UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2020 Bc. Jan Houžvička

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Návrh a implementace systému pro správu personální agendy

Diplomová práce

2020 Bc. Jan Houžvička

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Jan Houžvička**
Osobní číslo:	**I18240**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Návrh a implementace systému pro správu personální agendy**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

V teoretické části práce budou popsány současné trendy v oblasti informačních systému se změřením na personální činnosti. Dále pak veškeré technologie, které budou finálně pro realizace páce vybrány. V rámci vlastní implementace bude třeba navrhnout vhodný databázový model a navrhnout samotný informační systém pro správu personální agendy s využitím frameworku JAVA Spring. Dále pak bude pro potřebu vzdálené správy soborů vybrat a zaimplementovat vhodný přístup nebo službu, která bude vyhovovat bezpečnému uložení dat. Systém musí být řešen i z pohledu různých uživatelských rolí.

Rozsah pracovní zprávy: **50-60 stran**
Rozsah grafických prací: **–**
Forma zpracování diplomové práce: **tištěná**

Seznam doporučené literatury:

CARNELL, John. Spring microservices in action: a multiplatform approach to building chatbots. Shelter, Island, NY: Manning Publications Co., 2017. ISBN 16-172-9398-9.
CARNELL, John. Beginning spring boot 2: applications and microservices with the spring framework. New York, NY: Springer Science Business Media, 2017. ISBN 978-148-4229-309.

Vedoucí diplomové práce: **Ing. Jan Fikejz, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **5. listopadu 2019**
Termín odevzdání diplomové práce: **15. května 2020**

Univerzita Pardubice Česká republika

Ing. Zdeněk Němec, Ph.D.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2019

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., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, 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. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 13. 8. 2020

Jan Houžvička

PODĚKOVÁNÍ

Na tomto místě bych rád poděkoval vedoucímu práce, panu Ing. Janu Fikejzovi, Ph.D., za jeho rady, pomoc i konzultace. V neposlední řadě bych také chtěl poděkovat celé své rodině za podporu, kterou mi dodávala během celého navazujícího studia.

ANOTACE

Tato diplomová práce se zabývá problematikou informačních systémů pro správu personální agendy. Nejdříve představuje některé dnes používané systémy, ale primárně je zaměřena na navržení a implementaci vlastního řešení za použití Java frameworku Spring (konkrétně jeho nadstavy Spring Boot), Java FX a dalších technologií v této práci uvedených.

KLÍČOVÁ SLOVA

informační systém, desktopová aplikace, Java, JavaFX, framework Spring, Spring Boot, personální agenda

TITLE

Design and implementation of personnel administration system

ANNOTATION

This diploma thesis deals with the issue of information systems for the personnel agenda management. First, it introduces some systems that are being used today but it is primarily focused on design and implementation of own solution using Java framework Spring (specifically its extension called Spring Boot), JavaFX and other technologies mentioned further in this diploma thesis.

KEYWORDS

information system, desktop application, Java, JavaFX, Spring framework, Spring Boot, personnel agenda

OBSAH

SEZNAM ILUSTRACÍ

SEZNAM UKÁZEK KÓDU

SEZNAM ZKRATEK A ZNAČEK

Amazon S3	Amazon Simple Storage Service
API	Application Programming Interface
CSS	Cascading Style Sheets
DAO	Data Access Object
DI	Dependency Injection
DTO	Data Transfer Object
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IČO	Identifikační číslo osoby
IDE	Integrated Development Environment
IoC	Inversion of Control
IoT	Internet of Things
IS	Informační systém
JDK	Java Development Kit
JPA	Java Persistence API
JRE	Java Runtime Environment
MVC	Model View Controller
ORM	Object-Relational Mapping
SDK	Software Development Kit
URL	Uniform Resource Locator
XML	Extensible Markup Language

TERMINOLOGIE

Kandidát	Osoba hledající zaměstnání
Klient	Společnost/firma hledající kandidáty na určené pracovní pozice
Pracovní pozice	Pozice nabízená klientem pro kandidáty
Náborový proces	Proces náboru konkrétního kandidáta na vybranou pracovní pozici

ÚVOD

Informační technologie jsou vzhledem ke svému neustálému dynamickému rozvoji již řadu let nedílnou součástí našeho každodenního života. Není tedy divu, že své uplatnění naleznou ve velkém množství oborů lidské činnosti. Aktuálními příklady mohou být IoT pro tvorbu tzv. chytrých domácností, pokroky v autonomním řízení vozidel, neustálý rozvoj eGovernment, nebo používání informačních systémů jakožto podporu pro celou řadu denních činností.

Jedním z těchto oborů je také obor personalistiky. V něm se již upustilo od pouhého používání tužky a papíru, které bývalo dříve nahrazováno vedením personální agendy pomocí softwarů jako je např. MS Excel. Také toto řešení je z dnešního hlediska spíše zastaralé a aktuálním trendem je využívání efektivních informačních systémů.

Cílem této diplomové práce je pak návrh a implementace právě informačního systému pro správu personální agendy.

Nejdříve je tedy uveden pojem informační systém obecně, následně pak v kontextu personalistiky společně s vybranými zástupci. Pro vytvoření komplexního informačního systému je často nutné použít více technologií a ani tato práce není výjimkou. Dále jsou proto představeny všechny technologie, které byly zvoleny k vývoji tohoto informačního systému.

Následující část zaměřená na samotný návrh a implementaci pak představuje praktické použití těchto technologií ve významných partiích vývoje jako je tvorba grafického uživatelského rozhraní, implementace funkčních požadavků, správa dat a dokumentů nebo nasazení samotného systému.

1 INFORMAČNÍ SYSTÉMY

Pojem informační systém, často označován pouze zkratkou IS, může být ve své obecnosti chápán různě a nemusí být vždy nutně spojován s počítačem a informacemi zpracovávanými v elektronické podobě, i když k tomu dnešní moderní doba dosti nabádá. Např. papírová kartotéka u lékaře může být také považována za IS.

Nicméně tato kapitola i celá práce je zaměřena právě na moderní IS. Tyto mají podobu aplikace nebo více vzájemně komunikujících aplikací provozovaných na konkrétním hardwaru a provádějících určité operace s informacemi uchovávanými v elektronické formě. Zmíněné operace představují rychlé a efektivní zpracování těchto informací tomu kdo systém používá, který tak může pohodlně a jednoduše provádět určité podnikové procesy nebo jejich části.

Takovéto systémy, které lidé při své práci používají, se v dnešní době vyskytují běžně a svým zaměřením mohou spadat do mnoha oborů, přičemž ani správa personální agendy není výjimkou. Právě tímto směrem zaměřené IS budou představeny dále.

1.1 Informační systém pro správu personální agendy

Tyto IS se od ostatních vyznačují především tím, že jsou zaměřeny na práci s tzv. lidskými zdroji. Své využití tedy najdou především u personálních agentur, které za jejich pomoci pomáhají kandidátům najít práci u jednoho ze svých klientů a naopak pro své klienty najít vhodné kandidáty na dostupné pracovní pozice. Stejně tak je ale mohou používat také společnosti v rámci svých oddělení pro řízení lidských zdrojů pro interní správu a vedení informací o jejich zaměstnancích.

V obou případech může být využito řešení, které je vytvořeno přímo na míru podle konkrétních požadavků nebo jedno z komerčně dostupných řešení.

1.1.1 Charakteristické funkce

Mezi charakteristické funkce takto zaměřených IS patří např. ukládání a uchovávání velkého množství informací a dokumentů (v případě personální agentury o již zmíněných kandidátech, klientech a pracovních pozicích), rychlé filtrování na základně mnoha zadaných parametrů, možnost vytvářet náborové procesy (pracovní pohovory) nebo také integraci s dalšími IS, e-mailovými účty nebo klienty jako např. MS Outlook, pracovními portály a sociálními sítěmi. V případě interního použití v rámci jedné společnosti pak tyto IS mohou být rozšířeny o další moduly související např. se mzdami nebo detailnějšími informacemi o zaměstnancích.

1.1.2 Platformy

Stejně jako u ostatních IS může být primárně k dispozici webová aplikace nebo tzv. desktopová aplikace. První uvedená varianta je pro uživatele dostupná v rámci internetového prohlížeče a není nutné ji instalovat. Touto nutností pak disponuje druhá uvedená varianta, která je po instalaci dostupná jako klasická aplikace v počítači.

Tyto primární varianty mohou být také doplněny mobilní aplikací, kterou lze používat na chytrých telefonech nebo tabletech. Často ale nedisponují veškerou funkcionalitou.

1.2 Zástupci informačních systémů pro správu personální agendy

V rámci této kapitoly budou dále uvedeny některé komerčně dostupné IS pro správu personální agendy spolu s jejich klíčovými vlastnostmi a případně také zvolenou cenovou politikou.

1.2.1 OTYS recruitment software

OTYS recruitment software je modulární webová aplikace, které má také svou mobilní variantu. Některé z dostupných modulů jsou zaměřeny právě na požadavky personálních agentur a zahrnují tedy nábor a výběr kandidátů a jejich přidělování k pracovním pozicím dostupných klientů [1].

Mezi jeho vlastnosti patří např. možnost odesílat reporty o náborech přímo jednotlivým klientům nebo kontaktovat klienty a kandidáty pomocí e-mailu nebo SMS [2]. Rovněž se chlubí také možností automatického hledání a oslovování nových kandidátů.

Informace o cenách nejsou na oficiálních webových stránkách dostupné.

1.2.2 Recruit CRM

Recruit CRM je, stejně jako předchozí příklad, webová aplikace spojující správu kandidátů, klientů a jejich pracovních pozic, ovšem bez dalších rozšiřujících modulů.

Zajímavá je např. funkce Radius Search, která díky integraci s Google Maps dokáže vyhledávat kandidáty, klienty a pracovní pozice v určené vzdálenosti od specifikované pozice [3]. Další funkcí je plná synchronizace s e-mailovým účtem, která díky možnostem jako je např. hromadné zasílání e-mailových zpráv nebo vytváření šablon pro tyto zprávy může do jisté míry nahradit e-mailového klienta [4].

Cenová politika tohoto produktu vychází buď z měsíční nebo roční částky za licenci pro jednoho uživatele [5]. V obou dvou případech se částka mění v závislosti na jedné ze dvou dostupných verzí, které se od sebe liší především omezením počtu uživatelů.

1.2.3 Recruitly

Také v tomto případě je Recruitly IS dostupný výhradně přes webový prohlížeč. Svými hlavními vlastnostmi se podobá předchozímu uvedenému příkladu.

Navíc disponuje např. inteligentním vyhledáváním a párováním a rovněž možností již provedená vyhledávání uložit a znovu použít [6]. Uživateli umožňuje také pro každého kandidáta uchovávat dva soubory s životopisem. První interní životopis ve formě, ve které ho kandidát zaslal agentuře a druhý agenturní životopis, pro který Recruitly umí na základě prvního zmíněného automaticky nastavit určitý formát a vzhled. Je tak zajištěno, že personální agentura svým klientům může poskytnout životopisy případných uchazečů v jednotném stylu.

Cenová politika je v tomto případě nastavena na jednotou cenu za licenci pro jednoho uživatele za měsíc používání a všechna funkcionalita je dostupná v rámci jediné verze [7]. V porovnání s přechozím příkladem je cena sice o něco příznivější, ale některé funkce jako je např. příprava a odeslání e-mailových nebo SMS zpráv nebo vytvoření agenturního životopisu, jsou zpoplatněny zvlášť.

1.3 Zhodnocení zástupců

Jednotliví zástupci se od sebe kromě ceny částečně odlišují specifickými funkcemi, které byly uvedeny v rámci jejich předchozích popisů. Každý ze zástupců tedy své využití nalezne tam, kde je na základě těchto funkcí výhodnější než konkurence. Vzhledem k tomu že první zástupce, OTYS recruitment software, je navíc vyvíjen také v České republice, může ho to zvýhodňovat u případných tuzemských zájmenu.

Ve všech případech však platí, že umožňují pracovat s moduly kandidátů, klientů a pracovních pozic. Dalším společným faktorem všech uvedených zástupců je jejich dostupnost v podobě webové aplikace.

2 POUŽITÉ TECHNOLOGIE

Tato kapitola je zaměřena na představení všech technologií, které byly použity v průběhu praktické implementace aplikace.

2.1 Java

Na prvním místě je třeba zmínit samotný programovací jazyk. Je jím Java, objektově orientovaný, multiplatformní jazyk vhodný stejně pro vývoj konzolových aplikací i aplikací s grafickým uživatelským rozhraním.

2.1.1 JavaFX

Právě k návrhu a vývoji grafického uživatelského rozhraní pro webové i desktopové aplikace lze použít platformu JavaFX. Ta byla nejprve ve verzi 1.0 samostatným skriptovacím jazykem a teprve až později se stala nativní knihovnou jazyka Java. Během svého vývoje nakonec přeběhla dnes již méně využívanou knihovnu Swing [8]. V dnešní době již však v JDK od verze 11 automaticky dostupná není a je potřeba ji do projektu pro vývoj dané aplikace zahrnout jako samostatný modul [9].

Pro popis struktury všech prvků grafického uživatelského rozhraní konkrétního okna aplikace se používá tzv. jazyk FXML. Jedná se o značkovací jazyk odvozený od známého jazyka XML a používá tedy párové i nepárové tagy, které lze do sebe vzájemně vnořovat.

Pro nastavení stylů, tedy popis vzhledu jednotlivých prvků lez pak využít CSS, které se používají stejným způsobem jako v případě propojení s HTML. Je tedy možné definovat styly pro vlastní třídy a tyto poté přidělit jednotlivým prvkům v FXML souborech nebo pouze upravit některé z již existujících tříd se jménem začínajícím prefixem „-fx“.

Vše předchozí popsané lze jednoduše a rychle vytvořit např. v návrháři SceneBuilder, kde stačí jednotlivé prvky uživatelského rozhraní myší umisťovat na požadované místo. Program se poté sám postará o vygenerování FXML souboru.

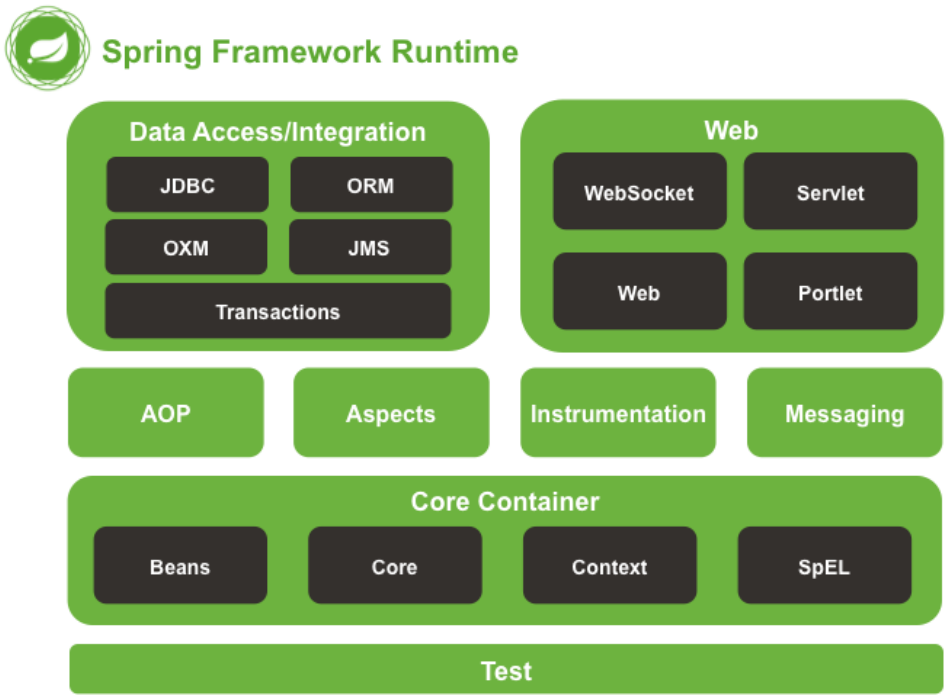
K takto vytvořeným popisům vzhledu aplikace lze přiřadit tzv. kontrolery, což jsou klasické třídy jazyka Java. Tyto jsou již použity k naprogramování metod, které popisují akce spojené s interakcí s prvky v navrženém uživatelském rozhraní.

2.2 Spring framework

Spring je v dnešní době velmi hojně využívaný framework (česky aplikační rámec), který zjednodušuje vývoj webových i desktopových aplikací převážně v programovacím jazyce Java, ale podporuje také jazyky Groovy nebo Kotlin [10]. Velmi dobře se hodí pro vývoj aplikací s tzv. architekturou MVC.

Samotný framework je označován jako modulární, tzn., že je vnitřně rozdělen do přibližně dvaceti modulů, přičemž programátor má volbu v tom, které moduly použije a jak [11]. Tyto moduly jsou sdruženy do skupin, které jsou znázorněny na obrázku číslo 1.

Skupina modulů příznačně pojmenovaná jako Core Container tvoří jádro frameworku a je tedy používána vždy. Za zmínku pak stojí např. moduly Core a Beans, které jsou zodpovědné za tzv. Inversion of Control a Dependency Injection, což jsou hlavní vlastnosti, na kterých je framework Spring založen.



Obrázek 1: Architektura frameworku Spring. Zdroj: [11].

2.2.1 Inversion of Control

Inversion of Control, často uváděno pouze zkráceně jako IoC, je princip vycházející z návrhového vzoru Dependency Injection, zkráceně DI [12]. Jedná se o proces, kdy objekty pouze definují svoje závislosti, ale nejsou již za jejich vytváření zodpovědné. Pod pojmem závislosti jsou v tomto případě myšleny další objekty, které objekt definující závislosti potřebuje ke své práci.

Odpovědnost za vkládání (injektování) zmíněných závislostí je přenesena na samotný framework. Zjednodušeně lze tedy říci, že framework sám umí vytvářet instance tříd a tyto poté vložit tam, kde je to potřeba. Vkládání závislostí lze realizovat např. pomocí parametrů konstruktoru nebo pomocí metody anglicky označované jako tzv. setter.

2.2.2 Beans

Předchozí podkapitola pracovala s pojmem objekt, který je ve spojení s frameworkem Spring poněkud nepřesný. Objekty vytvořené v rámci tohoto frameworku bývají spíše označovány jako Beans.

Tyto jsou frameworkem vytvářeny na základě metadat. Tato metadata pak mohou mít formu buď samostatného XML souboru nebo anotací použitých přímo v naprogramovaném kódu.

2.2.3 Spring Boot

Někdy bývají Spring Boot a Spring mylně považovány za dvojí pojmenování naprosto stejné technologie. Ve skutečnosti se jedná o dva rozdílné názvy, které je třeba od sebe odlišovat, ačkoliv Spring Boot, jak již název napovídá, se samotného Springu samozřejmě týká.

Jedná se v podstatě o nadstavbu nad frameworkem Spring, která ještě více ulehčuje práci, a to především v samotném počátku vývoje, kdy odpadá nutnost projít procesem často dlouhé a složité konfigurace např. již zmíněných bean nebo nastavování zabezpečení atd. [13]. Je zde tedy aplikován přístup nazývaný jako convention over configuration, tedy uplatnění konvence před konfigurací [14]. Manuální konfigurace je tedy nutná pouze v případech kdy automatická konfigurace, kterou Spring Boot připraví sám, nevyhovuje nebo není dostačující.

2.3 Apache Maven

Apache Maven je nástroj využívaný pro správu závislostí aplikací a jejich sestavování (často označované anglicky jako build). Jako výsledek takového sestavení pak lze získat spustitelný balíček s příponou .jar nebo .war v případě projektu vytvořeném v programovacím jazyce Java.

Pro kompilaci a sestavení se používají informace, které jsou obsahem souboru pojmenovaného jako pom.xml, který je uložen v kořenovém adresáři konkrétního projektu [15]. Jednotlivé párové tagy v tomto souboru pak umožňují nastavovat vlastnosti projektu jako např. základní informace o tom, jaký balíček bude výsledkem sestavení aplikace (tag <packaging>), název projektu (tag <name>), aktuální verze (tag <version>) a mnoho dalších.

Jak již bylo zmíněno, tento nástroj umožňuje také správu závislostí. V tomto případě jsou pojmem závislosti myšleny další projekty nebo knihovny, které konkrétní projekt vyžaduje. Všechny potřebné závislosti jsou vnořeny do tagu <dependencies>, přičemž každá z nich obsahuje další informace v rámci svého vlastního tagu <dependency>. Mezi tyto informace patří např. identifikátor závislosti (tag <artifactId>), v předchozím odstavci již zmíněná verze nebo informace o tom, kdy je potřeba danou závislost použít (tag <scope>).

Všechny uvedené závislosti ve specifikovaných verzích Maven při změně v souboru pom.xml nejdříve stáhne nebo je odebere z projektu. Při kompilaci a sestavování aplikace jsou pak již všechny závislosti ihned dostupné.

Maven může být obsluhován pomocí zadávaných příkazů v příkazové řádce, ale základní obsluhu dnes poskytují již i samotná IDE v rámci jejich grafických uživatelských rozhraní.

2.4 Hibernate

Hibernate je jednou z nejznámějších a nejpoužívanějších implementací JPA pro tzv. persistenci dat. Ve své výchozí konfiguraci je tento nástroj využíván právě frameworkem Spring Boot [16].

Své uplatnění nalezne jako tzv. ORM nástroj, tedy nástroj pro objektově relační mapování, který usnadňuje práci s databází. S databázovými záznamy v rámci kódu aplikace umožňuje pracovat jako s klasickými třídami, označovanými jako entity, a opačně na základě dat uložených v těchto entitách umožňuje manipulovat se záznamy v databázi.

Tyto entity lze vytvořit dvojím způsobem [17]. Tím prvním a zastaralejším je vytvoření na základě XML souboru s obsahem popisujícím entitu, tak jak je uvedeno na ukázce kódu číslo 1.

```xml
<hibernate-mapping>
  <class name="entity.Candidate" table="candidates">
    <id name="id" column="id_candidate">
      <generator class="increment" />
    </id>
    <property name="firstName" type="string">
      <column name="fname"/>
    </property>
  </class>
</hibernate-mapping>
```

Ukázka kódu 1: Vytvoření entity pomocí XML souboru. Zdroj: vlastní.

Druhým řešením je použití tzv. anotací přímo v kódu třídy popisující konkrétní entitu. Tyto anotace je možno uvádět buď u jednotlivých atributů nebo u metod, pomocí kterých lze získat jejich hodnotu (tzv. getterů). V kódu, jak je zachyceno v následující ukázce kódu číslo 2, jsou tyto anotace vždy uvozeny znakem zavináč.

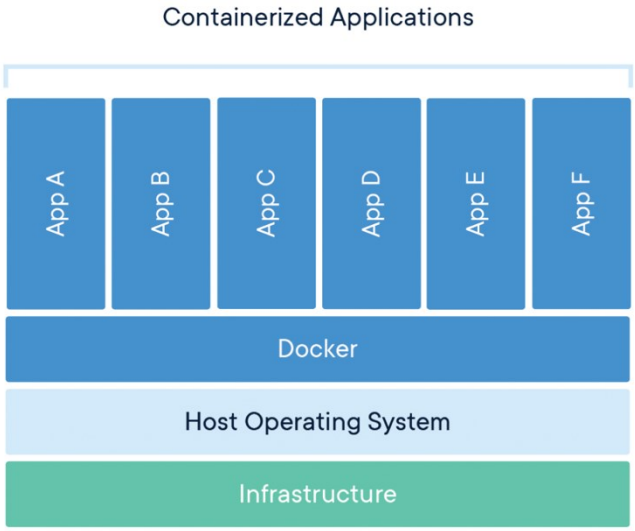
```java
@Entity
@Table(name = "candidates")
public class Candidate {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_candidate")
    private Integer id;

    @Column(name = "fname")
    private String firstName;
}
```

Ukázka kódu 2: Vytvoření entity pomocí anotací. Zdroj: vlastní.

2.5 Docker

Docker je nástroj pro nasazení aplikací izolovaně v rámci tzv. kontejnerů. Každý kontejner obsahuje pouze aplikaci samotnou a prostředky, které bezprostředně potřebuje pro svůj běh [18].

Tento princip fungování Dockeru je označován jako jistá forma virtualizace. Hlavním rozdílem však je, že oproti klasické virtualizaci není v kontejneru zahrnut operační systém, který by v něm umístěná aplikace potřebovala pro svůj běh. Docker a jeho kontejnery tedy využívají pouze hostující operační systém, kterým může být MS Windows nebo Linuxová distribuce. Koncepce Dockeru je nejlépe zachycena na obrázku číslo 2.



Obrázek 2: Koncepce Dockeru. Zdroj: [18].

Z informací uvedených v předchozím odstavci tedy vyplývá, že kontejnery jsou proto dobře přenositelné. Další výhodou takového řešení je malá velikost kontejnerů, a tedy jejich rychlé spuštění nebo ukončení. Takto běžící aplikace navíc nijak nezasahují do hostujícího operačního systému a po svém ukončení v něm po sobě nezanechávají stopy.

Každý Docker kontejner s konkrétní aplikací je vytvořen na základě tzv. image. Tyto image jsou dnes dostupné pro velké množství aplikací, nástrojů, databází atd. Rovněž je také možné publikovat image pro tvorbu kontejnerů s vlastní vyvinutou aplikací.

V případě této práce byl Docker využit pro nasazení databáze.

2.6 MySQL databáze

Za účelem ukládání dat, se kterými aplikace operuje, je v rámci této práce využita relační databáze MySQL. Jedná se o rychlou a především volně dostupnou databázi. Pro práci s ní se jak už název napovídá, používá dotazovací jazyk SQL. Pokud jsou ale pro vývoj aplikace použity právě již zmíněné technologie Spring Boot v kombinaci s Hibernate, pak je možné do jisté míry s databází pracovat bez jakékoliv nutnosti SQL dotazy přímo používat.

2.7 OpenCV

OpenCV je knihovna pro počítačové vidění, zpracovávání obrazu a videa v reálném čase a rozeznávání objektů v obraze [19]. Za těmito účely knihovna poskytuje více než 2500 algoritmů. Používání této knihovny je právě díky využívání těchto optimalizovaných algoritmů jednoduché a především rychlé oproti nutnosti za účelem zpracování obrazu vyvíjet algoritmy vlastní. Jedinou nevýhodou může být větší velikost této knihovny, která má poté dopad na velikost celé aplikace.

V této práci je OpenCV použito právě kvůli jednoduché implementaci rozeznávání objektů, konkrétně obličejů, v obraze.

2.8 Amazon S3

S3 je technologie poskytovaná společností Amazon pro ukládání a správu dat různého druhu. Toto cloudové úložiště je často využíváno také pro archivaci dat, ukládání záloh nebo tvorbu tzv. Data Lake pro analýzu uložených dat [20].

Amazon S3 je s omezeními týkajícími se především dostupné kapacity úložiště a počtu operací, které je možné s uloženými daty provést, dostupné bezplatně [21]. Další varianty bez těchto omezení jsou již zpoplatněny a měsíční poplatek není fixní, ale je vypočítán na základě objemu zpracovaných dat.

Více informací o používání této technologie je uvedeno v kapitole 4.8 zaměřené na správu dokumentů.

3 ANALÝZA A NÁVRH INFORMAČNÍHO SYSTÉMU PRO SPRÁVU PERSONÁLNÍ AGENDY

Tato kapitola je zaměřena na představení analýzy a návrhu praktické části diplomové práce.

Formulace požadavků na funkčnost aplikace a její následnou implementaci vychází z praxe a dalších aplikací, které jsou dlouhodobě používány v oboru personalistiky a správy personální agendy.

3.1 Moduly aplikace

Ještě před uvedením požadavků samotných je třeba uvést, že aplikace stojí na použití tří základních kamenů, dále označovaných jako moduly.

Jedná se o modul kandidátů (uchazečů o zaměstnání), klientů (společností nabízejících zaměstnání) a pracovních pozic (pozic poskytovaných klienty kandidátům). Všechny moduly disponují řadou funkcionalit, přičemž některé jsou pro všechny moduly společné a další naopak pro každý modul specifické. Některé funkcionality jsou navíc omezeny rolí, která je přidělena aktuálně přihlášenému uživateli.

3.2 Požadavky na aplikaci

Pro vývoj aplikace je zásadní vymezení požadavků, které jsou na ni kladeny. Jsou jimi základní funkcionality a vlastnosti, které jsou rozděleny do dvou základních skupin.

První skupinou jsou tzv. funkční požadavky, které, jak už název napovídá, jsou zaměřeny na funkce, které musí aplikace splňovat a které od ní očekává uživatel, který ji bude používat při své práci.

Druhou skupinou jsou pak tzv. požadavky nefunkční, které mají často formu omezení samotného návrhu aplikace, specifikují množství dat, se kterými aplikace musí umět pracovat nebo vymezuje technologie, které budou použity při její implementaci atd.

Zástupci náležící do obou uvedených skupin jsou představeny v následujících podkapitolách.

3.2.1 Funkční požadavky

Tato podkapitola uvádí vybrané klíčové požadavky na funkcionalitu aplikace, které jsou rozděleny do několika skupin.

Přístup do aplikace

- Přihlášení do aplikace,
- odhlášení z aplikace,
- správa uživatelských profilů (vytvoření, úprava, smazání),
- přidělování uživatelských rolí.

Společné funkce modulů

Pokud není v závorce uvedeno jinak, požadavek se vztahuje na všechny moduly.

- Vytvoření nového profilu, uložení úprav existujícího profilu a deaktivace existujícího profilu,
- kontrola požadovaného minima vyplněných informací pro uložení,
- ukládání a mazání dokumentů,
- výpis základních informací všech profilů konkrétního modulu do seznamu,
- výběr řazení základních informací v seznamu všech profilů konkrétního modulu,
- výpis všech informací aktuálně vybraného profilu,
- rychlé vyhledávání,
- rozšířené vyhledávání (na základě většího množství parametrů než u rychlého vyhledávání),
- vytvoření e-mailové zprávy v aplikaci MS Outlook (modul kandidát a klient),
- naplánování události do kalendáře v MS Outlook (modul kandidát a klient),
- vedení interní komunikace (modul kandidát a klient),
- správa všech náborových procesů (modul kandidát a pracovní pozice).

Funkce modulu kandidátů

- Hodnocení kandidáta pomocí systému hvězdiček,
- detekce obličeje v ukládaném dokumentu kandidáta,
- správa interních pohovorů pro každého kandidáta (vytvoření nového, uložení úprav, odstranění),
- export informací o kandidátech do souboru s příponou .xlsx,
- přesměrování na pracovní pozici, pro kterou je vytvořený konkrétní náborový proces spojený s kandidátem,
- kontrola existence kandidáta na základě e-mailové adresy.

Funkce modulu klientů

- Kontrola zadaného IČO klienta s použitím API ARES,
- správa poboček klienta,
- správa kontaktních osob jednotlivých poboček.

Funkce modulu pracovních pozic

- Export informací o pracovní pozici do dokumentu s příponou .docx,
- přesměrování na profil klienta vlastnícího konkrétní pracovní pozici,
- přesměrování na profil kandidáta pro kterého je vytvořen konkrétní náborový proces spojený s pracovní pozicí.

Funkce koše

- Obnovení deaktivovaných profilů (kandidátů, klientů, pracovních pozic),
- trvalé odstranění deaktivovaných profilů (kandidátů, klientů, pracovních pozic) a všech informací s nimi spojenými.

3.2.2 Nefunkční požadavky

V této podkapitole jsou uvedeny nefunkční požadavky, které musí být během vývoje splněny. Aplikace by měla:

- pracovat řádově s tisíci profilů (kandidátů, klientů, pracovních pozic),
- používat MySQL databázi,
- být kompatibilní s daty uloženými v databázi, která je aktuálně využívána jinou, v oboru personalistiky nasazenou, aplikací,
- být multiplatformní,
- dodržovat koncept MVC,
- být implementována v programovacím jazyce Java (JavaFX) ve spojení s frameworkem Spring,
- být dostupná jako tzv. desktopová aplikace,
- umět zkontrolovat dostupnost nové verze aplikace (případně tu tuto verzi stáhnout a spustit její instalaci),
- umožňovat nezávislé využívání více společnostmi – připojení ke správné databázi na základě výběru společnosti.

3.3 Koncept MVC

Jedním z nefunkčních požadavků, je dodržování konceptu MVC. Jedná se o obecnou architekturu, která je založená na rozdělení aplikace do tří logických částí [22]. Jednotlivé části jsou dále krátce popsány a také znázorněny na obrázku číslo 3.

3.3.1 Model

Model je část, která zaštiťuje práci s daty. Pod touto prací si lze představit zpřístupňování dat z databáze, manipulace s nimi a operace a výpočty s těmito daty spojené. Často bývá uváděno, že tato část má na starost tzv. byznys logiku aplikace.

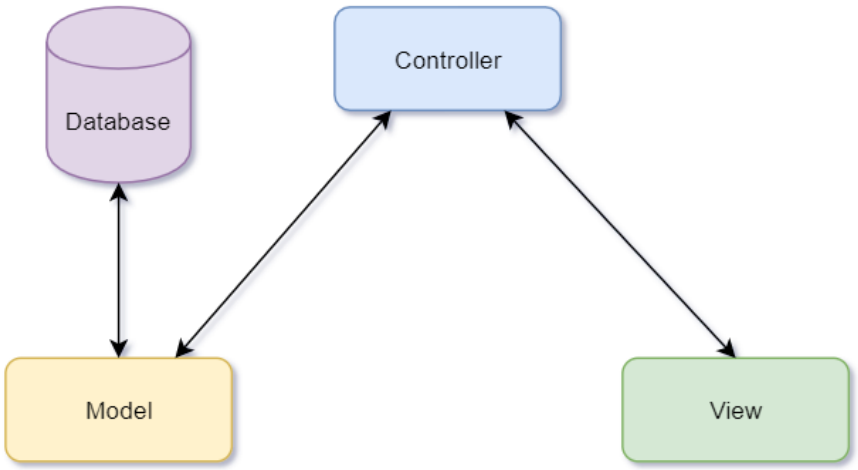
3.3.2 View

View je část, která je zaměřena na interakci s uživatelem aplikace a na prezentaci výstupu na obrazovku. Jedná se tedy o grafické uživatelské rozhraní, které by se nijak nemělo starat o samotnou logiku aplikace.

3.3.3 Controller

Controller je poslední část, která je velmi důležitá, jelikož slouží k propojení a komunikaci dvou předchozích částí Model a View.

Ve chvíli, kdy uživatel interaguje s částí View (např. zadá data do formuláře), je tato interakce zpracována nejdříve právě částí Controller, která poté k jejímu obsloužení použije příslušným způsobem část Model (např. předaná data uloží do databáze). Vše funguje rovněž opačným způsobem, kdy data nebo výsledek z části Model jsou pomocí části Controller předány části View, která se postará o příslušnou vizualizaci.



Obrázek 3: Koncept MVC. Zdroj: vlastní.

3.3.4 MVC a JavaFX

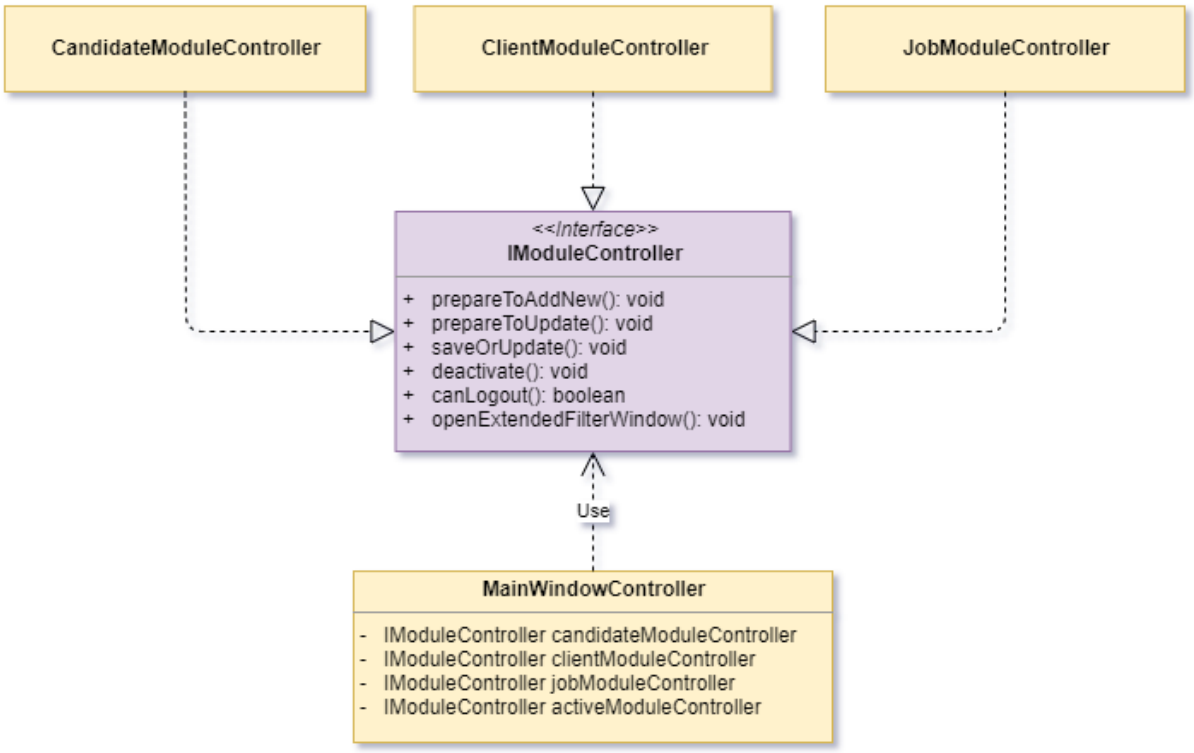
JavaFX sama o sobě dobře podporuje vývoj aplikací s architekturou MVC. Jak již bylo popsáno v kapitole 2.1.1, JavaFX používá tzv. FXML soubor, který popisuje složení prvků, ze kterých se skládá grafické uživatelské rozhraní aplikace. Každý takový soubor tedy náleží právě do části View. S každým FXML souborem může být svázána třída programovacího jazyka Java, která bývá často nazývána jako Controller a náleží tedy do stejnojmenné části. JavaFX tedy již ze své podstaty od sebe odděluje části View a Controller.

3.3.5 Diagramy tříd

V rámci této podkapitoly jsou představeny stěžejní diagramy tříd zpracovaných v jazyce UML. Vzhledem k rozsáhlosti aplikace zde není uveden jeden diagram obsahující všechny třídy, ale uvedené diagramy zachycují několik podstatných částí návrhu a vycházejí z požadavků popsaných na začátku této kapitoly.

Diagram tříd primárních kontrolerů

Na obrázku číslo 4 je zobrazen diagram třídy primárních kontrolerů, které zajišťují chod celé aplikace a jejich princip je dále popsán.



Obrázek 4: UML diagram tříd primárních kontrolerů. Zdroj: vlastní.

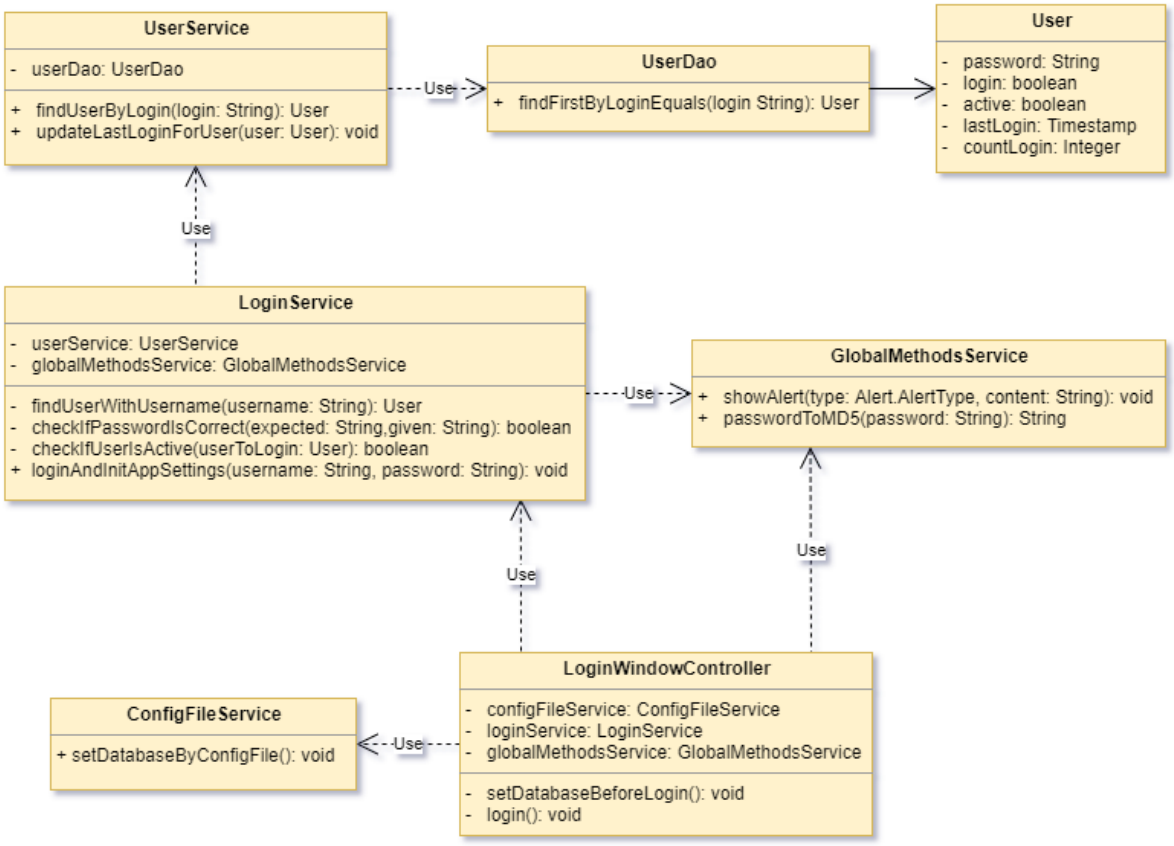
Třída MainWindowController je zodpovědná za chování hlavního okna celé aplikace, především pak za funkce hlavního menu a klávesových zkratek, jejichž důsledkem je přepínání modulů nebo otevírání dalších oken aplikace. Navíc obsahuje kontrolery všech tří modulů (podrobněji popsaných v kapitole 3.1), z nichž ten, se kterým se aktuálně pracuje, je veden jako aktivní.

Práce s těmito kontrolery je poté možná pomocí základních operací (metod), které jim předepisuje implementace rozhraní IModuleController. Jedná se o metody:

- prepareToAddNew pro přípravu modulu na vkládání nové položky,
- prepareToUpdate pro přípravu modulu na úpravu aktuální položky,
- saveOrUpdate pro uložení nové nebo upravované položky,
- deactivate pro deaktivaci (přesunutí do koše) aktuální položky,
- canLogout pro kontrolu, jestli je modul ve stavu, kdy není možné odhlášení,
- openExtendedFilterWindow pro otevření nového okna pro rozšířené vyhledávání.

Diagram tříd zajišťujících přihlášení uživatele

Předchozí diagram samozřejmě strukturu všech navržených tříd velmi zjednodušuje. Ve skutečnosti všechny kontrolery použité v rámci aplikace pro zajištění její logiky na základě konceptu MVC využívají další třídy. Jako příklad vizte UML diagram tříd umožňujících přihlášení uživatele do aplikace, který je uveden na obrázku číslo 5.



Obrázek 5: UML diagram tříd zajišťujících přihlášení uživatele. Zdroj: vlastní.

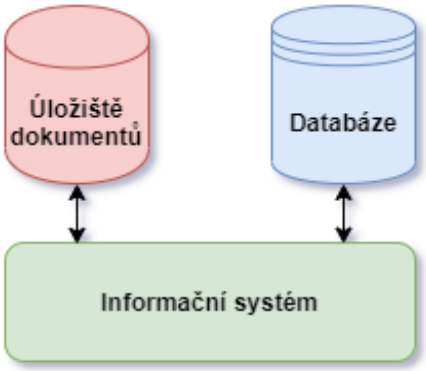
Třída LoginWindowController za účelem přihlášení uživatele nejdříve používá službu ConfigFileService pro načtení potřebných informací z konfiguračního souboru. Poté za využití služby LoginService dojde k samotnému pokusu o přihlášení, kdy je nejdříve potřeba najít uživatele se zadaným uživatelským jménem, ověřit správnost zadaného hesla a zkontrolovat, jestli je účet daného uživatele aktivní a může se tedy přihlásit do aplikace.

Pro nalezení samotných informací o uživateli a práci s nimi je využita služba UserService, která je závislá na třídě UserDao představující repositář pro přístup do databáze. V tomto případě pracuje s třídou User, která na základě ORM představuje samotnou databázovou tabulku.

Poslední v diagramu uvedená služba s názvem GlobalMethodsService obsahuje pomocné metody, které lze využít v mnoha dalších třídách. V tomto případě je použita k zobrazení nového okna s upozorněním pro zpětnou vazbu uživateli v případě chyby (např. špatně zadaných přihlašovacích údajů) a šifrování hesla.

3.4 Koncept informačního systému

Koncept informačního systému je uveden na obrázku číslo 6. Z něj je patrné, že informační systém za účelem splnění všech požadavků nemůže stát pouze samotný, ale musí pracovat s daty, která jsou uložena v databázi a také s vybraným úložištěm pro ukládání dokumentů. Na implementaci a další informace, spojené se všemi na obrázku uvedenými částmi, je zaměřena následující kapitola.



Obrázek 6: Koncept informačního systému. Zdroj: vlastní.

4 IMPLEMENTACE INFORMAČNÍHO SYSTÉMU PRO SPRÁVU PERSONÁLNÍ AGENDY

Tato kapitola se zaměřuje na implementaci informačního systému pro správu personální agendy, která vychází z analýzy a návrhu představených v předchozí kapitole číslo 3.

4.1 Struktura projektu

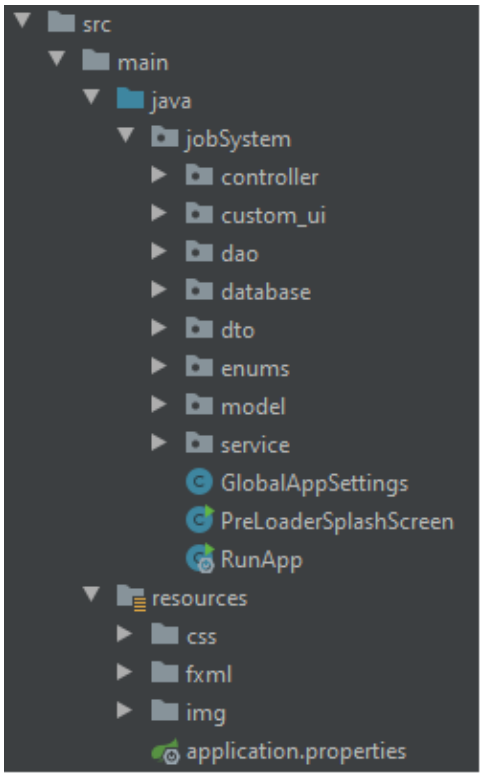
Projekt, v jehož rámci je aplikace realizována, je poněkud rozsáhlý a budou tedy probrány pouze některé části implementace. Pro představu o množství vytvořených tříd a dalších artefaktů je ale dále krátce popsána a na obrázku číslo 7 také zobrazena struktura projektu. Podrobný obsah projektu je také uveden v příloze A.

Nejdůležitější je hlavní balíček jobSystem ve složce java, který obsahuje všechny naprogramované třídy rozdělené podle své koncepce do složek (dalších balíčků):

- controller – třídy implementující část s Controller podle MVC (14 tříd),
- custom_ui – vlastní prvky grafického uživatelského rozhraní (3 třídy),
- dao – třídy rozšiřující JpaRepositry pro přístup k datům v databázi (35 tříd),
- database – pro konfiguraci databáze (3 třídy),
- dto – třídy pro přenos dat (8 tříd),
- enums – pro výčtové typy (14 tříd),
- model – třídy pro reprezentaci dat z databáze ve formě objektů v aplikaci (36 tříd),
- service – třídy (služby) implementující část Model podle MVC (24 tříd).

Tyto třídy pak dále využívají zdroje umístěné v následujících složkách v rámci složky resources:

- fxml – soubory definující strukturu prvků tvořících GUI (17 FXML souborů),
- css – kaskádové styly pro definici vzhledu prvků tvořících GUI (3 soubory),
- img – obrázky zobrazované v aplikaci.

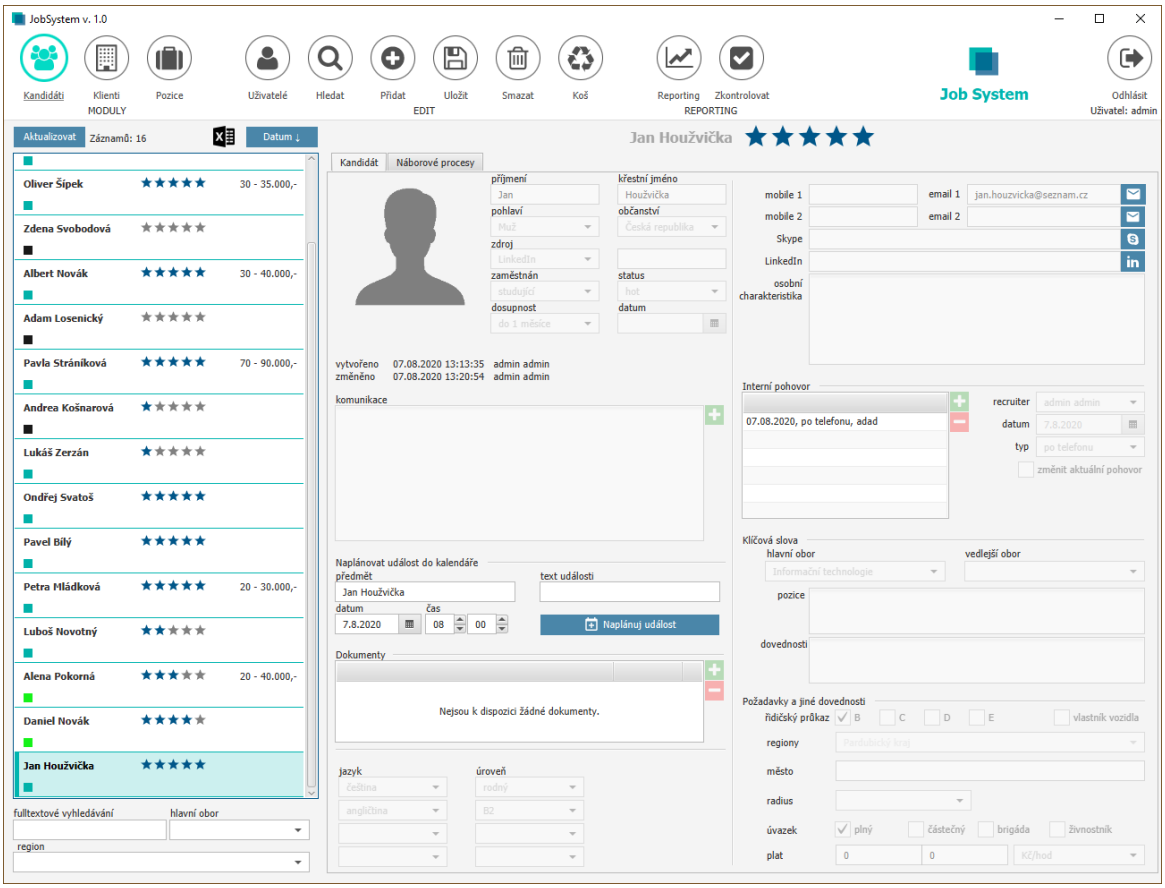


Obrázek 7: Struktura projektu. Zdroj: vlastní.

4.2 Grafické uživatelské rozhraní

Implementace GUI je s využitím velkého množství již předem připravených prvků v rámci JavaFX velmi jednoduchá a rychlá. Vytvořené FXML soubory pro moduly a okna aplikace navíc využívají kaskádové styly upravující výchozí vzhled použitých prvků. Navíc byla využita knihovna FontAwesomeFX umožňující využít množství kvalitních ikon.

Hlavní okno celé aplikace je, jak je vidět na obrázku číslo 8, složeno ze dvou hlavních částí. Tou první je hlavní menu, které je horizontálně umístěno nahoře a obsahuje všechna potřebná tlačítka pro rychlé ovládání aplikace. Druhá, jež je umístěna pod ní, je pak určena k zobrazení všech prvků aktuálně používaného modulu. U všech tří modulů se vlevo nachází prvek ListView, který je použitý k výpisu všech profilů konkrétního modulu. Pod ním je umístěn formulář pro rychlé vyhledávání. V pravé části jsou umístěny prvky pro zobrazení všech informací profilu, který je v ListView aktuálně vybrán. V případě modulu kandidátů a pracovních pozic jsou tyto informace zobrazeny ve dvou záložkách prvku TabPane (v první záložce se nachází základní informace a záložka druhá obsahuje informace o náborových procesech). Další obrázky GUI jsou uvedeny mezi přílohami.



Obrázek 8: Ukázka grafického uživatelského rozhraní. Zdroj: vlastní.

4.2.1 Práce s okny

Práci s okny (v kódu označovanými jako Stage) umožňuje třída StageManagerUIService. Prvky uživatelského rozhraní nejsou umístěny přímo v okně samotném, ale jsou součástí scény, kterou okno zobrazuje.

Vytvoření scény

Následující ukázka kódu číslo 3 obsahuje metodu pro vytvoření scény s využitím FXML souboru, jehož název je předán jako parametr. Samotné vytvoření umožňuje použitá třída FXMLLoader. Za povšimnutí stojí způsob nastavení kontroleru pro novou scénu (řádek 3), který je získán jako Bean z aplikačního kontextu frameworku Spring. Název kontroleru, který má být použit je rovněž obsahem FXML souboru.

```
private Scene createScene(String sceneFxmlFileName) throws IOException {
    FXMLLoader fxmlLoader = new FXMLLoader();
    fxmlLoader.setControllerFactory(springContext::getBean);
    fxmlLoader.setLocation(getClass().getResource( name: "/fxml/" + sceneFxmlFileName));
    Parent rootNode = fxmlLoader.load();

    return new Scene(rootNode);
}
```

Ukázka kódu 3: Vytvoření nové scény. Zdroj: vlastní.

Vytvoření okna

Metoda pro vytvoření okna samotného je zachycena v ukázce kódu číslo 4. Novému oknu je nastaveno několik atributů jako titulek, možnost měnit velikost okna atd. Nejdůležitějším je však v předešlé kapitole zmíněná scéna, která je předána jako parametr.

```
private Stage createStage(String stageTitle, boolean resizable, Modality modality, Scene scene){
    Stage stage = new Stage();
    stage.getIcons().add(new Image( url: "img/Icon.png"));
    stage.setTitle(stageTitle);
    stage.initModality(modality);
    stage.setScene(scene);
    stage.setResizable(resizable);
    stage.sizeToScene();

    return stage;
}
```

Ukázka kódu 4: Vytvoření nového okna. Zdroj: vlastní.

4.2.2 Vlastní prvek GUI

Pro zobrazování seznamu se základními informacemi (kandidátů, klientů, pracovních pozic) v prvku ListView je potřeba použít specifický prvek, kterým JavaFX nedisponuje. Pro tyto účely je tedy vytvořen prvek zcela nový (ve skutečnosti se jedná o 3 prvky – pro každý modul zvlášť). Následující obrázek číslo 9 zobrazuje vzhled tohoto prvku pro zobrazení informací o kandidátovi tak, jak byl vytvořen v návrháři SceneBuilder.

Obrázek 9: Vlastní prvek GUI. Zdroj: vlastní.

Tento prvek má nastavenou třídu kaskádových stylů s názvem listViewCell, pomocí které je drobně upraven jeho vzhled (konkrétně barva a rozměry rámečku). Tato třída je zobrazena v následující ukázce kódu číslo 5 ze souboru s kaskádovými styly s názvem GeneralStyles.

```
listViewCell {
    -fx-border-color: rgb(0, 181, 176);
    -fx-border-width: 0 0 1 0;
}
```

Ukázka kódu 5: Příklad třídy CSS stylů pro úpravu vzhledu prvku GUI. Zdroj: vlastní.

Kontroler, který definuje výpis požadovaných informací v tomto prvku je pak implementován ve třídě s názvem CandidateListCell. Tato dědí od generické třídy ListCell popisující výchozí chování prvku zobrazovaném v rámci ListView. Pomocí kódu v ukázce číslo 6 pak lze ListView nastavit tak, aby k zobrazení informací použil právě vlastní prvek.

```
listViewCandidates.setCellFactory(
    new Callback<ListView<CandidateBasicInfo>, ListCell<CandidateBasicInfo>>() {
        @Override
        public ListCell<CandidateBasicInfo> call(ListView<CandidateBasicInfo> list) {
            return new CandidateListCell();
        }
    }
);
```

Ukázka kódu 6: Nastavení ListView pro použití vlastního zobrazovacího prvku. Zdroj: vlastní.

4.3 Přihlášení a kontrola nové verze

Splnění požadavku na přihlášení uživatele a kontrolu nové verze aplikace zajišťuje kontroler LoginWindowController. Vstupy od uživatele (přihlašovací jméno a heslo) přebírá z okna, jehož struktura vychází z FXML souboru LoginWindow. Aby jakýkoliv kontroler mohl pracovat s prvkem grafického uživatelského rozhraní, musí mít tento prvek v FXML souboru své jednoznačné fx:id. Kontroler pak musí obsahovat atribut stejného jména s anotací @FXML. Příkladem může být ukázka kódu číslo 7, která zachycuje několik takových atributů již zmíněného kontroleru, pomocí kterých jsou zadávány přihlašovací údaje. Pro práci s heslem jsou zde využity dva prvky místo jednoho, protože JavaFX nenabízí prvek pro zadání hesla, který by sám umožňoval přepínat mezi zobrazením zadaného hesla v podobě čitelného textu a ve skryté podobě, kdy jsou znaky nahrazeny např. znaky kolečka nebo hvězdičky.

```
@FXML   // visible by default
private PasswordField passwordField;
@FXML   // invisible by default
private TextField textFieldPassword;
@FXML
private TextField textFieldUsername;
```

Ukázka kódu 7: FXML atributy. Zdroj: vlastní.

Podle potřeb uživatele je vždy jeden z prvků pro zadávání hesla zobrazen a druhý ne. Kontroler pak obsahuje metodu, která heslo během zadávání udržuje stejné v obou těchto prvcích. Tato jednoduchá metoda, zahrnutá v rámci další ukázky kódu číslo 8, je spuštěna po každém uvolnění klávesy u zobrazeného prvku a zadaný text je poté nastaven jako obsah prvku skrytého.

```
private void syncPasswordText() {
    if (!showPassword) {
        textFieldPassword.setText(passwordField.getText());
    } else {
        passwordField.setText(textFieldPassword.getText());
    }
}
```

Ukázka kódu 8: Práce s prvky pro zadávání hesla. Zdroj: vlastní.

4.3.1 Proces přihlášení uživatele

Samotné přihlášení uživatele je obsaženo v rámci metody s názvem login a skládá se ze tří základních kroků.

Nejdříve je na základě konfiguračního souboru nastavena databáze, se kterou aplikace bude pracovat a ve které jsou tedy uloženy informace o uživatelích. (pro podrobnější popis práce s konfiguračním souborem a nastavení databáze vizte kapitolu 4.7.6). Tento krok vychází z požadavku, že by aplikace měla být nezávisle využitelná více společnostmi a tím pádem by se měla při startu umět připojit ke správné databázi.

Následně dochází k samotnému pokusu o přihlášení, což zajišťuje služba LoginService. Nejdříve jsou v rámci metody loginAndInitAppSettings zkontrolovány zadané přihlašovací údaje, přičemž logika této kontroly je blíže popsána již v podkapitole 3.3.5 v části zaměřené na diagram tříd zajišťujících přihlášení uživatele. V případě, že kontrola proběhla v pořádku, dojde k nastavení přihlášeného uživatele a inicializaci nastavení aplikace ve statické třídě GlobalAppSettings.

Posledním krokem je již pouze nahrazení scény pro přihlášení uživatele scénou s hlavním oknem aplikace.

4.3.2 Proces kontroly nové verze aplikace

Jedním z požadavků je také možnost zkontrolovat a stáhnout novou verzi aplikace, pokud je tato dostupná. K tomuto účelu slouží metoda checkNewVersion využívající metod služby VersionService. K jejímu spuštění dochází při inicializaci okna pro přihlášení uživatele, a to pouze v případě, že dostupnost nové verze dosud nebyla zkontrolována a zároveň je konfigurační soubor a jeho obsah v pořádku. Totiž pokud je v pořádku konfigurační soubor, pak je možné se na základě jeho obsahu připojit se ke správné databázi.

Informace o poslední dostupné verzi je k dispozici v databázové tabulce s názvem Version. Tato informace je porovnána s verzí aktuálně běžící aplikace, která je přístupná ze třídy GlobalAppSettings. V případě, že z databáze byly získány informace o verzi novější, je uživatel dotázán, jestli chce tuto verzi stáhnout a nainstalovat. Pokud uživatel souhlasí, je obsah okna změněn na formulář, který pomocí prvku ProgressBar informuje o průběhu stahování. Po dokončení je aplikace ukončena a je spuštěn stažený instalátor.

4.4 Moduly

Tato podkapitola je zaměřena na implementace zajímavých částí jednotlivých modulů, jejichž popis a funkce (funkční požadavky) jsou uvedeny v kapitolách 3.1 a 3.2. Funkce těchto modulů jsou implementovány v kontrolerech:

- CandidateModuleController pro modul kandidátů,
- ClientModuleController pro modul klientů,
- JobModuleController pro modul pracovních pozic.

Stejně jako v předchozích ukázkách v rámci kapitoly 4, pak tyto kontrolery nerealizují své funkce sami, ale využívají pro to některé z řady implementovaných služeb tak, aby byl dodržován koncept MVC.

Výhodou použití frameworku Spring je v tomto případě také to, že pro jednotlivé třídy lze použít tzv. anotace. V tomto případě se jedná o anotace @Controller a @Service. Obě jsou potomkem anotace @Component, nicméně první uvedená označuje třídu s implementací kontroleru a druhá pak třídu s implementací služby. Služby a další třídy, se kterými tyto služby dále pracují pak představují část Model již zmíněného konceptu MVC. Jak již bylo navíc zmíněno v kapitole 2.2.2, na základě anotací může framework Spring pracovat s těmito třídami jako s tzv. Beans.

4.4.1 Práce s dokumenty

Práce s dokumenty je příznačná pro všechny dostupné moduly, ve kterých je implementována jako metoda saveDocument a skládá se z následujících tří kroků:

- výběr dokumentu,
- získání obsahu dokumentu,
- uložení dokumentů na úložiště.

K vytvoření nového okna pro výběr dokumentu je použita třída FileChooser, kterou nabízí JavaFX. Metoda createFileChooser ve službě GlobalMethodService pak umožňuje takové okno vytvořit a použít kdekoliv v aplikaci. Vytvoření okna, jeho otevření a výběr souboru je zachyceno na následující ukázce kódu číslo 9.

```
FileChooser fileChooser = globalMethodsService.createFileChooser( title: "Výběr nového dokumentu pro kandidáta");

File loadedDocumentFile = fileChooser.showOpenDialog(paneCandidate.getScene().getWindow());
```

Ukázka kódu 9: Práce s oknem pro výběr souboru. Zdroj: vlastní.

Pro další práci s již vybraným dokumentem je použita služba s názvem DocumentService, která mimo jiné umožňuje získat jeho obsah pomocí metody getContentFromFile. Pokud má vybraný dokument jednu z přípon .doc, .docx nebo .pdf, je jeho textový obsah uložen do databáze a je využíván během rozšířeného vyhledávání.

V případě dokumentů se zmíněnými příponami lze také volitelně podle potřeby z jejich obsahu získat fotografii s obličejem, pokud taková existuje. Této možnosti je využíváno pouze u modulu kandidátů, kdy pomáhá najít fotografii kandidáta z ukládaného dokumentu (často životopisu) a uložit ji jako profilovou fotografii.

Detekce obličeje je více popsána v další kapitole a třetímu kroku, tedy správě dokumentů na úložišti, se samostatně věnuje kapitola 4.8.

4.4.2 Detekce obličeje v obrázku

Pro rozpoznání obličeje v obrázcích je využita knihovna OpenCV. S její pomocí je implementace této funkce jednoduchá a zabírá jen několik řádků kódu, což dokazuje ukázka kódu číslo 10.

```
OpenCV.loadShared();
CascadeClassifier classifier = new CascadeClassifier();
classifier.load( filename: "./OpenCV/haarcascade_frontalface_alt.xml");

Mat src = Imgcodecs.imdecode(new MatOfByte(picture.toByteArray()), Imgcodecs.CV_LOAD_IMAGE_UNCHANGED);
MatOfRect faceDetections = new MatOfRect();
classifier.detectMultiScale(src, faceDetections);
```

Ukázka kódu 10: Detekce obličeje v obrázku. Zdroj: vlastní.

Stačí pouze knihovnu načíst a následně za pomoci třídy CascadeClassifier načíst tzv. klasifikátor. Jedná se o soubor s příponou .xml, který obsahuje informace napomáhající algoritmu vyhledávat objekty v obraze, v tomto případě lidské tváře. K detekování obličeje slouží metoda detectMultiScale, která pracuje s n-rozměrnými poli. Tím prvním je pole obsahující data konkrétního obrázku. Druhým je pak pole, do kterého metoda ukládá informace o nalezených tvářích v podobě oblastí (obdélníků), ve kterých se tyto tváře nacházejí.

4.4.3 Práce s IČO klienta

V případě vytváření nového klienta v příslušném modulu, je jedním ze zadávaných vstupů tzv. IČO, které musí mít podobu osmimístného čísla. Pokud je zadán jiný znak než číslo, nebo je jeho délka překročena, uživatel je na tento stav upozorněn.

Přímo vedle prvku pro zadávání IČO je umístěno tlačítko pro jeho kontrolu. Tato kontrola se skládá ze dvou kroků. V prvním kroku kontroly je zjištěno, jestli již existuje klient s IČO, které se shoduje se zadávaným. V případě shody opět dochází k informování uživatele, který musí IČO změnit. V případě opačném je proveden druhý krok kontroly, který spočívá v nalezení informací o klientovi s pomocí API ARES. Tento krok je popsán v následující kapitole.

4.4.4 Využití API ARES

ARES je webová aplikace Ministerstva financí, která umožňuje z informačních systémů získávat informace o ekonomických subjektech, jako je např. jméno, adresa a další [23]. Tuto službu lze ale také použít s využitím dostupného API přímo z aplikace. Za tímto účelem byla implementována služba AresService, která obsahuje dvě metody.

Metoda getResponseFromUrl slouží k získání informací o klientovi se zadaným IČO z adresy URL předané v parametru (adresa je konkrétně uvedena a použita v ukázce kódu číslo 11). Získaná data jsou formátu XML a metoda je vrací v textové formě.

Takto získané informace jsou dále zpracovány ve druhé metodě s názvem findClientInfoInAres. Z textu je vytvořen skutečný XML dokument, s jehož obsahem lze jednoduše pracovat a vyhledávat v jeho struktuře pomocí instance třídy XPath. Jednoduchý příklad je uveden v ukázce kódu číslo 11. Všechny vyextrahované informace o klientovi jsou z metody vráceny jako atributy třídy ClientAresInfo, která funguje jako tzv. DTO, tedy objekt, který pouze přesouvá informace mezi dvěma místy aplikace. V tomto případě mezi službou a kontrolerem modulu klientů, který na základě nalezených dat ihned předvyplní formulář o pobočce klienta.

```
String xml = getResponseFromUrl("http://wwwinfo.mfcr.cz/cgi-bin/ares/darv_std.cgi?ico=" + tradeIdToFind);
DocumentBuilder documentBuilder = DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document xmlDocument = documentBuilder.parse(new ByteArrayInputStream(xml.getBytes(Charset.forName("UTF-8"))));

XPath xPath =  XPathFactory.newInstance().newXPath();
String name = xPath.compile( expression: "/Ares_odpovedi/Odpoved/Zaznam/Obchodni_firma").evaluate(xmlDocument);
```

Ukázka kódu 11: Nalezení jména klienta z XML souboru pomocí XPath. Zdroj: vlastní.

4.4.5 Práce s položkami prvku ListView

Jak již bylo uvedeno v kapitole 4.2.2, každý modul pracuje s prvkem ListView pro zobrazení seznamu se základními informacemi všech profilů, které jsou aktuálně označeny jako aktivní, (tedy nebyly deaktivací přesunuty do koše). Nespornou výhodou tohoto prvku je způsob, kterým svůj obsah uchovává a umožňuje s ním pracovat. Pracuje totiž s obsahem v podobě kolekce implementující rozhraní ObservableList. To umožňuje na základě principu návrhového vzoru observer, že jakákoliv změna v kolekci se příslušným způsobem ihned projeví také v GUI. Takovou změnou může být např. seřazení, přidání, odebrání prvků kolekce nebo nahrazení celé kolekce kolekcí obsahující pouze vyhledávané položky atd. Právě jednoduchost změny řazení profilů v modulu pracovních pozic je znázorněna na příkladu z metody sortJobs v ukázce kódu číslo 12.

```
listViewJobs.getItems().sort(Comparator.comparing(JobBasicInfo::getCreated));
```

Ukázka kódu 12: Řazení pracovních pozic v ListView podle data vytvoření. Zdroj: vlastní.

Způsob získání kolekce s požadovanými informacemi o profilech pro naplnění ListView je probrán v kapitole 4.7.5.

4.4.6 Koš

Profily spravované v jednotlivých modulech lze samozřejmě odstranit a to pomocí tlačítka v menu nebo klávesové zkratky CTRL + Delete. Ve skutečnosti se nejedná o odstranění úplné, ale o tzv. deaktivaci, kdy profil již není dále v příslušném modulu zobrazován, v databázi je označen jako deaktivovaný a v rámci aplikace je přesunut do tzv. koše.

Tento koš, implementovaný pomocí kontroleru RecycleBinWindowController, je přístupný v podobě samostatného okna a umožňuje uživateli trvale odstranit nebo zpátky obnovit jeden nebo více vybraných profilů. Pokud je profil odstraněn, jsou odstraněna všechna s ním spojená data v databázi a také příslušné dokumenty. V případě obnovy je pak konkrétní profil opět veden jako aktivní a je ihned zobrazen v modulu, do kterého patří.

4.5 Logování

Aplikace prostřednictvím řady metod služby LogService provádí logování a to ve dvou základních případech.

Prvním je logování úspěšně provedené operace v případě práce s profily (kandidátů, klientů, pracovních pozic), náborovými procesy a dokumenty. Text logu nese informace o datu a času, kdy k operaci došlo, název operace, název profilu se kterým byla operace provedena a jeho identifikátor. Odlišnou podobu má text logu v případě hledání informací o klientovi pomocí ARES (více popsáno v kapitole 4.4.4), který obsahuje pouze datum, čas a výsledek hledání.

Druhým případem je logování výjimky v situaci, kdy došlo k chybě. Text logu obsahuje vždy opět datum, čas, jméno metody, ve které výjimka nastala a její text. Navíc může být doplněn o identifikátor profilu, se kterým uživatel v době chyby pracoval.

Pomocí metody saveLogIntoDatabase, jejíž implementace je znázorněna v ukázce kódu číslo 13, je vytvořena instance třídy Log obsahující text a identifikátor uživatele, pokud je tento přihlášen. Tyto informace jsou nakonec uloženy do databáze.

```
private void saveLogIntoDatabase(String logMessage) {
    Log newLog = new Log();
    if (GlobalAppSettings.loggedInUser != null) {
        newLog.setLoggedInUser(GlobalAppSettings.loggedInUser.getId());
    }
    newLog.setText(logMessage);

    logDao.save(newLog);
}
```

Ukázka kódu 13: Uložení logu do databáze. Zdroj: vlastní.

4.6 Uživatelské role

Každý uživatel používající aplikaci má přidělenu svou roli, na základě které aplikace rozhoduje, ke kterým modulům a funkcím má konkrétní uživatel přístup. Ihned po přihlášení je v rámci inicializace hlavního okna (třída MainWindowController) vybrán modul, který je veden jako aktivní a uživatel tedy uvidí jeho obsah při zobrazení okna. Metodu disableMenuButtonsByUsersRole ve stejné třídě je poté možné použít pro nastavení přístupnosti jednotlivých tlačítek v menu aplikace. S nepřístupnými tlačítky nelze nijak pracovat a tedy nemůže být využita ani konkrétní funkce aplikace nebo modul. Tento fakt je brán v potaz také při používání klávesových zkratek. Klávesová zkratka totiž zprostředkovává konkrétní funkci pouze v případě, že tlačítko pro tuto funkci v menu je přístupné.

4.7 Správa dat

Obsah této kapitoly je zaměřen na představení použité databáze a částí implementace umožňujících připojení aplikace k databázi a správy jejího obsahu.

4.7.1 Databáze

Jak již bylo uvedeno v kapitole 3.2.2, jedním z požadavků je schopnost aplikace pracovat s databází, která je aktuálně využívána v oboru personalistiky. Z tohoto důvodu tedy dále představená databáze přejímá strukturu dat z již používané předlohy, která však prošla několika nezbytnými změnami.

Tyto změny mají podobu úprav a oprav, jelikož na základě analýzy dostupné databáze bylo zjištěno, že její struktura a obsah by nebyl vhodný k okamžitému použití. Došlo tedy k vytvoření několika souborů s příponou .sql obsahujících databázové procedury a scripty, které je nutné nad databází spustit. Přiložená databáze, jejíž model je také popsán v následující kapitole 4.7.2, má již podobu po provedení všech úprav. Nicméně samotné soubory s procedurami a scripty jsou součástí této práce.

Pořadí a popis změn

Celkem bylo vytvořeno 17 souborů na jejichž základě jsou následně popsané změny realizovány a také zkontrolovány.

První potřebnou změnou je vytvoření nových tabulek, které bude aplikace využívat. Konkrétně se jedná se o tabulky Company a Version, které jsou vytvořeny pomocí scriptů obsažených v souborech s názvy CompanyTable_create a VersionTable_create. Pro popis těchto tabulek vizte kapitolu 4.7.2.

Následuje spuštění procedury vytvořené na základě souboru s názvem Changes. Tato procedura realizuje provedení základních změn, mezi které patří např. odstranění nepotřebných záznamů s duplicitními hodnotami identifikátorů, přidání nových sloupců do tabulek, změny datových typů sloupců atd.

V dalším kroku je nutné spustit všechny procedury vytvořené ze scriptů v souborech jejichž název končí na „_repair“. Tyto procedury slouží k opravě chyb, které byly v průběhu analýzy objeveny v jednotlivých tabulkách, přičemž název tabulky odpovídá začátku názvu souboru se scriptem pro vytvoření procedury. V případě tabulky Job má tedy příslušný soubor název Job_repair.

Oprava chyb se v tomto případě týká převážně problému s hodnotami cizích klíčů. Tedy případů, kdy hodnota sloupce v tabulce odkazuje pomocí cizího klíče na záznam v jiné tabulce, který ale neexistuje, a proto hodnota cizího klíče musí opravena na hodnotu existující.

Posledním krokem je spuštění procedur vytvořených pomocí scriptů v souborech s názvem končícím na „_check“. Tyto procedury zkontrolují všechna místa, která předchozí procedury měly opravit a vypíšou počet chyb, které zbývají. Vždy bylo dosaženo požadovaného počtu 0.

4.7.2 Databázový model

Samotný databázový model obsahuje celkem 36 tabulek a zobrazen je na obrázku v příloze E. Vzhledem k množství tabulek si tato kapitola nebere za cíl představit a popsat všechny využívané tabulky. Především pak ne ty tabulky, které plní funkci tzv. číselníků. Zmíněny jsou tedy pouze hlavní tabulky udržující nejdůležitější informace.

Tabulka candidates udržuje informace o kandidátech (např. jméno, příjmení, požadovaný plat, pracovní obor atd.). Jedná se o největší tabulku co do počtu sloupců i velikosti. Nejvíce místa pak zabírají fotografie, které kandidáti mohou mít. Za účelem úspory místa tedy aplikace fotografii před uložením nejdříve zmenší na požadované rozměry, které odpovídají rozměrům prvku, ve kterém se fotografie zobrazuje v GUI.

Tabulka inter_interview pak slouží k uložení informací o tzv. interních pohovorech, které si s kandidátem může sjednat uživatel aplikace. Jedná se o datum pohovoru a jeho typ.

Tabulka client obsahuje základní informace spojené s klienty (např. název, IČO, webové stránky, odvětví působnosti atd.). Dále jsou pak pomocí dalších tabulek client_contacts a client_branch uchovány informace o jednotlivých pobočkách daného klienta a také jeho kontaktech, resp. kontaktních osobách.

Pro uchování informací o pracovních pozicích slouží tabulka job (např. název, popis, požadavky, platové ohodnocení atd.). Každá pracovní pozice je svázána s klientem, kterému patří a může být spojena také s kontaktem na klienta.

Další důležitá je tabulka interview, která obsahuje informace o pohovorech sjednaných kandidátům přímo s klienty na vybranou pracovní pozici (např. jestli a kdy byl odeslán životopis nebo uskutečněn pohovor, výsledek pohovoru, případný nástup na danou pracovní pozici atd.).

Jelikož záznamy většiny již představených tabulek s sebou nesou informaci o tom, kdo je vytvořil a případně upravil, jsou tyto tabulky svázány také s tabulkou user. Tato obsahuje informace o uživatelích aplikace (např. jméno, příjmení, role, login, heslo, počet přihlášení, kontaktní informace atd.).

Jak již vyplývá z kapitoly 4.4.1, aplikace umožňuje pracovat s dokumenty. Za účelem uložení informací o těchto dokumentech, nikoliv dokumentů samotných, pak v databázi existuje tabulka s názvem documents. Každý záznam o dokumentu pomocí sloupce id_source určuje, jestli je určen kandidátovi (hodnota 100), klientovi (hodnota 200) nebo pracovní pozici (hodnota 300). Sloupec id_record je pak hodnota identifikátoru odkazujícího na příslušného kandidáta, klienta nebo pracovní pozici. Důležitý je také sloupec content, který u některých typů dokumentů udržuje jejich textový obsah.

Tabulka company je využita k uchování informací o společnosti, která s aplikací pracuje. Obsahuje název společnosti, její logo a také tzv. bucket pro práci s dokumenty, která je popsána v kapitole 4.8.

Poslední zmíněnou je tabulka version, která obsahuje informace o verzi aplikace a adrese URL, ze které je možné ji stáhnout v případě, že verze aktuálně používané aplikace je nižší.

Soubory s SQL scripty pro vytvoření všech tabulek jsou přiloženy.

4.7.3 Vytvoření entit

Jelikož nástroj Hibernate, představený v rámci kapitoly 4.2, umožňuje používání ORM, je možné při implementaci částí aplikace pracujících s databází používat za tímto účelem třídy programovacího jazyka často označované také jako entity.

K samotnému vytvoření entit lze přistoupit dvojím způsobem. Prvním způsobem je ruční vytvoření entit, na jejichž základě jsou následně automaticky vytvořeny také příslušné tabulky v databázi. Druhým, v této práci použitým způsobem, je tzv. reverse engineering, který spočívá v automatickém vygenerování entit na základě existujících tabulek v databázi. S využitím IDE, které tuto funkci podporuje (v případě této práce se jedná o IntelliJ IDEA), je pak vytvoření všech potřebných entit otázkou pouhých několika minut. Jednoduchý příklad takto vytvořené entity je uveden v kapitole 2.4 v ukázce kódu číslo 2. Všechny vytvořené entity jsou v projektu s aplikací součástí balíčku s názvem model.

4.7.4 Práce s entitami

Aplikace na základě použitého JPA pro manipulaci s entitami používá tzv. repositáře (repositář pro každou entitu zvlášť), které zajišťují tzv. CRUD operace. Každé písmeno této zkratky má následující význam:

- C – create (vytváření nových záznamů),
- R – read (čtení záznamů),
- U – update (modifikace existujících záznamů)
- D – delete (odstranění existujících záznamů).

Jedná se o rozhraní, které je navíc označeno anotací @Repository a dědí od generického rozhraní s názvem JpaRepositry, jehož prvním generickým parametrem je typ entity, se kterou pracuje a druhým pak datový typ identifikátoru.

Pro provádění operací lze využívat některé ze základních zděděných metod jako např. findAll, save, delete atd. nebo je možné definovat vlastní metody pro realizaci operací složitějších. V případě složitějších operací pak metody musí mít správný název, který je složen z jednotlivých klíčových slov. V obou případech však odpadá nutnost samotné implementace těchto metod, nebo přímé uvádění SQL dotazů, jelikož JPA právě na základě názvu metody, návratového typu a případných parametrů vše za programátora udělá. Repositáře však umožňují také zápis operací pomocí SQL jazyka (tento způsob je popsán v kapitole 4.7.5). Příklad repositáře , v tomto případě pro práci s dokumenty, je uveden v ukázce kódu číslo 14.

```
@Repository
public interface DocumentDao extends JpaRepository<Document, Integer> {
    // find all documents with specified sourceId and content containing specified parameter
    List<Document> findAllByContentContainingAndIdSourceEquals(String parameter, int sourceId);

    // find all documents with specified recordId and sourceId
    List<Document> findAllByIdRecordAndIdSource(int recordId, int sourceId);

    // find last saved document with specified recordId and sourceId
    Document findFirstByIdRecordAndIdSourceOrderByCreatedDesc(int recordId, int sourceId);

    // check if exists document with specified name, sourceId and recordId
    boolean existsByIdRecordAndIdSourceAndFilename(int recordId, int sourceId, String documentFileName);
}
```

Ukázka kódu 14: Repositář pro práci s dokumenty. Zdroj: vlastní.

Všechny vytvořené repositáře jsou v projektu s aplikací uloženy v balíčku s názvem dao.

4.7.5 Třídy se základními informacemi

Aplikace v příslušných modulech pomocí speciálního prvku popsaném v kapitole 4.2.2 zobrazuje v rámci prvku ListView seznam se základními informacemi o profilech všech aktivních kandidátů, klientů a pracovních pozic. Záznamy jednotlivých profilů v databázi obsahují velké množství informací (např. tabulka kandidátů obsahuje celkem 59 sloupců). S přihlédnutím k požadavku na schopnost aplikace pracovat s tisíci záznamy o profilech, by pak načítání informací o všech profilech za účelem jejich zobrazení v ListView mohlo být pomalé a načítány by byly zbytečně také informace, které nejsou v seznamu vůbec zobrazeny.

Z tohoto důvodu jsou navíc vytvořeny třídy, které obsahují pouze minimum základních informací o profilech, potřebných k jejich zobrazení v ListView a také pro filtrování. Jedná se o třídy plnící koncept DTO a v projektu s aplikací se nachází ve stejnojmenném balíčku. Jejich název pak končí na „BasicInfo“ (např. CandidateBasicInfo pro základní informace o kandidátech). Z databáze pak tím pádem nejsou načítány informace z nepotřebných sloupců.

Repositáře pak pro načtení informací pro tyto třídy obsahují metody s anotací @Query pro definici vlastního SQL dotazu. Příkladem jsou metody pro získání základních informací o profilech klientů, které jsou uvedeny v ukázce kódu číslo 15 ze třídy ClientDao.

```
String basicInfoColumns = "id_client, name, status, date_created, id_created_by, industry, " +
        "region_list, note, trade_id";

// find basic info of all active clients
@Transactional(readOnly = true)
@Query(value = "SELECT " + basicInfoColumns + " FROM client WHERE activ = 1", nativeQuery = true)
List<Object[]> findAllBasicInfo();

// find basic info of client with given id
@Transactional(readOnly = true)
@Query(value = "SELECT " + basicInfoColumns + " FROM client WHERE id_client = ?1", nativeQuery = true)
List<Object[]> findBasicInfoById(Integer id);
```

Ukázka kódu 15: Metody repositáře pro získání základních informací o klientech. Zdroj: vlastní.

4.7.6 Připojení k databázi

K připojení aplikace k databázi dochází těsně před přihlášením uživatele. Výběr databáze probíhá na základě konfiguračního souboru, který obsahuje název společnosti s jejíž databází by aplikace měla pracovat. Ten by měl být uložen v domovském adresáři uživatele. Pokud tento soubor neexistuje, musí uživatel na přihlašovací obrazovce nejdříve vybrat požadovanou společnost a konfigurační soubor je při pokusu o přihlášení vytvořen. Pro práci s konfiguračním souborem je vytvořena služba s názvem ConfigFileService.

Tato služba pomocí metody setDatabaseByCompanyByConfigFile zajišťuje nastavení správné databáze. Kontext, který udržuje informaci o aktuálně používané databázi a konfigurace připojení k požadovaným databázím implementují třídy z balíčku database. Právě jednotlivé konfigurace připojení jsou pomocí Bean typu DataSource implementovány ve třídě s názvem DataSourceRoutingConfiguration, která je za tímto účelem anotována pomocí @Configuration. Jako příklad nastaveného DataSource vizte ukázku kódu číslo 16.

```
@Bean (name = "dataSourceExample")
public DataSource getDataSourceExample() {
    DataSourceBuilder dataSourceBuilder = DataSourceBuilder.create();
    dataSourceBuilder.url("jdbc:mysql://192.168.99.100:3306/JobSystem");
    dataSourceBuilder.username("root");
    dataSourceBuilder.password("admin");

    return dataSourceBuilder.build();
}
```

Ukázka kódu 16: Příklad konfigurace připojení k databázi pomocí DataSource. Zdroj: vlastní.

4.8 Správa dokumentů

Tato kapitola je zaměřena na práci s úložištěm dokumentů.

4.8.1 Zvolené úložiště dokumentů

Zvoleným úložištěm pro dokumenty je Amazon S3, již uvedený v kapitole 2.8. Pro možnost nahrávat dokumenty na toto úložiště je nejdříve v jednom z dostupných regionů potřeba vytvořit tzv. bucket, který si lze představit jako hlavní složku s unikátním jménem, do které jsou dokumenty ukládány [24]. Pro každý bucket je poté možné nastavit bezpečností opatření. Pro zajištění bezpečnosti uložených dat, je však ve výchozím stavu každý objekt (bucket, dokument) označený jako private, tedy je přístupný pouze tomu, kdo příslušný objekt vytvořil a je tedy jeho vlastníkem [25].

Název každého dokumentu v rámci bucketu může obsahovat znaky lomítka. Klienti, jako např. WinSCP, na základě lomítek pak uložené dokumenty vizuálně strukturují do složek, ale ve skutečnosti není obsah bucketu fyzicky strukturován a nemůže obsahovat další buckety/podadresáře.

4.8.2 Připojení k Amazon S3

K Amazon S3 se lze z aplikace připojit prostřednictvím dostupného API. K tomuto účelu lze využít AWS SDK pro programovací jazyk Java, který lze do projektu zahrnout jednoduše pomocí Maven závislosti.

Využívání klienta k připojení a manipulaci s dokumenty na úložišti je implementováno v rámci služby DocumentService. Následující ukázka číslo 17, obsahuje část kódu konstruktoru této služby pro vytvoření klienta na základě vyžadovaných klíčů access key a secret key a také specifikace kódu regionu (v tomto případě je pro zvolený region Europe (Frankfurt) použit kód „eu-central-1“).

```
BasicAWSCredentials credentials = new BasicAWSCredentials(GlobalAppSettings.accessKey,
        GlobalAppSettings.secretKey);

amazonS3Client = AmazonS3Client.builder()
        .withRegion(GlobalAppSettings.region)
        .withCredentials(new AWSStaticCredentialsProvider(credentials)).build();
```

Ukázka kódu 17: Vytvoření klienta pro připojení k úložišti. Zdroj: vlastní.

4.8.3 Manipulace s dokumenty

S dokumenty lze pomocí klienta zmíněném v předchozí kapitole pracovat velmi jednoduše na základě jejich názvu a bucketu, ve kterém jsou uloženy (příklad metody pro získání dokumentu je znázorněn v ukázce kódu číslo 18). Název bucketu aplikace zjišťuje z databáze (tabulky Company) během procesu přihlášení uživatele a uchováván jako atribut ve třídě GlobalAppSettings.

```
private S3Object getDocumentFromCloudStorage(String name) {
    return amazonS3Client.getObject(new GetObjectRequest(GlobalAppSettings.bucket, name));
}
```

Ukázka kódu 18: Získání dokumentu z úložiště. Zdroj: vlastní.

Název, pod kterým je dokument dostupný na úložišti, je skládán na základě typu profilu, kterému dokument patří (kandidát, klient, pracovní pozice), jeho identifikátoru a názvu souboru. Např. v případě potřeby provést akci s dokumentem text.docx, který náleží kandidátovi s hodnotou identifikátoru 2345, bude dokument na úložišti v příslušném bucketu nalezen pod názvem „Candidates/2000/2345/test.docx“.

4.8.4 Migrace dokumentů

Aplikace pracuje s databází, jejíž podoba vychází z již existující a využívané varianty. Je tedy možné všechna data z původní databáze uchovávat také v databázi nové, jejíž podoba je představena v kapitole 4.7.2. Společně s případnou migrací dat na úrovni databáze pak vzniká také potřeba migrace samotných dokumentů z původního úložiště na úložiště nové. Z tohoto důvodu je ve zvláštním projektu s konzolovou aplikací s názvem DocumentMigrationTool implementován také nástroj pro zmíněnou migraci dokumentů.

Migrace je složena ze čtyř po sobě jdoucích kroků, které jsou implementovány v jednotlivých metodách třídy s názvem DocumentMigrator. Tato pro svou správnou funkci využívá následující připojení:

- připojení k databázi (třída MySQLConnection),
- připojení k původnímu úložišti (třída FTPConnection),
- připojení k novému úložišti (třída S3Connection).

První krok slouží k nalezení dokumentů na původním úložišti, které mají poškozený název. Toto poškození se navzdory tomu, že připojení k úložišti používá kódování UTF-8, může projevit u některých souborů tak, že dochází k problémům se znaky s diakritikou. Soubor je např. fyzicky i v databázi uložen pod názvem operátor.docx, ale klient pro připojení k úložišti se s ním snaží pracovat pomocí názvu oper?tor.docx. V tomto případě soubor nelze soubor migrovat programově. Skutečný název se správnou diakritikou je možné nalézt na základě záznamu o dokumentu v databázi, ale ani pod tímto názvem se souborem nelze pracovat.

Druhý krok zahrnuje samotnou migraci dokumentů na nové úložiště. Při migraci je rovněž upraven název zpracovávaného dokumentu a také jeho název uložený v databázi. V obou případech dochází k odstranění diakritiky, mezer a speciálních znaků kromě pomlčky a podtržítka (za účelem práce s názvy souborů je implementována třída StringFunctions). Do konzole je během migrace vypisován výsledek zpracování každého dokumentu.

Třetí krok je potřebný pouze v případě, že krok první odhalil problémové soubory. Tyto tedy musí být migrovány „ručně“ včetně úpravy názvu dokumentu stejným způsobem jako v předchozím kroku.

V posledním kroku dochází ke kontrole záznamů uložených v databázi. V konzoli je pro každý zpracovaný záznam uvedeno, jestli na novém úložišti skutečně existuje uložený soubor. V závislosti na parametru metody implementující tento krok pak navíc může být záznam, ke kterému neexistuje fyzický soubor, smazán z databáze.

5 NASAZENÍ APLIKACE

V této kapitole jsou probrány kroky potřebné k úspěšnému nasazení aplikace.

5.1 Nastavení

Samotnému nasazení předchází nastavení důležitých parametrů, které jsou pro chod aplikace podstatné. Jedním z nich je nastavení minimálně jednoho DataSource pro připojení aplikace k databázi, a to v rámci třídy DataSourceRoutingConfiguration. Připojení aplikace k databázi společně s příkladem nastaveného DataSource je podrobněji popsáno v kapitole 4.7.6. Každý DataSource je navíc spojený s názvem společnosti, k jejíž databázi se aplikace připojuje. Z tohoto důvodu musí být název společnosti uveden ve výčtu s názvem EnumCompany. V příslušné databázi pak musí být v rámci tabulky Company nastaven název společnosti, její logo a pro práci s dokumenty také název bucketu na úložišti.

Právě pro připojení k úložišti Amazon S3 je dále nutné manuálně nastavit potřebné atributy (accessKey, secretKey, region) v rámci třídy GlobalAppSettings. Vytvoření klienta pro připojení s využitím těchto atributů je uvedeno v kapitole 4.8.2. Nastavení atributu s názvem bucket je provedeno automaticky při přihlašování uživatele na základě informace z tabulky Company zmíněné v předchozím odstavci. Aplikace, která je přiložena k práci tyto atributy nastaveny nemá. V případě potřeby je nutné nastavit jejich hodnoty na základě vlastního účtu k Amazon S3. Z tohoto důvodu je také část kódu vytvářející instanci zmíněného klienta pro připojení uvedena pouze v komentáři. Aplikaci lze tedy spustit i bez tohoto nastavení, ale práce s dokumenty nebude funkční.

Ve zmíněné třídě GlobalAppSettings musí být nastaven také atribut s názvem version pro určení aktuální verze aplikace. Jedná se o řetězec obsahující číslo s přesností na jedno desetinné místo, které se používá pro kontrolu existence novější verze aplikace (pro více informací o kontrole nové verze aplikace vizte kapitolu 4.3.2).

5.2 Vytvoření spustitelného .jar souboru

Pro vytvoření spustitelného souboru s příponou .jar je využit nástroj Maven, konkrétně jeho příkaz mvn package. V rámci projektu je vytvořený soubor uložen do složky target a zahrnuje všechny závislosti uvedené v souboru s názvem pom.xml. Od velikosti a množství všech souborů v projektu a jeho závislostí se odvíjí samotná velikost spustitelného souboru s aplikací. Ta se v tomto případě pohybuje nad hranicí 130 MB. Tato vyšší velikost je způsobena především použitím závislostí pro práci s OpenCV a Amazon S3.

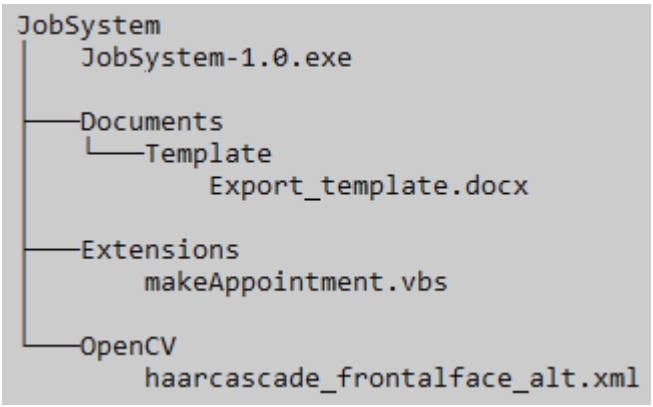
5.3 Vytvoření spustitelného .exe souboru

Ze souboru popsaném v předchozí kapitole je následně vytvořen rovněž spustitelný soubor. V tomto případě má ale příponu .exe, která je pro většinu uživatelů povědomější. K tomuto účelu je využita aplikace s názvem Launch4J, která umožňuje nastavit pro výsledný soubor mnoho volitelných parametrů jako je ikona, nastavení proměnných prostředí, bližší informace o verzi aplikace atd. Hlavním povinným parametrem je pak minimální vyžadovaná verze JRE, která je v tomto případě nastavena na hodnotu 1.8.0.

5.4 Vytvoření instalátoru aplikace

Posledním krokem je vytvoření souboru, který po spuštění provede uživatele procesem instalace aplikace. Instalátor v podobě souboru s příponou .exe je vytvořen pomocí aplikace Inno Setup Compiler. Tato umožňuje nastavit základní informace o aplikaci, které jsou zobrazeny při spuštění instalátoru, výchozí adresář pro instalaci, jazyk, volbu vytvoření ikony na ploše atd. Nejdůležitější je ovšem výběr všech souborů a adresářů, které mají být v instalaci zahrnuty.

Struktura adresářů a souborů použitá v tomto případě, je uvedena na obrázku číslo 10. V kořenovém adresáři se nachází spustitelný soubor vytvořený v předchozím kroku a dále adresáře obsahující soubory, které aplikace bezvýhradně potřebuje pro svou správnou funkci. V adresáři Documents je uložen soubor s šablonou pro export informací o pracovní pozici do textového souboru. Adresář Extensions obsahuje soubor se scriptem pro plánování událostí do kalendáře v aplikace MS Outlook. Poslední adresář s názvem OpenCV obsahuje soubor s klasifikátorem pro rozpoznání lidských obličejů v obraze (jeho použití je popsáno v kapitole 4.4.2).



Obrázek 10: Struktura adresářů a souborů pro instalátor. Zdroj: vlastní.

ZÁVĚR

Návrh a vývoj informačního systému je bez pochyby proces vyžadující nemalé množství času a úsilí. Ne jinak tomu bylo také v případě této práce, která si položila za cíl navrhnout a vyvinout komplexní informační systém pro správu personální agendy.

V rámci této textové části práce byl nejdříve představen pojem informační systém obecně i v kontextu personalistiky a to především s ohledem na charakteristické funkce, kterými disponuje. Rovně bylo také uvedeno několik příkladů zástupců takových systémů, které jsou dnes v oboru komerčně využívány.

Dále se práce soustředila na technologie a nástroje, které byly zvoleny pro samotnou implementaci. Tyto byly blíže popsány z hlediska jejich zaměření a využití, které se prokázalo jako vhodné, jelikož umožnilo v některých případech práci zjednodušit a urychlit. Nejlepším příkladem je využití samotného frameworku Spring, v tomto případě jeho nadstavby Spring Boot.

Následně byly v rámci části zaměřené na analýzu představeny základní kameny aplikace (moduly), výchozí funkční a nefunkční požadavky, které aplikace měla splňovat a také koncept MVC architektury, na základě které bylo nutné přistupovat k následné implementaci.

Právě samotná implementace dostala svůj prostor v následujících kapitolách. Je samozřejmé, že v rámci této diplomové práce nebylo možné postihnout implementaci v celé její obsáhlosti, a proto se primárně soustředila na její stěžejní části, které byly zaměřeny jak na samotnou logiku aplikace, tak na práci s databází a rovněž na správu dokumentů v rámci zvoleného úložiště.

Na závěr bylo také uvedeno tzv. nasazení aplikace. Jedná se o proces na jehož počátku byly vytvořeny zdrojové kódy v projektu a výstupem byl instalátor, který uživatele provádí instalací aplikace.

Ve výsledku se podařilo implementovat desktopovou aplikaci, která splňuje stanovené požadavky a funguje rychle a efektivně. Tím došlo k náplni požadavků diplomové práce v celém rozsahu. Celý projekt aplikace je stejně jako další vytvořené artefakty přiložen k této práci.

POUŽITÁ LITERATURA

[1] OTYS CZ. *OTYS – Nábor a výběr – automatizace procesů pro staffing agentury – Otys CZ* [online]. 2020 [cit. 2020-06-29]. Dostupné z: https://www.otys.cz/agentury

[2] OTYS CZ. *OTYS – Nábor zaměstnanců – automatizace procesů, generování smluv - Otys CZ* [online]. 2020 [cit. 2020-06-29]. Dostupné z: https://www.otys.cz/nabor

[3] WORKFORCE CLOUD TECH, INC. *Radius Search* [online]. 2020 [cit. 2020-06-29]. Dostupné z: https://recruitcrm.io/radius-search

[4] WORKFORCE CLOUD TECH, INC. *Email Integration* [online]. 2020 [cit. 2020-06-29]. Dostupné z: https://recruitcrm.io/email-integration

[5] WORKFORCE CLOUD TECH, INC. *Pricing of Recruit CRM* [online]. 2020 [cit. 2020-06-29]. Dostupné z: https://recruitcrm.io/pricing

[6] RECRUITLY RECRUITMENT SOFTWARE. *Applicant Tracking Software (ATS)* [online]. 2015 [cit. 2020-06-29]. Dostupné z: https://recruitly.io/applicant-tracking-software

[7] RECRUITLY RECRUITMENT SOFTWARE. *Pricing that works for you* [online]. 2015 [cit. 2020-06-29]. Dostupné z: https://recruitly.io/pricing

[8] STEPHENS, Mark. *What is the difference between Java and JavaFX?* [online]. 2014 [cit. 2020-06-30]. Dostupné z: https://blog.idrsolutions.com/2014/11/difference-java-javafx/

[9] KRILL, Paul. *Removed from JDK 11, JavaFX 11 arrives as a standalone module* [online]. 2018 [cit. 2020-06-30]. Dostupné z: https://www.infoworld.com/article/3305073/removed-from-jdk-11-javafx-11-arrives-as-a-standalone-module.html

[10] JOHNSON, Rod, aj. *Spring Framework Overview* [online]. 2002 [cit. 2020-07-01]. Dostupné z: https://docs.spring.io/spring/docs/5.2.x/spring-framework-reference/overview.html#overview

[11] JOHNSON, Rod, aj. *Introduction to Spring Framework* [online]. 2004 [cit. 2020-07-01]. Dostupné z: https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/overview.html

[12] JOHNSON, Rod, aj. *The IoC container* [online]. 2004 [cit. 2020-07-01]. Dostupné z: https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/beans.html#beans-introduction

[13] BAELDUNG. *Spring Boot Tutorial - Bootstrap a Simple App* [online]. 2017 [cit. 2020-07-02]. Dostupné z: https://www.baeldung.com/spring-boot-start

[14] JOHNSON, Rod, aj. *Convention over configuration* [online]. 2004 [cit. 2020-07-02]. Dostupné z: https://docs.spring.io/spring/docs/3.0.0.M3/reference/html/ch16s10.html

[15] APACHE SOFTWARE FOUNDATION. *Maven – POM Reference* [online]. 2020 [cit. 2020-07-02]. Dostupné z: https://maven.apache.org/pom.html

[16] PARASCHIV, Eugen. *A Guide to JPA with Spring* [online]. 2013 [cit. 2020-07-02]. Dostupné z: https://www.baeldung.com/the-persistence-layer-with-spring-and-jpa

[17] *Hibernate Getting Started Guide* [online]. 2020 [cit. 2020-07-02]. Dostupné z: https://docs.jboss.org/hibernate/orm/5.3/quickstart/html_single/

[18] DOCKER INC. *What is a Container?* [online]. 2020 [cit. 2020-07-02]. Dostupné z: https://www.docker.com/resources/what-container

[19] OPENCV TEAM. *About* [online]. 2020 [cit. 2020-07-03]. Dostupné z: https://opencv.org/about/

[20] AMAZON WEB SERVICES INC. *Amazon S3* [online]. 2020 [cit. 2020-07-03]. Dostupné z: https://aws.amazon.com/s3/?nc=sn&loc=0

[21] AMAZON WEB SERVICES INC. *Amazon S3 pricing* [online]. 2020 [cit. 2020-07-03]. Dostupné z: https://aws.amazon.com/s3/pricing/?nc=sn&loc=4

[22] KLÍMA, Tomáš. *Architektura MVC* [online]. 2017 [cit. 2020-07-07]. Dostupné z: http://jakpsatphp.cz/MVC/

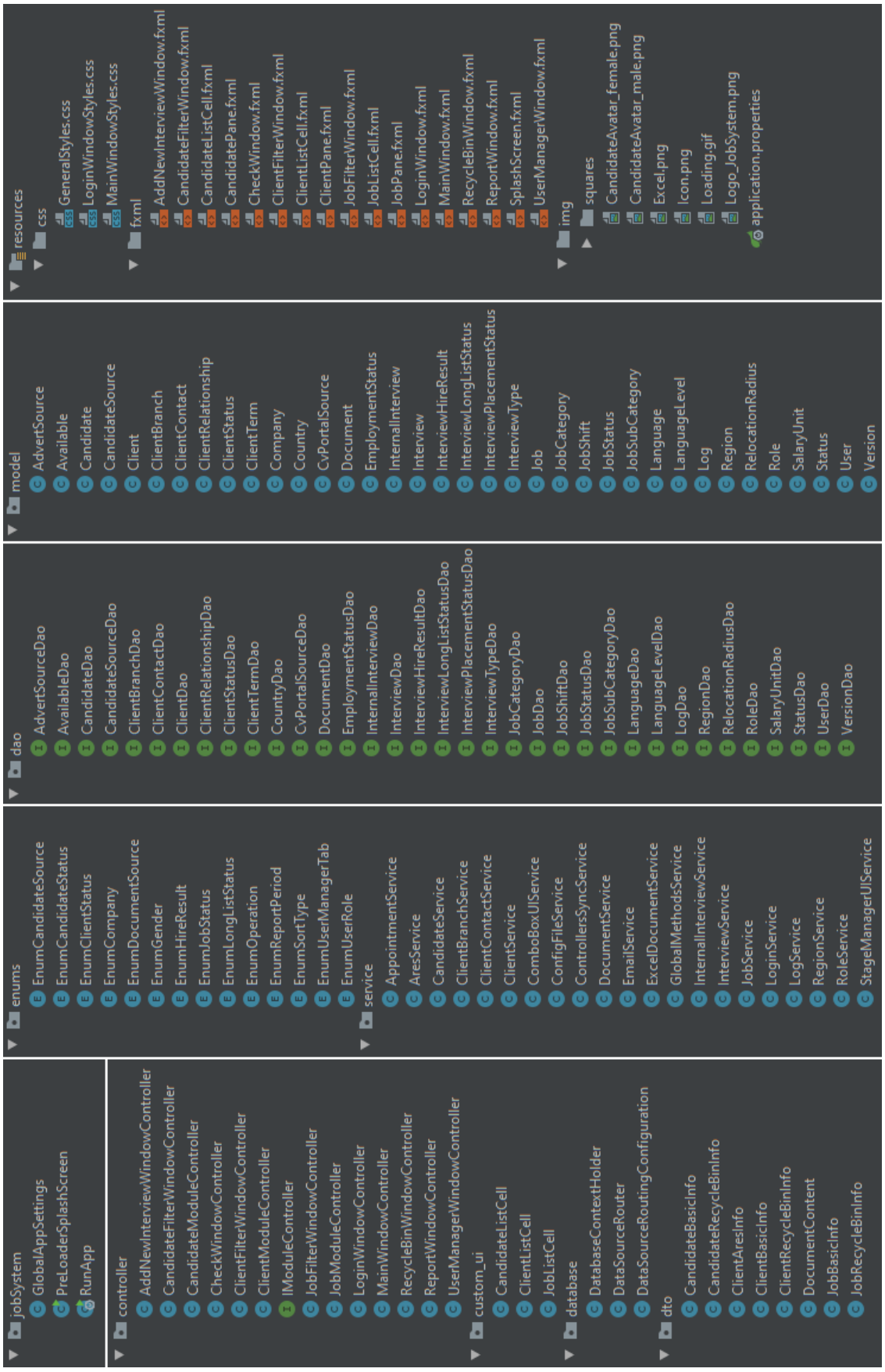
[23] MINISTERSTVO FINANCÍ ČR. *ARES – Ekonomické subjekty* [online]. 2013 [cit. 2020-07-15]. Dostupné z: https://wwwinfo.mfcr.cz/ares/ares_es.html.cz

[24] AMAZON WEB SERVICES INC. *Working with Amazon S3 Buckets* [online]. 2020 [cit. 2020-07-24]. Dostupné z: https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingBucket.html

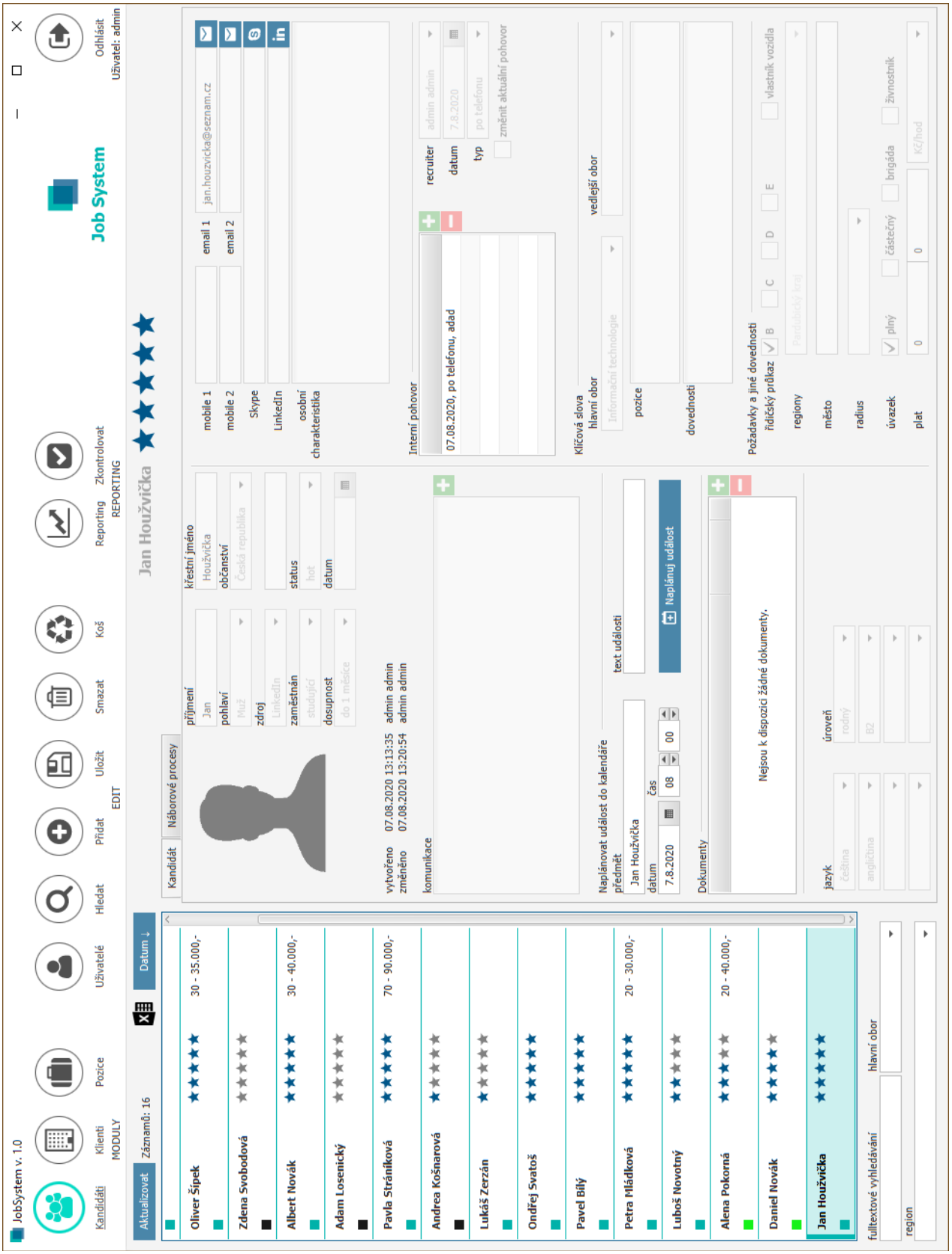
[25] AMAZON WEB SERVICES INC. *Identity and access management in Amazon S3* [online]. 2020 [cit. 2020-07-24]. Dostupné z: https://docs.aws.amazon.com/AmazonS3/latest/dev/s3-access-control.html

PŘÍLOHY

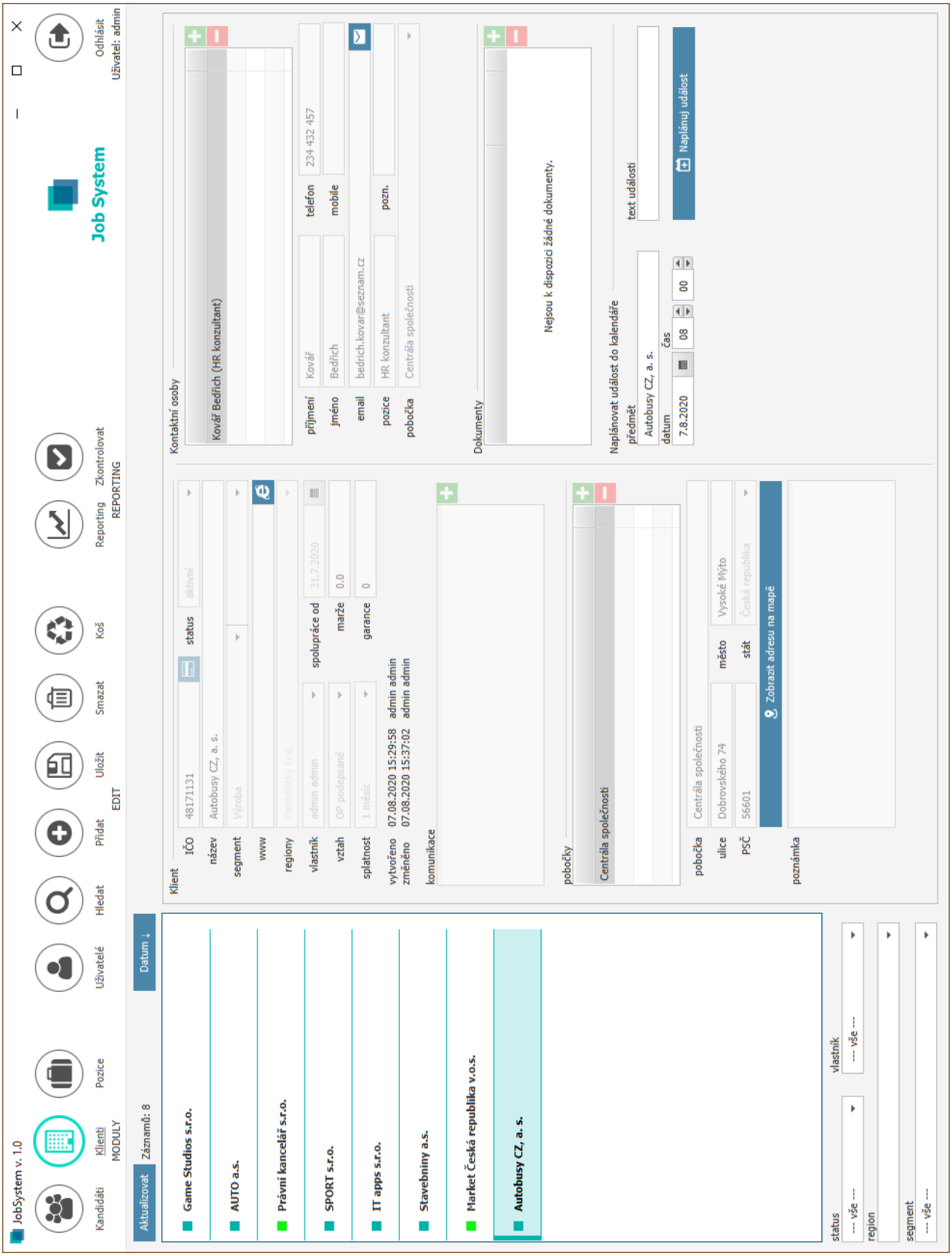
PŘÍLOHA A – OBSAH PROJEKTU



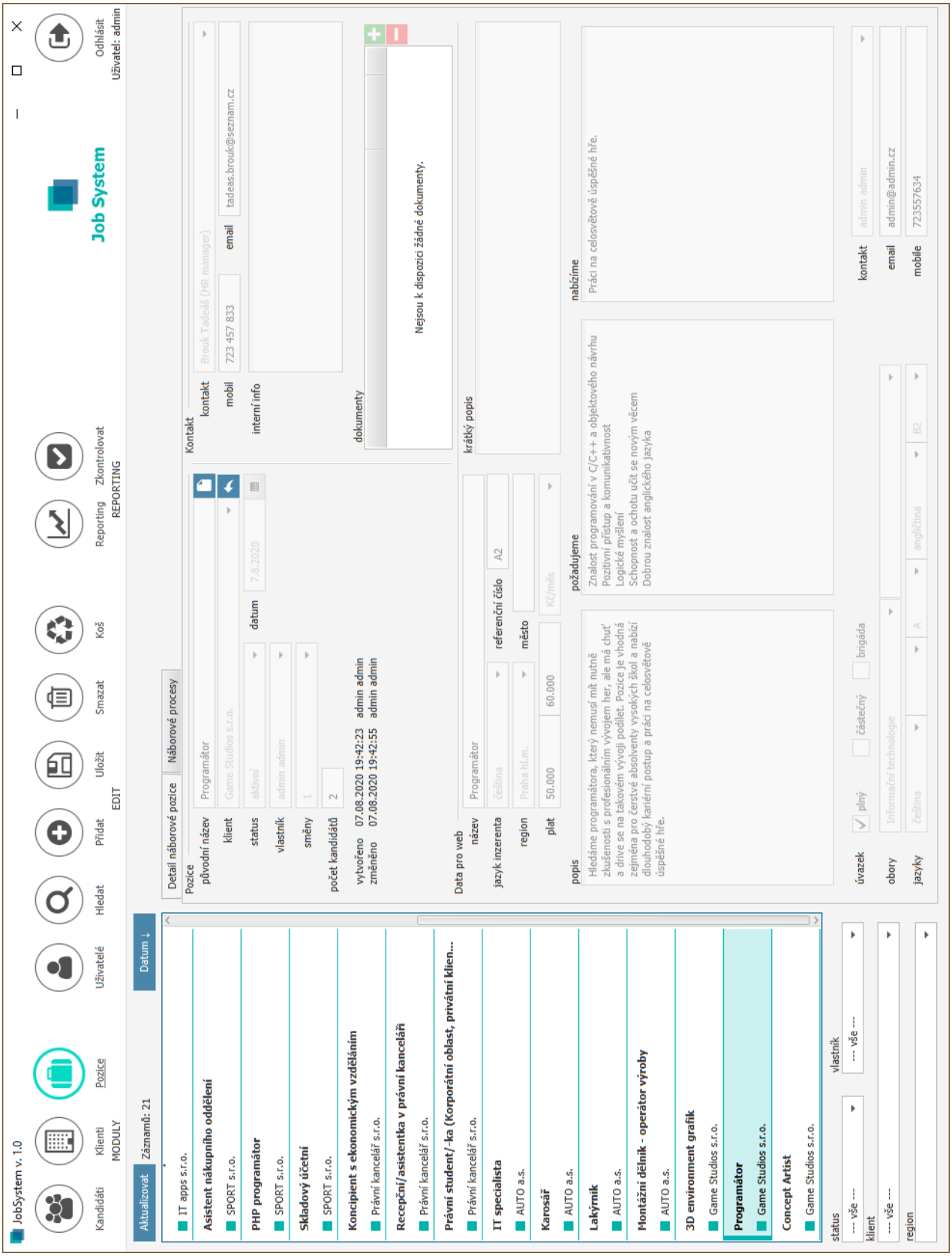
PŘÍLOHA B – UKÁZKA MODULU KANDIDÁTŮ



PŘÍLOHA C – UKÁZKA MODULU KLIENTŮ



PŘÍLOHA D – UKÁZKA MODULU PRACOVNÍCH POZIC



PŘÍLOHA E – DATABÁZOVÝ MODEL