEDMĚTŮ V SOUPRAVĚ: Souprava 2

Souprava 2

SEZNAM PŘEDMĚTŮ V SOUPRAVĚ

Výrobní číslo: 7536419

Utvar 3

Obrázek 4.1: Titulní strana generovaného dokumentu s informacemi o soupravě

SEZNAM PŘEDMĚTŮ V SOUPRAVĚ: Souprava 2

Pořadové číslo	Původní pořadové číslo	Název	Výrobní číslo	Katalogové číslo	Evidenční číslo	Jednotka	Počet	Poznámky	Modernizace	Rozhodnutí	KČM	Předání k TZ	Komplexní prohlídky	Zástavba/výnos	Obrázek	Funkční celek
Lokalita 4																
3	2	Kontajner	654123987	456	3	2	2			SKLAD AČR	123		NEPRAVDA	zástavba		FC1
4	333	Predmet v kontajneru 1	123	258	7					EKO/LIKVIDACE	123	Nepředáno k TZ	PRAVDA			
5	123	Predmet 2.1.1								zůstává	123		NEPRAVDA			FC2
6	85	Predmet v kontajneru 2							dodáno	zůstává	123		NEPRAVDA			
Utvar 3																
7	123	Predmet 1	123456789	123	321	ks	5	pon		zůstává	2	Nepředáno k TZ	NEPRAVDA	zástavba		
8		Propiska	Predmet 2		123456	ks				zůstává	123		NEPRAVDA			
9	5	Predmet 5	987654321	2	1	ks				SKLAD AČR	123		NEPRAVDA			
10	4	Predmet 4	123	7	2	ks				SKLAD AČR	123		PRAVDA			
11	8	Nezarazeny predmet k presunu								zůstává	123		NEPRAVDA			
12	7	Dometic FreshJet 2200 Upraveno	789		456	ks				zůstává	123		NEPRAVDA			
13	17	Dometic FreshJet 2200			456	ks	5			zůstává	123		NEPRAVDA			
Lokalita 1																
14	3	Predmet 3	123	1	2	2	5			původní	123456		NEPRAVDA			FC2
15	200	Predmet se sablonou	123	123456	123456789	ks	12	poznamka	dodáno	zůstává	123		PRAVDA			Server

2

Obrázek 4.2: Seznam předmětu v soupravě

Cross-origin resource sharing

CORS je bezpečnostní mechanismus implementovaný v rámci webového prohlížeče, který umožňuje posílat http požadavky pouze na stejnou doménu se stejným portem. Každý http požadavek obsahuje hlavičku Origin, jež říká doménu, z níž požadavek odešel. V případě, že se jedná o požadavek na jinou doménu, tak server musí v rámci odpovědi poslat hlavičku

Access-control-allow-origin s hodnotou, která je shodná s hlavičkou Origin v požadavku. Tento problém je nutné řešit právě při zvolené architektuře, kde běží dvě separátní aplikace v podobě klienta a serveru. V rámci konfigurace naší serverové části tedy uvedeme adresu, kde běží klient jako jediný povolený zdroj. Klientovi nastavíme možnost posílat libovolné hlavičky a využívat libovolné http metody. Pomocí http hlavičky expose headers umožníme klientovi číst hlavičku požadavku Content-disposition, kde v případě posílání vygenerovaného souboru bude jeho název. [11]

Autentizace
V rámci nastavení autentizace přečteme konfigurační soubor a deserializujeme data. Uživatele máme rozdělené do tří rolí: uživatel, manažer a vývojář, tyto uživatele roztřídíme a přidáme do patřičných skupin dané politiky.

Nastavení Cross-origin resource sharing pro statické soubory
V rámci serverové části ukládáme mediální obsah na server a cesty k těmto souborům do databáze. K tomu, aby klient mohl přistoupit k daným souborům, je nutné explicitně nastavit cors politiky pro tyto soubory.

Registrace závislostí
Registrace veškerých závislostí dané aplikace do kontejneru probíhá instanciací veškerých instalátorů a zavoláním metody Install. Metoda Install dostane přes parametr seznam služeb dané aplikace, do kterého může zaregistrovat služby, za něž daný instalátor odpovídá.

Nastavení endpointů API
Pomocí metody UseEdpoints namapujeme veškeré endpointy kontrolerů a našeho nofikačního hubu pro websocket.

4.2 Implementace frontendu

Frontendová část je rozdělena podle standardní šablony pro React. Soubor .gitignore slouží pro specifikaci souborů, které nemají být verzovány nástrojem git. Soubor package.json obsahuje seznam použitých knihoven a závislostí projektu, včetně jejich verzí, které je nutné při stažení projektu doinstalovat pomocí nástroje npm. Soubor env obsahuje seznam proměnných prostředí, kde uchováváme informaci o URL, kde běží serverová část. Pro specifikaci URL API v rámci lokálního vývoje slouží soubor env.development. Soubor editorconfig slouží pro nastavení formátování zdrojových souborů. V další části rozebereme obsah složky src, která obsahuje veškerou implementaci klienta.

4.2.1 Komunikace se serverovou částí aplikace

Client

Soubor obsahuje instanciaci objektu pro knihovnu axios, kterou zpřístupňuje pomocí funkce getClient. V rámci nastavení nastavujeme bázovou url, takže není nutné při každém requestu uvádět celou url požadavku. Dále se zde nastavuje hlavička WithCredentials sloužící k tomu, aby se ke každému požadavku přidaly informace nutné k autentizaci uživatele. Dále se zde nastavuje reakce na úspěšnou i neúspěšnou odpověď serveru.

ApiClient

Třída ApiClient si pomocí funkce getClient vytvoří objekt pro přístup k funkcím knihovny axios. Následně práci s knihovnou axios zaobalí a vytvoří funkce pro jednotlivé HTTP metody. V závislosti na metodě se předávají parametry. Veškeré metody mají společné parametry url a konfigurace, díky nimž můžeme dále specifikovat na jakou url adresu poslat daný požadavek a jaké hlavičky přidat. Metody jako POST, PUT a PATCH mají navíc parametr, přes který se předávají data.

Klienti

Každá entita má vlastního klienta, což je objekt obsahující volání konkrétních koncových bodů serverové části. Entity mají mnoho koncových bodů stejných, například CRUD operace. Společné funkce jsou vytknuté do bázového objektu BaseClient, který konkrétní klienti entit pouze rozšiřují.

4.2.2 Komunikace mezi jednotlivými klienty pomocí websocketu

Synchronizace klientů probíhá pomocí knihovny signalR od Microsoftu. Z knihovny signalR se instanciuje HubConnectionBuilder, jemuž se nastaví url, na kterém websocket běží. Dále nastavujeme, aby se automaticky připojoval v případě výpadku a asociujeme typy zpráv s obslužnými rutinami. V rámci klienta existují dvě instance, jedna pro synchronizaci záznamů a druhá pro aktualizaci menu. Tyto instance obalíme službou NotifyService, díky níž implementujeme návrhový vzor jedináček a zpřístupníme jednotlivé instance. Tato služba je dále využita v komponentě SignalHandler, která definuje obslužnou rutinu pro jednotlivé zprávy. Na základě typu upravovaných záznamů se zvolí soubor akcí z Redux paternu a provolá se akce UpdateFromSignal, která zneplatní záznamy s identifikátory v poli oldIds a přidá záznamy z pole listModels.

4.2.3 Centrální stav aplikace pomocí knihovny Redux

Centrální stav store je objekt vytvořený funkcí createStore, která je exportována knihovnou Redux. Pro tvorbu storu je potřeba funkci předat výchozí stav a rootReducer, o němž si více povíme dále v této kapitole. Dále při tvorbě store zapneme možnost používat thunk pomocí middlewaru. Thunk je funkce, která sestaví novou funkci a vrátí ji. Vrácenou funkci je možné zavolat později. Tato opožděná volání funkcí využijeme v rámci akcí. Následně pomocí composeWithDevTools zapneme možnost využití rozšíření Redux DevTools, které nám umožní při běhu aplikace nahlížet do centrálního stavu aplikace a na typy akcí, jež byly nad centrálním stavem aplikace zavolány.

Akce

Funkce getBaseActions je exportována souborem ActionBase a vrací objekt, který obsahuje společné akce pro všechny entity. Každý objekt akcí má přístup ke konkrétnímu klientovi pro volání endpointů API. Každá entita obsahuje vlastní soubor s akcemi, jež rozšiřují společný objekt o specifické akce. Každá akce je thunk, který vrací asynchronní funkci. Struktura akcí je jednoduchá, celý blok je obalen do bloku try, kde je v případě chyby pomocí akce notifikací uživatel o chybě informován. V případě, že akce proběhne v pořádku, tak jsou získána data z odpovědi a je zavolán konkrétní reducer pro reflektování změny stavu. Po

úspěšné změně stavu je uživatel opět informován o úspěšném provedení operace pomocí notifikačního banneru.

Reducery

Každý reducer je funkce, která má přístup ke stavu, konkrétní akci a množině typů zpráv. Funkce obsahuje jeden velký switch statement, který na základě typu zprávy vezme data ze zprávy a upraví aktuální stav. Upravený stav je funkcí vrácen a knihovna ho uloží jako nový stav. Stav dokáže pracovat pouze s jedním reducerem zároveň, proto se používá funkce combineReducer, jež sloučí všechny reducery v rámci aplikace do jednoho. Z tohoto důvodu je důležité, aby každý název typu akce byl unikátní v rámci celé aplikace a nedošlo tak k souběžné úpravě stavu ve více reducerech. Pro zachování jedinečnosti názvu typů akcí byla zvolena suffixová notace, kdy na konec názvu akce se přidá název entity, pro kterou je tato akce určená.

Selektory

Selektory slouží pro zapouzdření stavu a přístup k datům stavu pomocí funkcí. Implementačně se jedná o objekt obsahující jednotlivé selektory. Každý selektor je funkce, která jako parametr bere stav a vrací libovolnou část stavu nebo logickou hodnotu. Objekt stavu je injektován knihovnou při použití hooku useSelector nebo při použití funkce connect v části mapování stavu na závislosti.

4.2.4 Služby

Služby jsou samostatné funkce, které neupravují centrální stav a provádí volání serverové části, jako je například získání souborů ze serverové části. Další služba slouží pro tvorbu instancí objektu pro spojení s klientem, kde je důležité, aby každý klient měl právě jednu instanci pro komunikaci se serverovou částí.

4.2.5 Utils

Utils obsahují definici pomocných funkcí, definic a výčtových typů. Hlavní část těchto pomocných funkcí jsou funkce pro dynamické generování tabulek, pohledů a formulářů.

Filtry

Složka filters obsahuje definice vlastních filtrů, jako je například kontrola hierarchického pořadového čísla. Filtry jsou funkce, které vrací logickou hodnotu, případně chybovou hlášku. Většina filtrů je využita v rámci formulářů pro validaci.

Hooky

Složka Hooks obsahuje implementace vlastních React hooků pro naše vlastní React komponenty. Hook useLocalStorage slouží pro stavovou proměnnou, která je uložena a případně obnovena z JS local storage. Hook useLocalStorage umožňuje ukládání komplexních objektů, jež stringifikuje a uloží jako řetězec.

Modely

Modely obsahují definici popisu entit. Každá entita je popsána komplexním JSON souborem, který specifikuje, jaké položky daná entita obsahuje a jaký má mít textový popisek. Dále je zde informace o typu komponenty pro formulář a validační funkce. V případě statického select boxu je zde výčet hodnot. Pokud se jedná o dynamický select box, tak obsahuje informaci v jaké položce jsou možnosti pro select box a do jaké položky má být uložena odpověď. Dále záznam položky obsahuje informaci, kde má být položka použita, jelikož jsou položky, které se zobrazují pouze na některých pohledech.

4.2.6 Renderovací knihovny

Renderovací knihovny slouží k dynamickému renderování částí aplikace, které jsou závislé na datech ze serverové části. V rámci zadání se často modifikuje schéma dat, aby nebylo nutné po každé změně měnit veškeré pohledy, jako jsou tabulky, formuláře nebo detailní pohledy. Z tohoto důvodu jsou v rámci projektu samostatně definované funkce pro renderování jednotlivých částí aplikace. Funkce potřebují objekt obsahující data ze serverové části. Dále potřebuje model entity, který obsahuje definici položek. K tomu, Aby bylo možné rozdělit pohled do více sloupců, tak jsou předány i dva indexy, jež se použijí k ohraničení pole položek z modelu.

Knihovna pro vykreslení detailního pohledu

V rámci detailního pohledu se vykresluje label, který popisuje danou položku a hodnotu dané položky. Přílohy umožňuje stáhnout a select boxy, které většinou slouží k vázání entit mezi sebou, renderuje jako odkaz na přidružený detailní pohled.

Knihovna pro vykreslení formulářového pohledu

Funkce pro vykreslování formulářů má krom čtyř zmíněných parametrů ještě parametr pro přístup k funkcím knihovny react-hook-form sloužícím pro obsluhu formulářů. Další přidaný parametr je pro starý záznam v případě, že dojde k problémům s vícenásobným přístupem k jedné entitě. V případě, že více uživatelů má otevřenou úpravu jedné entity a dojde k uložení, tak se daný záznam zneplatní a změny se projeví do nového záznamu, který dostane nový identifikátor. Ostatní klienti jsou o této skutečnosti informováni otevřeným websocketem, kde přijde informace o zneplatnění aktuálního záznamu. Všichni uživatelé s otevřeným formulářem starého záznamu jsou přesměrování na úpravu nového záznamu. K zamezení ztráty rozpracované úpravy formuláře, je záznam uložen a uživateli je po přesměrování u každé položky zpřístupněna původní rozpracovaná hodnota.

Knihovna pro vykreslení pohledu pro exportování

Tato funkce slouží pro generování formuláře pro export seznamu komponent. Pro každou datovou položku komponent se vygeneruje malý formulář, který umožňuje volbu, zda bude položka v seznamu zahrnuta, na kolikáté pozici a jaký bude mít nadpis, jelikož se tyto věci mohou měnit v závislosti na tom, komu má být dokument určen. Zadané pořadí je validováno na klientovi. Dále je možné specifikovat, jaké kategorie záznamů budou exportovány.

4.2.7 Komponenty

Komponenty jsou základním stavebním kamenem každého pohledu. V rámci složky components jsou nadefinované obecné komponenty, jež jsou znovupoužitelné v rámci celé aplikace. Většina použitých komponent jsou funkcionální komponenty, které pro nastavování stavových proměnných využívají hook useEffect, jemuž je možné předat seznam závislostí a zamezit tak nekonečnému pře-načítání komponenty, jelikož při každé změně stavové proměnné nebo závislostí komponenty se daná komponenta překreslí.

Komponenta tabulka

Mezi hlavní komponentu celého projektu patří tabulka, která byla optimalizována podle případů užití na efektivní práci.

Úprava zobrazení tabulky
Každý uživatel má možnost si pomocí nastavení upravit, jaké sloupce mají být zobrazené a šířku jednotlivých sloupců. Nastavení se ukládá pomocí vlastního hooku useLocalStorage v rámci webového prohlížeče.

Filtrování dat
Filtrovat jednotlivé záznamy je možné globálním filtrem nebo filtrem specifickým pro každý sloupec tabulky. Z důvodu verzování záznamů je možné zadat datum a zobrazit seznam položek, které byly aktuální v dané datum. V případě, že je filtrováno pomocí data, tak se ze serverové strany dostahují zneplatněné záznamy.

Manipulace s daty
Jednotlivé záznamy je dále možno řadit podle libovolného sloupce pomocí kliknutí na záhlaví sloupce. Pomocí přetažení záhlaví sloupce lze měnit pořadí sloupců. Výchozí řazení pro běžné zobrazení je podle pořadového čísla a pro historii záznamu podle platnosti. Držením záznamu a přesunu lze záznamy přeuspořádat, čímž dojde k automatickému přepočítání pořadových čísel.

Úprava záznamů
Pro rychlou úpravu dat lze využít možnost dvojkliku na buňku v tabulce, která se na základě datového typu transformuje na editační buňku a umožní úpravu dané položky. Další možností úpravy dat jsou ovládací ikony u každého záznamu a kontextové menu, které jde nad každým záznamem vyvolat. Dále je možné zaškrtávacím polem vybrat a upravovat jednotlivé záznamy z vyvolaného menu v záhlaví. V případě vybrání více záznamů lze tyto záznamy hromadně smazat.

4.2.8 Stránky

Stránky jsou speciální případy komponent, jež se mapují pomocí směrovače na jeden z koncových bodů klienta. Jednotlivé stránky přistupují k centrálnímu stavu aplikace, který ovšem není perzistentně uložen, proto je v rámci každé stránky ověření hodnot v centrálním stavu. V případě, že hodnoty chybí, tak jsou znovu staženy ze serverové části. K tomuto scénáři může dojít u přenačtení stránky nebo například u odeslání odkazu na detailní pohled či formulář k úpravě záznamu.

Hlavní pohled na entity

Každá entita má vlastní pohled, do něhož se dá dostat pomocí navigačního menu či pomocí postranního hierarchického menu. Pohled každé entity se skládá z bočního menu a tabulky. Nad tabulkou je možnost přidat další záznam, vygenerovat seznam či přečíslovat seznam. Dále se zde nachází hierarchická cesta, která umožňuje zobrazit detailní pohledy nadřazených záznamů daného pohledu. V případě výběru jednoho či více záznamů v tabulce pomocí zaškrtávacího pole se v horní části pohledu zobrazí panel umožňující manipulaci s vybranými záznamy. V případě, že je vybrán jeden záznam, je možné upravit záznam, vytvořit podřazený záznam, smazat záznam či přejít na detailní pohled záznamu. V případě, že je vybráno více záznamů, tak je možné je všechny smazat najednou. Hlavní pohled pro komponenty dále obsahuje vpravo panel, který obsahuje informace o funkčních celcích dané soupravy. Dále je možné zobrazit na panelu funkční celky a to jaké komponenty obsahuje. Dále je možné přes tento panel vyexportovat Excel soubor s daným seznamem funkčních celků.

Obrázek 4.3: Pohled na seznam předmětů v soupravě

Formulář pro úpravu entit

Každá entita má vlastní pohled pro formulář, který slouží pro úpravu existujícího záznamu nebo pro tvorbu nového. Pohled volá funkci renderovací knihovny. Hlavní úlohou komponenty pohledu je ze serverové části dotáhnout výchozí hodnoty pro daný záznam na základě parametrů a umístění nové entity. Mezi další funkcionalitu v rámci pohledu patří detekce úpravy záznamu jiným klientem a přesměrování na nový záznam s uloženým rozpracovaným záznamem.

Obrázek 4.4: Pohled na editační formulář předmětu

Detailní pohledy

Detailní pohled entit slouží pro zobrazení všech informací o daném záznamu. Jsou zde informace o autorovi záznamu, platnosti a všech návazností záznamu, na které se dá pomocí odkazů přejít. Pod hodnotami samotného záznamu je tabulka původních záznamů seřazená podle platnosti.

Obrázek 4.5: Pohled na detail předmětu s historií záznamu

Pohled pro generování seznamu předmětů

Pohled slouží pro nastavení parametrů generovaného dokumentu. Pro každou hodnotu je možné nastavit text záhlaví, pořadí sloupce a zda má být sloupec vůbec zahrnut v generovaném dokumentu. Jelikož častým případem užití je generování seznamu předmětů, jež jsou k ekologické likvidaci, tak je možnost zaškrtnout, jaké kategorie chceme v daném dokumentu zahrnout.

Pořadové číslo
Nadpis sloupce: Pořadové číslo — Pořadí *: 1
Původní pořadové číslo
Nadpis sloupce: Původní pořadové číslo — Pořadí *: 2
Název
Nadpis sloupce: Název — Pořadí *: 3
Výrobní číslo
Nadpis sloupce: Výrobní číslo — Pořadí *: 4
Katalogové číslo
Nadpis sloupce: Katalogové číslo — Pořadí *: 5
Modernizace
Nadpis sloupce: Modernizace — Pořadí *: 10
Rozhodnutí
Nadpis sloupce: Rozhodnutí — Pořadí *: 11
KČM
Nadpis sloupce: KČM — Pořadí *: 12
Předání k TZ
Nadpis sloupce: Předání k TZ — Pořadí *: 13
Komplexní prohlídky
Nadpis sloupce: Komplexní prohlídky — Pořadí *: 14
Generovat kategorie:
zůstává
SKLAD AČR
EKO/LIKVIDACE
původní
nové

Obrázek 4.6: Pohled pro nastavení parametrů generování seznamu předmětů

Pohledy pro prezentaci kritických chyb

V případě, že dojde ke kritické chybě, je aplikace přesměrována na pohled, který informuje o dané chybě. Za kritickou chybu se považuje nedostupnost serverové části aplikace nebo chyba autorizace. Přesměrování na chybovou stránku u problému s autorizací je i z důvodu nezobrazení struktury dat, kterou by bylo možné vyčíst z navigačního menu.

Obrázek 4.7: Pohled pro prezentaci kritických chyb

4.2.9 Směrování v rámci single page application

Směrování v rámci Reactu je pomocí knihovny react-router-dom. Index celého systému vytváří instanci komponenty App zaobalující celou aplikaci. Tato aplikace je obalena komponentou, jež poskytuje centrální stav a dále BrowserRouterem, který řeší směrování v rámci celé aplikace. V rámci aplikace je dále instanciována vlastní komponenta RouterSwitch, kterou již definujeme asociace mezi url adresami a stránkami. RouterSwitch obsahuje seznam všech endpointů pro klientskou část aplikace. Využívají se dvě vlastní komponenty PublicRoute pro veřejné endpointy a GuardedRoute pro endpointy s omezeným přístupem. Oběma komponentám se předá odkaz na stránku a cesta, na níž má být namapována. GuardedRoute před samotným přesměrováním pošle dotaz na serverovou část, která vrátí jméno uživatele, jeho roli a zda je autorizován. V případě, že je autorizován, dojde k přesměrování na předanou komponentu v parametrech. V opačném případě GuardedRoute rozhodne, zda došlo k chybě autentizace nebo je serverová část nedostupná. V případě kritické chyby GuardedRoute uživatele přesměruje na adekvátní stránku s chybovou hláškou. Přesměrování přes GuardedRoute ještě obalí komponentu rozložením stránky pro autorizované uživatele obsahující základní strukturu aplikace, jako je navigační menu, notifikační banner a patička stránky.

Kapitola 5

Testování aplikace

V této části textu se zaměříme na testování a kontrolu zda se jednotlivé funkce chovají podle specifikace a celá aplikace odpovídá vstupním požadavkům.

S testováním aplikace mi pomohla firma Vojenský technický ústav, která tento problém řešila bez informačního systému pomocí mnoha Excel souborů, jež se posílaly mezi zaměstnanci firmy. Firma mi umožnila software nasadit a otestovat na uživatelích. Testování probíhalo ve více kolech, kde zaměstnanci již byli seznámeni s rozhraním systému. Pro otestování, zda je uživatelské rozhraní dostatečně intuitivní, jsem aplikaci dále testoval na rodinných příslušnících, kteří mají základní znalost ovládání PC a neměl by pro ně být problém s daným systémem pracovat.

5.1 Typy testů

Testování softwaru probíhá na několika úrovních, a to od testování separátních funkcí až po testování softwaru jako celku.

Unit testy jsou nejnižším typem testů, který testuje izolovaně části kódu a zbylé komponenty systému jsou mockovány (nahrazeny). Testuje se funkcionalita části kódu a korektnost volání ostatních částí systému. Mockování může být implementováno jen jako kontrola volání funkcí nebo může být implementováno modulem, který bude simulovat korektní chování daného modulu. Mockování zrychluje běh unit testů a umožňuje testovat jednotlivé moduly izolovaně.

Jakmile máme otestovány jednotlivé moduly aplikace separátně, tak je nutné ověřit, že moduly spolu komunikují korektně. K tomuto slouží integrační testy. V unit testech jsou mockované části systému nahrazeny reálnými implementacemi, které budou použity ve finální aplikaci.

Systémové testy představují nejvyšší úroveň testů, která má za cíl testovat kompletní systém. Úkolem je otestovat, zda systém plní požadavky a funguje jako celek.

Akceptační testy slouží pro ověření požadavků klientem daného systému a probíhají před nasazením do ostrého či testovacího provozu.

5.2 Testování serverové části

Testování serverové části je pomocí automatických testů implementovaných pomocí open source knihovny xUnit.net. V rámci architektury serverové části je každý projekt testován modulem stejného jména a příponou tests. Automatické testy jsou velice důležité pro

	Řádky kódu	Počet testů	Řádky kódu testů	Pokrytí v %
Repozitáře	1639	72	3075	91
Facades	845	38	1335	73

snadnou možnost refaktorizace a rozšiřování funkcionality, u kterých díky testům máme jistotu, že jsme nezměnili původní chování. Testy pro svůj běh potřebují pomocné funkce, instanci továren a mnoho dalšího. Tyto pomocné funkce zabalíme do třídy fixture, kterou pak pomocí interface IClassFixture zpřístupníme samotným testům.

5.2.1 Struktura testů

Každý test je strukturovaný do tří částí. První část Arrange slouží pro inicializaci objektů a dat, jež budou dále v testu využity. Část Act obsahuje zavolání testovaného modulu a provedení testované operace. Třetí část Assert obsahuje ověření výsledků získaných v části Act. Ověření probíhá na nižší úrovni abstrakce, než je testovaný modul. [2]

5.2.2 Integrační testy pro Data access layer

V rámci integračních testů pro databázovou vrstvu se napřed vytvoří prvotní fixture EvidentionDbContextSetupFixture, která zpřístupňuje továrnu na tvorbu databázového kontextu a funkce pro ověření korektního vytvoření a smazání databáze. V případě testů se nepřistupuje k SQL databázi, ale využívá se rozhraní IDbContextFactory, do kterého se dosadí továrna pro tvorbu databází v paměti. Jelikož zapouzdřujeme databázový kontext repozitáři, které nemohou provádět transakce bez unitOfWork, tak vytvoříme další fixture s názvem UnitOfWorkFixture, který bude dědit od EvidentionDbContextSetupFixture. V rámci unitOfWork zpřístupníme testům abstraktní továrnu na entity, komparátor přirozených čísel a továrnu na unitOfWork. UnitOfWork pro svůj běh vyžadují webHostEnvironment zpřístupňující cestu pro ukládání souborů a httpContextAccessor pro přístup k danému uživateli, který daný požadavek poslal. Tyto třídy v rámci fixture mockujeme a definujeme jim chování. UnitOfWorkFixture dále obsahuje metody pro seedování dat do databáze. Pro naseedování konkrétního typu záznamů slouží generická metoda SeedData, která bere typ entity. Pro složitější testy, kde je nutné otestovat korektní úpravu dat napříč různými typy záznamů, slouží metoda SeedDataWithHierarchy, jež vytvoří komplet strom záznamů s hierarchií a nasimuluje tak chod systému. Bázové funkce repozitářů jsou testovány v rámci třídy UnitOfWorkTests. Specifické metody a generické metody přepsané se specifickým chováním jsou testovány separátně pro každý repozitář.

5.2.3 Unit testy pro Business layer

Unit testy pro business layer slouží pro izolované testování fasád a konfigurací pro knihovnu Automapper. Testování knihovny Automapper má vlastní fixture, kde jsou inicializovány abstraktní továrny pro jednotlivé typy modelů a jsou nainicializována data pro přetypování. Unit testy pro fasády netestují korektní úpravu dat v databázi, ale pouze kontrolují, zda byly volány korektní metody. V těchto testech je potřeba veškeré závislosti nahradit mocky, které vytvoříme pomocí knihovny Moq. Každému mocku je potřeba nastavit chování pro všechny metody, jež jsou na něm v rámci testu volány. Volání těchto metod ověřujeme v sekci assert pomocí metody Verify nad mockem.

5.2.4 Integrační testy pro Business layer

Integrační testy již pracují s reálným unitOfWork a repozitáři, proto FacadeBaseTestsFixture bude dědit z UnitOfWorkTestsFixture a naše volání nad fasádami již budeme ověřovat nad unitOfWork, který je již otestovaný z integračních testů pro data access layer. V rámci FacadeBaseTestsFixture zpřístupníme abstraktní továrnu pro form modely, inicializujeme Mapper a abstraktní továrnu na fasády. Funkce pro seedování dat využijeme ze zděděného fixture.

5.2.5 Systémové testy serverové části aplikace

Systémové testy tvoří nejvyšší úroveň testů pro serverovou část. Budeme potřebovat továrnu na tvorbu instance aplikace. Továrna EvidentionApiApplicationFactory obsahuje metodu CreateHost, kde provádíme konfiguraci aplikace pro testy. Pro jednotný přístup k potřebným závislostem pro testy opět využijeme fixture. Fixture bude zpřístupňovat továrnu pro aplikaci a pro fasády, ovšem tentokrát budou fasády již vytvořeny pro SQL databázi namísto databáze v paměti. Testy testují klienta voláním url adres, čímž simulujeme reálné volání serverové části z klienta. Po volání kontrolujeme návratový kód http a data ověřujeme vůči fasádě.

5.3 Testování frontendu

K testování frontendu byl využit diagram případů užití. Frontend byl testován ručně a byly zkoušeny různé scénáře, včetně vkládání kompletních sestav a generování výsledných dokumentů. Díky vkládání reálných dat do reálně nasazené aplikace byly odhaleny chyby v návrhu uživatelského rozhraní, které se povedlo opravit. Jednalo se hlavně o neperzistentní nastavení tabulek a stránkování při přechodu mezi pohledy nebo při aktualizaci záznamů přímo z tabulky. Byly objeveny nové náměty na pozdější rozšíření aplikace a další urychlení práce zaměstnanců.

Kapitola 6

Nasazení aplikace

Aplikace je hostována v aplikaci IIS na virtuálním stroji v interní síti. Z důvodu použití websocketu je minimální verze IIS 8.0, která vyžaduje minimální verzi operačního systému Windows 8 nebo Windows Server 2012. V rámci IIS v sekci weby přidáme nový web. Název dáme Records.API a specifikujeme cestu k buildu aplikace. Port pro serverovou část zadáme 8080. Pro správný chod musíme nainstalovat dotnet-hosting pro verzi .NET Core 3.1 a dále musíme nainstalovat rewrite a cors modul pro IIS. V konfiguraci serverové části v sekci ověřování zapneme ověřování systému Windows pro requesty a anonymní ověření pro přístup ke statickým souborům. Dále v editoru konfigurací nastavíme proměnnou prostředí s klíčem db_Evidence, která obsahuje connection string do databáze. Tyto změny se zapíší do souboru web.config. Dále je nutné pro složku wwwroot pro skupinu IIS_IUSRS povolit práva úpravy a zápisu dat z důvodu ukládání příloh a obrázků na server. Pro správné fungování komunikace mezi klienty je na serveru nutné zapnout v sekci funkce systému Windows podporu websocketu a pro ověřování pomocí Windows účtu je ve funkcích nutné zapnout Windows authentication. Dále je do IIS nutné nahrát SSL certifikát a zapnout https komunikaci.

Pro klienta vytvoříme nový web s názvem Records, specifikujeme cestu k buildu a jako port dáme výchozí port pro HTTP 80. Pro správný chod jednostránkové aplikace je nutné přidat web.config s definicí pro rewrite modul, jinak by po přenačtení stránky mimo domovskou stránku došlo k chybě 404, stránka nenalezena. Dále nastavíme přesměrování na HTTPS v rámci web.config. Dále je nutné složkám se sestavenými aplikacemi přes vlastnosti v sekci zabezpečení přidat oprávnění přístupu skupině IIS_IUSRS v případě, že se nenachází ve složce inetpub/www. Pro zjednodušení nahrávání nové verze vznikl script, který přehraje data pro klienta a server z předdefinovaných složek. Součástí přehrání je zachování původních konfigurací a wwwrootu u serverové části.

Kapitola 7

Závěr

Cílem této bakalářské práce bylo vytvořit webový informační systém, jehož položky je možné konfigurovat pomocí JSON souborů a korektně řeší vícenásobný přístup k záznamům. V rámci systému jsou uživatelé autorizováni pomocí Windows Authentication. Každá úprava dat je zaznamenána, takže jsme schopni zobrazit stav systému pro určité datum a čas.

Při tvorbě systému bylo dle mého názoru nejsložitější vhodně zvolit architekturu. Prvotně aplikace vznikala pouze s použitím C# ASP.NET Core v kombinaci s Razor Pages na klientské části aplikace. Toto řešení vedlo na standardní dynamický web. Při zavolání koncového bodu se na serveru počkalo na data z databáze, následně se sestavil pohled a ten se poslal na klienta, což vedlo k tuhnutí aplikace. Tento problém jsem vyřešil posláním pohledu bez dat s tím, že pohled si následně pomocí technologie AJAX dostahoval data asynchronně a při stahování systém zobrazoval načítací animaci. Tohle řešení již obsahovalo většinu kódu v technologii javascript na klientské části aplikace. Celkově parciální řešení pomocí Razor pages nesplnilo požadované nároky na rychlost odezvy aplikace. Z tohoto důvodu jsem začal hledat modernější alternativu v podobě reaktivních javascriptových frameworků, které mi nabídly vše, co jsem k vývoji potřeboval. Při použití Razor Pages stejně bylo nutné přidat javascript, aby web interagoval s uživatelem, tak proč již nedat javascript do popředí.

V rámci bakalářské práce jsem se nejprve věnoval teoretickému úvodu do problematiky a posléze i návrhu řešení informačního systému. Dále jsem navržený informační systém implementoval a otestoval. Po úspěšném otestováním došlo k jeho nasazení.

Cíl bakalářské práce se dle mého názoru podařilo splnit, jelikož aplikace byla nasazena v rámci firemní sítě a otestována uživateli. Aplikace byla pravidelně konzultována jak s uživateli, tak s vedením firmy a náměty byly iterativně zapracovávány do řešení. V budoucnu je plánováno stávající systém rozšířit o další moduly pro začlenění více pracovních procesů do informačního systému.

Díky této práci jsem se naučil moderní technologie jako je ASP.NET a javascriptový framework React, které jsou reálně aplikované a nasazované v praxi. Za největší přínos beru právě zkušenost s návrhem architektury a strukturování rozsáhlejšího projektu. V rámci informačního systému jsem si prakticky vyzkoušel mnoho znalostí, které jsem v rámci bakalářského programu získal.

Tématu tvorby generických šablon pro informační systémy bych se chtěl dále věnovat a rozšiřovat získané znalosti v návrhu komplexních systémů.

Literatura

[1] *Flask web development, one drop at a time* [online]. [cit. 2022-06-03]. Dostupné z: https://flask.palletsprojects.com/en/2.0.x/.

[2] *Unit Test Frameworks* [online]. 2006. ISBN 0596006896. Dostupné z: https://flylib.com/books/en/1.104.1.23/1/.

[3] *Spring Framework* [online]. 2019 [cit. 2022-06-03]. Dostupné z: https://spring.io/why-spring.

[4] *ASP.NET Core* [online]. 2021 [cit. 2022-06-03]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core.

[5] *JQuery* [online]. 2021 [cit. 2022-24-03]. Dostupné z: https://jquery.com/.

[6] *The MVT Design Pattern of Django* [online]. 2021 [cit. 2022-06-03]. Dostupné z: https://python.plainenglish.io/the-mvt-design-pattern-of-django-8fd47c61f582.

[7] *An overview of HTTP* [online]. 2021 [cit. 2021-22-09]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

[8] *Přednášky předmětu IIS, FIT VUT v Brně* [online]. 2021 [cit. 2022-22-04].

[9] *Web application* [online]. 2021 [cit. 2021-22-09]. Dostupné z: https://en.wikipedia.org/wiki/Web_application.

[10] *Angular (web framework)* [online]. 2022 [cit. 2022-23-03]. Dostupné z: https://angular.io/docs.

[11] *Cross-Origin Resource Sharing (CORS)* [online]. 2022 [cit. 2022-23-03]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

[12] *Django introduction* [online]. 2022 [cit. 2022-06-03]. Dostupné z: https://www.djangoproject.com/start/overview/.

[13] *Express.js* [online]. 2022 [cit. 2022-06-03]. Dostupné z: https://expressjs.com/.

[14] *Injektáž závislostí v ASP.NET Core* [online]. 2022 [cit. 2022-22-03]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/fundamentals/dependency-injection.

[15] *Prohlídka jazyka C* [online]. 2022 [cit. 2022-06-03]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/csharp/tour-of-csharp/.

[16] *Práce s DbContext* [online]. 2022 [cit. 2022-21-03]. Dostupné z: https://docs.microsoft.com/cs-cz/ef/ef6/fundamentals/working-with-dbcontext.

[17] *React (JavaScript library)* [online]. 2022 [cit. 2022-23-03]. Dostupné z: https://reactjs.org/docs.

[18] *Ruby (programming language)* [online]. 2022 [cit. 2022-06-03]. Dostupné z: https://www.ruby-lang.org/en/about.

[19] *Svelte* [online]. 2022 [cit. 2022-24-03]. Dostupné z: https://svelte.dev.

[20] *Vue* [online]. 2022 [cit. 2022-24-03]. Dostupné z: https://vuejs.org/guide/introduction.html.

[21] Cameau, J. *Persisting React State in localStorage*. 2020 [cit. 2022-22-04]. Dostupné z: https://www.joshwcomeau.com/react/persisting-react-state-in-localstorage.

[22] Gibb, R. *What is a Web Application?* [online]. 2016 [cit. 2021-22-09]. Dostupné z: https://blog.stackpath.com/web-application/.

[23] Sodomka, P. *Informační systémy v podnikové praxi*. 1. vyd. Academia, 2006. ISBN 80-251-1200-4.

Příloha A

Slovník použitých zkratek

AJAX Asynchronous JavaScript and XML. 9, 40

API Application program interface. 7, 8, 17, 27

BL Business layer. 8

CRUD Create, Read, Update, Delete. 27

DAL Data access layer. 8, 15

ER Entity Relationship. 15

HTTP Hypertext Transfer Protocol. 6, 7, 15, 17, 27, 39

IIS Internet Information Services. 39

JIT Just in Time. 9

JSON JavaScript Object Notation. 8, 18, 24, 29, 40

LINQ Language Integrated Query. 20

MVC Model, View, Controller. 8, 9

MVT Model, View, Template. 8

PHP Hypertext Preprocessor. 9

REST Representational State Transfer. 10, 15, 24

SPA Single Page application. 16, 18

SQL Structured Query Language. 10, 20

SSL Secure Sockets Layer. 39

TCP Transmission Control Protocol. 6

UML Unified Modeling Language. 1, 12, 13

URL Uniform Resource Locator. 6