FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Informační systém na rozúčtování nájemného a energií
Student:	David Šmolík
Vedoucí:	Ing. Tomáš Vondra, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2020/21

Pokyny pro vypracování

Vytvořte jednoduchý informační systém na rozúčtování nájemného a energií v kancelářském nebo obytném areálu o několika budovách.
Systém bude umožňovat evidenci nájemních smluv, prostor, měřících zařízení, odečtů, zákazníků a záloh, a bude z těchto údajů vypočítávat nájemné.
Bude možné se přihlašovat a budou rozlišeny role administrátora a zadavatele odečtů.
Výsledkem bude prototyp webové aplikace postavený na technologiích dle vlastního výběru a jeho vývojová dokumentace.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 21. března 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Informační systém na rozúčtování nájemného a energií

David Šmolík

Katedra softwarového inženýrství
Vedoucí práce: Ing. Tomáš Vondra, Ph.D.

7. ledna 2020

Poděkování

V první řadě bych rád poděkoval Ing. Tomáši Vondrovi, Ph.D. za odborné konzultace a vedení práce. Dále bych chtěl poděkovat kamarádce Kačce za analytické konzultace a podporu. Nakonec bych rád poděkoval rodině a blízkým za pomoc při uživatelském testování a za podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. ledna 2020 ……

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2020 David Šmolík. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Šmolík, David. *Informační systém na rozúčtování nájemného a energií.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Obsahem této práce je analýza, návrh a implementace prototypu webové aplikace, která má pomoci v efektivitě chodu firem pronajímajících obytné prostory. Tato aplikace umožňuje zadávat energetické odečty, spravovat částečná data smluv, evidovat a upravovat informace o nájemcích, spravovat a upravovat převody jednotek, na základě hodnot odečtů vyhodnocovat energetickou ztrátu, a do jisté míry počítat vyúčtování. Do aplikace se uživatelé přihlašují přes uživatelské údaje.

Software je psán v jazyce Python s využitím frameworku Django a architekturou MVT.

Klíčová slova informační systém, rozúčtování, energie, webová aplikace, Django framework, Python

Abstract

This thesis includes analysis, architectural design and implementation of a prototype of a web application that aims to provide useful tools for companies that specialize in residential rental with tools to work more efficiently. This application allows users to fill in energetical readings, edit data in contracts, edit information about customers and register new ones, customize units and calculate conversions between them. Furthermore to calculate energetical loss based on data from readings and calculate billing. Users sign in the application through their accounts.

The code is written in Python with framework Django and with architectural design of MVT.

Keywords information system, billing, energy, web application, Django framework, Python

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Placení poplatků za nájem je dnes samozřejmá a periodicky se opakující záležitost. Jedna nejmenovaná firma vlastní komplex několika budov. Jednotlivé prostory v těchto budovách jsou pronajímány zákazníkům. Je tedy potřeba vytvořit software, který zajistí možnost rozúčtování poplatků za nájem, což zahrnuje i sazby za spotřebu, vztahující se k různým druhům energie a další související poplatky.

Ve výše zmíněné firmě mají na starost rozúčtování dva zaměstnanci firmy. Každý z nich zachází s daty různým způsobem a oba zastávají ve vztahu k datům částečně odlišnou roli. V současné době se veškeré údaje ukládají do tabulek ve formátu CSV, a proto existuje několik jejich verzí, každá s vlastní logikou. Tyto fakty byly inspirací pro vznik zadání této práce.

Úkolem tohoto softwaru, který je součástí mé bakalářské práce, je vytvoření univerzálního prototypu softwaru, který by byl použitelný pro jakoukoliv firmu, která se zabývá odečty vyúčtováním energií.

Téma jsem si zvolil právě z výše uvedených důvodů, jelikož je zapotřebí, aby se práce s daty sjednotila a bylo tak snazší přijmout např. nového zaměstnance. Zároveň samotná práce s programem bude rychlejší než zadávání a vyhodnocování údajů z tabulek, které jsem již zmiňoval. Podklady pro mou práci jsou data, která mi dala firma k dispozici (např. CSV tabulky) a odborné konzultace.

Práce se zabývá podrobnou analýzou daného problému. V této části je klíčová komunikace a porozumění si se zaměstnanci firmy, kteří umožnili skrze rozhovory autorovi porozumět problematice. V analýze jsou rozebrány jednotlivé případy užití, dále funkční a nefunkční požadavky, případy užití a volba technologie. Následuje návrh řešení pro daný problém, včetně doménového diagramu a návrhu uživatelského rozhraní, dále je popsána implementace spolu s následující kapitolou věnující se rozšiřitelnosti softwaru této práce a nakonec závěr.

Kapitola 1

Cíl práce

Hlavním problémem je absence takového softwaru na trhu, který by byl dostatečně univerzální a umožnil zaměstnancům firem v oblasti pronájmu jednoduchým způsobem zadávat odečty a monitorovat energetickou situaci na měřicích zařízeních.

Hlavním cílem této bakalářské práce je návrh a implementace prototypu softwaru (jímž je webová aplikace), která bude univerzální pro jakoukoliv firmu pronajímající nějaké prostory. Na základě odborných konzultací se zaměstnanci v oboru bylo zjištěno, že není žádný veřejně prodávaný software, které by umožňoval zaměstnancům sledovat průběžně stav měřidel z naměřených dat, a který by uměl vyhodnotit výši poplatků za nájem a energetickou spotřebu. Je tedy cílem vytvořit prototyp softwaru, který by toto umožňoval.

Kapitola 2

Analýza

Tato část bakalářské práce se zaměřuje na analýzu funkčních a nefunkčních požadavků, případů užití a celkovým rozborem jednotlivých částí zadání.

2.1 Úvod do problematiky

Na úvod této kapitoly je uveden stručný popis fungování rozúčtování a odečtů energií ve firmě, jejíž zaměstnanci mi poskytli konzultace. Pro lepší představu je zde čtenář seznámen s běžným chodem firmy po dobu jednoho měsíce.

Řekněme, že na začátku měsíce přijde nový zákazník. Musí se tedy zavést do databáze a vytvořit smlouva o pronájmu. Ve smlouvě je důležitá doba platnosti, výše nájemného a prostory, na které se vztahuje. Dále jeden ze stávajících zákazníků odstoupí od smlouvy. U smlouvy se tedy musí upravit datum platnosti a zákazníkovi je třeba předložit vyúčtování. Za uplynulé období se sečtou jeho měsíční spotřeby na energiích a nájem. Ukáže se, že zákazník, který odešel, poškodil vybavení ubytování, tak si u něj firma poznamená, že je nespolehlivý.

Musí se obejít všechna měřicí zařízení a provést odečty. Údaje z odečtů jsou posléze ukládány. Po provedení odečtů se zaměstnanci firmy dívají na rozdíl naměřených hodnot u hlavního měřidla a součtu hodnot z měřidel vedlejších. Tento rozdíl ukazuje, jaké jsou energetické „ztráty“ – např. když se nějaké teplo přenese na potrubí, jímž proudí voda a není součástí obytného prostoru v nějaké smlouvě.

2.2 Software na rozúčtování nájemného a energií

Software vzniká na základě poptávky od soukromé firmy. Tato firma vlastní několik budov, které pronajímá svým zákazníkům. Je proto pro ně důležité mít nějaký nástroj, který jim umožní vyúčtovat platby za jednotlivá období svým zákazníkům, a to je hlavním důvodem vzniku této bakalářské práce.

Momentálně vyúčtování mají na starost zejména dva zaměstnanci, přičemž každý z nich provádí činnost odlišným způsobem, což je velmi neefektivní a nepřehledné. Je žádoucí, aby byly postupy sjednoceny a mohli tak například nově příchozí zaměstnanci systém jednoduše převzít.

2.3 Požadavky

Seznam jednotlivých funkčních a nefunkčních požadavků byl diskutován s jednotlivými zaměstnanci firmy, kteří se danou problematikou zabývají a je jejich náplní práce. Jednalo se o dva zaměstnance, kteří popsali, jak by pro ně měl systém fungovat, aby jej využili v praxi. Dále vyplynuly ze zadání některé zřejmé požadavky a na závěr byl celkový souhrn všech požadavků doplněn vedoucím práce.

2.3.1 Funkční požadavky

Funkční požadavky – jedná se o popis služeb, které by měl systém poskytovat, reakce systému na určité vstupy a chování systému v určitých situacích. V některých případech mohou funkční požadavky také explicitně určovat, co systém nesmí provést. [1]

F1 Zadání nové nájemní smlouvy – uživatel může zadat novou nájemní smlouvu, jejíž součástí je datum vzniku a zániku její platnosti, dále pak fixní měsíční výše nájemného a nastavitelné datum pro upozornění na vypršení platnosti smlouvy. Ke smlouvě se váže nájemník a prostory, které si pronajme.

F2 Upozornění na vypršení platnosti smlouvy – jak bylo naznačeno v minulém požadavku, systém eviduje u smlouvy datum pro upozornění o vypršení její platnosti. To se projeví zčervenáním záznamu smlouvy v tabulce.

F3 Vložení nového nájemníka – firma přijímá nové zákazníky, a tudíž je potřeba je evidovat. Krom běžných kontaktních údajů je u nájemníka k dispozici i poznámka pro případné dodatečné informace o spolehlivosti.

F4 Editace nájemníka – u každého nájemníka může nastat nějaká změna (např. změna kontaktních údajů nebo spolehlivosti), proto je důležité mít možnost záznamy upravovat.

F5 Správa budov a prostor – je nutné mít přehled o jednotlivých prostorách a jejich evidenci. Systém je navržen tak, aby bylo možné „namodelovat“ například celý areál. Pro příklad lze definovat prostor typu chodba a jí „nadřazený“ prostor typu patro. Všechny typy prostor a závislostí jsou navrženy tak, že je uživatel sám nadefinuje, tudíž není při modelování areálů omezen.

F6 Přiřazení typu prostoru – jelikož je možné definovat jakkoliv velké obytné prostory, je nezbytné mít o každém objektu znalost, jakého je typu – je třeba rozlišit např. chodbu od budovy.

F7 Evidence vlastníků – každý prostor typu budova má svého vlastníka. Je nutné udržovat informace o těchto vlastnících podobně, jak je tomu u nájemníků.

F8 Evidence měřicích zařízení – měřicí zařízení jsou stěžejní pro funkcionalitu tohoto systému. Je tak nezbytné evidovat jednotlivá zařízení, včetně jejich výrobních čísel a příslušnosti k jednotlivým prostorám.

F9 Určení typu měřicího zařízení – je nutné vědět, jakého je zařízení typu (např. rozlišit vodoměr od elektroměru).

F10 Zadání odečtu k příslušnému zařízení – zpravidla po měsíci probíhají odečty na měřicích zařízení. Uživatel si vyhledá příslušné zařízení ze seznamu a zadá do něj naměřenou hodnotu v příslušné jednotce. Zároveň se eviduje i sám zadavatel z důvodů zodpovědnosti při zadávání údajů.

F11 Uživatelské účty – systém umožňuje uživatelům se přihlásit přes jejich přihlašovací údaje. Tyto záznamy zároveň slouží i jako evidence zaměstnanců.

F12 Finanční ohodnocení za určité období – uživatel si může zvolit určité období a konkrétního zákazníka a vyhodnotit příslušné poplatky za energie a nájem.

F13 Exportovatelné CSV – veškeré vyhodnocené údaje (viz funkční požadavek **F12**) lze následně exportovat do formátu CSV – např. pro tisk.

F14 Evidence jednotek – údaje ze zadaných odečtů jsou v různých jednotkách, tudíž je potřeba je zaznamenávat.

F15 Převody jednotek – pro vyúčtování je důležité mít všechny záznamy z odečtů stejného typu měřidla ve stejné jednotce (nejlépe základní), aby se mohly sečíst. Jelikož se ale jednotky v záznamech mohou lišit, je potřeba je umět převádět.

2.3.2 Nefunkční požadavky

Definují kvalitativní kritéria a omezení, jež se vztahují k celému systému, k jeho dílčím částem (subsystémům) anebo jen ke skupině jiných požadavků. [2]

N1 Webová aplikace – systém je implementován jako webová aplikace, takže musí být dostupný přes webový prohlížeč.

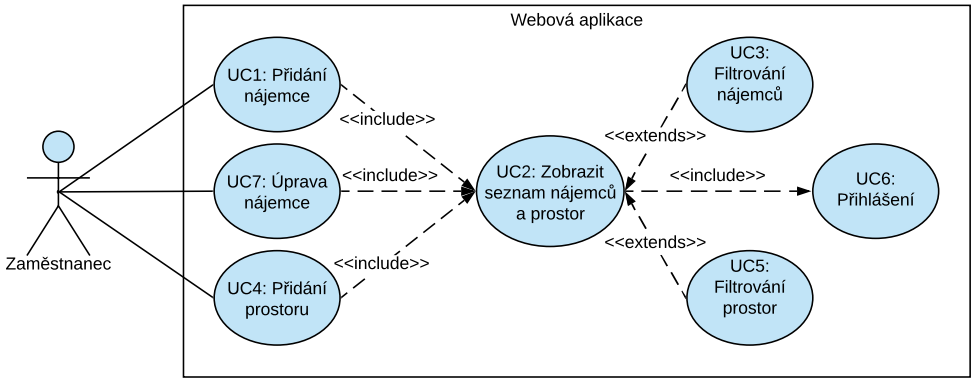
N2 Uživatelské rozhraní – aplikace je svým návrhem uživatelsky co nejpřívětivější. Design byl diskutován s potenciálními uživateli.

N3 Český jazyk – po diskuzi s potenciálními zákazníky vznikl požadavek na použití češtiny jakožto jazyka použitém v uživatelském rozhraní.

2.4 Případy užití

Případy užití („use cases“) slouží k popisu určité funkcionality, respektive tedy vždy jednoho ze způsobů, jakým může být systém používán. Tento model popisuje funkcionalitu jako interakci mezi systémem a uživateli při dosahování konkrétního cíle. Soubor případů užití tak zachycuje kompletní funkčnost systému a představuje funkční specifikaci. [3]

UC1: Přidání nájemce – zaměstnanec je na záložce *Obytné prostory* schopen přidat nového nájemce s příslušnými parametry.



Obrázek 2.1: Případy užití 1

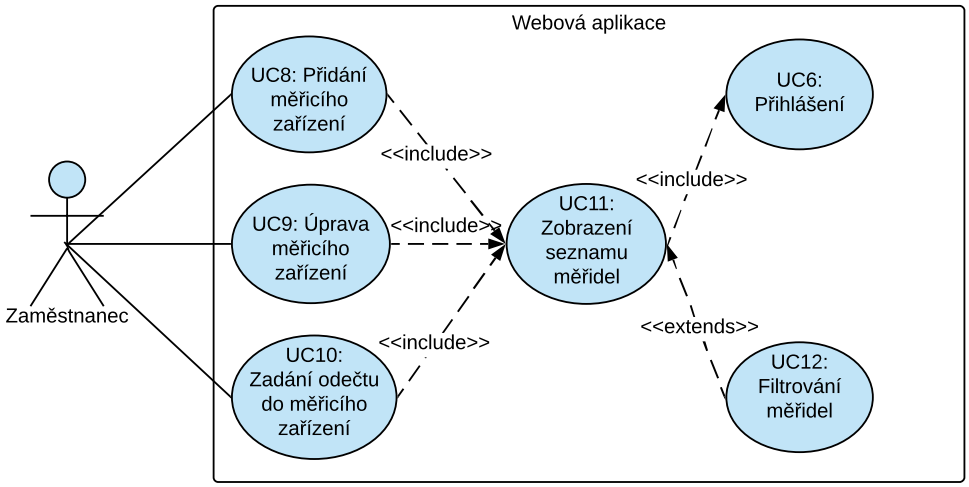
UC2: Zobrazit seznam nájemců a prostor – zaměstnanec si prohlédne seznam všech nájemců a prostor.

UC3: Filtrování nájemců – zaměstnanec si vyfiltruje seznam nájemců na základě objektu, ve kterém nájemci bydlí. Dále pak podle toho, zda má nájemce aktivní smlouvu o pronájmu.

UC4: Přidání prostoru – zaměstnanec přidá na záložce *Obytné prostory* nový prostor s příslušným typem prostoru a případným přiřazením k obytnému objektu.

UC5: Filtrování prostor – zaměstnanec si zobrazí prostory, které budou splňovat volitelná kritéria (budova, vlastník, ulice, patro, obsazenost).

UC6: Přihlášení – pokud chce zaměstnanec používat program, musí se prokázat přihlašovacími údaji. Registraci uživatelů řeší administrátor aplikace.



Obrázek 2.2: Případy užití 2

UC7: Úprava nájemce – zaměstnanec upraví údaje u již existujícího nájemce.

UC8: Přidání měřicího zařízení – zaměstnanec zaeviduje nové měřicí zařízení.

UC9: Úprava měřicího zařízení – zaměstnanec upraví údaje u měřidla.

UC10: Zadání odečtu do měřicího zařízení – zaměstnanec zadá odečet do měřicího zařízení.

UC11: Zobrazení seznamu měřidel – zaměstnanec zobrazí záložku se seznamem měřidel.

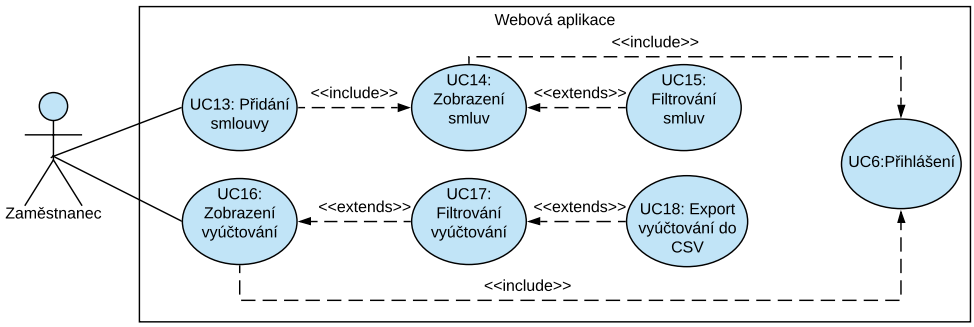
UC12: Filtrování měřidel – zaměstnanec zobrazí pouze měřidla, která splňují vybraná kritéria.

UC13: Přidání smlouvy – do systému je zadána nová smlouva.

UC14: Zobrazení smluv – zaměstnanec si prohlédne seznam se smlouvami.

UC15: Filtrování smluv – zaměstnanec si zobrazí seznam se smlouvami na základě kritérií, která zvolil.

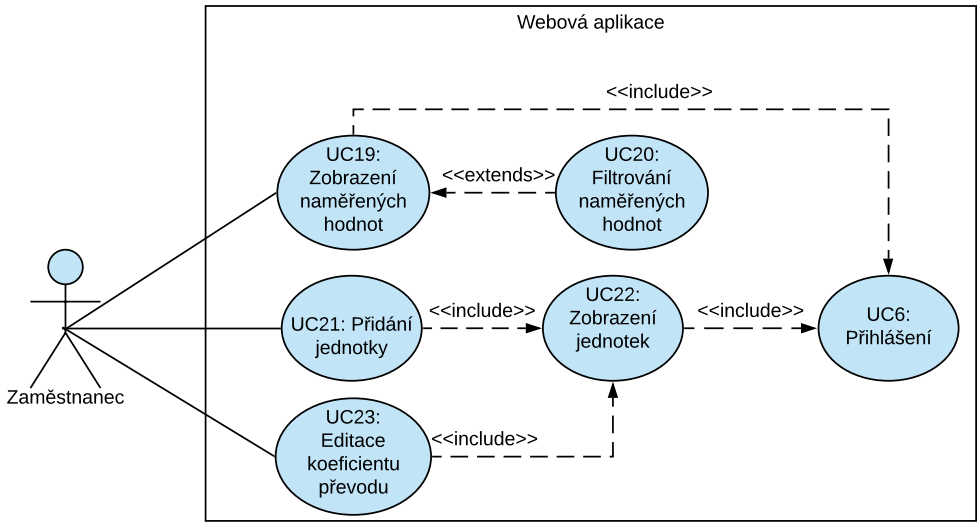
UC16: Zobrazení vyúčtování – zaměstnanec zobrazí vyúčtování.



Obrázek 2.3: Případy užití 3

UC17: Filtrování vyúčtování – zaměstnanec vyhodnotí údaje o vyúčtování na základě parametrů.

UC18: Export vyúčtování do CSV – vyhodnocené údaje exportuje uživatel do souboru ve formátu CSV.



Obrázek 2.4: Případy užití 4

UC19: Zobrazení naměřených hodnot – zaměstnanec zobrazí stav jednotlivých měřidel, jejich hodnot a porovná naměřené hodnoty na hlavním měřidle proti vedlejším.

UC20: Filtrování naměřených hodnot – zaměstnanec si zobrazí měřicí zařízení a naměřené hodnoty.

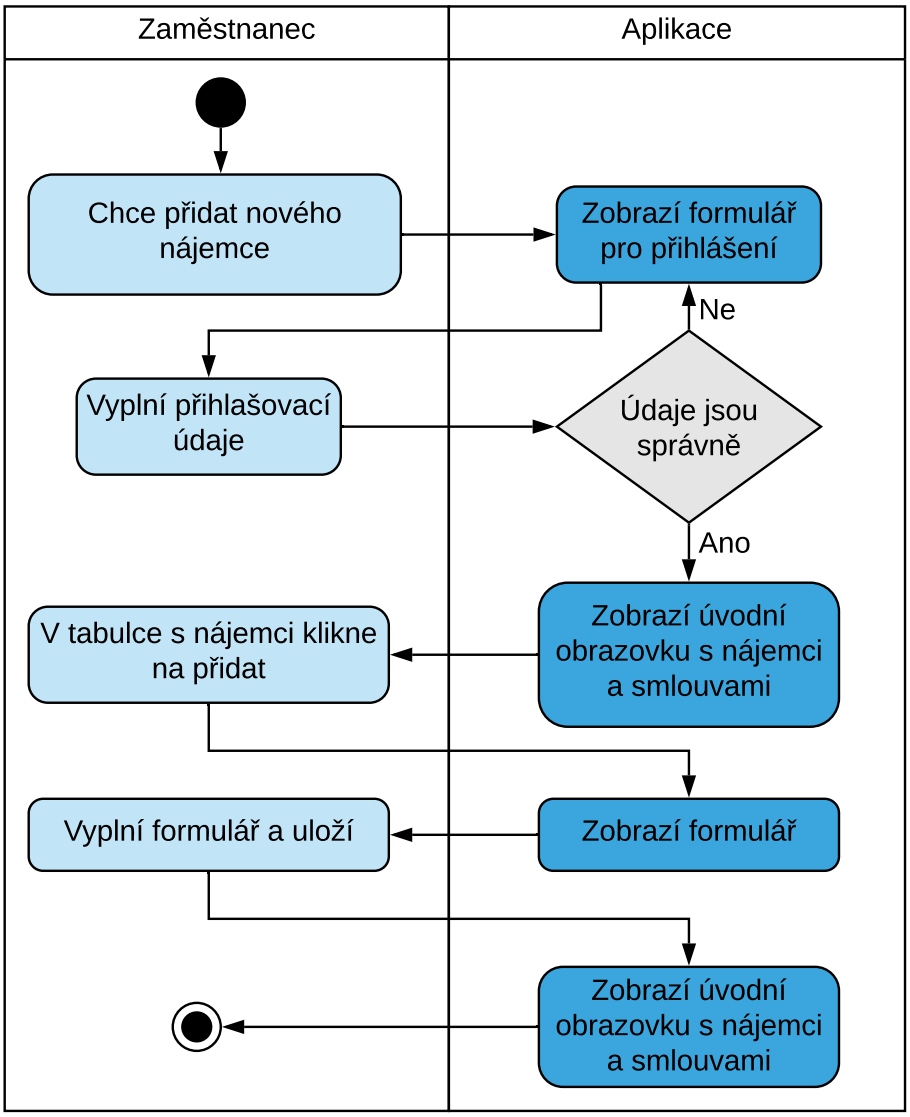
UC21: Přidání jednotky – zaměstnanec zavede do systému novou jednotku.

UC22: Zobrazení jednotek – zaměstnanec může procházet všechny jednotky.

UC23: Editace koeficientu převodu – zaměstnanec upraví koeficient převodu mezi jednotkami.

Podrobnější popis případů užití lze nalézt v příloze B (Detail případů užití).

Na závěr této podkapitoly je uveden diagram aktivit případů užití. Nezahrnuje zdaleka všechny možné scénáře, ale slouží k lepšímu porozumění chodu aplikace. Následující scénář zahrnuje přidání nového nájemníka.



Obrázek 2.5: Swimline diagram přidání nájemce

2.5 Analýza technologií

Nedílnou součástí vývoje softwaru je rozhodování se pro technologie, které budou užity. Tato rozhodnutí jsou velmi důležitá, jelikož na nich závisí stěžejní faktory vývoje. Pokud programátor zvolí jazyk a framework(y)[1] tak, že se mu s nimi dobře pracuje, tvorba systému je kratší a výsledný produkt má lepší kvalitu.

2.5.1 Hodnotící kritéria

Pro volbu jazyka a s tím přímo spojené technologie byly zavedeny vlastní kategorie hodnocení. Každému bodu hodnocení bude přiřazen počet bodů v rozmezí 0–5 bodů, kde 0 je minimum a 5 je maximum – čím vyšší počet bodů, tím lepší hodnocení.

Vhodnost pro vyvíjený systém – V tomto bodě je posuzováno, nakolik je technologie vhodná pro vývoj softwaru, který je předmětem této práce a proč.

Rozšiřitelnost – Toto kritérium je věnováno rozšiřitelnosti. Hodnotí se zde, jak obtížné by bylo software obohatit o další funkcionality.

Znalost – Zde se hodnotí vlastní zkušenosti s danou technologií a na základě znalosti přiřazuje počet bodů.

Dokumentace – Tato sekce vyhodnocuje zpracovanost a přehlednost oficiální dokumentace, která je k dispozici.

2.5.2 Jazyk C# a framework ASP.NET MVC

C# [4] je elegantní a typově bezpečný[2] objektově orientovaný[3] programovací jazyk, který umožňuje vývojářům tvořit celou řadu bezpečných a robustních aplikací, které běží nad .NET Framework [5]. [6]

ASP.NET MVC [7] je open-source[4] framework, který vznikl za účelem tvorby webových aplikací. Jak již napovídá název, architekturou je MVC.

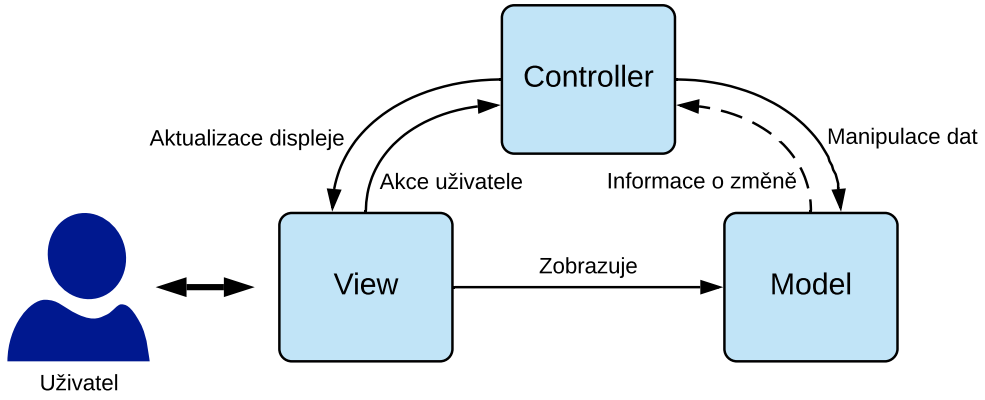
Vhodnost pro vyvíjený systém – MVC je pro předmět této práce velmi dobrou architekturou, protože dobře vystihuje způsob použití aplikace. Je zde uživatel, který chce přes rozhraní (view) manipulovat data z databáze (model) a správný chod a logiku zajišťuje controller. ASP.NET

[1] Framework je knihovna, která ulehčuje vývoj části softwaru.

[2] Typově bezpečný znamená, že jazyk kontroluje, jestli přiřazovaný výraz odpovídá deklarovanému typu proměnné.

[3] Objektově orientované jazyky rozlišují datové struktury jakožto objekty s atributy a metodami.

[4] Open-source je software s veřejně přístupným zdrojovým kódem.



Obrázek 2.6: Architektura MVC (překreslil autor dle [8])

MVC je proto dobrým nástrojem a lze si usnadnit práci řadou dalších frameworků. Nevýhodou toho ovšem je, že musí mít vývojář povědomí o všech těchto technologiích, což vyžaduje mnoho času na nastudování a případnou praxi.

Hodnocení tohoto kritéria je 5 bodů.

Rozšiřitelnost – pokud vývojář dodrží MVC architekturu a nerozhodne se přidat velké množství dalších frameworků, pak je aplikace velmi dobře udržitelná a snadno rozšiřitelná. Pouze přidáním nevhodné knihovny by se mohla rozšiřitelnost pokazit (mohou tak vzniknout například redundantní části kódu). Když se ale vývojář těchto chyb vyvaruje, je aplikace dobře rozšiřitelná.

Hodnocení jsou 4 body.

Znalost – autor se seznámil s touto technologií při tvorbě jednoho rozsáhlejšího projektu. Osvojil si architekturu MVC a vyzkoušel několik rozšiřujících knihoven. Ačkoliv se celkově projekt zdařil, chybí autorovi povědomí o užitečných frameworcích, které by zvýšily rychlost a efektivitu psaní kódu.

Hodnocení jsou 3 body.

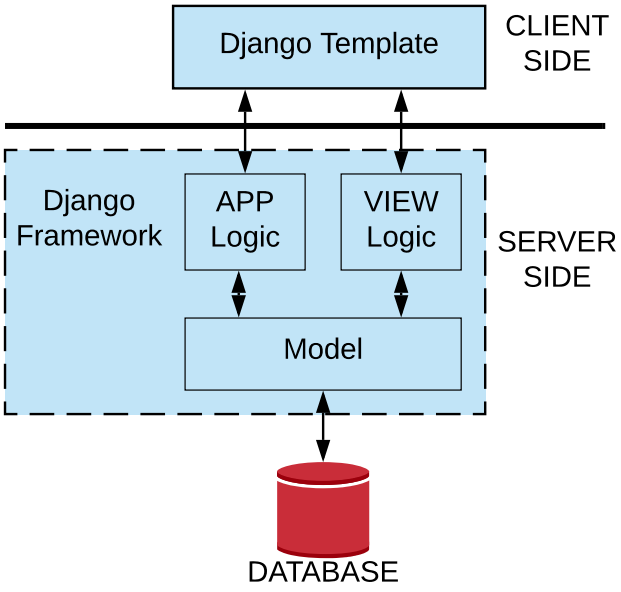
Dokumentace – dokumentace je velmi rozsáhlá a obsahuje i návody. Sama o sobě je dokumentace dobře srozumitelná a návody jsou pochopitelné. Vyhledávání v dokumentaci už ale není tak dobré, protože hledaný výraz je často součástí mnoha sekcí dokumentace a vybrat tu správnou trvá čas a další hledání. Je tedy často lepší použít pro hledané výrazy nějaký vyhledávač.

Toto kritérium je hodnoceno 4 body.

2.5.3 Jazyk Python a framework Django

Python [9] je interpretovaný, objektově orientovaný, vysoko-úrovňový programovací jazyk s dynamickou sémantikou. Jeho vysoko-úrovňové zabudované datové struktury společně s dynamickým typováním a dynamickými vazbami jej činí velmi atraktivní pro Rapid Application Development[5], skriptování nebo jako „slepovací“ jazyk pro již existující části kódu. Jednoduchost Pythonu a lehce naučitelná syntaxe zvyšuje čitelnost a tím pádem i redukuje režii nad udržováním kódu. Python podporuje tvorbu modulů a balíčků, což zvyšuje modularitu a znovupoužitelnost kódu. Pythonovký interpret a rozšiřující se standardní knihovny jsou k dispozici ve zdrojové nebo binární podobě zdarma pro všechny základní platformy a jsou volně distribuovatelné. [10]

Django [11] je vysoko-úrovňový webový framework pro jazyk Python, který umožňuje rychlý vývoj a čistý, pragmatický design. Díky tomu, že byl framework vytvořen zkušenými vývojáři, je možné se soustředit na čistou tvorbu aplikace bez nutnosti větší režie. Je zdarma a open-source. [12]



Obrázek 2.7: Architektura Djanga (překreslil autor dle [13])

Django používá MTV architekturu. Model reprezentuje nějakou datovou strukturu, se kterou framework pracuje. Template je ta část, která se věnuje koncovému uživateli – tomu, co skutečně vidí. View plní roli business vrstvy[6]. Narozdíl od MVC architektury se o roli controlleru stará samotné Django.

[5] Jedná se o moderní přístup vývoje softwaru, kde se klade důraz na vysokou flexibilitu a rychlost vývoje.

[6] Business logika je ta část softwaru, která má na starosti logiku a funkce programu.

Vhodnost pro vyvíjený systém – MVT je architektura do jisté míry podobná MVC, o které již bylo řečeno, že je vhodná pro tuto aplikaci. Výhodou MVT je, že část režie controlleru je převedena na samotný framework a není tedy nutné psát další části kódu a vývojář může věnovat více času jiným částem softwaru.

Zde je hodnocení 5 bodů.

Rozšiřitelnost – samotná architektura tohoto frameworku nabádá vývojáře k modularitě. Kód je přehledně rozdělen na části MVT ,a na rozdíl od ASP.NET MVC není potřeba přidávat externí knihovny. Díky vlastnostem Pythonu může být kód navíc kratší, než je tomu u C#, což může vést k lepší čitelnosti kódu.

Tato kategorie je hodnocena 5 body.

Znalost – autor se aktivně podílel na 3 projektech menšího rozsahu s využitím této technologie. Měl možnost vícekrát vyhodnotit svůj postup při psaní kódu. Rozsah projektů nebyl dostatečný proto, aby vyzkoušel všechny vlastnosti tohoto frameworku, ale dosáhl větší praxe se základními prvky.

Hodnocení je zde 4 bodů.

Dokumentace – dokumentace je příjemně vizuálně zpracovaná. Obsahuje dostatečná vysvětlení, a to včetně ukázek kódu a návodů. Vyhledávání v dokumentaci funguje uspokojivě, ale výsledek vyhledávání si musí vývojář opět vybrat ze seznamu nabízených (např. podle verze frameworku).

Toto kritérium je ohodnoceno 4 body.

2.5.4 Vyhodnocení

Kritérium	ASP.NET MVC	Django
Vhodnost pro vyvíjený systém	5	5
Rozšiřitelnost	4	5
Znalost	3	4
Dokumentace	4	4
Celkem	16	18

Tabulka 2.1: Výsledek srovnání technologií

Z dat z tabulky 2.1 je patrné, že bodově je na tom lépe framework Django. Hlavními důvody tohoto výsledku jsou: lepší znalost frameworku autorem, lepší vhodnost pro tento konkrétní software a potenciálně i větší pravděpodobnost rozšiřitelnosti. Django je zvoleno pro implementaci softwaru, který je předmětem zadání.

Kapitola 3

Návrh

3.1 Doménový model

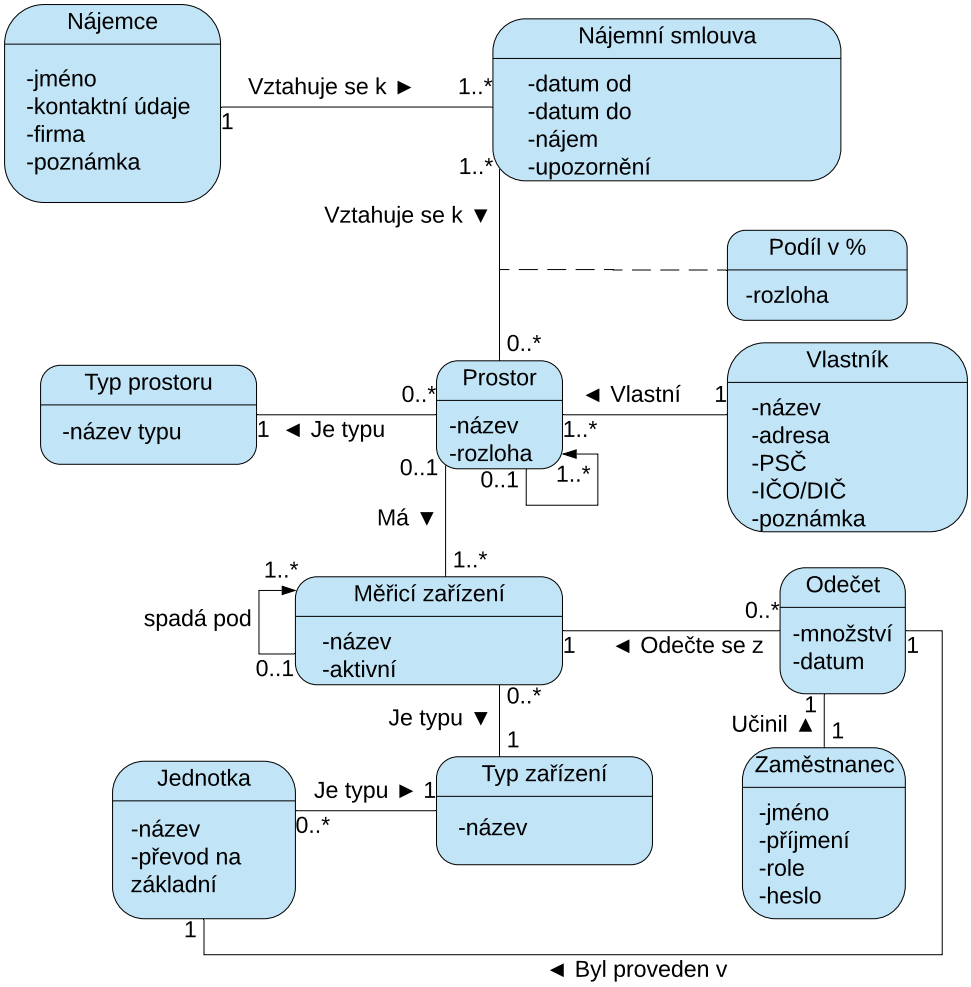
Doménový model je takový druh diagramu, který zachycuje objekty reálného světa a popisuje jejich vlastnosti pomocí atributů. Cílem doménového modelu je popsat nějaké entity a vztahy mezi nimi tak, aby později bylo možné vytvořit model databázový. Zároveň může doménový model fungovat jako kontrolní prvek mezi tvůrcem softwaru a jejím zadavatelem – obě strany se tak přesvědčí, že si vzájemně porozuměly při návrhu.

Následující doménový diagram vznikl na základě rozhovoru s lidmi pohybujícími se v oboru, který vzniknuvší webová aplikace postihuje. Model byl diskutován se dvěma zaměstnanci nejmenované firmy, kteří provozují činnost vyúčtování nájemného a energií.

Nájemce – představuje jakoukoliv osobu, se kterou je podepsána nájemní smlouva. Je tedy důležité ji evidovat a znát údaje typu: jméno, příjmení, kontaktní údaje (adresa, e-mail, telefonní číslo), případně název firmy, kterou zastupuje. U nájemce se dále eviduje poznámka, aby si mohl pronajímatel například zaznamenat spokojenost s daným nájemcem. Nájemce může uzavřít libovolný počet nájemních smluv.

Nájemní smlouva – je druh smlouvy, kterou nájemce uzavírá s pronajímatelem (majitelem softwaru). Má své datum vzniku a zániku platnosti. Zahrnuje výši nájemného a prostory, které jsou pronajímány. Ne všechny prostory jsou však pronajímány zcela. Existují například společné prostory typu „chodba“, které jsou podílem zahrnuty částečně. Pronajímatel má možnost nastavit datum upozornění. To způsobí, že od určitého data bude pronajímatel upozorněn na blížící se konec platnosti smlouvy.

Prostor – je nějaká část obytného komplexu (jakkoliv velká), která má svůj název (např. „Budova A“ nebo „Byt 3C“) a svou rozlohu v metrech



Obrázek 3.1: Doménový model

čtverečních. Pomocí prostoru lze namodelovat jakýkoliv obytný komplex, protože každý prostor má svůj typ (např. „kancelář“) a zároveň si mohou být prostory nadřazené (např. prostor „budova“ bude nadřazen prostoru „chodba“). Každý prostor je také snímán měřicími zařízeními (maximálně jedno zařízení na stejný druh energie).

Vlastník – je společnost/osoba, která vlastní daný objekt. Vlastník je vždy vázán pouze s prostorem typu „budova“.

Měřicí zařízení – je takový typ zařízení, který snímá spotřebu určitého druhu energie na daném prostoru. I tato měřidla mají svou hierarchii. Může být například jedno měřidlo pro celé patro a pak menší měřidlo

pro jednu kancelář. Na rozdílech naměřeného součtu údajů z podružných měřidel proti nadřazenému lze vidět energetické ztráty.

Odečet – se provádí jednou měsíčně. Reprezentuje naměřenou hodnotu spotřeby energie na konkrétním měřicím zařízení a v určité jednotce.

Zaměstnanec – je osoba patřící k firmě vlastnící software. Má své přihlašovací údaje a může do aplikace zadávat naměřené hodnoty z energetických odečtů.

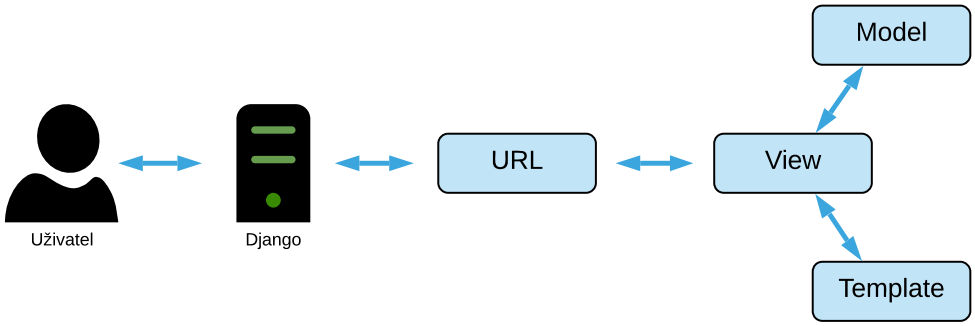
Typ zařízení – určuje, jakého druhu energie snímá zařízení spotřebu. Zároveň sjednocuje všechny jednotky vztahující se ke stejnému druhu energie.

Jednotka – reprezentuje míru energetické veličiny. Odečty se mohou lišit v jednotkách a když se mají záznamy sčítat a porovnávat, musí být ve stejné jednotce. Zároveň součet na zařízeních může být požadován v jakékoliv jednotce ,a proto je důležité znát u každé jednotky její konstantu převodu na základní jednotku.

3.2 Architektura

Softwarová architektura systému líčí systémovou organizaci nebo strukturu a poskytuje vysvětlení o tom, jak funguje. Jedná se o systém reprezentující kolekci komponent, které plní specifické funkce nebo jejich množinu. Jinými slovy, softwarová architektura poskytuje robustní základ, na kterém může být software postaven. [14]

Základní architekturou pro framework Django je MVT, proto bude tvořit i architekturu tohoto softwaru. Už podle zkratky lze odhadnout, že bude mít architektura tři prvky – model, view, template. Každé z těchto částí je věnována samostatná podkapitola. Pro celkovou představu funkčnosti architektury je uveden obrázek 3.2.

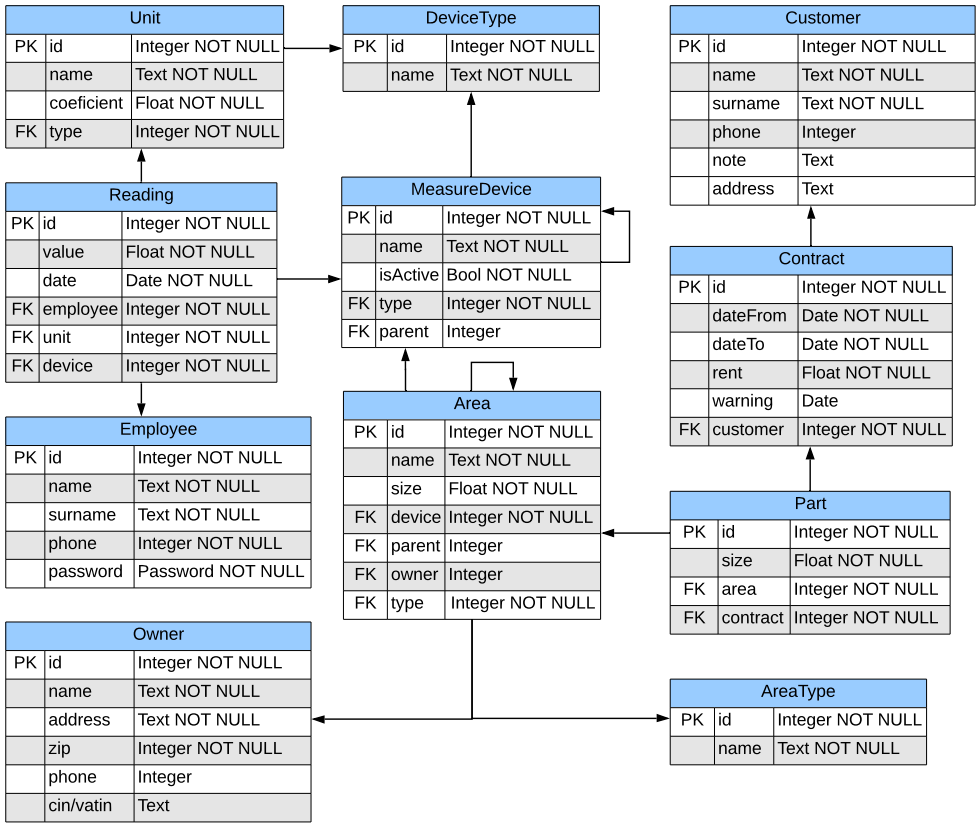


Obrázek 3.2: Architektura MVT (překreslil autor dle [15])

3.2.1 Model

Model je takový prvek architektury, který reprezentuje množinu datových tříd jednotlivých objektů, se kterými systém zachází. Jeden objekt je často reprezentací nějaké konkrétní věci (není-li abstraktní) – např. smlouvy. Model tedy určuje, jak taková třída vypadá – určuje jaké má atributy a jaká data je třeba uchovávat (nejčastěji formou nějaké databáze). Neslouží ale jen pro ukládání dat do databáze, nýbrž obsahuje i metody pro manipulaci s daty (např. můžeme definovat metodu, která vrátí všechny instance této třídy z databáze nebo jejich podmnožiny). Na obrázku 3.3 je vidět návrh struktury databáze.

Pro ukládání dat byla zvolena softwarová knihovna SQLite [16]. Mezi hlavní důvody pro použití této knihovny patří její rychlost a jednoduchá práce s frameworkem Django. Dále není zapotřebí server pro ukládání dat a zároveň se jedná o spolehlivý software.



Obrázek 3.3: Databázový model

Významový popis těchto datových tříd najdeme v podkapitole 3.1. Jediný vztah v doménovém modelu s vazbou „M:N“ najdeme mezi *Nájemní smlouvou* (Contract) a *Prostory* (Area). Tento vztah je vyřešen tabulkou *Podíl* (Portion),

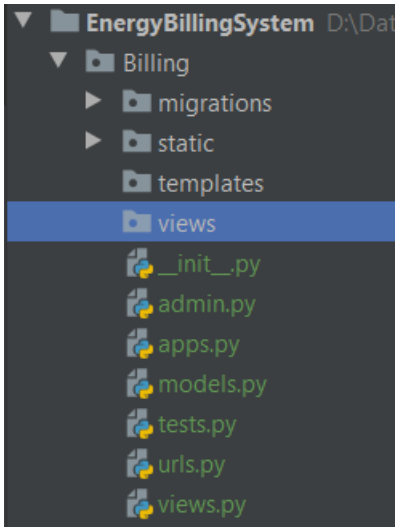
která obsahuje cizí klíče z obou předtím zmíněných tabulek. Zároveň obsahuje i informaci o části daného prostoru, která se přímo týká smlouvy.

3.2.2 View

Již v předchozí části práce 2.5 byl zmíněn návrhový vzor MVC. V tomto typu architektury najdeme také (stejně jako je tomu u MVT) „model“ a v obou případech se jedná o reprezentaci tříd pro ukládání dat. Obě architektury obsahují také „view“. I přes to, že se v obou architekturách nalézá tento prvek, nejedná se významově o stejnou část.

U MVC reprezentuje view často nějaký statický soubor, např. typu HTML, který zastupuje roli zobrazení dat z databáze v podobě nějakého uživatelského rozhraní.

V MVT jsou views soubory obsahující nějaké funkce. Tyto funkce mají na starost zpracovat request[7] od uživatele a na jeho základě odpovědět v podobě zobrazení nějakého obsahu nebo přesměrování na jiné URL. Zmíněné funkce samozřejmě neobsahují jen jednoduchá přesměrování, ale i logiku (jaká data jsou potřeba k zobrazení obsahu, případné ošetření výjimek v rámci neúspěchu). Zpravidla se všechny tyto funkce přidávají do souboru *views.py*. Protože je ale očekáváno, že funkcí bude potřeba větší množství a autorovi nepřišlo rozumné ukládat všechny do jednoho souboru, byl vytvořen adresář *views* obsahující větší množství souborů s odpovídajícím obsahem.



Obrázek 3.4: Adresářová struktura

[7]Request je balíček informací odeslaných klientem, požadující od serveru zobrazení určitých dat.

3.2.3 Template

Poslední část návrhového vzoru MVT tvoří takzvané „templates“, ty zajišťují vzhled uživatelského rozhraní. To znamená, že obsahují statickou část kódu – např. HTML, samozřejmě s plnou podporou CSS a dalších souvisejících komponent, jsou-li potřeba. Dále ale obsahují i dynamickou část kódu, kterou tvoří specifická syntax. Tyto soubory se zobrazují uživateli na základě „views“, kdy uživatel zadá request, ten je zpracován a na základě logiky „view“ se zobrazí příslušný „template“.

Hojně používanou část kódu tvoří tzv. DTL (Django template language). Jedná se o speciální druh syntaxe, stvořený za účelem psaní kódu, který je dynamicky generován – např. obsah tabulek z databáze.

Důležitým faktem je, že „templates“ plně podporují dědičnost. Je tedy možné vytvořit např. nějakou univerzální šablonu (třeba s navigací) a na jejím základě rozšířit všechny ostatní „templates“ s příslušnými podrobnostmi.

3.3 Uživatelské rozhraní

UI design je činnost zaměřená na vývoj rozhraní aplikací používaných v počítačích, strojích, mobilních zařízeních a internetových prohlížečích, která se zaměřuje na efektivitu uživatelské interakce a ergonomie jejího používání.

Jejím cílem je zajistit co nejjednodušší a nejúčinnější používání tak, aby požadované úkoly byly splněny, aniž by se uživatel musel zabývat aplikací jako takovou. Proces tvorby uživatelského rozhraní vytváří rovnováhu mezi technickou funkcionalitou a vizuálními elementy tak, aby vytvořil systém, který je nejen funkční, ale i jednoduše a logicky použitelný pro uživatele aplikace. [17]

3.3.1 Návrh obrazovek

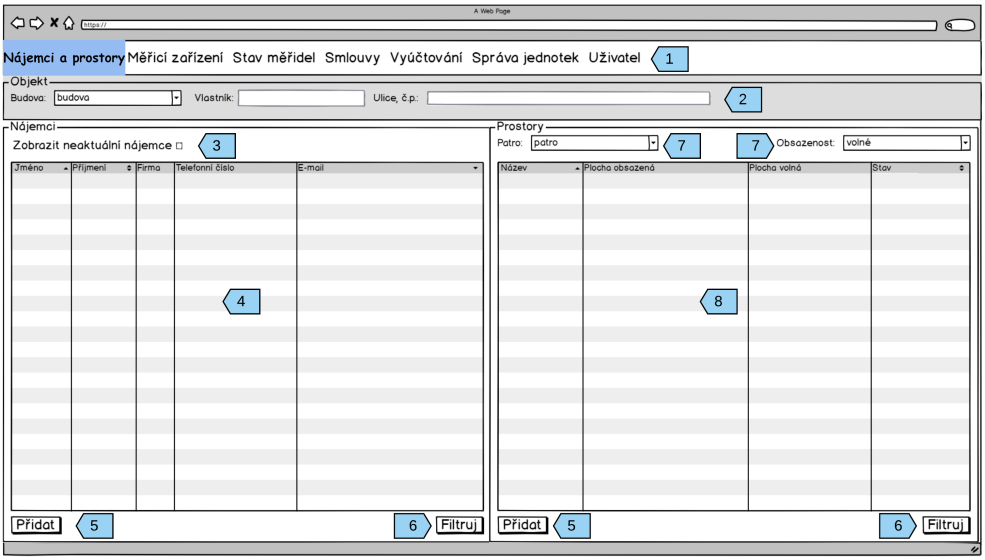
Design jednotlivých obrazovek byl diskutován s lidmi z oboru, pro které je tento software primárně určen. Autor práce se s těmito lidmi dohodl a postupně jednotlivé obrazovky vznikaly tak, že autor kladl otázky na téma, jak by měla aplikace vypadat. Na základě odpovědí rovnou na místě rozhovoru modeloval autor wireframy[8], které mu byly odborníky schváleny. Jelikož se jedná pouze o návrh, má tak v zásadě dvojí funkci. První je, že při tvorbě softwaru slouží tyto návrhy do jisté míry jako pojištění, že si zadavatel a vykonavatel práce rozumějí. Za druhé slouží jako podklad pro grafické zpracování, které se ovšem ve výsledku může lišit.

Následující část této kapitoly je věnována právě těmto wireframům, jak vznikaly a jejich popisu. Bohužel nebylo možné z časových důvodů s odborníky prodiskutovat návrh všech stránek, je tak uvedena jen část z nich na ukázku.

[8] Wireframe je návrh rozmístění jednotlivých prvků na stránce.

3.3.1.1 Obytné prostory

Na obrazovce 3.5 jsou vidět tabulky s jednotlivými nájemci a také s prostory. Obě tabulky je možné podle údajů filtrovat. Do každé z tabulek je možné přidávat nové údaje nebo ty staré měnit.



Obrázek 3.5: Návrh obrazovky nájemců a prostor

1 Navigační menu – slouží k přepínání mezi stránkami aplikace.

2 Objekt – tvoří údaje, podle kterých lze filtrovat výsledky hledání. Všechny tyto parametry jsou pro vyhledávání volitelné.

3 Zobrazení neaktuálních nájemců – je také údaj pro vyhledávání. Neaktuálními nájemci se myslí ti, kteří v současné době nemají uzavřenou smlouvu.

4 Tabulka nájemců – ve které lze prohlížet a editovat údaje o nájemcích.

5 Přidat – je tlačítko plnící stejnou funkci u nájemců i prostor – přidá nový údaj do tabulky.

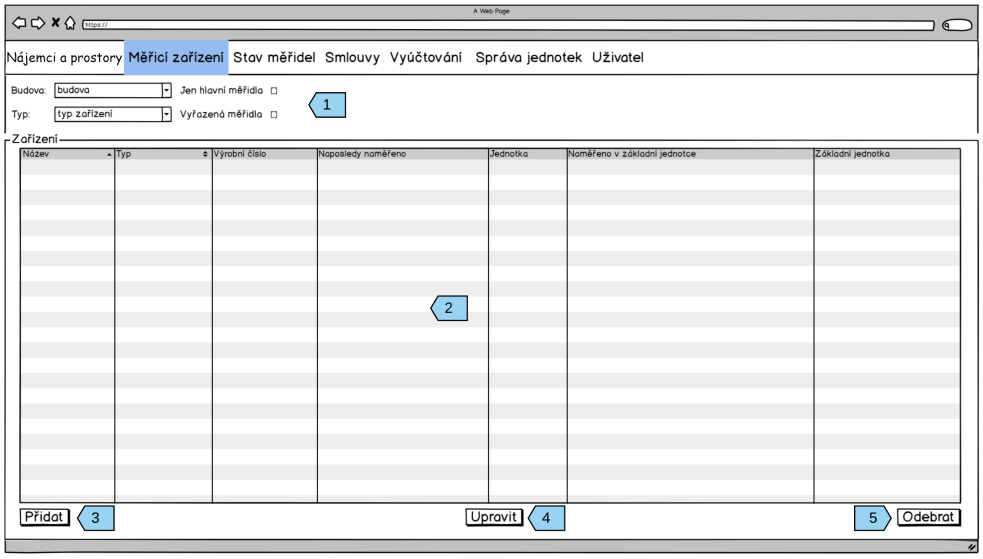
6 Filtruj – je tlačítko pro potvrzení vyhledávání na základě vyplněných údajů.

7 Patro a obsazenost – jsou údaje pro filtrování jednotlivých prostor. U obsazenosti se rozlišuje, zda je prostor volný k obsazení, nebo je již alespoň částečně obsazen.

8 Tabulka prostorů – slouží k editaci a evidenci jednotlivých prostor.

3.3.1.2 Měřicí zařízení

Hlavní rolí této obrazovky je možnost celkově spravovat všechna měřicí zařízení. Spolu s přibývajícími budovami mohou přibývat nová zařízení, stará se mohou vyměňovat za nová atd.



Obrázek 3.6: Návrh obrazovky měřicích zařízení

1 Údaje pro filtrování – slouží hlavně pro rychlejší nalezení požadovaných měřicích zařízení. Všechna tato pole jsou nepovinná. Typ zařízení se rozlišuje podle energie, jejíž spotřebu měří (voda, plyn, elektřina). V tomto případě se hlavními zařízeními míní ta, která nemají už žádná jiná zařízení jim nadřazená. Vyřazená měřidla jsou ta, která se již nepoužívají a uchovávají se z historických důvodů.

2 Tabulka zařízení – zobrazuje požadovaná zařízení na základě možností filtrování, viz bod 1 Údaje pro filtrování. Jednotlivá zařízení je možná rozkliknout „dvojklikem“ pro editaci. Záznamy je možné řadit podle sloupečků tabulky vzestupně i sestupně.

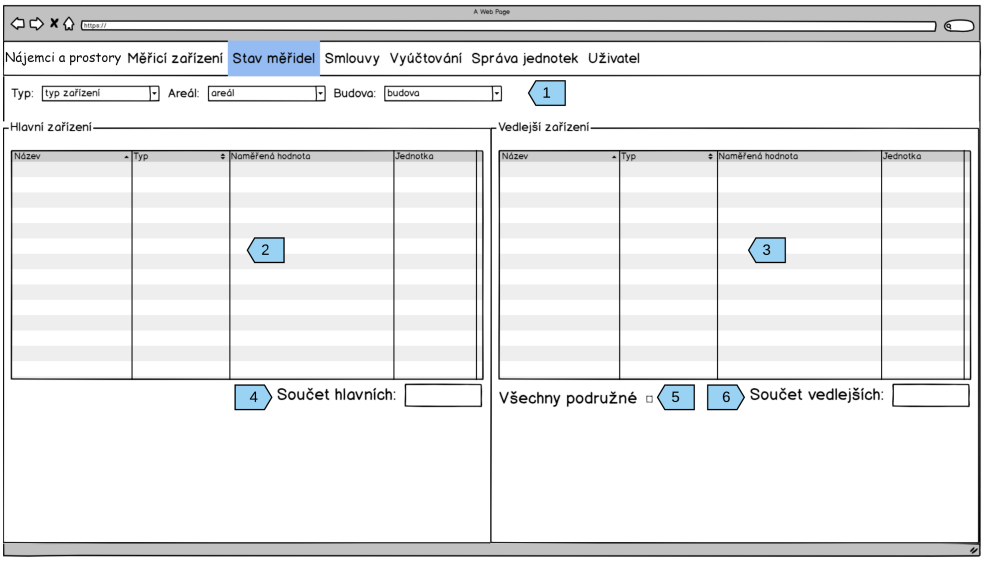
3 Přidat – je tlačítko, které po stisknutí zobrazí formulář pro zadání nového zařízení.

4 Upravit – otevře okno pro úpravu měřicího zařízení.

5 Odebrat – umožňuje uživateli odstranit existující zařízení, není-li již potřeba jej dále evidovat v databázi.

3.3.1.3 Stav měřidel

Tato část aplikace je důležitá zejména pro měření energetických ztrát. Máme-li například nějaké měřicí zařízení na konkretním topení a na nějakém tepelném rozvaděči, lze předpokládat, že se nějaké teplo po cestě „ztratí“. Proto se na této obrazovce lze podívat na naměřené hodnoty měřidel hlavních oproti součtu hodnot z vedlejších měřidel. Rozdíl hodnot na hlavních měřidlech proti vedlejším je vnímán jako „ztráta“.



Obrázek 3.7: Návrh obrazovky stavů měřidel

1 Údaje pro filtrování – jsou všechny nepovinné a jsou zde od toho, aby mohl uživatel nalézt zařízení, která např. patří jen do nějaké budovy.

2 Tabulka hlavních zařízení – zobrazuje zařízení nadřazená vedlejším.

3 Tabulka vedlejších zařízení – zobrazuje zařízení vedlejší, proti těm hlavním. Vedlejší zařízení ale může také být hlavním pro další podřadná zařízení (dalo by se říci „o vrstvu níže“). Proto si jej lze rozkliknout jako měřidlo hlavní a podívat se na jeho vlastní vedlejší zařízení.

4 Součet hlavních – je číselná hodnota v základní jednotce, která je tvořena sumou naměřených hodnot z tabulky hlavních zařízení.

5 Všechna podružná – jsou myšlena zařízení pouze v „nejnižší“ vrstvě (tzn., že nejsou hlavní pro žádná jiná zařízení), nikoliv jen o vrstvu „níže“, jak jsou data zobrazována v ostatních případech.

6 Součet vedlejších – je sumou součtu zobrazených naměřených hodnot vedlejších měřidel v základní jednotce.

3.3.1.4 Smlouvy

Je samozřejmě nutné mít možnost pracovat se smlouvami. Tato obrazovka je velmi jednoduchá a sestává pouze z tabulky, která obsahuje záznamy smluv, a z jednoduchého formuláře pro vyhledávání smluv podle jejich data. Samotné záznamy v tabulce je možné upravovat a přidávat nové. Smlouvy se z důvodů uchování historie nesmí mazat.

3.3.1.5 Vyúčtování

Jedná se o takovou obrazovku, která uživateli umožňuje spočítat vyúčtování za volitelné období. Software tedy na základě údajů (počet měsíců, výše nájemného, energetická spotřeba na daném prostoru) spočítá výslednou částku. Uživatel zde má dále možnost tyto údaje exportovat do souboru formátu CSV pro případ potřeby.

3.3.1.6 Správa jednotek

V této části může uživatel manipulovat s jednotkami. Může přidávat nové jednotky a upravovat ty staré. Mazat jednotky však může jen v případě, že v nich není naměřen dosud žádný odečet. Krom výše zmíněného obsahuje obrazovka i nástroj v podobě převodníku. Uživatel může převádět mezi jednotkami, které jsou uloženy v databázi.

3.3.1.7 Uživatel

Uživatel zde může vidět své údaje, avšak jedinou změnu, kterou smí provést ,je změna vlastního hesla. Pokud bude chtít běžný uživatel změnit jakýkoliv jiný údaj (např. příjmení), musí o to požádat administrátora.

3.4 Uživatelské role

Z hlediska návrhu je důležité rozlišovat role uživatelů aplikace. Každá z rolí má svá práva a zodpovědnost vůči vyplňovaným údajům.

3.4.1 Administrátor

Administrátor je někdo, kdo má přístup do administrátorské části aplikace. Tato část umožňuje uživateli plnou kontrolu nad databází. Smí například upravovat nebo mazat údaje, které běžný uživatel nesmí. Velkou zodpovědnost má administrátor na začátku projektu. Musí definovat jednotlivé typy prostor a přidává ostatní zaměstnance.

3.4.2 Běžný uživatel

Má svůj účet, pod kterým se přihlašuje do aplikace. Manipulovat s daty může jen do té míry, do jaké mu to aplikace umožňuje (bez administrátorské části). Povinností těchto zaměstnanců je ukládat správně data, zejména u odečtů, proto se také eviduje, kdo odečet zadal.

3.5 Technologie

Již v sekci 2.5 bylo zmíněno, že jazyk, ve kterém je aplikace napsána ‚je Python. Hlavním frameworkem, který do velké míry ovlivnil architekturu softwaru ‚je také již zmíněné Django. O těchto technologiích tedy už není potřeba cokoliv zmiňovat. Tato kapitola je věnována jiným technologiím, které podporují tvorbu softwaru.

3.5.1 Bootstrap

Bootstrap [18] je volně dostupný open-source front-endový[9] framework na výrobu webových stránek a aplikací. Bootstrap framework je postaven na HTML, CSS a JavaScriptu, aby umožnil snadnější vývoj responzivních[10], mobile-first[11] stránek a aplikací. [19]

Bootstrap je tedy používán ve statických souborech obsahujících HTML. V tomto případě, protože se jedná o architekturu MVT, tvoří tuto část „templates“.

[9] Front end je vizuální část aplikace, která je vidět uživatelem.

[10] Responzivní je taková stránka/aplikace, která se umí přizpůsobit velikosti zařízení, na kterém je spuštěna.

[11] Mobile-first jsou stránky/aplikace, u kterých se předpokládá, že většina zařízení, na kterých software poběží budou smartphony nebo tablety.

Kapitola 4

Realizace

Tato kapitola je věnována samotné realizaci aplikace. Je zde popsáno, jak aplikace vznikala a jaké jsou její části. Konkrétně je popsána realizace zmíněné architektury MVT, která je zde rozdělena na frontend a backend.

4.1 Příprava

Před samotnou implementací aplikace bylo nutné vykonat jisté přípravné kroky, aby byl vývoj co nejefektivnější.

4.1.1 Verzování

Pro verzování vývoje aplikace byl zvolen systém GIT. Konkrétně byl zvolen server gitlab.com [20].

GitLab je moderní (nejen) webový správce Git repozitářů, velmi inspirovaný populární službou GitHub. Svou koncepcí je tedy zaměřený primárně na práci se zdrojovým kódem, čímž se odlišuje od klasických „manažerských“ nástrojů typu Redmine. Oproti GitHubu je určený do prostředí intranetu, pro neveřejné projekty.

Hlavní komponentu tvoří komfortní prohlížeč kódu s úzkou návazností na Git. Stejně jako GitHub umožňuje psaní komentářů k jednotlivým commitům či přímo konkrétním řádkám kódu. Poskytuje minimalistický issue tracking, který uživatele neobtěžuje zbytečnými políčky. Samozřejmě nechybí ani jednoduchá wiki a všudepřítomná podpora syntaxe Markdown.

Podporuje tzv. merge requests (obdoba pull request z GitHubu) – vytvoření požadavku na začlenění commitů z jedné vývojové větve do druhé, o kterém pověření členové mohou rozhodnout, zda ho začlenit či nikoli, případně u něj vést diskuzi (vhodné pro revizi kódu od přispěvovatelů). [21]

Gitlab byl zvolen právě pro účely privátního projektu. Mezi hlavní motivy patřily: zvyk autora, dostupnost, snadné nasazení CI.

4.1.2 Vývojové prostředí (IDE)

IDE, Integrated Development Enviroment je programovým vybavením, které slouží vývojářům a programátorům, ve většině případů se zaměřením na určitý programovací jazyk. Některé systémy IDE jsou dokonce vybavené možností vyvíjet v nich aplikace, a to ve velmi rychlém čase. Systém pro vývoj aplikací se nazývá RAD. V IDE se může nacházet také object browser, a to tehdy, kdy hovoříme o nástrojích pro objektově orientované programování. IDE poskytuje určitému programovacímu jazyku soubor vlastností, které se umí velice dobře přizpůsobit právě paradigmatům tohoto jazyka. Najdou se ale také vývojová prostředí IDE, která dokážou pracovat s více jazyky. [22]

Jako vývojové prostředí pro tento systém by zvolen PyCharm [23]. Nástroje PyCharmu umožňují snadné psaní kódu díky napovídání klíčových slov a celkové nápovědě při formátování kódu vzhledem ke konvencím. Dále obsahuje podpůrné nástroje pro ladění aplikace, v tomto konkrétním případě s využitím frameworku Django.

4.1.3 Počáteční inicializace

Počáteční inicializace projektu byla vytvořena ve vývojovém prostředí PyCharm, kde je již připraven vzor pro vytvoření aplikace s frameworkem Django. Dále byly nainstalovány součásti *PhoneNumberField* pro vkládání telefonního čísla do databáze. Byl vytvořen soubor *requirements.txt* pro snadnou instalaci závislostí a nakonfigurován soubor *.gitlab-ci.yml*, který slouží k nastavení CI. Dále na začátek byly definovány datové třídy v *model.py*, z něhož byla vygenerována databáze. Aby byla možnost přistupovat k administrátorské části aplikace, byl vytvořen tzv. „superuser“. Po těchto krocích byla aplikace nasazena na GitLab.

4.2 Frontend

Tak jako se uživatelé internetu dělí na konzumenty a producenty, tedy ty, kdo webové stránky navštěvují a čerpají jejich obsah, a dále ty, kdo je vytvářejí i spravují, podobně se dělí i části webu.

Frontend je tedy ta část webové stránky, která je viditelná konzumentovi – běžnému návštěvníkovi webu, který si chce přečíst zprávy, koupit zboží, zahrát online hru či cokoli jiného.

Bývají proto producenty zpracovány tak, aby návštěvníka zaujmuly, poskytly mu informace, které potřebuje. Ty jsou uspořádány nejlépe přehledně a pro konzumenta atraktivně. [24]

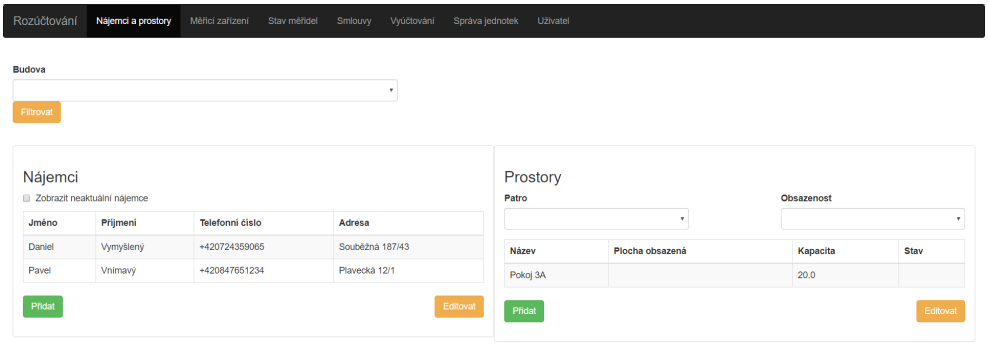
V již zmíněné architektuře MVT reprezentují frontendovou vrstvu „templates“. Tuto část kódu tvoří statické soubory HTML, kde jednotlivé elementy tvoří grafické položky celkového vzhledu. Design těchto prvků je stylizován pomocí CSS. Krom CSS a HTML je ale i použit již zmíněný jazyk samotného

Django. Ten umožňuje dynamické generování HTML a CSS kódu na základě datových tříd (tzv. „forms“). Dále určuje jak a kam se příslušná data zobrazí nebo například umožňuje dědičnost mezi HTML soubory. Podobně jako u tříd v objektově orientovaném programování zde lze definovat části kódu, které si specifikují „potomci“, v Djangu se těmto částem říká bloky.

```
<div class="table-wrapper-scroll-y my-custom-scrollbar">
<table class="table table-bordered table-striped mb-0">
    <thead>
    <tr>
        <th>Jméno</th>
        <th>Příjmení</th>
        <th>Telefonní číslo</th>
        <th>Adresa</th>
    </tr>
    </thead>
    <tbody>
    {% for customer in customers %}
        <tr>
        <td>{{ customer.name }}</td>
        <td>{{ customer.surname }}</td>
        <td>{{ customer.phone }}</td>
        <td>{{ customer.address }}</td>
        </tr>
    {% endfor %}
    </tbody>
</table>
</div>
```

Obrázek 4.1: Ukázka kódu s HTML, CSS a Djangem

Jazykem Django lze krom zobrazení dat vyjádřit i logika jako například podmínky atp. Velikou výhodou jsou automaticky generovatelné formy. Tyto formy mají funkcionalitu pouze na základě nějaké datové třídy vygenerovat formulář pro její manipulaci – například pro vkládání nových dat nebo editaci starých. Všechny HTML soubory jsou podle konvencí uloženy v adresáři *templates*, kde jsou rozděleny do dalších podadresářů.



Obrázek 4.2: Ukázka obrazovky aplikace

4.3 Backend

Jako backend se označuje část webové aplikace, která slouží k administraci webu a ke zpracování dat. V případě redakčního systému tedy backend umožňuje vkládání a úpravy článků, zakládání účtů dalších administrátorů či třeba manipulace se strukturou celého webu. U internetového obchodu bude backend sloužit především ke vkládání nového zboží a k úpravám jeho vlastností, zejména cen.

Pojem backend pochází z anglických slov back a end, backend tedy znamená něco jako „zadní část“. Stojí za zmínku, že koncept backend→frontend je nejčastějším schématem, podle kterého se dnes staví webové aplikace s dynamickým obsahem. [25]

Backend v architektuře MVT zahrnuje „models“ a „views“. Models jsou datové třídy, z nichž je generována databáze. Dalo by se říci, že jde o takzvanou ORM vrstvu[12]. Krom atributů nesoucích data lze v těchto třídách definovat i metody – jak metody „magické“[13], tak klasické. Všechny tyto třídy jsou podle konvencí uloženy v souboru s názvem *models.py*.

Druhou významnou backendovou položkou MVT architektury jsou „views“. V této části se nachází nejvíce logiky z celé aplikace. Views jsou funkce, které reagují na uživatelský vstup (např. stisknutí tlačítka nebo změna URL). Na základě uživatelského vstupu zobrazí požadovaný výstup, který se skládá z libovolného „template“, kterému předá příslušná data z „models“. Dále pak tyto funkce mohou zajišťovat takzvané CRUD[14] operace a ošetření výjimek

[12] Jedná se o vrstvu, která popisuje vztah mezi jednotlivými datovými třídami.

[13] Magické metody jsou speciálním druhem metod v Pythonu, značí se dvěma podtržítky před a za názvem.

[14] CRUD jsou 4 základní operace prováděné nad databází.

```python
class Customer(models.Model):
    """This class represents a customer."""
    name = models.CharField(max_length=256, blank=False, verbose_name='Jméno')
    surname = models.CharField(max_length=256, blank=False, verbose_name='Příjmení')
    phone = PhoneNumberField(null=True, blank=True, verbose_name='Telefon')
    note = models.TextField(null=True, blank=True, verbose_name='Poznámka')
    address = models.CharField(max_length=2056, null=True, blank=True, verbose_name='Adresa')

    def __str__(self):
        return self.name + ' ' + self.surname
```

Obrázek 4.3: Ukázka třídy „Model“

v případě nesprávného vstupu uživatelem. Standardně zmíněné funkce bývají uložené v souboru *views.py*. Vzhledem k množství těchto funkcí se však autor rozhodl rozdělit funkce do více souborů kvůli přehlednosti. Tyto soubory se nachází v adresáři *billing_views*.

```python
def add_customer(request):
    form = CustomerForm(request.POST or None)
    if form.is_valid():
        customer = form.save(commit=True)
        customer.save()
        return customers_and_contracts(request)
    context = {
        'form': form
    }
    return render(request, 'customers_areas/add_customer.html', context)
```

Obrázek 4.4: Ukázka funkce „View“

4.4 Uživatelská příručka

Tato sekce je věnována uživateli, aby věděl, jak lze aplikaci spustit. Jedná se pouze o serverovou část, jelikož se uživatelé budou do aplikace přihlašovat pomocí internetového prohlížeče. Samotné uživatelské rozhraní je navrženo tak, aby bylo co nejvíce intuitivní, což ukáží zejména uživatelské testy v následující kapitole.

Pro následující kroky instalace je nutné mít v zařízení nainstalován *python* (systém vznikal s verzí pythonu 3.7.4) a *pip* (při vývoji byla použita verze 19.3.1). Dále jsou uvedeny kroky pro zprovoznění serverové části:

- Je potřeba spustit příkazový řádek v kořenovém adresáři aplikace (nachází se v něm soubor *manage.py*)
- Pokud se v projektu nenachází virtuální prostředí (adresář s názvem *venv*), je potřeba jej vytvořit příkazem:

  $ virtualenv venv

  Pokud není příkaz *virtualenv* nainstalován, je potřeba jej doinstalovat příkazem:

  $ pip install virtualenv
- V případě, že není virtuální prostředí aktivované, je třeba jej aktivovat následujícím příkazem:

  $ source venv/Scripts/activate (Linux)

  $.\venv\Scripts\activate.bat (Windows)
- V aktivovaném prostředí se před spuštěním musí nainstalovat všechny chybějící komponenty. To se provede následujícím příkazem:

  $ pip install -r requirements.txt
- Dále je zapotřebí připravit databázi. Toho se dosáhne spuštěním následujících dvou příkazů:

  $ python manage.py makemigrations

  $ python manage.py migrate
- Aby bylo možné používat část pro administrátora (která je pro aplikaci naprosto nezbytná), musí se vytvořit tzv. „superuser“. To se udělá následujícím příkazem:

  $ python manage.py createsuperuser

  A dále se postupuje podle pokynů po spuštění toho příkazu.
- Server je nyní možno zprovoznit zadáním příkazu:

  $ python manage.py runserver
- Do části pro administrátory se vstoupí přes url …/admin
- V případě zájmu je možno nahrát do databáze připravená testovací data zadáním:

  $ python manage.py loaddata db.json

Obvykle po zprovoznění serveru je potřeba se přihlásit do části pro administrátory a vytvořit data, aby mohla aplikace správně fungovat. Je potřeba vytvořit uživatelské profily pro zaměstnance. Dále je potřeba vytvořit typy budov atd. Pro některé filtrovací údaje (např. když se vyhledává podle patra)

je potřeba do databáze zadat přesný typ prostoru s názvem „Patro“. Do části pro běžné uživatele je možné se přihlásit příslušným účtem pracovníka (který musí vytvořit administrátor).

Kapitola 5

Testování

Testování softwaru je empirický technický výzkum kvality testovaného produktu nebo služby prováděný za účelem poskytnutí těchto informací všem zainteresovaným (=stakeholdrům). Testování je tedy zejména o hledání určitých informací o produktu jeho zkoumáním. [26]

Tato kapitola je tedy věnována testování systému a aplikace. Bylo provedeno několik druhů různých testů a každému druhu je zde zvlášť věnována podkapitola.

5.1 Continuous integration (CI)

Pojem Continuous Integration (CI) je dnes v IT již celkem zažitý. Jedná se o sadu nástrojů a postupů, které urychlují vývoj, testují software a zároveň provedou nasazení aplikace a to automaticky několikrát denně. [27]

V této práci bylo nasazeno CI v počátcích vzniku aplikace. konkrétně bylo použito CI přímo na serveru gitlab. Testy nebyly prováděny denně, ale automaticky při nahrání nové verze kódu na gitlab. Tyto testy měly za úkol zejména kontrolovat, zda se přidáním nové části kódu neporušila stávající funkcionalita.

5.2 Integrační testy

Integrační testování je fáze testování softwaru, ve které jsou jednotlivé softwarové moduly kombinovány a testovány jako skupina. Objevuje se po testování jednotky a před validačním testováním. Testování integrace trvá jako vstupní moduly, které byly testovány jednotkou, seskupí je do větších agregátů, aplikují testy definované v plánu integračních testů na tyto agregáty a dodávají jako svůj výstup integrovaný systém připravený pro testování systému. [28]

Integrační testy byly prováděny ručně autorem práce. Pokaždé při vytvoření větší části práce bylo ověřováno, zda spolu jednotlivé části kódu správně komunikují.

5.3 Unit testy

Jde o první fázi testování, která se zaměřuje na nejmenší testovatelné části aplikace. Jde obvykle o testy jednotlivých komponent aplikace na úrovni modulů, objektů a tříd. Tento druh testování obvykle provádějí vývojáři a nebývá zahrnut do plánů testů aplikace. [29]

Unit testy byly psány autorem práce. Většina unit testů byla posléze převedena do automatického testování – CI.

5.4 Uživatelské testování

Uživatelské testování je jedním ze základních nástrojů ke zlepšení použitelnosti webových stránek. Slouží k ověření, jak naše stránky používají jejich skuteční uživatelé, nikoliv jak si to myslí jejich programátoři a návrháři. Nikdo totiž nenavrhne dokonalý web, ale právě pomocí uživatelského testování se mu můžeme méně či více přiblížit. [30]

Pro pět různých lidí rozličných věkových kategorií byl vytvořen scénář s pokyny, na kterých bylo napsáno, co mají s aplikací udělat. Konkrétně měli za úkol přidat nového zákazníka, upravit konkrétní prostor, zkusit převést jednotku apod. Kompletní scénář je k dispozici v příloze C.

Z těchto testů vyšlo najevo, že používání aplikace je více či méně intuitivní, což bylo cílem. Uživateli byly navrhnuty rozličné změny (zejména v grafice), přičemž jich ale bylo málo a rovněž bylo řečeno, že se nejedná o nutné změny, spíše o osobní preference a ty se mezi uživateli lišily.

Kapitola 6

Rozšiřitelnost

6.1 REST API

Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsat konkrétními daty). Všechny zdroje mají vlastní identifikátor URI. [31]

Pro tento typ softwaru je jistě rozumné vytvořit REST API. Je to z toho důvodu, že lze pak snadno vytvořit webovou, desktopovou i mobilní aplikaci naráz. Důvodem, proč toto API nebylo vytvořeno rovnou, byl zejména nedostatek času. Dále v zadání je řečeno, že má jít o prototyp webové aplikace, takže to nebylo nutné. Pokud se měl tento prototyp rozšířit, je REST API dobrou volbou.

6.2 Databázový server

Jelikož se jedná o aplikaci psanou v Djangu, velmi dobře se pracovalo s databází SQLite. Avšak pokud by program využívala větší firma, je dost časté, že tyto velké firmy mají samostatné databázové servery. Bylo by tak dobré v rámci rozšíření vytvořit spojení s nějakým samostatným databázovým serverem. V této práci to však nebylo nutné ani vyžadované.

6.3 Vícejazyčnost

Jelikož se jedná o typ softwaru, který by se mohl potenciálně používat v jakékoliv zemi, bylo by vhodné vytvořit aplikaci tak, aby podporovala více jazyků. Jelikož bylo jedním nefunkčním požadavkem vytvořit samotnou aplikaci v češtině, nebylo nutné ji vytvořit pro více jazyků. Pro možné rozšíření je to ale možná myšlenka.

6.4 Účetnictví

Samotné účetnictví není původní podstatou této aplikace. V reálu však někteří zaměstnanci, kteří by potenciálně do aplikace zasahovali, mají na starosti účetnictví přímo spojené se spotřebami energií. Nabízí se zde nápad na rozšíření v podobě podrobnějšího účetnictví. Samotná položka „nájem“ v databázové tabulce *smlouva* zahrnuje velkou část účetnictví, která ve výsledku vytvoří číslo, které je zapsané v tabulce.

Závěr

Cílem této práce bylo vytvořit prototyp aplikace pro rozúčtování nájemného a energií, která by mohla potenciálně pomáhat zaměstnancům nějaké firmy pronajímající obytné prostory. Cílem bylo vytvořit takový prototyp aplikace, jaký by zaměstnancům umožnil intuitivní manipulaci s daty.

Na začátku dostal autor práce možnost komunikovat s potenciálními uživateli softwaru. Na základě diskuzí a rozhovorů byly analyzovány funkční a nefunkční požadavky, případy užití a podle jistých kritérií byly zvoleny technologie, které byly použity později při samotné implementaci.

Na základě analytických údajů byl později vytvořen návrh aplikace. První vznikl doménový model. Dále byly vytvořeny wireframy, opět na základě rozhovoru s potenciálními uživateli aplikace. S ohledem na zvolené technologie byla zvolena architektura systému.

Po analýze a návrhu byl vytvořen prototyp aplikace, který byl na závěr testován uživateli. Autor v poslední kapitole navrhuje úpravy a možná rozšíření, které by software udělaly snadněji udržovatelným a rozšířily případnou klientelu.

Bibliografie

1. SOMMERVILLE, Ian. *Softwarové inženýrství*. Computer Press, Albatros Media a.s., 2017. Č. 9788025145258.
2. BUREŠ, Miroslav; RENDA, Miroslav; DOLEŽEL, Michal; KOLEKTIV. *Efektivní testování softwaru: Klíčové otázky pro efektivitu testovacího procesu*. Grada Publishing a.s., 2016. Č. 9788027193899.
3. ROUDENSKÝ, Petr; HAVLÍČKOVÁ, Anna. *Řízení kvality softwaru*. Computer Press, Albatros Media a.s., 2017. Č. 9788025145197.
4. MICROSOFT CORPORATION. *C#* [software]. 2019. Verze 8.0. Dostupné také z: `https://docs.microsoft.com/en-gb/dotnet/csharp/`.
5. MICROSOFT CORPORATION. *.NET Framework* [software]. 2019. Verze 4.7.2. Dostupné také z: `https://docs.microsoft.com/en-gb/dotnet/framework/migration-guide/versions-and-dependencies`.
6. MICROSOFT CORPORATION. *Introduction to the C# Language and the .NET Framework* [online]. 2019. Dostupné také z: `https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework`. [cit. 2019-09-10] (překlad vlastní).
7. MICROSOFT CORPORATION. *ASP.NET MVC* [software]. 2019. Verze 5. Dostupné také z: `https://docs.microsoft.com/cs-cz/aspnet/mvc/`.
8. SATHAYE, Ninad. *MVC architecture* [online]. Packt Publishing Limited. Dostupné také z: `https://subscription.packtpub.com/book/application_development/9781785889196/10/ch10lvl1sec88/mvc-architecture`. [cit. 2019-09-13].
9. PYTHON SOFTWARE FOUNDATION. *Python* [software]. 2019. Verze 3.7.4. Dostupné také z: `https://www.python.org/`.

10. PYTHON SOFTWARE FOUNDATION. *What is Python? Executive Summary* [online]. 2019. Dostupné také z: `https://www.python.org/doc/essays/blurb/`. [cit. 2019-09-12] (překlad vlastní).
11. DJANGO SOFTWARE FOUNDATION. *Django* [software]. Verze 2.2. Dostupné také z: `https://www.djangoproject.com/`.
12. DJANGO SOFTWARE FOUNDATION. *Meet Django* [online]. Dostupné také z: `https://www.djangoproject.com/`. [cit. 2019-09-13] (překlad vlastní).
13. GOUR, Rinu. *Working Structure of Django MTV Architecture* [online]. Towards Data Science. Dostupné také z: `https://towardsdatascience.com/working-structure-of-django-mtv-architecture-a741c8c64082`. [cit. 2019-09-13].
14. SYNOPSIS. *What is software architecture?* [online]. Dostupné také z: `https://www.synopsys.com/glossary/what-is-software-architecture.html`. [cit. 2019-10-13] (překlad vlastní).
15. JAVATPOINT. *Django MVT* [online]. Dostupné také z: `https://www.javatpoint.com/django-mvt`. [cit. 2019-09-14].
16. SQLITE CONSORTIUM. *SQLite* [software]. Verze 3.25.1. Dostupné také z: `https://www.sqlite.org/index.html`.
17. L-PRODUCTION WEBDESING. *Což znamená design uživatelského prostředí?* [online]. Dostupné také z: `http://www.lproduction.cz/uzivatelske-prostredi-96.htm`. [cit. 2019-12-11] (překlad vlastní).
18. BOOTSTRAP TEAM. *Bootstrap* [software]. Verze 4.4. Dostupné také z: `https://getbootstrap.com/`.
19. TECHTARGET. *Bootstrap* [online]. Dostupné také z: `https://whatis.techtarget.com/definition/bootstrap`. [cit. 2019-12-12] (překlad vlastní).
20. GITLAB, INC.; OPEN-SOURCE KOMUNITA. *Gitlab* [software]. 2019. Verze 12.2.4. Dostupné také z: `https://about.gitlab.com/`.
21. JAKUB JIRŮTKA. *GitLab* [online]. Dostupné také z: `https://rozvoj.fit.cvut.cz/Main/GitLab`. [cit. 2019-12-22].
22. IT-SLOVNIK.CZ TEAM. *IDE* [online]. Dostupné také z: `https://it-slovnik.cz/pojem/ide`. [cit. 2019-12-22].
23. JETBRAINS S.R.O. *PyCharm* [software]. 2019. Professional 2019.3. Dostupné také z: `https://www.jetbrains.com/pycharm/promo/?gclid=Cj0KCQiAl5zwBRCTARIsAIrukdPiTj6mSL6-pQAyBsUoCfqLCaAFzMI7qMemT-uqfy_c88-Qb8kl_xgaAnMDEALw_wcB`.
24. TOPRANKER.CZ. *CO JE TO FRONTEND* [online]. Dostupné také z: `https://topranker.cz/slovnik/frontend/`. [cit. 2019-12-20].

25. ADAPTIC, S. R. O. *Backend* [online]. Dostupné také z: https://www.adaptic.cz/znalosti/slovnicek/backend/. [cit. 2019-12-20].

26. KITNER, Radek. *Testování softwaru* [online]. Dostupné také z: https://kitner.cz/slovnik/testovani-softwaru/. [cit. 2019-12-23].

27. ACKEE. *CI v Ackee* [online]. Dostupné také z: https://www.ackee.cz/blog/continuous-integration-pro-tvorbu-mobilnich-aplikaci/. [cit. 2019-12-23].

28. KITNER, Radek. *Integrační testování* [online]. Dostupné také z: https://kitner.cz/slovnik/integracni-testovani/. [cit. 2019-12-23].

29. SOFTWARE, Testování. *UNIT testy* [online]. Dostupné také z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=18:druhy-testovani-v-procesu-vyvoje-sw&catid=3:zaklady&Itemid=11. [cit. 2019-12-23].

30. HLAVÁČ, Jan. *Uživatelské testování na dálku* [online]. Dostupné také z: https://www.sherpas.cz/blog/newsletter-uzivatelske-testovani-na-dalku. [cit. 2019-12-25].

31. MALÝ, Martin. *REST: architektura pro webové API* [online]. 2009. Dostupné také z: https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/. [cit. 2019-12-25].

Příloha A

Seznam použitých zkratek

API Application programming interface

CI Continuous integration

CRUD Create read update delete

CSS Cascading style sheets

CSV Comma separated values

DTL Django template language

HTML Hypertext markup language

IDE Integrated Development Enviroment

MTV Model template view

MVC Model view controller

OOP Object-oriented programming

ORM Object relational mapping

REST Representational State Transfer

UI User interface

URL Uniform resource locator

Příloha B

Detail případů užití

UC1: Přidání nájemce – zaměstnanec na záložce *Obytné prostory* je schopen přidat nového nájemce s příslušnými parametry.

1. *Include* (UC2: Zobrazit seznam nájemců a prostor)
2. Zaměstnanec zvolí tlačítko *Přidat* v seznamu *Nájemci*
3. Systém zobrazí formulář k vyplnění údajů
4. Zaměstnanec jej vyplní a stiskne tlačítko *Přidat*
5. Systém přesměruje zaměstnance na původní stránku

UC2: Zobrazit seznam nájemců a prostor – zaměstnanec si může prohlédnout seznam všech nájemců a prostor.

1. *Include* (UC6: Přihlášení)
2. Zaměstnanec stiskne navigační tlačítko *Obytné prostory*
3. Systém zobrazí záložku *Obytné prostory*

UC3: Filtrování nájemců – zaměstnanec má možnost si seznam s nájemci filtrovat na základě objektu, ve kterém nájemci bydlí. Dále pak podle toho, zda má nájemce aktivní smlouvu o pronájmu.

1. *Extends* (UC2: Zobrazit seznam nájemců a prostor)
2. Zaměstnanec může vyplnit v horním panelu *Budovu, vlastníka, ulici* a případně označit pole pro zobrazení nájemců bez aktivní smlouvy o pronájmu
3. Zaměstnanec stiskne tlačítko *Filtruj*
4. Systém zobrazí seznam splňující kritéria

UC4: Přidání prostoru – zaměstnanec na záložce *Obytné prostory* je schopen přidat nový prostor s příslušným typem prostoru a případným přiřazením k obytnému objektu.

1. *Include* (UC2: Zobrazit seznam nájemců a prostor)
2. Zaměstnanec zvolí tlačítko *Přidat* v seznamu *Prostory*
3. Systém zobrazí formulář k vyplnění údajů
4. Zaměstnanec jej vyplní a stiskne tlačítko *Přidat*
5. Systém přesměruje zaměstnance na původní stránku

UC5: Filtrování prostor – zaměstnanec má možnost si zobrazit prostory, které budou splňovat volitelná kritéria (budova, vlastník, ulice, patro, obsazenost).

1. *Extends* (UC2: Zobrazit seznam nájemců a prostor)
2. Zaměstnanec může vyplnit v horním panelu *Budovu, vlastníka, ulici*, zvolit patro (pokud hledá prostory „podřazené“ patru) a obsazenost prostoru
3. Zaměstnanec stiskne tlačítko *Filtruj*
4. Systém zobrazí seznam splňující kritéria

UC6: Přihlášení – pokud chce zaměstnanec používat program, musí se prokázat přihlašovacími údaji. Registraci uživatelů řeší administrátor aplikace.

1. Zaměstnanec chce spustit webovou aplikaci
2. Systém zobrazí formulář pro vyplnění přihlašovacích údajů
3. Zaměstnanec vyplní údaje a klikne na *Přihlásit*
4. Systém přihlásí uživatele a přesměruje na záložku *Obytné prostory*

UC7: Úprava nájemce – zaměstnanec má možnost upravit údaje u již existujícího nájemce.

1. *Include* (UC2: Zobrazit seznam nájemců a prostor)
2. Zaměstnanec vybere se seznamu nájemce
3. Systém zobrazí formulář s údaji o nájemci
4. Zaměstnanec upraví údaje a klikne na *Uložit*
5. Systém přesměruje zaměstnance na původní stránku

UC8: Přidání měřicího zařízení – zaměstnanec může zaevidovat nové měřicí zařízení.

1. *Include* (UC11: Zobrazení seznamu měřidel)
2. Zaměstnanec zvolí tlačítko *Přidat* v seznamu *Zařízení*
3. Systém zobrazí formulář k vyplnění údajů

4. Zaměstnanec jej vyplní a stiskne tlačítko *Přidat*
5. Systém přesměruje zaměstnance na původní stránku

UC9: Úprava měřicího zařízení – zaměstnanec má možnost upravit údaje u měřidla.

1. *Include* (UC11: Zobrazení seznamu měřidel)
2. Zaměstnanec vybere se seznamu měřidlo
3. Systém zobrazí formulář s údaji o měřicím zařízení
4. Zaměstnanec upraví údaje a klikne na *Uložit*
5. Systém přesměruje zaměstnance na původní stránku

UC10: Zadání odečtu do měřicího zařízení – zaměstnanec může zadat odečet do měřicího zařízení.

1. *Include* (UC11: Zobrazení seznamu měřidel)
2. Zaměstnanec vybere se seznamu měřidlo a stiskne *Odečet*
3. Systém zobrazí formulář s údaji o měřicím zařízení
4. Zaměstnanec upraví údaje a klikne na *Uložit*
5. Systém přesměruje zaměstnance na původní stránku

UC11: Zobrazení seznamu měřidel – zaměstnanec může zobrazit záložku se seznamem měřidel.

1. *Include* (UC6: Přihlášení)
2. Zaměstnanec klikne na *Měřicí zařízení*
3. Systém zobrazí záložku s měřidly

UC12: Filtrování měřidel – zaměstnanec má možnost zobrazit pouze měřidla, která splňují vybraná kritéria.

1. *Extends* (UC11: Zobrazení seznamu měřidel)
2. Zaměstnanec může zvolit typ zařízení a budovu, dále pak zaškrtnout, zda chce pouze hlavní měřidla a jestli chce vidět vyřazená měřidla
3. Systém zobrazí seznam splňující kritéria

UC13: Přidání smlouvy – do systému je možnost zadat novou smlouvu.

1. *Include* (UC14: Zobrazení smluv)
2. Zaměstnanec stiskne tlačítko *Přidat*
3. Systém zobrazí formulář pro vyplnění údajů smlouvy

4. Zaměstnanec vyplní údaje a stiskne tlačítko *Uložit*
5. Systém přesměruje zaměstnance na původní stránku

UC14: Zobrazení smluv – zaměstnanec může prohlédnout seznam se smlouvami.

1. *Include* (UC6: Přihlášení)
2. Zaměstnanec stiskne *Smlouvy*
3. Systém zobrazí záložku se smlouvami

UC15: Filtrování smluv – zaměstnanec může prohlédnout seznam se smlouvami v daném období.

1. *Extends* (UC14: Zobrazení smluv)
2. Zaměstnanec může zvolit období, ve kterém nabývá smlouva platnosti
3. Systém zobrazí příslušný seznam smluv

UC16: Zobrazení vyúčtování – zaměstnanec může zobrazit vyúčtování.

1. *Include* (UC6: Přihlášení)
2. Zaměstnanec zvolí *Vyúčtování*
3. Systém zobrazí záložku *Vyúčtování*

UC17: Filtrování vyúčtování – zaměstnanec má možnost vyhodnotit údaje o vyúčtování na základě parametrů.

1. *Extends* (UC16: Zobrazení vyúčtování)
2. Zaměstnanec vyplní parametry pro vyúčtování a stiskne *Vyhodnotit*
3. Systém vyhodnotí údaje na základě parametrů

UC18: Export vyúčtování do CSV – vyhodnocené údaje je možné exportovat do souboru ve formátu CSV.

1. *Extends* (UC17: Filtrování vyúčtování)
2. Zaměstnanec stiskne *Exportovat* a zvolí soubor, do kterého se údaje exportují
3. Systém uloží údaje do souboru

UC19: Zobrazení naměřených hodnot – zaměstnanec si může zobrazit stav jednotlivých měřidel, jejich hodnot a porovnat naměřené hodnoty na hlavním měřidle proti vedlejším.

1. *Include* (UC6: Přihlášení)

2. Zaměstnanec stiskne *Stav měřidel*
3. Systém zobrazí Stav měřidel

UC20: Filtrování naměřených hodnot – zaměstnanec může zobrazit měřicí zařízení a naměřené hodnoty.

1. *Extends* (UC19: Zobrazení naměřených hodnot)
2. Zaměstnanec může zvolit typ, budova, areál
3. Systém zobrazí údaje podle parametrů

UC21: Přidání jednotky – zaměstnanec může do systému zavést novou jednotku.

1. *Include* (UC22: Zobrazení jednotek)
2. Zaměstnanec klikne na přidat
3. Systém zobrazí formulář pro novou jednotku
4. Zaměstnanec zadá údaje a klikne na *Uložit*
5. Systém přesměruje zaměstnance na původní stránku

UC22: Zobrazení jednotek – zaměstnanec může procházet všechny jednotky.

1. *Include* (UC6: Přihlášení)
2. Zaměstnanec zvolí *Správa jednotek*
3. Systém zobrazí seznam s jednotkami

UC23: Editace koeficientu převodu – zaměstnanec může opravit koeficient převodu mezi jednotkami.

1. *Include* (UC22: Zobrazení jednotek)
2. Zaměstnanec zvolí jednotky
3. Systém zobrazí formulář pro převod
4. Zaměstnanec zadá koeficient a klikne na *Uložit*
5. Systém přesměruje zaměstnance na původní stránku

Příloha C

Uživatelské testování – scénář

1. Přihlaste se do aplikace s následujícími přihlašovacími údaji:

   Uživatelské jméno: Thomas
   Heslo: Angellino

2. Vložte nového nájemce:

   Jméno: Olga
   Příjmení: Kyselá
   Adresa: Šťastná 12/2

3. Přidejte novou jednotku:

   Název: MWh
   Převod na základní jednotku: 0,000001
   Typ: Elektřina

4. Převeďte 5MWh na Wh.

5. Zadejte nový odečet:

   Hodnota: 120
   Jednotka: wH
   Datum: 03.01.2020
   Typ: Elektřina

Příloha D

Obsah přiloženého USB

- `readme.txt` ..... stručný popis obsahu USB
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF