VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO SPRÁVU A ÚHRADU FAKTUR

SYSTEM FOR MANAGING AND PAYING INVOICES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — RICHARD SEIPEL
AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ HYNEK, Ph.D.
SUPERVISOR

BRNO 2022

Ústav informačních systémů (UIFS) Akademický rok 2021/2022

Zadání bakalářské práce

24405

Student: **Seipel Richard**
Program: Informační technologie
Název: **Systém pro správu a úhradu faktur**
System for Managing and Paying Invoices
Kategorie: Informační systémy
Zadání:

1. Prostudujte problematiku elektronického zasílání, přijímání a splácení faktur, včetně existujících nástrojů určených pro tento účel.
2. Seznamte se s principy tvorby REST aplikačních rozhraní určených pro zpracování a poskytování dat v programovacím jazyce Python a frameworku Falcon.
3. Analyzujte požadavky uživatelů na možnost přijímání a splácení faktur s ohledem na existující aplikaci firmy Roger.
4. Dle analýzy navrhněte vhodné rozšíření současného systému firmy Roger. Pro tento účel dále navrhněte API pro zasílání, zpracování a zobrazení faktur.
5. Navržené řešení API implementujte.
6. Implementované řešení otestujte a navrhněte další možné rozšíření.

Literatura:

- Pilgrim, M.: *Ponořme se do Python(u) 3* [online]. Praha : CZ.NIC, 2010, CZ.NIC. ISBN 978-80-904248-2-1.
- Falcon: *User Guide* [online]. 2021 [cit. 2021-08-27]. Dostupné z: https://falcon.readthedocs.io/en/stable/user/index.html
- SQLAlchemy: *SQLAlchemy Documentation* [online]. 2021 [cit. 2021-08-27]. Dostupné z: https://docs.sqlalchemy.org/en/14/

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hynek Jiří, Ing., Ph.D.**
Konzultant: Bartoš Stanislav, Ing., Roger
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 25. října 2021

Abstrakt

Cieľom tejto práce je zjednodušiť proces platenia faktúr a taktiež zredukovať potrebu ich zasielania v papierovej podobe. Súčasťou systému je možnosť zadávania faktúr zo strany firiem, rozhranie pre ich zobrazenie a využitie smernice PSD2 pre ich platbu. Jadro systému je implementované v jazyku Python, s využitím frameworku Falcon na tvorbu REST API. Potrebné dáta sú ukladané do databázy vytvorenej v PostgreSQL pomocou SQLAlchemy. Klientská časť je vytvorená formou webovej aplikácie, využitím zaužívaných nástrojov HTML, CSS, Javascriptu a frameworku React. Zadávateľom práce je Platební instituce Roger a.s., ktorá na používanie PSD2 platieb disponuje licenciou.

Abstract

This work aims to simplify the process of paying invoices and also reduce the need to send them in paper form. The system includes the ability to enter invoices from companies, the interface for their display, and the use of the PSD2 directive for their payment. The core of the system is implemented in Python, using the Falcon framework to create the REST API. The necessary data is stored in a database created in PostgreSQL using SQLAlchemy. The client-side is created as a web application, using the usual tools HTML, CSS, Javascript, and the React framework. The submitter of this assignment is the Payment Institution Roger a.s., which has a license to use PSD2 payments.

Kľúčové slová

faktúra, vystavovanie, platba, účet, PSD2, Roger, web, aplikácia, systém, Python, Falcon, SQLAlchemy, PostgreSQL, REST, React, API

Keywords

invoice, issuing, payment, bill, PSD2, Roger, web, application, system, Python, Falcon, SQLAlchemy, PostgreSQL, REST, React, API

Citácia

SEIPEL, Richard. *Systém pro správu a úhradu faktur*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jiří Hynek, Ph.D.

Systém pro správu a úhradu faktur

Prehlásenie

Vyhlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Jiřího Hyneka, Ph.D. Ďalšie informácie mi poskytol konzultant Ing. Stanislav Bartoš. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

. . . . . .
Richard Seipel
8. mája 2022

Poďakovanie

Chcel by som poďakovať vedúcemu mojej práce Ing. Jiřímu Hynekovi Ph.D. a konzultantovi Ing. Stanislavovi Bartošovi z firmy Platební instituce Roger a.s., za všetok čas a úsilie, ktoré mi venovali pri tvorbe mojej práce. Ďalej ďakujem Ing. Tomášovi Slobodníkovi a Ing. Petrovi Judovi za poskytnutie vhodného prostredia na jej tvorbu. Taktiež ďakujem mojej rodine a kamarátom za povzbudenie a podporu.

Obsah

1 Úvod **3**

2 Význam faktúr **4**
2.1 Platenie dane z pridanej hodnoty (DPH) 4
2.1.1 Platiteľ DPH 4
2.1.2 Neplatiteľ DPH 5
2.2 Obsah faktúry 5
2.2.1 Identifikačné číslo osoby (IČO) 6
2.2.2 Identifikačné číslo k DPH (DIČ, VAT ID) 6
2.3 Overenie faktúry 7

3 Spôsoby platieb **8**
3.1 Platobné karty 8
3.2 Platba bankovým prevodom 9
3.3 PSD2 9
3.3.1 PSD2 platby 10
3.3.2 Tretie strany 10
3.3.3 Silná autentifikácia zákazníka (SCA) 10

4 Komunikácia klient-server **12**
4.1 Vrstvy aplikácie 12
4.2 Sieťové protokoly 13
4.2.1 HTTP 13
4.2.2 HTTPS 14
4.3 Rozhrania medzi vrstvami 14
4.3.1 API 14
4.3.2 Formáty na prenos dát 15
4.3.3 Typy API 16
4.3.4 Technológie na tvorbu REST API 18
4.4 Zabezpečenie komunikácie 18
4.4.1 Autentifikácia 18
4.4.2 Autorizácia 19
4.4.3 Zabezpečenie pri komunikácii cez protokol HTTP/HTTPS 19

5 Analýza požiadavok **21**
5.1 Základné požiadavky 21
5.2 Podobné systémy 22
5.2.1 Fakturačné systémy 22

5.2.2 Interné spôsoby prijímania a úhrady faktúr . . . 24
5.3 Profil používateľa . . . 25
5.4 Pridaná funkcionalita na základe analýzy . . . 25
5.5 Podrobné požiadavky . . . 25

6 Návrh **27**
6.1 Ukladanie dát . . . 27
6.1.1 Používatelia . . . 27
6.1.2 Kontakty . . . 28
6.1.3 Faktúra . . . 28
6.2 Operácie s dátami . . . 29
6.2.1 Používateľ . . . 29
6.2.2 Podnikajúci používateľ . . . 29
6.2.3 Zamestnanec . . . 30
6.2.4 Kontakty . . . 30
6.2.5 Faktúry . . . 30
6.2.6 Prihlásenie . . . 31
6.3 Návrh používateľského rozhrania . . . 32
6.3.1 Používateľská časť . . . 32
6.3.2 Administrátorská časť . . . 33

7 Implementácia **35**
7.1 Architektúra . . . 35
7.2 Databáza . . . 35
7.3 Server . . . 36
7.4 Klient . . . 37

8 Testovanie **40**
8.1 Testovanie aplikačného rozhrania . . . 40
8.1.1 Postman . . . 40
8.1.2 Automatické testy . . . 40
8.1.3 pgAdmin . . . 41
8.2 Testovanie požívateľského rozhrania . . . 41
8.3 Navrhované rozšírenia . . . 41

9 Záver **43**

Literatúra **44**

Kapitola 1

Úvod

Už dlhodobo sa vraví o digitalizácii rôznych úradných procesov, no nie je to vždy aj realitou. Stále je čo zlepšovať a ponúkať prístupnejšie a priamočiarejšie možnosti. V dnešnom svete ľudia a firmy neustále za niečo platia, či už je to nová elektronika, telefónne paušály, nájom, energie, alebo poplatky sa dodanie služieb. Z týchto poplatkov musia byť všetky riadne zdokumentované a zdanené. Na tento účel firmám slúži faktúra.

Dnes je už bežné, že nám faktúry chodia mailom, no ak sa k nim chceme po nejakom čase vrátiť, je vyhľadávanie v emailovej schránke dosť nepraktické a pomalé. Ešte podstatnejšie je to pre firmy, ktoré okrem prijatých faktúr potrebujú mať prehľad aj v tých, ktoré vystavili. Ďalším procesom je samotné zaplatenie faktúry, ktoré môže trvať aj pár dní, spolu s osobnou cestou na poštu, alebo na druhej strane len pár sekúnd platbou na internete.

V tejto práci je mojou úlohou takýto systém na vytváranie, správu a platenie faktúr vytvoriť. Rozoberám v nej na čo faktúry slúžia, ako ich do systému zadávať a aké sú možnosti ich platby. Analyzujem, čo firmy a ich odberatelia potrebujú, ako tento problém rieši konkurencia a taktiež, ako má tento systém používateľom uľahčiť život. Návrh na zadanie som dostal od platobnej inštitúcie Roger, ktorá výsledky tejto práce plánuje využiť vo svojej novej aplikácii.

V kapitole 2 popisujem, na čo faktúry slúžia a aké údaje o nich je potrebné do systému zadávať. V nasledujúcej kapitole 3 sa venujem existujúcim možnostiam platieb, vrátane smernice PSD2. Kapitola 4 popisuje, pomocou akých technológií je možné zasielať informácie medzi rôznymi časťami systému, ako vytvoriť aplikačné rozhranie a ako systém zabezpečiť. Zámerom je, aby mohli s faktúrami v systéme pracovať používatelia z rôznych zariadení a systém mohol byť prepojený s existujúcou aplikáciou firmy Roger. V kapitole 5 bližšie opisujem cieľ mojej práce, požiadavky firmy Roger a analyzujem aktuálnu situáciu na trhu. Kapitola 6 je venovaná návrhu systému na základe požiadavok vyplývajúcich z analýzy. Následne kapitola 7 popisuje, aké technológie som v práci použil a ako som pomocou nich navrhnutý systém implementoval. V poslednej kapitole 8 popisujem, ako bol systém testovaný a navrhujem jeho budúce rozšírenia.

Kapitola 2

Význam faktúr

Faktúra je štandardizovaný dokument, ktorý slúži ako účtovný doklad o predaji tovaru alebo poskytnutí služby. Zo zákona je povinnosťou viesť účtovníctvo pre každú osobu, ktorá vykonáva ekonomickú činnosť [4]. Osoby, ktoré účtovné doklady vydávajú, sú predovšetkým podnikatelia. Účtovný doklad môže vystaviť taktiež nepodnikateľ, ktorý predáva tovar alebo poskytuje službu len zriedkavo. Aj keď vystaviť faktúru nie je jeho povinnosťou, môže si ju od neho odberateľ vyžiadať ako doklad do svojho účtovníctva. Pre osoby, ktoré sú platiteľmi DPH je faktúra taktiež daňovým dokladom. Dôležité je tak uviesť, ako funguje platenie daní z pridanej hodnoty.

2.1 Platenie dane z pridanej hodnoty (DPH)

Daň z pridanej hodnoty je najvýznamnejšou nepriamou daňou, teda daňou, ktorú odvádza iná osoba než tá, ktorá ju reálne platí. Vzťahuje sa na obchodné činnosti zahŕňajúce výrobu a distribúciu tovaru a tiež poskytovanie služieb. Je zároveň aj spotrebnou daňou, keďže osoba, ktorá ju vo výsledku platí, je konečný spotrebiteľ.

Osoby sa delia na platiteľov a neplatiteľov DPH. Platiteľom DPH je osoba, ktorá daň odovzdáva štátu. Bežní občania nie sú platiteľmi DPH, keďže DPH platia v cene tovaru a služieb, ktoré nakupujú práve od platiteľov DPH (obrázok 2.1, pravá časť). Platitelia túto daň neskôr zaplatia štátu podaním daňového priznania.

Tovar či služby môžu osoby zakúpiť aj od neplatiteľov DPH a vtedy predajca túto daň štátu neodvádza. Preto môže ponúkať nižšie ceny alebo si viac zárobku nechať pre seba [11]. Platia ale určité podmienky, kedy sa osoba vykonávajúca ekonomickú činnosť musí platiteľom DPH stať.

2.1.1 Platiteľ DPH

Podľa zákona č. 235/2004 [25] sa osoba so sídlom alebo miestom podnikania v Českej republike sa stáva platiteľom DPH napríklad:

- ak jej obrat za posledných 12 kalendárnych mesiacov prekročí čiastku 1 000 000 Kč,
- ak nadobudne majetok privatizáciou alebo predajom obchodného podniku,
- ak preberie činnosť po inom platiteľovi DPH.

Platiteľ DPH je potom donútený navýšiť cenu ponúkaného tovaru alebo služieb práve o túto daň. Môže ale ponúkať tovar alebo služby aj inému platiteľovi DPH, ktorý má

možnosť v daňovom priznaní požiadať štát o jej vrátenie. Vo výsledku tak platiteľ DPH dokáže nakúpiť od iného platiteľa DPH za cenu bez dane (obrázok 2.1, ľavá časť). V daňovom priznaní platiteľ DPH odovzdáva štátu daň z toho, čo zarobil od svojich zákazníkov a štát mu vráti daň, ktorú zaplatil naviac keď nakupoval.

Výnimkou je, ak nakupuje od neplatiteľa DPH, kedy je platená suma bez DPH. V tomto prípade mu zo sumy nie je čo vrátiť a je tak pre neho rovnako vysoká, ako pre neplatiteľov DPH.

Obr. 2.1: Rozdiel medzi platiteľom a neplatiteľom DPH pri nákupe [13].

2.1.2 Neplatiteľ DPH

Neplatiteľom DPH je podnikateľ alebo nepodnikateľ, ktorý nespĺňa požiadavky na platiteľa DPH, alebo sa ním dobrovoľne nestal.

2.2 Obsah faktúry

Aby mohla byť faktúra užitočným dokladom pre dokumentáciu každého predaja tovaru alebo služby, je potrebné, aby mala špecifikované údaje, ktoré musí obsahovať. Ak má slúžiť aj ako daňový doklad, musí obsahovať výšku DPH v percentách a výslednú vypočítanú daň. Tú odberateľ platí naviac k požadovanej základnej sume. Vďaka týmto údajom slúži faktúra platiteľovi DPH ako doklad, koľko na daniach vybral a koľko zaplatil. Podľa smernice európskej únie z roku 2006 [4] musí každá faktúra obsahovať:

- dátum vyhotovenia faktúry,
- poradové číslo ktoré jednoznačne identifikuje faktúru,
- identifikačné číslo pre DPH, pod ktorým zdaniteľná osoba dodala tovar alebo poskytla služby,
- identifikačné číslo pre DPH odberateľa pod ktorým mu bol dodaný tovar alebo poskytnuté služby, za ktoré je povinný platiť daň,
- úplné meno a adresu zdaniteľnej osoby a odberateľa,
- množstvo a druh dodaného tovaru alebo rozsah a druh poskytnutých služieb,

- dátum, kedy sa uskutočnilo alebo ukončilo dodanie tovaru alebo poskytovanie služieb, alebo dátum, kedy bola prijatá zálohová platba, ak tento dátum možno určiť a ak sa odlišuje od dátumu vyhotovenia faktúry,
- základ dane pre každú sadzbu alebo údaj o oslobodení od dane, jednotkovú cenu bez DPH, ako aj prípadné zľavy alebo rabaty, ak nie sú obsiahnuté v jednotkovej cene,
- uplatnenú sadzbu DPH,
- výšku DPH, ktorá sa má zaplatiť, okrem prípadov, keď sa uplatňuje osobitná úprava, pri ktorej sa podľa tejto smernice tento údaj vypúšťa,
- v prípade oslobodenia od dane alebo ak je zaplatiť daň povinný odberateľ, odkaz na uplatňované ustanovenie tejto smernice alebo na príslušné vnútroštátne ustanovenie, alebo iný odkaz, ktorý uvádza, že dodanie tovaru alebo poskytovanie služieb je oslobodené od dane alebo je predmetom prenesenia daňovej povinnosti,
- v prípade dodania nových dopravných prostriedkov uskutočneného v súlade s podmienkami ustanovenými v článku 138 ods. 1 a článku 138 ods. 2 písm. a) údaje uvedené v článku 2 ods. 2 písm. b),
- v prípade uplatňovania osobitnej úpravy pre cestovné kancelárie odkaz na článok 306 alebo na príslušné vnútroštátne ustanovenia, alebo iný odkaz, ktorý uvádza, že sa uplatnila táto úprava,
- v prípade uplatňovania jednej z osobitných úprav na použitý tovar, umelecké diela, zberateľské predmety a starožitnosti odkaz na články 313, 326 alebo 333 alebo na príslušné vnútroštátne ustanovenia, alebo iný odkaz, ktorý uvádza, že sa uplatnila jedna z týchto úprav,
- ak je osobou povinnou platiť daň daňový zástupca v zmysle článku 204, identifikačné číslo pre DPH tohto daňového zástupcu uvedené v článku 214 spolu s jeho celým menom a adresou.

Členské štáty nevyžadujú, aby sa faktúry podpisovali. Na územiach, na ktorých sa dodávajú tovary alebo poskytujú služby, môžu povoliť, aby sa na faktúrach vynechali niektoré povinné údaje. Osoby ktoré nepodnikajú, sú identifikované iba osobnými údajmi. Ostatné osoby majú štátom priradené identifikačné čísla.

2.2.1 Identifikačné číslo osoby (IČO)

Identifikačné číslo osoby je v Českej republike 8-miestne číslo, ktoré jednoznačne identifikuje podnikajúcu fyzickú alebo právnickú osobu, napríklad živnostníka alebo firmu. Táto osoba nemusí byť platiteľom DPH.

2.2.2 Identifikačné číslo k DPH (DIČ, VAT ID)

Identifikačné číslo k DPH, v Českej republike uvádzané ako daňové identifikačné číslo (DIČ) je číslo, ktoré je pridelené každému platiteľovi DPH a jednoznačne ho identifikuje. Podľa smernice európskej únie začína toto číslo každého členského štátu kódom krajiny a ďalej nasleduje postupnosť čísel alebo iných znakov zvolených osobitne každým štátom [4]. Každé takéto identifikačné číslo, označované aj ako „VAT ID“ alebo „VAT number“ jednoznačne

identifikuje osobu medzi všetkými štátmi európskej únie a taktiež aj štát v ktorom daná osoba vykonáva ekonomickú činnosť [10].

2.3 Overenie faktúry

Zdaniteľná osoba je povinná zabezpečiť vierohodnosť pôvodu, neporušenosť obsahu a čitateľnosť faktúry od jej vydania do konca obdobia na uchovávanie faktúry. Vierohodnosťou pôvodu faktúry sa rozumie potvrdenie totožnosti dodávateľa tovaru alebo služby alebo osoby, ktorá v mene dodávateľa vyhotovila faktúru. Neporušenosťou obsahu faktúry sa rozumie zachovanie obsahu faktúry. Členské štáty európskej únie akceptujú elektronické faktúry, ak je ich vierohodnosť a neporušenosť zaručená:

- prostredníctvom elektronického podpisu,
- prostredníctvom elektronickej výmeny údajov (EDI).

Pripadne aj inými spôsobmi za predpokladu, že ich akceptujú dotknuté členské štáty [4]. Podľa slovenského zákona [29] možno použiť ešte:

- kontrolné mechanizmy podnikových procesov, ktoré spoľahlivo zabezpečia priraditeľnosť faktúry k dokumentom súvisiacim s dodaním tovaru alebo služby,
- iný spôsob zabezpečujúci vierohodnosť pôvodu a neporušenosť obsahu faktúry.

Viac sa spôsobu overenia odosielateľa, prijímateľa a obsahu faktúry venujem v sekcii o zabezpečení 4.4.

Kapitola 3

Spôsoby platieb

Existujúce faktúry je potrebné nejakým spôsobom zaplatiť. Spôsob platby na pošte je už považovaný za zastaralý a platby sa presúvajú do online sveta. Bežným spôsobom platenia pravidelných faktúr je trvalý príkaz alebo inkaso. Pri nepravidelných platbách je to už trochu komplikovanejšie. Možnosťou je opäť odniesť peniaze na poštu, pobočku banky či rovno na pobočku spoločnosti. No ak človeku nerobí problém pracovať s počítačom, zvolí radšej online bankový prevod, alebo ešte jednoduchšiu platbu kartou.

3.1 Platobné karty

Používanie platobných kariet sa v poslednom čase sa veľmi rozšírilo. Neprítomnosť platobného terminálu v obchode či reštaurácii je už skôr výnimkou. Pre rýchlosť a pohodlnosť ich použitia sa dostávajú stále viac do popredia.

Stále viac systémov podporuje aj služby Google Pay[1] a Apple Pay[2], pri ktorých naviac odpadá potreba zaoberať sa bezpečným uložením údajov z karty, alebo ich prepisovaním pred každou platbou. Tiež nie je potrebné platbu dodatočne potvrdzovať PIN kódom alebo v aplikácii banky, keďže pred jej vykonaním sú vyžadované údaje používané pre odomknutie zariadenia (kód, heslo, biometria).

Platobné karty však nevlastnia priamo banky, ale spoločnosti zaoberajúce sa spracovaním transakcií. Medzi najrozšírenejšie na európskom trhu patria napríklad Visa a Mastercard [32]. Tie si za vykonané platby účtujú poplatky. Typická transakcia platobnou kartou zahŕňa päť strán [24]. Okrem samotného spracovateľa platieb (Mastercard, Visa) zahŕňa:

- spotrebiteľa a banku, ktorá vydala jeho kartu (*issuer bank*),
- obchodníka a jeho banku, do ktorej platba smeruje (*acquirer bank*).

Platobná spoločnosť si účtuje od oboch bánk poplatky za každú vykonanú transakciu a jej autorizáciu. Banka obchodníka platí banke spotrebiteľa poplatok, ktorým kompenzuje jej časť poplatkov a risk možného podvodu od držiteľa karty. Odmenou pre banku obchodníka je, že má garantované pripísanie peňazí od banky spotrebiteľa a môže tak peniaze ihneď zaslať obchodníkovi. Keďže tento proces prináša výhody hlavne obchodníkovi, jeho banka si od neho vyžaduje poplatok, ktorý pokrýva jej náklady [24] (obrázok 3.1).

Každým nákupom kartou tak obchodník prichádza o časť zisku. Čím viac platieb prebieha prostredníctvom kariet, tým citeľnejšie sú tieto straty pre obchodníka.

[1] Viac o službe Google Pay – https://pay.google.com/intl/cs_cz/about/
[2] Viac o službe Apple Pay – https://www.apple.com/cz/apple-pay/

Obr. 3.1: Priebeh okamžitej platby obchodníkovi od banky vydávajúcej kartu, spolu s poplatkami. Z účtu spotrebiteľa sú prostriedky stiahnuté až dodatočne.

3.2 Platba bankovým prevodom

Platba na účet je tiež možnou alternatívou. Keďže pri tomto spôsobe komunikuje zákazník priamo s bankou bez ďalších strán, odpadajú poplatky naviac. Banka si ale musí prevod peňazí skontrolovať a vykonať sama. Postupne sa dĺžka trvania prevodu skrátila z niekoľkých dní až na pár sekúnd [43]. To otvára nové možnosti týchto platieb. Výrazným krokom bolo aj rozšírenie platieb pomocou QR kódu.

Banky ponúkajú aj možnosť trvalého príkazu, čo je výhodou pri platení dopredu známej sumy pravidelne. Tiež ponúkajú inkaso, to je vhodné, ak sa platená suma za každé obdobie líši.

3.3 PSD2

PSD2 (*Payment Services Directive 2*) [6] je smernica vydaná Európskou komisiou, platná od 13. januára 2018. Jej účelom je prispieť k jednotnému trhu elektronických platieb v rámci Európskej únie, pomôcť ich celkovému rozvoju a tiež príchodu inovácii. Zameriava sa napríklad na regulovanie poskytovania služieb prostredníctvom *FinTech*[3] spoločností a vyššiu ochranu spotrebiteľa [33].

Jej predchodcom bola smernica (PSD1) [5], ktorá definovala rovnaké pravidlá pre spracovanie platobných príkazov v rámci Európy. Ďalší vývoj v oblasti platieb však priniesol nové služby, ktoré touto smernicou regulované neboli. Cieľom smernice PSD2 je uľahčiť vstup nových poskytovateľov platobných služieb na európsky trh a zároveň priniesť pravidlá, podľa ktorých budú môcť tieto subjekty poskytovať platobné služby koncovým používateľom [37].

[3]Spoločnosti, ktoré prinášajú technológie do sveta financií [15].

3.3.1 PSD2 platby

Pre možnosti platby je najvýznamnejšou prinášanou zmenou povinnosť sprístupniť vybrané služby bánk tzv. *tretím stranám* prostredníctvom technologického rozhrania API (*Application Programming Interface*)[4].

Cez tieto rozhrania je na základe súhlasu klienta možné **iniciovať platobný príkaz** alebo získať vybrané **informácie o platobnom účte**. Práve pre zaručenie ochrany klienta banky a jeho údajov, PSD2 definuje komplexné pravidlá pre prístup tretích strán k jeho bankovému účtu a tiež pre vykonávanie operácií nad ním. [37].

3.3.2 Tretie strany

Rozsah PSD2 sa rozširuje na inovatívne platobné služby a nových poskytovateľov na trhu, ako sú FinTechy. Tieto spoločnosti sa tiež nazývajú poskytovatelia platobných služieb tretích strán (TPP) [12]. Tí sú delení do dvoch kategórií [37]:

- **AISP** – Account Information Service Provider *(Poskytovateľ služieb informovania o účte)*. Ten na základe súhlasu klienta spracováva základné informácie o jeho účte a tiež transakciách, ktoré na ňom prebiehajú. Napríklad, ak má klient vedené účty vo viacerých bankách, cez aplikáciu tohto poskytovateľa môže vidieť zostatky na všetkých svojich účtoch naraz.

- **PISP** – Payment Initiation Service Provider *(Poskytovateľ platobných inicializačných služieb)*. Poskytovateľ môže vytvoriť platobný príkaz z klientovho účtu v banke, na základe jeho žiadosti. Napríklad, pri nákupe v e-shope si klient zvolí, že nezaplatí kartou, ale využije nový spôsob úhrady za tovar. Platobný príkaz zadá prostredníctvom aplikácie poskytovateľa, pričom príkaz je následne zrealizovaný z bankového účtu klienta. Do tejto kategórie spadá aj aplikácia firmy Roger, ktorej rozšíreniu sa venujem.

3.3.3 Silná autentifikácia zákazníka (SCA)

PSD2 stanovuje, že na prístup k platobnému účtu a na uskutočňovanie online platieb sa musí použiť silná autentifikácia zákazníka. Banky a iní poskytovatelia platobných služieb sú povinní zaviesť potrebnú infraštruktúru pre SCA. Spotrebitelia a obchodníci musia byť taktiež pripravení a vyškolení.

Vďaka PSD2 sú spotrebitelia lepšie chránení pri uskutočňovaní elektronických platieb alebo transakcií, napríklad pri používaní online bankovníctva alebo nákupe online. Silná autentifikácia zákazníka (SCA) je základom pre prístup k platobnému účtu, ako aj pre uskutočňovanie platieb online.

Na preukázanie svojej totožnosti sú používatelia povinní poskytnúť aspoň dva samostatné prvky z týchto troch:

- niečo, čo vedia (heslo alebo PIN kód),

- niečo, čo vlastnia (karta, mobilný telefón),

- niečo, čo sú (biometria, napr. odtlačok prsta alebo skenovanie tváre).

[4] Podrobnejšie v kapitole o komunikácii 4.3.1

Silná autentifikácia zákazníka je aj mimo online platieb bežne používaná v celej EÚ. Napríklad, keď zákazníci platia kartou v kamenných obchodoch, sú povinní overiť transakciu zadaním PIN kódu na čítačke kariet [12].

Kapitola 4

Komunikácia klient-server

Na začiatku tvorby systému je dôležité zvoliť, na aké podčasti je ho vhodné rozdeliť. Závisí to od účelu, ktorý sa od neho očakáva. Pri vhodnom rozdelení sa môže napríklad každá časť nachádzať na inom zariadení, môže byť vyvíjaná iným tímom, či dokonca v inom programovacom jazyku. Na to, aby takýto systém fungoval, je dôležité, aby dokázali jeho jednotlivé časti efektívne komunikovať. Popisované informácie som vyberal s ohľadom na technológie použité v tejto práci, ako trojvrstvová architektúra, protokol HTTP, architektúra REST, formát JSON či JWT token.

4.1 Vrstvy aplikácie

Základné rozdelenie systému je na prezentačnú, aplikačnú a dátovú vrstvu (v angličtine *layer*) [18]. Následne existujú architektúry, určujúce, ktoré z týchto vrstiev majú byť oddelené a ktoré spojené. Nazvané sú podľa počtu výsledných vrstiev (*tier*), ktoré daná architektúra má [30]:

- **Jednovrstvová architektúra** – spája všetky tri vrstvy do jednej, dáta sú ukladané na zariadení, na ktorom je program vykonávaný a taktiež zobrazovaný. Komunikácia medzi vrstvami prebieha interne.
- **Dvojvrstvová architektúra** – oddeľuje databázovú vrstvu od aplikačnej a prezentačnej. Operácie programu sú vykonávané na klientskom počítači, kde sú následne aj zobrazované. Dáta klientov sú uložené na databázovom serveri a klient s nimi pracuje vzdialene. Systém je rozdelený na klienta a server, ktorí medzi sebou komunikujú, preto je táto architektúra nazývaná aj ako *klient-server*.
- **Trojvrstvová architektúra** – oddeľuje všetky tri vrstvy samostatne. Dáta sú ukladané na databázovom serveri, aplikačná logika programu beží na aplikačnom serveri a na klientskom počítači ostáva úloha ovládať systém a prezentovať výsledky. Týmto prístupom je klient odľahčený od náročnejších výpočtov a môže bežať aj v internetovom prehliadači. Preto je táto architektúra vhodná najmä pre webové aplikácie.
- **N-vrstvová architektúra** – všeobecne popisuje architektúru s akýmkoľvek počtom vrstiev. Ďalej sa môže napríklad aplikačná vrstva rozdeliť na viacero samostatných vrstiev, aby lepšie abstrahovala prácu s databázou, prípadne klientom.

Na to, aby mohli medzi sebou dve časti systému komunikovať, je potrebné, aby sa dohodli na spoločnom komunikačnom rozhraní. Ak komunikácia prebieha po sieti, potrebuje využívať nejaký sieťový protokol.

4.2 Sieťové protokoly

Technológie určené na komunikáciu vrstiev, ako protokol SOAP alebo architektúra REST, využívajú na tieto účely protokoly aplikačnej vrstvy sieťového modelu. Najpoužívanejší je protokol HTTP, na ktorom je založená práve REST architektúra použitá v tejto práci.

4.2.1 HTTP

HTTP (*Hypertext Transfer Protocol*) je protokol aplikačnej vrstvy, prvotne navrhnutý pre prenos hypertextových dokumentov medzi klientom a serverom. V dnešnej dobe je využívaný všeobecne na prenos dát po sieti. Funguje na princípe zaslania požiadavky klientom a očakávania odpovede od severu [19].

Požiadavka obsahuje použitú HTTP metódu a URL adresu, na ktorú je odosielaná. Ďalej verziu protokolu, potrebné hlavičky s informáciami o požiadavke a v tele požiadavky samotné dáta. Zvolená HTTP metóda odpovedá typu požiadavky. Základné sú tieto [14]:

- GET – požiadavka klienta na zaslanie dát serverom,
- POST – odoslanie dát klienta na server,
- PUT – zmena vybraných dát na serveri,
- DELETE – vymazanie dát zo servera.

V odpovedi servera sa následne nachádza kód reprezentujúci výsledok operácie a prípadne aj dáta, ktoré klient od servera požadoval. Stavové kódy sú rozdelené do piatich kategórií [17]:

- 1XX – informačná odpoveď, operácia ešte nie je dokončená,
- 2XX – operácia akceptovaná,
- 3XX – presmerovanie alebo potreba ďalších operácií pre splnenie požiadavky,
- 4XX – požiadavka nemôže byť splnená (napr. nenájdená adresa, nesprávna syntax, nedostatočná autorizácia),
- 5XX – chyba na strane servera.

Protokol je bezstavový, čo znamená, že server si nedokáže prepájať jednotlivé požiadavky medzi sebou a nadväzovať na tie predchádzajúce. Zo vznikajúcej potreby rozlišovať používateľov webu a udržiavať relácie, sa začali používať v hlavičkách požiadavok identifikátory ako session id alebo token, ktoré sú hlbšie rozoberané v sekcii 4.4 o zabezpečení.

4.2.2 HTTPS

HTTPS (*Hypertext Transfer Protocol Secure*) je zabezpečenou verziou protokolu HTTP. Jeho hlavným prínosom je ochrana súkromia, integrity vymieňaných údajov a taktiež autentifikácia navštívenej webovej stránky. Obojsmerné šifrovanie komunikácie medzi klientom a serverom chráni komunikáciu pred odpočúvaním a manipuláciou (útok *man-in-the-middle*). Pre možnú *autentifikáciu*[1] navštívenej stránky HTTPS vyžaduje, aby dôveryhodná tretia strana (*certifikačná autorita*) podpísala jej digitálne certifikáty [31].

Historicky išlo o drahú operáciu, čo znamenalo, že plne overené HTTPS pripojenia sa zvyčajne používali iba pri platobných službách a firemných informačných systémoch. V dôsledku zvyšovania nárokov na bezpečnosť a dostatku zdrojov, je už dnes protokol plne používaný na väčšine webových stránok [42].

4.3 Rozhrania medzi vrstvami

Aby mohla komunikácia prebiehať nezávisle na implementácii jednotlivých častí, je potrebné vytvoriť rozhranie, ktoré prácu s nimi abstrahuje. Na tento účel sa využíva API.

4.3.1 API

API (*Application Programming Interface*) je komunikačné rozhranie – súbor programov a funkcií, ktoré umožňujú aplikáciam komunikovať v online prostredí. Cez API zadáva jedna aplikácia požiadavku, ktorú druhá aplikácia spracuje (obrázok 4.1).

Obr. 4.1: API prijíma požiadavku klienta a interpretuje aplikácii, ako ju má spracovať. To umožňuje klienta odizolovať od spôsobu, akým sú jeho požiadavky vykonávané. Nepotrebuje napríklad vedieť, s akými ďalšími servermi aplikačný server komunikuje, alebo ako sú dáta ukladané.

[1] Autentifikácia je bližšie popísaná v sekcii 4.4.1.

API jednoznačne oddeľuje prezentačnú časť od aplikačnej a umožňuje napríklad, že v systéme môže byť klientom popri webovej aplikácii aj mobilná, ktorá využije rovnaké API aplikačného servera.

Aby bola zabezpečená kompatibilita komunikujúcich aplikácii, sú na výmenu dát medzi nimi využívané formáty ako JSON, XML (4.3.2) a ďalšie. Existuje viacero spôsobov, ako API implementovať (4.3.3).

4.3.2 Formáty na prenos dát

Dáta prenášané medzi klientom a serverom musia byť nejakým spôsobom reprezentované pri prenose po sieti. Na tento účel slúžia formáty ako JSON alebo XML. Konvertovanie dátového objektu na tieto textové formáty sa nazýva *serializácia*.

XML

XML (*Extensible Markup Language*) je jednoduchý značkovací jazyk vyvinutý webovým konzorciom (W3C) na základe jazyka SGML. Narozdiel od jazyka HTML nemá preddefinované významy značiek, preto je všeobecne využiteľný na popis a štruktúrovanie dát.

XML dokument obsahuje hlavičku a koreňový element, v ktorom môžu byť zanorené ďalšie elementy alebo dáta. Element sa skladá z počiatočnej `<názov>` a koncovej značky `</názov>`, môže obsahovať atribúty `<názov atribút="hodnota">dáta</názov>` a ak neobsahuje dáta, môže byť skrátený `<názov atribút="hodnota"/>` [41]. Pri použití na serializáciu sa hlavička vynecháva.

```
<?xml version="1.0" encoding="UTF-8"?>
<autor>
    <meno>Ján</meno>
    <priezvisko>Kováč</priezvisko>
</autor>
```

JSON

JSON (*JavaScript Object Notation*) je formát na výmenu dát založený na spôsobe zápisu objektov v jazyku JavaScript. Môže byť tvorený dátami typu reťazec, číslo, pravdivostná hodnota, objekt alebo pole objektov. Objekt je tvorený kolekciou dvojíc `"názov": hodnota` a ohraničený zloženými zátvorkami, pričom hodnotu v dvojici predstavuje niektorý zo spomenutých dátových typov. Použitím objektu alebo poľa na mieste hodnoty je možné dáta ďalej štruktúrovať [9].

```
{
    "meno": "Štefan Veľký",
    "email": "stefan@email.com",
    "adresa": "Kolejní 2",
    "psc": "61200",
    "mesto": "Brno",
    "stat": "Česká republika",
    "body": 238,
    "ico": null,
    "tel_cisla": ["+421949183627"]
}
```

V porovnaní s formátom XML má ľahšie čitateľný a kratší zápis. Vďaka menšiemu počtu znakov potrebných na štruktúrovanie dát a kompatibilite s objektami jazyka Javascript, ktorý je často využívaný na klientskej strane, je podstatne efektívnejší [28].

4.3.3 Typy API

RPC

RPC (*Remote Procedure Call*) je prvý spôsob vzdialeného volania funkcií. Umožňuje zavolať funkciu, ktorá môže byť napísaná v inom programovacom jazyku, nachádzať sa na inom zariadení a byť súčasťou iného systému [27].

SOAP

SOAP (*Simple Object Access Protocol*) je protokol na tvorbu API využívaný hlavne v minulosti. Zasielané dáta sú striktne štruktúrované vo formáte XML. Požiadavka sa skladá z hlavičky, tela a prípadného chybového hlásenia. Tie sú ešte zabalené v obálke (*envelope*). Pri odosielaní obálky už nezáleží na použitom prenosovom protokole, no najčastejšie je na jej prenos využívaný protokol HTTP [2].

Jeho výhodou je možnosť zabezpečenia rozšírením WS-Security, ktoré umožňuje zašifrovať telo požiadavky. Prístup k jej obsahu má tak len konkrétny proces, ktorému je určená. Narozdiel od protokolu HTTPS, kde je požiadavka dešifrovaná už na úrovni servera. K spoľahlivosti prispieva integrovaný systém na opakovanie neúspešných požiadavok. Pre tieto vlastnosti sa ešte stále využíva v niektorých finančných a bankových systémoch. Ďalej môžu byť, narozdiel od REST API, zasielané požiadavky zreťazované, pre udržanie kontextu. Nevýhodou je neflexibilnosť pri zmenách na klientovi alebo serveri a celková komplexnosť protokolu. Vo výsledku je oproti REST API náročnejší na implementáciu, naučenie a taktiež pomalší pri komunikácii [34].

REST

REST (*Representational State Transfer*) je dnes najpoužívanejšou architektúrou. Na rozdiel od architektúry SOAP, kde vďaka obálke nezáleží na použitom prenosovom protokole, architektúra REST komunikuje priamo prostredníctvom protokolu HTTP 4.2.1. Využíva jeho metódy, hlavičky, telo aj stavové kódy na oznámenie výsledku požiadavky [34].

Základnými metódami GET, POST, PUT, DELETE popisuje CRUD operácie (*Create, Read, Update, Delete*), ktoré majú byť na serveri vykonané [23]:

- GET → Read – získanie dát,
- POST → Create – pridanie dát,
- PUT → Update – aktualizovanie dát,
- DELETE → Delete – vymazanie dát.

Adresou URL, na ktorú je požiadavka odosielaná, ďalej špecifikuje, na akých dátach má byť vybraná operácia vykonaná. Tieto serverom určené adresy, tzv. koncové body (*endpoints*), tvoria rozhranie pre komunikáciu so serverom a abstrahujú implementáciu jeho funkcionality na pozadí. To umožňuje nezávisle vykonávať zmeny na serveri alebo klientovi, bez potreby zmien v implementácii rozhrania. Hlavičky využíva napríklad na autentifikáciu

klienta alebo špecifikovanie formátu dát, zasielaných v tele správy. Zasielané dáta môžu byť narozdiel od architektúry SOAP štruktúrované v rôznych formátoch. Najčastejšie je využívaný JSON [34].

V HTTP odpovedi je potom stavovým kódom oznámený výsledok požiadavky. Môže napríklad označovať úspech alebo druh chyby, ktorá nastala. Ak majú byť súčasťou odpovede dáta, sú zasielané v jej tele. Taktiež v ňom môže byť bližšie špecifikovaná vzniknutá chyba.

REST architektúra má definované princípy, ktoré by mali byť pri vytváraní API dodržiavané. Ak ich API spĺňa, používa sa pojem „RESTful API“[16]:

- **Jednotné rozhranie** – jeden zdroj dát je reprezentovaný v adrese vždy rovnako (zdroj `users` → `api/users/`, `api/users/123`). Klient volaním rôznych HTTP metód nad rovnakým zdrojom určuje, akú operáciu chce s dátami vykonať. Keďže každá metóda je reprezentovaná slovesom, zdroje sú pomenovávané podstatnými menami v množnom čísle. Spolu tak vytvárajú dobre čitateľnú formu pre človeka (`GET api/users` → získaj zoznam používateľov, `DELETE api/users/123` → vymaž používateľa 123).

- **Architektúra klient-server** – implementácia klienta je nezávislá na implementácii servera. Komunikujú spolu len volaniami na endpointy API.

- **Bezstavová logika** – jedným z najdôležitejších princípov je neukladať žiadne dáta o predchádzajúcich požiadavkách klienta. Ak je pre funkčnosť aplikácie potrebné identifikovať, aký klient požiadavku odoslal, musí klient s každou požiadavkou odosielať autorizačné a autentifikačné údaje.

- **Ukladanie dát do cache** – ak je možné ukladať odpovede servera do *cache*[2], mali by obsahovať informáciu, že to je možné a akú dobu majú byť považované za aktuálne. Tým sa zmenší počet odosielaných požiadavok. Server môže rovnako ukladať odpovede od iných serverov, ak s nejakými komunikuje.

- **Vrstvená architektúra** – má umožniť rozdelenie aplikačnej logiky na viacero serverov bez toho, aby o tom klient potreboval vedieť. Klient komunikuje cez API s jedným serverom a ten môže odosielať požiadavky na ďalšie. Napríklad autentifikačný, databázový alebo server iného systému s ktorým spolupracuje.

- **Kód na vyžiadanie** – ak je to potrebné, server môže zaslať klientovi spustiteľný kód. Príkladom môže byť systém pre šifrovanú komunikáciu. Server zašle klientovi kód na vygenerovanie kľúča a klient tak môže naviazať komunikáciu s iným klientom priamo, bez toho aby mal server možnosť správy dešifrovať.

Priamym používaním protokolu HTTP bez dodatočných procesov a možnosťou voľby úspornejšieho formátu dát, je komunikácia cez REST API podstatne rýchlejšia ako SOAP [26]. Zároveň je jednoduchšia na pochopenie, implementáciu a taktiež flexibilnejšia, keďže nevyžaduje dopredu vytvorenú schému tela požiadavok. Postupne tak architektúru SOAP vytláča.

GraphQL

GraphQL je najnovším z vybraných spôsobov implementácie API. Reaguje na nevýhodu REST API, kde sú zasielané celé štruktúry dát, aj keď klient často potrebuje len malú časť

[2] vyrovnávacia pamäť

z nich. To je pri veľkom množstve požiadavok zbytočne náročné na zdroje a čas. GraphQL tento problém rieši schémou, ktorá je implementovaná na serveri. Klient odosiela na server jej časť s atribútmi, o ktoré má záujem. Server tieto dáta vyhľadá v databáze, agreguje do jednej odpovede a odošle klientovi.

GraphQL tak narozdiel od REST API nepotrebuje špecifikovať zdroj dát v adrese URL a na zasielanie požiadavok mu stačí jeden endpoint. Pre implementáciu klienta je namiesto zoznamu koncových bodov potrebné poznať schému. Sloboda pri výbere parametrov prináša aj nevýhody. Napríklad pri ich nevhodnej kombinácii môžu vzniknúť komplikované databázové dotazy. Taktiež nastáva problém s ukladaním do pamäte cache, keďže nie je možné dopredu určiť, aká kombinácia atribútov bude zvolená.

Táto architektúra je vhodná na projekty s vysokým vyťažením, veľkou variabilitou pri využívaní dát a taktiež pri využívaní mikroslužieb [35].

4.3.4 Technológie na tvorbu REST API

Aplikačné rozhranie je pri komunikácii klient-server implementované na aplikačnom serveri. Existujú rôzne technológie, ktorými je možné API implementovať. Keďže architektúra REST je aktuálne najviac rozšírenou, je ju možné implementovať v rôznych programovacích jazykoch. Medzi najčastejšie používané patria jazyky JavaScript, Python, PHP, Ruby, Java a C#.

Pre každý z jazykov sú rozšírené určité technológie na jeho implementáciu. *Node.js* je open-source behové prostredie pre spustenie jazyka JavaScript mimo webového prehliadača. Môže tak v ňom byť implementovaná aplikačná logika systému na serveri spolu s REST API. Pre jazyk Python sú najpopulárnejšie technológie *Flask* a *Django* [36]. Patrí k ním aj framework Falcon, ktorý je využitý v mojej práci. Viac sa jeho použitiu venujem v kapitole 7. Pre tvorbu REST API v jazyku PHP sú populárne napríklad frameworky *Laravel* a *Lumen*. V jazyku Ruby je využívaný predovšetkým framework *Ruby on Rails* [22]. Pre jazyk Java je populárny napríklad framework *Spring* a pre jazyk C# je najpoužívanejším *ASP.NET*.

4.4 Zabezpečenie komunikácie

Ak implementácia systému vyžaduje rozpoznávanie používateľov, je potrebné, aby sa vedeli pri jeho používaní identifikovať. Používateľ napríklad zadá používateľské meno, ktoré je jedinečné pre daný systém. To však ešte nezabezpečuje, že identifikátor zadal naozaj používateľ, ktorému patrí. Ak má byť systém bezpečný, je potrebné, aby bola identita každého používateľa overená.

4.4.1 Autentifikácia

Pojem autentifikácia označuje proces, ktorý umožňuje elektronické overenie identity. Okrem overenia identity osôb, môže autentifikácia tiež potvrdiť pôvod a integritu údajov v elektronickej forme, ako je napríklad vydanie digitálneho certifikátu na potvrdenie pravosti webovej stránky [8].

Prvým krokom pri overení používateľa je jeho identifikácia, tá slúži na preukázanie používateľovej identity. Po nej nasleduje autentifikácia [38]. Doteraz sú najbežnejšou a najrozšírenejšou forma autentifikácie heslá. Okrem nich je stále viac využívaná biometria, pre jej rýchlosť a jednoduchosť použitia. Ďalej aj rôzne bezpečnostné kódy, PIN kódy či prístupové frázy. V závislosti od systémových požiadavok a typu organizácie sa používa aj viac

druhov autentifikácie súčasne. Kvôli zvýšeným hrozbám a únikom prihlasovacích údajov používateľov, organizácie stále viac využívajú *silnú autentifikáciu*[3], aby ochránili dôverné informácie používateľov pred útočníkmi [7].

4.4.2 Autorizácia

Autorizácia je bezpečnostný mechanizmus na určenie úrovní prístupu alebo oprávnení používateľa/klienta vo vzťahu so systémovými prostriedkami vrátane súborov, služieb, počítačových programov, údajov a funkcií aplikácií. Ide o proces udeľovania alebo odopierania prístupu, ktorý umožňuje používateľovi prístup k rôznym prostriedkom na základe jeho identity.

Väčšina systémov zabezpečenia webu je založená na dvojstupňovom procese. Prvým krokom je autentifikácia, ktorá zabezpečuje identitu používateľa a druhým je autorizácia, ktorá mu na jej základe umožňuje prístup k rôznym zdrojom [39]. Napríklad prihlásený zákazník v eshope bude mať iné možnosti ako zamestnanec alebo administrátor. Tieto skupiny používateľov sa nazývajú *role*.

4.4.3 Zabezpečenie pri komunikácii cez protokol HTTP/HTTPS

Ak server vyžaduje autorizáciu používateľa pre prístup k niektorému z jeho zdrojov, je potrebné, aby ju pred splnením požiadavky vedel overiť. Keďže protokol HTTP je bezstavový, musí server tieto údaje vedieť získať z každej požiadavky odoslanej na daný zdroj. Existuje viacero spôsobov, ako takéto zabezpečenie implementovať.

Autentifikácia prostredníctvom cookies

Cookies sú malé bloky dát ukladané na klientskej strane [3]. Server môže používateľovi po jeho autentifikácii súbor cookie vytvoriť a uložiť mu doň vygenerovaný identifikátor. Tento identifikátor sa nazýva *session id*. Sever si v databáze vytvorí novú reláciu (*session*) a identifikátor k nej uloží. Následne klient identifikátor zasiela v HTTP hlavičke `Cookie` pri každej požiadavke [40].

Server po prijatí požiadavky podľa session id vyhľadá príslušnú reláciu a získa z nej potrebné informácie o používateľovi. Udržiavanie relácie umožňuje pamätať si predchádzajúce požiadavky a nadväzovať na ne. Tým pridáva k protokolu HTTP stavovosť. Tento prístup je pri vyššom zaťažení náročný na zdroje, keďže pre každého prihláseného používateľa musí byť v databáze udržiavaný záznam o relácii [1]. Taktiež je k záznamu v databáze potrebné pristúpiť pred vykonaním každej požiadavky. Komplikuje aj škálovanie databázy, tak aby bol stále zachovaný prístup k rovnakej relácii.

Autentifikácia prostredníctvom tokenu

Token je reťazec znakov nesúci informáciu o používateľovi, ktorý chce byť autentifikovaný. Na prenos tokenu je využívaná HTTP hlavička `Authorization`, so schémou typu *Bearer* (slovensky nositeľ). Táto schéma je určená na zasielanie tokenov podľa protokolu OAuth 2.0 [1]. Bearer token môže použiť ktokoľvek, kto ním disponuje. Jeho použitie nevyžaduje, aby používateľ preukázal jeho vlastníctvo [21]. Jeden z formátov bearer tokenu je JWT.

JWT (*JSON Web Token*) je otvorený štandard (RFC 7519) [20], ktorý definuje spôsob bezpečného prenosu informácií medzi stranami v objekte JSON. Dôveryhodnosť ním pre-

[3] Viac o silnej autentifikácii sa nachádza v kapitole o platbách pri smernici PSD2 (sekcia 3.3.3).

nášaných informácii je zabezpečená digitálnym podpisom. Na podpísanie môže byť použitý algoritmus HMAC alebo RSA. JWT token je zložený z troch častí:

- **Hlavička** (*header*) - obsahuje typ tokenu a zvolený šifrovací algoritmus.
- **Obsah** (*payload*) - dáta používateľa, ktoré sú pre server potrebné (napr. identifikátor či rola). Ďalej voliteľné atribúty napr. dátum vydania tokenu, dátum expirácie alebo označenie vydavateľa.
- **Podpis** (*signature*) - podpis autentifikačného serveru.

Tieto tri časti sú prevedené do formátu *Base64* a spojené bodkou. Zasielaná autorizačná hlavička s tokenom potom vyzerá nasledovne:

```
Authorization: Bearer i72mkx267.lusowng396.hdon7vsj
```

Autentifikácia tokenom je bezstavová, preto je vhodná na použitie pre zabezpečenie endpointov REST API. Na začiatku sa používateľ musí voči systému autentifikovať. Po úspešnej autentifikácii server vygeneruje používateľovi token, ktorý následne zasiela spolu s každou požiadavkou na zabezpečený endpoint API. Server po jeho prijatí vďaka podpisu overí integritu dát v tokene a následne podľa nich môže zistiť napríklad identitu odosielateľa alebo jeho rolu. Ak pozná jeho identitu, môže prípadne získať dodatočné informácie z databázy a podľa toho postupovať vo vykonávaní požiadavky. Použitie tokenov vyťažuje server menej ako udržiavanie relácie, keďže najčastejšie využívané dáta pri požiadavkách sú súčasťou tokenu, ukladaného na klientskej strane [1]. Server tak nemusí nutne pristupovať do databázy pri každej požiadavke. Nie je potrebné ani udržiavať informáciu o aktuálnom prihlásení. Stačí, že prijatý token je platný. Nevznikajú ani problémy pri škálovani, keďže žiadne dáta o relácii nie sú ukladané.

Kapitola 5

Analýza požiadavok

Platobná inštitúcia Roger sa zaoberá skracovaním splatnosti faktúr. Spolupracuje s menšími aj veľkými spoločnosťami, ktoré jej faktúry zasielajú do aukcií. V aukciách investori súťažia, ktorý z nich poskytne dodávateľovi najnižší úrok za splatenie faktúry predčasne, aby tak dodávateľ mohol peniaze využívať skôr, ako ich zaplatí odberateľ. Už nejaký čas firma Roger taktiež disponuje PSD2 licenciou, ktorú jej udelila Česká národná banka. Momentálne má vo vývoji aplikáciu na jej využitie pri online platbách.

Aplikácia slúži ako alternatíva k doteraz ponúkaným spôsobom online platieb. Ako popisujem v kapitole 3, obchodníci stále častejším používaním kariet prichádzajú o časť zisku. PSD2 platby sú taktiež jednoduché a vo väčšine bánk bezplatné, preto majú potenciál platbám kartou v tomto smere konkurovať. Taktiež ponúka okamžité *peer-to-peer* platby.

Aplikácia umožňuje vykonávať platby napríklad v podporovaných internetových obchodoch, kde je ponúkaná ako jeden z možných platobných spôsobov. Používateľ si aplikáciu stiahne a pri prvej platbe s ňou spáruje svoj zákaznícky účet vybraného obchodu. Platby sú vykonávané z používateľovho bankového účtu, ktorý si do aplikácie pridá a povolí jeho používanie aplikáciou tretej strany. Pred každou platbou si potom používateľ vyberie, z ktorého pridaného bankového účtu chce platbu odoslať a následne platbu potvrdí na strane banky.

Mojou úlohou bolo využiť, že firmy, ktoré už faktúry zasielajú na preplatenie, môžu faktúry zasielať priamo zákazníkom. Pri prepojení odoslaných faktúr s novou platobnou aplikáciou, by tak mohli jej používatelia od vybraných firiem dostávať faktúry elektronicky na jedno miesto a tiež ich v priamo aplikácii zaplatiť. To využíva možnosti oboch produktov firmy.

5.1 Základné požiadavky

Základné požiadavky boli rozdelené podľa dvoch druhov používateľov – dodávateľa a odberateľa. Systém umožní dodávateľovi odoslať faktúru cez API a následne ju dokáže spárovať s účtom odberateľa. Faktúra bude spárovaná s účtom odberateľa podľa údajov, ktoré dodávateľ pozná a odošle ich spolu s ňou.

Ďalej systém bude obsahovať API, ktoré odberateľovi umožní zobrazovať si prijaté faktúry. Vytvorené API pre odberateľa bude môcť byť zahrnuté do vyvíjanej aplikácie firmy Roger. Súčasťou prijatia faktúry bude možnosť automatického generovania PSD2 platby, v už existujúcej časti aplikácie. Odberateľovi v nej bude vygenerovaná žiadosť o platbu za faktúru rovnako, ako keby napríklad nakupoval v eshope a bude ju môcť ihneď po prijatí zaplatiť. Požadovaná spolupráca systémov sa nachádza na obrázku 5.1.

Obr. 5.1: Požadovaná spolupráca rozšírenia s existujúcou aplikáciou

5.2 Podobné systémy

Rozšírenie aplikácie má slúžiť na odosielanie, prijímanie a platenie faktúr. Sledoval som existujúce riešenia na trhu, ktoré by mohli byť pre toto rozšírenie konkurenciou.

5.2.1 Fakturačné systémy

Funkcie na odosielanie, evidenciu a platenie faktúr spĺňajú existujúce online fakturačné systémy. Medzi najznámejšie patria Fakturoid, iDoklad, Vyfakturuj, SuperFaktura či mPohoda. Z pohľadu mojej práce tieto systémy ponúkajú podobnú funkcionalitu. Pre účely porovnania som si vybral Fakturoid[1].

[1] Fakturoid: https://www.fakturoid.cz/

Fakturoid

Fakturoid je online fakturačný systém, ktorý slúži na vydávanie faktúr, evidenciu nákladov a udržiavanie prehľadu. Údaje prijímateľa je možné automaticky vyplniť zadaním jeho IČO a obsahuje tiež možnosť pridávať kontakty. Odosielateľ si môže vybrať šablónu faktúry, nastaviť vlastné logo alebo upraviť farby. Ponúka aj možnosť nastaviť odosielanie faktúr pravidelne. Dodávateľovi po odoslaní faktúry systém zobrazuje, či si odberateľ faktúru pozrel. Nemôže tak tvrdiť, že ju nevidel.

Vytvorená faktúra môže byť zaslaná emailom, alebo klientovi môže byť zaslaný odkaz na jej webovú verziu. Súčasťou webového zobrazenia je naviac možnosť ihneď faktúru zaplatiť. Fakturoid na zaplatenie ponúka platobnú bránu GoPay[2], PayPal[3] alebo bankový prevod. GoPay zabezpečuje platbu kartou a iné ním ponúkané možnosti. Pri platbe bankovým prevodom je možné využiť QR kód.

Ďalšou funkciou je možnosť párovania faktúr s platbami na vybraný bankový účet. Podľa prijatých platieb označuje, ktoré faktúry boli zaplatené. Párovanie funguje s vybranými bankami[4]. Momentálne ich plne podporuje 7, ďalšie 4 podporuje čiastočne.

Okrem vystavovania faktúr systém umožňuje aj evidenciu nákladov. Náklady sú zadávané ručne a systém preto umožňuje pridať používateľa, ktorý bude mať prístup len k ich zadávaniu a od zvyšku systému bude odizolovaný. Párovanie platieb je možné aj pri zadaných nákladoch. Používateľ tak vie evidovať čo už zaplatil. Ďalší prístup môže udeliť svojmu účtovníkovi. Ten si môže dokumenty prezerať a exportovať do svojho účtovného systému. Zaúčtované doklady môže zamknúť alebo k nim pridať poznámku, ak je niečo potrebné upraviť.

Faktúry je možné označovať štítkami, pre priradenie kategórie do ktorej patria. Tie sú využité v časti systému ponúkajúcej štatistiky. V tej používateľ môže sledovať svoje náklady, výnosy, výsledky a *cashflow*. Tiež si môže vytvoriť plán a sledovať jeho priebežné plnenie.

Porovnanie s účelom fakturačných systémov

Požadované rozšírenie aplikácie má byť narozdiel od fakturačných systémov určené hlavne pre prijímateľov faktúr, ktorí nie sú podnikateľmi a faktúry platia za bežné nákupy, služby či iné poplatky. Faktúry majú byť do systému zasielané priamo dodávateľom a následne byť automaticky priraďované jeho používateľom podľa zadaného identifikátora. To fakturačné systémy neumožňujú. Naviac má rozšírenie priamo spolupracovať s platobnou aplikáciou, preto firma Roger ako poskytovateľ nebude musieť platiť poplatky za externú platobnú bránu, ani poplatky za platby kartou, ak by neboli v jej cene. Prepojenie s platobnou aplikáciou bude umožňovať prijímateľovi faktúry sledovať, či už faktúru zaplatil bez potreby párovania platieb v banke.

Fakturačné systémy sa sústreďujú na pohodlné vystavovanie a odosielanie faktúr zo strany dodávateľa. Snažia sa mu čo najviac zjednodušiť ich vyplňovanie a udržiavanie prehľadu. Napríklad, či už mu boli peniaze doručené na účet, alebo či si odberateľ vystavenú faktúru otvoril. Ponúkajú prepracované možnosti spravovania uložených faktúr na účely účtovníctva a sledovania štatistík. Týmto smerom sa sa moja práca nebude zameriavať. Jej účelom je dodávateľom poskytnúť len API, cez ktoré faktúru odošlú.

[2] GoPay: https://www.gopay.com/cs/.

[3] PayPal: https://www.paypal.com/cz/home.

[4] Aktuálny zoznam Fakturoidom podporovaných bánk: https://www.fakturoid.cz/podpora/automatizace/parovani-plateb-s-bankou.

Podľa tohto prieskumu nie sú fakturačné systémy a požadované rozšírenie konkurenciou, skôr sa navzájom dopĺňajú. Faktúra by sa napríklad z fakturačného systému mohla dať odoslať do rozšírenia aplikácie firmy Roger cez vytvorené API, ako jedna z ponúkaných možností odoslania.

5.2.2 Interné spôsoby prijímania a úhrady faktúr

Dodávatelia, na ktorých sa plánované rozšírenie zameriava, majú už určité zaužívané spôsoby, ako faktúry zasielajú. Elektronicky sú zasielané najčastejšie emailom, alebo do zákazníckej webovej či mobilnej aplikácie. Je na používateľovi, aby si vybral, aký spôsob platby zvolí.

Pri opakovaných platbách si klient môže zvoliť trvalý príkaz alebo platbu inkasom. Ak firma od klienta požaduje rovnakú sumu pravidelne, je pre neho najjednoduchšie nastaviť si trvalý príkaz. Pri jeho použití sú platby odosielané z jedného prednastaveného účtu a klient musí zabezpečiť, aby mal v dohodnutom čase na vybranom účte dostatok prostriedkov. Ak sa suma za každé platobné obdobie mení, môže použiť inkaso. Pri tom musí taktiež určiť, z akého účtu bude suma strhávaná. Ak má klient na maximálnu výšku inkasa nastavený limit, musí okrem dostatku prostriedkov zabezpečiť, aby bol dostatočne vysoký. Nevýhodou môže byť, že klient nad sťahovanou čiastkou nemá kontrolu a ak by pri jej vypočítavaní nastala chyba, nemusí si ju hneď všimnúť. Taktiež môže prehliadnuť, ak sa platba nevykoná zo spomínaného nedostatku prostriedkov alebo nízkeho limitu. Preto ak platba nie je vykonávaná často, klient môže zvoliť aj manuálne zasielanie prostriedkov na účet. Ak platba nie je pravidelná, iné možnosti už ani nemá.

Ak klientovi faktúry chodia napríklad emailom, môže o nich medzi ostatnými správami stratiť prehľad. Preto firmy pridávajú prehľad faktúr do svojich aplikácií. Klient v nich všetky faktúry nájde na jednom mieste, vidí ich stav a vie tak ktoré z nich už zaplatil a ktoré ešte na zaplatenie čakajú. Tieto funkcie sú užitočné aj pre klientov, ktorí majú nastavené automatické platby a chodia si ich stav alebo zaplatenú čiastku do aplikácie kontrolovať.

Pri jednorázových platbách sa ponuky dodávateľov líšia. Niektorí ponúkajú len platobné údaje, kam prostriedky zaslať, u iných je súčasťou faktúry QR kód pre jej zjednodušenie. Niektoré firmy však ponúkajú aj možnosť platby v ich aplikácii. Klient si môže okrem QR kódu vybrať aj platbu kartou alebo služby Google Pay a Apple Pay spomínané v kapitole 3. Narozdiel od opakovaných platieb si klient môže pri každej platbe vybrať, z akého účtu platbu odošle a má nad ich platením plnú kontrolu.

Porovnanie s internými aplikáciami spoločností

Práve zákaznícke systémy, ktoré ponúkajú prehľad faktúr a možnosť ich úhrady sa čiastočne prekrývajú s mojou prácou. Majú ich napríklad firmy Vodafone, O2, E.ON alebo ČEZ. Firmy ako tieto, by taktiež mali mať možnosť zasielať faktúry do požadovaného rozšírenia. Jeho výhodou pre používateľa by malo byť zasielanie faktúr do jednej aplikácie, namiesto používania aplikácie každej firmy samostatne. Na druhej strane, moje rozšírenie nemôže poskytnúť všetky možnosti ako systém vytvorený na mieru pre danú firmu. Mohlo by pre používateľov týchto aplikácii slúžiť ako nástroj na udržiavanie celkového prehľadu vo faktúrach na jednom mieste a ich rýchle platenie.

Najzaujímavejšie je použitie pri dodávateľoch, ktorí zákazníckym systémom nedisponujú. Pre tých je výhodou, že ich zákazníci budú mať aj inú možnosť platby. Zákazníci

získajú aplikáciu, v ktorej budú mať prehľad vo svojich faktúrach a možnosť ich pohodlne zaplatiť.

5.3 Profil používateľa

Používateľ aplikácie je bežná osoba, ktorá prijíma faktúry za tovar, služby alebo rôzne poplatky. Záujem o jej používanie by mal mať vtedy, ak dostáva faktúry, na ktoré nemôže alebo nechce použiť trvalý príkaz či inkaso. Chce napríklad zaplatiť faktúru od operátora za nejakú nepravidelnú službu, uhradiť nedoplatok za energie, ktorý má za každé obdobie inú výšku alebo uhradiť akúkoľvek jednorázovú faktúru. Na používanie aplikácie stačia základné skúsenosti s ovládaním internetového prehliadača a smartfónu. Základom je mať účet v banke a následne ho pridať do aplikácie. Pre udelenie súhlasu s pridaním jeho účtu do aplikácie firmy Roger, sa bude používateľ potrebovať prihlásiť do internetového bankovníctva. Po každom zaplatení faktúry v aplikácii, bude potrebovať potvrdiť transakciu na strane banky (napr. SMS, PIN, či biometria v bankovej aplikácii).

Používateľ na opačnej strane aplikácie je dodávateľ. Ten má do systému faktúry zadávať, pretože chce svojim zákazníkom poskytnúť ďalšiu alternatívu k doteraz ponúkaným spôsobom platby. Platby zasiela zo svojho systému cez API. Potrebuje programátora, ktorý jeho systém na API napojí.

5.4 Pridaná funkcionalita na základe analýzy

Z predchádzajúcich poznatkov vyplynulo, že aplikácia by mohla byť zaujímavá pre používateľa, ktorému sú často vystavované jednorázové faktúry za tovar alebo služby, ktoré si objednáva u dodávateľov. Môže to byť napríklad živnostník či malá firma. Aplikácia mu umožní udržiavať históriu prijatých faktúr a pohodlne ich platiť. Tento druh používateľa môže chcieť faktúry aj vystavovať za tovar alebo služby, ktoré ponúka. Preto by bolo vhodné do aplikácie zahrnúť možnosť faktúru vystaviť. Vystavovanie faktúr by bolo narozdiel od fakturačných systémov minimalistické a ponúkalo len základné rozhranie, tak aby spĺňalo účel. Faktúry by boli vystavované odberateľom do aplikácie firmy Roger, rovnako ako od iných dodávateľov. Poplatky za požívanie spomínaných fakturačných systémov, môžu tvoriť pre začínajúceho podnikateľa zbytočné výdavky naviac. Preto by mohol túto funkcionalitu oceniť.

5.5 Podrobné požiadavky

K základným požiadavkám boli firmou Roger pridané ďalšie požiadavky, upresňujúce ako má vyzerať výsledok mojej práce:

1. Systém umožní dodávateľovi zistiť, akí jeho odberatelia aplikáciu využívajú. Bude tak vedieť komu cez ňu faktúru môže zaslať.

2. Po odoslaní faktúry bude systém schopný obom stranám zobraziť stav PSD2 platby.

3. Ak už bola faktúra zaplatená, systém neumožní vytvorenie novej PSD2 platby.

4. Ak vo faktúre nastala chyba, dodávateľ môže faktúru stornovať, platba bude zrušená a nebude ju možné zaplatiť.

5. Systém umožní odberateľovi zobraziť históriu faktúr.
6. Odberateľ si bude môcť byť istý, že faktúra mu prišla od overeného dodávateľa.
7. Dodávateľ má možnosť v systéme potvrdiť, že mu bola platba doručená. Odberateľ toto potvrdenie uvidí.
8. Odberateľ môže faktúru odmietnuť zaplatiť.
9. Odberateľ sa môže odhlásiť z prijímania faktúr do aplikácie a dodávateľ mu ich bude zasielať opäť starou cestou.
10. Ak odberateľ nechce alebo nemôže zaplatiť metódou PSD2, má možnosť využiť inú metódu.
11. Odberateľ si môže vybrať, či si chce nechať automaticky generovať platbu pre každú prijatú faktúru.

Tieto a základné požiadavky budú ďalej zohľadnené pri návrhu.

Kapitola 6

Návrh

Podľa vykonanej analýzy požiadavok bol vypracovaný návrh. Táto kapitola popisuje, ako boli postupne navrhnuté jednotlivé časti systému.

6.1 Ukladanie dát

Zasielané faktúry a dáta o používateľoch je potrebné ukladať. Systém sa skladá z viacerých druhov používateľov, ich kontaktov pre spárovanie s faktúrami a samotných faktúr. Tomu odpovedajú aj navrhnuté entity v dátovom modeli 6.1.

6.1.1 Používatelia

Podľa analýzy budú prvou skupinou používateľov živnostníci alebo firmy, ktorí budú potrebovať na faktúre uvádzať aj svoje štátom pridelené identifikátory IČO, prípadne VAT ID, pridelené štátom[1]. Táto skupina bude najčastejšie faktúry aj odosielať. Druhou skupinou budú používatelia, ktorí nepodnikajú, popísaní ako bežní používatelia v sekcii 5.3. Tí rozširujúce informácie potrebovať nebudú a faktúry budú hlavne prijímať. Preto sú v návrhu používatelia rozdelení do dvoch entít, kde je podnikajúci používateľ špecializáciou obyčajného používateľa.

Používateľ potrebuje pre vystavenie faktúry názov a adresné údaje. Tie musia byť jej súčasťou. Aby sa mohol prihlásiť, potrebuje sa voči systému autentifikovať 4.4.1. Na tento účel v systéme slúži email a heslo. Tie sú ďalšími atribútmi používateľa. Ďalšie dva atribúty slúžia na uloženie používateľových nastavení, pre vyhovenie požiadavkám 9 a 11 zo sekcie 5.5. Prvý atribút nastavuje, či si chce používateľ nechať zasielať faktúry do aplikácie. Druhý, či si chce pri prijatí novej faktúry nechať automaticky vygenerovať platbu.

Rozšírený používateľ má naviac spomínané atribúty IČO, VAT ID a kód štátu, pre možné rozšírenie okruhu používateľov aj na okolité štáty.

K používateľom patria aj zamestnanec a administrátor. Keďže požiadavkou 6 je, aby faktúry prichádzali od overených dodávateľov, je potrebné overiť ich používateľské údaje. Emailová adresa a telefónne číslo môžu byť overené zaslaním potvrdzovacej emailovej alebo SMS správy. Vlastníctvo identifikátorov IČO a VAT ID nie je možné takýmto spôsobom overiť. Keďže sú verejne dostupné, mohol by sa za danú firmu vydávať ktokoľvek. Preto údaje IČO a VAT ID používateľovi priraďuje zamestnanec firmy Roger, po predchádzajúcom overení jeho totožnosti. Jeho úlohou je tak overovať, pridávať a spravovať podnikajú-

[1] Podrobnejšie v kapitole 2

cich používateľov. Administrátor naviac pridáva zamestnancov. Rola každého používateľa je označená atribútom rola v entite používateľa.

6.1.2 Kontakty

Pre spárovanie používateľa s faktúrou, bude systém primárne využívať jeho email alebo telefónne číslo. Keďže používateľ môže mať u rôznych dodávateľov odlišné kontaktné informácie, systém obsahuje jednu entitu pre emailové adresy a druhú pre telefónne čísla. Požívateľ ich tak môže mať priradených viacero. Obe entity obsahujú daný kontakt a ako cudzí kľúč id používateľa.

6.1.3 Faktúra

Ďalšiu entitu tvorí faktúra, tá bude obsahovať najviac atribútov. Základ tvoria údaje, ako číslo faktúry, cena, mena, dátum vytvorenia, dátum splatnosti atď. Každá faktúra tiež obsahuje časový odtlačok, kedy bola do systému pridaná. Pre získavanie údajov o dodávateľovi a odberateľovi obsahuje dva cudzie kľúče s ich primárnymi kľúčmi. K odberateľovi naviac obsahuje jeho email a telefónne číslo, aby bolo možné vo faktúre zobraziť, podľa ktorého z kontaktov dodávateľ odberateľa eviduje. Ďalej obsahuje platobné údaje. Hlavné je číslo účtu, na ktoré má byť faktúra zaplatená. Môže byť pridaný aj variabilný, konštantný a špecifický symbol a tiež poznámka k platbe. Nasledujú údaje o vytvorení transakcie. Je to id transakcie priradenej k faktúre a jej aktuálny stav. Posledný atribút slúži na označenie faktúry ako vymazanej. Taká faktúra sa viac nebude zobrazovať v systéme.

Obr. 6.1: ER diagram zobrazujúci model databázy

6.2 Operácie s dátami

Na prácu s dátami zo strany klienta, systém využíva REST API 4.3.3. Jeho endpointy sú v prvom kroku rozdelené podľa entít, napr. `/users, /emails, /invoices`. Prefixom každej cesty je `/invoicepay/v1`. `/invoicepay` odlišuje endpointy rozšírenia od zvyšku aplikácie. `/v1` označuje prvú verziu API.

6.2.1 Používateľ

Prvý endpoint `/users` metódou POST umožňuje registráciu základného používateľa. Dáta pre jeho registráciu sú odosielané v tele požiadavky. Pre prístup k ostatným endpointom už musí byť používateľ prihlásený a zasielať s každou požiadavkou JWT token. Endpoint `/{user_id}` vyžadujúci id používateľa slúži na získanie a úpravu jeho dát. Pri získavaní detailov používateľa s podnikateľským účtom sú v odpovedi odosielané aj jeho dodatočné údaje. Použitím `/self` je id používateľa získané z pridaného JWT tokenu. Endpoint `/self/pairing_code` umožňuje používateľovi získať párovací kód do existujúcej platobnej aplikácie firmy Roger. Ten v nej zadá, aby spároval svoj účet v platobnej aplikácii s účtom v rozšírení a mohli mu do nej byť zasielané požiadavky na platbu. `/user/status` slúži pre splnenie požiadavky 1 zo sekcie 5.5 na zistenie dodávateľom, či jeho odberateľ systém používa. Pre zistenie dodávateľ zadá do parametrov požiadavky jeden alebo viacero identifikátorov hľadaného používateľa, ktorými disponuje. Endpoint `/user/update_to_business` slúži na pridanie identifikátorov IČO a VAT ID obyčajnému používateľovi, a tým aj prevedenie jeho účtu na účet podnikajúceho používateľa. Vybraný používateľ je identifikovaný emailovou adresou, pridanou do tela požiadavky. Tento prevod účtu môže vykonať iba zamestnanec alebo administrátor.

POST	`/invoicepay/v1/users`
GET	`/invoicepay/v1/users/{user_id}`
PUT	`/invoicepay/v1/users/{user_id}`
GET	`/invoicepay/v1/users/self`
PUT	`/invoicepay/v1/users/self`
GET	`/invoicepay/v1/users/self/pairing_code`
GET	`/invoicepay/v1/users/user/status`
PUT	`/invoicepay/v1/users/user/update_to_business`

6.2.2 Podnikajúci používateľ

Endpoint `/business_user` umožňuje metódou POST novú registráciu podnikateľského účtu. Metódy PUT a GET endpointu `/business_users/user_by_email/{main_email}` umožňujú získanie a úpravu dát používateľa identifikovaného pomocou emailu. Metóda PUT endpointu `/business_users/{user_id}` umožňuje úpravu dát používateľa jeho identifikáciou cez id. Všetky tieto operácie môže vykonávať iba zamestnanec alebo administrátor.

POST	`/invoicepay/v1/business_users`
PUT	`/invoicepay/v1/business_users/{user_id}`
GET	`/invoicepay/v1/business_users/user_by_email/{main_email}`
PUT	`/invoicepay/v1/business_users/user_by_email/{main_email}`

6.2.3 Zamestnanec

Metóda POST slúži na vytvorenie zamestnaneckého účtu. Ak zamestnanec napríklad prestane pre firmu pracovať, je možné jeho zamestnanecký účet vymazať metódou DELETE. Tieto operácie môže vykonávať iba administrátor. Operácie na správa zamestnaneckých účtov sú v tejto práci jedinými, ktoré ním musia byť vykonávané administrátorom. Administrátorsky účet môže vytvoriť iba osoba s prístupom k databáze manuálne alebo pomocou skriptu.

POST	`/invoicepay/v1/internal_users`
DELETE	`/invoicepay/v1/internal_users/{user_id}`

6.2.4 Kontakty

Endpointy `/emails` slúžia na pridávanie a odoberanie emailových adries z účtu používateľa.

POST	`/invoicepay/v1/emails`
DELETE	`/invoicepay/v1/emails/{email_address}`

Endpointy `/phones` slúžia na pridávanie a odoberanie telefónnych čísel.

POST	`/invoicepay/v1/phones`
DELETE	`/invoicepay/v1/phones/{phone_number}`

Každý používateľ si kontakty spravuje sám. Jeho id je získané z JWT tokenu odoslaného spolu s požiadavkou.

6.2.5 Faktúry

Práca s faktúrami je rozdelená do viacerých častí. Prvou sú operácie so zoznamom faktúr. Tie umožňujú pridanie novej faktúry a získanie zoznamu faktúr. Nasledujú operácie s jednou faktúrou, a s k nej pridaným PDF súborom. Poslednou časťou sú operácie na správu platieb.

Zoznam faktúr

Metóda GET pre získanie zoznamu obsahuje viacero parametrov určených pre filtrovanie faktúr, stránkovanie, či výber zobrazenia prijatých alebo odoslaných pre daného používateľa. Metóda POST rovnakého endpointu slúži na pridanie faktúry do zoznamu. V tele požiadavky sa nachádzajú údaje faktúry a identifikátor, prípadne identifikátory používateľa pre jeho spárovanie s faktúrou. Identifikátormi sú spomínané kontakty email a telefónne číslo alebo identifikátory podnikajúceho používateľa IČO a VAT ID. Pri párovaní podľa IČO je potrebné pridať aj kód štátu, keďže nemusí byť unikátne medzi rôznymi štátmi.

GET	`/invoicepay/v1/invoices`
POST	`/invoicepay/v1/invoices`

Faktúra

Endpoint slúži na prácu s konkrétnou faktúrou podľa jej identifikátora. Metóda GET slúži na získanie detailu faktúry, spolu s podrobnými údajmi dodávateľa a odberateľa. Metódu môže použiť iba dodávateľ alebo odberateľ pre danú faktúru. DELETE nastaví faktúru ako vymazanú. Túto metódu môže použiť iba používateľ, ktorý faktúru vytvoril, kým faktúra nie je zaplatená.

GET	`/invoicepay/v1/invoices/{invoice_id}`
DELETE	`/invoicepay/v1/invoices/{invoice_id}`

PDF

Metóda GET slúži na získanie PDF dokumentu podľa čísla faktúry. POST slúži na jeho pridanie. Dokumenty sú ukladané ako súbory na serveri a preto nie sú zahrnuté v modeli databázy. Pri prístupe k dokumentu prebieha rovnaká kontrola ako pri prístupe k faktúre.

GET	`/invoicepay/v1/invoices/{invoice_id}/pdf`
POST	`/invoicepay/v1/invoices/{invoice_id}/pdf`

Platby

Endpointy obsahujúce `/transaction` sú určené na správu žiadostí o platbu a ich stavu. Metóda POST endpointu `/transaction` slúži na vygenerovanie žiadosti o platbu pre danú faktúru. Odberateľ ju môže využiť, ak má vypnuté automatické generovanie požiadavok po prijatí novej faktúry, alebo požiadavku zamietol v platobnej aplikácii (požiadavka 8). Metóda DELETE slúži na zrušenie možnosti platby dodávateľom, popísanej v požiadavke 4. Dodávateľ môže túto metódu použiť, iba kým faktúra nie je zaplatená. Táto faktúra aj naďalej ostane dostupná v systéme. Metóda GET pri endpointe `/invoice/transaction/status` získava od platobnej časti aplikácie aktuálny stav platby, podľa id žiadosti zadaného ako parameter požiadavky. Zistený stav spracuje, konvertuje a uloží do databázy k faktúre, ku ktorej bola žiadosť o platbu vygenerovaná. Pre splnenie požiadavky 7 na potvrdenie prijatia platby dodávateľom slúži metóda PUT endpointu `/transaction/finish`, tá nastaví platbu ako dokončenú.

POST	`/invoicepay/v1/invoices/{invoice_id}/transaction`
DELETE	`/invoicepay/v1/invoices/{invoice_id}/transaction`
GET	`/invoicepay/v1/invoices/invoice/transaction/status`
PUT	`/invoicepay/v1/invoices/{invoice_id}/transaction/finish`

6.2.6 Prihlásenie

Endpoint `/login` slúži na prihlásenie používateľa. V tele požiadavky je odoslaná emailová adresa a heslo používateľa. Ak je autentifikácia úspešná, v odpovedi je používateľovi zaslaný vygenerovaný JWT token.

POST	`/invoicepay/v1/login`

6.3 Návrh používateľského rozhrania

Zadaním mojej práce bolo implementovať API. Jeho používateľské rozhranie má byť implementované do mobilnej aplikácie externou firmou. Po konzultácii vo firme Roger sme sa zhodli, že vytvorenie testovacieho webového rozhrania by mohlo byť vhodným rozšírením mojej práce. Prvým dôvodom bolo, že rozhranie umožní lepšie znázorniť a otestovať funkcionalitu API. Webová aplikácia môže byť spustená na rôznych druhoch zariadení a umožňuje tak vyskúšať rôzne prípady použitia. Pri testovaní na mobilných zariadeniach môže responzívne webové rozhranie dočasne nahrádzať mobilnú aplikáciu a otestovať takéto použitie ešte pred investíciou do jej implementácie. Firma Roger však zamýšľa vytvoriť webové rozhranie aj k už existujúcej mobilnej aplikácii a tak môže byť moje testovacie rozhranie po úprave jeho súčasťou. Tiež môže pomôcť zistiť, či by klienti mali o webovú verziu záujem. Keďže firma Roger pripravuje aktualizáciu dizajnu svojho webu a mobilných aplikácií, nevenoval som sa do detailov štýlovaniu prvkov rozhrania a voľbe farieb.

6.3.1 Používateľská časť

Návrh rozhrania sa skladá z hlavnej časti s prehľadom faktúr, detailu faktúry, časti pre jej vytvorenie a používateľskej časti s detailami účtu a nastaveniami. Model prehľadu faktúr vo webovej verzii sa nachádza na obrázku 6.2. Navigácia je rozdelená do vrchného a bočného navigačného panelu. Vo vrchnom paneli je prístup k správe účtu prihláseného používateľa a možnosť odhlásiť sa. V bočnej časti je na výber prechod medzi prehľadom prijatých a odoslaných faktúr a obrazovkou na vytvorenie faktúry. V mobilnej verzii je bočný panel skrytý pod tlačidlom menu vo vrchnom paneli. Jej model zobrazujúci prehľad faktúr sa nachádza na obrázku 6.3.

Obr. 6.2: Používateľská časť rozhrania

Prehľad faktúr obsahuje zoznam náhľadov faktúr podľa požiadavky 5 a možnosť vyhľadávania v zozname podľa zvoleného filtra. Detaily faktúry zobrazené v zozname prijatých a odoslaných faktúr sa líšia. Prijatá faktúra v náhľade obsahuje názov odosielateľa, dátum splatnosti a požadovanú sumu. Odoslaná obsahuje názov príjemcu, dátum vystavenia a taktiež sumu. Po zvolení faktúry je zobrazená obrazovka s jej podrobnými detailami. Obrazovka pre zobrazenie faktúry obsahuje okrem jej detailov a detailov oboch odosielateľov aj detaily o platbe a jej stav (požiadavka 2). Po zobrazení všetkých platobných údajov môže odberateľ faktúru zaplatiť aj prevodom na účet manuálne, pre splnenie požiadavky 10. Taktiež je možné stav faktúry ovládať podľa zadaných požiadavok 4, 7, 11, 3 a 8. Dodávateľ môže zrušiť možnosť platby nezaplatenej faktúry a potvrdiť doručenie platby na jeho bankový účet. Odberateľ si môže vygenerovať požiadavku na platbu do mobilnej aplikácie, ak má ich automatické generovanie vypnuté v nastaveniach, alebo požiadavku v aplikácii zamietol a chce si ju nechať zaslať znova.

Obr. 6.3: Používateľská časť rozhrania na mobilnom zariadení

Časť pre vytvorenie faktúry obsahuje formulár na vyplnenie jej detailov, informácií pre prijatie platby a výber identifikátoru odberateľa.

V časti používateľského účtu sa nachádzajú podrobné údaje používateľa a vybrané nastavenia. Z nej je možný prechod na obrazovku pre úpravu údajov a pridávanie alebo odoberanie kontaktov.

6.3.2 Administrátorská časť

Administrátorské rozhranie slúži zamestnancovi firmy Roger na správu podnikateľských účtov. Keďže tie, ako som spomínal v sekcii 6.1.1 obsahujú údaje, ktoré je potrebné dodatočne overiť.

Zamestnanec môže vytvoriť nový rozšírený používateľský účet s týmito údajmi, ak používateľ ešte nie je zaregistrovaný, alebo rozšíriť bežný účet o tieto údaje a tým ho konvertovať na rozšírený. Taktiež môže v prípade potreby tieto údaje upraviť.

Administrátorské rozhranie má narozdiel od používateľského na bočnom navigačnom paneli na výber rozšírenie z bežného účtu, aktualizáciu rozširujúcich údajov na účte a vytvorenie nového rozšíreného účtu. Tomu zodpovedajú aj tri možné obrazovky rozhrania. Prístup k prihlásenému účtu a jeho nastaveniam z vrchného panelu ostáva nezmenený. Aj keď tie primárne zamestnanec nebude využívať. Model jeho webovej verzie sa nachádza na obrázku 6.4. V mobilnej verzii je bočné menu presunuté rovnako do vrchného panelu ako v používateľskej časti.

Pre pridávanie a úpravu údajov používateľa zamestnanec zadá jeho emailovú adresu a následne sú mu zobrazené polia na vyplnenie. Pri úprave sú v poliach zobrazené aktuálne uložené údaje.

Obr. 6.4: Administrátorská časť rozhrania pre zmenu typu účtu

Kapitola 7

Implementácia

V tejto kapitole je popísané, ako bol navrhnutý systém implementovaný. Postupne je preberaná jeho architektúra, použité technológie a zaujímavé implementačné detaily.

7.1 Architektúra

Systém je webová aplikácia používajúca trojvrstvovú architektúru spomínanú v sekcii 4.1. Základom systému je aplikačný server *NGINX* s nadstavbou *uWSGI*, na ktorom beží aplikačná logika systému napísaná v jazyku *Python*. Ďalšie časti systému tvoria webový klient a relačná databáza. Na komunikáciu s klientom server využíva REST API (v sekcii 4.3.3), napísané vo frameworku *Falcon*[1]. Dáta zasielané cez REST API, sú medzi klientom a serverom prenášané vo formáte JSON (4.3.2). S databázou server komunikuje prostredníctvom knižnice *SQLAlchemy*[2]. Architektúra systému so zameraním na server sa nachádza na obrázku 7.1.

7.2 Databáza

Databáza využíva na správu dát systém *PostgreSQL*[3]. PostgreSQL je open source objektovo-relačný databázový systém. Narozdiel od iných relačných systémov umožňuje budovať komplexnejšie dátové štruktúry a používať napríklad dedičnosť. Pre prácu s databázou z aplikačného servera využívam knižnicu SQLAlchemy. Tá umožňuje v jazyku Python pracovať s viacerými známymi databázovými systémami vrátane PostgreSQL. SQLAlchemy existuje vo verzii *core* a verzii *ORM*. Verzia ORM umožňuje mapovanie relačnej databázy na objekty, pridáva viac abstrakcie nad jazykom SQL a uľahčuje tak jeho používanie. Core naopak dovoľuje vytvárať špecifickejšie dotazy, prináša väčšiu kontrolu nad databázou a umožňuje tak lepšiu optimalizáciu. Z dôvodu potreby rýchleho vývoja prvotného produktu a nie veľmi zložitého databázového modelu bola pre existujúcu platobnú aplikáciu zvolená verzia ORM. Preto som túto verziu použil aj v mojom rozšírení.

Schému databázy som pomocou knižnice SQLAlchemy popísal v priečinku *database*. Pre každú tabuľku je v nej vytvorená trieda obsahujúca jej atribúty a integritné obmedzenia, tiež ešte obsahuje definíciu triedy na inicializáciu databázy. S databázou potom aplikačný server komunikuje pomocou tried určených na tento účel. Tieto triedy sú rozdelené podľa

[1] Falcon: https://falcon.readthedocs.io/en/stable/
[2] SQLAlchemy: https://docs.sqlalchemy.org/en/14/
[3] PostgreSQL https://www.postgresql.org/

existujúcich entít a uložené v priečinku *storage*. Pre jednoduchší prístup k ich metódam je použitý návrhový vzor továreň (*factory*). Objekt továrne je potom používaný na miestach, kde sú vykonávané operácie s dátami.

7.3 Server

Na implementáciu operácii nad dátami popísanými v návrhu 6.2 je použitý framework Falcon. Falcon je minimalistický framework na tvorbu REST API a mikroslužieb. Je zameraný na vysokú spoľahlivosť a rýchlosť.

Podľa syntaxe frameworku som v priečinku *endpoints* vytvoril súbor na inicializáciu API. Na začiatku je v ňom inicializovaná databáza použitím triedy z predchádzajúcej sekcie. Nasleduje inicializácia triedy pre autentifikáciu. Tá sa stará o kontrolu JWT tokenu pri každej požiadavke a extrahovanie dát o používateľovi, ak bol token pridaný. Treťou inicializovanou triedou je trieda, určená na komunikáciu s API platobnej aplikácie. Triedu využívajú endpointy pracujúce s platbami popísané v sekcii 6.2.5. Inicializácia týchto tried ešte pred vytvorením API umožňuje zmenu inštancie databázy, spôsobu autentifikácie a implementácie komunikácie s externým API. To je výhodné napríklad pri testovaní.

Ďalej nasleduje spustenie hlavnej funkcie `create_api`, ktorá tieto tri vytvorené objekty využíva. Hlavná funkcia pre vytvorenie API definuje zoznam endpointov. Pre každý z nich inicializuje triedu vybraného zdroja, ktorý bude pri prijatí požiadavky na daný endpoint použitý. Ako bude interpretovaná konkrétna HTTP metóda danej požiadavky, definujú metódy zvoleného zdroja. Zdroj môže obsahovať napríklad metódy `on_get` a `on_post`, alebo iné pre ostatné HTTP metódy. V každej implementovanej metóde je definované, ako požiadavku spracovať a akú odpoveď zaslať.

Obr. 7.1: Architektúra systému so zameraním na závislosti medzi rôznymi časťami serveru. Šípka vždy smeruje od časti, na ktorej časť na druhom konci závisí. Čiarkované časti sú inicializované pri spomínanom vytváraní API cez funkciu `create_api` a sú ďalej odovzdávané pomocou parametrov až na miesto ich použitia.

Kód na serveri je rozdelený do viacerých častí. Triedy zdrojov sú ukladané v priečinku *resources* a v nich sú používané obslužné funkcie z priečinka `use_cases`. Framework Falcon automaticky konvertuje prijaté a odosielané dáta z formátu JSON na slovník jazyka Python (*Python Dictionary*) a naopak. Na prevod medzi slovníkmi jazyka Python a objektami SQLAlchemy sú využívané funkcie z priečinka *adapters*. Na operácie s databázou sú používané metódy z už spomínaného priečinka *storage*.

Autorizácia pre prístup k zdrojom je riešená vytvorením *dekorátora* pre každú rolu. Dekorátor je potom pridávaný k metódam zdrojov podľa toho, aká najnižšia rola má byť pre použitie metódy vyžadovaná. Ak sa pri niektorej metóde dekorátor nenachádza, potom jej použitie nevyžaduje autorizáciu, ani odosielanie JWT tokenu. Trieda pre autorizáciu sa nachádza v súbore *auth*. Ďalší použitý dekorátor slúži na validáciu prijímaných a odosielaných dát. Šablóny na ich kontrolu sa nachádzajú v priečinku *validators*. Ďalšou funkciou dekorátora je popis koncových bodov podľa špecifikácie *OpenAPI*[4]. Tá slúži pre automatické generovanie dokumentácie nástrojmi *Redoc* a *Swagger*. Kód pre generovanie dokumentácie som prevzal z platobnej aplikácie a upravil pre svoje potreby. Nachádza sa v priečinku *doc* a po pripojení rozšírenia k aplikácii už nebude samostatne potrebný.

7.4 Klient

Klientom v systéme je webová aplikácia implementovaná v jazyku *JavaScript* s využitím knižnice *React*[5]. React je JavaScript knižnica od spoločnosti Meta na tvorbu používateľských rozhraní. Jej základom je tvorba *komponentov*. Komponent slúži na generovanie znovupoužiteľného *HTML* elementu a tiež zapuzdruje jeho funkcionalitu. Kombináciou komponentov vzniká komplexné používateľské rozhranie.

Obr. 7.2: Implementácia zoznamu faktúr

[4]OpenAPI: https://www.openapis.org/
[5]React: https://reactjs.org/

Komponenty môžu byť vytvárané formou funkcií alebo tried. V tejto práci som použil modernejšiu formu funkcií. Pri nej každý komponent tvorí funkcia, ktorá môže prijímať parametre a obsahovať ďalšie funkcie pre prácu s komponentom. Jej návratovou hodnotou je výsledný vygenerovaný HTML kód komponentu. Na správu stavov komponentu slúžia okrem zaužívaných JavaScript funkcií aj vstavané funkcie ako *useState* alebo *useEffect* nazývané *hooks*. Pre jednoduchší popis HTML elementov sa využíva jazyk JSX (*JavaScript XML*), ktorý umožňuje zapisovať HTML kód s miernymi odlišnosťami priamo v jazyku JavaScrpt. JSX umožňuje tiež zapisovať JavaScript kód do vytváraného HTML. Ten sa zapisuje v množinových zátvorkách, napr. `<p>Cena je {suma + daň}</p>`. Ak je takto použitá premenná vytvorená funkciou useState, komponent reaguje na zmenu jej hodnoty jeho opätovným vygenerovaním s novou hodnotou. Vďaka tomu môžu byť na stránke zobrazované vždy aktuálne hodnoty bez potreby ju obnovovať celú.

Obr. 7.3: Implementácia zoznamu faktúr pri zobrazení na mobilnom zariadení

Rozhranie som vytvoril ako *single-page* aplikáciu (SPA). Po jej zostavení vznikne hlavný HTML súbor, ktorý je dynamicky prepisovaný počas práce s aplikáciou. Pre vytvorenie smerovania medzi obrazovkami aplikácie som použil knižnicu *react-router-dom*[6]. Tá umožňuje navigáciu medzi cestami zavolaním funkcie, či tvorbou HTML odkazov s predaním parametrov. Udržiava štruktúru existujúcich ciest a pre každú cestu označuje komponent, ktorý má byť pri jej použití zobrazený. Tiež umožňuje pre jednu cestu vytvoriť viac podciest s odlišnými zanorenými komponentami. Informácie môžu byť inej podstránke odovzdané aj v samotnej ceste, napr. `.../invoices/{id_faktúry}`, podobne ako pri REST API.

Na komunikáciu s API aplikačného serveru som využil vstavanú metódu *fetch* jazyka JavaScript. Jej kombináciou s kľúčovými slovami *async* a *await* som zabezpečil pri komu-

[6]React Router: https://reactrouter.com/

nikácii asynchrónnosť. Po asynchrónnom volaní funkcie používateľ môže rozhranie ďalej ovládať a po získaní nových dát alebo inej odpovede od servera je mu zobrazený nový stav zmenených komponentov.

Metóda fetch je v kóde súčasťou asynchrónnej funkcie, ktorá je zavolaná po nastatí určitej udalosti. Napríklad používateľom vypĺňané vstupy sú ukladané do objektu v komponente a funkcia na ich odoslanie je spustená po kliknutí na tlačidlo. Pre získanie dát, ktoré majú byť zobrazené na obrazovke, napr. detail faktúry, je funkcia s metódou fetch zavolaná pri každom načítaní stránky. Pri zozname faktúr je naviac periodicky kontrolované, či nepribudli nové faktúry. Ak je odpoveď pozitívna, funkcia je spustená opäť.

Na vygenerovanie základnej adresárovej štruktúry rozhrania som použil *npm* balík *create-react-app*[7]. Ten vytvoril základné súbory pre spustenie prázdnej stránky a štruktúru adresára. Priečinok `src` obsahuje základný komponent *App*, v ktorom som definoval štruktúru ciest aplikácie. V priečinku sú uložené aj ostatné, mnou vytvorené komponenty. Každý komponent má vytvorený vlastný priečinok, ktorý obsahuje JavaScript súbor komponentu a CSS súbor pre pridanie štýlu. Na obrázkoch 7.2 a 7.3 sa nachádza implementácia zoznamu faktúr z návrhu (obrázky 6.2 a 6.3). Do aplikácie bol ešte k ostatným filtrom naviac implementovaný filter pre vyhľadávanie podľa čísla faktúry, ktorý na obrázkoch nie je zobrazený.

[7]Create React App: https://create-react-app.dev/

Kapitola 8

Testovanie

Testovanie systému prebiehalo na viacerých úrovniach. Počas vývoja som využíval nástroj *Postman*[1], prístup k databáze cez rozhranie *pgAdmin*[2], automatické testy aj manuálne testovanie používateľského rozhrania. Testovanie počas vývoja prebiehalo na lokálnom virtuálnom stroji, ktorý simuloval funkcionalitu testovacieho servera. Na tento server bol systém priebežne vydávaný a mohol tak byť testovaný v reálnych situáciach použitia na rôznych zariadeniach.

8.1 Testovanie aplikačného rozhrania

V začiatkoch vývoja som na testovanie aplikačného rozhrania používal primárne nástroj Postman. S pribúdaním ďalších funkcionalít som postupne pridával aj automatické testy. Na ich implementáciu som použil framework *unittest*[3], ktorý je využívaný firmou Roger aj pri ostatných aplikáciach implementovaných v jazyku Python. Po implementácii testovacieho používateľského rozhrania som aplikačné rozhranie testoval aj jeho používaním.

8.1.1 Postman

Postman je nástroj na navrhovanie, vytváranie a testovanie API. Pre testovanie je výhodné, že umožňuje vytvárať *kolekcie* požiadavok pripravených na odoslanie. Pripravené požiadavky nemusia mať len pevne predurčený obsah, ale môžu obsahovať aj premenné, ktorých hodnoty záležia na použitom *prostredí*.

Vytvorenú HTTP požiadavku je v rozhraní možné jednoducho odoslať a následne získať odpoveď. Požiadavku aj odpoveď je potom v aplikácii možné zobraziť rôznymi spôsobmi podľa potreby.

8.1.2 Automatické testy

Framework unittest je súčasťou štandardnej knižnice jazyka Python. Podporuje automatizáciu testov, zdieľanie ich nastavení, agregáciu do kolekcií a zároveň ich vzájomnú nezávislosť.

Automatickými testami je pokrytá funkcionalita metód aplikačného rozhrania. Je testované, či sú správne zasielané dáta ukladané do databázy alebo vyžadované dáta naozaj odosielané v správnej podobe späť. Kontrolovaná je aj správna autorizácia, odosielaním

[1] Postman: https://www.postman.com/
[2] pgAdmin: https://www.pgadmin.org/
[3] unittest: https://docs.python.org/3/library/unittest.html

požiadavok s nedostatočným oprávnením a tiež autentifikácia nesprávnym zadaním emailu alebo hesla.

8.1.3 pgAdmin

PgAdmin je open-source nástroj pre správu a tvorbu PostgreSQL databázových systémov. Používal som ho pri manuálnom testovaní aplikačného rozhrania, pre kontrolu očakávanej práce s databázou. Prípadne som mohol dáta v databáze upraviť pre rýchle overenie špeciálnych prípadov.

8.2 Testovanie požívateľského rozhrania

Keďže samotné webové rozhranie má slúžiť na otestovanie API a použiteľnosti aplikácie, bolo používateľské rozhranie testované na rôznych zariadeniach už od začiatkov jej vývoja. Na druhej strane rozhranie nebolo súčasťou zadania, ani počiatočných požiadavok a pre účely testovania som sa skôr zameriaval na jeho funkcionalitu. Preto nebol kladený až taký dôraz na jeho dizajn a elegantné zadávanie všetkých vstupov pri jeho používaní neskúseným používateľom.

V rozhraní je tak pred komerčným použitím potrebné upraviť dizajn. Konkrétne dizajn vstupných polí, tlačidiel, použitého písma a farebnej kombinácie, tak aby rozhranie zapadalo do ostatných produktov firmy Roger. Ďalej by bolo vhodné vylepšiť formát upozornení na zle zadané vstupy alebo im priamo predchádzať ich automatickým formátovaním. Tiež by mohla byť zvýšená spätná väzba pre používateľa, napríklad použitím indikátorov načítavania viacerých miestach, kde hrozí dlhšia odozva, alebo výraznejším upozornením používateľa na úspešné, prípadne neúspešné dokončenie akcie.

8.3 Navrhované rozšírenia

Najvýznamnejším rozšírením systému by bolo pridanie možnosti zasielať správy medzi dodávateľom a odberateľom. To by fungovalo na rovnakom princípe ako zasielanie faktúr. Dodávateľ by buď automaticky cez API alebo v aplikácii zaslal správu pre odberateľa, s jeho pridaným identifikátorom, ktorý dodávateľ pozná. Odberateľa by tak mohol informovať napríklad o zmene podmienok, zmene platenej sumy, alebo mu zaslať upomienku o nezaplatení, či inú informáciu, ktorú by inak zasielal poštou alebo emailom.

Zamedziť prípadnému zasielaniu nevyžiadaných faktúr či správ, by mohlo potvrdenie vyžadované pri prvom kontakte odberateľa dodávateľom. V databáze by boli tieto nadviazané kontakty dvoch používateľov ukladané spolu s ich aktuálnym stavom. Odberateľ by tak mohol žiadosť o kontakt zamietnuť hneď na začiatku, alebo stav jednotlivých nadviazaných kontaktov meniť neskôr v nastaveniach. Používateľ by tak nemal len možnosť zastaviť prijímanie faktúr do aplikácie úplne, ale mohol by toto nastavenie upravovať pre každého dodávateľa samostatne.

Zjednodušením pri vytváraní faktúry vo webovom rozhraní, by mohla byť možnosť pridať si do používateľského účtu čísla používaných bankových účtov. Používateľ by si pri vytváraní faktúry vybral jeden z nich, na ktorý mu má byť faktúra uhradená, a nemusel by tak jeho číslo písať ručne. Bankové účty si používateľ pridáva a overuje v platobnej aplikácii firmy Roger pre uskutočnenie platby, preto by po zjednotení rozšírenia s aplikáciou mohli byť práve tieto údaje využité.

Z dôvodu budúceho zjednotenia používateľov rozšírenia a platobnej aplikácie, rozšírenie neobsahuje zasielanie overovacích správ pre potvrdenie pridávanej emailovej adresy a telefónneho čísla. Tieto operácie sú v platobnej aplikácii vykonávané v spolupráci s autentifikačným serverom, na ktorý moje rozšírenie z bezpečnostných dôvodov nebolo napojené. Pre praktické nasadenie sa s napojením počíta a nebolo tak potrebné túto existujúcu funkcionalitu implementovať samostatne.

Rozšírením, na ktoré bola táto aplikácia pripravovaná, je možnosť jej použitia v rôznych štátoch európskej únie. Napríklad pre firmy a zákazníkov zo Slovenskej republiky alebo Poľska.

Potreba venovať čas zamestnancov firmy na overenie používateľov by mohla byť eliminovaná použitím overenia identity používateľa cez *BankID*[4]. V budúcnosti je jeho zavedenie plánované do platobnej aplikácie. Preto opäť po zjednotení používateľa rozšírenia s používateľom platobnej aplikácie, by mohla byť jeho identita overená aj pre prijímanie a odosielanie faktúr.

[4]BankID: https://www.bankid.cz/

Kapitola 9

Záver

V tejto práci som sa venoval téme pohodlnej správy a platenia faktúr. Jej cieľom bolo vytvoriť aplikačné rozhranie systému, do ktorého budú môcť dodávatelia zasielať faktúry s identifikátormi odberateľov a systém tieto faktúry spáruje s konkrétnymi používateľmi. Používatelia si v systéme môžu prijaté faktúry prezerať a tiež zaplatiť vygenerovaním žiadosti o PSD2 platbu v platobnej aplikácii firmy Roger.

Pre uvedenie do problematiky som skúmal na čo vytváranie faktúr slúži, čo musia faktúry obsahovať a aké sú možnosti ich použitia. Venoval som sa existujúcim spôsobom platieb, pre porovnanie s mnou použitou technológiou PSD2. Podrobne som sa venoval technológiám pre vytvorenie aplikačného rozhrania, na umožnenie práce s faktúrami v systéme rôznym používateľom a jeho napojenia na existujúci systém firmy Roger. Popisoval som, aké technológie sú používané na komunikáciu medzi klientom a serverom, a ako túto komunikáciu zabezpečiť.

Po získaní teoretických znalostí som sa venoval požiadavkám firmy Roger, porovnaniu požadovaného systému s konkurenciou a zosumarizoval som, čo by mal vo výsledku systém spĺňať. Po ujasnení požiadavok som vytvoril návrh databázového modelu, aplikačného rozhrania a tiež som popísal rozhodnutie vytvoriť testovacie webové rozhranie a to, ako by malo vyzerať. Následne som systém implementoval. V začiatkoch implementácie som sa bližšie oboznamoval s technológiami vyžadovanými firmou Roger a ich postupmi využívanými na testovanie systémov počas vývoja. Nakoniec som aplikáciu vo firme odprezentoval a drobné pripomienky ešte zapracoval jej do implementácie. Výsledky mojej práce splnili očakávania firmy Roger a boli ohodnotené pozitívne.

Časovo náročnejšie rozšírenia, ktoré neboli hlavným cieľom tejto práce som popísal v kapitole testovanie a bude ich možné zapracovať v ďalšom vývoji. Nasadenie platobnej aplikácie, pre ktorú som rozšírenie vytváral je plánované na jeseň roku 2022. Pred nasadením rozšírenia prebehne spomínané zjednotenie používateľov a overovania ich údajov. Používateľské rozhranie bude zjednotené s pripravovaným dizajnom webu a aplikácií firmy. Nasadenie rozšírenia spolu s novým dizajnom je očakávané začiatkom roku 2023.

Literatúra

[1] Balaj, Y. Token-Based vs Session-Based Authentication: A survey. 2017.

[2] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N. et al. *Simple Object Access Protocol (SOAP) 1.1* [online]. w3.org, 2015. [vid. 2022-04-07]. Dostupné z: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[3] Cisco. *What are cookies? What are the differences between them (session vs. persistent)?* [online]. cisco.com, 2018. [vid. 2022-04-13]. Dostupné z: https://www.cisco.com/c/en/us/support/docs/security/web-security-appliance/117925-technote-csc-00.html.

[4] Council of the European Union. *Smernica Rady 2006/112/ES z 28. novembra 2006 o spoločnom systéme dane z pridanej hodnoty* [online]. 2006. [vid. 2022-04-07]. Dostupné z: https://eur-lex.europa.eu/legal-content/SK/TXT/?uri=CELEX:32006L0112.

[5] Council of the European Union. *Smernica Európskeho parlamentu a Rady 2007/64/ES z 13. novembra 2007 o platobných službách na vnútornom trhu* [online]. 2007. [vid. 2022-04-07]. Dostupné z: https://eur-lex.europa.eu/legal-content/SK/TXT/?uri=CELEX:32007L0064.

[6] Council of the European Union. *Smernica Európskeho parlamentu a Rady (EÚ) 2015/2366 z 25. novembra 2015 o platobných službách na vnútornom trhu* [online]. 2015. [vid. 2022-04-07]. Dostupné z: https://eur-lex.europa.eu/legal-content/SK/TXT/?uri=celex%3A32015L2366.

[7] Dasgupta, D., Roy, A., Nag, A. et al. *Advances in user authentication.* Springer, 2017.

[8] Dawn M. Turner. *Digital Authentication - the basics* [online]. cryptomathic.com, 2016. [vid. 2022-04-09]. Dostupné z: https://www.cryptomathic.com/news-events/blog/digital-authentication-the-basics.

[9] ECMA international. *The JSON Data Interchange Syntax* [online]. ecma-international.org, 2017. [vid. 2022-03-19]. Dostupné z: https://www.ecma-international.org/publications-and-standards/standards/ecma-404/.

[10] European Commision. Taxation and Customs Union, VAT identification numbers. [online]. [vid. 2022-04-06]. Dostupné z: https://ec.europa.eu/taxation_customs/vat-identification-numbers_en.

[11] European Commission. *Taxation and Customs Union, What is VAT?* [online]. [vid. 2022-04-04]. Dostupné z: https://ec.europa.eu/taxation_customs/what-vat_en.

[12] European Commission. *Regulatory Technical Standards (RTS) enabling consumers to benefit from safer and more innovative electronic payments* [online]. 2017. [vid. 2022-03-22]. Dostupné z: https://ec.europa.eu/commission/presscorner/detail/sk/MEMO_17_4961.

[13] Fakturoid. *Daň z přidané hodnoty – DPH* [online]. fakturoid.cz. [vid. 2022-04-04]. Dostupné z: https://www.fakturoid.cz/almanach/dane/dph-dan-z-pridane-hodnoty.

[14] Fielding, R. a Reschke, J. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content* [online]. datatracker.ietf.org, 2014. [vid. 2022-04-07]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc7231#section-4.3.

[15] Goldstein, I., Jiang, W. a Karolyi, G. A. To FinTech and Beyond. *The Review of Financial Studies.* Apríl 2019, zv. 32, č. 5, s. 1647–1661. DOI: 10.1093/rfs/hhz025. ISSN 0893-9454. Dostupné z: https://doi.org/10.1093/rfs/hhz025.

[16] Gupta, L. *REST Architectural Constraints* [online]. restfulapi.net, 2022. [vid. 2022-03-19]. Dostupné z: https://restfulapi.net/rest-architectural-constraints/.

[17] IANA. *HTTP Status Codes* [online]. iana.org, 2021. [vid. 2022-04-07]. Dostupné z: https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml.

[18] IBM. *Three-Tier Architecture* [online]. ibm.com, 2020. [vid. 2022-03-19]. Dostupné z: https://www.ibm.com/cloud/learn/three-tier-architecture.

[19] IBM. *HTTP Protocol* [online]. ibm.com, 2021. [vid. 2022-03-14]. Dostupné z: https://www.ibm.com/docs/en/cics-ts/5.3?topic=concepts-http-protocol.

[20] Jones, M., Bradley, J. a Sakimura, N. Rfc 7519: Json web token (jwt). *IETF. May.* 2015.

[21] Jones, M. a Hardt, D. *The oauth 2.0 authorization framework: Bearer token usage.* RFC 6750, October, 2012.

[22] Larkins, K. *A Comparison of Python Web Development Microframeworks.* Southern Connecticut State University, 2020.

[23] Masse, M. *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces.* 1. vyd. Ö'Reilly Media, Inc.", 2011. ISBN 978-1-449-31050-9.

[24] McFarlane, G. *How Mastercard Makes Money* [online]. investopedia.com, 2021. [vid. 2022-03-22]. Dostupné z: https://www.investopedia.com/articles/markets/032615/how-mastercard-makes-its-money-ma.asp.

[25] Ministerstvo financí České republiky. *Zákon č. 235/2004 Sb* [online]. 2004. [vid. 2022-04-05]. Dostupné z: https://www.zakonyprolidi.cz/cs/2004-235.

[26] Mumbaikar, S., Padiya, P. et al. Web services based on soap and rest principles. *International Journal of Scientific and Research Publications.* Citeseer. 2013, zv. 3, č. 5, s. 1–4.

[27] Nelson, B. J. *Remote procedure call.* Carnegie Mellon University, 1981.

[28] Nurseitov, N., Paulson, M., Reynolds, R. a Izurieta, C. Comparison of JSON and XML data interchange formats: a case study. *Caine.* 2009, zv. 9, s. 157–162. Dostupné z: https://www.cs.montana.edu/izurieta/pubs/IzurietaCAINE2009.pdf.

[29] Národná rada Slovenskej republiky. *Zákon č. 222/2004 Z. z. Zákon o dani z pridanej hodnoty* [online]. 2004. [vid. 2022-04-05]. Dostupné z: https://www.zakonypreludi.sk/zz/2004-222.

[30] Peterson, R. *Database Architecture in DBMS: 1-Tier, 2-Tier and 3-Tier* [online]. guru99.com, 2022. [vid. 2022-03-19]. Dostupné z: https://www.guru99.com/dbms-architecture.html.

[31] Rescorla, E. *HTTP Over TLS* [online]. datatracker.ietf.org, 2000. [vid. 2022-04-07]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc2818.

[32] Rolfe, A. *MasterCard and Visa continue to grow share of the European cards market* [online]. paymentscardsandmobile.com, 2015. [vid. 2022-04-07]. Dostupné z: https://www.paymentscardsandmobile.com/mastercard-and-visa-continue-to-grow-share-of-the-european-cards-market/.

[33] Sedliaková, A. *Inovácie a zmeny v platobných službách podľa PSD2* [online]. Národná banka Slovenska, 2018. [vid. 2022-02-04]. Dostupné z: http://www.nbs.sk/_img/Documents/_PUBLIK_NBS_FSR/Biatec/Rok2018/01-2018/Biatec_18_1_web.pdf.

[34] Soni, A. a Ranga, V. API features individualizing of web services: REST and SOAP. *International Journal of Innovative Technology and Exploring Engineering.* 2019, zv. 8, č. 9, s. 664–671.

[35] Spasev, V., Dimitrovski, I. a Kitanovski, I. An Overview of GraphQL: Core Features and Architecture. 2020.

[36] Taneja, S. a Gupta, P. R. Python as a tool for web server application development. *JIMS8I-International Journal of Information Communication and Computing Technology.* Jagan Institute of Management Studies, Rohini. 2014, zv. 2, č. 1, s. 77–83.

[37] Tatra banka. *Všetko, čo by ste mali vedieť o novej smernici PSD2* [online]. Dec 2017. [vid. 2022-02-04]. Dostupné z: https://www.tatrabanka.sk/sk/blog/tlacove-spravy/vsetko-co-ste-mali-vediet-novej-smernici-psd2/.

[38] The Economic Times. *Definition of 'Authentication'* [online]. economictimes.indiatimes.com. [vid. 2022-04-09]. Dostupné z: https://economictimes.indiatimes.com/definition/authentication.

[39] The Economic Times. *Definition of 'Authorization'* [online]. economictimes.indiatimes.com. [vid. 2022-04-09]. Dostupné z: https://economictimes.indiatimes.com/definition/authorization.

[40] Tirtea, R., Castelluccia, C. a Ikonomou, D. Bittersweet cookies. Some security and privacy considerations. *European Union Agency for Network and Information Security-ENISA.* 2011.

[41] W3C. *XML essentials* [online]. w3.org, 2015. [vid. 2022-03-19]. Dostupné z: https://www.w3.org/standards/xml/core.

[42] W3Techs. *Usage statistics of Default protocol https for websites* [online]. w3techs.com, 2022. [vid. 2022-04-13]. Dostupné z: https://w3techs.com/technologies/details/ce-httpsdefault.

[43] Česká národní banka. *Popis okamžitých plateb* [online]. cnb.cz. [vid. 2022-04-07]. Dostupné z: https://www.cnb.cz/cs/platebni-styk/certis/popis-okamzitych-plateb/.