ŽILINSKÁ UNIVERZITA V ŽILINE

**FAKULTA ELEKTROTECHNIKY
A INFORMAČNÝCH TECHNOLÓGIÍ**

Katedra multimédií a informačno-komunikačných technológií

28260120222019

APLIKÁCIA PRE REZERVÁCIU ZDIEĽANÝCH PRACOVNÝCH MIEST (SHARE DESK)

2022 **Bc. Dominik Bullo**

ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta elektrotechniky
a informačných technológií

Katedra multimédií a informačno-komunikačných technológií

APLIKÁCIA PRE REZERVÁCIU ZDIEĽANÝCH PRACOVNÝCH MIEST (SHARE DESK)

DIPLOMOVÁ PRÁCA

Študijný odbor:	informatika
Študijný program:	multimediálne inžinierstvo
Vedúci diplomovej práce:	Ing. Kečkešová Barbora
Pracovisko vedúceho práce:	Ringier Axel Springer SK, a.s.
Konzultant:	Ing. Miroslav Markovič, PhD.
Pracovisko konzultanta:	Katedra multimédií a informačno-komunikačných technológií, FEIT, ŽU v Žiline

Žilina, 2022 **Bc. Dominik Bullo**

ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta elektrotechniky
a informačných technológií

Katedra multimédií
a informačno-komunikačných technológií

Akademický rok 2021/2022

ZADANIE DIPLOMOVEJ PRÁCE

Meno, priezvisko: **Bc. Dominik Bullo**

Študijný odbor: **informatika**
Študijný program: **multimediálne inžinierstvo**

Téma diplomovej práce: **Aplikácia pre rezerváciu zdieľaných pracovných miest (share desk)**

Pokyny na vypracovanie diplomovej práce:

- Analýza technológií pre vytváranie web aplikácií.
- Analýza požiadaviek na systém.
- Používatelia rôznych typov (zamestnanec, administrátor).
- Prihlásenie/Registrácia používateľov.
- Rozdeľovanie zamestnancov do tímov.
- Správa zdieľaných pracovných miest s vizuálnou reprezentáciou miest a dispozícií priestorov.
- Priraďovanie miest tímom ale aj jednotlivým zamestnancom (majú nárok na stále pracovné miesto).
- Možnosť realtime vizualizácie obsadenosti pracovných miest.
- Rezervovanie pracovných miest .
- Možnosť reportovať technické alebo iné problémy spojené s pracovným miestom.
- Analýza nefunkcionálnych požiadaviek, výber vhodných technológii (implementačný jazyk, framework, databáza).
- Implementácia, otestovanie a produkčné nasadenie.
- Vyhodnotenie prínosov.

Vedúci diplomovej práce: Kečkešová Barbora, Ing., Ringier Axel Springer SK, a.s.

Konzultant diplomovej práce: Markovič Miroslav, Ing. PhD., Katedra multimédií a informačno-komunikačných technológií, FEIT, ŽU v Žiline

Dátum odovzdania diplomovej práce: 02. 05. 2022

ŽILINSKÁ UNIVERZITA V ŽILINE
FAKULTA ELEKTROTECHNIKY
A INFORMAČNÝCH TECHNOLÓGIÍ
KATEDRA MULTIMÉDIÍ A INFORMAČNO-
-KOMUNIKAČNÝCH TECHNOLÓGIÍ
Univerzitná 8215/1, 010 26 Žilina
-1-

prof. Ing. Róbert Hudec, PhD.
vedúci katedry

V Žiline dňa 29. 10. 2021

ANOTAČNÝ ZÁZNAM

Žilinská univerzita v Žiline
Fakulta elektrotechniky a informačných technológií
Katedra multimédií a informačno-komunikačných technológií

Typ práce:	Diplomová práca
Meno a priezvisko:	Dominik Bullo
Akademický rok:	2021/2022
Názov práce:	Aplikácia pre rezerváciu zdieľaných pracovných miest (share desk)
Počet strán:	76
Počet obrázkov:	39
Počet príloh:	1
Počet použ. lit.:	11

Anotácia v slovenskom jazyku:

Táto práca obsahuje analýzu, návrh a implementáciu aplikácie slúžiacou ako nástroj na správu a rezerváciu pracovných miest na pracovisku. Je vytváraná ako webová aplikácia, ktorá je použiteľná odkiaľkoľvek a z ľubovoľného zariadenia. Základy tejto aplikácie na serverovej časti tvoria technológie Django a Django REST framework a na klientskej je to framework Vue.js.

Annotation in foreign language (english/deutsch):

This work contains analysis, design and implementation of an application serving as a tool for managing and reserving jobs in the workplace. It is created as a web application that is available at any time and from any device. The basis of this application on the server part is the Django and Django REST framework technologies and on the client part it is the Vue.js framework.

Kľúčové slová: webová aplikácia, Python, Django, JavaScript, Vue.js, share desk

Vedúci diplomovej práce:	Ing. Kečkešová Barbora
Konzultant ZP:	Ing. Miroslav Markovič, PhD.
Dátum odovzdania ZP:	02.05.2022

POĎAKOVANIE

Ďakujem vedúcej a konzultantovi diplomovej práce za odbornú pomoc a rady pri tvorení práce, ktoré mi pomohli prácu úspešne dokončiť. Rovnako tak ďakujem mojej rodine a priateľke za podporu a trpezlivosť, ktorú mi počas tvorby tejto práce preukazovali.

ABSTRAKT

Abstrakt v slovenskom jazyku

Cieľom diplomovej práce je analyzovať požiadavky zadávateľa na aplikáciu pre rezerváciu zdieľaných pracovných miest. Aplikáciu následne podľa týchto požiadaviek náležite navrhnúť. Vybrať vhodný typ aplikácie a technológie, ktoré sa pri vývoji aplikácií použijú. Súčasťou práce je aj implementácia aplikácie, ktorá bude spĺňať vyššie spomenuté požiadavky zadávateľa. Jej účelom bude primárne možnosť rezervovania si pracovného miesta jednotlivcov, alebo celých tímov. Aplikácia bude obsahovať rozdeľovanie do tímov, rôzne používateľské role, možnosť reportovať technické problémy spojené s pracovným miestom. Takáto aplikácia uľahčí zamestnancom prístup na pracovisku a zároveň prinesie zamestnávateľom možnosť lepšie využiť svoje priestory, či získať adekvátne dáta. Výsledkom praktickej časti je naprogramovaná webová aplikácia, rovnako tak aj produkčné nasadenie na verejne prístupe adrese a na vlastnom serveri. Aplikácia sa skladá z klientskej aj serverovej aplikácie.

Kľúčové slová: webová aplikácia, Python, Django, JavaScript, Vue.js

Abstract in foreign language (english/deutsch)

The aim of the thesis is to analyze the requirements of the seller for the application for the reservation of shared jobs. The application should then be properly designed according to these requirements. Select the appropriate application type and technology to use in application development. every job is also the implementation of an application that will need the above-mentioned requirements of the client. Its goal will be the primary possibility of reserving jobs for individuals or entire teams. The application will include division into teams, various user tasks, the ability to report technical issues associated with the job. Such an application will facilitate access in the workplace and at the same time bring the opportunity to make better use of your space or obtain the data obtained. if the practical part is a programmed web application, as well as production deployment on public access addresses and on its own server. The application consists of a client and a server application

Keywords/Schlüsselwörter: web application, Python, Django, JavaScript, Vue.js

OBSAH

Anotačný záznam i
Poďakovanie ii
Abstrakt iii
Abstrakt v slovenskom jazyku iii
Abstract in foreign language (english/deutsch) iii
Obsah iv
Zoznam obrázkov a tabuliek vii
Zoznam skratiek viii
Slovník termínov ix
Úvod 1
1 Cieľ 2
2 Analýza 4
2.1 Novovzniknutá situácia 4
2.2 Analýza existujúcich riešení 4
2.3 Analýza problému 4
2.4 Analýza požiadaviek 5
2.4.1 Mimofunkčné požiadavky 5
2.4.2 Multiplatformovosť 5
2.5 Analýza logických celkov aplikácie 6
2.5.1 Používatelia 6
2.5.2 Tímy 6
2.5.3 Pracoviská 6
2.5.4 Pracovné miesta a ich rezervácia 7
2.5.5 Nahlásenie problémov 7
2.6 Používateľské role a aktori v systéme 7
2.6.1 Rola zamestnanca 8
2.6.2 Rola administrátora 8
2.6.3 Rola superadministrátora 8
3 Návrh aplikácie 10
3.1 Voľne dostupná 10
3.2 Architektúra aplikácie 10
3.3 Klient-server architektúra 10
3.4 Komunikácia so vzdialenými službami 11

3.4.1 REST API ... 12
3.5 Návrh serverovej časti aplikácie ... 12
3.5.1 Prístup k dátam ... 13
3.5.2 MVC ... 13
3.6 Návrh klientskej časti aplikácie ... 14
3.6.1 Single-page aplikácia ... 14
3.7 Návrh databázy ... 15
3.7.1 Používatelia ... 16
3.7.2 Tímy ... 16
3.7.3 Pracoviská ... 16
3.7.4 Rezervácia ... 16
4 Implementácia ... 18
4.1 Štruktúra projektu ... 18
4.2 Vývoj serverovej aplikácie ... 19
4.2.1 Django ... 19
4.2.2 Štruktúra serverovej časti ... 20
4.2.3 ORM ... 21
4.2.4 Migrácie ... 21
4.2.5 Administrátorské rozhranie ... 21
4.2.6 Django REST Framework ... 22
4.2.7 JWT ... 23
4.2.8 Modely ... 24
4.2.9 Polymorfiznus ... 25
4.2.10 Views ... 26
4.2.11 Oprávnenia ... 28
4.2.12 Filtrovanie/ Zoraďovanie/ Vyhľadávanie ... 29
4.2.13 Serializátory ... 30
4.2.14 Koncové body ... 32
4.3 Vývoj klientskej aplikácie ... 34
4.3.1 Vue.js ... 34
4.3.2 Vuexy template ... 34
4.3.3 Tmavý mód ... 35
4.3.4 Štruktúra klientskej časti ... 35
4.3.5 JWT ... 36
4.3.6 Vuex ... 36
4.3.7 Axios ... 37

4.3.8 Používateľské práva ... 38
4.3.9 Internacionalizácia ... 38
4.3.10 Notifikácie ... 39
4.3.11 Detail na rezervovania pracovného miesta ... 40
4.3.12 Filtrovanie špecifického miesta ... 41
4.4 Problémy spojené s vývojom webovej aplikácie ... 41
4.5 Docker ... 42
4.5.1 Kontajnerizácia ... 42
4.5.2 Docker Compose ... 42
4.6 Celková štruktúra aplikácie ... 43
4.7 Implementácia systému a používateľského rozhrania ... 43
4.7.1 Registrácia ... 43
4.7.2 Menu ... 44
4.7.3 Zoznam používateľov ... 45
4.7.4 Vytváranie tímov ... 46
4.7.5 Vytvorenie pracovísk ... 48
4.7.6 Rezervácia miesta ... 48
4.7.7 Dlhodobé rezervovanie miesta ... 50
4.7.8 Problémy ... 51
4.7.9 Moje rezervácie ... 53
5 Verziovanie a produkčné nasadenie ... 54
5.1 Git ... 54
5.1.1 Git vetvy ... 54
5.2 Github Actions ... 55
5.3 Nasadenie na server ... 55
5.4 Spôsob produkčného nasadenia ... 56
6 Testovanie a možný rozvoj aplikácie ... 57
6.1 Unit testy ... 57
6.2 Možnosti rozvoja ... 58
Záver ... 59
Zoznam použitej literatúry ... 1
Čestné vyhlásenie ... 2
Prílohová časť ... 3
Zoznam príloh ... I
Príloha č. 1 – Diagram procesu aplikácie pri novej verzií na Github ... II

ZOZNAM OBRÁZKOV A TABULIEK

Obrázok 3.1 Architektúra klient-server [3] 11
Obrázok 3.2 Návrh databázy 15
Obrázok 4.1 Vizuálne zobrazenie štruktúry projektu 18
Obrázok 4.2 Pohľad na administrátorské rozhranie 22
Obrázok 4.3 Pohľad na spracovanie žiadosti serverom 23
Obrázok 4.4 Ukážka modelov 24
Obrázok 4.5 Detail modelu pre rezerváciu miesta 25
Obrázok 4.6 Zobrazenie využitia polymorfizmu v modeloch 26
Obrázok 4.7 Pohľad na ReservationViewSet 26
Obrázok 4.8 Rozšírenie view o akciu zobrazenia rezervácií používateľa 27
Obrázok 4.9 Zmena statusu rezervácie 28
Obrázok 4.10 Vlastné pravidlo pre oprávnenia 28
Obrázok 4.11 Vlastný filter pre rezervácie 30
Obrázok 4.12 Serializátor pre tímovú rezerváciu 31
Obrázok 4.13 Ukážka zloženia kontrolóra s využitím polymorfizmu 31
Obrázok 4.14 Základné rozdelenie koncových bodov 32
Obrázok 4.15 URL adresy pre pracoviská 33
Obrázok 4.16 Príklad koncového bodu pre miestnosti 33
Obrázok 4.17 Diagram fungovania Vuex [9] 37
Obrázok 4.18 Ukážka prekladového súboru 39
Obrázok 4.19 Označenie textu na frontend-e určeného na preklad 39
Obrázok 4.20 Ukážka notifikácií 40
Obrázok 4.21 Ukážka rezervovania miesta v aplikácií 40
Obrázok 4.22 Ukážka registrácie 44
Obrázok 4.23 Ukážka menu 45
Obrázok 4.24 Zoznam používateľov 45
Obrázok 4.25 Vytvorenie tímu 46
Obrázok 4.26 Pridanie používateľov do tímu 47
Obrázok 4.27 Rozhranie pre vytváranie pracovísk 48
Obrázok 4.28 Rezervácia pracovných miest svetlý mód 49
Obrázok 4.29 Rezervácia pracovných miest tmavý mód 49
Obrázok 4.30 Manažér rezervácií 51
Obrázok 4.31 Vytvorenie problému 52
Obrázok 4.32 Zoznam problémov 52
Obrázok 4.33 Pohľad na rezervácie prihláseného používateľa 53
Obrázok 5.1 Git vetvenie 54
Obrázok 5.2 Príklad správania CI/CD [11] 55
Obrázok 6.1 Základné nastavenie pred každým testom 57
Obrázok 6.2 Unit test pre kontrolu tokenu po uplynutí jednej hodiny 57

ZOZNAM SKRATIEK

Skratka	Anglický význam	Slovenský význam
SOAP	Simple Object Access Protocol	
REST	Representational state transfer	
SPA	Single-page application	
URL	Uniform Resource Locator	
JSON	JavaScript Object Notation	
HTML	HyperText Markup Language	Hypertextový značkovací jazyk
HTTP	Hypertext Transfer Protocol	Hypertextový prenosový protokol
MVC	Model-View-Controller	
MTV	Model-Template-View	
ORM	Object-Relational Mapping	
SQL	Structured Query Language	Štruktúrovaný vyhľadávací jazyk
JWT	JSON Web Token	
UX	User Experience	Užívateľský zážitok
UI	User Interface	Používateľské rozhranie
CSS	Cascading Style Sheets	Kaskádové štýly
ACL	Access Control List	Zoznam používateľský práv
i18n	I - eighteen letters -N	
API	Application Programming Interface	Rozhranie pre programovanie aplikácií

SLOVNÍK TERMÍNOV

Termín	Význam termínu
Framework	*Framework (aplikačný rámec) je softvérová štruktúra, ktorá slúži ako podpora pri programovaní a vývoji a organizácii iných softvérových projektov.*
Frontend	*Časť stránky alebo aplikácie, ktorú vidí návštevník alebo používateľ*
Backend	*Časť stránky alebo aplikácie, ktorá slúži k administrácii webu alebo zabezpečuje jej logiku*
Open Source	*Počítačový softvér, ktorého zdrojový kód je verejne dostupný*
localStorage	*HTML API umožňujúca webovým aplikáciám ukladať dáta do webového prehliadača na strane používateľa*
branch	*Vo všeobecnosti je vývojová vetva, ktorá rozdeľuje kód. Je podobná vetve stromu.*

ÚVOD

Vnímajúc fakt, že svet sa pomaly začína znovu ekonomicky zotavovať, spoločnosti sa rozhodujú o tom, ako bude vyzerať pracovisko po období ovplyvnenom pandémiou COVID-19. Zatiaľ čo niektoré organizácie sa snažia vrátiť svojich zamestnancov späť do kancelárií, iné sa avšak rozhodujú pre hybridné modely alebo zostávajú pri práci na diaľku na plný úväzok.

Signifikantným benefitom hybridného modelu práce je ušetrený čas, peniaze, či zníženie stresu z dochádzania. Eliminácia fyzického dochádzania do práce v praxi môže znamenať vyriešenie problému vo forme nadmerného strávenia času v dopravnej zápche alebo možného riešenia frekventovaného meškania spojov verejnej dopravy.

Je dôležité poznamenať, že súčasne to neznamená, že zamestnanci sa už nikdy fyzicky neuvidia. Skôr to poukazuje na fakt, že obchodný model práce od 8:00 do 16:00 od pondelka začína čeliť značným pochybnostiam či nevôle zamestnancov. Práca na diaľku nie je vyhovujúca len pre zamestnancov – má preukázané výhody aj pre zamestnávateľov.

Toto vytvára veľký potenciál na zavedenie zmien formou hybridného systému. Takýto systém práce umožňuje zamestnancovi pracovať z domu, rovnako ale aj ponúka možnosť prísť do zamestnania a vykonávať prácu na pracovisku Môže to byť pár krát do roka, mesiaca či týždňa. Zamestnávateľ v takomto prípade ale musí rátať aj s tým, že sa môžu vyskytnúť zamestnanci, ktorí budú prejavovať záujem pracovať denne z kancelárie. Výsledkom takéhoto nastavenia pracoviska je, že zamestnávateľ nepotrebuje mať tak veľký počet pracovných miest ako mal doteraz a môže tento priestor využiť iným Spôsobom.

Vzhľadom na to, že pri takomto modely sa predpokladá, že počet miest na prácu bude menší ako počet zamestnancov, je potrebné zabezpečiť, aby zamestnanci pri príchode do práce mali svoje miesto garantované. Najvhodnejším riešením je taký druh systému, ktorý by jednotlivým zamestnancom zabezpečil jednoduché rezervovanie takéhoto miesta odkiaľkoľvek.

Mnohé firmy v takom prípade siahnu po aplikácií, ktorá by im toto pomohla dosiahnuť. Účelom tejto aplikácie je zefektívniť prideľovanie pracovných miest, lepší prehľad o vyťaženosti ich pracoviska. Následne môžu napríklad benefitovať z priestorov ktoré firma momentálne vlastní. Napríklad ich zmenšením, prípadne prenájmom inej firme.

1 CIEĽ

Cieľom tejto práce je zabezpečiť kompletný vývoj aplikácie pre rezerváciu zdieľaných pracovných miest, ktorá by spĺňala požiadavky spoločnosti Ringier Axel Springer Slovakia, a. s, ktorá vznikla primárne kvôli prechodu na tzv. hybridné pracovisko. Tento cieľ si rozdelíme na viaceré menšie celky, s ktorými sa nám následne bude jednoduchšie a efektívnejšie pracovať.

Prvým čiastkovým cieľom je analyzovať požiadavky spoločnosti na danú aplikáciu. Preskúmať funkčné, ale aj mimofunkčné požiadavky, ktoré by naša aplikácia mala spĺňať. Rovnako tak analyzovať procesy vo firme, na zabezpečenie správneho fungovania aplikácie. Následne na základe tejto analýzy vhodne navrhnúť typ aplikácie, použité technológie, implementačný jazyk či databázu.

Následne sa pozrieme na samotnú funkcionalitu a popíšeme si jednotlivé celky do detailov a to napríklad z rôznych používateľských rolí a ich pohľadom na aplikáciu. K základným funkcionalitám teda bude patriť rozdeľovanie zamestnancov do tímov. Každý zamestnanec bude priradený do tímu z dôvodu, že tieto tímy budú mať tiež možnosť rezervovať si pracovné miesta.

Ďalším cieľom tejto aplikácie je správa zdieľaných miesta priraďovanie miest zamestnancom. To znamená, že ak sa zamestnanec rozhodne ísť do práce bude mať možnosť prihlásiť sa do aplikácie. Následne uvidí jednotlivé pracovné miesta a v ďalšom kroku si vybrané miesto dokáže rezervovať pre seba a to na určitý termín. Takúto možnosť budú mať ale aj tímy, ktoré si budú taktiež vedieť zarezervovať niekoľko pracovných miest, či už pre používateľov softvéru, alebo aj pre hosťov.

Pre lepšiu prehľadnosť takéhoto systému je veľmi podstatnou funkciou vizualizácia takýchto miest a jednoduchá orientácia v používateľskom rozhraní. Takýto dizajn nesie so sebou výhody v podobe ušetreného času, či vyhnutie sa možného odporu voči rezerváciám. Rovnako slúži aj na to, aby zamestnanci, tak aj zamestnávateľ videli v reálnom čase obsadenosť jednotlivých pracovných miest. Tieto dáta potom môžu využiť ďalej.

Nemenej dôležitou funkcionalitou je, že firma sa o takéto miesta ale musí aj starať a napríklad pri nejakom probléme s týmto miestom chce byť o tom informovaná, aby následne mohla podstúpiť kroky na jeho odstránenie. Preto naša aplikácia bude

obsahovať možnosť miesto nie len rezervovať, ale aj nahlásiť problémy, ktoré sa na ňom vyskytujú.

V neposlednom rade naším cieľom je produkčné nasadenie aplikácie a s ním spojené úspešné uvedenie do testovacej prevádzky a sprístupnenie aplikácie na verejne dostupnom mieste.

Nasledujúca kapitola nás oboznámi s analýzou samotnej aplikácie z pohľadu softvérového návrhu rovnako tak aj implementácii programu s rôznymi grafickými ukážkami. Popíšeme si ako prebiehalo nasadenie aplikácie na server a takisto výsledok aplikácie.

2 ANALÝZA

Táto časť práce je zameraná na analýzu problematiky, ktorú daná firma potrebuje vyriešiť. Zameriame sa hlavne na požiadavky, ktoré boli dodané zo strany zadávateľa. Rovnako tak ale nemôžeme zabúdať aj na funkcionality, ktoré síce zamestnávateľ nežiadal, ale sú v súlade s dnešnými štandardmi vývoja aplikácií. Či už sa jedná o dizajn, bezpečnosť, rýchlosť či možnosť aplikáciu ďalej rozširovať.

2.1 Novovzniknutá situácia

V doterajšom modely spoločnosti bolo zaužívané, že každý zamestnanec mal pridelené svoje vlastné miesto. Toto miesto mohol využívať na každodennej báze a bolo len jeho. Tento model fungoval aj počas pandémie, kedy boli niektorí zamestnanci nútení k práci z domu.

Takéto fungovanie nie je ekonomicky efektívne a zamestnanec má požiadavku na aplikáciu, ktorá by im pomohla problém vyriešiť.

Aplikácia by mala zabezpečovať jednoduchú a prehľadnú funkcionalitu a jej prínos bude nie len pre zamestnancov, ale aj zamestnávateľov.

Hlavným problémom je rezervácia pracovných miest, ktorá momentálne nie je nijako možná. Preto sa zadávateľ rozhodol pre aplikáciu, ktorá by im takúto situáciu uľahčila.

2.2 Analýza existujúcich riešení

Na trhu existujú aplikácie, ktoré takýto problém riešia, no pre požiadavky firmy a následne vyvíjanie aplikácie sú takéto riešenia zbytočne robustné. Rovnako tak ak by sme chceli do tejto aplikácie doprogramovať niečo špecifické, je potrebné za to zaplatiť extra mesačné poplatky.

Pretože firma je technologicky zameraná a má svojich vlastných programátorov, z rozhovorov vyplynulo, že vlastný softvér by si chceli a vedeli aj naďalej spravovať samy.

2.3 Analýza problému

Tento koncept rezervovania pracovných miest je známy už dávnejšie. Primárne ho využívali takzvané *zdieľané* pracoviská. Takže takýto nápad nie je síce nový, napriek

tomu si žiada analyzovanie pre konkrétne potreby spoločnosti. Tento fakt nám ale pomôže v lepšom pochopení problematiky.

Analýzu si rozdelíme na menšie logické celky, ktoré si navrhneme so snahou o čo najviac samostatné komponenty. Tieto komponenty potom budú navzájom spolu komunikovať.

2.4 Analýza požiadaviek

Zo strany zadávateľa sme dostali pár základných pokynov, ktoré sú z ich strany požadované v našej aplikácií. Požiadavky ktoré sme dostali zo strany firmy si takisto neskôr rozdelíme do logických celkov, ktoré počas vývoja aplikácie budeme postupne vytvárať a pridávať do aplikácie.

2.4.1 Mimofunkčné požiadavky

Jednou z najdôležitejších mimofunkčných požiadaviek je, aby bola aplikácia jednoducho dostupná pre každého zamestnanca, na ktoromkoľvek jeho zariadení (mobil, tablet, desktop a pod.) bez nutnosti niečo si inštalovať. Takýto spôsob aplikácií je v dnešnej dobe čoraz viac populárny.

V takomto prípade je častou voľbou webová aplikácia, ktorá je nasadená na serveri a je verejne prístupná. Netreba ale zabúdať, že pri takomto návrhu je vysoko žiadané, aby údaje boli prístupné len používateľom aplikácie so správnymi používateľskými právami.

Ďalšou mimofunkčnou požiadavkou je moderné a responzívne používateľské rozhranie, ktoré zabezpečí príjemný pocit z používania aplikácie. Samozrejmosťou v dnešnej dobe je responzívny dizajn, ktorý vie prispôsobiť zobrazenie veľkosti obrazovky.

2.4.2 Multiplatformovosť

V predchádzajúcej časti sme pri našej analýze zistili, že je potrebné navrhnúť systém, s ktorým žiadny z používateľov nebude mať problém, a bude sa k nemu môcť dostať odkiaľkoľvek na svete a z ľubovoľného zariadenia.

V našom dizajne nemôžeme zanedbať dôležitosť mobilných zariadení. Vzhľadom na to, že samotná spoločnosť Google sa od 01.07.2019 rozhodla indexovať svoje vyhľadávania prevažne na základe mobilnej verzie. To len nasvedčuje trendu, že mobilné zariadenia sú čoraz viac populárnejšie. Ďalším dôkazom našej dedukcie je fakt, že v roku 2020 bolo globálne percentuálne zastúpenie používania mobilu 68%, používania tabletu 3% a používania desktopu 29% [1] .

Predpokladáme, že používatelia budú v našej aplikácií preferovať mobilné zariadenie. To ale neznamená, že aplikácia bude výlučne mobil, pretože predpokladáme, že v aplikácií budú určite činnosti primárne vykonávané na počítači či notebooku. Takouto činnosťou môže byť organizovanie rozložení miestností. Rovnako tak aj správa administrátormi by bola na mobile veľmi nepraktická.

Preto by sme pre našu aplikáciu potrebovali niečo platformovo nezávislé s možnosťou používania aplikácie odkiaľkoľvek, kedykoľvek a na ľubovoľnej platforme.

2.5 Analýza logických celkov aplikácie

Aplikáciu si teda môžeme rozdeliť na jednotlivé logické celky. Takéto rozdelenie nám pomôže v lepšom prehľade. Či už to bude v kóde, v aplikácií samotnej alebo v našej teoretickej časti práce.

Z programátorského hľadiska by niektoré časti mohli byť spojené, ale keďže je v záujme jednoduchá možnosť rozšírenia, tieto časti budú samostatné. Pri takomto rozhodnutí bude v budúcnosti možné relatívne jednoducho rozširovať aplikáciu o ďalšie funkcionality, bez väčšieho zásahu do už existujúcej štruktúry.

2.5.1 Používatelia

Základom našej aplikácie bude modul používateľov. Tento modul sa bude sústrediť primárne na to, aby bolo zabezpečené že sa používatelia vedia registrovať a prihlásiť.

Rovnako tak úlohou tohto modulu bude zabezpečiť potrebné práva pre používateľov. V našom prípade to budú používatelia viacerých typov a to konkrétne administrátor a zamestnanec.

2.5.2 Tímy

Keď sme používateľom zabezpečili prístup do aplikácie a práva, ešte je potrebné ich rozdeliť do tímov. To nám zabezpečí tento modul. V ňom sa budú dať vytvárať jednotlivé tímy a rovnako tak priraďovať používateľov do týchto vytvorených tímov. Dôležité je spomenúť, že jeden používateľ môže mať aj viacero tímov.

2.5.3 Pracoviská

Pre potreby aplikácie ešte predtým ako bude možné rezervovať pracovné miesta, najskôr musia byť vytvorené. Preto v tomto module budeme potrebovať vytváranie jednotlivých

pracovísk a pracovných miest. To zahŕňa najmä budovy v ktorých sú tieto pracoviská umiestnené, jednotlivé poschodia týchto budov a následne miestnosti.

Nemôžeme zabudnúť aj na vytváranie približného vizuálneho rozloženia v jednotlivých miestnostiach, pre rýchlu a jednoduchú orientáciu v aplikácií.

2.5.4 Pracovné miesta a ich rezervácia

Keď takéto pracovné miesta budú vytvorené, ďalším modulom aplikácie je samotná rezervácia pracovného miesta na pracovisku. Rezervácia pracovného miesta, bude musieť obsahovať konkrétny čas a dátum rezervácie, aby bolo jasné, že táto rezervácia nie je na stálo ale len na určitý čas.

Netreba ale zabúdať na to, že jedna z požiadaviek je dlhodobé rezervovanie miesta resp. stále pracovné miesto pre zamestnanca, preto pri navrhovaní aplikácie musíme plánovať aj s takouto možnosťou.

Rezervácia pracovného miesta môže byť určená buď pre používateľa, alebo pre tím. To znamená, tím si bude môcť zvoliť na rezerváciu viacero pracovných miest. Tento počet miest môže presahovať počet používateľov v tíme. Je to komunikované so zamestnávateľov a toto rozhodnutie bude slúžiť hlavne pre situácie kedy by mal tím (prípade používateľ) pozvaných hostí a chcel by si rezervovať miesto/miestnosť aj pre týchto hostí. Je žiadané takúto funkcionalitu zabezpečiť.

2.5.5 Nahlásenie problémov

Finálnym celkom je nahlásenie problémov spojených s daným miestom. Keďže sa predpokladá, že sa objavia problémy spojené s pracovnými miestami. V takomto prípade, je potrebné aby v našej aplikácií bola možnosť nahlásenia takéhoto problému s príslušnými informáciami o ňom.

2.6 Používateľské role a aktori v systéme

V našom systéme budeme mať niekoľko úrovní používateľov. Či už z dôvodu požiadavky zadávateľa, alebo samotného dizajnu aplikácie. Títo používatelia budú mať spoločnú funkcionalitu a následne podľa príslušnej role budú mať buď nejakú navyše, alebo niektorú odobratú.

2.6.1 Rola zamestnanca

Ako prvé sa pozrieme na rolu zamestnanca. Táto rola bude rola s najmenším počtom funkcií. Napriek tomu, túto rolu budú mať ako predvolene nastavenú všetci používatelia pri ich vytvorení.

Takýto používateľ bude mať ale niekoľko potrebných funkcií a to najmä:

- Rezervácia pracovného miesta.
- Nahlásenie problému spojeného s pracovným miestom.
- Zobrazenie všetkých tímov.
- Zobrazenie všetkých používateľov.

2.6.2 Rola administrátora

Rola administrátora bude zahŕňať všetku funkcionalitu, ktorú má zamestnanec a zároveň bude mať funkcie navyše, ktoré mu umožnia používať ale aj spravovať aplikáciu. Medzi významné funkcie ktoré bude mať táto rola patrí:

- Pridanie používateľa do tímu.
- Vytváranie rozložení jednotlivých miestností.
- Prijímanie a zamietanie permanentných rezervácií.

2.6.3 Rola superadministrátora

Rola superadministrátora je predvolená v Djangu a daná samotným princípom programovania v tomto *framework-u*. Takýto používateľ ma úplný prístup ku všetkým častiam systému. Pomocou administrácie môže dokonca meniť hodnoty v databázových tabuľkách, alebo prepísať užívateľmi zadané údaje. Takýto používateľ býva v aplikáciách spravidla len jeden jediný.

Aplikácia vie fungovať bez zmeny funkcionality aj bez takéhoto používateľa, ale zabezpečenie takejto role nám pomôže s riešením problémov v aplikácií, ktoré by sa mohli v budúcnosti objaviť. Okrem toho takáto rola zabezpečí nasledovné funkcie:

- Vytváranie budov.
- Vytváranie poschodí.
- Vytváranie miestností.

- Oprava chybných dát pri používaní aplikácie.

3 NÁVRH APLIKÁCIE

V tejto časti našej práce sa podrobnejšie pozrieme na návrh aplikácie na základe požiadaviek a našej analýzy. Popíšeme si podrobnejšie technickú časť, rovnako tak aj rozhodnutia, ktoré nás viedli k takémuto návrhu.

3.1 Voľne dostupná

Prvým krokom v našom návrhu je, že aplikáciu budeme vyvíjať ako *Open Source* aplikáciu. Takýto typ aplikácie má svoj zdrojový kód verejne prístupný a umožňuje jeho študovanie, prípadne vkladanie zmien a vylepšení do zdrojového kódu.

Z tohto rozhodnutia vyplýva ešte jedna skutočnosť a to, že aplikácia bude programovaná v anglickom jazyku. Je to z viacerých dôvodov, ale jedným z najdôležitejších je, že programovanie v tomto jazyku je nepísaným štandardom vo svete. Takémuto zdrojovému kódu bude potom prakticky rozumieť ktorýkoľvek programátor vo svete.

Používatelia ale budú mať možnosť zvoliť si jazyk samotnej aplikácie. Na výber budú mať z anglického jazyka a slovenčiny.

3.2 Architektúra aplikácie

Od našej aplikácie vyžadujeme, aby používateľovi poskytovala relevantné dáta, aby si potrebné dáta uložila a aby zabezpečila logiku práce s týmito dátami. Tie musia byť zobrazené prehľadne, tak aby im používateľ jednoducho porozumel.

Tieto dáta získa klientska aplikácia z centralizovaného servera, ktorý bude prístupný pomocou viacerých inštancií webového prehliadača - klientov. To znamená, že aplikácia bude typu klient-server. Takýto typ architektúry je veľmi často používaný pre webové aplikácie v prostredí internetu.

V ďalších kapitolách sa preto pozrieme na to, akým spôsobom bude klient žiadať o dáta a akým spôsobom mu ich server poskytne.

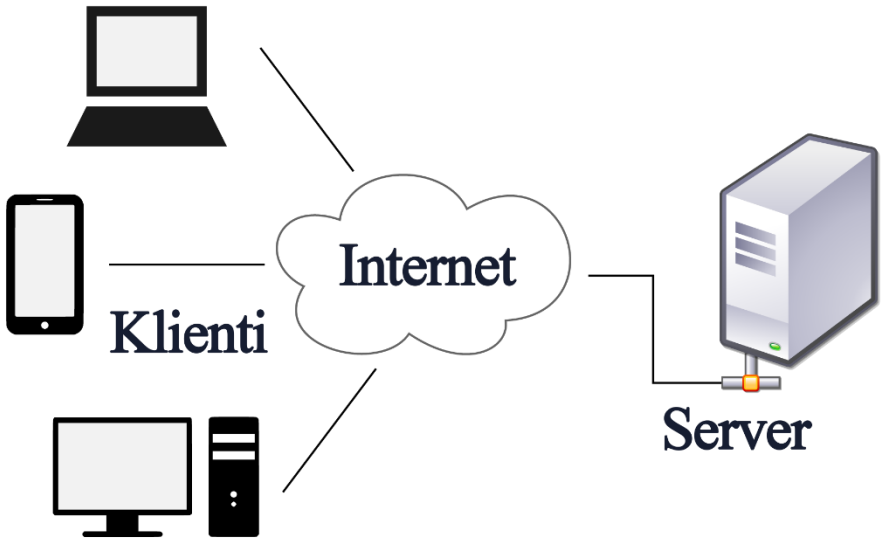
3.3 Klient-server architektúra

Architektúra klient-server je architektúra počítačovej siete, v ktorej mnoho klientov (vzdialených procesorov) požaduje a prijíma službu od centralizovaného servera (hostiteľského počítača). Klientske počítače poskytujú rozhranie, ktoré umožňuje

používateľovi počítača požadovať služby servera a zobrazovať výsledky, ktoré server vracia. Servery čakajú na požiadavky od klientov a následne na ne odpovedajú.

V ideálnom prípade server poskytuje klientom štandardizované transparentné rozhranie, takže klienti si nemusia byť vedomí špecifík systému (t. j. hardvéru a softvéru), ktorý poskytuje službu.

Klienti sú často umiestnení na osobných počítačoch či mobilných zariadeniach, zatiaľ čo servery sú umiestnené inde v sieti, zvyčajne na výkonnejších počítačoch. Tento výpočtový model je obzvlášť efektívny, keď klienti a server majú odlišné úlohy, ktoré bežne vykonávajú [2].



Obrázok 3.1 Architektúra klient-server [3]

3.4 Komunikácia so vzdialenými službami

Výber správnej technológie na komunikáciu medzi službami nie je úplne priamočiary. Existujú viaceré technológie, ktoré dokážu zabezpečiť komunikáciu medzi serverom a klientom. Medzi najznámejšie patrí *SOAP*, *REST* a *GraphQL*.

V prípade, že porovnávame tieto technológie, nachádzame ich výhody aj nevýhody. V našej aplikácii ale použijeme technológiu REST na komunikáciu klienta so serverom. Takýto typ komunikácie je častokrát používaná na podobné webové aplikácie. Dôvodov môže byť viac, ale najpodstatnejší je zrejme ten, že jej údaje sú v priebehu času relatívne konzistentné. Rovnako tak aj fakt, že je omnoho jednoduchšia než iné webové služby, ako napríklad *SOAP*. Oproti tomuto protokolu je navrhnutá tak, že dokáže zvládnuť

vyššie zaťaženie a spracovávať dáta rýchlejšie. Ďalším plusom pre tento typ komunikácie je, že je dlhodobo používaná a menej komplexná napríklad oproti *GraphQL*.

3.4.1 REST API

REST API je architektonický štýl pre komunikáciu klienta a servera. Skladá sa z dvoch slov *representational state transfer* (REST) a rozhrania pre programovanie aplikácií (API). Je to vlastne akýsi súbor definícií a protokolov postavený na princípe zdrojov (resources). Každý zdroj, ktorý môže byť čokoľvek od používateľov, tímov alebo rezervácií, je identifikovaný samostatnou adresou URL a môže byť vyzdvihnutý odoslaním jednoznačnej žiadosti GET. Na rozdiel od iných API neposkytuje REST žiadne špeciálne nástroje ani knižnice. Je to iba súbor pokynov ako získať prístup k všetkým potrebným údajom. To však v žiadnom prípade neznamená, že je to jeho nevýhodou. Najmä v prípade protokolu HTTP2 poskytuje výnimočne vysoký výkon [4].

Každá požiadavka je bezstavová, čo znamená, že server neukladá informácie o stave aplikácie. Namiesto toho požiadavky klientov obsahujú všetky informácie potrebné na vybavenie žiadosti.

Medzi najčastejšie používané metódy REST technológie patria, GET, POST, DELETE, PUT, PATCH. Keď vykonávame žiadosti na server pomocou týchto metód hypertextového prenosového protokolu (HTTP), server nám vráti odpoveď, z ktorej môžeme získať napríklad jej kód. Tieto HTTP kódy môžu byť následne využívané v našej klientskej časti aplikácie na ošetrenie zlyhaní, alebo oboznámení o úspešnom vykonaní žiadosti. Napríklad odpoveď 200 znamená, že požiadavka bola úspešná. Z odpovede takisto získame aj dáta, ktoré nám server poskytne, spravidla to bývajú súbory typu *JavaScript Object Notation* (JSON*)*. Ide o štandardizovaný formát ktorý obsahuje kľúč a hodnotu pre jednotlivé záznamy. Niektoré odpovede ale môžu obsahovať napríklad aj odkazy HTML, ktoré možno použiť na získanie ďalších informácií.

3.5 Návrh serverovej časti aplikácie

Ako sme si už opísali, v architektúre klient-server je server ten, ktorý má za účel logiku aplikácie a poskytovanie dát. Slúži ako prijímateľ požiadavky od klienta a následne ich spracuje a odpovedá. V tejto časti sa teda podrobnejšie pozrieme na časti serverovej aplikácie, ktoré tvoria jej základ.

3.5.1 Prístup k dátam

Ako už z návrhu vyplynulo, aplikácia bude obsahovať aj citlivé údaje, ku ktorým by sa nemal dostať ktokoľvek mimo tohto systému. To znamená musíme zabezpečiť napríklad to, že napríklad miesto si nezarezervuje výhradne len používateľ systému. Nie len táto časť, ale takmer všetky časti aplikácie budú dostupné len pre prihlásených používateľov, výnimkou je len pár stránok a to hlavne prihlásenie a registrácia.

Náš návrh počíta s tým, že používateľ, ktorý nemá správne autorizačné údaje alebo práva, nedostane prístup ku zdrojom, čo sa prejaví v stavovom kóde HTTP odpovede.

3.5.2 MVC

Model-View-Controller (MVC) je jeden z najčastejšie používaných architektonických vzorov v softvérovom dizajne. Bežne sa používa na implementáciu používateľských rozhraní, údajov a riadiacej logiky. Zdôrazňuje oddelenie medzi logikou a zobrazením.

Toto oddelenie by malo zabezpečiť lepšie rozdelenie úloh pre samotnú aplikáciu či zlepšenie údržby softvéru ako takého. Skladá sa z troch základných častí, ktoré je možné opísať nasledovne.

- **Model** definuje, aké dáta, resp. údaje má žiadosť obsahovať.
- **View** definuje, ako sa majú údaje zobrazovať.
- **Controller** obsahuje logiku, ktorá aktualizuje **model** alebo **view** v reakcii na žiadosť od používateľov aplikácie.

Tento návrhový model je častokrát je integrovaný v populárnych webových *framework-och*. Výnimkou nie je ani Django. Tento nástroj architektúru má už má v sebe zabudovanú. Konkrétne môžeme v prípade Djanga hovoriť o takzvanom Model-Template-View (MTV) modely. Tento model je logikou úplne rovnaký ako MVC, ale jednotlivé časti sú vo *framework-u* inak pomenované, preto je skratka rozličná. Bohužiaľ názov jedného komponentu prehodil, čo môže byť mätúce.

- **Model** - Modelom hovorí Django **Model**, v preklade model.
- **Views** - pohľadom hovorí Django **Template**, v preklade šablóna.
- **Controller** - kontrolérom hovorí Django **View,** v preklade pohľad.

Takýto návrhový model Djanga sa spravidla používa pri vývoji celej aplikácií len v Djangu. Kedže *framework* obsahuje aj systém na tvorbu HTML stránok. Pri takomto

type aplikácie slúžia zabudované *templates* ako súbory, ktoré Django potom vykreslí pre používateľa aplikácie.

Takýto typ aplikácie má ale svoje nedostatky a túto funkcionalitu v našej aplikácií nevyužijeme. Neexistuje k tomu žiadne pravidlo, ktoré by určovalo, či potrebujeme alebo nepotrebujeme ďalší *framework* na klientsku časť aplikácie, alebo nie. Napriek tomu sa v ďalšej kapitole pozrieme na benefity, ktoré to môže mať, a prečo by sme sa tým mali zaoberať.

Dôležité je v takomto prípade spomenúť, že ak sa rozhodneme používať Django ako len serverovú časť, budeme potrebovať aj nástroj *Django REST framework*, ktorý nám zabezpečí, že naša aplikácia sa bude správať podľa modelu MVC a jedoduchšie vytváranie rozhrania pre REST API. Pri jeho implementácií sa potom pozrieme na to, ako sa tieto žiadosti spracovávajú.

3.6 Návrh klientskej časti aplikácie

V predchádzajúcej kapitole sme už naznačili, že Django síce vie poskytnúť aj klientsku časť aplikácie. V tejto časti sa pozrieme na to, či je pre nás takéto riešenie vhodné, alebo existujú ešte nejaké iné možnosti, ako by sme klientsku časť aplikácie mohli navrhnúť.

3.6.1 Single-page aplikácia

Jednou z najnemenších a najčastejšie používaných techník vo vývoji webových aplikácií je takzvaná Single-page aplikácia (SPA). Toto je typ aplikácie ktorá funguje vo vnútri prehliadača a počas používania nevyžaduje opätovné načítanie stránky. Toto je jednou z najsilnejších výhod takéhoto typu aplikácie. Aplikácia sa načítava rýchlejšie bez nutnosti spracovania každej jednej stránky serverom pri interakcií používateľa s aplikáciou. Hlavný rozdiel teda spočíva v tom, že aplikácia sa stiahne len raz pri jej spustení.

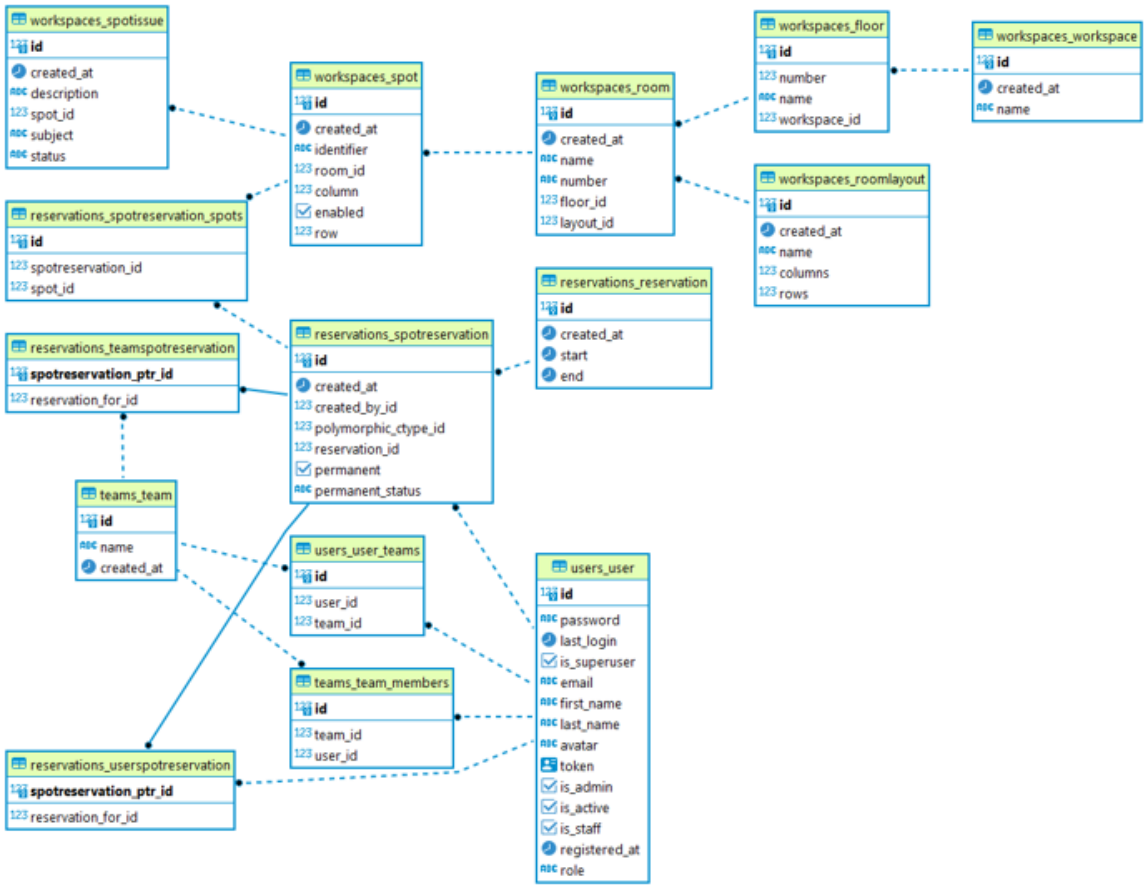
Výhodami takéhoto návrhu sú hlavne:

- Výkonnosť a rýchlejšia interakcia.
- Lepšia používateľská skúsenosť podobná zabudovaným aplikáciám.
- Nerušené používanie aj pri pomalšom internete, napríklad na mobilnom zariadení.

3.7 Návrh databázy

Ak chceme v našej aplikácií používať *framework* Django, ten nám automaticky poskytuje databázu SQLite. Takáto databáza je síce vhodná na testovanie, alebo na menšie aplikácie. Nie je ale zaužívané, že takáto databáza by sa používala na produkcií v plnej prevádzke, pri takom type aplikácie ako potrebujeme my. Preto v našej aplikácií použijeme databázu PostgreSQL. Takéto rozhodnutie nám zabezpečí, že v budúcnosti bude práca s databázou jednoduchšia, jej správa a záloha môže byť pomocou nástroja pgAdmin a pri viacerých používateľoch nebude spomaľovať našu aplikáciu.

Keď sme si určili typ databázy, ešte musíme databázu navrhnúť. Jej návrh by mal byť zložený s viacerých krokov. A to najmä zistiť potrebné informácie, ktoré budú uložené v databáze, rozdeliť tieto informácie do tabuliek (v našom prípade aj do logických celkov, pre Django modely), pridať vzťahy medzi jednotlivými tabuľkami a aplikovať normalizáciu.



Obrázok 3.2 Návrh databázy

Pri postupovaní týmito krokmi sme databázu rozdelili na 4 hlavné celky, pre ktoré ale nebudeme opisovať všetky tabuľky databázy podrobne. Priblížime si hlavne na logiku

týchto celkom a ozrejmime si časti, ktoré sú pre aplikáciu špecifické. Celý návrh môžeme vidieť na Obrázok 3.2.

Dôležité je ešte spomenúť, že pri implementácií takejto databázy nám pribudnú automaticky generované tabuľky z Djanga. Túto skutočnosť využijeme pri používateľoch.

3.7.1 Používatelia

Táto časť databázy sa bude skladať z tabuľky ***Users***. Ako sme si už spomenuli vyššie, táto tabuľka je vytváraná Djangom. My túto skutočnosť využijeme vo svoj prospech. V tabuľke necháme údaje, ktoré sú potrebné pre *framework*, ale doplníme ich ešte o pár vlastných. Konkrétne doplníme stĺpec pre používateľskú rolu a tímy, kde sa budú zobrazovať jednotlivé tímy v ktorých je používateľ.

3.7.2 Tímy

Ďalšiu logickú časť našej databázy tvoria tímy. Keďže takáto požiadavka bola zo strany zadávateľa. Táto logická časť by sa v našej aplikácií dala spojiť aj pod celok používatelia. Vzhľadom na to, že sa predpokladá budúci vývoj tejto aplikácie, takéto rozhodnutie zabezpečí lepší prehľad a jednoduchšie rozširovanie o nové funkcionality. Obsahuje tabuľku ***Teams.***

3.7.3 Pracoviská

Kde naša aplikácia bude špecifická je práve modul pracoviská a rezervácie. Pri návrhu databázy rozdelíme logiku do týchto dvoch logických celkom. V časti pracoviská sa budú nachádzať tabuľky pre budovy, poschodia, miestnosti, či jednotlivé rozloženia týchto miestností a následnej konkrétne pracovné miesta. Rovnako tak sa tu bude nachádzať aj tabuľka s problémami spojenými s pracovným miestom.

3.7.4 Rezervácia

Tento logický celok bude zastrešovať všetko, čo sa týka konkrétnych rezervácií. To znamená, pre koho je rezervácia určená, či je to používateľ, alebo je to tím. Rovnako tak je potrebné aby rezervácia vedela, či je permanentná alebo nie. Samozrejmosťou je informácia o začiatku a konci rezervácie.

Pre tieto údaje sme zvolili samostatnú tabuľku a to z toho dôvodu, že v budúcnosti pri optimalizácií databázy by bolo možné jednotlivé miesta prepojiť vzťahom *many-to-many* k tejto tabuľke a pri dlhodobom používaní by sa databáza zbytočne nehromadila údajmi.

Náš návrh zahŕňa skutočnosť, že rezervácie budú môcť byť v ľubovoľných časoch, pre viac dní, alebo aj permanentne. Rovnako tak v našom návrhu zohľadňujeme skutočnosť, že rezervácia môže byť pre dve rôzne tabuľky a to konkrétne „**Users**“ alebo „**Teams**“

4 IMPLEMENTÁCIA

Táto kapitola našej práce bude poukazovať na použité technológie v našej aplikácií. Pozrieme sa detailnejšie na serverovú a klientsku časť. Vysvetlíme si základné princípy programovania vo vybraných technológiách, alebo samotnej práce s nimi. Pozrieme sa na výhody aj nevýhody a taktiež si viac opíšeme technickú stránku projektu. Od jeho celkovej štruktúry až po problémy, ktorým sme pri implementácií aplikácie museli čeliť a ich riešenia.

4.1 Štruktúra projektu

Obrázok 4.1 Vizuálne zobrazenie štruktúry projektu

Základnú štruktúru projektu môžeme rozdeliť na 3 časti. Týmito časťami sú serverová časť, klientska časť a zvyšné potrebné súbory. Môže sa jednať o súbory potrebné pre správne fungovanie aplikácie ako napríklad súbory pre Docker a Docker Compose, alebo súbory ktoré umožňujú programátorom rôzne funkcie (code style, verziovanie, git, Github Actions a pod.) a množstvo ďalších.

Ako môžeme vidieť na Obrázok 4.1 takáto štruktúra zabezpečuje lepší prehľad v aplikácií. Zložka ***backend*** nám pokrýva serverovú časť a zložka ***frontend*** nám spravuje klientsku časť.

4.2 Vývoj serverovej aplikácie

Prvou časťou práce bolo vyvinúť serverovú aplikáciu, ktorá bude poskytovať údaje klientovi. Tieto údaje budú uložené v databáze, tak ako to vyplynulo z nášho návrhu.

Úlohou tejto časti aplikácie bude okrem iného prístup k týmto dátam a ich úprava, ako aj zabezpečenie celej logiky či administrátorského rozhrania.

V našom projekte túto časť nájdeme v zložke ***backend*** , kde sa nachádzajú všetky potrebné súbory pre serverovú časť aplikácie. Tá sa primárne skladá z *framework-u* Django v kombinácií s rôznymi doplnkami.

4.2.1 Django

Django je pomerne komplexný webový *framework*, ktorý ponúka všetko, čo v našej aplikácií budeme zo strany servera potrebovať. *Framework* je zbierka modulov a predpísaného kódu, ktorý pomáha vývojárom s konkrétnymi úlohami. Django za nás prevezme správu databázy, poskytne veľa užitočných funkcií a ponúkne svoju vstavanú administráciu. Jeho cieľom je podľa slov vývojárov flexibilita, jednoduchosť, škálovateľnosť a spoľahlivosť.

Tento nástroj sme použili kvôli jeho vlastnostiam, ktoré sa presne hodia na našu aplikáciu. Taktiež má za sebou dlhú históriu a vývoj, ktorý je dobrým ukazovateľom toho, že bude schopný v budúcom vývoji.

Popularita tohto *framework-u* je taktiež na vysokej úrovni a používa sa v širokom spektre vývoja aplikácií. Je to pravdepodobne aj vďaka jazyku Python, ktorý je pomerne jednoduchý na učenie. Rovnako tak aj fakt, že nám poskytuje nám užitočné zabudované funkcionality, ktoré nemusíme programovať nanovo, ako napríklad administrátorské rozhranie, modul objektovo-relačného mapovania (ORM) a pod.

O tom, že Django je naozaj populárny nástroj, svedčí aj fakt, že ho používajú aj niektoré veľké a známe spoločnosti, ako napríklad Instagram, Spotify, Dropbox, Mozzila, či Disqus.

4.2.2 Štruktúra serverovej časti

Ako sme už v našej práci spomenuli, stále sa držíme 4 logických celkov aplikácie. Takýto prístup sa odráža aj v štruktúre serverovej časti. Tá sa skladá z celkovo štyroch „aplikácií“ (názov Djanga - dal by sa prirovnať k balíčkom), pričom každá z nich sa snaží o čo najväčšiu samostatnosť. Vychádzame z odporúčaní vývojárov Django, keďže takéto rozdelenie a implementácia nám zabezpečí jednoduchú škálovateľnosť do budúcna a rovnako tak aj lepšiu prehľadnosť projektu.

```
[ShareDesk]          <- koreňový priečinok aplikácie
+---backend
|   |   manage.py
|   |   Pipfile               <- súbor obsahujúci všetky potrebné knižnice
|   +---apps
|   |   +---reservations
|   |   |   |   admin.py
|   |   |   |   apps.py
|   |   |   |   models.py
|   |   |   +---api
|   |   |   |   |   permissions.py
|   |   |   |   |   serializers.py
|   |   |   |   |   urls.py
|   |   |   |   |   views.py
|   |   |   +---migrations    <- priečinok s vygenerovanými migráciami pre
                                  databázu
|   |   |   +---tests         <- priečinok pre unit test súbory
|   +---config
|   |   |   api.py
|   |   |   settings.py
|   |   |   urls.py
|   |   |   wsgi.py
```

V takomto rozdelení je každá aplikácia plne zodpovedná za svoju logickú časť. Vyššie môžeme vidieť príklad jednej aplikácie, konkrétne rezervácie. Aplikácia obsahuje vlastné modely, smerovania, unit testy, či náhľady, špecifické pre tento celok. Toto rozloženie je preto veľkou výhodou, pretože aplikácie v prípade potreby dokážu navzájom medzi sebou komunikovať a funkcionalitu druhého modulu rozširovať, nie obmedzovať. V neposlednom rade, takéto aplikácie môžu byť znovu použité v iných projektoch, alebo naopak, jednoducho pridané do už existujúceho projektu.

4.2.3 **ORM**

Jednou z výhodou Djanga, ktorú v našej aplikácií využívame ORM. Toto je technika, ktorá programátorom umožňuje manipulovať a vyhľadávať údaje v databáze pomocou objektov.

V našej aplikácií ho využívame ako prekladač medzi databázou a objektovo orientovaným prístupom k modelom od Djanga. Výhodou takéhoto mapovania je fakt, že vývojári nemusia mať až tak veľkú znalosť práce s databázou, ako napríklad databázový špecialisti, napriek tomu môžu ľahko a jednoducho využívať množstvo zabudovaných funkcionalít.

V našom konkrétnom prípade Django ORM je spôsob, akým môžeme vytvárať *Structured Query Language* (SQL) príkazy pre databázu pomocou objektov a zabudovaných funkcií. Tieto funkcie sa spravila ovládajú resp. volajú pomocou špeciálneho súboru *manage.py.* Tento súbor so špecifickými argumentami dokáže spustiť aplikáciu, alebo vytvoriť a aplikovať migrácie databázy.

4.2.4 **Migrácie**

Migrácie databázy sú spôsob, akým Django šíri zmeny, ktoré vykonáte vo svojich modeloch (pridanie poľa, odstránenie modelu atď.) do schémy databázy. Sú navrhnuté tak, aby boli väčšinou automatické, ale musíte vedieť, kedy vykonať migráciu, kedy ich spustiť a aké bežné problémy môžete naraziť.

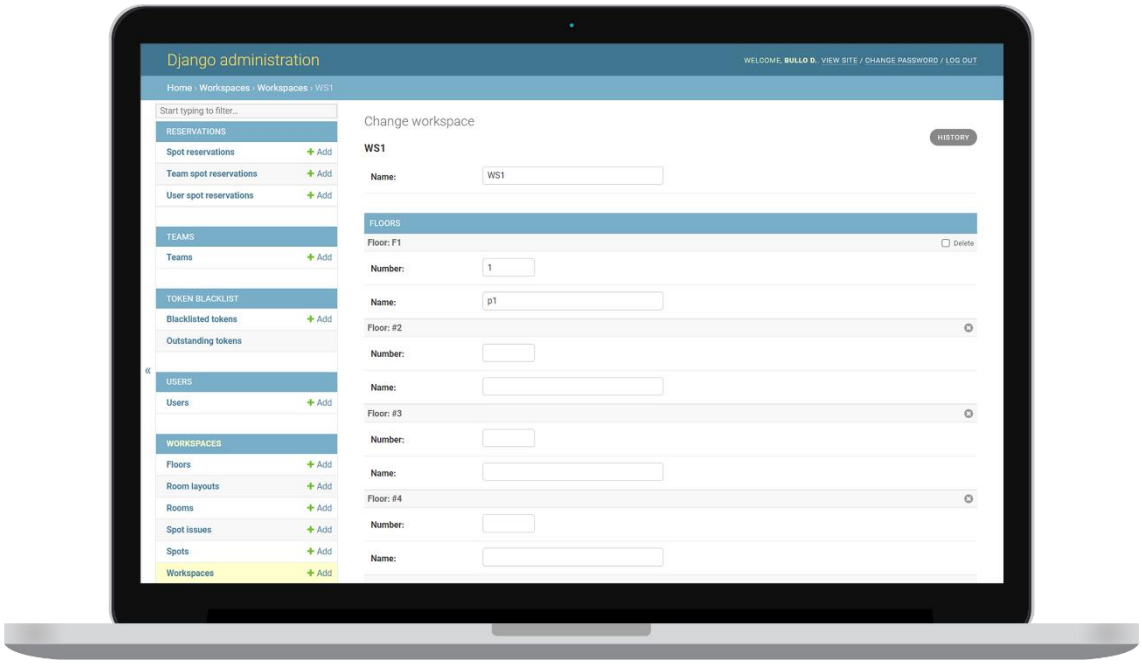
Migrácie si ukladajú vlastné súbory a výhodou je ich konzistencia. Čo znamená, že rovnaké migrácie aké vytvárame pri vývoji aplikácie sa pretavia aj do produkčného nasadenia.

4.2.5 **Administrátorské rozhranie**

Ďalšou pre nás vhodnou funkcionalitou je, že obsahuje zabudované administrátorské rozhranie, takzvané Django Admin. Tento nástroj používa aplikácie a modely na automatické vytvorenie rozhrania, ktoré môžeme použiť na vytváranie, zobrazovanie, aktualizáciu a odstraňovanie záznamov.

Tento zabudovaný pomocný program nám ušetrí veľa času počas vývoja aplikácie, či už je to pri samotnom vývoji ako nástroj pre získanie lepšej predstavy o naších modeloch, alebo po nasadení do prevádzky ako užitočná pomôcka na správu dát v produkcii.

Takéto rozhranie je teda automaticky generované no v našej aplikácií si ho prispôsobíme našim potrebám. To znamená, vytvoríme si vlastné filtrovanie a zoraďovanie dát, skryjeme či zobrazíme dáta na potrebných miestach a celkovo zlepšíme používateľský zážitok.



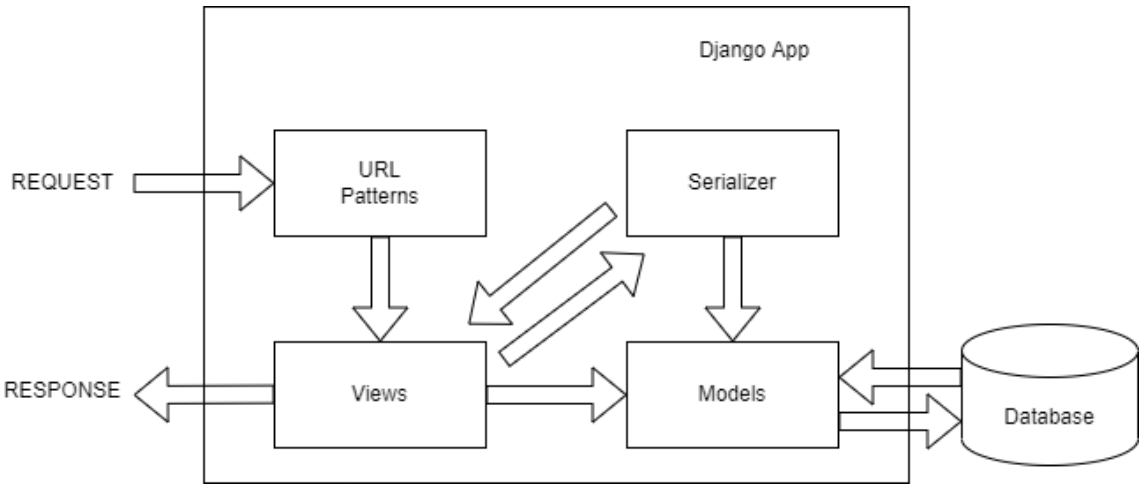
Obrázok 4.2 Pohľad na administrátorské rozhranie

4.2.6 **Django REST Framework**

Django ako nástroj vie zabezpečiť kompletné celú aplikáciu pri modely MVT. To znamená stránky vytvárané na stránke servera. My sme si ale našu aplikáciu navlhli tak, že bude typu SPA.

Veľmi často používaným nástrojom pri takomto návrhu aplikácie je Django REST *framework*. Ide o knižnicu, ktorá funguje so štandardnými Django modelmi a dokáže vytvoriť flexibilné API pre náš projekt.

Na Obrázok 4.3 môžeme vidieť, ako sa správa naša aplikácia s použitím tohto nástroj a následnom pri prijatí žiadosti.



Obrázok 4.3 Pohľad na spracovanie žiadosti serverom

4.2.7 **JWT**

JSON Web Token (JWT) je otvorený štandard, ktorý sa používa na zdieľanie bezpečnostných informácií medzi dvoma stranami – klientom a serverom. Každý JWT obsahuje zakódované objekty JSON. JWT sú podpísané pomocou kryptografického algoritmu, aby sa zabezpečila konzistencia údajov.

V našej aplikácií tento token používame pre autentifikáciu používateľa. V našom prípade server vytvorí JWT token na základe žiadosti od klienta. Klient potom odosiela tento token spolu so žiadosťami na server. Ten následne overí platnosť JWT tokenu. Táto platnosť sa overuje nie len správnosť údajov, ale aj vypršanie daného tokenu. S takýmto dizajnom prichádzajú dva typy tokenu.

- ***Access token***, v preklade môžeme hovoriť o prístupovom tokene. Tento token obsahuje všetky informácie, ktoré server či klient potrebuje. Zvyčajne ide o tokeny s krátkou a ukončenou platnosťou. V našej aplikácií je táto hodnota nastavená na 5 minút.
- ***Refresh token*** alebo v preklade obnovovací token sa používa na generovanie nového prístupového tokenu. Typicky, ak má prístupový token dátum vypršania platnosti, po jeho uplynutí sa používateľ bude musieť znova overiť, aby získal prístupový token. S obnovovacím tokenom možno tento krok preskočiť a s požiadavkou na API získať nový prístupový token, ktorý používateľovi umožní pokračovať v prístupe k prostriedkom aplikácie [5].

Pre lepšiu predstavu to môžeme priblížiť napríklad preukazu, ktorý Vás oprávňuje na vstup do budovy. Obsahuje špeciálne povolenia, do rôznych oblastí budovy. Pri každých

dverách, ktoré chcete otvoriť, tento preukaz musíte ukázať autorizačnej autorite. Tá overí, že Váš preukaz je platný. Ak sa pokúsite vstúpiť do miestnosti do ktorej nemáte prístupové práva na kartičke, dvere sa vám neotvoria. Čo sa týka generovania nových kartičiek, to si môžeme predstaviť ako to, že preukaz má dátum platnosti, po ktorom Vás dvere nepustia, aj keď máte právo na ich prístup ale preukaz Vám vypršal.

Takýto návrh podlieha najnovším a najbezpečnejším trendom v oblasti vývoja webových aplikácií. Výhodou takéhoto riešenia je, že jeho nasadenie je veľmi populárne a používa ho množstvo aplikácií. S tým prichádza lepšia integrácia a jednoduchší proces škálovania, či zmien v systéme. Nehovoriac o tom, že prichádza s výkonom.

4.2.8 Modely

Webové aplikácie Django pristupujú a spravujú údaje prostredníctvom objektov programovacieho jazyka Python, označovaných ako modely. Modely definujú štruktúru uložených údajov vrátane typov polí a prípadne aj ich maximálnej veľkosti, predvolených hodnôt, možností výberového zoznamu, pomocného textu pre dokumentáciu, textu štítkov pre formuláre atď. Definícia modelu je nezávislá od použitej databázy. Keď si vyberieme databázu, ktorú chceme použiť, nemusíme s ňou vôbec priamo interagovať – stačí napísať štruktúru vášho modelu a o ďalšiu prácu a komunikáciou s databázou sa postará Django [6].

```python
class Workspace(models.Model):
    created_at = models.DateTimeField(auto_now_add=True)
    name = models.CharField(max_length=255)

    def __str__(self):
        return f"{self.name}"


class Floor(models.Model):
    workspace = models.ForeignKey(Workspace, on_delete=models.CASCADE)
    number = models.IntegerField()
    name = models.CharField(max_length=255)

    def __str__(self):
        return f"F{self.number}"

```

Obrázok 4.4 Ukážka modelov

Na Obrázok 4.4 môžeme vidieť štruktúru takýchto modelov priamo v kóde. Vieme tu definovať rôzne atribúty objektu, ktoré následne pomocou migrácií vytvoria príslušné tabuľky či stĺpce v databáze. Tieto modely odzrkadľujú náš návrh databázy, až na pár výnimiek, ktoré sa vyskytli. Či už sú to špecifiká pre nástroj ktorý používame, alebo následné vylepšenie nášho návrhu priamo pri implementácií.

Jedným z týchto príkladov a rovnako tak aj zaujímavostí, ktoré sme v pri implementácií našich modelov použili je knižnica *djang-polymorphic*.

4.2.9 Polymorfiznus

V našom návrhu sme počítali s tým, že rezerváciu môže vykonávať aj tím, aj používateľ. Pri implementácií sa ale naskytol spôsob, ako takýto dizajnový model implementovať tak, aby sme zbytočne nevytvárali duplicitu kódu. Dôležité je pochopiť, že pri bežnom návrhu modelu, potrebujeme mať preň samostatné *view*, a *serializer*. Rovnako tak aj nejakú jednoznačnú URL adresu.

Tento problém vyriešime tak, že budeme vyžívať ten istý základný model, ako môžeme vidieť na Obrázok 4.5. Tento model rezervacií miesta využíva polymorfizmus a ide teda o polymorfistický model.

```python
class SpotReservation(PolymorphicModel):
    created_at = models.DateTimeField(auto_now_add=True)
    created_by = models.ForeignKey(User, related_name='created_spot_reservations', on_delete=models.DO_NOTHING)
    reservation = models.OneToOneField(Reservation, related_name='reservations_for_spot', on_delete=models.CASCADE)
    spots = models.ManyToManyField(Spot)
    permanent = models.BooleanField(default=False)
    permanent_status = models.CharField(
        blank=True, null=True,
        max_length=20,
        choices=SpotPermanentStatusChoices.choices,
        default=SpotPermanentStatusChoices.SUBMITTED
    )
```

Obrázok 4.5 Detail modelu pre rezerváciu miesta

Polymorfizmus vo všeobecnosti v programovaní označuje vlastnosť objektovo orientovaného programovania, ktorá umožňuje objektom odvodeným z rôznych tried volať tú istú metódu s rôznou implementáciou. Takéto správanie môžeme vidieť aj na Obrázok 4.6.

```python
class UserSpotReservation(SpotReservation):
    reservation_for = models.ForeignKey(User, related_name='user_reservations_for_spot', on_delete=models.CASCADE)

class TeamSpotReservation(SpotReservation):
    reservation_for = models.ForeignKey(Team, related_name='team_reservations_for_spot', on_delete=models.CASCADE)

```

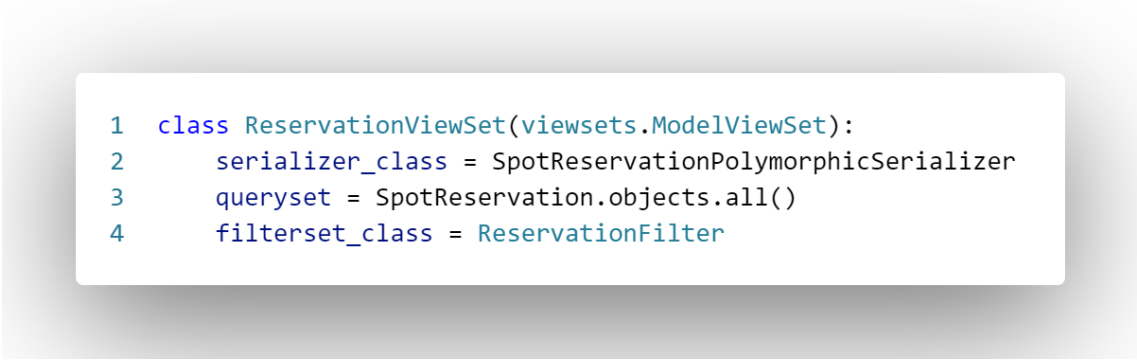
Obrázok 4.6 Zobrazenie využitia polymorfizmu v modeloch

Trieda „***UserSpotReservation***“, rovnako ako aj trieda „***TeamSpotReservation***“ vychádzajú zo všeobecnej triedy „***SportReservation***“ pre rezerváciu miest. Pričom táto trieda obsahuje všetky potrebné informácie spoločné pre prípad, keď je rezervácia pre používateľa, alebo pre tím. Následne triedy, ktoré sú od odvodené (dedia), definujú doplňujúce vlastnosti. V našom prípade je to cudzí kľúč do prislúchajúcich tabuliek, ktorý uchováva informácie o tom, pre koho je rezervácia určená.

Takýto dizajn umožňuje pridať jednotlivým odvodením triedam ďalšie špecifické vlastnosti. Pri rezervácií pre tím by to mohlo byť napríklad doplňujúce informácie ako účel rezervácie, ktoré by tím potreboval zadávať, ale používateľ nie

4.2.10 **Views**

Po slovensky by sa views, dali nazvať aj ako pohľady, ale tieto výrazy sa nezvyknú prekladať. V našej implementácií slúžia ako vstupný bod pre údaje. Nachádzajú sa v jednotlivých aplikáciách v súbore *views.py.*



```python
class ReservationViewSet(viewsets.ModelViewSet):
    serializer_class = SpotReservationPolymorphicSerializer
    queryset = SpotReservation.objects.all()
    filterset_class = ReservationFilter
```

Obrázok 4.7 Pohľad na ReservationViewSet

Naša aplikácia prevažne využíva zabudovanú triedu „*ModelViewSet*“ z ktorej si odvodíme triedu pre náš model. Môžeme to vidieť na Obrázok 4.7 . Pre fungovanie táto trieda potrebuje minimálne dve základné veci. Špecifikáciu, aký serializátor sa má použiť, to vidíme v atribúte *serializer_class*. Zatiaľ čo *queryset* nám určuje pre aké modely je tento pohľad určený.

Takáto trieda obsahuje preddefinované správanie, ktoré nám jednoducho umožní zobraziť list objektov, detail na objekt či možnosť vymazania objektu. Tieto metódy sú zdedené a fungujú bezproblémovo, pokiaľ máme len základné modely a jednoduchú logiku.

Pri niektorých situáciách, sme v našej aplikácií boli nútení takúto triedu rozšíriť o vlastné akcie. Na Obrázok 4.8 môžeme takéto rozšírenie vidieť implementované v kóde. Jedná sa o novú takzvanú akciu, pre konkrétne zobrazenie, ktorá má automaticky vytvorenú aj URL adresu podľa príslušného view, v ktorom je definované a názvu metódy.

Na tomto príklade chceme, aby sa používateľovi zobrazili len jeho rezervácie, alebo rezerváciu tímu, v ktorom je zaradený. Takéto rezervácie získame tak, že vidíme, že prechádzame viacero tabuliek a filtrujeme údaje podľa príslušnej logiky.

V premennej *user_reservations* si získame všetky rezervácie používateľov, ktoré obsahujú používateľa, ktorý žiada o takúto informáciu resp. ktorý poslal žiadosť na server.

V druhej premennej s názvom *user_team_reservations* dopytujeme na databázu všetky rezervácie tímov, ktorých súčasťou je aj používateľ, ktorý poslal žiadosť.

Následne tieto dva dopyty spojíme a vrátime relevantné dáta, spolu so statusom 200.

```python
@action(methods=['GET'], detail=False)
def my_reservations(self, request):
    user_reservations = SpotReservation.objects.filter(Q(UserSpotReservation___reservation_for=request.user))
    user_team_reservations = SpotReservation.objects.filter(
        Q(TeamSpotReservation___reservation_for__in=Team.objects.filter(user=request.user)))
    queryset = user_reservations | user_team_reservations
    serializer = self.serializer_class(queryset, many=True)
    return Response(status=status.HTTP_200_OK, data=serializer.data)

```

Obrázok 4.8 Rozšírenie view o akciu zobrazenia rezervácií používateľa

Inou akciou, ktorú sme do našej aplikácie doplnili je zmena statusu rezervácie. Vyplýva to najmä z dizajnu, ktorý sme zvolili. V tomto prípade používame *ModelViewSet*, ku ktorému chceme aby mal prístup každý. To ale nie je žiadané, v tomto príklade, pretože očakávame, že status môže zmeniť len administrátor. Preto máme na výber dve viaceré možnosti. Môžeme napríklad spraviť nové view. Rovnako tak ale môžeme naše existujúce view opäť rozšíriť o akciu, ktorej následne povieme, že je možná len pre určené používateľské práva. Jednoduchším, prehľadnejším a aj odporúčaním

riešením je druhý spomenutý spôsob a teda, pomocou akcie. Ako vyzerá implementácia takéhoto riešenia, môžeme vidieť na Obrázok 4.9.

Takáto situácia nás plynule privádza k tomu, aby sme si opísali, ako funguje zabezpečenie jednotlivých pohľadov, hlavne z oblasti používateľských práv. Rovnako sa pozrieme aj na možnosť filtrovania, zoraďovania či vyhľadávania.

```python
@action(methods=['POST'], detail=True)
def change_status(self, request, pk=None):
    reservation = self.get_object()
    new_status = request.data.get('status', None)
    if new_status not in SpotPermanentStatusChoices:
        return Response(status=status.HTTP_400_BAD_REQUEST)
    reservation.permanent_status = new_status
    reservation.save()
    return Response(data=self.serializer_class(reservation).data, status=status.HTTP_200_OK)

```

Obrázok 4.9 Zmena statusu rezervácie

4.2.11 **Oprávnenia**

Zabezpečenie jednotlivých pohľadov prebieha pomocou oprávnení (permissions), ktoré nám definujú, kto daný pohľad môže upravovať, resp. aké metódy môže nad pohľadom zavolať. Sú uložené v samostatnom súbore s názvom permissions.py, pripadne sa môžu nachádzať v samotnej akcií.

Celá naša aplikácia má globálne nastavenie také, že ku koncovému bodu sa dostane len používateľ, ktorý je prihlásený. To zabezpečí jednotlivé prístupové body pred ľuďmi mimo systém. Treba pritom ale myslieť na to, že v takomto nastavení je potrebné koncové body pre prihlásenie či registráciu explicitne povoliť aj pre neautentifikované žiadosti.

Ďalšie zobrazenia, ktoré sú určené len pre administrátorov by mohli vyzerať tak, ako Obrázok 4.10.

```python
class IsAdminOrReadOnly(permissions.IsAuthenticatedOrReadOnly):
    def has_permission(self, request, view):
        return request.user.is_admin()
```

Obrázok 4.10 Vlastné pravidlo pre oprávnenia

Na tomto príklade je vytvorená trieda, v ktorej vyhodnocujeme žiadosti na základe žiadanej funkcionality. Funguje to nasledovne. Keďže v triede dedíme triedu *IsAuthentificatedOrReadOnly*, tá nám zabezpečí, že používateľ bude autentifikovaný, alebo sa pozerá, či žiadosť prišla pomocou takzvaných SAFE_METHODS. Tieto sú určené ako žiadosti typu GET, HEAD, OPTIONS, pretože pri takýchto žiadostiach nie je možná zmena dát.

Našou úlohou je teda ešte zabezpečiť to, že používateľ bude mať potrebné príslušné práva. Pri dobrom návrhu aplikácie je to jednoduché, pretože naše modely obsahujú metódu *is_admin()*, ktorá nám vracia, či je používateľ administrátor alebo nie. Túto metódu teda využijeme aj pri oprávneniach.

Ak používateľ má potrebné oprávnenia pohľad zobrazíme, v opačnom prípade vrátime HTTP odpoveď s kódom 403. Takýto kód sa zvyčajne používa na označenie toho, že užívateľ, ktorý poslal žiadosť na server, nemá dostatočné používateľské práva na tento pohľad.

Takýmto štýlom si vieme vytvoriť aj veľmi komplikované oprávnenia, ako napríklad aby sa k dátam o tímoch vedel dostať len užívateľ ktorý je v takomto tíme a pod.

4.2.12 **Filtrovanie/ Zoraďovanie/ Vyhľadávanie**

Naše zobrazenie už máme teda vytvorené, priradený model, nastavené správne oprávnenia. Čo ale pre našu aplikáciu ešte potrebujeme a budeme to častokrát využívať, je filtrovanie, zoraďovanie a vyhľadávanie na takomto zobrazení. Takéto správanie je žiadúce v podstate počas celého používania aplikácie. Od filtrovania tímov, či používateľov v tabuľkách až pod filtrovanie jednotlivých pracovných priestorov a rezervácií, ktoré sú v týchto priestoroch spravené.

Filtrovanie je pre našu aplikáciu zásadné. Ešte podstatnejší je ale fakt, kde sa takéto filtrovanie vykonáva. Napríklad potrebujeme vyfiltrovať údaje, aby sme používateľovi mohli rovno zobraziť len rezervácie pre konkrétnu miestnosť, ktorú si vybral na konkrétny dámu. Predstavme si situáciu, že máme 100 miestností, ktoré majú plno rezervácií na mesiac dopredu. Keby sme poslali takéto množstvo dát na klienta a následne to potrebovali nejako rozumne zobraziť, dáta by sme museli vyfiltrovať od nepodstatných dát a zbytočne by sme zvyšovali množstvo prenášaných dát.

Toto môžeme vyriešiť filtrovaním priamo na serveri. To sa prejaví tak, že klientska aplikácia nám povie, akú miestnosť potrebuje a v akom dátume a serverová aplikácia jej

pošle len tie dáta, ktoré sú relevantné. Taktiež je za predpokladu architektúry klient-server pravdepodobné, že server je niekoľkonásobné výkonnejší, takže takéto úlohy spracuje oveľa rýchlejšie.

```python
class ReservationFilter(django_filters.FilterSet):
    room = django_filters.ModelChoiceFilter(
        label='Room',
        empty_label='All rooms',
        queryset=Room.objects.all(),
        to_field_name="id",
        method='filter_room',
    )
    reservation_start = django_filters.DateTimeFilter(field_name="reservation__start__date",
                                                      lookup_expr=('lte'), )
    reservation_end = django_filters.DateTimeFilter(field_name="reservation__end__date", lookup_expr=('gte'), )

    def filter_room(self, queryset, name, value):
        return queryset.filter(spots__room_id=value)

    class Meta:
        model = SpotReservation
        fields = "__all__"
```

Obrázok 4.11 Vlastný filter pre rezervácie

Niektoré jednoduché filtre vieme využiť aj priamo z Djanga resp. Django REST, no niektoré musíme do aplikácie doprogramovať. Ako môžeme vidieť na Obrázok 4.11, tu sme si vytvorili vlastnú triedu, kde povieme podľa čoho sa majú jednotlivé vlastnosti filtrovať. Na príklade môžeme vidieť, že sa pozeráme na miestnosti, ktoré sú obsiahnuté v žiadosti rovnako tak na dátumy. V situácií, že by tieto polia v žiadosti neboli uvedené, aplikácia zobrazí všetky výsledky.

Veľmi obdobne pristupujeme k zoraďovaniu a k vyhľadávaniu v týchto views. Znovu sa jedná o to, že klient odošle parameter, podľa ktorého chce vyhľadať dotaz a server mu vráti odpoveď. Výhodou je, že vyhľadávanie, filtrovanie a zoraďovanie sa dajú navzájom kombinovať v jednej žiadosti.

4.2.13 **Serializátory**

V predošlých častiach sme si opísali ako fungujú modely a zobrazenia. Finálnou časťou pre správne zabezpečenie fungovania API sú takzvané serializátory, čo v Djangu označuje výraz *serializers*. Jedná sa o akúsi šablónu pre API. Ako tak pre prijímanie, tak aj pre odosielanie žiadostí.

Serializátory v Django REST Framework sú zodpovedné za konverziu objektov na dátové typy zrozumiteľné pre JavaScript a front-end *framework-y*. Serializátory tiež poskytujú deserializáciu, čo umožňuje konvertovať analyzované údaje späť na komplexné typy po prvom overení prichádzajúcich údajov [7].

V našej aplikácií využívame komplexnejšej zobrazenie objektov. Hlavne čo sa týka polymorfistických serializátorov, tieto si musíme upraviť podľa vlastnej potreby. Na príklade Obrázok 4.12 je zobrazený konkrétny príklad takejto implementácie.

```python
class SpotReservationByTeamSerializer(SpotReservationSerializer):

    def create(self, validated_data):
        reservation_data = validated_data.pop('reservation')
        spots = validated_data.pop('spots')
        reservation = Reservation.objects.create(**reservation_data)
        instance = TeamSpotReservation.objects.create(**validated_data, reservation=reservation)
        instance.spots.set(spots)
        return instance

    def to_representation(self, instance):
        self.fields["reservation_for"] = TeamSerializer(read_only=True)
        to_representation = super().to_representation(instance)
        return to_representation

```

Obrázok 4.12 Serializátor pre tímovú rezerváciu

Na tomto obrázku môžeme sledovať hneď dve metódy. Prvá metóda ***create*** je využívaná, ako už názov napovedá, pri vytváraní nového záznamu. Taktiež je tu ukázané, že si zo žiadosti vyberieme potrebné údaje, vytvoríme objekt v databáze pre rezerváciu tímu a nastavíme miesta, ktoré sme taktiež dostali v žiadosti.

Druhou metódou je metóda ***to_representation***. Táto metóda je využívaná pri odosielaní dát zo servera. To znamená, namiesto toho, aby sme vrátili napríklad len id toho objektu, pre ktorý bola rezervácia spravená, využime serializátor daného objektu. Takýmto spôsobom dostaneme všetky údaje o danom objekte, pri odpovedi servera. Tie môžeme následne využiť v klientskej aplikácií.

Ako môžeme vidieť, serializátory môžu byť rôzne vnorené a využívané v náš prospech. Treba ale dbať na ich správne využitie, aby sme opäť neprenášali zbytočne veľa dát.

```python
class SpotReservationPolymorphicSerializer(PolymorphicSerializer):
    model_serializer_mapping = {
        UserSpotReservation: SpotReservationByUserSerializer,
        TeamSpotReservation: SpotReservationByTeamSerializer,
    }
```

Obrázok 4.13 Ukážka zloženia kontrolóra s využitím polymorfizmu

Pre zabezpečenie správneho fungovania polymorficstických modelov, si vytvoríme serializátor, ktorý nám bude obaľovať všetko polymorfistické modely do jedného základného sterilizátoru viď. Obrázok 4.13. Tým docielime to, že pri dopytovaní rezervácií sa nám na koncovom bode zobrazia ako rezervácie pre používateľov, tak aj zobrazenie pre tímy.

4.2.14 **Koncové body**

Keď sme si prešli celý návrh MCV a vieme ako sa žiadosti správajú pri ich prijímaní či odoslaní posledným krokom sú koncové body, alebo takzvané *endpoint-y*. Jedná sa o URL adresy definované pre jednotlivé zobrazenia.

Ako môžeme vidieť Obrázok 4.14 nám ukazuje základné zobrazenie koncových bodov. To znamená pri zadaní „*./admin/*“ do URL adresy za doménu nás aplikácia nasmeruje na Django admin URL adresy. V prípade zadania „*./api/“* nás aplikácia smeruje ďalej na nami definované URL adresy.

```
1  urlpatterns = [
2      path('admin/', admin.site.urls, name='admin'),
3      path('api/', include('config.api')),
4  ]
```

Obrázok 4.14 Základné rozdelenie koncových bodov

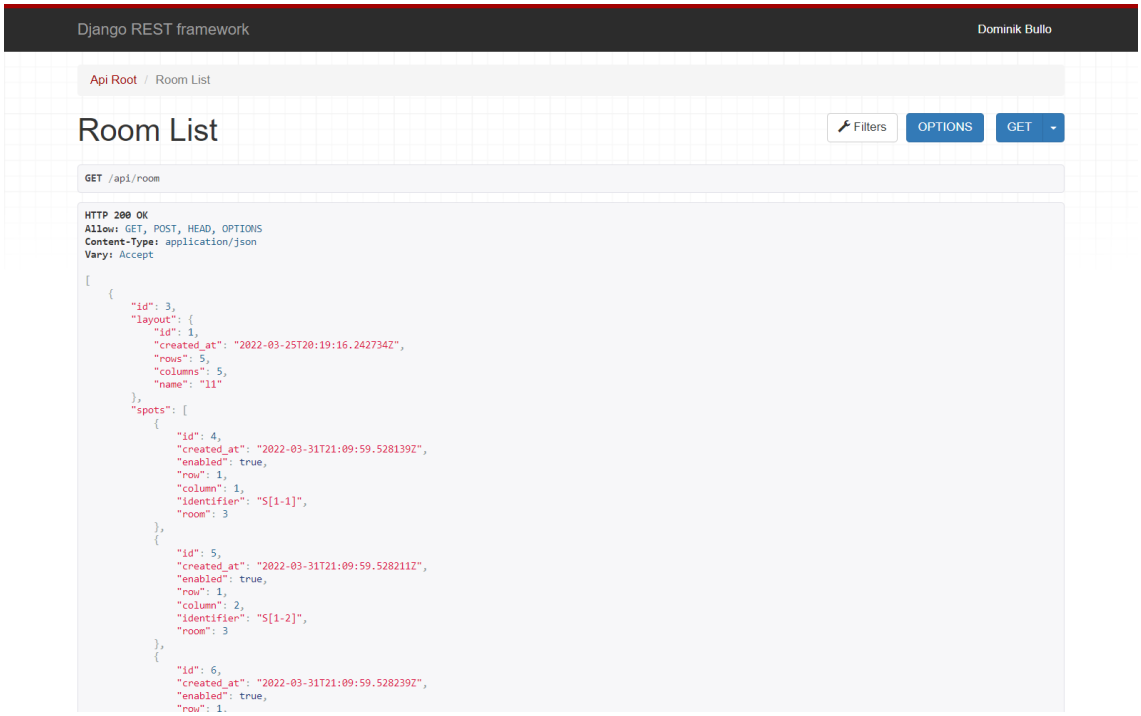
Každá z týchto adries smeruje do samostatného súboru urls.py v konkrétnej aplikácií, podľa jednotlivých celkov. Následne máme možnosť v týchto súboroch definovať URL adresy špecifické pre daný modul.

```python
router.register(r"spots?", SpotViewSet, basename="spots")
router.register(r"spots?-issues?", SpotIssueViewSet, basename="spot-issues")

router.register(r"rooms?-layouts?", RoomLayoutViewSet, basename="room-layouts")

router.register(r"workspaces?", WorkspaceViewSet, basename="workspaces")
router.register(r"floors?", FloorViewSet, basename="floors")
router.register(r"rooms?", RoomViewSet, basename="rooms")
```

Obrázok 4.15 URL adresy pre pracoviská

Ako môžeme vidieť na Obrázok 4.15 takéto koncové body majú ešte rôzne doplňujúce možnosti. Napríklad môžeme zadať „*spot-issues, spots-issues, spot-issue, spots-issues*“ do prehliadača a všetky tieto URL adresy budú presmerované na rovnaké a správne view. Toto uľahčí prácu hlavne programátorom, pretože pri využívaní listu je logické, že voláme napríklad „*rooms*“, ale ak už chceme konkrétnu miestnosť dopytujeme adresu pomocou id tejto miestnosti a to v tvare „*/room/1/*“.

Na záver tejto časti by sme si mohli ukázať, ako takýto koncový bod vo výsledku vyzerá. Na Obrázok 4.16 vidíme aké dáta a v akej podobe sa posielajú na klientsku aplikáciu napríklad pri nevyfiltrovanom zozname miestností. Následne sa pozrieme aj na jej vývoj a ako s týmito dátami pracuje.



Obrázok 4.16 Príklad koncového bodu pre miestnosti

4.3 Vývoj klientskej aplikácie

Pri vývoji klientskej aplikácie by sme sa mali zamerať hlavne na to, aby bola používateľský prívetivá, rýchla a responzívna. Takisto nemôžeme zanedbať štandardy, ktoré sa vo vývoji webových aplikácií odporúčajú či vyžadujú.

Pri vývoji aplikácie sa budeme snažiť, aby aplikácia mala najmä na starosti zobrazovanie dát a všetko ostatné necháme na serverovú aplikáciu. Poďme sa teda pozrieť na proces vývoja klientskej aplikácie.

4.3.1 Vue.js

Vue.js je progresívny framework pre JavaScript používaný na vytváranie webových rozhraní a jednostránkových aplikácií. Je inšpirovaný dvoma skvelými webovými *famework-ami*, t.j. React a Angular. Je určený nielen pre webové rozhrania, ale používa aj na vývoj desktopových a mobilných aplikácií s *framework-om* Electron. Primárne však je využívaný v oblasti vývoja webových aplikácií. Rozšírenie HTML a základňa JS rýchlo urobili z Vue obľúbený *frontend-ový* nástroj, čoho dôkazom je prijatie takými gigantmi ako Adobe, Behance, Alibaba, Gitlab a Xiaomi [8].

4.3.2 Vuexy template

V dnešnej dobe vývoja aplikácií sa na urýchlenie práce často používajú šablóny. Takéto šablóny umožňujú vývojárom sústrediť sa hlavne na funkcionalitu a jej programovanie a dizajn prenechať na špecializovaného dizajnéra.

Vzhľadom na to, že s touto konkrétnou šablónou má autor pozitívnu skúsenosť už z bakalárskej práce, použijeme túto šablónu opakovane. Aktívnej na nej pracuje celý tím dizajnérov a programátorov, čo je predpoklad na kvalitu. Navyše šablóna sa neustále zlepšuje a aktualizuje aby odrážala trendy vo vývoji webových aplikácií.

Základ našej klientskej aplikácie preto tvorí šablóna Vuexy, ktorá obsahuje preddefinované usporiadania a komponenty, ktoré si následnej vieme doupravovať, podľa našich požiadaviek. Takýto prístup nám zabezpečí jednotný dizajn užívateľského zážitku (UX) a používateľské rozhranie (UI) naprieč celou našou aplikáciou. Netreba ale zabúdať, že takáto šablóna prináša najmä vzhľad komponentov. Logiku týchto komponentov je stále potrebné naprogramovať.

Šablóna disponuje takzvaným štartovací balíčkom, ktorý sme si rozbalili a vložili do nášho projektu. Ako už bolo spomenuté, autor má s touto šablónou skúsenosti, preto zrejme aj jej nainštalovanie a samotné implementovanie bolo bez väčších problémov.

Šablóna síce prešla veľkým vývojom. pretože medzičasom došlo k aktualizácií *framework-u* Vue.js na verziu 3. Pritom došlo k dosť výrazným zmenám prístupu samotného Vue šablóna by nemohla fungovať na novšej verzií. Išlo o veľké zmeny, ktoré definovali nové smerovanie Vue.js Aj takáto znena viedla vývojárov k tomu, že napríklad aj nástroj pre kaskádové štýly (CSS), Tailwind bol nahradený Bootstrap-om. Rovnako tak aj množstvo menších knižníc bolo vymenených. Tieto rozhodnutia autori, podľa ich vyjadrení, spravili pre zabezpečenie správneho fungovania šablóny na novej verzií *framework-u* Vue. Rovnako tak touto aktualizáciiou dostali príležitosť k lepšej implementácií knižníc, ktoré sú viac populárne, alebo robustnejšie prípadne spoľahlivejšej spolupráci s vývojármi spomínaných nástrojov.

Šablóna prichádza s veľmi dobrou dokumentáciou, ktorá popisuje, ako používať jednotlivé komponenty. Ich implementácia je ale na vývojárovi. Užitočný je aj fakt, že obsahuje množstvo ukážok, ako tieto komponenty môžu byť využívané a aké vlastnosti môžu byť použité pri ich nastavovaní.

4.3.3 Tmavý mód

Šablóna nám ponúka možnosť nadizajnovania a implementovania rôznych farebných tém. V našej aplikácií sme zvolili ako predvolený tmavý mód aplikácie. Takýto mód je v dnešnej dobe vo veľkej obľube a okrem toho nesie množstvo výhod. Môže to byť len dizajnová stránka a teda príjemnejšie zobrazenie v prostredí s nízkym svetlom, ale aj napríklad fakt, že takýto mód by mal šetriť batériu zariadenia a znižovať záťaž na oči.

Samozrejme tmavý mód nesie aj nevýhody, preto naša aplikácia obsahuje aj svetlý mód, ktorý je možné jednoducho prepnúť. Výber módu preto necháme na užívateľa aplikácie. Rovnako tak, pre lepšiu viditeľnosť v našej práci, používame ukážky práve zo spomínaného svetlého módu.

4.3.4 Štruktúra klientskej časti

Podobne ako serverová aplikácia, aj klientska aplikácia je rozdelená do jednotlivých zložiek. Kvôli lepšiemu prehľadu. Kombinujeme tu tak už vytvorenú, alebo odporúčanú štruktúru šablóny spolu s našimi potrebami. Opisovať jednotlivé priečinky v tejto časti práce nemá význam, štruktúru môžeme vidieť na Obrázok 4.1.

Čo ale určite spomenúť treba je, že aj táto časť našej aplikácie nasleduje naše logické celky. To znamená, rovnako tak aj tu sú jednotlivé priečinky rozdelené a pomenované podľa týchto celkom a takisto URL adresy získavajú pred sebou názov logického celku v ktorom sa nachádzajú.

4.3.5 JWT

Autentifikácia v našej klientskej časti funguje nasledovným spôsobom. Pri prihlásení sa do *localStorage* uložia dva tokeny, ktoré aplikácia príjme v úspešnej odpovedi na prihlásenie zo strany servera. Ako sme už opisovali v časti 4.2.7, jeden token nesie názov *accessToken* a druhý *refreshToken*.

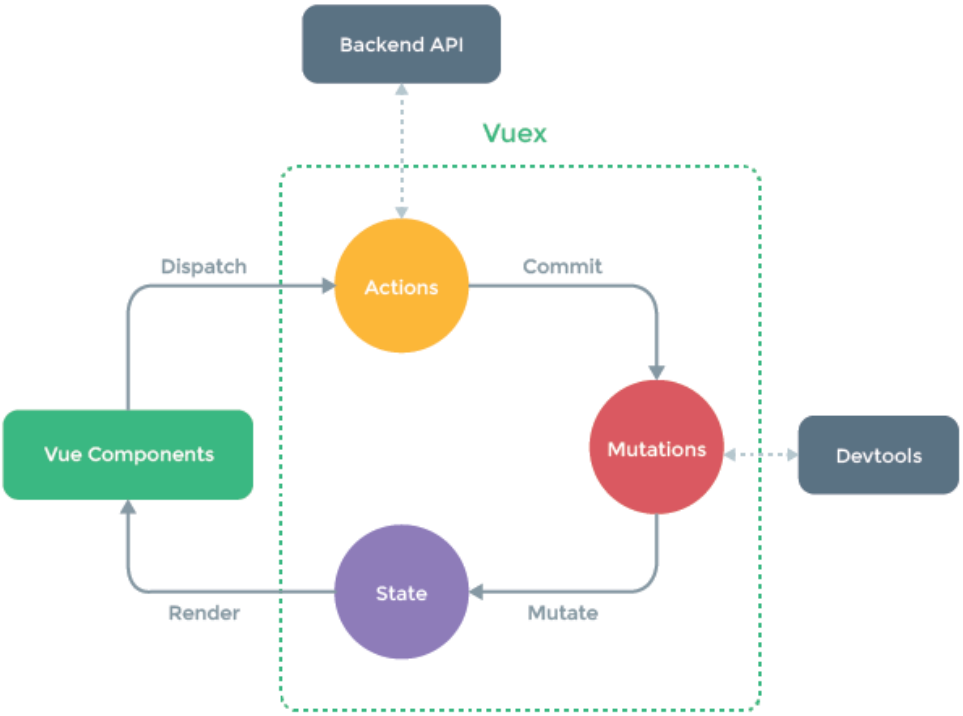
Tieto tokeny sa následne používajú na autentifikáciu každej žiadosti na server. Ak aplikácia dostane odpoveď zo servera, že užívateľ nie je autorizovaný, pokúsi sa tento token obnoviť a pošle tú istú žiadosť s novým tokenom. Toto správanie zabezpečíme v nástroji Axios.

4.3.6 Vuex

Vuex je knižnica, ktorá slúži na správu stavu aplikácie. V aplikáciách s väčším množstvom komponentov môže byť výmena a úprava informácií medzi jednotlivými komponentmi pomerne zložitá. Vuex nám v našej aplikácií pomáha držať stav jednotlivých údajov na jednom mieste, do ktorého budú mať prístup všetky komponenty.

Vo Vuex sa robí zmena hodnôt v existujúcom stave cez mutácie, ktoré sa vykonávajú vždy **synchrónne**. Priamo menia hodnoty v stave.

Okrem mutácií tu existujú aj akcie, ktoré sa na rozdiel od mutácii vykonávajú **asynchrónne**. Akcie obsahujú logiku zmeny stavu a následne vyvolávajú mutácie, čím dochádza k zmene stavu aplikácie [9].



Obrázok 4.17 Diagram fungovania Vuex [9]

4.3.7 **Axios**

Jednou zo základných úloh každej webovej aplikácie je komunikácia so servermi prostredníctvom protokolu HTTP. To sa dá ľahko dosiahnuť pomocou knižnice Axios. Táto je už priamo implementovaná v našej šablóne.

Výhodou, ktorú nám tento nástroj poskytuje je fakt, že pomocou takzvaných „*interceptors*“ prerušiť každú jednu žiadosť ktorú posielame, alebo príjmami. To má pre nás hneď dve výhody.

Prvou je, že pri vývoji aplikácie si môžeme každú jednu žiadosť dať napríklad vypísať do konzole a získame tak lepší prehľad čo a kedy naša aplikácia posiela alebo príma zo strany servera.

Druhou je túto možnosť využiť pri autentifikácii. Pretože využívame token, ktorý má len určitú časovú platnosť napr. 5 minút, vieme sa v žiadosti pozrieť, či odpoveď zo servera ktorá nám prišla neobsahuje status 401. Tento status značí, že zlyhala autorizácia. V takom prípade si môžeme od servera vyžiadať nový token, tak ako to bolo popísané v predošlých častiach a našu pôvodnú žiadosť zopakovať, už s platným tokenom.

4.3.8 **Používateľské práva**

Ďalej nám šablóna ponúka správu používateľských práv, známe pod skratkou *ACL*. Toto taktiež využijeme vo svoj prospech, kedy si v aplikácií zadefinujeme, čo jednotlivé môžu a nemôžu v našej aplikácií robiť, resp. na čo majú používateľské práva a na čo nie.

Následne môžeme takéto správanie v našej aplikácií využiť napríklad tak, že zablokujeme kompletné zobrazenie určitej stránky používateľovi, ktorý nemá príslušné práva.

Môžeme to ale využiť aj na úrovni jednotlivých komponentov, ktoré budú vypisovať iné informácie pre administrátora a iné pre zamestnanca. Rovnako tak môžeme zobraziť alebo skryť tlačidla, na základe toho, akú má používateľ rolu.

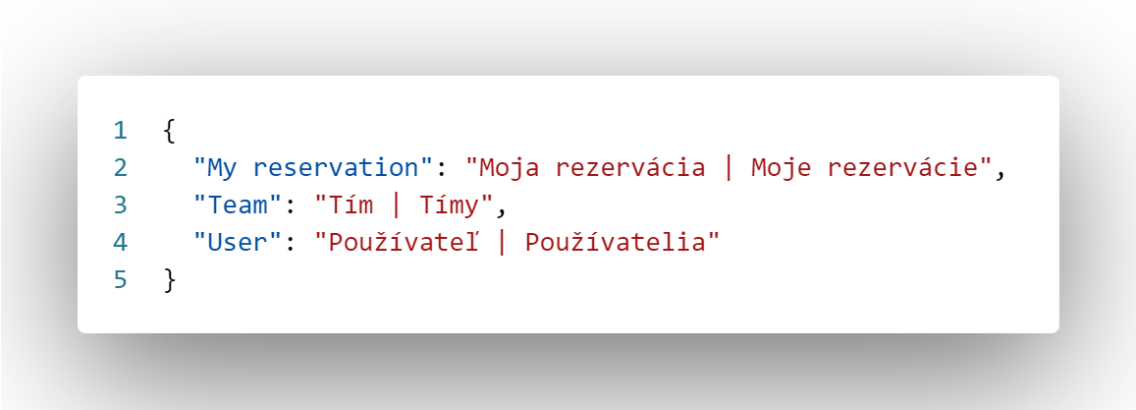
Pri tejto správe ale nemôžeme zabúdať na to, že takéto zabezpečenie funguje skôr len pre lepší pocit z používania aplikácie, ale nijako nechráni naše dáta na strane servera, kde je žiadané zabezpečiť jednotlivé koncové body, aby zohľadňovali používateľské práva-

4.3.9 **Internacionalizácia**

Pre tento termín sa zvyčajne používa skratka *i18n* a ide o proces, ktorý by mal byť plánovaný a jeho implementovaný do aplikácie tak, aby bola pripravená na jednoduchú adaptáciu nového jazyka.

Pretože sme v návrhu zvolili dva jazyky, naša aplikácia bude zabezpečovať aj to, že texty sa budú dať jednoducho preložiť do akéhokoľvek jazyka.

Implementácia v aplikácií je takým spôsobom, že máme JSON súbor so štandardizovanou skratkou jazyka. Tento súbor nám zastáva úlohu slovníka, kde ako kľúče sú anglické slovíčka a ako hodnoty sú uložené preklady v danom jazyku. Keďže pri implementácií využívame špecifickú knižnicu *Vue I18n*, vieme v tomto súbore povedať okrem iného aj to, akým spôsobom sa prekladá jednotné a množné číslo.



Obrázok 4.18 Ukážka prekladového súboru

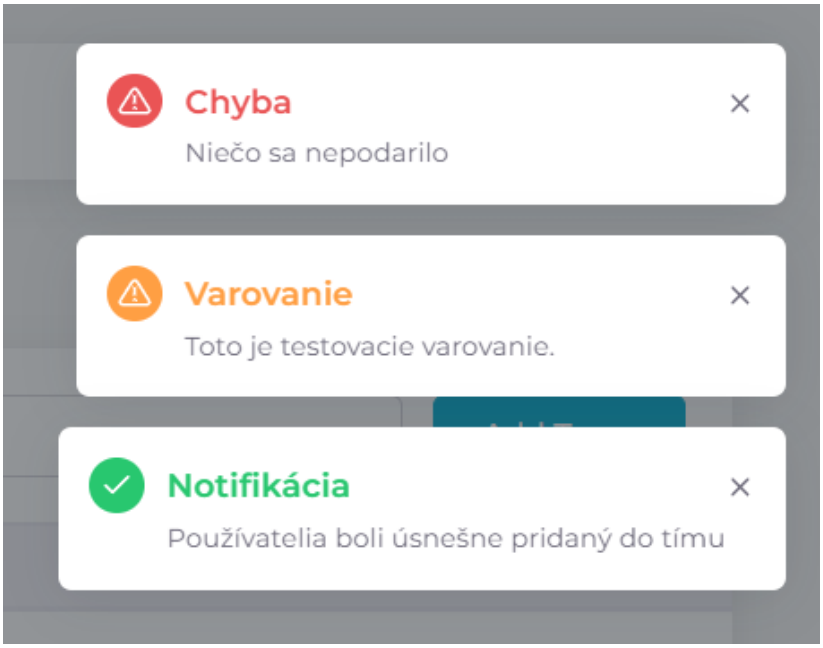
Ako môžeme na príklade Obrázok 4.18 vidieť, na ľavej strane modrou farbou sú vyznačené kľúče podľa ktorých vie aplikácia vyhľadať text na preklad, podľa implementácie v kóde a červenou na pravej strane sú už preklady konkrétneho výrazu. Najskôr ide jednotné číslo a následne znakom | je oddelené množné číslo.

Obrázok 4.19 Označenie textu na frontend-e určeného na preklad

4.3.10 **Notifikácie**

Dôležitou súčasťou každej kvalitnej aplikácie je, aby interagovala s používateľom. Či už sa niečo poradilo, alebo nepodarilo, alebo je potrebné niečo upresniť pre používateľa aplikácie býva častokrát podstatné len vedieť danú informáciu.

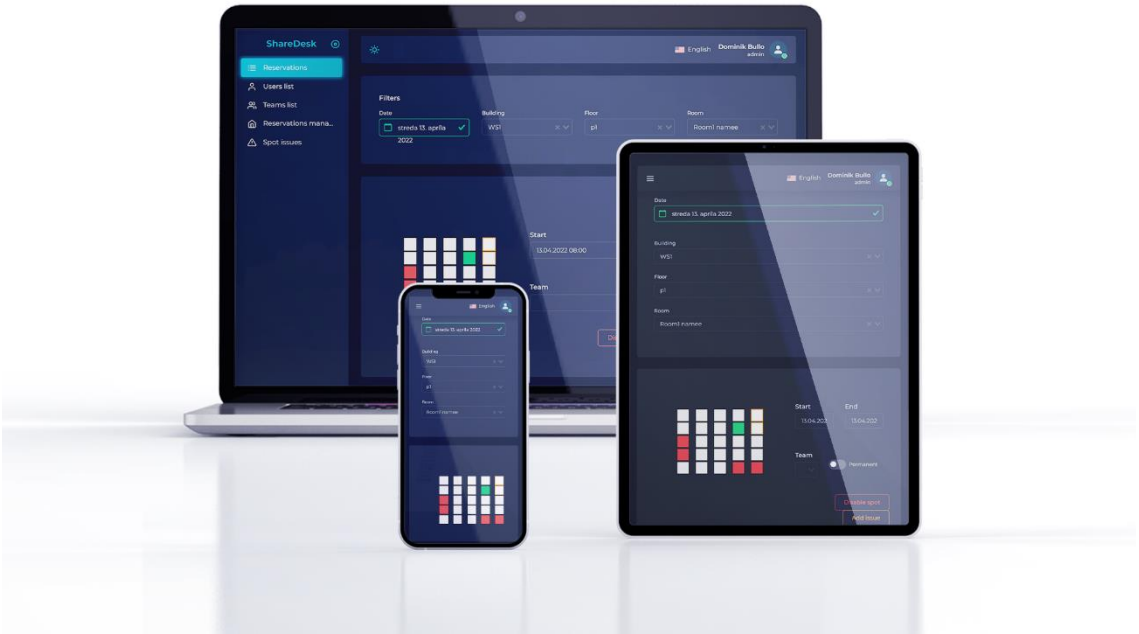
Presne z tohto dôvodu naprieč celou našou aplikáciou využívame takéto notifikácie v podobe takzvaného toastu resp. vyskakovanie notifikácie ako môžeme vidieť na Obrázok 4.20 Takéto notifikácie sú spravidla zelené, oranžové a červené a obsahujú si príslušný text pre používateľ.



Obrázok 4.20 Ukážka notifikácií

4.3.11 **Detail na rezervovania pracovného miesta**

Ako sme už viackrát spomínali, aplikácia je responzívne a vie sa prispôsobiť veľkosti zariadenia. Na Obrázok 4.21 môžeme vidieť, ako aplikácia vyzerá na rôznych typoch zariadenia.



Obrázok 4.21 Ukážka rezervovania miesta v aplikácii

Za povšimnutie určte stojí aj fakt, že jednotlivé miesta sú farebne vyznačené, pre rýchlu a jednoduchú orientáciu používateľa. Každá farba určuje špecifický status a vysvetlivky týchto farieb budú k dispozícií v aplikácii.

4.3.12 Filtrovanie špecifického miesta

Pre logiku našej aplikácie je žiadúce, aby používateľ videl presne len miesta zo špecifickej miestnosti ktorú si vyberie. To znamená, že na výber mu dáme zo všetkých možností, ale postupne budeme náš výber filtrovať o tie, ktoré nie sú žiadúce. V praxi to môže vyzerať nasledovne.

Máme Budovu č. 1 a Budovu č. 2. V Budove č. 1 Máme na prvom poschodí miestnosť R101B1 a na druhom poschodí R201B1. V Budove č. 2 máme len prvé poschodie s miestnosťou R101B2.

Používateľ vie, že chce miestnosť z budovy číslo 1. Pri zvolení tejto budovy mu v jeho vyhľadávaní vyfiltrujeme len poschodia a miestnosti, ktoré sa nachádzajú v danej budove. V jednotlivých políčkach v ktorých si tieto miestnosti vyberá sa dá aj hľadať, to znamená, ak by náhodou bolo miestností tak veľa, že by sa nedalo graficky vybrať, používateľ môže na klávesnici zadať názov a aplikácia mu ukáže priamo výsledky, ktoré spĺňajú jeho vyhľadávanie. Takýmto spôsobom pokračujeme až kým používateľ nevyberie miestnosť.

V takom prípade sa mu zobrazí vizuálne rozloženie danej miestnosti ako vizuálne potvrdenie. Na to ale, aby si takéto miesto vedel zarezervovať ale musí ešte vyplniť jeden potrebný údaj a to dátum, na ktorý miesto hľadá. Pokiaľ tak neurobí bude na to vyzvaný notifikáciou a aj vizuálnym označením políčka s dátumom červenou farbou.

Takáto logika aplikácie by mala zabezpečiť intuitívne používanie rovnako tak aj jednoduché a prehľadné zobrazenie potrebných informácií.

4.4 Problémy spojené s vývojom webovej aplikácie

Pri vývoji softvéru, sa takmer vždy vyskytnú nejaké problémy. Môže ísť o problémy spojené s vývojom softvéru, s jeho nasadením na server, alebo problémy pri jeho používaní. V tejto časti si opíšeme aké problémy sa vyskytli pri vývoji našej aplikácie a ako sme takéto problémy dokázali identifikovať a vyriešiť.

Zo skúsenosti vieme, že jedným z veľmi častých problémoch pri vývoji aplikácie je, že aplikáciu vyvíjate na jednom zariadení. Následne by ste potrebovali aplikáciu vyvíjať na inom zariadení. To môže byť napríklad ďalšie vaše zariadenie, iný vývojár, ale

najpodstatnejšie server, na ktorý chcete vašu aplikáciu nasadiť. Pri takejto situácií sa častokrát vyskytnú chyby, ktorým ste vy nikdy nemuseli čeliť. Pri podobnom type webových aplikácií zväčša ide o chyby knižníc, ich rozličné verzie, či nekompatibility s vaším systémom.

Tento problém je tak častý, že vznikol nástroj, ktorý takéto problémy za cieľ riešiť. Keďže autor vyvíjal bakalársku prácu na podobných technológiách ako aj túto prácu, mal predstavu, aké veci sa môžu vyskytnúť a aký postup zvoliť.

Aplikácia je preto zložená a zaobalená pomocou kontajnerov, konkrétne sa jedná o nástroj Docker.

4.5 Docker

Docker je jeden z popredných platforiem a nástrojom pre kontajnerizácie aplikácií. Poskytuje spôsob, ako zabaliť infraštruktúru aplikácie do kontajnera, kde môže bežať izolovane od zvyšku systému. Rovnako tak aj nástroje na kontrolu kontajnerov, či exportovanie údajov z nich. V krátkosti je to veľmi nápomocný nástroj, či už pri vývoji aplikácie, alebo jej nasadení na server.

4.5.1 Kontajnerizácia

Obľúbeným príkladom kontajnerizácie, koniec koncov na tomto princípe vznikla táto technológia, je prirovnanie ku kontajnerov na prepravu tovaru. Tieto kontajneri sú vytvorené tak, že viete v nich ľahko prepraviť prakticky čokoľvek na ľubovoľnom dopravnom prostriedku. Je jedno či sa jedná o loď, vlak, lietadlo tieto kontajneri sú univerzálne. Rovnako tak pre prepravu takéhoto kontajnera vôbec nie je pre dopravný prostriedok podstatné, čo sa nachádza vnútri takéhoto kontajneru.

Toto prirovnanie si môžeme preniesť aj na vývoj softvéru. V prípade, keď si vytvoríme kontajner pomocou nástroja Docker, takýto kontajner bude nezávislý od prostredia v ktorom sa nachádza a bude sa správať rovnako na každom type zariadenia.

4.5.2 Docker Compose

V našej aplikácií si preto vytvoríme takýchto kontajnerov hneď niekoľko. Jeden pre **databázu**, jeden pre ***frontend***, ďalší ***backend*** a napríklad jeden pre **nginx**. Takéto rozdelenie má množstvo výhod pri škálovateľnosti softvéru. My ale v našej aplikácií využívame hlavne fakt, že aplikácia sa bude správať predvídateľne.

Pri vytváraní kontajnerov nám bude slúžiť ešte jeden nástroj a to Docker Compose. Ide o nástroj ktorý zabezpečí, že pri nadefinovaní súboru typu .yml sa všetky kontajneri spustia a nastavia sa im jednotlivé vlastnosti, či spustia potrebné príkazy. Je to pomocný nástroj na to, aby sme nemuseli stále opakovať príkazy na spustenie kontajnerov manuálne, kedy by sa mohli vyskytnúť zbytočné chyby.

V našej aplikácií tento nástroj zabezpečí napríklad aj to, že celú aplikáciu pri prenesení súborov na ľubovoľné zariadenie by sme mali jednoducho spustiť. Pri predpoklade, že má používateľ nainštalovaný Docker a Docker Compose, spustenie prebieha jediným príkazom a to „*docker-compose up -d --build"*.

4.6 Celková štruktúra aplikácie

Celkovo tak aplikácia pozostáva v podstate dvoch aplikácií spolu s obslužnými nástrojmi. Serverovú aplikáciu vieme spustiť aj samostatne, pomocou jazyka Python a virtuálneho prostredia, ktoré sa spravidla používa. Rovnako tak vieme nainštalovať a spustiť klientsku časť aplikácie pomocou iných nástrojov.

Takéto spúšťanie je ale vysoko neefektívne a prichádzame o niektoré benefity. Napríklad sme zistili, že v prípade takéhoto spustenia sa zmeny v aplikácií neprejavujú okamžite, či stránka sa automaticky neaktualizuje.

Primárny spôsob spúšťania našej aplikácie bude Docker a Docker Compose, čo nám zabezpečí konzistentnosť a jednoduchosť naprieč celým vývojom aplikácie.

4.7 Implementácia systému a používateľského rozhrania

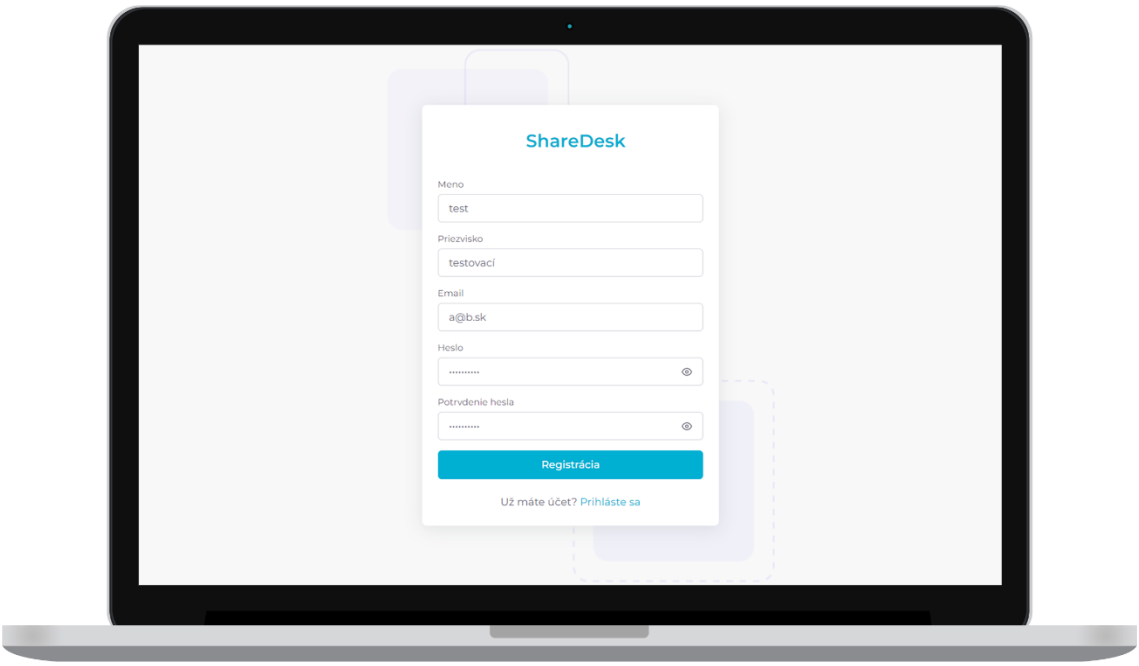
V nasledujúcej časti si priblížime reálnu implementáciu aplikácie, pre jej vybrané časti. Pomocou ukážok sa pozrieme sa na dizajn. Pri ukážkach si ale priblížime, akú majú tieto časti logiku, prípadne aké dáta sú potrebné. Ukážeme si na čo sme sa v aplikácií sústredili a ako sme to implementovali.

Nasledujúce pohľady budú z role admina, takže budeme mať prístup ku všetkým funkcionalitám. Netreba ale zabúdať na to, že niektoré tlačidlá, alebo celé stránky budú pred zamestnancami skryté.

4.7.1 Registrácia

Ako prvou časťou ktorú si ukážeme bližšie je registrácia do našej aplikácie. Ako môžeme vidieť na Obrázok 4.22, v registrácií vyžadujeme len tie najnákladnejšie údaje od

používateľa ako je email, meno, priezvisko a heslo. Pri nesprávnom vyplnených týchto údajov nás aplikácia graficky upozorní a vyznačí pole, ktoré by nebolo v poriadku. Overenie prebieha ako na strane klienta, tak aj na strane servera.

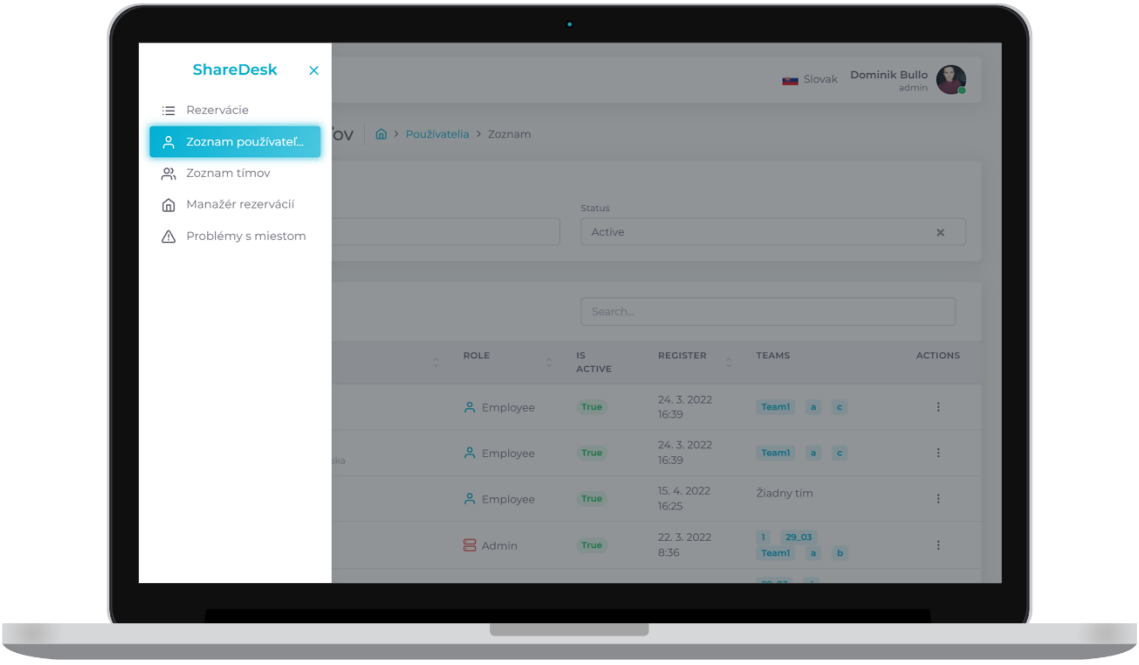


Obrázok 4.22 Ukážka registrácie

Po úspešnom zaregistrovaní sa používateľovi zobrazí notifikácia s potvrdením úspešnosti a následne bude používateľ presmerovaný na stránku prihlásenia. Na tejto stránke môže zadať svoje údaje a ihneď sa do aplikácie prihlásiť.

4.7.2 **Menu**

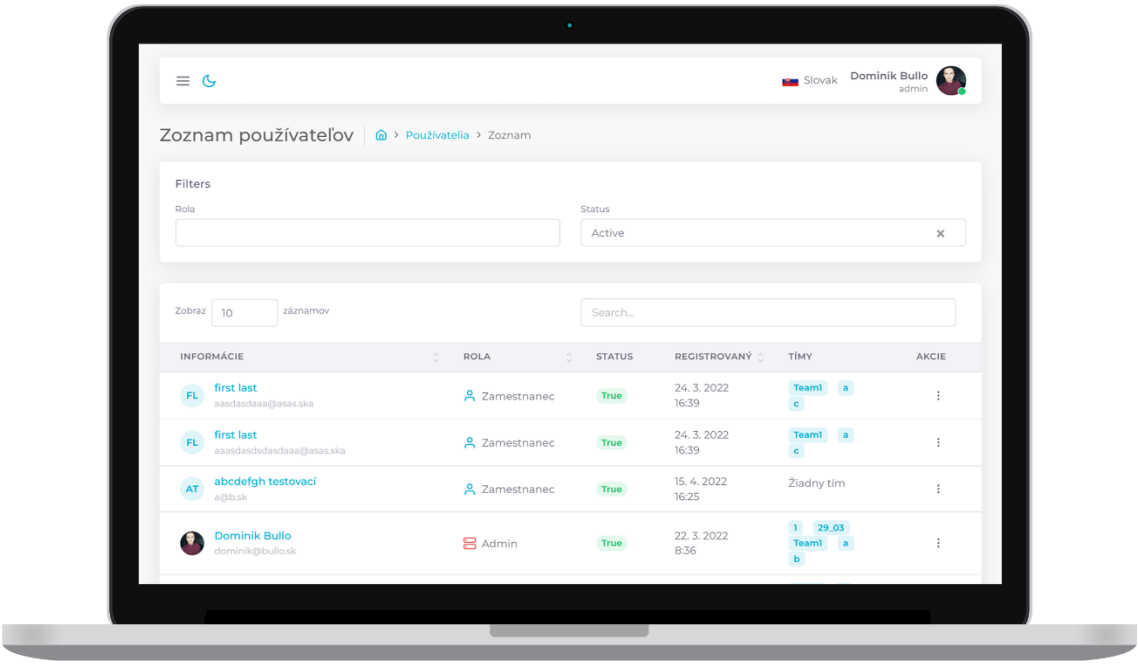
Po prihlásení do aplikácie, sa používateľ dostane na stránku rezervovania pracovného miesta, keďže predpokladáme, že táto stránka bude najviac navštevovaná. Ak by používateľ ale nechcel ísť na túto stránku, má k dispozícii menu, ktoré mu umožní rýchle smerovanie na jednotlivé časti aplikácie. Toto menu môžeme vidieť na Obrázok 4.22.



Obrázok 4.23 Ukážka menu

4.7.3 **Zoznam používateľov**

Zo spomínaného menu sa vieme jedným klikom dostať napríklad k zoznamu všetkých používateľom aplikácie. Tento zoznam zobrazujeme v prehľadnej tabuľke, ako je možné vidieť na Obrázok 4.24.



Obrázok 4.24 Zoznam používateľov

Z takéhoto zobrazenia pomocou tabuľky môžeme benefitovať jeho prehľadnosťou a intuitívnosťou. Okrem toho jej informácie sa dajú okamžite vyfiltrovať podľa želaných

parametrov. Príkladom môže byť používateľská rola alebo status. Takéto filtrovanie môžeme vidieť vo vrchnej časti obrázka.

Ešte detailnejšie manipulovanie s údajmi je možné nižšie, priamo v tabuľke ktorá zobrazuje dáta. Túto tabuľku má používateľ možnosť zoraďovať vzostupne či zostupne na základe jej stĺpcov.

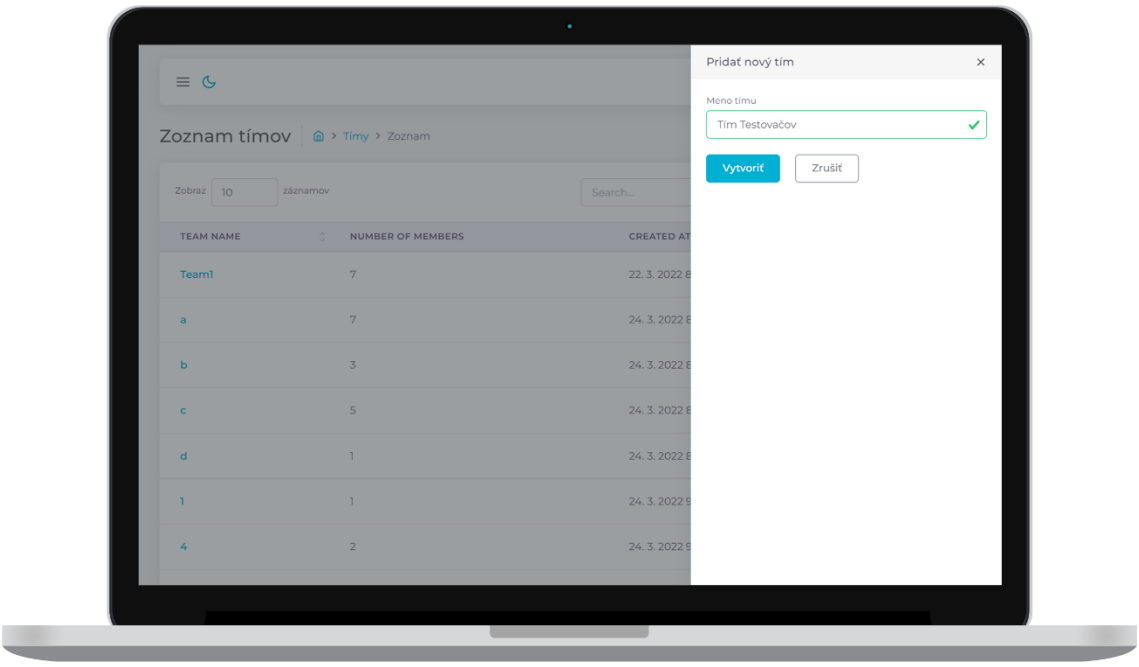
V neposlednom rade vieme v tabuľke hľadať podľa zadaného textu. Pri zadaní textu nám tabuľka vráti všetky záznamy, ktoré v ktorejkoľvek hodnote obsahujú zadaný text.

Všetky požiadavky na filtrovanie, zoradenie aj vyhľadávanie prebiehajú pomocou servera, resp. požiadavky na server. To znamená, že tabuľka pri akcií odošle žiadosť na server, ktorý vráti potrebné údaje zodpovedajúce parametrom. Klientska aplikácia následne tieto údaje spracuje a zobrazí v tabuľke.

Ako posledným stĺpcom spravidla bývajú rôzne ďalšie akcie, ktoré je možné robiť so záznamom. V našom konkrétnom prípade máme po kliknutí na tri bodky možnosť zobraziť detail používateľa alebo ho vymazať. Samozrejme, ak máme príslušné práva.

4.7.4 Vytváranie tímov

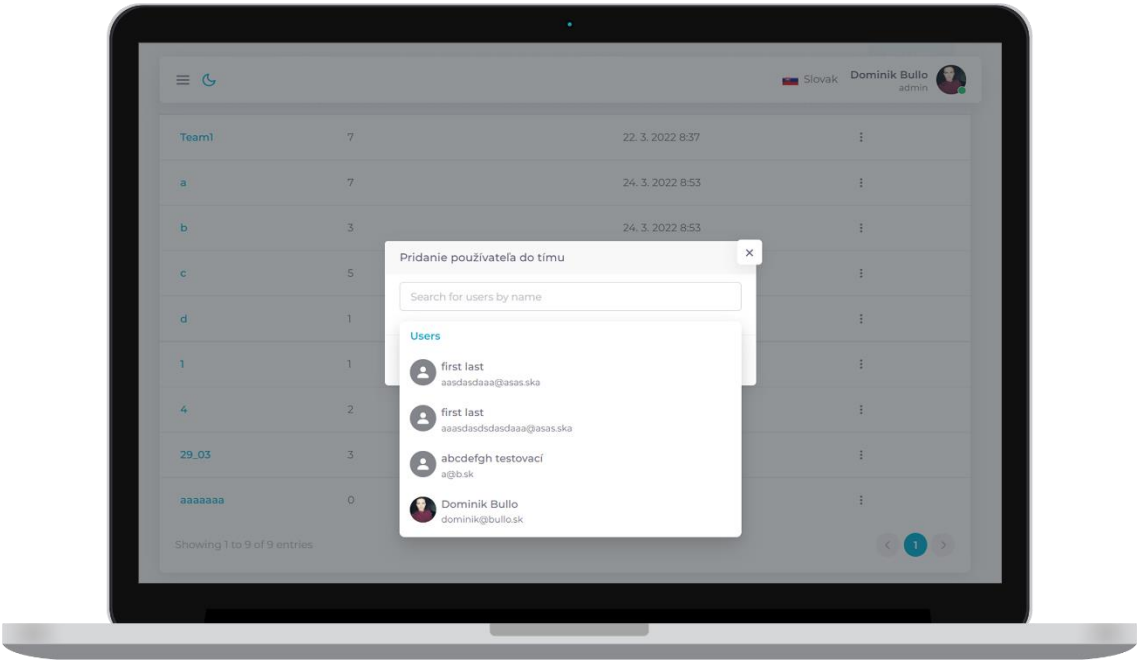
Tímy a ich vytváranie tímov funguje z časti veľmi podobne ako pri používateľoch. Rovnako aj tu sa nám zobrazí prehľadná tabuľka, ktorá disponuje stĺpcami, ktoré chceme používateľovi zobraziť. Znovu obsahuje možnosť vyhľadávania, či možnosť triedenia údajov.



Obrázok 4.25 Vytvorenie tímu

Prvou funkcionalitou navyše oproti tabuľke používateľov je, že pribudlo tlačidlo na pridanie tímu. Po jeho kliknutí sa nám zobrazí bočný panel, ako môžeme vidieť na Obrázok 4.25. V tomto panely môžeme vyplniť názov tímu. Na príklade môžeme vidieť, vizuálne potvrdenie správnosti údajov pomocou symbolu a zafarbenia políčka zelenou farbou. Následne môžeme kliknúť na tlačidlo vytvorenia tímu a takýto tím je vytvorený a zobrazený v zozname všetkých tímov.

V tejto tabuľke pribudla nová akcia. Okrem možnosti zobraziť detail a vymazania ako tomu bolo pri používateľoch, je pri tímoch navýšená o pridanie používateľa do tímu.



Obrázok 4.26 Pridanie používateľov do tímu

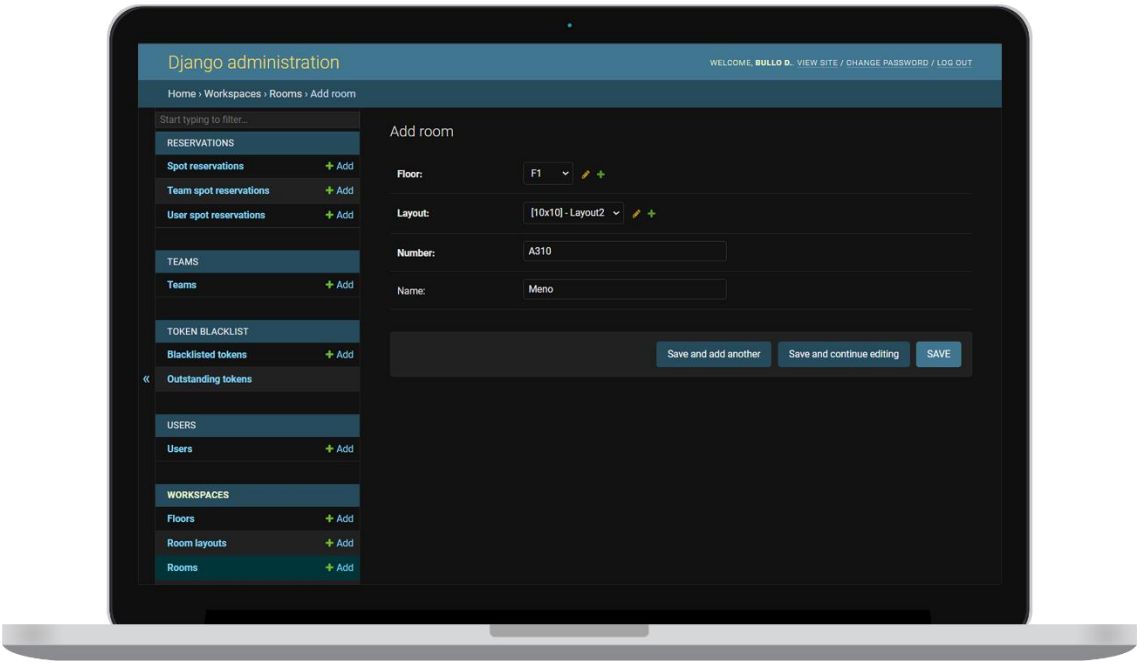
Po kliknutí na túto možnosť sa otvorí vyskakovacie okno viď. Obrázok 4.26. Toto okno obsahuje políčko, ktoré nám ponúka na výber z používateľov aplikácie, ktorých je možné do tohto tímu vybrať. Dôležité je pritom dodať, že takéto pole nám nebude zbytočne ponúkať používateľov, ktorý už v danom tíme sú, ale len takých, ktorých je do neho reálne možné pridať.

Pohodlnosť a jednoduchosť v tomto komponente zabezpečuje vyhľadávanie, ktoré je viazané na meno používateľa a taktiež aj fakt, že takýmto spôsobom môžeme naraz pridať niekoľko používateľov. Pri ich zvolení sa nám zobrazia v okne a následne kliknutím na tlačidlo pridania sa títo používatelia pridajú do tímu.

4.7.5 **Vytvorenie pracovísk**

Pri našej analýze vyplynulo, že pracoviská budú vytvorené raz a potom sa už výrazným spôsobom nebudú meniť. Keďže takéto správanie aplikácie nepotrebujeme pre všetkých používateľov a je predpoklad, že takáto vec sa nastaví raz jedným používateľov naše pracoviská sa vytvárajú pomocou administrácie Djanga.

Po prihlásení môže používateľ vytvoriť pracoviská, poschodia, rozloženia miestností a samotné miestnosti. Následne sa v týchto miestach automaticky vytvoria miesta. Ukážku rozhrania môžeme vidieť na Obrázok 4.27.



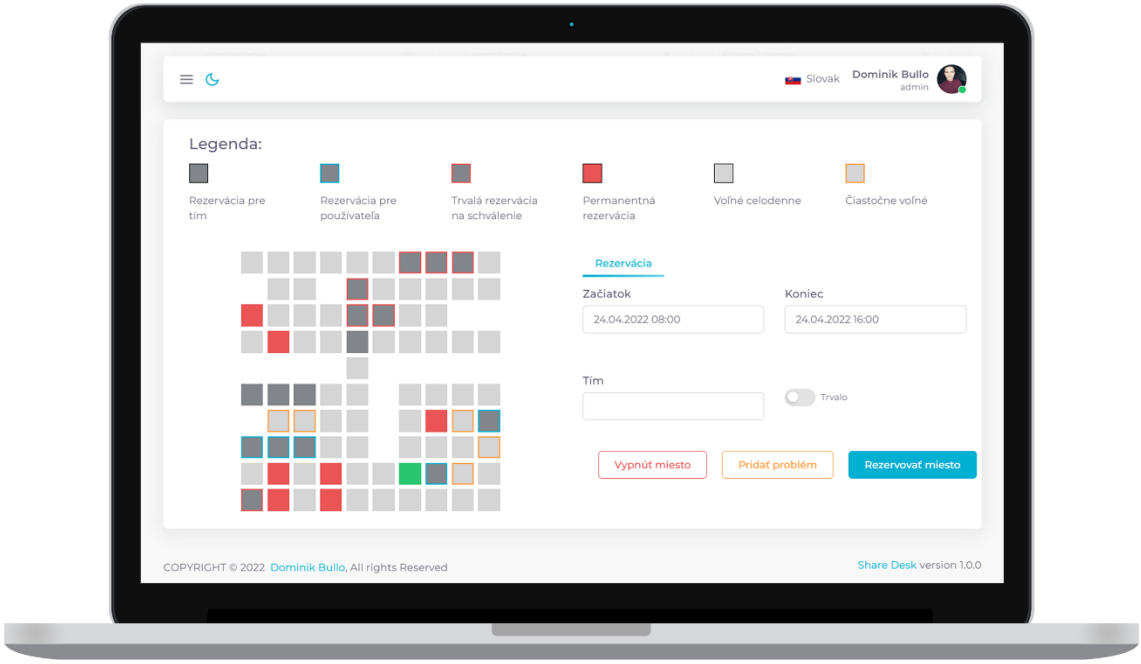
Obrázok 4.27 Rozhranie pre vytváranie pracovísk

4.7.6 **Rezervácia miesta**

Takto vytvorené miestnosti sa následne zobrazia na *frontend-e*, ako môžeme vidieť na Obrázok 4.28 a Obrázok 4.29. Tieto miestnosti majú dané rozloženie podľa nastavení administrátorov.

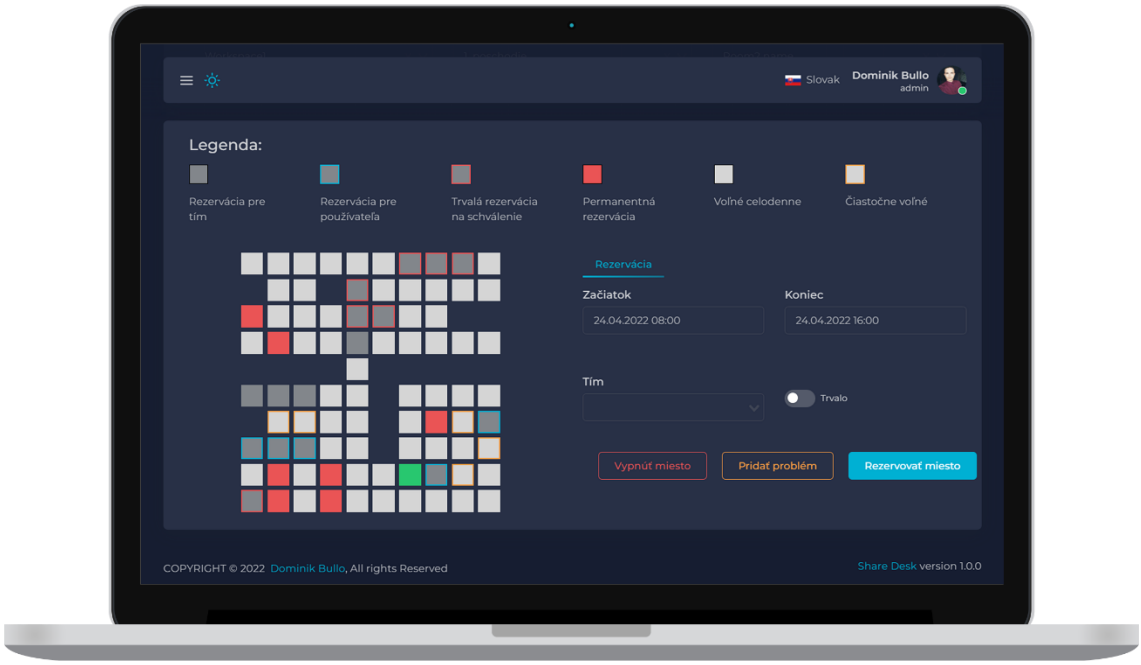
Aplikácia nás navádza tak, aby sme najskôr vybrali dátum a potom postupne budovu, poschodie, či miestnosť v ktorej chceme vybrať naše miesto. Pri vybratí potrebných možností sa nám zobrazí rozloženie danej miestnosti spolu s rôznymi zafarbeniami.

Tieto zafarbenia majú veľký význam pre rýchlu orientáciu v rezerváciách. Noví používatelia by ale nemuseli úplne intuitívne vedieť, aké farebné označenie je používané pre typ miesta. Preto sa v aplikácií nachádza aj legenda, ktorá nám vlastnosti týchto zafarbení popisuje bližšie.



Obrázok 4.28 Rezervácia pracovných miest svetlý mód

Toto farebné rozdelenie je veľmi dôležité a ideálne by malo byť nemenné. Farby sú vybrané tak, aby rozdelenie vyzeralo dobre, bolo nemenné a rozlíšiteľné aj na svetlej (Obrázok 4.28) aj na tmavej (Obrázok 4.29) téme. Vysvetlivky samotné by mali dobre popisovať, čo jednotlivé označenia znamenajú a načo sa používajú.



Obrázok 4.29 Rezervácia pracovných miest tmavý mód

Dôležitým detailom pri čiastočne voľnom mieste máme v aplikácii nastavenú hodnotu, ktorú považujeme za ešte voľné miesto. To znamená, že ak je táto hodnota nastavená napríklad na 3h, tak v prípade ak máme rezervácie, ktoré nechajú miesto voľné aspoň na

3h v daný deň, miesto označíme ako čiastočne voľné. V opačnom prípade ho označíme ako rezervované

Ďalej tu môžeme vidieť, po kliknutí na miesto sa nám zobrazí možnosť rezervácie pracovného miesta. Pri podržaní klávesy Ctrl spolu s kliknutím môžeme zvoliť naraz viacero miest. Pri takomto riešení musíme ale myslieť aj na dotykové displeje. Tie vie aplikácia zistiť a zobraziť na nich špeciálne tlačidlo pre zapnutie tejto funkcionality.

Pri vypĺňaní samotnej rezervácie miesta stoja za zmienku aj drobné detaily, napríklad, že dátum začiatku je nastavený na vybratý dátum rezervovania miesta, rovnako tak aj dátum konca. Tieto dátumy majú nastavené aj predvolené časy začiatku a konca. Rovnako tak aj fakt, že pokiaľ vyplníme tím, rezervácia bude pre celý tím, ale v prípade, že políčko tím ostane prázdna, rezervácia bude len na používateľa samotného.

Poslednou kolónkou je možnosť trvalej rezervácie. Takáto rezervácia sa najskôr dá do čakajúceho stavu. Čo sa s ňou ďalej môže stať si popíšeme v ďalšej časti.

V pravej dolnej časti obrázka môžeme vidieť tri tlačidla. Prvý z nich je tlačidlo na vypnutie sedadla. Tlačidla sa zobrazujú podľa používateľských rolí, čo znamená, že takéto tlačidlo vidí len admin. Tlačidlo funguje na vypnutie miest a teda vybudovanie lepšieho vizuálneho rozloženia miestnosti.

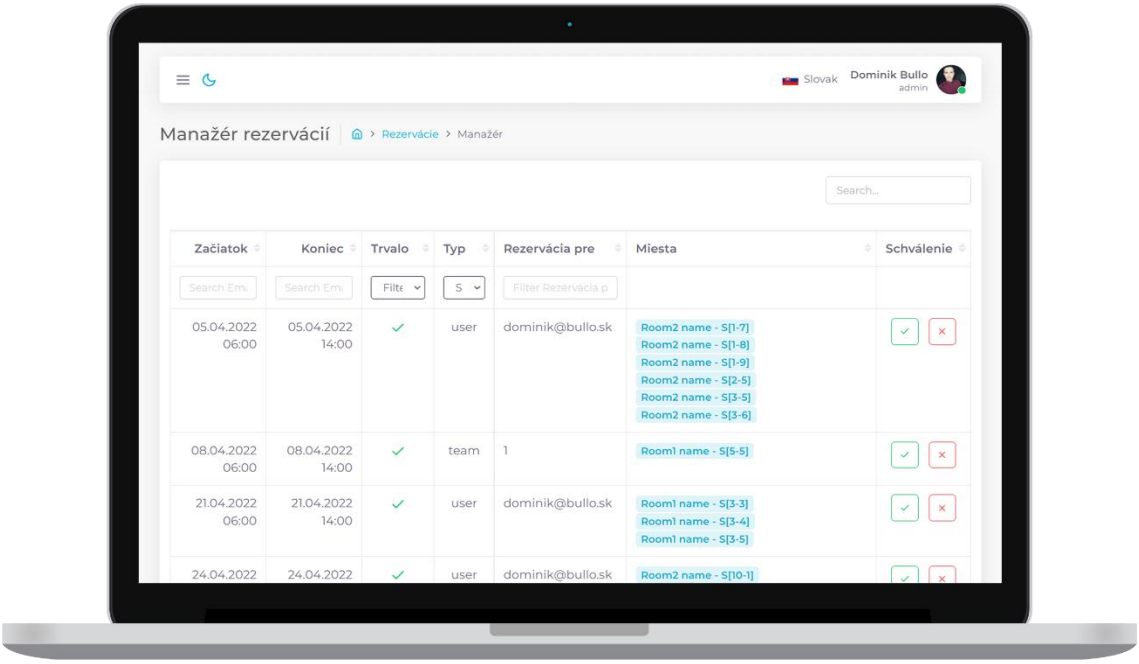
Ďalším tlačidlom je pridanie problému, ktoré sa zobrazí len pri vybraní jedného miesta. K tomuto miestu je potom možné pridať problém, ktorý pri ňom nastal. Viac do detailov tomuto procesu sa taktiež budeme ešte venovať v našej práci.

Ako posledným z radu týchto tlačidiel, je tlačidlo samotnej rezervácie, kedy sa po jeho kliknutí ešte zobrazí vyskakovanie okno na potvrdenie a následne sa vykoná samotná rezervácia.

4.7.7 Dlhodobé rezervovanie miesta

Pri odoslaní takejto rezervácie dostane užívateľ notifikáciu o jej úspešnosti. V prípade dlhodobej rezervácie používateľa aplikácia upovedomí, že jeho miesto je zatiaľ v procese schvaľovania a miesto je v statuse čakajúceho na schválenie.

Takéto schvaľovanie sa deje pomocou manažéra dlhodobých rezervácii, čo môžeme vidieť na Obrázok 4.30.



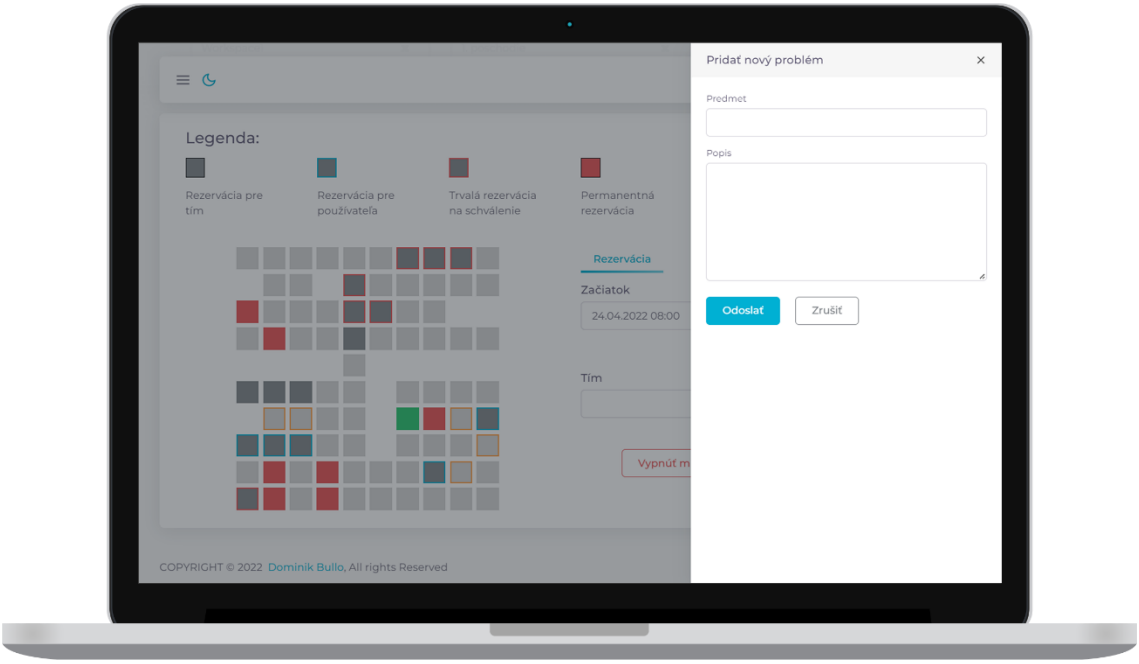
Obrázok 4.30 Manažér rezervácií

Na tejto stránke vidíme tabuľku, ktorou najdôležitejšiu funkcionalitou je schváliť, alebo zamietnuť dlhodobú rezerváciu. Administrátorom ponúkneme všetky dostupné informácie o takejto rezervácií, opäť v podobe prehľadnej tabuľky. Čo je v tomto prípade iné, je prístup k filtrovaniu, či vyhľadávaniu. V tejto tabuľke využívame len klientsku časť na filtrovanie a vyhľadávanie dát.

V procese schvaľovania už neumožňujeme iným používateľom toto miesto rezervovať. V prípade schválenia dlhodobej rezervácie je to rovnako. Takéto miesto je po schválení označené červenou farbou a nie je možné ho zarezervovať. V prípade zamietnutia dlhodobej rezervácie sa miesto správa ako voľné, len pri ňom máme uchovaný údaj o jeho zamietnutí.

4.7.8 **Problémy**

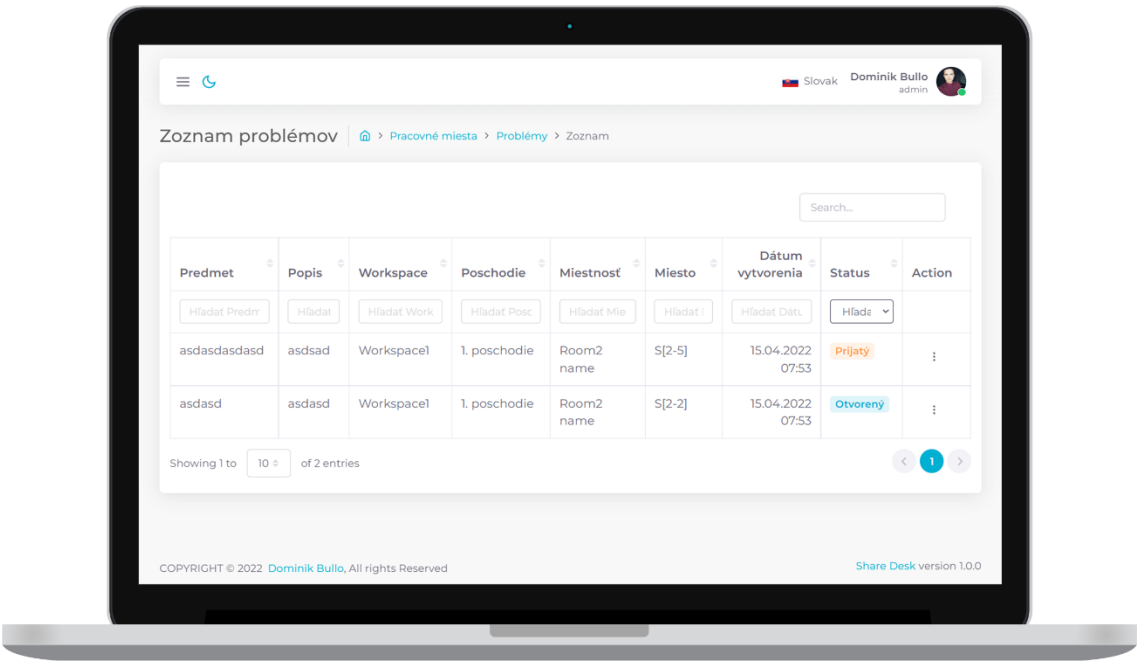
Pre nahlasovanie problémov v aplikácií, sme zvolili podobný bočný panel ako pri vytváraní tímov viď. Obrázok 4.31. Ako sme už opisovali nahlasovanie funguje pomocou výberu sedadla a následného kliknutia na tlačidlo. Tu autor problému vyplní požadované informácie o probléme a následne ich odošle.



Obrázok 4.31 Vytvorenie problému

Takýto problém je ale potrebné niekde zobraziť, na to aby ho následnej mohol niekto aj riešiť. Toto prebieha pomocou tabuľky, ako môžeme vidieť na Obrázok 4.31. Tu prebieha zobrazenie jednotlivých problémov. Táto tabuľka opäť využíva len klientsku časť na filtrovanie a vyhľadávanie.

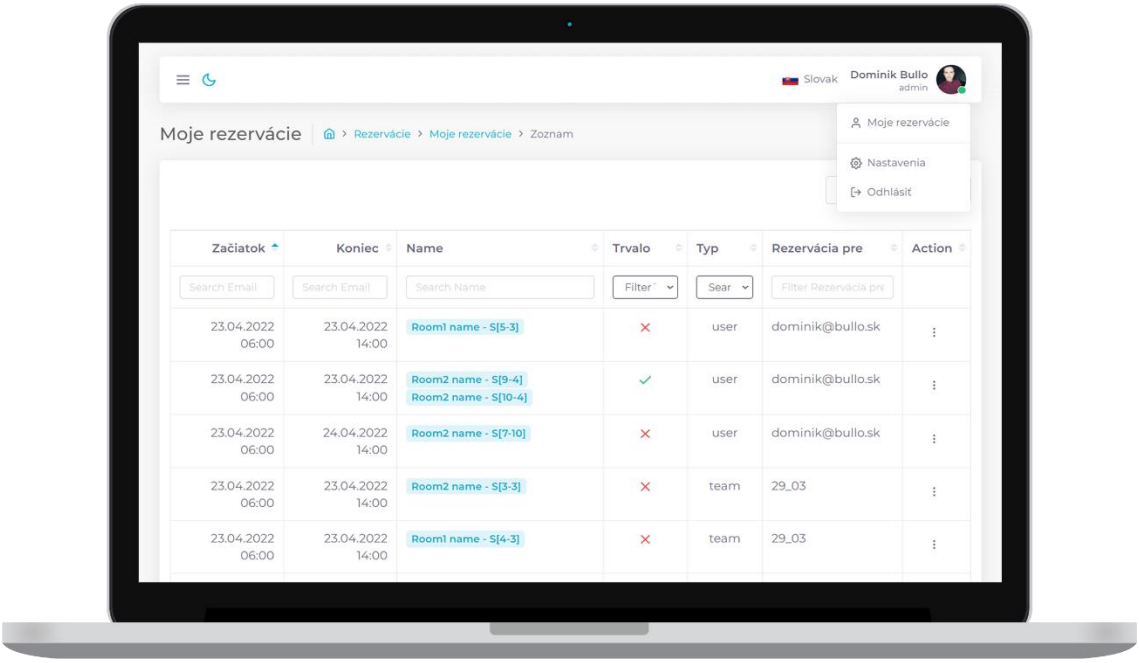
Za zmienku stojí hlavne to, že vidíme status daného problému. To znamená, že vidíme či bol nahlásený, otvorený, v riešení alebo vyriešený. Toto napomôže lepšiemu prehľadu problémov a menej duplicít pri nahlasovaní problémov.



Obrázok 4.32 Zoznam problémov

4.7.9 **Moje rezervácie**

Pre nahlásenia problému spätne by používateľ zrejme rád vedel, na ktorom mieste sedel. Alebo to môže byť aj pre prípad, že miesto sa mu páčilo a chce si ho rezervovať znova. V takejto situácií môže po kliknutí v pravom hornom rohu zvoliť položku, ktorá ho presmeruje na jeho rezervácie. V tejto tabuľke vidí komplet všetky rezervácie určené pre neho, ale rovnako tak aj rezervácie, ktoré boli vykonané tímov v ktorom sa nachádza.



Obrázok 4.33 Pohľad na rezervácie prihláseného používateľa

5 VERZIOVANIE A PRODUKČNÉ NASADENIE

V požiadavkách našej práce bolo špecifikované aj produkčné nasadenie aplikácie. V tejto kapitole si priblížime to, aké nástroje sme používali počas vývoja, aby sme zabezpečili čo najplynulejší vývoj a následne jednoduché a vo výsledku aj automatické nasadenie aplikácie.

5.1 Git

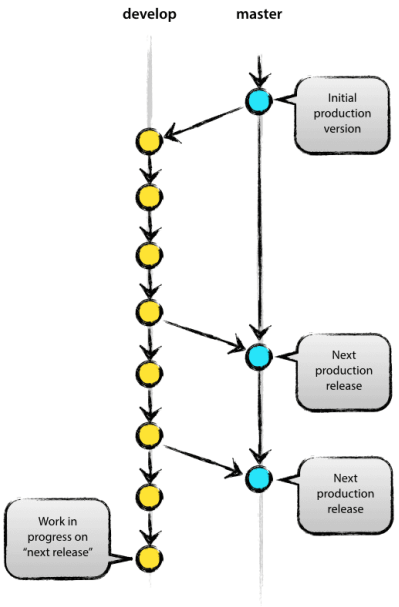
Git je najbežnejšie používaný systém na správu verzií. Git sleduje zmeny, ktoré vykonáte v súboroch, takže máte záznam o tom, čo ste urobili, a môžete sa vrátiť ku konkrétnym verziám, ak by ste to niekedy potrebovali. Takýto systém sa používa vo veľkých tímoch, ale aj pri malých vývojárov.

V našom konkrétnom prípade používame git spolu so službou Github, ale je možné ho použiť aj s inými službami ako napríklad Bitbucket či Gitlab.

Git a jeho „*branch*“ sú veľmi užitočným nástrojom pri vývoji softvéru. Poďme sa spolu pozrieť na to, ako tieto vetvy využívame pri vývoji a nasadení nášho systému.

5.1.1 Git vetvy

Naša aplikácia bude vychádza z dlhodobo používaného modelu verziovania [10]. Ako môžeme vidieť na Obrázok 5.1 základom tohto modelu sú dve vetvy konkrétne **master** a **develop**.



Obrázok 5.1 Git vetvenie

Myšlienka tohto modelu je taká, že vetva ***master***, slúži len na produkčné verzie systému a obsahuje vždy plne funkčnú verziu systému pripravenú na nasadenie.

Zatiaľ čo ***develop*** je vetva, z ktorej sa buď vytvárajú ďalšie vetvy pre jednotlivé funkcie softvéru, alebo sa do nej pripájajú. Môže obsahovať aj verziu aplikácie, ktorá nie je plne pripravená na produkčné nasadenie, alebo obsahuje chyby.

Výhodou tohto modelu je, že teraz máme k dispozícií aplikáciu z vetvy ***master***, ktorá bude vždy pripravená do produkcie. Existuje viacero spôsobov, ako by sme takúto aplikáciu mohli nasadiť do produkcie, v našej práci sme sa rozhodli pre plne automatizovaný postup.

5.2 Github Actions

CI/CD je skratka, ktorou sa označuje termín pre sériu krokov, ktoré je potrebné vykonať na správne vydanie novej verzie systému. Implementácia v našej aplikácií je riešená pomocou služby *Github Actions*.

Táto služba nám umožňuje, primárne zostavenie (*build*), unit testovanie a nasadenie (*deploy*) aplikácie priamo zo služby GitHub. V prípade, že by unit testy neboli úspešné, nástroj nás na to automaticky upozorní a aplikáciu pochopiteľne v tejto verzií nenasadí na server. Pre lepšiu vizualizáciu uvádzame Obrázok 5.2.

Obrázok 5.2 Príklad správania CI/CD [11]

5.3 Nasadenie na server

Pri vývoji sme používali lokálny server resp. Docker na vlastnom počítači, ktorý bol úplne postačujúci pre vývojárske účely. Takéto riešenie je postačujúce pre vývoj, ale aplikácia je len na počítači vývojára.

Preto sme potrebovali nájsť vhodný spôsob nasadenia. Tým je zatiaľ súkromný server. Je to z toho dôvodu, aby aplikácia bola verejne prístupná a dala sa testovať, rovnako tak aj preto, aby po testovaní bolo možné aplikáciu hneď nasadiť na vlastný server spoločnosti.

Aplikácia je momentálne dostupná na súkromnej subdoméne sharedesk.bullo.sk. Jej DNS záznamy sú nasmerované na IP adresu poskytovateľa servera. Náš server je nastavený tak, aby prijímal žiadosti na túto verejnú URL adresu. Takisto je na ňom zabezpečené aj to, že obsahuje potrebné súbory a prístupy (napríklad na Github), či environmentálne premenné, tak aby sme zabezpečili správny chod aplikácie.

Pri takomto nastavení servera je pri novej verzií aplikácie potrebných len pár príkazov v príkazovom riadku a nová verzia aplikácie bude nasadená a prístupná. Toto všetko sa deje v našom prípade automaticky.

5.4 Spôsob produkčného nasadenia

Keď už sme si uviedli akú verziu softvéru nasadíme a kde túto verziu nasadíme, pre lepšie pochopenia celej logiky si uvedieme diagram takéhoto spôsobu nasadenia. Takýto spôsob opäť vychádza zo zaužívaných spôsobov pri vývoji aplikácií. V prílohe č. 1 môžeme vidieť celý tok aplikácie.

Tieto kroky sú plne automatizované a ich výsledkom je:

1. Otestovanie pomocou unit testov.
2. Publikovanie najnovších Docker image do príslušných repozitárov.
3. Nasadenie aplikácie na produkčný server.

6 TESTOVANIE A MOŽNÝ ROZVOJ APLIKÁCIE

6.1 Unit testy

Počas takmer celého vývoja sme aplikáciu testovali pomocou unit testov. Jedná sa o testy, ktoré na základe nejakého predpokladu vyhodnocujú správanie aplikácie. Napríklad máme test, kde aplikácia skúma, či sa token, ktorý vytvoríme pre používateľa dá použiť aj pri odhlásení používateľa.

```
1  class TestLoginCase(APITestCase):
2      login_url = reverse('token_obtain_pair')
3      refresh_token_url = reverse('token_refresh')
4      logout_url = reverse('auth_logout')
5      profile_url = reverse('user-profile')
6
7      email = 'test@user.com'
8      password = 'kah2ie3urh4k'
9
10     def setUp(self):
11         self.user = get_user_model().objects.create_user(self.email, self.password)
12
```

Obrázok 6.1 Základné nastavenie pred každým testom

Ďalším zaujímavým testom, na ktorý by sme mali poukázať je test, ktorý skúma, či vytvorený token je platný aj po uplynutí jednej hodiny. Token by platný nemal byť a aplikácia by nám mala vrátiť status s kódom 401- Takéto správanie totižto taktiež očakávame od aplikácie a od spôsobu autentifikácie, ktorý sme použili. Kód takéhoto testu vidíme na Obrázok 6.2.

```
1  def test_access_token_invalid_in_hour_after_logout(self):
2      _, body = self._login()
3      self.client.post(self.logout_url, body)
4      m = mock.Mock()
5      m.return_value = aware_utcnow() + timedelta(minutes=60)
6      with mock.patch('rest_framework_simplejwt.tokens.aware_utcnow', m):
7          r = self.client.get(self.profile_url)
8          body = r.closed
9      self.assertEquals(r.status_code, 401, body)
10
```

Obrázok 6.2 Unit test pre kontrolu tokenu po uplynutí jednej hodiny

6.2 Možnosti rozvoja

Aplikácia bola vyvíjaná spolu s cieľom zabezpečiť čo najjednoduchšiu implementáciu ďalších funkcionalít, prípadne celých modulov. Tento prístup by mal zabezpečiť jednoduchú rozšíriteľnosť.

Ďalšie funkcionality záležia primárne od potrieb zamestnávateľa, či používateľov takejto aplikácie. Počas vývoja sme ale narazili na funkcionality, ktorými by mohla aplikácia disponovať.

Pre príjemnejší dojem s používania by mohla aplikácia disponovať napríklad nejaký vizuálnym nástrojom na jednotlivých rozložení pre miestnosti. Miestnosti by mohli obsahovať plány či fotky, ktoré by odrážali skutočné rozloženie pracovných miest. Takáto funkcionalita by ale bola zrejme dôležitejšia, ak by zamestnanci vôbec nepoznali pracovisko.

Takisto by mohla byť aplikácia rozšírená o notifikačný systém pomocou oznámení či emailom. Napríklad by poslala používateľovi notifikáciu, keby sa jeho rezervácia z nejakého dôvodu zmenila, alebo keď bola permanentná rezervácia prijatá, či zamietnutá.

Posledným návrhom na zlepšenie je rozšíriteľnosť o podrobnejšie štatistiky jednotlivých miest. Ich najčastejšie problémy či podrobná využiteľnosť, rovnako tak aj štatistika pre jednotlivých používateľov. To znamená, ako často jednotlivý používateľ používa pracovné miesta, aké sú jeho najobľúbenejšie miesta a pod.

Ako sa hovorí, kreativite sa medze nekladú a naša aplikácia môže byť ako dobrý základ slúžiaci na ďalší vývin funkcionalít.

ZÁVER

Výsledkom diplomovej práce je analýza, návrh a následne implementácia webovej aplikácie, ktorá bude slúžiť na rezervovanie pracovných miest jednotlivým zamestnancom spoločnosti Ringier Axel Springer SK, a.s. Aplikácia je navrhnutá podľa aktuálne najnovších a najpoužívanejších štandardov vo svete vývoja webových aplikácií. Aplikácia spĺňa vo všetkých bodoch požiadavky zadávateľa.

Je rozdelená na klientsku a serverovú časť. Serverová aplikácia je postavená na nástroji Django spolu s knižnicou Django REST. Klientska aplikácia je vytvorená pomocou nástroja Vue.js spolu s pomocou šablóny Vuexy.

Táto práca obsahuje popis ako aj implementáciu produkčného nasadenia, ktoré je plne automatizované. Aplikácia je verejne prístupná a momentálne prebieha testovanie, ktoré zabezpečí jej bezproblémové nasadenie do „ostrej“ prevádzky.

Pri vývoji aplikácie sme sa oboznámili s rôznymi nástrojmi, ktoré nám vedia vývoj zjednodušiť, zrýchliť alebo zlepšiť bezpečnosť. To hlavne za predpokladu, že prácu s takýmito nástrojmi už ovládame, alebo aspoň vieme základné koncepty, ktoré poskytujú. V prípade, žeby to tak nebolo, tieto *framework-y* môžu byť spočiatku veľmi náročné na pochopenie a vyžadujú si veľa času na pochopenie.

Ako najzaujímavejšiu technológiu s ktorou sme sa mohli v tejto práci a pri vývoji aplikácie stretnúť by sme mohli vyzdvihnúť koncept kontajnerizácie, konkrétne nástroj Docker. Pomocou tohto nástroja sme eliminovali značnú časť problémov, ktoré sú častokrát spojené s vývojom takéhoto typu aplikácie. Táto technológia je čoraz viac populárnejšia vo svete aplikácií. Technológia síce stále prebieha intenzívnym vývinom, takže sa môže stať, že dokonca aj naša aplikácia nebude fungovať bezproblémovo so všetkými verziami nástroja Docker, napriek tomu je to dostatočne spoľahlivý pre našu aplikáciu. Podľa autora tejto práce má veľký potenciál a určite to bola pozitívna skúsenosť.

Záverom našej práce by sme mohli spomenúť to, že aplikácia sa počas testovania ešte vyladí, opravia sa drobné chyby. Predpokladáme, že testovanie zrejme odhalí niektoré nedostatky aplikácie, ktoré budú následne opravené. Takýto postup pri vývoji je úplne bežný a potrebný pre aplikáciu samotnú, pretože správne fungovanie aplikácie sa overí až pri jej praktickom použití, a preto má testovanie v reálnej prevádzke nesmierny význam. Po tomto procese môže aplikácia naplno slúžiť svojmu účelu.

ZOZNAM POUŽITEJ LITERATÚRY

[1] „Perficient,“ [Online]. Available: https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage. [Cit. 04 2022].

[2] „Britannica,“ [Online]. Available: https://www.britannica.com/technology/client-server-architecture. [Cit. 03 2022].

[3] „Wikipedia,“ [Online]. Available: https://en.wikipedia.org/wiki/Client%E2%80%93server_model#cite_note-1. [Cit. 03 2022].

[4] „Software Planet Group,“ [Online]. Available: https://softwareplanetgroup.co.uk/graphql-vs-rest-what-to-choose-in-2020/. [Cit. 03 2022].

[5] „izertis,“ [Online]. Available: https://www.izertis.com/en/-/refresh-token-with-jwt-authentication-in-node-js. [Cit. 04 2022].

[6] [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Models. [Cit. 2022].

[7] „Geeks for Geeks,“ [Online]. Available: https://www.geeksforgeeks.org/serializers-django-rest-framework/. [Cit. 04 2022].

[8] „altexsoft,“ [Online]. Available: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/. [Cit. 04 2022].

[9] „vuex.vuejs.org,“ [Online]. Available: https://vuex.vuejs.org/.

[10] [Online]. Available: https://nvie.com/posts/a-successful-git-branching-model/. [Cit. 04 2022].

[11] „Red Hat,“ [Online]. Available: https://www.redhat.com/en/topics/devops/what-is-ci-cd. [Cit. 04 2022].

ČESTNÉ VYHLÁSENIE

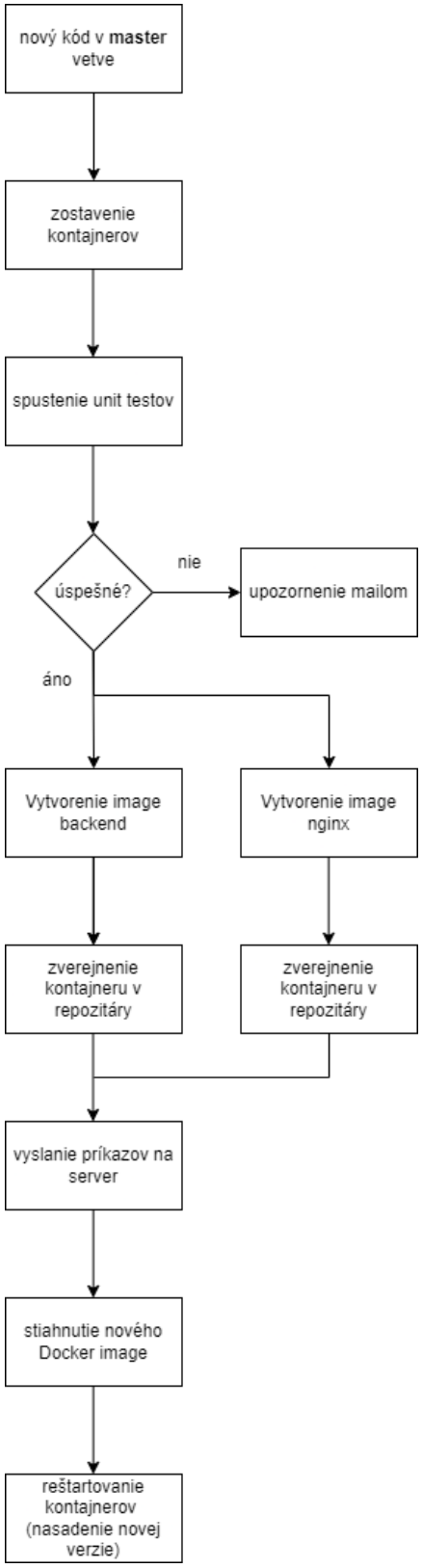
Vyhlasujem, že som zadanú diplomovú prácu vypracoval samostatne, pod odborným vedením vedúceho diplomovej práce Ing. Kečkešová Barbora a konzultanta diplomovej práce Ing. Miroslav Markovič, PhD. a používal som len literatúru uvedenú v práci.

V Žiline dňa 2. mája 2022

Bc. Dominik Bullo

PRÍLOHOVÁ ČASŤ

ZOZNAM PRÍLOH

Príloha č. 1 – Diagram procesu aplikácie pri novej verzií na Githube

Príloha č. 1 – Diagram procesu aplikácie pri novej verzií na Githube