Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Webová aplikace pro analýzu firemních faktur

Andrii Andrusenko

Bakalářská práce
2022

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Andrii Andrusenko**
Osobní číslo:	**I18102**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Webová aplikace pro analýzu firemních faktur**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je vyvinout webovou aplikaci pro vyhledávání a analýzu firemních faktur. Vyhledávání bude probíhat na základě vstupních textových souborů obsahujících důležitá data o fakturách a na základě vyhledání faktur ve vnitřní databázi nebo v souborovém systému budou poskytovány výstupní detailní informace – rozdíly mezi vstupními daty a daty uloženými v databázi/souborovém systému

Předpokladem je:

- ověřování uživatelů pomocí Azure Active Directory
- výstupní data budou poskytována v txt/csv souborech
- aplikace bude vytvořena pomocí technologií Spring Boot, Hibernate, React

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Hands-On Full Stack Development with Spring Boot 2 and React: Build modern and scalable full stack applications using Spring Framework 5 and React with Hooks, 2nd Edition. Packt Publishing, 2019, 316 pp. ISBN 978-1838822361.
2. Learning React: A Hands-On Guide to Building Web Applications Using React and Redux, 2nd edition. Addison-Wesley Professional, 2018, 304 pp. ISBN 978-0134843551.

Vedoucí bakalářské práce: **doc. Ing. Michael Bažant, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **17. prosince 2021**
Termín odevzdání bakalářské práce: **13. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2022

PROHLÁŠENÍ

Prohlašuji:

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

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

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 7. 5. 2022

Andrii Andrusenko

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat svému vedoucímu doc. Ing. Michaelu Bažantu, Ph.D. za pomoc v zpracováni této bakalářské práce a odborní podporu. Dekuji i všem kdo pomáhal a inspiroval mě.

ANOTACE

Tato bakalářská práce je zaměřena na vývoj webové aplikace, sloužící k analýze firemních faktur přijatých v různých formátech (txt, csv atd.) a z různých zdrojů (DB, SFTP atd.). Aplikace je vyvinuta jako SPA a skládá se ze dvou nezávisle běžících části: backend a frontend.

Hlavní technologie použité pro vývoj této webové aplikace jsou Spring Boot a React z Redux. Backend je vytvořen pomocí Spring Boot, Hibernate a používá se jako REST API pro zpracování, ukládání a poskytování dat na základě požadavku z frontendu. Frontend byl vyvinut jako SPA pomocí React a Redux. Ověřování a autorizace uživatelů se provádí pomocí Azure Active Directory.

KLÍČOVÁ SLOVA

Webová aplikace, firemní faktura, migrace, backend, frontend, SPA, Spring Boot, React, Redux, Azure Active Directory, DB, SFTP, Java.

TITLE

Web application for analysis of firm invoices.

ANNOTATION

This bachelor thesis is focused on developing the web application which is used for analysis of firm invoices received in different formats (txt, csv etc.) and from different sources (DB, SFTP etc.). The application is developed as an SPA, and it consist of two independently running parts: backend and frontend.

The main technologies used for development of this web application are Spring Boot and React with Redux. Backend is created with help of Spring Boot, Hibernate and is operating as a REST API for processing, storing, and providing data by request from frontend. Frontend was developed as an SPA using React and Redux. User authentication and authorization is done by Azure Active Directory.

KEYWORDS

Web application, firm invoice, migration, SPA, Spring Boot, React, Redux, Azure Active Directory, DB, SFTP, Java.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

AA	Authentication Authorization
AAD	Azure Active Directory
API	Application Programming Interface
CLI	Command Line Interface
CORS	Cross-origin Resource Sharing
CSS	Cascading Style Sheets
CSV	Comma-separated values
DB	Database
FS	File System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JDK	Java Development Kit
JPA	Java Persistence API
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
JSX	JavaScript XML
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MSAL	Microsoft Identification Library
MUI	Material User Interface
ORM	Object-Relational Mapping
REST	Representational State Transfer
SFTP	SSH File Transfer Protocol
SPA	Single Page Application
SSH	Secure Shell
SW	Software
TXT	Text file
UI	User Interface
URI	Uniform Resource Identifier
VPN	Virtual Private Network
XML	Extensible Markup Language

ÚVOD

V dnešní době každý podnik řeší problematiků změny SW. Řešená aplikace slouží pro kontrolu kvality migračního procesu. Tato bakalářská práce popisuje obecný design, požadavky a technologie používané pro aplikaci a jak je strukturována, testována a následně nasazena na server.

Aplikace byla vyvinuta jako interní nástroj pro hromadné porovnávání různých faktur generovaných různými projekty. Používá se při migraci z jednoho projektu do druhého. A jejím hlavním účelem je automatizovat proces porovnávání a vytvářet výstupní výsledek s rozdíly porovnávaných souborů. Podporuje několik různých formátů faktur a používá různé konfigurace ve formě JSON ke spuštění srovnání. Tato konfigurace umožňuje ignorovat různá pole řádků faktury v závislosti na formátu faktury. Protože tato aplikace pracuje s citlivými informacemi, jako poskytovatel AA se použije Azure Active Directory. Uživatelský přístup bude poskytován prostřednictvím webového uživatelského rozhraní, které bude nasazeno na interním serveru.

Cílem bakalářské práce je návrh a vývoj webové aplikace pro porovnání hromad firemních faktur v různých formátech a poskytovaných z různých zdrojů. K dosažení tohoto cíle je potřeba zpracovat několik úkolu:

1. Sepsat funkční a nefunkční požadavky na aplikace.
2. Navrhnout aplikaci.
3. Zvolit vhodné technologie a frameworky.
4. Poskytnout strukturu API backendu a návrh UI.
5. Vyvíjet backend a frontend a otestovat je.

1 POŽADAVKY A ANALÝZA APLIKACE

Vzhledem k rychlému růstu příchozích denních transakcí bylo přijato rozhodnutí vytvořit nový framework, který by to podporoval. Aplikace vytvořená během této bakalářské práce je určena pro zajištění bezchybného přechodu mezi frameworky. Hlavním důvodem bylo, aby si zákazník ani nevšimnul, že k převodu došlo. Tato aplikace je testovací nástroj zaměřený pouze na interní použití.

Hlavním účelem této aplikace je porovnat soubory faktur původního frameworku s novým. Faktury jsou poskytovány ve formátu TXT nebo XML. Jeden soubor může obsahovat jednu nebo více faktur. Všechny faktury vytvořené novým frameworkem jsou umístěny na FS serveru, který je vytvořil, z důvodu omezení architektury serveru a právních důvodů.

Legálně by všechny soubory měly být umístěny pouze na serveru FS nebo oficiálním serveru SFTP. Takže všechny soubory použité pro porovnání a soubor JSON s výsledky porovnání budou nahrány buď na FS nebo na SFTP. Soubor s výsledky porovnání musí obsahovat všechny zjištěné rozdíly mezi fakturami a také jejich přesné umístění – číslo řádku a pozici. Kvůli tomuto omezení bylo rozhodnuto nepovolit nahrávání souborů z uživatelského rozhraní SPA.

Hlavním požadavkem aplikace je porovnat soubor obsahující faktury (dávkový soubor faktur), který byl vytvořen původním frameworkem, s podobným souborem vytvořeným novým frameworkem. Původní soubory jsou poskytovány pomocí serveru FS nebo serveru SFTP. Vytvořené soubory jsou uloženy pouze na FS serveru, který je vytvořil. Aplikace by tedy měla umět:

- Příjem souborů z FS a SFTP serveru.
- Musí být schopen se připojit k poskytnuté DB.
- Musí mít uživatelské rozhraní.
- Nesmí měnit obsah vytvářených souborů a měnit stav frameworkové DB.
- Musí podporovat všechny formáty faktur.

Vzhledem k povaze frameworku, formátů nebo poskytovatelů souborů se mohou časem měnit, aplikace musí podporovat změny bez větších předělávek a s minimální dobou vývoje, aby se snížily náklady. Pokud je to možné, každá součást by měla být modulární. Je třeba se vyhnout těsnému spojení komponentu.

Aplikace by také měla podporovat uživatelskou konfiguraci pro srovnání. Tato konfigurace bude ve formátu souboru JSON a umožní například ignorovat některá pole nebo řádky při porovnávání.

Hlavními uživateli této aplikace budou interní pracovníci (konfigurátory a analytici), kteří konfigurují nové formáty nebo mění již existující.

Při vývoji teto aplikací je také třeba vzít v úvahu omezení serverového SW. Existuje zákonné omezení, po kterému SW lze instalovat na server. Byly tedy použity pouze již podporované technologie: Java 8, Node.js.

Data by měla být chráněna před neoprávněným přístupem i v případě, že aplikace bude spuštěna v lokální chráněné síti přístupné pomocí VPN klienta. Možné systémy AA, které by mohly být použity, jsou LDAP a AAD.

1.1 Funkční a nefunkční požadavky

Funkční požadavky:

1. Mass Compare Tool bude porovnávat soubory v režimu DB a SFTP.
2. Mass Compare Tool bude porovnávat všichni poskytované referenční a zpracované soubory faktur.
3. Mass Compare Tool bude parsovát a porovnávat soubory faktur následujících formátu: GLOBALEX, FFINV1, FFINV3, FF_KPMG, FF_PWC, ANSI_X12, GENERIC_FF, RBC_CA_NOENV, XML, EDIFACT.
4. Mass Compare Tool bude ukládat do DB uživatelsky vytvořené porovnávací konfigurace.
5. Mass Compare Tool bude ukládat do DB výsledky porovnání.
6. Mass Compare Tool bude filtrovat uživatelsky vytvořené porovnávací konfigurace podle formátu a režimu provádění.
7. Mass Compare Tool bude filtrovat vygenerované výsledky podle názvu a data.
8. Mass Compare Tool bude zobrazovat upozornění pro uživatele.
9. Mass Compare Tool bude poskytovat výsledky porovnání v TXT nebo CSV formátu.
10. Mass Compare Tool bude validovat uvedena uživatelem data.

Nefunkční požadavky:

1. Mass Compare Tool bude vytvořen ze dvou části: backend a frontend.
2. Backend pro Mass Compare Tool bude vytvořen pomoci Java 8, Spring Boot, Hibernate.
3. Frontend pro Mass Compare Tool bude vytvořen pomoci JavaScript, React, Redux.
4. Mass Compare Tool bude používat AAD pro AA.
5. Mass Compare Tool bude podporovat přidáváni nových modulu pro porovnání faktur.
6. Mass Compare Tool bude přistupovat ke zpracovaným datovým souborům v režimu pouze pro čtení.
7. Mass Compare Tool bude podporovat uživatelsky poskytované konfigurační soubory pro porovnání.

1.2 Formáty faktur

Formáty představují jiný typ souboru faktur, který vytváří framework. Zákazník je pak použije k analýze a rozboru faktury. Příklad formátů podporovaných aplikací: ANSI_X12, XML, FFINV3, GENERIC_FF a další.

Každý řádek ANSI_X12 představuje jiný segment faktury a začíná identifikátorem segmentu (viz. Obrázek 1). Formát GENERIC_FF je zobecněný formát pro všechny faktury, řádky, kterého nemají segmenty, trailery nebo záhlaví. Každý řádek formátu GENERIC_FF představuje jiné objednávku.

Některé formáty podporují batching – když jeden soubor obsahuje více faktur. To je třeba vzít v úvahu při vývoji nových porovnávacích modulů pro tuto aplikaci.

```
ISA*00*          *00*          *02*               *ZZ*               *200807*0920*U*00401*000000040*1*T*^
GS*IA*AAA*BB*20200807*0920*000000040*X*004010
ST*110*000000040
B3**LPLIXXXXXX**MX**20180409*4937****XXXX*20200807
B3A*21*1
C3*GBP*1.000*GBP
ITD*05*3****20180509*30
N1*BT*COMPANY_NAME_1*25*111111111
N3*ADDRESS_STR_1*ADDRESS_DESC_1
N4*CITY_1**IND_1*GB*CC*GB
N9*AC*111111111
PER*CN*Name_1
LX*1
N1*SH*COMPANY_NAME_2*25*111111111
N3*ADDRESS_STR_2*ADDRESS_DESC_2
N4*CITY_2**IND_2*TR
N9*AW*222222222
PER*SH*Name_2
N1*CN*COMPANY_NAME_3
N3*ADDRESS_STR_3*ADDRESS_DESC_3
N4*CITY_3**IND_3*GB
PER*DC*Name_3
P1**20180404*011****1
R1*XXXX**TEQ*XXX*LON
L5*1
L0*1*2.50*K*2.50*A2***1*PCS**K
SL1*CX*CX**IMP1  /***PP**I
L1*1***9940****400****EXPRESS WORLDWIDE nondoc*******0
C3*GBP*1.000*GBP
L1*2***-5666****DSC****DISCOUNT CODE W*******0
C3*GBP*1.000*GBP
L1*3***0****750****Non-taxable**4274*PS****0.0
C3*GBP*1.000*GBP
L1*4***1541****405****FF/FUEL SURCHARGE*******0
C3*GBP*1.000*GBP
L1*5***-878****DSC****DISCOUNT CODE W*******0
C3*GBP*1.000*GBP
L1*6***0****750****Non-taxable**663*PS****0.0
C3*GBP*1.000*GBP
L3*2.5*B***4937******1*K
SE*40*000000040
GE*1*000000040
IEA*1*000000040
```

Obrázek 1: Příklad faktury formátu ANSI_X12. Zdroj: vlastní

1.3 Srovnávací režimy

Tato aplikace má běžet ve dvou různých režimech: DB a SFTP. Každý režim má svého poskytovatele souborů. Poskytovatelé představují metodu, kterou jsou soubory přijímány. V současné době existují dva typy: poskytovatel DB a SFTP.

DB režim:

1. Soubory, které je třeba porovnat, se nahrávají přímo do FS serveru, na kterém běží aplikace.
2. Každý referenční soubor je poté analyzován a jeho ID je extrahováno.

3. ID se pak používá k příjmu dat vytvořeného souboru z DB a souboru z FS a provedení porovnání.

SFTP režim:

1. Všechny soubory jsou poskytovány pomocí serveru SFTP.
2. Každý referenční soubor z dané cesty je analyzován, jeho ID je extrahováno.
3. ID se pak používají k vyhledávání zpracovaných souborů na SFTP v konkrétní složce podle formátu.

1.4 Konfigurační soubory

Konfigurační soubory se používají při spouštění porovnání pro daný formát. Použije se buď výchozí, nebo poskytnutý konfigurační soubor porovnání. Pomocí konfiguračního souboru lze nastavit různé porovnávací parametry. Tyto parametry se liší v závislosti na formátu souboru.

Příklad obsahu souboru parametrů JSON pro formát GLOBAL_FF:

```
{
 "ignoredFields": [ ],  "compareNumericFields": [ ],
  "fieldSeparator": "\t",  "hasHeader": false,  "invoiceIdPosition": 0
 }
```

ignoredFiels – parametr, který představuje seznam segmentů, které musí být na daném řádku ignorovány. Například obsahuje datum vystavení faktury, protože na dvou fakturách bude vždy jiný.

compareNumericFields – parametr, který představuje seznam segmentů s číselnými hodnotami, které mají různé formátování na referenční a vyrobené faktuře. Tento segment má obvykle stejnou hodnotu, ale různé formátování v závislosti na požadavcích v době vytvoření souboru.

fieldSeparator – parametr, který představuje znak oddělovače segmentů, který se používá k analýze souboru faktury.

hasHeader – parametr, který má booleovskou hodnotu (true nebo false) a používá se během analýzy k rozhodnutí, zda má faktura globální řádek záhlaví.

invoiceIdPosition – parametr, který má hodnotu počáteční pozice čísla faktury v řádku faktury.

2 POUŽITÉ TECHNOLOGIE

V této sekci jsou uvedeny všechny vývojové jazyky, technologie a některé balíčky použité při vývoji této webové aplikace.

Aplikace je designově rozdělena na dvě samostatné části: backend a frontend. Tyto části spolu komunikují pomocí HTTP požadavků (GET, POST, PUT, DELETE, OPTION). Tento design byl zvolen proto, aby se uvolnilo spojení mezi dvěma částmi a umožnilo v budoucnu větší modularitu a škálovatelnost.

2.1 Backend

Backend pro tuto aplikaci byl vyvinut pomocí Java 8 a frameworku Spring Boot. Bylo to uděláno jako REST API, které může přijímat externí požadavky HTTP a jako odpověď odesílá objekt JSON. Hlavním požadavkem backendu bylo, že by měl být modulární a neměl by být závislý na technologii zvoleného uživatelského rozhraní.

2.1.1 Java 8

Hlavní technologií použitou pro vývoj backendu pro tuto aplikaci byl jazyk Java 8. Java je dnes jedním z nejpoužívanějších vývojových jazyků. Poprvé byl vyvinut v roce 1990 Jamesem Goslingem ve společnosti Sun Microsystems a poté publikován v roce 1995 společností Sun Microsystems.

Java se používá k vytváření široké škály aplikací od desktopů po web. JDK je potřeba pro vývoj Java aplikací, protože obsahuje všechny potřebné nástroje. Ke spuštění Java aplikace je potřeba JRE, které obsahuje JVM, základní třídy a knihovny.

Nejcennější vlastnosti Javy jsou bezpečnost, stabilita a vysoký výkon. Java je přenosný, objektově orientovaný jazyk nezávislý na platformě. Aplikace poběží stejně na jakémkoli systému, protože kompilované Java soubory jsou pak interpretovány pomocí JVM [1].

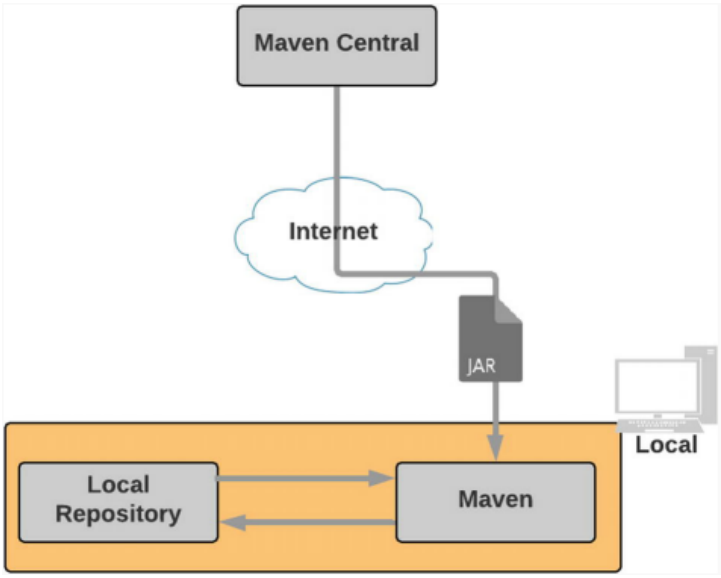
Java je hybridní jazyk. Je kompilován i interpretován. Nejprve jsou lidmi psané soubory s příponou .java zkompilovány do souborů .class obsahujících přechodný bajt kód pomocí kompilátoru javac, které pak spouští JVM [2].

Paměť v Javě je rozdělena na dvě části zásobník vláken a haldu. Zásobník obsahuje místní proměnné a kontext provádění vlákna, zatímco halda obsahuje všechny inicializované objekty. Garbage Collector se používá ke správě životního cyklu objektu a využití paměti haldy [3].

Podporuje vývoj vysoce efektivních souběžných aplikací pomocí nástrojů, které poskytuje. Java používá vlákna, synchronizaci a různé atomické proměnné ke zvýšení využití systémových zdrojů a pohodlnějšímu souběžnému vývoji [4].

2.1.2 Apache Maven

Apache Maven je jedním z nejdůležitějších nástrojů při vývoji Java aplikace. Poprvé byl vydán v roce 2004. Aktuální verze Maven je 3.0 z roku 2010. Jedná se o open-source framework, který poskytuje sestavení, závislosti a správu pluginů. Maven je globálně instalovaný nástroj, který je závislý na systému. Tento nástroj poskytuje standardní deklarativní způsob řízení projektů.



Obrázek 2: Správa závislostí Apache Maven. Zdroj: [5]

Hlavním bodem Apache Maven je, že přináší do projektu standardizovanou strukturu složek. Například všechny Java soubory projektu jsou uloženy ve složce: ***/src/main/java***. Existují různé typy těchto složek, které lze ručně nakonfigurovat: java, test, zdroje a další.

Konfiguruje se pomocí souboru **settings.xml** a může používat různé repozitáře maven k vyhledávání a stahování různých knihoven. Podporuje také proxy a šifrování hesel. Obecná

konfigurace závislostí, pluginů, cílů, sestavení se provádí v konfiguračním souboru **pom.xml** [5].

Maven poskytuje standardizovaný životní cyklus sestavení, který se skládá z různých fází: validate, compile, test, package, verify, install, deploy. Každá z těchto fází může být volána pomocí: *mvn <název_fáze> [6]*.

Pluginy Maven slouží k úpravě a změně jeho funkčnosti. Lze je použít k implementaci pracovního postupu projektu: vytváření, kompilace a další. Obvykle jsou vytvořeny v jazyce Java [7].

2.1.3 Spring Boot

Druhou hlavní technologií použitou pro vývoj backendu pro tuto aplikaci je framework Spring Boot.

Je to jeden z nejrozšířenějších Java open-source frameworků, který spravuje Pivotal. Pomocí něj lze vytvářet různé webové aplikace. Podporuje mnoho různých knihoven, které přidávají podporu pro další funkce. Spring Boot je nyní široce používán k vytváření REST služeb, které lze snadno škálovat pomocí cloudových technologií.

Spring Boot poskytuje platformu, kterou lze použít ke snadnému vytváření koncových bodů REST. Nevyžaduje mnoho konfigurace, téměř vše se konfiguruje automaticky nebo pomocí nějakých globálních anotací. Pro základní aplikaci obvykle stačí použít anotace @SpringBootApplication k předkonfigurování všech nezbytných závislostí [8].

Hlavní výhodou Spring Boot je, že nabízí prostředí, které výrazně zkracuje dobu vývoje. Aplikace Spring Boot se spouští ve vlastní předkonfigurované instanci HTTP serveru Tomcat nebo Jetty.

Poskytuje efektivní a snadný způsob vytváření a správy Beanu různých typů pomocí anotací: @Component, @Service, @Repository a další. Pomocí anotace @Autowired lze tyto Beany vložit pomocí pole, setteru nebo konstruktoru.

Projekty Spring Boot se obvykle vytvářejí a inicializují pomocí webové aplikace: *https://start.spring.io/* [9].

2.1.4 Spring Security

Spring Security je framework, který poskytuje ověřování a řízení přístupu. Jedná se o standardizovaný přístup pro zabezpečení aplikací Spring. Ve výchozím nastavení má ochranu proti většině moderních hrozeb a bere v úvahu i nefunkční požadavky.

Poskytuje rozhraní API, které vám umožňuje konfigurovat všechny potřebné nástroje zabezpečení, oprávnění a rozsahy. Lze jej použít k ochraně REST API a webových aplikací pomocí různých poskytovatelů autentizace: LDAP, AAD a další [10].

2.1.5 Hibernate

Hibernate je open-source implementace JPA, která poskytuje ORM. Jeho hlavním cílem je správa perzistence dat. Perzistence znamená, že daný objekt žije, i když je aplikace vypnutá. Hlavním účelem je, aby objekty byly vždy v konzistentním stavu, uložené v některých storage solutions, například DB.

JPA poskytuje anotační API, které umožňuje mapování DB tabulek na java objekty, nazývané „entity“. Hlavním účelem ORM je transformace DB objektů na java objekty a zpět.

Výhody Hibernate:

- Produktivita – Hhibernate snižuje množství standardního kódu a zkracuje dobu vývoje.
- Výkon – Hibernate umožňuje různé způsoby ladění výkonu DB komunikace a perzistence objektů.
- Nezávislost na poskytované DB – Hibernate ORM podporuje řadu různých řešení DBMS. To má za následek snadnější přepínání mezi databázemi. Například použití úložišť‘ do operačně paměti během testování a běžné DB v produkci [11].

2.1.6 REST API

Backend projektu je navržen pomocí konceptu REST API. Data jsou přenášena pomocí HTTP ve formátu: JSON, HTML, XLT, Python, PHP nebo prostý text. Formát JSON je obecně nejpoužívanějším formátem.

Hlavičky HTTP požadavků a odpovědí jsou důležité při komunikaci s REST API, protože obsahují potřebné nebo dodatečné informace o datech, ukládání do mezipaměti, autentizaci atd. HTTP požadavek může být různého typu:

- GET – přijímá data, obvykle nemá tělo a používá se pro operace pouze pro čtení.
- POST – vytváření dat, má tělo.
- PUT – aktualizace dat, má tělo.
- DELETE – smazání dat, obvykle nemá tělo [12].

API se nazývá restful, když se řídí architektonickými koncepty a omezeními REST:

- Klient-server architektura.
- Bezstavová komunikace klient-server – na serveru nejsou uložena žádná data klienta.
- Data uložitelná do mezipaměti (anglicky cache).
- Jasné a jednotné rozhraní mezi komponenty.
- Informace jsou přenášeny ve standardní formě.
- Různé typy serverů, z nichž každý je zodpovědný za jednu akci [13].

2.2 Frontend

Frontend je druhou hlavní částí této aplikace. Poskytuje uživateli přístup k datům. Byl vyvinut jako SPA.

Hlavním důvodem, proč používat SPA, je to, že neposkytuje žádné opětovné načítání stránky ani další čekací dobu. Tento typ stránky silně závisí na JavaScriptu a je vykreslován v prohlížeči. Když otevřete tento druh stránky, načte se všechny soubory, které mají veškerou logiku, kterou potřebuje ke správnému vykreslení všech komponentu.

Výhody SPA:

- Rychlejší přechod stránek – všechny potřebné zdroje pro tuto aplikaci se načtou pouze jednou.
- SPA se snadněji ladí pomocí vestavěných ladicích nástrojů prohlížeče.
- Ukládání do mezipaměti je mnohem efektivnější.
- Snazší vývoj díky různým frameworkům: AngularJS, React, Vue.js atd.

Nevýhody SPA:

- Vyžaduje, aby byl v prohlížeči povolen JavaScript.

- SPA jsou méně bezpečné.
- SPA využívají více hardwarových systémových prostředků.
- Úniky paměti mohou vést ke zpomalení systému.
- Vyvažování asynchronní povahy načítání dat vyžaduje více znalostí [14].

2.2.1 JavaScript

Toto je hlavní jazyk používaný pro vývoj uživatelského rozhraní pro tuto aplikaci.

JavaScript je standardizovaný skriptovací jazyk, který musí implementovat každý prohlížeč. Poprvé se objevil v roce 1995 v prohlížeči Netscape Navigator. Jeho implementace je regulována dokumentem ECMAScript, který je publikován společností Ecma Interanational. JavaScript umožňuje změnu obsahu webové stránky bez opětovného načítání stránky po každé akci.

JavaScript podporuje osm hlavních datových typů:

- String – textová hodnota.
- Number – celé číslo nebo číslo s plovoucí desetinnou čárkou.
- BigInt – celé číslo s libovolnou přesností.
- Boolean – true nebo false hodnota.
- Undefined – představuje neinicializovanou proměnnou.
- Null – představuje nulovou hodnotu.
- Symbol – neměnný textový objekt.
- Object – objekt je reprezentován kolekcí párů klíč-hodnota.

Proměnné JavaScriptu mohou mít tři rozsahy:

- Var – proměnná je lokální funkci nebo je globálně přístupná. Proměnné Var lze znovu deklarovat a jejich hodnotu lze změnit.
- Let – má blokový rozsah. Blok je reprezentován {}. Může být aktualizován, ale nelze jej znovu deklarovat.
- Const – má blokový rozsah. Nelze to změnit a znovu deklarovat.

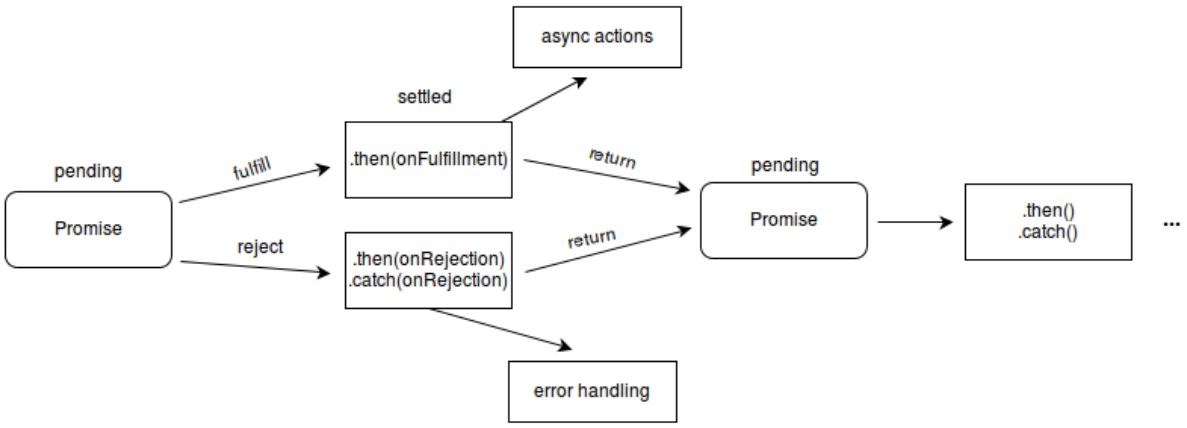
Jedním z hlavních negativních bodů JavaScriptu je to, že kvůli jeho dynamicky typované povaze je velmi obtížné jej během vývoje ladit. Ladění v JavaScriptových aplikacích se primárně provádí pomocí funkce *console.log()*. V JavaScriptu mohou být funkce uloženy uvnitř

proměnných, tento druh funkcí se nazývá zpětná volání. Jejich hlavním použitím je spuštění nějaké akce, například při stisknutí tlačítka [15].

JavaScript podporuje funkcionalitu Promise. Promise se používají, když potřebujete změnit asynchronní akci na synchronní. Může být ve třech stavech:

- Pending – výchozí stav.
- Fulfilled – stav, kdy byla operace dokončena.
- Rejected – stav, kdy operace selhala.

JavaScript má také snazší čekání na přeměnu asynchronního na synchronní pomocí funkce async/await. Každá funkce, která je označena jako async, vrací Promise, který lze očekávat [16].



Obrázek 3: Změna stavu Promise. Zdroj: [16]

2.2.2 HTML

HyperText Markup Language – poskytuje sadu stavebních bloků pro web. Představuje obsah a strukturu webové stránky. HTML je globální jazyk, který používá každý prohlížeč k vykreslování obsahu. Implementace různých prvků se může lišit v závislosti na prohlížeči.

HTML používá „markup“ k zobrazení textu, obrázků, odkazů a dalších pro uživatele. HTML-markup se skládá z různých značek: *<head>, <body>, * a dalších.

Pomocí HTML lze vytvářet datové tabulky. Podporuje také formuláře, které se používají pro různé uživatelské interakce s webovou stránkou [17].

2.2.3 CSS

Cascading Style Sheets – je jazyk, který se používá k popisu stylu HTML dokumentu. Většina současných webových stránek používá ke kreslení obsahu kombinaci CSS a HTML. Implementace CSS je závislá na prohlížeči a může se měnit v závislosti na typu prohlížeče nebo jeho verzi.

Existuje několik způsobů, jak přidat styly CSS do dokumentu HTML:

- Inline - může být zapsán například do HTML tagu jako atribut: *<p style="...">*.
- External – Lze jej přidat jako externí soubor .css pomocí značky *<link rel="..." href="...">*.
- Internal – uvnitř značky <style> v dokumentu HTML.

Pomocí CSS může mít každá značka vlastní design. Může být aplikován přímo na značku nebo pomocí třídy značky [18].

2.2.4 NPM

Node Package Manager – je open-source Command Line Interface nástroj používaný ke správě všech závislostí projektu. Ke svému fungování vyžaduje prostředí Node.js. Balíčky lze vyhledávat na webu NPM: https://www.npmjs.com/. NPM také umožňuje vytvářet, publikovat a spravovat nové balíčky a přístupová oprávnění k nim. Může snadno instalovat, aktualizovat nebo mazat balíčky na základě názvu balíčku pomocí příkazu [19]:

- Instalace nebo aktualizace – npm *install <package_name>*.
- Vymazání – *npm delete [-save] <package_name>*.

Balíčky NPM mohou být dvou typů:

- Bez rozsahu (anglicky unscoped) – jedná se o veřejné balíčky, ke kterým má přístup kdokoliv.
- S rozsahem (anglicky scoped) – tyto balíčky mohou být veřejné nebo soukromé. Pokud je balíček soukromý, může je nainstalovat pouze uživatel s přístupem pro čtení. Balíčky s omezeným rozsahem se instalují pomocí příkazu: *npm install @<scope>/<package_name>*.

Verze nainstalovaných balíčků NPM pro projekt lze spravovat pomocí souboru **package.json**. Pokud je příkaz instalace NPM spuštěn ve složce obsahující tento soubor, NPM vyhledá verzi a na základě toho nainstaluje správnou verzi balíčku [20].

NPX je CLI nástroj poskytovaný NPM, který umožňuje uživateli spouštět místní nebo vzdálené balíčky NPM. Provádí se příkazem [21]: *npx <package_name> [arguments]*.

2.2.5 React

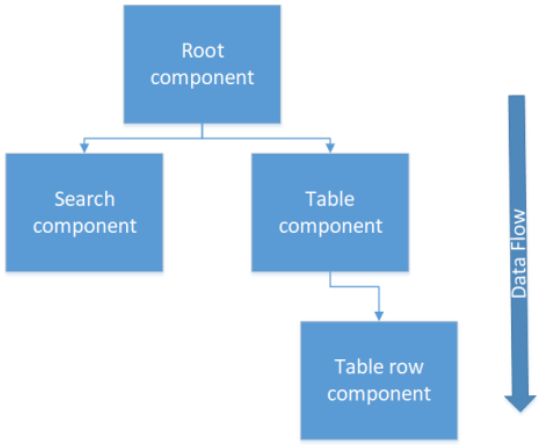
React je hlavní JavaScriptový framework používaný k vytvoření uživatelského rozhraní pro tuto webovou aplikaci. Ke spuštění vyžaduje prostředí Node.js.

Balíček Create React App NPM se používá k vytvoření React projektu. Jedná se o samostatný CLI nástroj vyvinutý společností Facebook, který se používá k vytváření nových aplikací pro reakce podle šablony. Příkaz použitý k vytvoření: *npx create-react-app <nazev> [22]*.

JSX je rozšíření syntaxe pro JavaScript, které se používá k vytváření React komponentu. Slouží k obohacení základního HTML jazyku o další funkce. Umožňuje jakýkoli platný výraz JavaScript, který je v {} jako atribut nebo v hodnotě značky. Všechny názvy atributů jsou v JSX zapsány v CamelCase [23].

React používá k vykreslování komponentu svou verzi DOM s názvem VDOM (virtuální DOM). Byl vytvořen jako rychlejší a lehčí verze DOM. Vykresleni všech React komponent vždy začíná od kořenového prvku.

React komponenta je definována funkcí JavaScript, která vrací kód JSX. Každý prvek je pak předán jako argument funkci *render(React.ReactElement)* volané na kořenovém prvku. Kořenové elementy jsou vytvářeny pomoci funkcí *createRoot(Element)*. React umožňuje různé režimy vykreslování, například: *<React.StrictMode>* se používá pro testovací účely, prvky se překresluje dvakrát. Data a funkce zpětného volání lze předat komponentám, které jsou ve stromu VDOM níže. Tyto hodnoty se nazývají „props“ [24].



Obrázek 4: Datový tok v React komponentách. Zdroj: [24]

Háky (anglicky Hooks) jsou zcela volitelné funkce React, které přidávají užitečné funkce a snižují standardní kód. Mohou být vytvořeny uživatelem nebo importovány z balíčku React. Některé příklady háčků React [25]:

- useState() – je hák, který umožňuje vytvořit stav komponenty Reactu bez použití tříd a konstruktorů.
- useEffect() – je hák, který je podmíněně spuštěn na základě stavu, který se mění. Všechny stavové objekty, které spouštějí tento háček, by mu měly být předány jako argument.

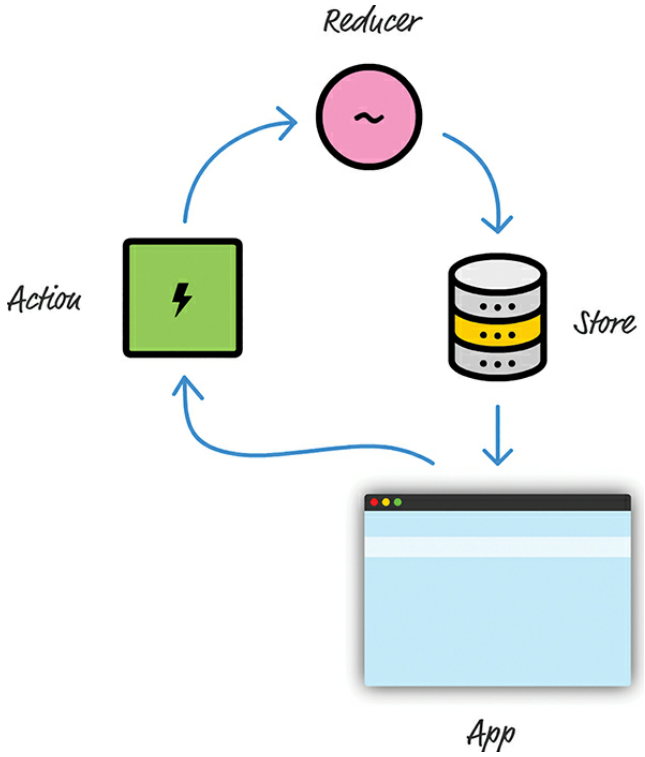
2.2.6 Redux

Redux je open-source správce stavu pro React aplikace. Umožňuje vytvořeni stavu aplikace, který obsahuje různé hodnoty nebo akce, ke kterým mají přístup komponenty Reactu. Redux je založen na třech jednoduchých aspektech:

- State – uchovává všechna potřebná data na jednom místě.
- Action – se používá k předání hodnot do reduceru.
- Reducer – je to funkce která může měnit stav.

Zajišťuje také, že data ve stavu jsou konzistentní. Redux umožňuje ukládat veškerý stav aplikace na jediné místo, což výrazně zkracuje dobu vývoje.

Stav je vždy přístupný v režimu pouze pro čtení. Lze jej upravit pouze pomocí reduktorů, které obdrží akci jako argument. To zajišťuje plnou kontrolu nad mutací stavu [26].



Obrázek 5: Datový tok v Redux. Zdroj: [26]

2.2.7 Redux toolkit

Je to sada nástrojů pro redux, která poskytuje novější, zjednodušené API pro práci s Redux. Hlavním účelem je zkrátit dobu vývoje a usnadnit konfiguraci Redux. Pomocí tohoto příkazu lze nainstalovat Redux toolkit: *npm install @reduxjs/toolkit.*

Některé z funkcí, které poskytuje [27]:

- configuredStore() – se používá k automatické konfiguraci úložiště Redux.
- createSlice() – vytvoří slice objekt, který má počáteční stav a redukce. Redukce lze poté exportovat jako akce. Stav lze změnit pouze z reduktorů.

2.2.8 MUI pro React

Jedná se o sadu open source konfigurovatelných šablon React komponentu, které lze použít k vytvoření různých React aplikací. Výrazně snižuje čas a náklady na vývoj. MUI lze nainstalovat pomocí [28]: *npm install @mui/material @emotion/react @emotion/styled.*

2.2.9 MSAL pro React

MSAL je open-source knihovna vytvořená společností Microsoft pro práci s AAD. Umožňuje zjednodušené přihlašování a odhlašování pomocí různých metod poskytovaných tímto balíčkem. MSAL má skvělou integraci s Reactem a poskytuje zabezpečení nezbytné pro AA. Lze jej nainstalovat pomocí příkazu [29]: *npm install @azure/msal-react @azure/msal-browser.*

2.3 Azure Active Directory

AAD je služba vytvořená společností Microsoft, která poskytuje AA pro jakoukoli registrovanou aplikaci.

Výhody použiti AAD:

- Snadná konfigurace.
- Může škálovat téměř nekonečně.
- Nepotřebuje DB ke správě uživatelů.
- Všechna oprávnění jsou spravována prostřednictvím uživatelského rozhraní na portálu Azure [30].

3 POPIS APLIKACE

Tato kapitola obsahuje podrobný popis všech částí aplikace. Tato aplikace se skládá ze dvou hlavních částí: backend, frontend (nefunkční požadavek 1). Jako AA systém používá AAD od Microsoftu. Zde bude ukázáno, jak jsou implementovány všechny funkční a nefunkční požadavky a jak může uživatel s touto aplikací pracovat.

3.1 Konfigurace AAD

AAD se používá jako hlavní poskytovatel AA pro tuto aplikaci (nefunkční požadavek 4). Umožňuje spravovat přidané uživatele a jejich oprávnění bez DB. To vše lze nakonfigurovat na Azure Portal.

Autentizace pouze pomocí backendu není možná, kvůli omezením CORS pro AAD. Používá se hybridní řešeni – autentizace uživatele se provádí na frontendu a backend ověřuje přijatý token. Ke správnému fungování vyžaduje konfiguraci na backendu i frontendu.

Každá nová aplikace musí být zaregistrována na stránce Azure Portal: https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/RegisteredApps. Aplikace má své vlastní jedinečné ID aplikace (klienta), ID objektu, ID adresáře (tenant), identifikátory API URI, které se používá ke konfiguraci AAD.

Po registraci je potřeba dokončit několik konfiguračních kroků [31]:

1. Je nutné zaregistrovat novou konfiguraci platformy. Tato aplikace vyžaduje registraci jednostránkové aplikace s přesměrováním URI: „http://localhost:3000“.
2. Poté musí být zaškrtnuta políčka přístupových a ID tokenů.
3. Poté musí být vytvořen nový rozsah. Tento identifikátor URI aplikace bude použit backendem k ověření uživatelských oprávnění.
4. Poté musí být tento nově vytvořený rozsah přidán do oprávnění aplikace a schválen správcem.

3.2 Backend

Backend je první hlavní částí této aplikace. Je zodpovědný za veškeré manipulace s daty. Backend je zodpovědný za většinu funkčních požadavků této aplikace. Poskytuje REST API, které lze použít k příjmu, ukládání, aktualizaci a mazání dat.

Backend byl vytvořen pomocí jazyka Java 8 a frameworků Spring Boot, Hibernate (nefunkční požadavek 2). Java 8 byla vybrána jako vývojový jazyk kvůli omezením hostitelského serveru. Všechny tyto technologie jsou používány kvůli nefunkčnímu požadavku číslo 2. Backend používá Apache Maven ke správě závislostí a procesu sestavení.

3.2.1 Konfigurace AAD pro Spring Security

Backend přijímá HTTP požadavek obsahující hlavičku „Authentication“ s autentizačním tokenem. Token se pak automaticky ověří pomocí knihovny Spring Boot Starter Azure Active Directory. Pokud je token platný a uživatel má všechna potřebná oprávnění, může uživatel komunikovat s REST API backendového serveru.

Backendová autentizace vyžaduje určitou konfiguraci, aby fungovala [32]:

- Potřebuje bean rozšiřující AADResourceServerWebSecurityConfigurerAdapter se správnými anotacemi: *@EnableWebSecurity* a *@EnableGlobalMethodSecurity(prePostEnabled = true)*. Obsahuje všechny potřebné konfigurace pro komunikaci se službou AAD.
- Vyžaduje vlastnosti konfigurace: azure.activedirectory.tenant-id, azure.activedirectory.client-id, azure.activedirectory.app-id-uri. Tyto vlastnosti musí obsahovat platné údaje registrované aplikace.
- Vyžaduje správnou konfiguraci filtru požadavků HTTP pro následující hlavičky:
  - Access-Control-Allow-Origin – musí obsahovat adresu UI serveru, aby byla umožněna komunikace CORS s backendem. Například: „http://localhost:3000“.
  - Access-Control-Allow-Methods – obsahuje všechny povolené metody HTTP. Například: „POST, GET, PUT, DELETE, OPTION“.
  - Access-Control-Allow-Headers – obsahuje všechny povolené hlavičky požadavků. Například: “authorization, content-type,x-auth-token, access-control-request-headers, access-control-request-method, accept, origin, x-requested-with, content-disposition”.

Ověření oprávnění REST API je přidáno pomocí anotace Spring Security. Toto je příklad takové anotace používané touto aplikací: @PreAuthorize("hasAuthority('SCOPE_api')"). To znamená, že uživatel, který je spojen s tokenem, musí mít oprávnění k přístupu k rozsahu zvanému „api“.

3.2.2 Srovnávací moduly

Nový modul porovnání je vytvořen jako implementace abstraktní třídy FileCompareService, která má následující abstraktní metody (nefunkční požadavek 5):

- public abstract Map<String, List<String>> splitBatch(InputStream referenceStream) throws InvalidFileException – tato metoda se používá k rozdělení dávkového souboru, pokud to formát faktury podporuje.
- public abstract Result compare(InputStream processed, List<String> referenceInvoiceLines) throws InvalidFileException – tato metoda se používá k porovnání zpracované a referenční faktury.
- protected abstract void deserializeConfigFile(ObjectMapper jsonMapper, InputStream inputStream) throws IOException – tato metoda se používá k deserializaci konfiguračního souboru JSON.

Modul je pak poskytnut službě, která vytváří výsledky porovnání pomocí FileCompareServiceFactory na základě formátu z konfigurace porovnání (funkční požadavek 3). Aby porovnávání fungovalo správně, musí být pro každou implementaci porovnávacího modulu nastaven porovnávací formát a porovnávací konfigurace (nefunkční požadavek 7).

3.2.3 Poskytovatelé souborů

Poskytovatel představuje režim, ve kterém tato aplikace funguje. Mohou být dvojího druhu (funkční požadavek 2):

- ReferenceFileProvider – který se používá k příjmu referenčních souborů (soubory generované starým frameworkém). API:
  - List<String> listFiles() throws FileProviderException.
  - InputStream getFile(String fileIdentifier) throws FileProviderException.
  - String getFileName(String fileIdentifier) throws FileProviderException.
  - default void onReferenceFileProcessed(String fileIdentifier) throws FileProviderException.

- ProcessedFilesProvider – který se používá pro příjem zpracovaných souborů (soubory generovaných novým frameworkem). API:
  - InputStream openProcessedFile(String invoiceId) throws FileProviderException.
  - String getProcessedFileName(String invoiceId) throws FileProviderException.
  - default void onFileProcessed(String processedFileName) throws FileProviderException.

Tato aplikace má dvě implementace poskytovatelů souborů, které podporují porovnávání souborů ve dvou režimech:

- DB – referenční soubory jsou přijímány z FS. Zpracované soubory jsou v DB a vyhledány podle ID faktury (funkční požadavek 1).
- SFTP – referenční soubory jsou přijímány z SFTP. Zpracované soubory jsou na SFTP a vyhledány podle ID faktury (funkční požadavek 1).

3.2.4 REST API

Všechna API této aplikace vyžadují oprávnění „SCOPE_api“. Tato aplikace obsahuje čtyři ovladače odpočinku, které jsou seskupeny do dvou základních cest:

- „/auth/v1“ – která se používá pro autentizační informace.
- „/api/v1“ – která se používá pro všechna ostatní aplikační API.

Seznam všech ovladačů aplikací:

- CompareConfigController – poskytuje API pro vytváření, aktualizaci, přijímání a odstraňování porovnávacích konfiguračních objektů:
  - „/compare-configs“ – se používá k příjmu všech porovnávacích konfiguračních objektů. Podporuje různé parametry požadavků, které se používají pro filtrování a stránkování (funkční požadavek 6):
    - Integer page – povinný parametr, který představuje číslo stránky.
    - Integer pageSize – povinný parametr, který představuje velikost stránky.
    - String format – volitelný parametr, který se používá k filtrování podle formátu.

        - String providerType – volitelný parametr, který se používá k filtrování podle typu poskytovatele.
    - „/compare-configs/{id}“ – se používá k příjmu porovnávací konfigurace podle id.
    - „/compare-configs“ – se používá k vytvoření nové porovnávací konfigurace, když je HTTP požadavek typu POST a obsahuje tělo s objektem porovnávací konfigurace (funkční požadavek 4).
    - „/compare-configs/{id}“ – se používá k aktualizaci porovnávací konfigurace, když je HTTP požadavek typu PUT a obsahuje tělo s objektem porovnávací konfigurace.
    - „/compare-configs/{id}“ – používá se ke smazání porovnávací konfigurace, když je HTTP požadavek typu DELETE.
- CompareController – poskytuje API pro spuštění porovnávací konfigurace, příjem seznamu formátů a poskytovatelů:
    - „/compare/{id}“ – přijímá požadavky GET a spouští porovnávací konfigurace podle id (funkční požadavek 5).
    - „/compare/formats“ – přijímá požadavky GET a vrací odpověď se seznamem porovnávacích formátů.
    - „/compare/providers“ - přijímá požadavky GET a vrací odpověď se seznamem poskytovatelů.
- CompareResultController – poskytuje rozhraní API pro příjem výsledků porovnání a stahování výsledků ve formátu TXT nebo CSV:
    - „/compare-results“ – přijme požadavek GET s parametry a vrátí odpověď se seznamem objektu s výsledky porovnání. Podporuje různé parametry požadavků, které se používají pro filtrování a stránkování (funkční požadavek 6):
        - Integer page – povinný parametr, který představuje číslo stránky.
        - Integer pageSize – povinný parametr, který představuje velikost stránky.
        - String name – volitelný parametr, který se používá k filtrování podle názvu.
        - String dateFrom – volitelný parametr, který se používá k filtrování podle data zahájení.
        - String dateTill – volitelný parametr, který se používá k filtrování podle data ukončení.

- „/compare-results/download/csv/{id}“ – obdrží požadavek GET a vrátí odpověď s výsledkem ve formátu CSV jako přílohu.
- „/compare-results/download/txt/{id}“ – obdrží požadavek GET a vrátí odpověď s výsledkem ve formátu TXT jako přílohu.

3.2.5 Hibernate

Tato aplikace používá Hibernate k zachování objektů DB. Používá se také k mapování DB tabulek na aplikační entity. Seznam entit:

- CompareConfig:
  - Long id.
  - ProviderType providerType.
  - String inputDir.
  - Long backendId.
  - CompareFormat format.
  - Long messageFormatId.
  - String reportFileName.
  - String configFile.
- CompareFormat – je Enum, který obsahuje všechny dostupné porovnávací formáty.
- ProviderType – je Enum, který obsahuje všechny dostupné typy poskytovatelů.
- CompareResult:
  - Long id.
  - String displayedName.
  - String filename.
  - String filetype.
  - Timestamp creationTime.
- ProcessedFile:
  - Long id.
  - Long messagePurposeId.
  - String filename.
  - Long messageFormatId.
  - String fileCompressed.

- Visibility visibility.
- Long reprocessNumber.

❖ Visibility:

- Long id.
- Long backendId.
- OffsetDateTime translationEnd.
- VisibilityAttributes visibilityAttributes.

❖ VisibilityAttribures:

- Long id.
- String invoiceId.
- Visibility visibility.

JpaRepository se používá pro přístup ke všem datům. K tabulkám FILES, VISIBILITY a VIS_ATTRIBUTES lze přistoupit pomocí ReadOnlyRepository, které má pouze specifické deklarace funkcí, které umožňují pouze číst data (nefunkční požadavek 6).

3.3 Frontend

Frontend je druhou nejdůležitější částí této aplikace a byl vytvořen pomoci JavaSctipt, React, Redux (nefunkční požadavek 3). Umožňuje veškeré uživatelské interakce s touto aplikací a poskytuje AA pomocí AAD od společnosti Microsoft (nefunkční požadavek 4).

Skládá se ze čtyř stránek:

❖ Welcome – je hlavní stránka aplikace.

❖ Compare Configs – stránku, která se používá k vytváření, úpravám, odstraňování a spouštění porovnávacích konfigurací.

❖ Results – seznam všech vygenerovaných výsledků porovnání.

❖ About – obsahuje základní popis aplikace.

3.3.1 Konfigurace AAD pro frontend

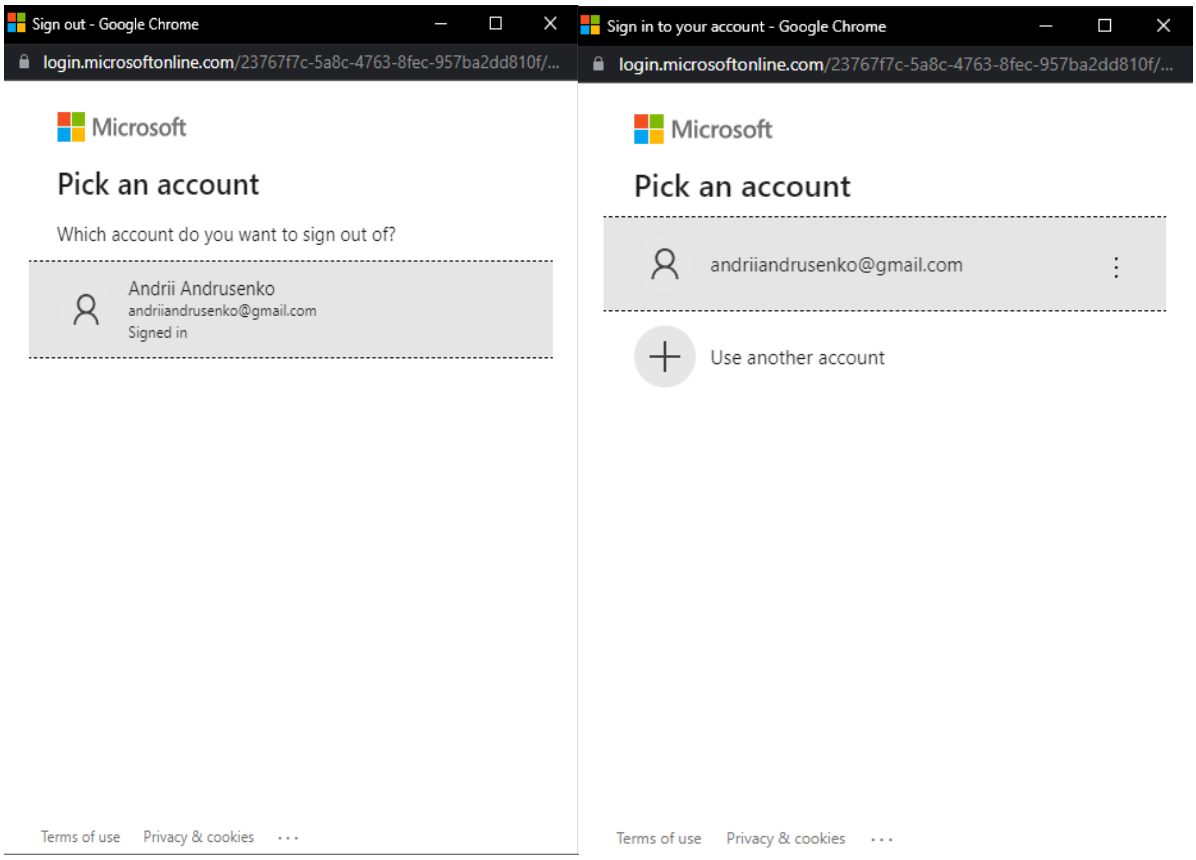
Frontend je zodpovědný za funkce přihlašování a odhlašování uživatelů. Funkce AA poskytuje knihovna MSAL od společnosti Microsoft.

Kroky potřebné k povolení ověřování AAD na SPA:

1. Vytvořte instanci MSAL pomocí třídy PublicClientApplication z balíčku „@azure/msal-browser“. Musí obdržet konfiguraci ověřování obsahující pole auth a cache.
2. Instance MSAL musí být vložena do aplikace pomocí: <MsalProvider instance={msalIntance}><App/></MsalProvider>.
3. K instanci MSAL lze přistupovat z libovolné komponenty aplikace pomocí hooku useMsal() z balíčku „@azure/msal-react“.

Podporuje dva různé typy přihlášení a odhlášení:

- Popup – vytvoří se nové autentizační okno. Příklad kódu JS: *instance.loginPopup(loginRequest)*.
- Redirect – uživatel je přesměrován na ověřovací stránku AAD, po dokončení akce je uživatel přesměrován zpět na nakonfigurovanou stránku. Příklad kódu JS: *instance.loginRedirect(loginRequest)*.



Obrázek 6: Přihlášení a odhlášení pomocí AAD. Zdroj: vlastní

Hook useIsAuthenticated() z balíčku „@azure/msal-react“ se používá ke kontrole, zda je uživatel ověřen [33].

3.3.2 React Router

React router slouží k přepínání mezi stránkami. Každá komponenta Route představuje stránku aplikace. Uživatel může přepínat mezi stránkami pomocí komponenty <Navbar/>. Tato komponenta se skládá z komponent <NavLink>, které se používají k přesměrování na stránku, a komponenty <NavBtnLink>, které se používají k přihlášení nebo odhlášení. Příklad Route: *<Route path="/" exact element={<Home />}></Route>*.

Tato aplikace používá chráněné cesty, které lze přistoupit pouze tehdy, když je uživatel přihlášen. Pokud uživatel není přihlášen, bude přesměrován na stránku „Welcome“, jinak bude přesměrován na chráněnou stránku. Příklad: *<Route path="/compare-configs" element={<ProtectedRoute> <CompareConfigs /> </ProtectedRoute> }></Route>*.

Pokud se uživatel pokusí zadat jakoukoli jinou cestu, než je uvedeno v této aplikaci, bude přesměrován na komponentu <NotFound/>. Příklad: *<Route path="*" element={<NotFound />} />*.

3.3.3 React

React je hlavní framework zodpovědný za uživatelské rozhraní aplikace. Skládá se z různých React komponent, které se vykreslují v prohlížeči od kořenové komponenty.

Tato aplikace využívá kombinaci třech typů React komponent:

- MUI komponenty – předem vytvořené komponenty React z balíčku „@mui/material“. [28].
- Styled komponenty – pomocí tohoto balíčku mohou být různé HTML tagy stylizování a exportovaní jako React komponenty [34]. Příklad:

```
export const CardContainer = styled.div`
width: 60vw;
display: flex;
border-style: solid;
border-radius: 5px;
border-width: 2px;
border-color: rgba(0, 0, 0, 0.23);
align-items: center;
justify-content: space-between;
padding: 1rem;
`;
```

- Základní komponenty – základní komponenty React, které vracejí kód JSX.

Každá běžná komponenta React může používat různé háčky React. Hlavní tři háčky, které byly použity v této aplikaci, jsou:

- useState()
- useEffect()

Například na stránce porovnání konfigurace se používá háček *useEffect()* k načtení formátů, poskytovatelů a porovnávací konfigurací, když se změní stav filtru:

```
useEffect(() => {
    dispatch(fetchFormats());
    dispatch(fetchProviders());
     dispatch( fetchCompareConfigs(page, pageSize, filter.format, filter.provider) );
     // eslint-disable-next-line react-hooks/exhaustive-deps
 }, [filter]);
```

Hook useState() se používá ke správě počtu otevřených stránek na stránkách „Compare Configs“ a „Results“. Používá jej komponenta MUI *<Pagination/>* k poskytování funkcí stránkování. Příklad ze stránky “Compare Config“:

```
const [page, setPage] = useState(1);
<PagginationFooterContainer>
<Pagination
        count={Math.ceil(totalCount / pageSize)}
        size="medium"
        onChange={handlePageChange}
/>
</PagginationFooterContainer>
```

3.3.4 Redux

Redux je v této aplikaci implementován pomocí @reduxjs/toolkit od Facebooku. Je zodpovědný za řízení stavu React komponent. Poskytuje také kontrolu nad změnami stavu.

Redux je implementován pomocí objektu store, který je vytvořen pomocí funkce *configureStore()* z @reduxjs/toolkit. Tato funkce přijímá objekt, který obsahuje všechny redukce. Objekt store lze poté exportovat a použít tam, kde je to potřeba. Vkládá se do aplikace pomocí *<Provider store={store}></Provider>* z balíčku “react-redux“.

Konfigurace úložiště, kterou tato aplikace používá [27]:

```
const store = configureStore({
  reducer: {
    auth: authSlice.reducer,
    compare: compareSlice.reducer,
    config: configSlice.reducer,
    result: resultSlice.reducer,
    ui: uiSlice.reducer,
  },
});
```

Každý reduktor je exportován z objektu Slice a je zodpovědný za různé funkce a obsahuje nezbytné stavy a akce [35]:

- Auth – obsahuje veškerou logiku potřebnou pro autentizaci.
- Config – obsahuje veškerou logiku, která je nezbytná pro zobrazení stránky s konfiguracemi porovnání. Počáteční stav obsahuje:
  - Seznam srovnávacích konfiguračních objektů.
  - Objekt filtru – používá se k filtrování srovnávacích konfigurací podle formátu a typu poskytovatele.
  - Celkový počet porovnávacích konfigurací – používá se k poskytování funkce stránkování.
- Result – obsahuje veškerou logiku potřebnou k zobrazení stránky s výsledky porovnání. Počáteční stav obsahuje:
  - Seznam objektů s výsledky porovnání.
  - Celkový počet výsledku.
- Compare – obsahuje všechna data, která jsou nezbytná pro práci se srovnávacími konfiguracemi: formáty a poskytovatelé.
- UI – odpovídá za různé funkce uživatelského rozhraní, například: zobrazení upozornění (funkční požadavek 8).

Slice je objekt vytvořený pomocí funkce *createSlice()* s balíku „@reduxjs/toolkit“. Tento objekt se skládá ze tří polí:

- Name – je String hodnota, která představuje název Slice objektu.
- InitialState – je objekt, který obsahuje stav Slicu.
- Reducers – je objekt, který obsahuje všechny akce, které tento Slice poskytuje. Tato akce se používá k mutaci stavu Slicu.

Příklad konfigurace Slicu, která se používá pro různé akce uživatelského rozhraní této aplikace (funkční požadavek 8):

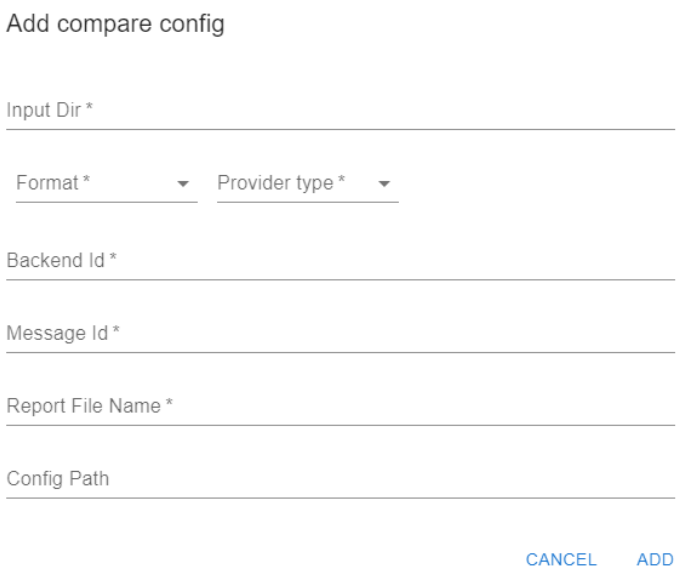
```
const uiSlice = createSlice({
  name: "ui",
  initialState: { notification: null,  },
  reducers: {
    showNotification(state, action) {
      state.notification = {
        message: action.payload.message,
        type: action.payload.type,
        open: action.payload.open,
      };
    },
  },
});
export const uiActions = uiSlice.actions;
export default uiSlice;
```

Stav lze zpřístupnit pomocí háku *useSelector()* z balíčku „react-redux“. Například: *const notification = useSelector((state) => state.ui.notification)*. Tento objekt lze použít pro různé účely uvnitř komponenty React [36].

3.3.5 Stránka „Compare Configs“

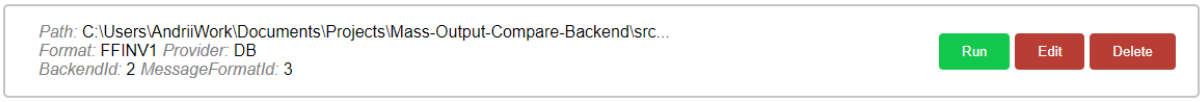
Stránka, která obsahuje všechny srovnávací konfigurace. Na tuto stránku mohou vstoupit pouze ověření uživatelé. Podporuje filtrování (funkční požadavek 11) podle formátu a poskytovatele. Tato stránka také podporuje stránkování, které se provádí pomocí komponenty MUI. Filtrování i stránkování se provádí na straně backendu.

Pomocí tlačítka „Add compare config“ uživatel může přidat novou konfiguraci porovnání. Po kliknutí na toto tlačítko se zobrazí dialog. Tento dialog obsahuje povinná pole označená hvězdičkou a volitelná pole a používá se k vytvoření nové konfigurace porovnání.



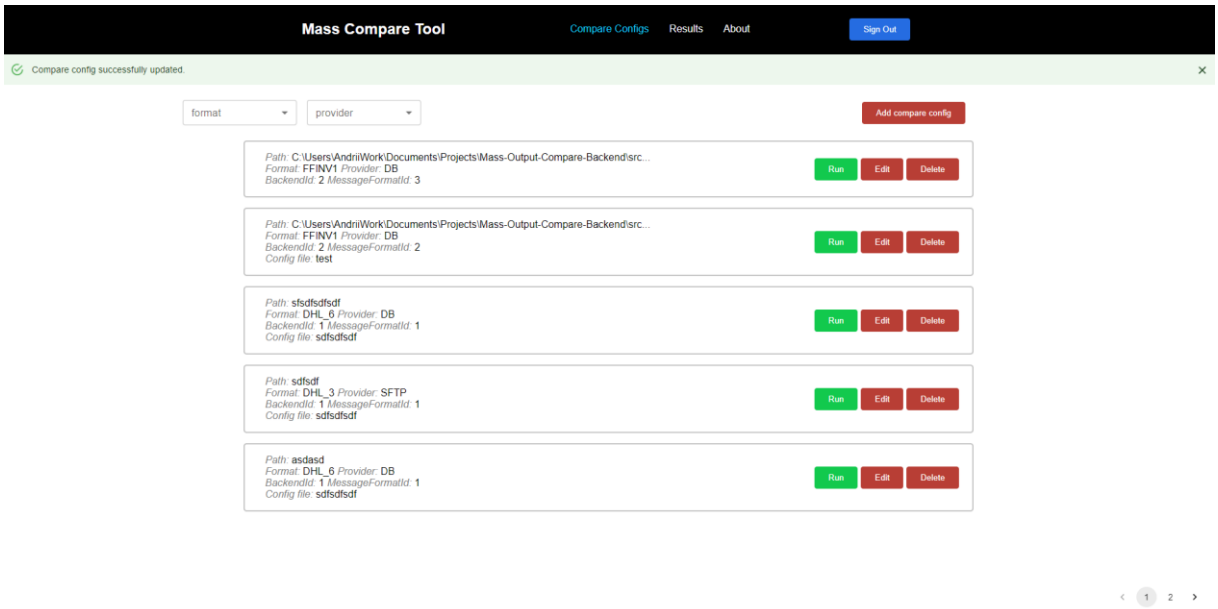
Obrázek 7: Dialog pro vytvoření nové konfigurace porovnání. Zdroj: vlastní

1. Input Dir – cesta k adresáři na FS nebo SFTP, který obsahuje všechny referenční soubory.
2. Format – je výběrové pole se všemi dostupnými formáty faktur.
3. Provider type – je výběrové pole se všemi dostupnými režimy provádění.
4. Backend ID – ID backendu, který vytvořil zpracované soubory.
5. Message ID – ID formátu zprávy zpracovávaných souborů.
6. Report file name – je název generovaného výsledného souboru.
7. Config path – je cesta ke konfiguračnímu souboru na FS nebo SFTP.



Obrázek 8: Komponent pro konfigurace porovnaní. Zdroj: vlastní

Jedna komponenta konfigurace porovnání je zobrazena na obrázku 8. Obsahuje popis všech polí a tlačítek, která umožňují tuto konfigurace spustit, upravit nebo smazat.

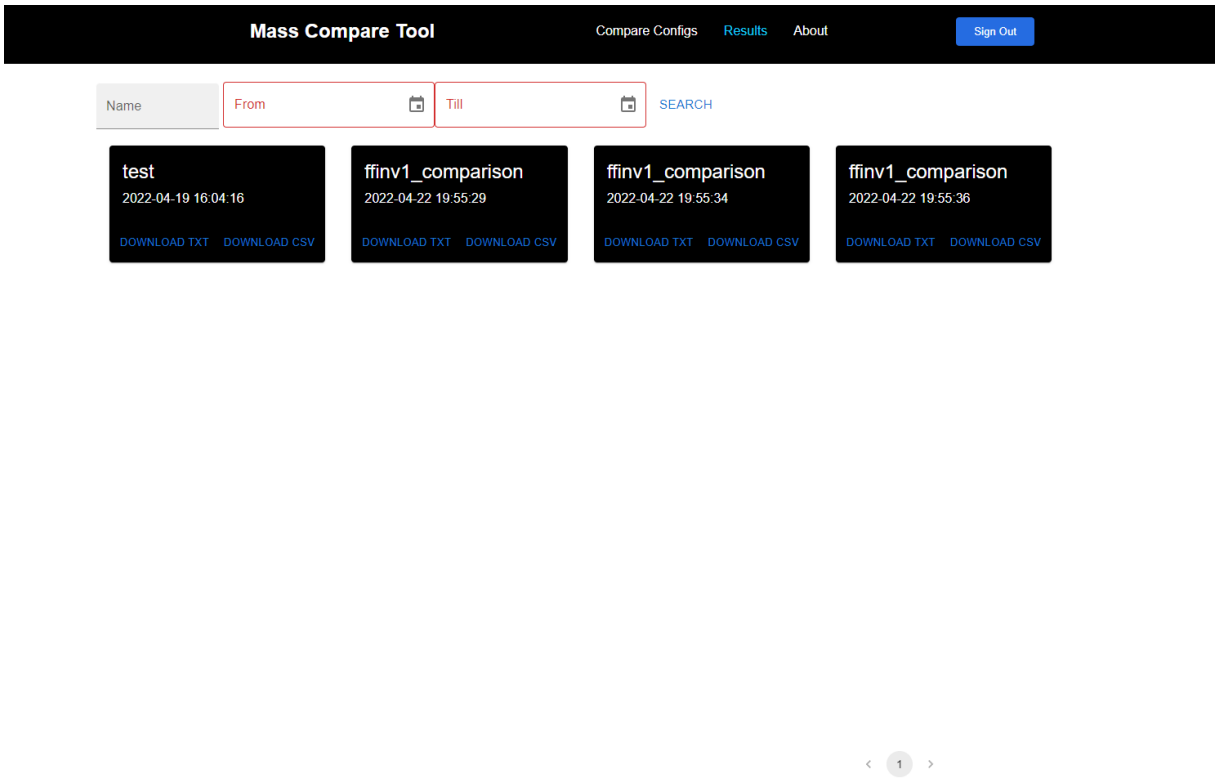


Obrázek 9: Stránka „Comape Configs“. Zdroj: vlastní

3.3.6 Stránka „Results“

Tato stránka se používá k zobrazení všech vygenerovaných výsledků porovnání. Na tuto stránku mohou vstoupit pouze ověření uživatelé. Každý nový výsledek je zobrazen v samostatném boxu, který zobrazuje jeho název, datum vytvoření a má dvě tlačítka:

- Download txt – stáhne JSON výsledek ve formátu TXT.
- Download csv – stáhne JSON výsledek ve formátu CSV.



Obrázek 10: Stránka „Results“. Zdroj: vlastní

Tato stránka může filtrovat výsledky porovnání podle jména a data. Podporuje také stránkování.

ZÁVĚR

Cílem této bakalářské práce bylo vytvořeni aplikaci, která umožní uživatelům porovnávat faktury různých formátů. Tato aplikace byla zaměřena především na interní pracovníky: konzultanty a analytiky. Má porovnávat faktury v různých formátech obdržené z různých zdrojů. Uživatel by měl mít přístup ke všem datům z jednoduchého uživatelského rozhraní, které je nezávislé na backendu této aplikace.

Všechny výše uvedené cíle byly úspěšně splněny. Aplikace byla řádně otestována a podporuje modularitu. Obsahuje všechny potřebné náležitosti jako je práce s porovnávacími konfiguracemi, spouštění porovnávacích konfigurací, ukládání dat, přístup k datům pomocí FS, DB, SFTP, formátování výsledků a další. Všechny nefunkční požadavky byly rovněž splněny.

Díky modulární struktuře této aplikace je možné snadno měnit technologie každé součásti, pokud to bude v budoucnu potřeba. Nové porovnávací formáty lze také snadno přidat, protože stačí rozšířit abstraktní třídu. Pokud budou v budoucnu vyžadovány nové způsoby přijímání souborů, lze je snadno přidat pomocí poskytovatele souborů používaného v této aplikaci.

POUŽITÁ LITERATURA

[1] MCGRATH, Mike. *Java in easy steps*. 7. vyd. Warwickshire: In Easy Steps Limited, 2019. ISBN 9781840788730.

[2] AUSTERLITZ, Howard. *Data Acquisition Techniques Using PCs*. 2. vyd. USA: Academic Press, 2003. ISBN 9780120683772.

[3] JENKOV, Jakob. Java Memory Model. *Jenkov.com* [online]. 2020-07-01 [cit. 2022-04-18]. Dostupné z: https://jenkov.com/tutorials/java-concurrency/java-memory-model.html

[4] GOETZ, Brian. *Java Concurrency in Practice*. Boston: Addison-Wesley Professional, 2006. ISBN 9780321349606.

[5] VARANASI, Balaji. *Introducing Maven: A Build Tool for Today's Java Developers*. 2. vyd. New York: Apress, 2019. ISBN 9781484254097.

[6] Introduction to the Build Lifecycle. *Apache Maven Project* [online]. 2022-04-16 [cit. 2022-04-18]. Dostupné z: https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

[7] Your First Plugin. *Apache Maven Project* [online]. 2022-04-16 [cit. 2022-04-18]. Dostupné z: https://maven.apache.org/guides/plugin/guide-java-plugin-development.html

[8] GUTIERREZ, Felipe. *Pro Spring Boot 2: An Authoritative Guide to Building Microservices, Web and Enterprise Applications, and Best Practices*. 2. vyd. New York: Apress, 2018. ISBN 9781484236758.

[9] Spring Quickstart Guide. *Spring* [online]. [cit. 2022-04-18]. Dostupné z: https://spring.io/quickstart

[10] SPILCA, Laurentiu. *Spring Security in Action*. New York: Manning Publications, 2020. ISBN 9781617297731.

[11] BAUER, Christian, Gavin KING a Gary GREGORY. *Java Persistence with Hibernate*. 2. vyd. Shelter Island, NY: Manning Publications, 2015. ISBN 9781617290459.

[12] HTTP. *Mdn web docs* [online]. 2022-04-17 [cit. 2022-04-18]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP

[13] FIELDING, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Irvine, 2000. Disertace. University of California.

[14] NEOTERICEU. Single-page application vs. multiple-page application. *Medium.com* [online]. 2016-12-02 [cit. 2022-04-19]. Dostupné z: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58

[15] HAVERBEKE, Marijn. *Eloquent JavaScript*. 3. vyd. San Francisco: No Starch Press, 2018. ISBN 9781593279509.

[16] Promise. *Mdn web docs* [online]. 2022-04-05 [cit. 2022-04-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

[17] HTML: HyperText Markup Language. *Mdn web docs* [online]. 2022-02-18 [cit. 2022-04-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTML

[18] CSS: Cascading Style Sheets. *Mdn web docs* [online]. 2022-04-13 [cit. 2022-04-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/CSS

[19] About npm. *Npm docs* [online]. [cit. 2022-04-19]. Dostupné z: https://docs.npmjs.com/about-npm

[20] Downloading and installing packages locally. *Npm docs* [online]. [cit. 2022-04-19]. Dostupné z: https://docs.npmjs.com/downloading-and-installing-packages-locally

[21] Npx. *Npm docs* [online]. [cit. 2022-04-19]. Dostupné z: https://docs.npmjs.com/cli/v8/commands/npx

[22] KARRYS, Luke. Getting Started. *Create React App* [online]. 2021-09-01 [cit. 2022-04-19]. Dostupné z: https://create-react-app.dev/docs/getting-started/

[23] Introducing JSX. *React* [online]. [cit. 2022-04-19]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html

[24] HINKULA, Juha. *Hands-On Full Stack Development with Spring Boot 2 and React: Build modern and scalable full stack applications using Spring Framework 5 and React with Hooks*. 2. vyd. Birmingham: Packt Publishing, 2019. ISBN 9781838822361.

[25] Hooks at a Glance. *React* [online]. [cit. 2022-04-19]. Dostupné z: https://reactjs.org/docs/hooks-overview.html

[26] CHINNATHAMBI, Kirupa. *Learning React: A Hands-On Guide to Building Web Applications Using React and Redux*. Boston: Addison-Wesley Professional, 2018. ISBN 9780134843551.

[27] Getting Started with Redux Toolkit. *Redux Toolkit* [online]. 2022-04-04 [cit. 2022-04-20]. Dostupné z: https://redux-toolkit.js.org/introduction/getting-started

[28] Installation. *MUI* [online]. [cit. 2022-04-20]. Dostupné z: https://mui.com/material-ui/getting-started/installation/

[29] Microsoft Authentication Library for React (msal-react). *Npm* [online]. [cit. 2022-04-20]. Dostupné z: https://www.npmjs.com/package/@azure/msal-react

[30] What is Azure Active Directory?. *Microsoft Docs* [online]. 2022-04-19 [cit. 2022-04-20]. Dostupné z: https://docs.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis

[31] Register a client application in Azure Active Directory. *Microsoft Docs* [online]. 2022-04-05 [cit. 2022-04-24]. Dostupné z: https://docs.microsoft.com/en-us/azure/healthcare-apis/register-application

[32] Spring Boot Starter for Azure Active Directory developer's guide. *Microsoft Docs* [online]. 2021-03-31 [cit. 2022-04-24]. Dostupné z: https://docs.microsoft.com/en-us/azure/developer/java/spring-framework/spring-boot-starter-for-azure-active-directory-developer-guide

[33] Tutorial: Sign in users and call the Microsoft Graph API from a React single-page app (SPA) using auth code flow. *Microsoft Docs* [online]. 2021-10-25 [cit. 2022-04-24]. Dostupné z: https://docs.microsoft.com/en-us/azure/active-directory/develop/tutorial-v2-react

[34] Basics. *Styled components* [online]. [cit. 2022-04-24]. Dostupné z: https://styled-components.com/docs/basics#getting-started

[35] CreateSlice. *Redux Toolkit* [online]. 2021-04-11 [cit. 2022-04-24]. Dostupné z: https://redux-toolkit.js.org/api/createSlice

[36] Hooks. *React Redux* [online]. 2022-04-16 [cit. 2022-04-24]. Dostupné z: https://react-redux.js.org/api/hooks