**Univerzita Hradec Králové**
**Fakulta informatiky a managementu**
**Katedra informatiky a kvantitativních metod**

# Vývoj webové aplikace v jazyce Python 3

Bakalářská práce

Autor: Filip Hanš
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Hradec Králové Srpen 2021

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

*vlastnoruční podpis*

V Hradci Králové dne 15.8.2021 Filip Hanš

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Mgr. Tomáši Kozlovi, Ph.D. za jeho ochotný přístup a metodické vedení práce.

Rovněž bych chtěl poděkovat své rodině za její poskytnutou podporu.

# Anotace

Práce se věnuje návrhu a vývoji webové aplikace v programovacím jazyce Python 3. Na začátku bakalářské práce je představen framework Flask a jeho fungování. Dále jsou představeny rozšiřující balíčky pro framework a podpůrné knihovny pro frontend aplikace. Vzorový projekt je pak zaměřen na monitoring insolvenčních řízení obchodních partnerů v České republice. Aplikace využívá pro své fungování spojení s informačními systémy poskytovanými státem. Administrativní registr ekonomických subjektů (ARES) je využíván k čerpání dat o obchodních partnerech. Insolvenční rejstřík (ISIR) pak slouží k čerpání dat o insolvenčních řízeních. Registrovaní uživatelé mají v rámci vlastního účtu možnost přidávat obchodní partnery k sledování a dostávají informační e-maily o případných změnách v insolvenčním rejstříku. Navržená aplikace rovněž poskytuje REST API pro možnou integraci s jinými systémy.

**Klíčová slova:** Python, webová aplikace, framework, Flask, ARES, ISIR

# Annotation

## Title: Development of Web Application in Python 3

The thesis deals with the design and development of a web application in Python 3 programming language. At the beginning of the bachelor thesis the Flask framework and its functioning is introduced. Next, extension packages for the framework and supporting libraries for frontend of application are introduced. A model project is focused on monitoring insolvency proceedings of business partners in the Czech Republic. The application uses for its functioning the connection with information systems provided by the state. The Administrative Register of Economic Entities (ARES) is used to retrieve data on business partners. The Insolvency Register (ISIR) is then used to retrieve data on insolvency proceedings. Registered users within their own account have the possibility to add business partners to be monitored and receive information emails about possible

changes in the insolvency register. The proposed application also provides a REST API for possible integration with other systems.

**Key words:** Python, web application, framework, Flask, ARES, ISIR

# Obsah

# Seznam obrázků

# 1 Úvod

Webové aplikace čelí v poslední době rostoucí popularitě mezi jednotlivci i obchodními firmami a představují tak nedílnou součást každodenního fungování internetu. Díky moderním technologiím jsou webové stránky čím dál více interaktivní a funkčně dosahují stejných možností, jakých se v minulosti dostávalo jen desktopovým aplikacím. S rostoucími nároky je spojena potřeba moderních programovacích jazyků, ve kterých lze takové aplikace programovat. Jedním z takových jazyků je Python, jenž umožňuje psát jak jednoduché, tak i obsahově a funkčně náročné webové aplikace. Je potřeba odlišovat druhou a třetí majoritní verzi jazyka Python, verze totiž nejsou plně kompatibilní a existují velké rozdíly v syntaxi a funkcích. Druhá majoritní verze je v době psaní této práce již bez oficiální podpory, a proto je v práci použita jen třetí verze.

Současný trend rapidního vývoje a co nejrychlejšího převodu myšlenek do produktu nutí vývojáře sáhnout po co možná nejefektivnějším řešení. Python a jeho webové frameworky vynikají svojí rychlostí, přehledností a velkým výběrem doplňkových knihoven. Programátoři tak mohou využít nepřeberné množství pokročilých funkcí bez vysokých nároků na vlastní implementaci. Vzhledem k tomu, že Python a většina jeho knihoven je distribuována jako volně šiřitelný kód, není potřeba řešit dodatečné náklady na pořízení technologie. Přehledná syntaxe je mimo jiné vhodná pro začínající vývojáře.

Tato bakalářská práce poskytuje přehled o současných možnostech využití jazyka Python pro programování malých až středně velkých webových aplikací za pomoci frameworku Flask. Za použití získaných informací z teoretické části je popsán návrh a implementace vzorové aplikace „Monitoring insolvencí“, která pravidelně prochází údaje z insolvenčního rejstříku. Vytvořená aplikace má za cíl zpřístupnit uživatelům možnost sledování insolvenčních řízení obchodních partnerů. Aplikace je v maximální míře napsána v jazyce Python 3 s využitím frameworku Flask a dalších podpůrných knihoven.

# 2 Literární rešerše

Na základě dostupných online zdrojů a přístupu k vybraným plnotextovým vědeckým databázím v rámci studia na Fakultě informatiky a managementu Univerzity Hradec Králové byla autorem práce připravena literární rešerše dostupných odborných článků. Výběr zdrojů byl mimo jiné uskutečněn s důrazem na jejich aktuálnost a relevantnost vůči tématu této bakalářské práce.

## *2.1 Proměna desktopových aplikací na webové*

Jedním ze současných trendů je transformace desktopových aplikací na webové aplikace. Tento přechod je poháněn změnou uživatelských zařízení, preferencí použití a zvyšujícími se nároky na uživatelský komfort.

Jung a kol. (2002) se ve svém článku věnují efektivním metodám vývoje webových systémů. V první části autoři analyzují základní požadavky na systém a navrhují metodiku sběru dat z konkrétních doménových oblastí (firemní strategie, business prostředí, interní procesy, data, uživatelské rozhraní a technologické prostředí). Analýza provedená autory se skládá z následujících fází: definice strategie a požadavků webu, vytvoření modelu obsahu a konfigurace, vytvoření funkčních modelů stránek, analýzy rozhraní aplikace a systémové struktury. Z analýzy požadavků autoři generují podrobný plán nasazení webové aplikace. Plán zohledňuje vnitřní uspořádání částí systému a návrh uživatelského prostředí. Autoři očekávají, že jejich práce bude nápomocná při vývoji informačních systémů, jelikož umožňuje efektivně popsat obsah webu pomocí metod vytváření podrobných plánů.

Současné trendy vývoje webových aplikací směřují ke konceptu celkového zjednodušování. To se projevuje jak v moderním UI (user interface, česky „Uživatelské prostředí“), tak i v kódu samotném. Nedílnou součástí tohoto konceptu je mimo jiné i důraz na bezpečnost a produktivitu vývoje.

## *2.2 Formy webových aplikací*

Pro sjednocení uživatelského prostředí aplikace v prohlížeči a nativní aplikace na mobilním zařízení se v současnosti začínají prosazovat čím dál tím populárnější

PWA (Progressive Web Application, česky „Progresivní webové aplikace“), které kombinují oba přístupy do jedné implementace. Richard a LePage (2020) poukazují na to, že takové aplikace zvyšují spokojenost uživatelů a usnadňují jim orientaci. Daná aplikace může následně využívat vyšších oprávnění na hardware a je možné ji instalovat na systém uživatele.

Dalším trendem je Single-Page Websites (česky „Jednostránkové webové stránky“). Tento koncept se vyznačuje svojí jednoduchostí a obsažením všech dat na jediné stránce. Na druhou stranu nelze tento koncept aplikovat na veškeré stránky. Rozdíly mezi mnohostránkovou a jednostránkovou aplikací popisuje Adiseshiah (2018). Autor uvádí výhody a nevýhody obou řešení. Jednostránkové aplikace jsou podle autora vhodné k prezentaci uceleného obsahu o konkrétním tématu bez složitějšího řazení. Vyznačují se vysokou mírou uživatelského zapojení a obsahují většinou jednu akci, například odeslání formuláře. Mnohostránkové aplikace se pak vyznačují možností zobrazení komplexních souborů témat a struktur. Autor však upozorňuje, že na rozdíl od jednostránkových aplikací jsou takové aplikace pro uživatele méně přehledné.

## 2.3 Webové služby

Webové aplikace již dávno neslouží jen interakci s uživateli. Mezi další funkce patří výměna dat mezi systémy. Tímto vzniká potřeba webových služeb. Webové služby jsou nabízeny a konzumovány primárně mezi systémy. Pokud nebyly služby součástí aplikace od jejího vzniku, Kerdoudi a kol. (2016) navrhují postup přidání webových služeb do webových aplikací pomocí analýzy jejich uživatelského rozhraní. Jako příklad autoři uvádí přidání webové služby do e-shopu. V rámci analýzy autoři hledají všechny případy užití aplikace v uživatelském prostředí. Nalezená užití přepisují jako koncové body nové webové služby. Ta je následně implementována jako další vrstva aplikace. Autoři poukazují na přínos webových služeb pro komunikaci mezi dvěma systémy, kdy aplikace nemusí řešit složité a neefektivní HTML interakce a získávají zvýšenou efektivitu díky možnosti pokročilých funkcí, které by byly pro člověka v uživatelském prostředí neintuitivní.

Distribuce webových aplikací přes síť patří mezi základní funkci, ačkoliv zajištění kvality takto poskytovaných služeb je často opomíjeno. Pro řešení tohoto problému Nikolaidou a Anagnostopoulos (2015) navrhují použít systémový přístup k analýze a efektivní konfiguraci samotné webové aplikace, systému, hardwaru a sítě, na které bude aplikace provozována. Ve fázi konfigurace webové aplikace Nikolaidou a Anagnostopoulos (2015) zkoumají její architekturu. Autoři následně ve svém článku přicházejí s návrhem na optimální konfiguraci, kterou prezentují pomocí notace podobné UML (Unified Modeling Language, česky „Unifikovaný modelovací jazyk“) a nazvané Web-based System Modeler (česky „Modelář pro systémy založené na webu“). Následuje fáze logické konfigurace. Tato fáze se zaměřuje na širší pohled užívání aplikace. Autoři analyzují, ve kterých lokacích bude aplikace užívána a provozována. Z výsledků vzniká návrh umístění provozních lokací aplikace. Jakmile jsou lokace vybrány, přichází na řadu fáze fyzické konfigurace. Ta řeší fyzickou topologii sítí a parametry spojení mezi uživateli a servery poskytujícími aplikaci. Za použití modelu topologie sítí autoři navrhují vytvoření série testů a identifikaci komponent ke sběru relevantních dat v síti pro ověření správné funkčnosti návrhu a systému. Autoři mimo jiné provedli příkladovou studii zaměřenou na implementaci jejich výše popsaného postupu v rámci vývoje webové aplikace pro National Diabetes Institute (česky „Národní diabetologický institut“).

Využívání webových služeb má však i svá úskalí. Jak poukazují Kahlon a kol. (2016), konzumace více takzvaných partnerských služeb může mít značně negativní důsledky na QoS (Quality of Service, česky „Kvalita služeb“), a to v případě, kdy se partnerské služby mění či jsou úplně nedostupné. Tyto problémy mohou vyústit v chyby v aplikaci, která dané služby konzumuje. Autoři rozebírají možnosti kontroly QoS u partnerských služeb a z toho vyplývající zlepšení QoS služby konzumentské. Klasickými variantami jsou kontroly typu service-client (česky „Služba-klient“), v jejichž případě probíhá kontrola při volání samotné služby, nebo typu server side (česky „Na straně serveru“), kdy je partnerská služba kontrolována v periodických intervalech serverem. Autoři navrhují vlastní variantu kontroly pomocí distribuovaného agentového systému, který má oproti klasickým řešením výhodu ve skoro nulové přidané režii u volání služeb. Jak ale

autoři přiznávají, současná implementace takového systému je problematická z hlediska velkého množství souběžných volání.

## 2.4 Bezpečnost při návrhu a vývoji webových aplikací a služeb

Každá webová aplikace či služba bude nakonec nevyhnutelně cílem škodlivého útoku. Proto je v současnosti nutné řešit bezpečnost vyvíjeného řešení. Muthukrishnan a kol. (2019) poukazují na nutnost zabezpečení webových služeb, ať už jsou služby na bázi SOAP (Simple Object Access Protocol), nebo REST (Representational State Transfer). Pro služby typu REST autoři doporučují:

- jako minimální variantu zabezpečení používání uživatelské autorizace a TLS (Transport Layer Security),
- jako pokročilejší variantu implementaci bezpečnostního frameworku třetí strany, např. OAuth2.

Pro služby postavené na bázi SOAP autoři doporučují implementaci efektivních šifrovacích algoritmů, mechanismus přístupových práv, efektivní monitoring a validaci zpráv.

Pokud je aplikace napsána v programovacích jazycích, které nejsou kompilovány do binární podoby (příkladem může být jazyk Python), je třeba se bránit i proti manipulaci s kódem samotným. Peng a kol. (2019) poukazují na nutnost ověřování spouštěného kódu vůči kódu, o němž víme, že je nekompromitovaný.

Nedílnou součástí vývoje aplikací je i následné nasazení do produkce. Praktická hardwarová virtualizace je už několik desítek let standardem ve sdílení výkonu mezi více systémy. Poslední roky se však objevují nové možnosti než jen virtualizace celého operačního systému. Hlavním takovým trendem je kontejnerizace. Vzhledem k tomu, že obě varianty mají skoro stejné koncové užití, je nasnadě otázka, které z řešení je výhodnější z hlediska funkčnosti. Tuto otázku řeší Chae a kol. (2017), kteří porovnávají užití mezi KVM (Kernel-based virtual machine) a Docker kontejnery. Autoři připravili testovací prostředí na jednom virtuálním systému a jednom kontejneru, kde testovali identické aplikace.

Prostřednictvím testu autoři k závěru, že aplikace provozované v kontejneru běžely rychleji a vyžadovaly celkově méně zdrojů hardwaru pro svůj běh.

Kontejnerizace je v současnosti užívána i v největších společnostech a projektech na světě. Například, jak poukazují Verma a kol. (2015), společnost Google Inc. používá na všechny svoje služby kontejnery. Navíc vyvíjí i vlastní orchestrační službu Kubernetes.

## 2.5 Vývoj webových aplikací

Jazyk Python se již mnoho let umisťuje mezi nejpoužívanější programovací jazyky současnosti. Tento trend je zřejmý u TIOBE Indexu (2021), kde se Python umisťuje mezi top pěti nejpoužívanějšími programovacími jazyky už řadu let. Poslední průzkumy dokonce naznačují, že bude v následujících letech celkově nejpoužívanějším jazykem vůbec. V době psaní této práce byl jazyk Python na druhém místě za jazykem C a k prvnímu umístění jej dělilo jen 0,7 %. V rámci webového vývoje je situace velice podobná. I v tomto případě je jazyk Python velice populární. O'Grady (2021) srovnává jazyky podle jejich popularity v rámci vývoje webů. Zde je na první pozici jazyk JavaScript, který v současnosti dominuje poli webového vývoje. Hned v závěsu je výše zmíněný Python.

Rozdělení vývoje webových aplikací na frontend a backend je v současné době již standardem. Python je převážně užíván k vývoji backendu, ale lze ho použít i v případě frontendu. Pramanick (2021) srovnává v současnosti nejpoužívanější frameworky pro vývoj backend služeb. Na nejvyšších pozicích se nachází velmi populární Python framework Django a Flask. Jak autor poznamenává, rozdíl mezi frameworky je už od začátku značný. Django cílí na jednoduchost vývoje, na rozdíl od Flask, který je určen k rapidnímu vývoji bez zbytečných omezení.

Počátek vývoje nových webových aplikací vyžaduje rychlé prototypování, jelikož se funkce, vzhled i rozsah mohou měnit na denní bázi. Jak poukazují Dolgert a kol. (2008), tento přístup má mnoho výhod, avšak aby byl efektivní, je nutné použít vhodné programovací jazyky a jejich komponenty. Jako příklad autoři uvádí výběr spojení programovacího jazyka Python a technologie AJAX (Asynchronous

JavaScript and XML). Jako hlavní faktory výběru těchto technologií autoři předkládají dvě hlavní výhody: rychlé provádění změn, včetně provádění interakce s databázemi, a jejich skoro okamžitá propagace u uživatelů. Autoři následně popisují vybrané komponenty z daných technologií. Jako hlavní komunikační vrstvu s databázemi volí balíček SQLAlchemy, který zajišťuje ORM (Object Relational Mapper) a abstrakční vrstvu mezi různými databázovými backendy. Dalšími komponentami jsou CherryPy a Cheetah template Framework, jež zajišťují běh aplikace a prezentační vrstvu. Autoři na závěr popisují vybrané AJAXové frameworky pro zajištění pokročilých funkcí v prezentační vrstvě. Rapidní vývoj za použití těchto technologií a komponent podle autorů vyústil ve velice produktivní řešení. Jeho jednoduchost, flexibilita a použitelnost umožnily rychlý přepis komponent a modelů aplikace.

Na základě provedené rešerše byl autorem stanoven cíl, který je dále upřesněn v následující kapitole.

# 3 Cíl práce

Cílem této práce je představit využití programovacího jazyka Python 3 při tvorbě webové aplikace, včetně příslušných webových služeb, která umožní monitoring insolvenčních řízení vybraných smluvních partnerů. Hlavním účelem aplikace je tedy sledování zvolených obchodních partnerů na přítomnost insolvence. Vhodný návrh by měl umožnit přihlášenému uživateli vytvořit v rámci účtu vlastní seznam partnerů, kteří budou automaticky zařazeni do sledování. Aplikace by měla mimo jiné mít přehledné uživatelské rozhraní.

# 4 Teoretická část

V této části autor uvádí zpracované primární a sekundární zdroje informací, které tak tvoří nezbytný teoretický základ této bakalářské práce. Uvedená kapitola obsahuje stručný popis základních stavebních prvků a částí webové aplikace, charakteristiku programovacího jazyka Python 3 a popis státních registrů interagujících s aplikací.

## *4.1 Moderní vývoj webových aplikací*

V současnosti je vývoj webových aplikací utvářen dominancí návrhového vzoru MVC (Model-View-Controller, česky „Model-Pohled-Řadič“), který separuje jednotlivé části aplikace do samostatných oddílů. Vzor byl sice původně navržen pro desktopové aplikace, ale hlavní osvojení získal v návrhu aplikací pro web. S příchodem moderních webových frameworků, jako jsou například Flask, Django nebo Rails, byl návrhový vzor modifikován do formy MTV (Model-Template-View, česky „Model-Šablona-Pohled“). Změna primárně spočívá ve skutečnosti, že řadič již není potřeba psát přímo v kódu, ale je řešen ve frameworku samotném. Návrhový vzor je následně aplikován v rámci modelu klient-server, který je v současné formě webu de facto standardem (Shadhin, 2021).

### 4.1.1 Frontend

Jedná se o část aplikace, která řeší zobrazení a funkci aplikace na straně klienta. Pro uživatele je výstupem uživatelsky přívětivé grafické rozhraní. Jelikož je jeho zobrazení řešeno přímo u klienta, musí být ze serveru poskytnuta všechna potřebná data pro vykreslení a správnou funkčnost. Jako hlavní technologie řešící frontend u webu lze uvést:

- značkovací jazyk HTML (HyperText Markup Language),
- kaskádové styly CSS (Cascading Style Sheets) a
- programovací jazyk JavaScript.

Díky kombinaci těchto technologií a jejich nadstaveb lze uživateli zobrazit interaktivní a vysoce personifikované prostředí. S vysokou mírou interaktivity

přichází však i možnost nežádoucího chování ze strany uživatelů, které může vyústit až v kompromitaci aplikace. Právě rozdělení vrstev poskytuje ochranu před výše zmíněným nežádoucím chováním. Vzhledem k tomu, že je frontend plnohodnotně uložen na straně klienta, může uživatel manipulovat s jeho obsahem a upravovat jeho fungování. Takové úpravy však zůstávají jen na straně klienta a nepromítají se na celou webovou aplikaci. V případě správné implementace je tím ochráněna konzistence interakcí a dat v aplikaci (Pastorino, 2021).

### 4.1.2 Backend

Jedná se o část aplikace, která řeší všechny akce na straně serveru. Pro uživatele je daná část neviditelná a umožňuje jen řízenou komunikaci. Backend zajišťuje interní fungování webové aplikace. To s sebou obnáší podávání a zpracování dat na frontend. Jelikož jsou moderní aplikace dynamické a jejich uživatelé vyžadují maximální míru interakce s webovými stránkami, je nutné podávat všechna relevantní a předzpracovaná data na požádání. To samé platí v případě zpracování poskytnutých dat z interakcí uživatele s frontendem. Na straně backendu se také provádějí případné validace poskytnutých dat tak, aby byla chráněna celková integrita dat a aplikace. Backend je vždy napsán v některém ze skriptovacích nebo kompilovaných jazyků, například Python, PHP (Hypertext Preprocessor, česky "Hypertextový preprocesor"), Java a dalších. Zařizuje také komunikaci s dlouhodobými úložišti dat, primárně relačními databázemi, případně s datovými úložišti založenými na formě klíč-hodnota (Pastorino, 2021).

## *4.2 Python a web*

Python je objektově orientovaný vysokoúrovňový programovací jazyk, navržený s důrazem na čitelnost, rychlost psaní a čistotu kódu. Čitelnost je docílena omezením používání oddělovacích znaků, jako je například středník či hranaté závorky. Na rozdíl od oddělovacích znaků používá Python odsazení kódu mezerami.

Jazyk byl vytvořen Guidem van Rossumem v 80. letech minulého století a do širšího povědomí se dostal až s nástupem verze 2, jež byla vydána v roce 2000. Na ni navazuje v současnosti podporovaná verze 3, která prošla značnou revizí při

přechodu mezi verzemi. Díky obsáhlé revizi není verze 3 plně kompatibilní s verzí 2 (Python Software Foundation, 2021).

Python je distribuován takzvaně batteries included (česky „s přibalenými bateriemi“), což znamená, že mnoho základních i pokročilých funkcí je již dostupných v základní knihovně jazyka. I přesto je jazyk modulárně navržen tak, aby bylo možné doplnit jeho možnosti podle potřeb uživatele. Python se v současnosti těší velké popularitě, a to i díky nepřebernému množství doplňkových modulů. Ty jsou distribuovány většinou zdarma pomocí balíčkovacího systému pip (Pip installs packages, česky „Pip instaluje balíčky“) (Python Software Foundation, 2021).

### 4.2.1 Framework Flask

Flask patří mezi hlavní webové frameworky napsané v jazyce Python. Vyznačuje se minimalistickým přístupem, a je proto označován jako micro framework. Minimalismus se v tomto případě vyznačuje tím, že v základním balíčku je jen naprosté minimum potřebné k provozování webového serveru. To však neznamená, že se uživatel musí s daným omezením smířit. Framework je navržen ve stejném duchu jako Python, a to na bázi rozšiřitelnosti. Proto existuje k frameworku Flasku velké množství rozšiřujících balíčků, které přidávají skoro jakoukoliv funkci. Nejpoužívanějšími rozšířeními jsou například abstrakce práce s databázemi pomocí ORM (Object–relational mapping, česky „Objektově relační mapování“), generování formulářů, management uživatelských relací a spousta dalších. Framework tím dává velkou svobodu v tom, co je v aplikaci opravdu potřeba a co je naopak zbytečná zátěž. Jednou z mála závislostí je šablonovací systém Jinja, starající se o dynamické generování HTML obsahu z šablon pro zobrazení u uživatele. Nakonec framework potřebuje ke svému nasazení do produkce WSGI (Web Server Gateway Interface) kompatibilní webový server, jako jsou například Apache, Nginx nebo Gunicorn (Pallets, 2010a).

Vzhledem k tomu, že je framework Flask postaven z malých částí, skvěle podporuje rapidní vývoj softwaru. Programátor má také maximální kontrolu nad komponentami aplikace. To samozřejmě přináší i určité nevýhody. Vývojář tak musí přesně a zároveň předem vědět, co chce s aplikací provést. Framework totiž

vyžaduje více času a práce pro správnou implementaci komponent, v případě využití více komponent lze narazit na nekompatibilitu. Flask je proto vhodný pro jednodušší webové aplikace, případně pro velké aplikace využívající menší množství rozšíření.

### 4.2.2 Flask rozšíření

Každé rozšíření bylo nutné do projektu nainstalovat a importovat. Nejjednodušší formou byla instalace přes balíčkovací systém pip z repositářů PyPy.

#### 4.2.2.1 Flask-SQLAlchemy

Jedná se o integrační rozšíření, které do aplikace přidává podporu pro ORM balíček SQLAlchemy. Díky němu je možné využít abstrakce nad relačními databázemi a plně se tak oprostit od psaní SQL (Structured Query Language, česky „Strukturovaný dotazový jazyk“) dotazů. Celá struktura databáze byla napsána jako objekty v aplikaci, nad nimiž lze volat příslušné metody, které jsou přeloženy do jazyka SQL (Pallets, 2010b).

#### 4.2.2.2 Flask-Login

Rozšíření přidává do aplikace podporu managementu uživatelských relací. Zajišťuje základní funkce přihlašování, odhlašování a uživatelské relace. Relace jsou spravovány na základě zabezpečených cookies (česky „sušenka“), které jsou uloženy na koncových zařízeních pro možnost udržení aktuálnosti spojení (Max Countryman, 2021).

#### 4.2.2.3 Flask-WTF

Umožňuje integraci mezi aplikací a balíčkem WTForms. Ten obstarává generování, validaci a zabezpečení formulářů na webu. Dané se hodí u jakýchkoliv interakcí s uživatelem, kde je zapotřebí vyplnění dat. Balíček umožňuje nastavení pokročilých validací všech polí ve formuláři, včetně propagace chyb k uživateli (WTForms, 2010).

#### 4.2.2.4 Flask-Mail

Přidává možnost odesílání e-mailů z aplikace přes SMTP (Simple Mail Transfer Protocol, česky „Jednoduchý protokol pro přenos mailů“). Podporuje také integraci s šablonovacími jazyky, je tedy možné generovat e-maily v HTML podobě (Matt Wright, 2014).

### 4.2.3 Podpůrné knihovny

Podpůrné knihovny byly v projektu využívány v různých částech. Knihovny jsou zde přiblíženy kvůli jejich nepostradatelnosti v aplikaci.

#### 4.2.3.1 Boostrap

Jedná se o frontend CSS framework s důrazem na responzivní webový design. Upravuje uživatelské prostředí pomocí typografie, vzhledu formulářů, tlačítek a navigace. Obsahuje i scripty v jazyce JavaScript, díky kterým umožňuje rozšíření interaktivity webových stránek s uživatelem. Výhodou uvedeného frameworku je nezávislost na backendu aplikace. Implementace se provádí v HTML kódu stránky a ke customizaci se používají HTML atributy class (Bootstrap, 2021).

#### 4.2.3.2 DataTables

Knihovna má za cíl zpříjemnit uživatelům práci s tabulkami a daty obecně. Díky vylepšením jsou již v základu implementovány pokročilé možnosti tabulek jako například:

- řazení záznamů podle sloupců,
- hledání v záznamech tabulky,
- stránkování.

Vše je uživatelsky sjednocené díky vzájemné integraci s frameworkem Bootstrap. Knihovna DataTables dále podporuje řešení pokročilých funkcí jak na straně klienta, tak na straně serveru, což šetří zdroje, hlavně v případě zobrazování velkého množství dat (SpryMedia Ltd., 2021).

#### 4.2.3.3 Requests a Zeep

Requests (česky „požadavky“) umožňuje jednoduše provádět HTTP (Hypertext Transfer Protocol) požadavky na vzdálené servery bez zbytečného řešení řetězců dotazů. Navíc disponuje pokročilými funkcemi, jako je používání cookies, stahování souborů v datovém proudu a další (Ken Reitz, 2016).

Na Requests staví knihovna Zeep, která vytváří programové rozhraní nad SOAP (Simple Object Access Protocol) službami. Po načtení a inspekci WSDL (Web Services Description Language) služby připraví volatelné objekty, které lze použít pro komunikaci se službou. Pro programátora tak odpadá nutnost řešení správného složení XML (Extensible Markup Language) pro volání a serializace odpovědí služeb (Michael van Tellingen, 2016).

## *4.3 Databáze*

Databáze je základním prvkem organizovaného persistentního ukládání a načítání dat v počítačových aplikacích. Data jsou ukládána na základě klíčů, pomocí nichž lze následně data také načítat. Dominantním typem jsou relační databáze. Ty řadí data do tabulek, sloupců a řádků. Většina relačních databází implementuje pro práci s daty jazyk SQL (Structured Query Language) (Oracle, 2021).

### 4.3.1 MySQL

Jedná se o jeden z nejstarších databázových systémů, který byl vydán jako systém s otevřeným kódem. Díky tomu je možné užívat systém bezplatně, ale také bez aktivní podpory. Napsán je v jazyce C a C++ a je dostupný na mnoha operačních systémech. Podporuje procedury, triggery, cursory, DDL (Data Description Language) a mnoho dalších funkcí. V současnosti vývoj zajišťuje Oracle Corporation, která nabízí MySQL i v placené variantě s aktivní podporou (Oracle Corporation, 2021).

## *4.4 Informační systémy poskytované státem*

V následujících kapitolách jsou představeny informační systémy ARES a Insolvenční rejstřík. Jedná se o státní registry, které jsou použity v rámci webové aplikace pro monitoring insolvencí vybraných obchodních partnerů.

### 4.4.1 ARES

Administrativní registr ekonomických subjektů (ARES) je veřejný informační systém provozovaný Ministerstvem financí České republiky. Systém agreguje data z vybraných registrů státní správy. ARES poskytuje obsáhlé informace o ekonomických subjektech registrovaných v České republice. Systém má grafické uživatelské prostředí dostupné na stránkách http://wwwinfo.mfcr.cz/ a aplikační rozhraní XML (Extensible Markup Language) služeb, jejichž popis je dohledatelný na stránkách http://wwwinfo.mfcr.cz/ares/xml_doc/schemas/index.html (Ministerstvo financí České republiky, 2013).

Služby informačního systému ARES je možné konzumovat přes SOAP protokol nebo pomocí HTTP metod GET a POST na konkrétní URL (Uniform Resource Locator). Jelikož by mohla být celá služba zneužívána k robotickému těžení dat, jsou upraveny její provozní podmínky. Hlavním omezením je limit možných dotazů na služby v době od 8:00 do 18:00 hodin na 10 000 dotazů a v době od 18:00 do 8:00 hodin na 50 000 dotazů. Ostatní omezení jsou dohledatelná v Podmínkách provozu služby na stránkách http://wwwinfo.mfcr.cz/ares/ares_podminky.html.cz. Získaná data ze systému jsou ve formátu XML a jsou omezena podle doptávané části ARESu. Schéma XML používá textové zkratky pro snížení náročnosti přenosů na datové linky. Tyto zkratky je následně nutné převést pomocí dodaného slovníku zkratek (Ministerstvo financí České republiky, 2013).

### 4.4.2 ISIR

Informační systém veřejné správy – Insolvenční rejstřík (ISIR) je rejstřík s relevantními informacemi o insolvenčních řízeních a jejich průběhu. Systém je provozován Ministerstvem spravedlnosti České republiky od roku 2008. Do rejstříku zapisují rejstříkové soudy zákonem stanovené údaje o subjektech v právě aktuálním insolvenčním řízení. Jeho grafickou uživatelskou část lze najít na stránkách https://isir.justice.cz/isir/common/index.do. Rozhraní pro automatické sledování insolvenčního rejstříku je pak dostupné na https://isir.justice.cz/isir/common/stat.do?kodStranky=SLEDOVANIWS (Ministerstvo spravedlnosti České republiky, 2021).

V současné chvíli jsou ještě stále dostupné dvě webové služby ISIR WS_1 a ISIR_CUZK_WS2. Obě jmenované poskytují koncové body pro strojové dotazování do rejstříku, jsou založeny na SOAP protokolu a poskytují ucelený pohled na insolvenční řízení. Služba ISIR WS_1, která byla zprovozněna ve stejný čas jako systém samotný, je v současné době nerozvíjena a pro moderní užití je již skoro nepoužitelná. Hlavní nevýhody jsou její pomalost a dotazování na ID akcí, které jsou špatně dohledatelné. Proto služba slouží spíše k zajištění zpětné kompatibility pro již napojené systémy. ISIR_CUZK_WS2 byla zprovozněna v roce 2014 jako nástupce ISIR WS_1, a to s důrazem na vyřešení problémů první webové služby. Hlavní vyhledávací parametry jsou IČO (Identifikační Číslo Osoby), rodné číslo nebo jméno a příjmení, která jsou na rozdíl od ID akcí hledateli známá. Dále je služba optimalizovaná na strojová vyhledávání, a proto i tisíce dotazů nejsou omezovány. Vstupy i výstupy služby jsou ve formátu XML, ale schéma je čitelné a jednoduše pochopitelné (Ministerstvo spravedlnosti České republiky, 2021).

## 4.5 Vývojové prostředí

Vývojovým prostředím je označován software usnadňující psaní a práci s programovým kódem. V současné době již není v lidských možnostech udržet krok s extrémně rychlým vývojem podpůrných balíčků a frameworků k programovacím jazykům. Proto jsou takzvaná IDE (Integrated Development Environment, česky "Integrovaná vývojová prostředí") hojně využívána ke zrychlení vývoje aplikací a usnadnění vývoje pro programátory. Vzhledem k tomu, že jazyk Python není kompilován, je teoreticky možné v tomto jazyce psát i v nejjednodušších textových editorech. Na rozdíl od nich specializovaná IDE nabízí nepřeberné množství výhod pro vývojáře.

### 4.5.1 PyCharm

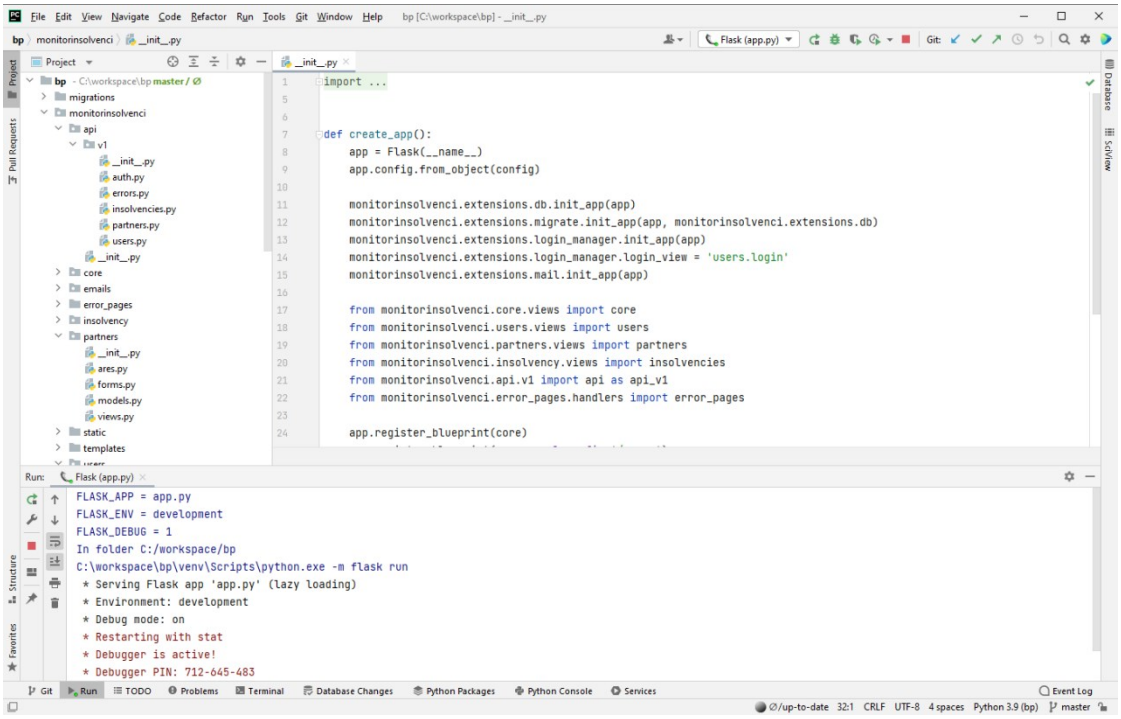
Produkt PyCharm Professional Edition od společnosti JetBrains s.r.o. byl autorem vybrán pro vývoj vzorové webové aplikace. Tato verze byla využita v rámci školní licence autora.

Společnost JetBrains s.r.o., dříve známá jako IntelliJ Software s.r.o., je česká firma, založena v roce 2000, specializující se na vývoj vývojových prostředí. Její

nejslavnější IDE je IntelliJ IDEA, které je určeno pro vývoj programů v jazyce JAVA a jeho derivátech. V roce 2010 společnost poprvé představila dedikované vývojové prostředí pro jazyk Python PyCharm (JetBrains s.r.o., 2021).

PyCharm je v současnosti považován za jeden z nejlepších a nejkompletnějších vývojových nástrojů, který nabízí ucelené prostředí pro všechny současné trendy ve vývoji python aplikací. IDE je nabízeno ve dvou edicích: Community a Professional. Edice se liší jak cenou, tak i některými funkcemi, které jsou dostupné jen v placené verzi Professional. Nedostupné funkce pro verzi Community sice neovlivňují funkčnost vyvíjené aplikace, ale snižují uživatelský komfort, případně mohou zpomalit vývoj samotný. Ke hlavním funkcím IDE PyCharm patří (JetBrains s.r.o., 2021):

- chytré doplňování kódu, včetně analýzy kódu,
- pokročilý grafický debugger,
- redaktorování skrze celý projekt,
- nativní integrace se systémy správy verzí, jako je např. Git,
- zvýrazňování chyb v kódu, včetně návrhů k opravám,
- integrace jednotkových testů a vizualizace pokrytí kódu,
- v Professional Edition podpora webových frameworků Flask a Django.




**Obrázek 1: Ukázka vývojového prostředí PyCharm**
Zdroj: Autor

# 5 Praktická část

V následujících kapitolách popisuje autor vlastní postup při tvorbě vzorové aplikace v jazyce Python 3.

## *5.1 Analýza současného prostředí (konkurence)*

V okamžiku psaní této práce již byla na českém trhu dostupná řada služeb nabízejících monitoring insolvenčních řízení obchodních partnerů. Ke společným rysům všech těchto služeb patří:

- možnost vložení obchodních partnerů jednotlivě nebo hromadně,
- funkcionalita periodické aktualizace stavů insolvenčních řízení (zpravidla jednou denně),
- možnost odeslání upozornění uživateli v případě změny v sledovaném insolvenčním řízení (přímo na stránkách služby, případně e-mailem),
- omezení služeb na počet hlídaných obchodních partnerů.

Využívání takových služeb je však zpoplatněno, a to od desítek korun až po desítky tisíc korun měsíčně.

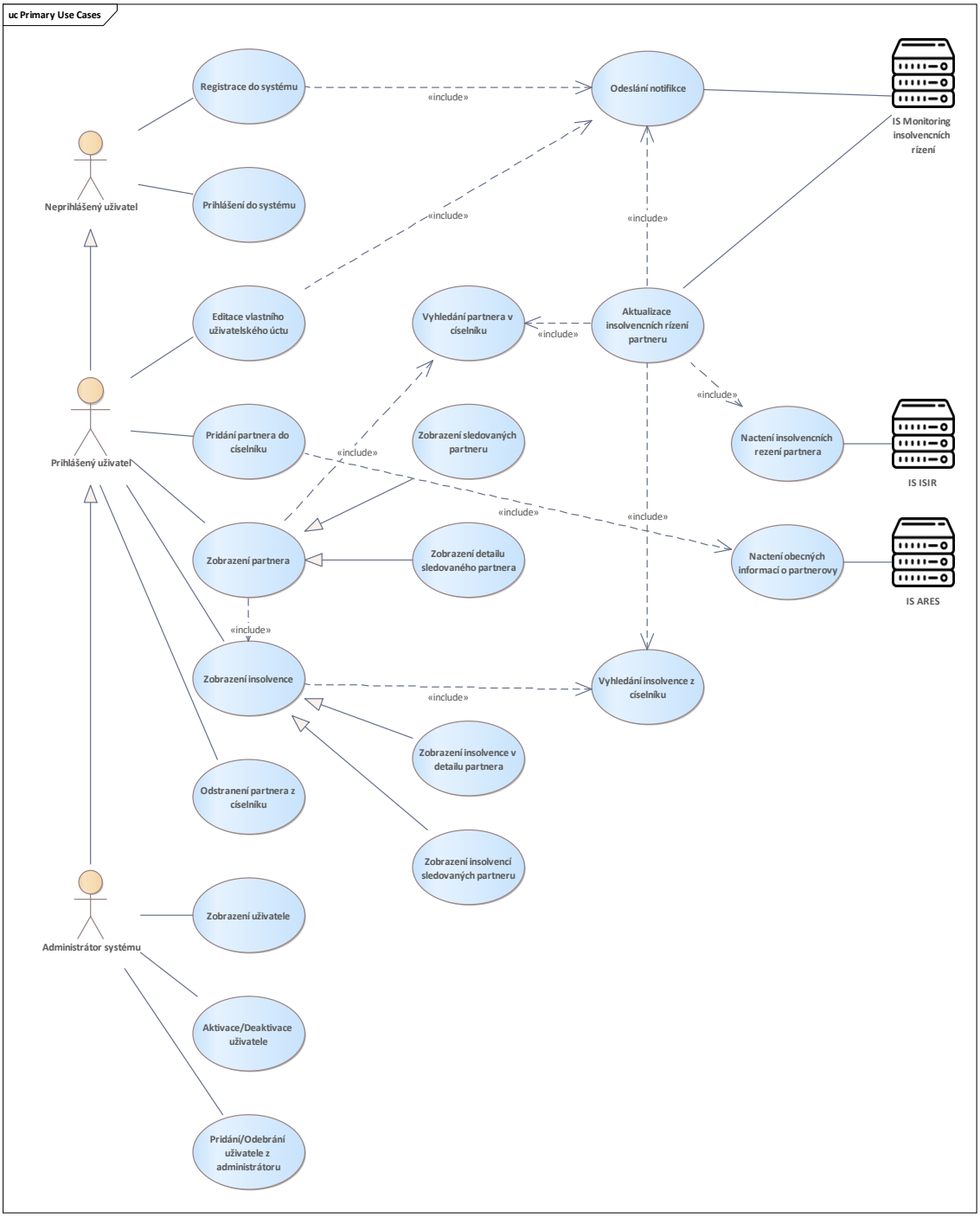
Mezi první čtyři společnosti poskytující podobné služby vyhledávač Google například řadí:

- společnost i4b s.r.o. na stránkách https://sledovani-insolvence.cz/
- společnost ADOL Monitor na stránkách https://www.adol.cz/sluzby/monitoring-insolvenci/
- společnost Credit Check, s.r.o. na stránkách https://www.creditcheck.cz/Default.aspx
- společnost Prowia system s.r.o. na stránkách https://www.isir.info

Každá ze služeb se snaží mírně diferencovat cenou a objemem služeb. U služby od společnosti Prowia system s.r.o. je jako benefit uvedena možnost poskytování API (Application Programming Interface) pro strojovou interakci se službou.

## 5.2 Případy užití vzorové aplikace

Pro vzorovou aplikaci „Monitoring insolvencí“ byl zpracován Use Case diagram (česky „diagram případů užití“), znázorňující případy užití aplikace. K modelování byl využit jazyk UML (Unified Modeling Language).

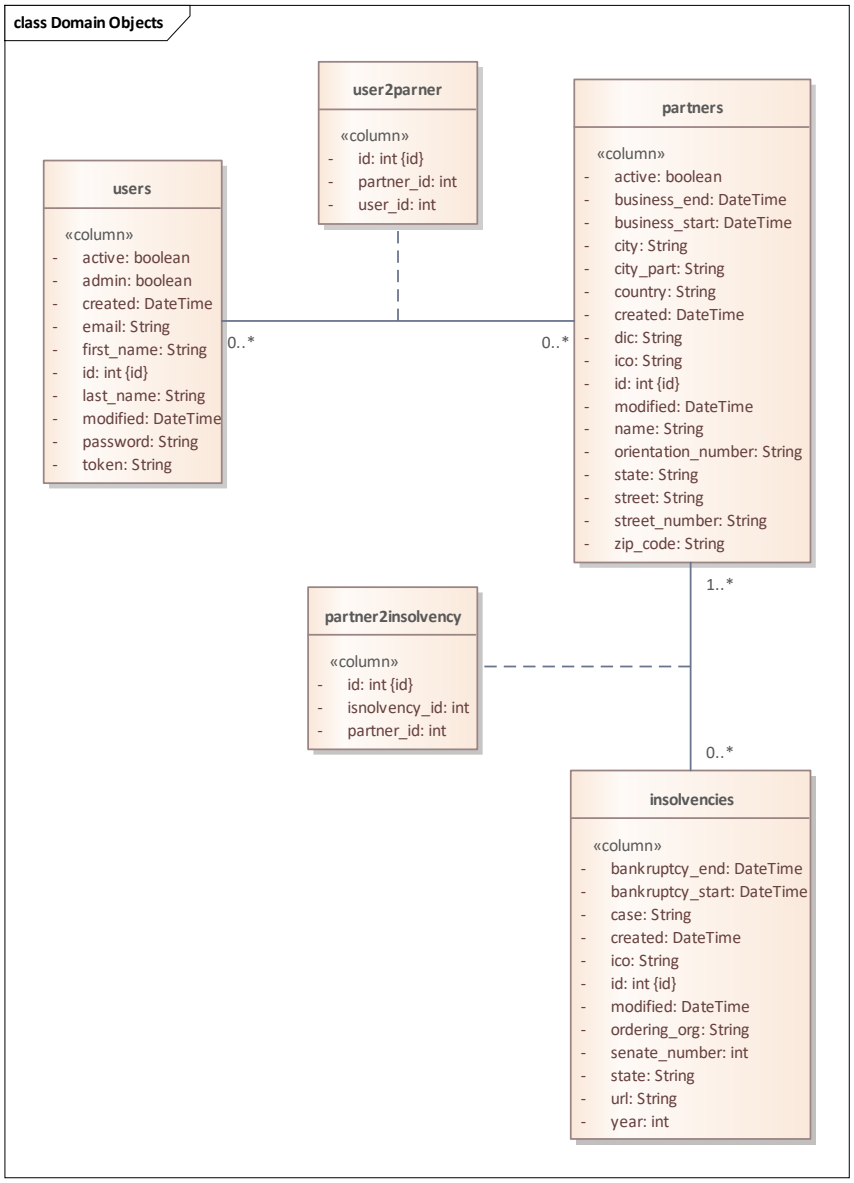



**Obrázek 2: Use Case diagram případů užití vzorové aplikace**
Zdroj: Autor

Uvedené případy užití vyjadřují funkcionality webové aplikace. Konkrétní případ užití je vždy propojen s určitou rolí, která danou funkcionalitu užívá. Systémová role je vyjádřena příslušným aktérem v modelu.

## 5.3 Návrh struktury databáze vzorové aplikace

Vzorová aplikace bude ukládat persistentní data do databáze. K tomuto účelu byl vytvořen návrh struktury databáze, který byl popsán v doménovém diagramu, modelovaném v jazyce UML.




**Obrázek 3: Doménový diagram návrhu struktury databáze vzorové aplikace**
Zdroj: Autor

## 5.4 Realizace

V následujících kapitolách je uveden podrobný postup realizace navržené aplikace.

### 5.4.1 Příprava prostředí

Před implementací projektu samotného bylo potřeba připravit vývojové prostředí na operačním systému tak, aby bylo možné aplikaci vyvíjet. Vývoj samotný byl prováděn na systému Fedora 34.

#### 5.4.1.1 Instalace Python 3

Instalace Python 3 se výrazně liší dle operačního systému, na kterém má prostředí běžet. Pro instalaci byl použit balíčkový systém DNF (Dandified YUM).

```
sudo dnf install -y python3 python3-pip
```

**Obrázek 4: Příklad instalace Python 3 a balíčkovacího systému pip**
Zdroj: Autor

Nainstalován byl Python 3 a balíčkovací systém pip.

#### 5.4.1.2 Vytvoření virtuálního prostředí

Ač virtuální prostředí není povinné, je vysoce doporučované. Díky oddělení defaultního prostředí od virtuálního bylo zajištěno, že v případě provozu více aplikací na jednom systému budou nainstalované balíčky stále ve správné verzi. Vytvoření virtuálního prostředí bylo provedeno pomocí volání.

```
python3 -m venv venv
```

**Obrázek 5: Vytvoření virtuálního prostředí**
Zdroj: Autor

Tím bylo vytvořeno v současné složce virtuální prostředí. Aktivace byla provedena voláním.

```
source ./venv/bin/activate
```

**Obrázek 6: Aktivace vytvořeného virtuálního prostředí**
Zdroj: Autor

#### 5.4.1.3 Instalace Python balíčků

Pro správný chod aplikace je potřeba doinstalovat balíčky, které nejsou distribuovány v základní instalaci Python 3. Seznam balíčků včetně potřebné verze

je v souboru requirements.txt. Instalace byla provedena hromadně pomocí balíčkovacího sytému pip.

```
pip install -r requirements.txt
```

**Obrázek 7: Instalace podpůrných balíčků**
Zdroj: Autor

#### 5.4.1.4 Instalace MySQL

Jako databáze byla zvolena MySQL. Pro instalaci byl použit balíčkovací systém DNF. Nejdříve však bylo nutné importovat aktuální repozitář MySQL 8.

```
sudo dnf install -y https://dev.mysql.com/get/mysql80-community-release-fc34-1.noarch.rpm
```

**Obrázek 8: Import aktuálního repozitáře MySQL 8**
Zdroj: Autor

Následně bylo možné provést instalaci aplikace mysql-comminity-server.

```
sudo dnf install -y mysql-community-server
```

**Obrázek 9: Instalace MySQL 8**
Zdroj: Autor

Nakonec bylo potřeba spustit službu mysqld.

```
sudo systemctl start mysqld
```

**Obrázek 10: Spuštění služby mysqld**
Zdroj: Autor

#### 5.4.1.5 Vytvoření databáze pro aplikaci v MySQL

Vzhledem k tomu, že aplikace vyžaduje databázi a uživatele pro ukládání dat, bylo nutné je vytvořit. K vytvoření byla použita posloupnost volání.

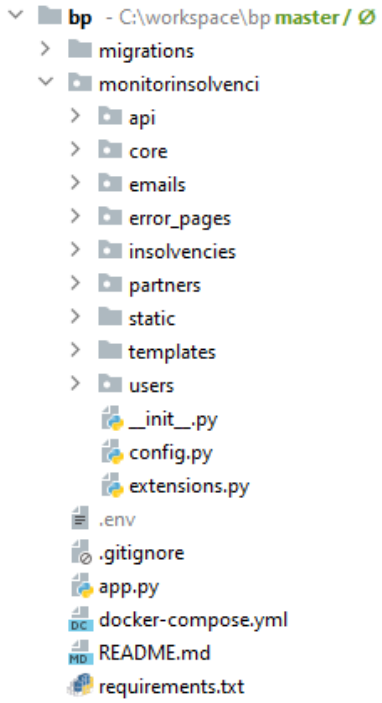
```
mysql -e "CREATE DATABASE monitorinsolvenci;"
mysql -e "CREATE USER monitorinsolvenci@localhost IDENTIFIED BY 'heslo';"
mysql -e "GRANT ALL PRIVILEGES ON monitorinsolvenci.* TO 'monitorinsolvenci'@'localhost';"
mysql -e "FLUSH PRIVILEGES;"
```

**Obrázek 11: Vytvoření databáze pro aplikaci**
Zdroj: Autor

### 5.4.2 Struktura projektu

Flask je velice flexibilní v možnostech uspořádání projektu. Autor se rozhodl pro členění projektu dle aplikací a jejich agend. V nejvyšším patře stromové struktury je pak jen aplikace samotná a migrace databáze. Vnitřní rozložení je v aplikaci

členěno na agendy, šablony, API, pomocné a inicializační scripty. Agendy mají model a pohled, případně formuláře a pomocné utilitární části. Šablony jsou seskupené pro celou aplikaci s dodatečným rozlišením pro šablony e-mailové a chybové. API je verzované a obsahuje všechny koncové body. Konečná struktura projektu je pak znázorněna na Obrázku 12.




**Obrázek 12: Finální struktura aplikace**
Zdroj: Autor

### 5.4.3 Příprava pro inicializaci aplikace

Inicializační scripty v aplikaci jsou rozděleny na __init__.py, config.py a extensions.py. Config.py obsahuje načtení nastavení aplikace, samotná data jsou však uložena v souboru .env. Tím je docíleno oddělení citlivých informací od programové části aplikace. Extensions.py zajišťuje inicializaci všech rozšíření aplikace. Díky oddělení této části do samostatného scriptu je možné importovat jen konkrétní rozšíření do dalších částí projektu bez nutnosti načtení nepotřebných rozšíření. Obě předchozí části jsou pak spojeny a uceleny do plnohodnotné aplikace v __init__.py. Zde se nachází jediná metoda create_app, která zajišťuje přípravu celé Flask aplikace. Config.py je načten jako objekt a nastavení jsou zapsána do aplikace. Všechna rozšíření z extensions.py jsou přidána do aplikace,

případně propojena mezi sebou. Nakonec jsou načteny všechny moduly aplikace skrze Blueprint (česky „Plán“).

### 5.4.4 Modely

Modely zajišťují v projektu objektovou abstrakci databázových tabulek skrze ORM. V agendách byly vytvořeny samostatné modely, které jsou uloženy v souboru models.py. V něm byly definovány třídy nebo pomocné tabulky dle návrhu struktury databáze.

#### 5.4.4.1 Model “Uživatelé”

Uživatelé mají v agendě users vytvořenou třídu User. Každý sloupec je následně definován v proměnné. Sloupce jako “id”, “email” a další mají navíc definovány specifické vlastnosti. Příkladem může být sloupec “active”, který určuje, zda je uživatel aktivní a může využívat služeb aplikace. Tento sloupec má pak definovány speciální vlastnosti, nemožnost být Null a výchozí hodnotu na True.

```
active = db.Column(db.Boolean, nullable=False, default=True)
```

**Obrázek 13: Speciální vlastnosti sloupců**
Zdroj: Autor

Speciálním sloupcem je sloupec “partners”. Ten je dynamicky generovaný v rámci ORM, a není tedy přímo uložen v tabulce users. Jedná se o takzvaný vztahový sloupec, jenž zobrazuje, kteří partneři jsou spojeni s uživatelem. V tomto případě je zde popsán vztah M:N mezi tabulkou users a partners. Jelikož se jedná o spojení typu „many to many” (česky „mnoho k mnoha“), bylo nutné definovat spojovou tabulku, kde se budou spojení nacházet. Ta byla definována v modelu uživatelů. Jedná se o tabulku user2partner.

```
partners = db.relationship('Partner', secondary=user2partner,
                           backref=db.backref('users'), lazy='dynamic')
```

**Obrázek 14: Vztah tabulek M:N**
Zdroj: Autor

Díky pokročilým možnostem ORM je možné přes parametr backref doplnit i do tabulky partners speciální dynamický sloupec „users“, který u partnera zobrazí všechny uživatele, kteří s ním mají spojení.

Třídě User bylo nutné kromě inicializace a utilitárních metod přidat i speciální metody odvíjející se od nutnosti autentifikace. Jako první byla vytvořena metoda password_check, jež porovnává předané heslo s heslem uloženým v databázi. Jelikož je heslo samozřejmě uloženo v hash variantě, je nutné převést předané heslo stejnou hashovací funkcí. Na stejném principu byla implementována metoda jwt_check, která místo hesla porovnává JWT token.

#### 5.4.4.2 Model ”Partneři”

Model pro partnery byl vytvořen velice podobným způsobem jako model uživatelů. Třída Partner abstrahuje tabulku partners v databázi. V tabulce byly vytvořeny dva vztahové sloupce. První je převzatý z třídy User a druhý je nový M:N sloupec s názvem „insolvencies“. Ten zobrazuje spojení mezi partnerem a jeho insolvencemi. Jako spojová tabulka byla použita tabulka partner2insolvency. Tento sloupec je pak zpětně propagován i v tabulce insolvencies pod názvem „partner“. V rámci třídy je pak i speciální metoda to_dict, která převádí proměnné do formátu slovníku. V rámci partnerů bylo však nutné před převodem provést úpravy v datech, aby byly lépe zpracovatelné v dalších částech aplikace. Úpravou prošly všechny proměnné s datem, kde bylo nutné převést datum na český formát dd.MM.yyyy.

```
'business_start': self.business_start.strftime("%d.%m.%Y")
```

**Obrázek 15: Úprava proměnné na český formát dd.MM.yyyy**
Zdroj: Autor

Převodem tak musela projít adresa partnera. Jelikož adresy z ARESu nejsou konzistentní, bylo nutné adresy upravit do formátu použitelného například při odesílání zásilky. Výsledkem těchto změn byla konzistentní data pro následné použití.

#### 5.4.4.3 Model “Insolvence"

Insolvence se nacházejí pod třídou Insolvency. Ta byla konstruována stejně jako předchozí třídy. Z partnerů získává dynamický sloupec partners, jinak je její struktura standardní. Kvůli častým aktualizacím insolvencí byla do třídy přidána

metoda update_insolvency. Metoda zpracovává předaná data k insolvenci a aktualizuje současný stav v databázi.

### 5.4.5 Formuláře

Pomocí rozšíření Flask_WTF byly v aplikaci vytvořeny formuláře, které budou následně podávány uživatelům skrze šablony. Formuláře byly využity pro interakci s uživatelem v agendě uživatelé a partneři. Definice byly vždy napsány v souboru forms.py jako třídy a importovány do pohledů. U každého formuláře byl pak definován validátor předaných dat tak, aby nebylo možné do polí plnit nekorektní data. Validace pak probíhá na straně serveru, aby ji nebylo možné vynechat úpravou klientské části.

```
class LoginForm(FlaskForm):
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Heslo', validators=[DataRequired()])
    submit = SubmitField('Přihlásit')
```

**Obrázek 16: Příklad definice formuláře pro přihlášení**
Zdroj: Autor

### 5.4.6 Šablony

Generování HTML dokumentů pro uživatele bylo řešeno pomocí šablonovacího jazyku Jinja, který je distribuován společně s frameworkem Flask.

#### 5.4.6.1 Šablona base.html

Pro projekt byla vytvořena jedna šablona base.html, z níž následně dědily obsah všechny další šablony. Tato šablona obsahuje všechny základní tagy HTML dokumentu, import frontend frameworku Bootstrap, vrchní navigační lištu, řešení Alerts (česky, „výstrahy“) a patičku. Jelikož se jedná o šablonu použitou ve všech obsahových stránkách, bylo nutné, aby řešila uživatelský kontext pro navigační lištu. To bylo vyřešeno if deklaracemi tak, aby uživatel získal jen relevantní navigaci. Části, které nevyhovují podmínkám, tak nejsou vůbec uživateli zaslány.

```
<ul class="navbar-nav bd-navbar-nav pt-2 py-md-0">
    <li class="nav-item col-6 col-md-auto">
        <a aria-current="page" class="nav-link" href="{{ url_for('core.index') }}">Domů</a>
    </li>
    {% if current_user.is_authenticated %}
        <li class="nav-item col-6 col-md-auto">
            <a class="nav-link" href="{{ url_for('partners.user_partners') }}">Partneři</a>
        </li>
        <li class="nav-item col-6 col-md-auto">
            <a class="nav-link" href="{{ url_for('insolvencies.user_insolvencies') }}">Insolvence</a>
        </li>
        {% if current_user.admin %}
            <li class="nav-item col-6 col-md-auto">
                <a class="nav-link" href="{{ url_for('users.administration') }}">Administrace uživatelů</a>
            </li>
        {% endif %}
    {% endif %}
    <li class="nav-item col-6 col-md-auto">
        <a class="nav-link" href="{{ url_for('core.about') }}">O nás</a>
    </li>
</ul>
```

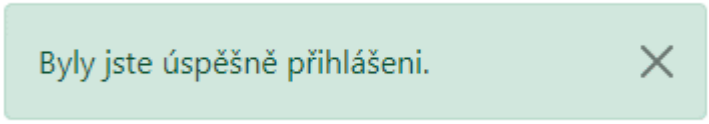
**Obrázek 17: Řešení navigační lišty v base.html**
Zdroj: Autor

Tak, aby bylo možné uživateli zobrazovat grafická upozornění přímo do stejné stránky, na které se nachází, bylo nutné implementovat výstrahy, jež se budou uživateli ukazovat. K tomu sloužil cyklus zobrazující vrácené výstrahy včetně jejich kategorií tak, aby bylo možné rozlišit, jak má výsledné upozornění vypadat.

```
{% for category, message in get_flashed_messages(with_categories=true) %}
    <div
            class="alert alert-{{ category }} alert-dismissible fade show"
            role="alert"
    >
        {{ message }}
        <button
                aria-label="Close"
                class="btn-close"
                data-bs-dismiss="alert"
                type="button"
        ></button>
    </div>
{% endfor %}
```

**Obrázek 18: Příklad implementace cyklu pro zobrazení výstrah**
Zdroj: Autor

V případě, kdy aplikace vyvolá výstrahu, uživateli byl zobrazen definovaný banner.




**Obrázek 19: Příklad zobrazení vyvolané výstrahy**
Zdroj: Autor

Pomocí deklarace bloků je možné do šablony přidávat další CSS, scripty a obsah samotný. Díky tomuto modulárnímu přístupu se ostatní šablony plně soustředí na zobrazovaný obsah.

```
{% block content %}
{% endblock %}
```

**Obrázek 20: Definice bloku content**
Zdroj: Autor

Dědičnost u ostatních šablon byla provedena přes deklaraci extends a doplnění bloků.

```
{% extends "base.html" %}
{% block csss %}
    # Přidané CSS
{% endblock %}
{% block content %}
    # Obsah stránky
{% endblock %}
{% block scripts %}
    # Přidáné scripty
{% endblock %}
```

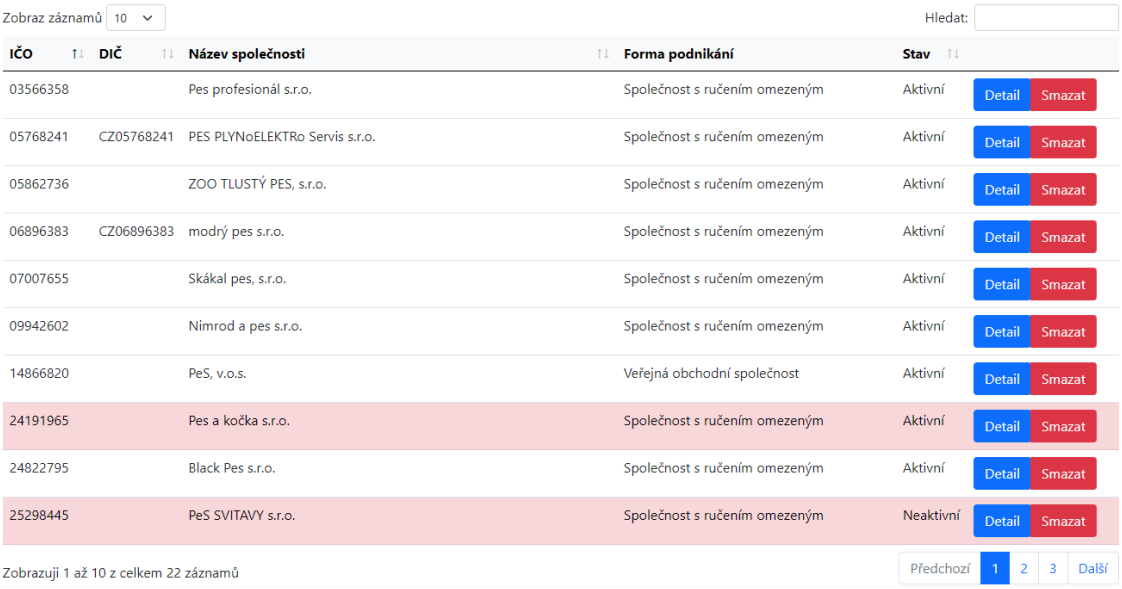
**Obrázek 21: Příklad rozšíření šablony base.html a využití bloků**
Zdroj: Autor

### 5.4.6.2 DataTables v šablonách

Speciálním případem byly stránky, kde byla využita knihovna DataTables. Zde bylo nutné doplnit CSS a scripty pro správné fungování komponent. To bylo zařízeno díky rozšíření bloků csss a scripts z šablony base.html. Samotné fungování komponenty DataTables je řízeno deklarací v JavaScriptu. Jelikož byl kladen důraz na rychlost fungování řešení, autor se pokusil o řešení dat takzvaně server-side (česky „na straně serveru“) pomocí AJAX (Asynchronous JavaScript and XML). Data

tabulek tak nejsou součástí odesílaného dokumentu klientovi, ale jsou vyžadovány klientem postupně, jak s tabulkou uživatel pracuje. To šetří zdroje na straně klienta i serveru, hlavně v případě rozsáhlých datových kolekcí. Díky implementaci knihovny získává uživatel rozšířené možnosti nad tabulkou:

- možnost dynamicky hledat skrze výsledky,
- řadit výsledky dle sloupců,
- určovat, kolik záznamů je zobrazeno na stránku,
- listovat stránkami výsledků.




**Obrázek 22: Příklad výsledného vzhledu tabulky při použití DataTables**
Zdroj: Autor

### 5.4.6.3 Šablony s formuláři

Pro formuláře vytvořené s využitím balíčku Flask-WTF použít v šablonách, bylo nutné dodržet kroky popsané v dokumentaci k balíčku. Ta vyžaduje, aby součástí každého formuláře byl výraz pro hidden tag (česky „skrytá značka“) a výrazy pro ostatní pole nebo tlačítka formuláře. Tím je docíleno zabezpečení formuláře proti nechtěné uživatelské manipulaci.

```
<form method="POST">
    {{ form.hidden_tag() }}
    <div class="form-floating">
        {{ form.email(type="email", class="form-control", placeholder="name@example.com") }}
        {{ form.email.label(class="form-label") }}
    </div>
    <div class="form-floating mb-3">
        {{ form.password(type="password", class="form-control", placeholder="Heslo") }}
        {{ form.password.label(class="form-label") }}
    </div>
    {{ form.submit(class="btn btn-primary btn-lg") }}
</form>
```

**Obrázek 23: Příklad užití Flask-WTF v šabloně přihlášení**
Zdroj: Autor

### 5.4.7 Pohledy

K řešení routingu a obsluhy požadavků jsou v projektu použity pohledy. K jejich registraci v aplikaci byla použita metoda Blueprint, která umožňuje oddělovat agendy a speciální obslužné části pomocí URL prefixů. Pohledy řeší všechny požadavky klienta jedním ze dvou možných řešení:

- odesláním HTML dokumentu pomocí vykreslení šablony funkcí render_template,
- přesměrováním klienta na jinou URL pomocí funkcí redirect a url_for.

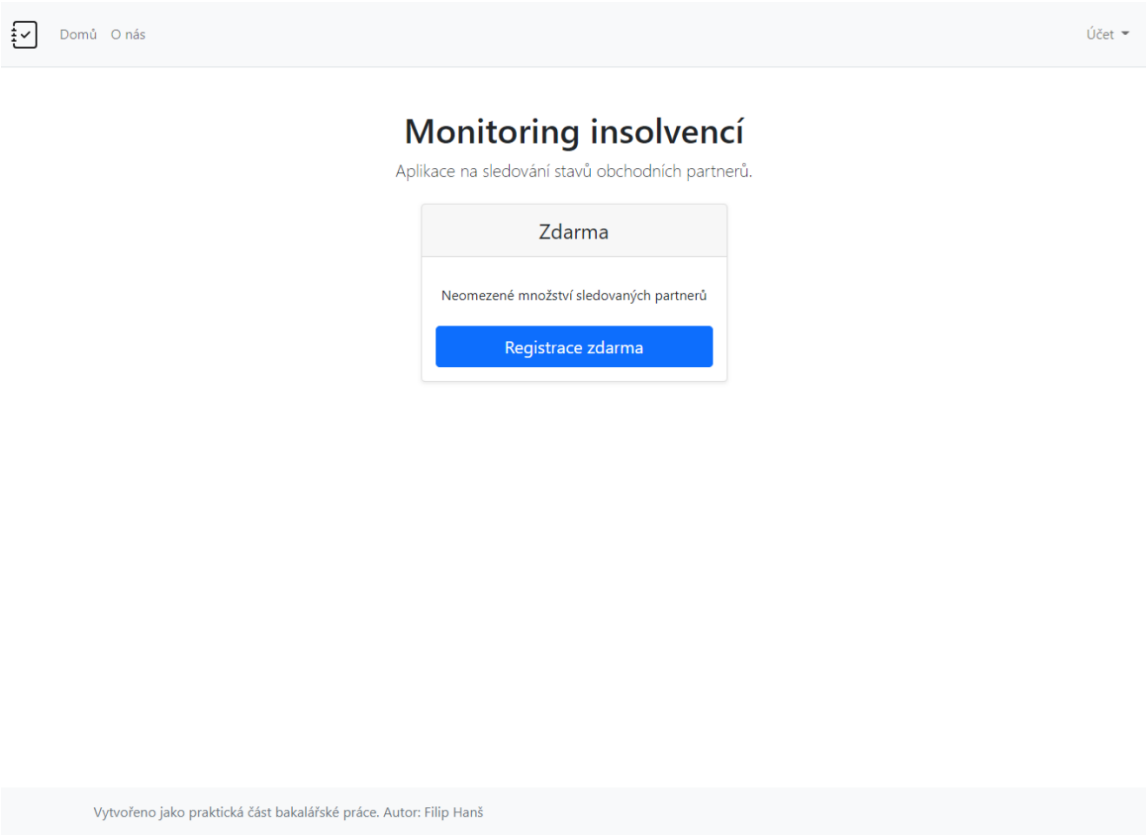
#### 5.4.7.1 Základní pohled

Nejzákladnější stránky byly seskupeny do takzvaného core (česky „jádro“) pohledu. Registrace blueprintu v projektu byla provedena vytvořením jeho instance, v pohledu.

```
users = Blueprint('users', __name__)
```

**Obrázek 24: Registrace blueprintu pohledu**
Zdroj: Autor

Směrování skrze blueprint core pak obstarává stránky index a about. Každý routing měl pak řešenu relativní cestu URL i metody, na které odpovídá. Jelikož zpracování požadavků nevyžadovalo žádnou speciální logiku, jedná se o příklad primitivních vrácení generované šablony.




**Obrázek 25: Příklad výsledné stránky index.html z pohledu core**
Zdroj: Autor

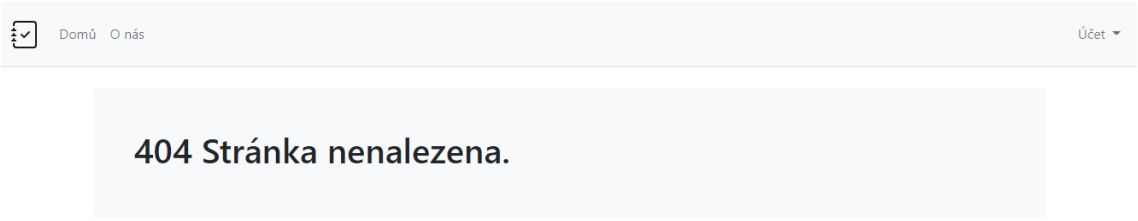
Pro nepřihlášené uživatele jsou pak dostupné stránky:

- úvodní stránka,
- stránka o projektu.

#### 5.4.7.2 Chybový pohled

Kvůli obsluze HTML chybových kódů byl vytvořen speciální pohled error_pages. Ten zajistil, že i v případě, kdy aplikace z nějakého důvodu skončí řešení požadavku chybou, uživateli se nezobrazí nestrukturovaná chybová hláška, ale vzhledově celistvá stránka s uživatelsky přívětivou zprávou.

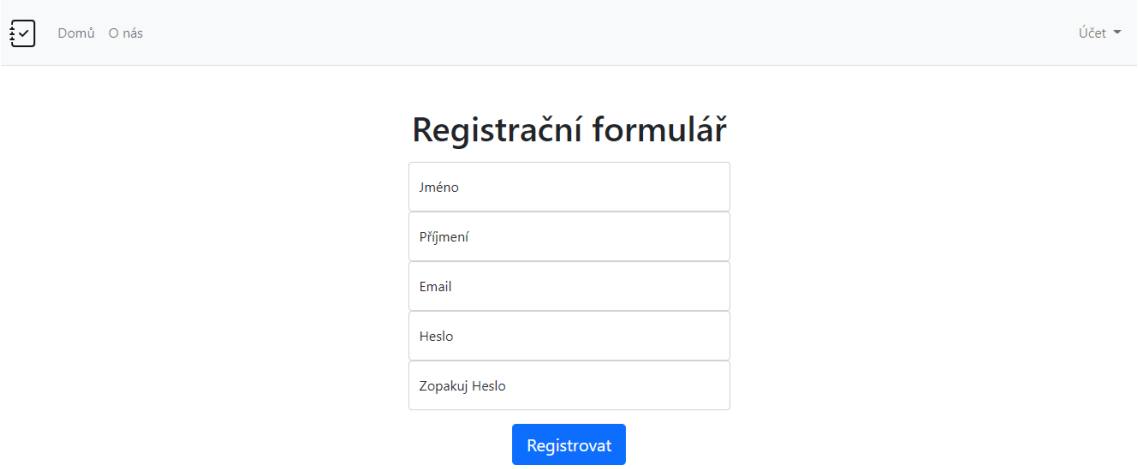
Zpracovány byly tři hlavní chyby s kódovým označením 403, 404 a 500. Jelikož bylo od začátku plánováno implementovat API, generovaná odpověď se liší podle přijímaného způsobu odpovědi klientem. Pokud je klientem stroj, je odpovědí JSON. Pokud je klientem webový prohlížeč, je odpovědí klasická webová stránka.




**Obrázek 26: Příklad výsledné chybové stránky**
Zdroj: Autor

### 5.4.7.3 Pohled "Uživatelé"

Jelikož byla aplikace navržena s částí funkcionalit dostupných jen registrovaným a přihlášeným uživatelům, bylo potřeba vytvořit funkce, které obstarají všechny požadavky na management uživatelských účtů. K tomuto účelu byl vytvořen pohled Uživatelé. Pro řešení uživatelských relací bylo použito rozšíření Flask-Login.




**Obrázek 27: Registrace nového uživatele**
Zdroj: Autor

Pro zajištění, aby nepřihlášení uživatelé nemohli vstoupit do pro ně nepřístupných částí, byl použit dekorátor login_required z rozšíření Flask-Login.

```
@users.route('/logout', methods=['GET', 'POST'])
@login_required
def logout():
    logout_user()
    flash('Byly jste úspěšně odhlášeni.', 'success')
    return redirect(url_for('core.index'))
```

**Obrázek 28: Příklad užití dekorátoru login_required**
Zdroj: Autor

Tím jsou obaleny všechny směrovače, které vyžadují přihlášeného uživatele. Speciální částí je pak administrace uživatelů skrze aplikaci. K té bylo potřeba oddělit standardní uživatele od administrátorů. To bylo zařízeno příznakem admin ve třídě User. Následně byla administrátorům vytvořena šablona pro zobrazení stránky s přehledem uživatelů.
Nepřihlášení uživatelé mohou v aplikaci použít tyto koncové body:

- registrace nového uživatele,
- přihlášení stávajícího uživatele.

Přihlášení uživatelé mohou navíc zobrazit stránky:

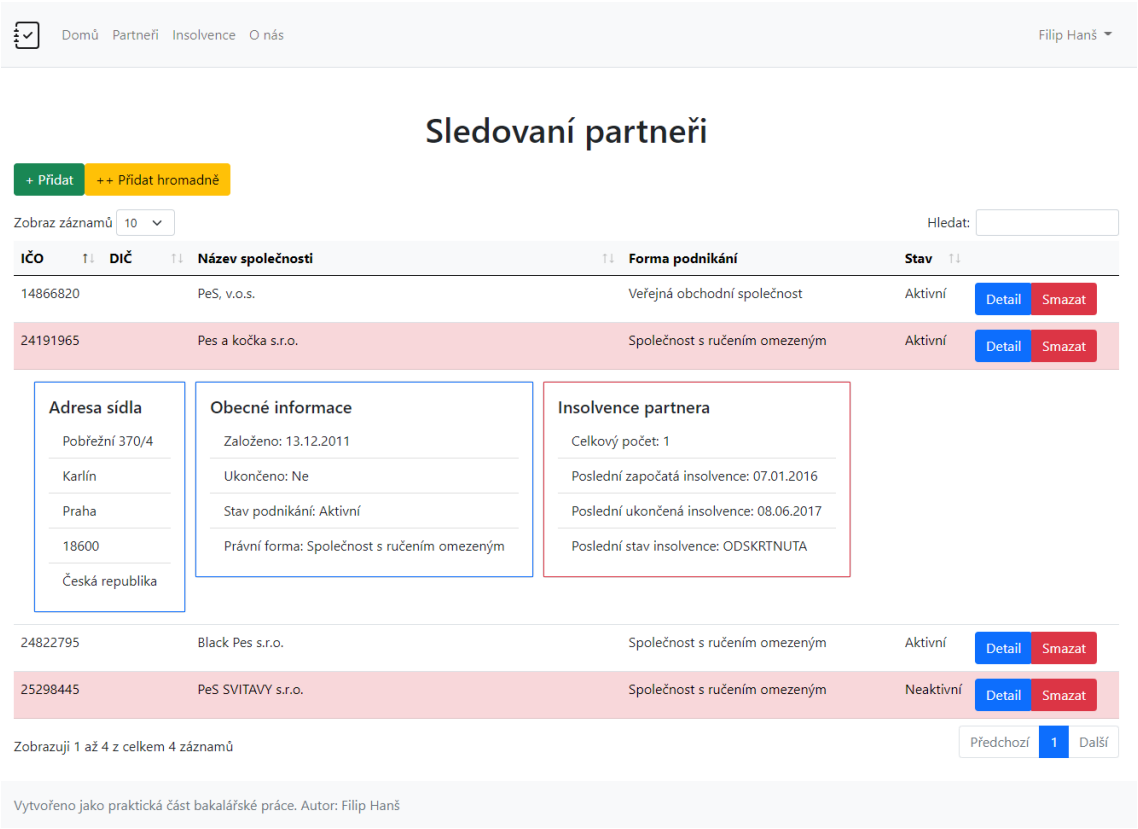
- odhlášení uživatele,
- editace údajů uživatelského účtu,
- vygenerování API tokenu.

Administrátoři mají přístup ke všem stránkám jako přihlášení a nepřihlášení uživatelé, a navíc mohou zobrazit stránky:

- hromadný pohled na všechny uživatele,
- přidání a odebrání uživatele z administrátorů,
- aktivace a deaktivace uživatelského účtu.

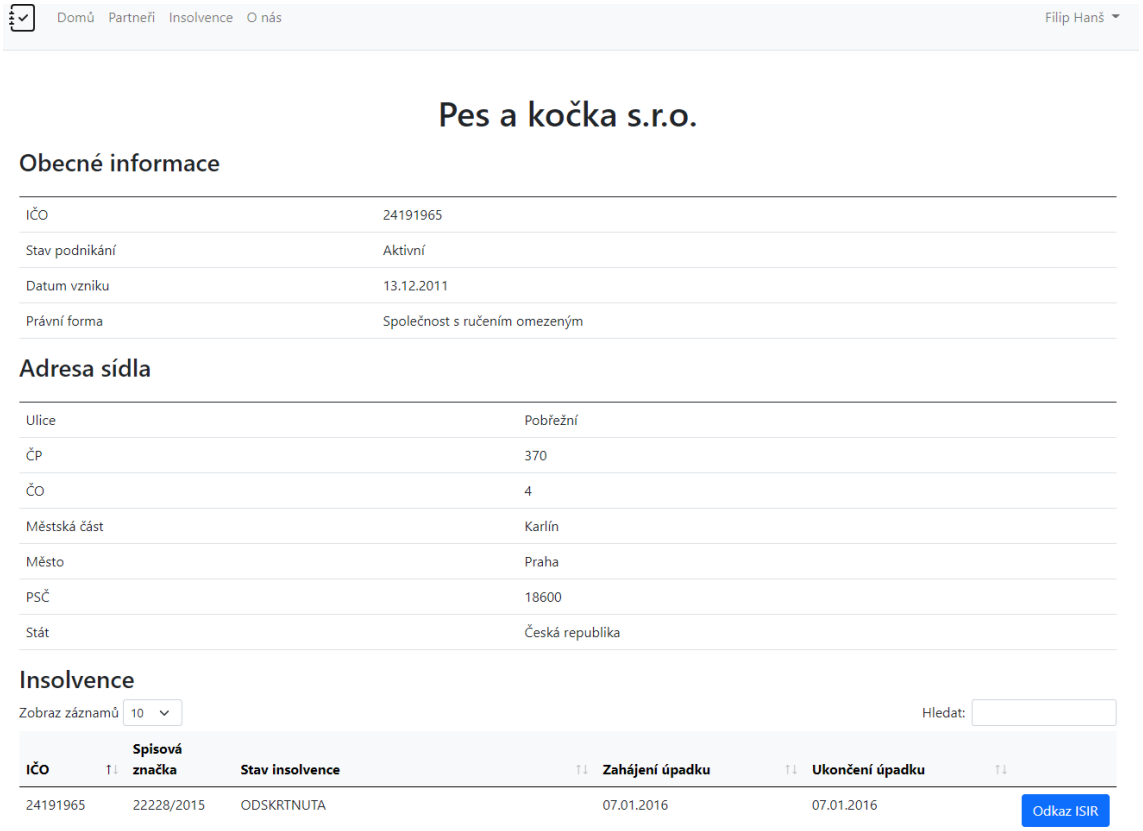
#### 5.4.7.4 Pohled "Partneři"

Možnost zobrazení a sledování stavu partnerů je klíčová funkce celé aplikace. K tomuto účelu byly vytvořeny dvě části pohledů. První část je svázána s hromadným pohledem na všechny partnery. Ten je přizpůsoben každému uživateli zvlášť tak, aby ukazoval relevantní data. K zobrazení dat je použita tabulka s rozšířením DataTables. Každý partner je pak zobrazen na jednom řádku tabulky se zobrazenými základními informacemi. Pokud má partner alespoň jednu vedenou insolvenci, je záznam zobrazen s červenou barvou. Stručný detail partnera je možné zobrazit pomocí kliknutí na řádek záznamu.




**Obrázek 29: Příklad zobrazení partnerů s insolvenčním řízením a zobrazeným detailem**
Zdroj: Autor

Z hromadného pohledu pak lze využít možnosti přidání jednoho partnera, hromadného importu partnerů ze souboru, smazání partnera a zobrazení plného detailu partnera. Plný detail partnera se skládá ze všech dostupných údajů uložených v databázi. V prvních dvou částech jsou zobrazeny obecné informace a celá adresa sídla. Ve třetí části jsou pak zobrazena všechna insolvenční řízení vedená s dodavatelem.




**Obrázek 30: Příklad detailu partnera**
Zdroj: Autor

Pro přihlášené uživatele byly vytvořeny tyto koncové body:

- hromadné zobrazení všech partnerů uživatele,
- založení jednoho partnera,
- hromadný import partnerů ze souboru,
- detail konkrétního partnera,
- smazání konkrétního partnera.

### 5.4.7.5 Pohled "Insolvence"

Pro uživatele bylo nakonec nutné vytvořit souhrnný pohled na všechna insolvenční řízení sledovaných partnerů. Pohled se chová podobně jako hromadný pohled na partnery. Jelikož byly insolvence získávány automaticky pro všechny sledované partnery, je v pohledu jediné funkční tlačítko. Tlačítko odkazuje na detail konkrétního insolvenčního řízení na stránkách https://isir.justice.cz/.


**Obrázek 31: Příklad hromadného zobrazení insolvenčních řízení sledovaných partnerů**

Přihlášeným uživatelům byl připraven koncový bod hromadné zobrazení všech insolvencí sledovaných partnerů.

### 5.4.8 Napojení státních systémů

Aby bylo možné čerpat data ze státních systémů, bylo nutné vytvořit implementace poskytovaných služeb.

#### 5.4.8.1 ares.py

Služby systému ARES jsou dostupné ve více formách. Pro jednodušší implementaci se autor rozhodl získávat data přes HTTP GET koncový bod na basic dotaz. K tomu byla použita knihovna Requests a xmltodict. Jelikož není nutné udržovat se službou dlouhodobé spojení, není integrace řešena přes plnohodnotnou třídu, ale jen jako volatelná metoda get_ares_data, přijímající jako jediný parametr IČO. Po zavolání služby je obdržena odpověď ve formátu XML. Aby bylo možné data zpracovat, je nutné je převést do struktur python, tedy slovníků a polí. To je v projektu provedeno pomocí parseru z knihovny xmltodict. Poslední úprava dat je převod zkratek služby ARES na čitelné názvy proměnných. Výsledkem je pak ucelený výstup s potřebnými údaji o partnerovi. Ve scriptu je také obsažena pomocná metoda fill_partner_with_ares, která vytvoří novou instanci partnera v databázi z dat, která poskytuje metoda get_ares_data.

#### 5.4.8.2 isir2.py

ISIR je možné konzumovat jen pomocí SOAP WSDL. Pro abstrakci webových služeb autor použil knihovnu Zeep. S API je potřeba udržovat otevřené spojení, proto bylo

rozhraní implementováno jako třída Isir, která zajišťuje inicializaci komunikace a udržuje její kontinuitu. Samotná třída po inicializaci má jedinou metodu, totiž get_ico_insolvencies, která přijímá IČO. Jelikož API vrací specifické odpovědi definované ve WSDL, je nutné převést odpověď v XML speciální metodou serialize_object. Ta převede odpověď do struktur python pomocí specifikace odpovědi z API. Díky tomu, že partner může být ve více insolvenčních řízeních, je výsledkem metody pole insolvenčních řízení.

Dále je ve scriptu přítomna i pomocná metoda fill_insolvency_with_isir, která se chová stejně jako metoda fill_partner_with_ares, jen s rozdílem, že tvoří instanci insolvence.

### 5.4.9 Pravidelné aktualizace

I když jsou informace poskytnuté z konzumovaného státního systémů ISIR v okamžiku volání aktuální, mohou se v průběhu času měnit. Kvůli tomu bylo nutné implementovat možnost pravidelných aktualizací informací.

Tak, aby bylo možné využít všech zdrojů aplikace, primárně databázových modelů, rozhodl se autor pro použití volání CLI (Command Line Interface, česky „příkazová řádka“) metod aplikace přes framework Flask. Ty jsou řešeny jako speciální typ pohledů, kde je místo routování metoda registrována jako cli.command. Takto byla v projektu vytvořena metoda update_insolvencies. Ta aktualizuje příslušné záznamy v databázi pomocí porovnání současných dat a aktuálně získaných dat ze systému. Samotné volání metody je pak prováděno v příkazové řádce pomocí příkazu flask [ název blueprintu ] [ název cli metody ]. Pokud název cli metody obsahuje „_“, je v příkazové řádce nahrazen „-“.

```
flask insolvencies update-insolvencies
```

**Obrázek 32: Příkaz v CLI pro aktualizaci dat insolvencí**
Zdroj: Autor

V rámci aktualizace údajů je pak vytvořen seznam změn insolvencí. Ten je pak odeslán e-mailem všem uživatelům, kteří dané partnery sledují. K odeslání je použit balíček Flask-Mail a nastavený mail server.

Pro automatizaci aktualizací použil autor softwarového démona Cron, dostupného na většině linuxových distribucí. Ten je sice jednodušší a starší než

moderní plánovače, jako je například Celery, ale pro potřeby projektu je naprosto dostačující. Cron je potřeba nastavit pomocí příkazu crontab -e.

```
0 1 * * * cd /home/Fildah/bp && /home/Fildah/bp/venv/bin/flask insolvencies update-insolvencies
```

**Obrázek 33: Příklad nastavení cron**
Zdroj: Autor

Nastavení zajistí, že cron spustí aktualizace každý den v 1:00.

### 5.4.10 REST API

Aplikace byla vyvinuta tak, aby poskytovala možnost strojového zpracování dat. Uživatelům tak dává možnost implementovat služby ve svých interních systémech a využít tak data i jinde než jen v aplikaci samotné. Jako architektura rozhraní byla autorem zvolena varianta REST (Representational state transfer). Pro zlepšení budoucí zpětné kompatibility je API verzováno. Docílilo se tím možnosti upgradu poskytovaných služeb bez nutnosti okamžitého přepsání integrací na straně uživatelů.

Koncové body jsou následně chráněny před nedovoleným použitím pomocí ověření vůči tokenu. Aby uživatel token získal, musí si ho vyžádat v uživatelské části aplikace. Token je poskytován ve formátu JWT (JSON Web Token). V něm je zakódován ID uživatele pro dodatečné ověření správnosti tokenu. Token musí být předán při každém volání API v hlavičce požadavku v rámci klíče Authorization.

Každá část aplikace má pak implementovány funkce, ke kterým lze přistupovat. Ty jsou řešeny pomocí HTTP metod GET, POST nebo DELETE. Odlišení funkcí je zařízeno díky specifikaci URL pro každou z nich a omezení jen na korektní metodu volání.

```python
@api.route('/partners/<int:partner_id>', methods=['DELETE'])
@api_auth.login_required
def api_partner_edit(partner_id):
    partner = Partner.query.filter_by(id=partner_id).first()
    if api_auth.current_user() in partner.users:
        if request.method == 'DELETE':
            partner.remove_user(api_auth.current_user())
            if len(partner.users) == 0:
                partner.toggle_active()
            return jsonify({'data': 'Partner byl smazán!'})
```

**Obrázek 34: Příklad implementace funkce smazání partnera pomocí REST API**
Zdroj: Autor

Každá funkce pak vrací klientovi jako odpověď JSON (JavaScript Object Notation) s příslušnými daty nebo zprávou o výsledku volání.

Speciální koncové body byly vytvořeny pro tabulky využívající DataTables. Tyto koncové body jsou pak ověřovány nikoliv tokenem, ale uživatelskou relací. Jelikož jsou dané koncové body volány výhradně skrze AJAX v rámci standardní uživatelské relace, není autorizace přes token vhodná.

Uživatelům byla dostupná tato volání REST API:

- získání informací o svém účtu,
- smazání účtu,
- získání seznamu sledovaných partnerů,
- přidání partnera do sledování,
- odebrání partnera ze sledování,
- získání seznamu insolvencí sledovaných partnerů,
- získání detailu insolvence.

### 5.4.11 Spuštění aplikace

Aplikace monitoring insolvencí je řešena jako python balíček, je tedy připravena pro import do WSGI kompatibilního webového serveru, případně lze aplikaci spustit v interním vývojovém webovém serveru, který je integrovaný přímo ve frameworku Flask. Před spuštěním je nutné doplnit nastavení souboru example.env a přejmenovat ho na .env.

#### 5.4.11.1 Spuštění vývojového prostředí

```
SQLALCHEMY_DATABASE_URI = mysql://uzivatel:heslo@host/databaze
SQLALCHEMY_TRACK_MODIFICATIONS = False
SECRET_KEY = tajneheslo
MAIL_SERVER = mail.server.com
MAIL_PORT = 465
MAIL_USE_SSL = True
MAIL_DEFAULT_SENDER = mail@mail.server.com
MAIL_USERNAME = uzivatel
MAIL_PASSWORD = heslo
```

**Obrázek 35: Příklad nastavení souboru .env**
Zdroj: Autor

Následně bylo potřeba provést takzvané nasazení aplikace, které vytvořilo strukturu databáze a vygenerovalo hlavní administrátorský účet.

```
flask core deploy
```

**Obrázek 36: Vytvoření interních struktur aplikace v databázi pomocí funkce nasazení**
Zdroj: Autor

Spuštění vývojového prostředí bylo provedeno skrze script app.py.

```
python app.py
```

**Obrázek 37: Spuštění vývojového prostředí**
Zdroj: Autor

#### 5.4.11.2 Spuštění produkčního prostředí

Finální spuštění produkčního prostředí bylo provedeno stejně jako spuštění vývojového prostředí, jen s rozdílem nastavení a platformy. Autor si pro produkční prostředí vybral platformu pythonanywhere.com. Ta se specializuje na programovací jazyk Python 3 a jeho běh v cloudu. Na platformě byl již nainstalován Python 3, webový server a databázový server. Stačilo tedy jen nakopírovat soubory aplikace, nainstalovat Python balíčky a nastavit WSGI server.

```
import sys

# add your project directory to the sys.path
project_home = '/home/Fildah/bp'
if project_home not in sys.path:
    sys.path = [project_home] + sys.path

# import flask app but need to call it "application" for WSGI to work
# noqa
from monitorinsolvenci import create_app
application = create_app()
```

**Obrázek 38: Příklad nastavení WSGI serveru ve službě pythonanywhere.com**
Zdroj: Autor

Aplikace byla v době psaní této bakalářské práce dostupná pro použití široké veřejnosti na stránkách https://fildah.pythonanywhere.com.

# 6 Shrnutí výsledků

Na základě informací získaných v rámci teoretického základu bakalářské práce byla navržena vzorová aplikace „Monitoring insolvencí“ psaná v jazyce Python 3. Celá aplikace byla napsána ve vývojového prostředí PyCharm, které mimo jiné napomáhalo v rychlosti vývoje a korekci chyb v programu.

Aplikace byla kompletně vyvinuta s využitím frameworku Flask. Ten řešil technologickou obsluhu požadavků klientů. Samotná struktura aplikace pak byla uzpůsobena návrhovému vzoru MTV. S tím souviselo i rozdělení aplikace podle agend v rámci interních balíčků.

Pro všechny interakce s databází byl použit rozšiřující balíček Flask-SQLAlchemy. Ten za pomocí techniky ORM abstrahoval volání nad databází do programové podoby. Následné interakce s databází byly řešeny programově přímo v aplikaci. Struktura databáze samotné vycházela z návrhu v UML a byla převedena do modelů v aplikaci. Pro ukládání dat byla využita databáze MySQL.

Struktura generovaných HTML dokumentů byla tvořena pomocí šablonovacího jazyku Jinja. Díky němu bylo možné generovat stylisticky stejné stránky s pokročilými funkcemi. Pro uživatelsky přívětivý výstup byl použit frontend framework Bottstrap. Za pomocí tohoto frameworku je také docíleno responzivního designu všech stránek. V částech aplikace, kde byly použity tabulky pro zobrazení velkého množství dat, je pak použita knihovna DataTables. Díky knihovně pak bylo možné také řešit data tabulek server-side a tím zvýšit rychlost aplikace. V rámci interakce s uživatelem byly také využity formuláře. Ty byly řešeny rozšířením Flask-WTF, které řešilo jejich bezpečnost a validaci.

Výše zmíněné technologie umožnily vznik pohledů. Díky tomu bylo možné vytvořit všechny pohledy tak, jak bylo navrženo v use case modelu případů užití.

Uživatelům tak byla zpřístupněna přehledná registrace, možnost přihlášení a odhlášení. Udržování uživatelských relací bylo řešeno za pomocí rozšiřujícího balíčku Flask-Login. Registrovaní uživatelé následně mohou využít plné dostupnosti služeb sledování insolvenčních řízení partnerů. Registrovaný uživatel může také spravovat svůj účet a měnit v něm používané jméno, příjmení a e-mail. Pro administrátora systému byl navíc zpřístupněn náhled na všechny uživatele.

V rámci osobního účtu má každý uživatel náhled na všechny své sledované partnery a jejich rozšířený detail. Partneři s insolvenčním řízením jsou graficky zvýrazněni. K načtení údajů partnerů byla využita státní služba ARES. S tou je komunikováno pomocí balíčku Requests.

Uživatel má také přistup k hromadnému pohledu na všechna insolvenční řízení všech svých partnerů. Data o insolvenčních řízeních byla získávána ze státní služby ISIR pomocí balíčku Zeep.

Pro integraci s dalšími aplikacemi uživatelů bylo vytvořeno REST API. K autorizaci v rámci užívání byl použit JWT token, který je generován pro každého uživatele. Speciálním případem pak byly koncové body pro získávání dat do pohledů s tabulkami využívající DataTables. Ty jsou ověřovány přes uživatelské relace.

Na pozadí aplikace jsou pak prováděny periodické aktualizace partnerů a insolvenčních řízení tak, aby data v aplikaci byla stále aktuální. O případných změnách jsou uživatelé informováni prostřednictvím e-mailu, který je odesílán pomocí balíčku Flask-Mail.

# 7 Závěry a doporučení

Programovací jazyk Python se ukázal v rámci práce jako lehce pochopitelný, přehledný a flexibilní. Jazyk dovoluje řešit průběžně vyskytlé problémy mnoha možnými cestami. Programátor má zároveň volné ruce ve výběru balíčků a míře vlastní implementace řešení. K vytvoření aplikace byl použit framework Flask, který obohatil základní možnosti jazyka o pokročilé funkce k řešení problematiky webových aplikací. Jeho rozšíření se stala esenciální a výrazně ulehčila vývoj celé aplikace. Celý projekt byl rozdělen dle MTV architektonického modelu. Návrh a implementace vzorové aplikace trvala přibližně měsíc. Aplikaci je možné spustit dle návodu umístěného v praktické části.

Framework Flask byl vybrán pro vzorový projekt, který je svým rozsahem menší až střední velikosti. Vytvořená aplikace je připravena díky použití modulárního řešení pro možnou implementaci dalších agend. V současné chvíli je výkonost aplikace omezena primárně hardwarovými zdroji webového serveru a databází. Pokud by bylo nutné obsluhovat více jak tisíce dotazů za minutu, musela by se aplikace upravit tak, aby mohla fungovat ve více instancích nad více databázemi.

Aplikace a její implementace přidávání partnerů musela být omezena jen na partnery s identifikačním číslem (IČO). Fyzické osoby nepodnikající bylo nutné z aplikace vynechat, a to kvůli Nařízení evropského parlamentu a rady (EU) 2016/679 o ochraně fyzických osob v souvislosti se zpracováním osobních údajů a o volném pohybu těchto údajů a o zrušení směrnice 95/46/ES (GDPR), které vymezuje poměrně striktní pravidla pro nakládání s osobními údaji fyzických osob. Implementace služeb ARES a ISIR lze považovat za úzké hrdlo aplikace, jelikož na integraci závisí jak načítání partnerů, tak i insolvenčních řízení. Kdyby některá ze služeb změnila své fungování, nebo dokonce byla úplně vypnuta, muselo by se přistoupit k přeprogramování fungování aplikace.

Jako příklady budoucích směrů vývoje aplikace lze uvést: sledování platby DPH (Daň z přidané hodnoty), monitoring insolvencí fyzických osob v souladu s GDPR a uchovávání historie změn v agendách.

# 8 Seznam použité literatury


- ADISESHIAH, Emily Grace, (2018). Single page vs multi-page websites: Design battle!. In: *Justinmind* [online]. 2018-05-10. [cit. 2021-06-30]. Dostupné z: https://www.justinmind.com/blog/single-page-vs-multi-page-websites-design-battle/
- Bootstrap, (2021). *Bootstrap*. Build fast, responsive sites with Bootstrap. [online]. [cit. 2021-08-05]. Dostupné z: https://getbootstrap.com
- BYUNG-KWON, Jung, DONG-SOO, Kim, SEOK-MIN, Yoon, GYU-SANG, Shin, CHONG-SUN, Hwang, (2002). Development and application of a model for analysis and design phases of Web-based system development. *Science in China Series F: Information Sciences.* [online]. Čína: Science China Press, 46, 241–249 [cit. 2021-06-27]. 1862-2836. Dostupné z: https://link.springer.com/article/10.1360/02yf0058Jung a kol. (2002)
- DOLGERT, A., GIBBONS, L., KUZNETSOV, V., (2008). Rapid web development using AJAX and Python. *Journal of Physics: Conference Series.* [online]. Velká Británie: IOP Publishing Ltd., 119. [cit. 2021-06-27]. 1742-6596. Dostupné z: https://doi.org/10.1088/1742-6596/119/4/042011
- Fahadul Shadhin, 2021. The MVT Design Pattern of Django. [online]. 2021-03-04. [cit. 2021-08-06]. Dostupné z: https://python.plainenglish.io/the-mvt-design-pattern-of-django-8fd47c61f582
- CHAE, MinSu, LEE, HwaMin, LEE, Kiyeol, (2019). A performance comparison of linux containers and virtual machines using Docker and KVM. *Cluster Computing*. [online]. Nizozemsko: Kluwer Academic Publishers, 22, 1765–1775 [cit. 2021-06-29]. 1573-7543. Dostupné z: https://doi.org/10.1007/s10586-017-1511-2
- JetBrains s.r.o., (2021). *PyCharm: the Python IDE for professional Developers by JetBrains.* [online]. [cit. 2021-08-05]. Dostupné z: https://www.jetbrains.com/pycharm/
- KAHLON, Navinderjit Kaur, CHAHAL, Kuljit Kaur, NARANG, Sukhleen Bindra, (2016). Managing QoS degradation of partner web services: A proactive and preventive approach. *Journal of Service Science Research.*

[online]. Spojené státy americké: Herndon VA: Scientific Research Pub., 8, 131–159 [cit. 2021-06-28]. 1940-9907. Dostupné z: https://doi.org/10.1007/s12927-016-0007-6

- Ken Reitz, (2016). Requests: HTTP for Humans. In: *GitHub*. [online]. [cit. 2021-08-05]. Dostupné z: https://docs.python-requests.org/en/master/
- KERDOUDI, Mohamed Lamine, TIBERMACINE, Chouki, SADOU, Salah, (2016). Opening web applications for third-party development: a service-oriented solution. *Service Oriented Computing and Applications.* [online]. Velká Británie: Springer London, 10, 437–463 [cit. 2021-06-27]. 1863-2394. Dostupné z: https://doi.org/10.1007/s11761-016-0192-7
- Marcelo Pastorino, 2021. Frontend vs. backend: what's the difference? [online]. [cit. 2021-08-06]. Dostupné z: https://www.pluralsight.com/blog/software-development/front-end-vs-back-end
- Matt Wright, (2014). Flask-Mail. *In: GitHub*. [online]. [cit. 2021-08-05]. Dostupné z: https://pythonhosted.org/Flask-Mail/
- Max Countryman, (2021). Flask-Login. In: *GitHub*. [online]. [cit. 2021-08-05]. Dostupné z: https://flask-login.readthedocs.io/en/latest/
- Michael van Tellingen, (2016). Zeep: Python SOAP client. In: *GitHub*. [online]. [cit. 2021-08-05]. Dostupné z: https://docs.python-zeep.org/en/master/
- Ministerstvo financí (2013). *Administrativní registr ekonomických subjektů.* [online]. [cit. 2021-08-05]. Dostupné z: http://wwwinfo.mfcr.cz/ares/
- Ministerstvo spravedlnosti České republiky, (2021). Nápověda. *Insolvenční rejstřík.* [online]. [cit. 2021-08-05]. Dostupné z: https://isir.justice.cz/isir/common/stat.do?kodStranky=NAPOVEDA
- MUTHUKRISHNAN, Priyadharshini, SAKTHIVEL, V., RAMACHANDRAN, Baskaran, SRIHARI, K., (2020). Technical analysis on security realization in web services for e-business management. *Information Systems and e-Business Management.* [online]. Německo: Springer Verlag, 18, 427–438 [cit.

2021-06-28]. 1617-9854. Dostupné z: https://doi.org/10.1007/s10257-019-00423-w

- NIKOLAIDOU, M., ANAGNOSTOPOULOS, D., (2005). A Systematic Approach for Configuring Web-Based Information Systems. *Distributed and Parallel Databases.* [online]. Nizozemsko: Springer Netherlands, 17, 267–290 [cit. 2021-06-27]. 1573-7578. Dostupné z: https://doi.org/10.1007/s10619-005-6832-0
- O'GRADY, Stephen, (2021). The RedMonk Programming Language Rankings: January 2021. In: *RedMonk* [online]. 2021-03-01. [cit. 2021-06-30]. Dostupné z: https://redmonk.com/sogrady/2021/03/01/language-rankings-1-21/
- Oracle Corporation, (2021). *MySQL*. [online]. [cit. 2021-08-05]. Dostupné z: https://www.mysql.com
- Oracle, (2021). Co je to databáze?. *Oracle Česká republika.* [online]. [cit. 2021-08-05]. Dostupné z: https://www.oracle.com/cz/database/what-is-database/
- Pallets, (2010a). Flask web development, one drop at time*. Flask.* [online]. [cit. 2021-08-05]. Dostupné z: https://flask.palletsprojects.com/en/2.0.x/
- Pallets, (2010b). Flask-SQLAlchemy. *Flask*. [online]. [cit. 2021-08-05]. Dostupné z: https://flask-sqlalchemy.palletsprojects.com/en/2.x/
- PENG, Shuanghe, LIU, Peiyao, HAN, Jing, (2019). A Python Security Analysis Framework in Integrity Verification and Vulnerability Detection. *Wuhan University Journal of Natural Sciences.* [online]. Čína: Wuhan University 24, 141–148 [cit. 2021-06-28]. 1993-4998. Dostupné z: https://doi.org/10.1007/s11859-019-1379-5
- PRAMANICK, Sumanta, (2021). Top 7 Backend Web Development Frameworks in 2021. In: *Kellton Tech* [online]. 2021-02-22. [cit. 2021-06-30]. Dostupné z: https://www.kelltontech.com/kellton-tech-blog/top-7-backend-web-development-frameworks-in-2021
- Python Software Foundation, (2021). *Python*. [online]. [cit. 2021-08-05]. Dostupné z: https://www.python.org

- SpryMedia Ltd., (2021). *DataTables*. Add advanced interaction controls to your HTML tables the free & easy way. [online]. [cit. 2021-08-05]. Dostupné z: https://www.datatables.net
- TIOBE, 2021. TIOBE Index for June 2021. [online]. [cit. 2021-06-30]. Dostupné z: https://www.tiobe.com/tiobe-index/
- VERMA, A., PEDROSA, L., KORUPOLU, M., OPPENHEIMER, D., TUNE, E., WILKES, J., (2015). Large-scale cluster management at Google with Borg. *Proceedings of the European Conference on Computer Systems (EuroSys)*. [online]. 1-17. [cit. 2021-06-29]. Dostupné z: https://research.google/pubs/pub43438/
- WTForms, (2010). Flask WTF. *Flask*. [online]. [cit. 2021-08-05]. Dostupné z: https://flask-wtf.readthedocs.io/en/0.15.x/

# 9 Přílohy

1) Archiv zdrojových kódů vzorové aplikace Monitoring insolvencí (dostupný pouze v elektronické formě)

Univerzita Hradec Králové
Fakulta informatiky a managementu

# Zadání bakalářské práce

| | |
|---|---|
| **Autor:** | **Filip Hanš** |
| Studium: | I1800487 |
| Studijní program: | B1802 Aplikovaná informatika |
| Studijní obor: | Aplikovaná informatika |
| **Název bakalářské práce:** | **Vývoj webové aplikace v jazyce Python 3** |
| Název bakalářské práce AJ: | Development of Web Application in Python 3 |

**Cíl, metody, literatura, předpoklady:**

Cíl: Popsat vývoj webových aplikací v jazyce Python 3 a vytvořit ukázkovou aplikaci s využitím vybraných knihoven.

Osnova:

1. Úvod
2. Literární rešerše
3. Python 3 a web
4. Analýza a návrh vzorové aplikace
5. Popis procesu implementace
6. Shrnutí výsledků
7. Závěr

\par-

| | |
|---|---|
| Garantující pracoviště: | Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu |
| Vedoucí práce: | doc. Mgr. Tomáš Kozel, Ph.D. |
| Datum zadání závěrečné práce: | 14.1.2018 |