ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

BAKALÁRSKA PRÁCA

DOMINIK BULLO

Webová aplikácia pre správu športového klubu

Vedúci práce: Ing. Patrik Hrkút, PhD.

Registračné číslo: 901/2019

Žilina, 2020

ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

BAKALÁRSKA PRÁCA

ŠTUDIJNÝ ODBOR:

INFORMATIKA

DOMINIK BULLO

Webová aplikácia pre správu športového klubu

Žilinská univerzita v Žiline

Fakulta riadenia a informatiky

Žilina, 2020

ZADANIE PRÁCE

ČESTNÉ VYHLÁSENIE

Čestne vyhlasujem, že som túto bakalársku prácu vypracoval samostatne, pod odborným dohľadom vedúceho práce a že som čerpal iba zo zdrojov uvedených v zozname použitej literatúry.

POĎAKOVANIE

Ďakujem vedúcemu bakalárskej práce za odbornú pomoc a rady pri tvorení práce, ktoré mi pomohli prácu úspešne dokončiť.

ABSTRAKT

BULLO, Dominik: *Webová aplikácia pre správu športového klubu*. [Bakalárska práca] – Žilinská Univerzita v Žiline. Fakulta riadenia a informatiky; Študijný program: Informatika – Vedúci práce: Ing. Patrik Hrkút, PhD. – Bakalár v odbore Informatika. – Žilina: FRI UNIZA, 2020 - Počet strán: **63**

Cieľom bakalárskej práce je vytvoriť webovú aplikáciu pre správu športového lyžiarskeho klubu. Jej účelom je zjednodušiť poskytovanie informácií o udalostiach, na ktorých sa členovia tohto klubu zúčastňujú, zber štatistík o jednotlivých športovcoch a ich výkonoch a taktiež možnosť zasielania notifikácii jednotlivým používateľom o pripravovaných udalostiach. Aplikácia sa skladá z klientskej aj serverovej aplikácie. Primárnou úlohou serverovej aplikácie je zabezpečovať aplikačnú logiku a prístup k dátam, zatiaľ čo klientska aplikácia predstavuje jednoduché a intuitívne rozhranie pre používateľov.

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

ABSTRACT

BULLO, Dominik: *Web application for sport club management*. [Bachelor's thesis] – University of Žilina in Žilina. Faculty of Management Science and Informatics. Field of study: Informatics. – Tutor: Ing. Patrik Hrkút, PhD, Degree of Professional qualification: Bachelor in the field of Informatics. Žilina: FRI UNIZA, 2020 - Total number of pages: **63**

The aim of the bachelor thesis is to create a web application for the management of a sports ski club. Its purpose is to simplify providing of information about events in which members of this club participate, the collection of data on individual athletes and their performances, and also the possibility of sending notifications to individual users about upcoming events. The application consists of a client and a server application. The primary task of the server application is to provide application logic and access to data, while the client application provides a simple and intuitive interface for users.

Key words: web app, Python, Django, JavaScript, Vue.js, booking system

OBSAH

Zoznam obrázkov **10**
Zoznam skratiek **11**
Úvod **12**
1 Cieľ práce **13**
2 Súčasný stav riešenej problematiky **14**
2.1 Diagram biznis tried 15
2.2 Ukážky aktuálneho riešenia 15
2.3 Výhody aktuálneho stavu 17
2.4 Nevýhody aktuálneho stavu 17
3 Analýza existujúcich riešení **18**
3.1 Rezervačný systém iSport 19
3.2 Reservio 20
3.3 Reservanto 21
3.4 Sportigio 22
3.5 Zhodnotenie 22
4 Analýza aplikácie **23**
4.1 Popis procesov v klube 23
4.2 Mimofunkčné požiadavky 24
4.3 Multiplatformovosť 24
4.4 Vzťah rodič - dieťa 24
4.5 Udalosti 24
4.6 Upozornenia 25
4.7 Štatistiky 25
4.7.1 Sezóna a kategórie 25
4.8 Dodatočné funkcionality 25
4.9 Aktori a role v systéme 26
4.10 Pohľad dieťaťa 26
4.11 Pohľad rodiča 26
4.12 Pohľad trénera 27
4.13 Pohľad admina 28
4.14 Celkový diagram prípadov použitia 30
5 Návrh aplikácie **31**
5.1 Architektúra aplikácie 31
5.1.1 Klient-server architektúra 31
5.2 Komunikácia so vzdialenými službami 32

5.2.1 SOAP ... 32
5.2.2 RESTfull API ... 32
5.2.3 GraphQL ... 33
5.2.4 Výber technológie ... 33
5.3 Návrh serverovej časti aplikácie ... 33
5.3.1 MVC (MTV) ... 34
5.3.2 Zabezpečenie zdrojov ... 35
5.4 Návrh databázy ... 35
5.4.1 Používatelia ... 36
5.4.2 Rodina ... 37
5.4.3 Kategórie a sezóny ... 38
5.4.4 Udalosti ... 39
5.5 Návrh klientskej časti aplikácie ... 40
5.5.1 Single-page aplikácie ... 41
5.6 PWA ... 42
6 Implementácia ... 44
6.1 Vývoj serverovej aplikácie ... 44
6.1.1 Django ... 44
6.1.2 Štruktúra projektu ... 44
6.1.3 Modely ... 46
6.1.4 Administrátorské rozhranie ... 46
6.1.5 URL adresy ... 48
6.1.6 Django Rest Framework ... 48
6.1.7 Views ... 49
6.1.8 Serializers ... 49
6.1.9 Koncový bod pre udalosti ... 50
6.2 Vývoj klientskej aplikácie ... 52
6.2.1 Vue.js ... 52
6.2.2 Vuexy ... 53
6.2.3 Vuex ... 53
6.2.4 Fullcalendar ... 54
6.3 Problémy spojené s vývojom webovej aplikácie ... 56
6.4 Docker ... 56
6.4.1 Kontajner ... 57
6.4.2 Docker Compose ... 57
6.5 Štruktúra celého projektu ... 57
6.6 Nasadenie aplikácie na server ... 58

6.7 Notifikácie ... 58
7 Testovanie a možnosti rozvoja aplikácie ... 60
7.1 Github Actions ... 60
7.2 Možnosti rozvoja ... 60
Záver ... 61
Zoznam použitej literatúry ... 62

ZOZNAM OBRÁZKOV

Obrázok 2.1 - Diagram biznis prípadov použitia 14
Obrázok 2.2 - Diagram biznis tried 15
Obrázok 2.3 - Pohľad na súčasne riešenie na PC 16
Obrázok 2.4 - Pohľad na súčasne riešenie na mobile 16
Obrázok 3.1 - Ukážka systému iSport 19
Obrázok 3.2 - Ukážka systému Reservio 20
Obrázok 3.3 - Ukážka systému Reservanto 21
Obrázok 3.4 - Ukážka systému Sportigio 22
Obrázok 4.1 - Diagram prípadov použitia - rodič a dieťa 27
Obrázok 4.2 - Diagram prípadov použitia – tréner a admin 29
Obrázok 4.3 - Diagram prípadov použitia pre všetkých aktorov 30
Obrázok 5.1 - MTV model Django [6] 35
Obrázok 5.2 - Databázový diagram - používatelia 36
Obrázok 5.3 - Databázový diagram - rodina 37
Obrázok 5.4 - Databázový diagram - kategórie a sezóny 38
Obrázok 5.5 - Databázový diagram - udalosti 39
Obrázok 5.6 - Databázový diagram 40
Obrázok 6.1 - Ukážka administrátorského rozhrania - Event 47
Obrázok 6.2 – Ukážka api zoznamu udalostí 51
Obrázok 6.3 - Pohľad na fungovanie Vuex 54
Obrázok 6.4 - Pohľad na kalendár udalostí 55

ZOZNAM SKRATIEK

API	**A**pplication **P**rogramming **I**nterface
CSV	**C**omma-**S**eparated **V**alues
DOM	**D**ocument **O**bject **M**odel
PWA	**P**rogressive **W**eb **A**pplication (**P**rogresívna **w**ebová **a**plikácia)
SPA	**S**ingle-**p**age **A**pplication
GUI	**G**raphical **U**ser **I**nterface
HTML	**H**yper**T**ext **M**arkup **L**anguage
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol
HTTPS	**H**yper**T**ext **T**ransfer **P**rotocol **S**ecure
IDE	**I**ntegrated **D**evelopment **E**nvironment
IP	**I**nternet **P**rotocol
JSON	**J**ava**S**cript **O**bject **N**otation
MVC	**M**odel-**V**iew-**C**ontroller
MTV	**M**odel-**T**emplate-**V**iew
ORM	**O**bject-**R**elational Mapping
REST	**RE**presentational **S**tate **T**ransfer
URI	**U**niform **R**esource **I**dentifier
URL	**U**niform **R**esource **L**ocator
UX	**U**ser **E**xperience
UI	**U**ser **I**nterface
SOAP	**S**imple **O**bject **A**ccess **P**rotocol

ÚVOD

Šport patrí k môjmu životu odmalička. Už ako malý chlapec som chodieval do lyžiarskeho klubu, kde sa športovci stretávali s rovnakým zámerom. Byť najlepší v tom, čo robia. Na to je však potrebný tréning vedený profesionálnym trénerom. Je teda prirodzené, že jednotliví členovia klubu medzi sebou komunikujú a odovzdávajú si informácie ohľadom tréningov, pretekov a sústredení, ale aj napríklad o objednávkach spoločného oblečenia.

V dnešnej dobe emailov, sociálnych sietí, či rôznych aplikácií na komunikovanie, nie je problém, aby jednotlivci či skupiny medzi sebou navzájom komunikovali. Častým problémom býva, že v takýchto aplikáciách užívateľ prehliadne dôležité informácie. Môže to byť napríklad z dôvodu veľkého množstva informácií, ktorými sa takýto informačný kanál zahlcuje. Taktiež tu neexistuje žiadna história dát a ich následné automatické spracovanie.

Existujú však oveľa modernejšie a praktickejšie spôsoby ako riešiť tento problém. Jedným z nich je aj webová aplikácia, ktorá môže všetky procesy v klube uľahčiť a zefektívniť. Webové aplikácie sú v posledných rokoch na vzostupe a začínajú sa používať naozaj vo veľkom. Tento jav je pochopiteľný, keďže poskytuje niekoľko zásadných výhod. Webové aplikácie fungujú odkiaľkoľvek, kedykoľvek, nezávisle na operačnom systéme či zariadení. Taktiež nie je nutné sťahovanie žiadnej aplikácie, jej inštalovanie a následné aktualizácie.

Úlohou tejto aplikácie je uľahčiť jej požívateľom komunikáciu a výmenu rôznych informácií v klube. Táto vymieňa oboma smermi. Či už zo strany športovcov a rodičov smerom k vedeniu klubu formou potvrdenia účasti na udalosti. Rovnako ale aj zo strany vedenia klubu a trénerov k športovcom pri vypisovaní nových tréningov, či zmeny už vypísaných tréningov. Dôležitou súčasťou je taktiež možnosť automatického spracovania dát a ich poskytnutie na analýzu trénerom.

1 CIEĽ PRÁCE

Cieľom tejto práce je vytvoriť webovú aplikáciu na správu športového klubu, a to konkrétne lyžiarskeho klubu AC UNIZA Ski Team.

Prvým čiastkovým cieľom je zanalyzovať podobné aplikácie, ktoré by mohli poskytovať požadovanú funkcionalitu, a zároveň zhodnotiť ich výhody a nevýhody, z pohľadu požiadaviek na moju aplikáciu.

Následne budeme detailne popisovať funkcionalitu z rôznych pohľadov, a to konkrétne z pohľadu používateľa-rodiča, používateľa-dieťa a z pohľadu administrátora. K hlavným funkcionalitám aplikácie bude patriť poskytovanie informácií o budúcich tréningoch, rovnako ako aj o tréningoch, ktoré sa už uskutočnili, a to vo forme rôznych štatistík. Ďalej by aplikácia mala umožňovať rodičom prihlasovať a odhlasovať svoje deti na udalosti. Deti sú zaradené do rôznych kategórií a všetky udalosti pre všetky kategórie budú zobrazené v kalendári.

Rodičia budú môcť prihlásiť či odhlásiť svoje dieťa na tréning, a tým vyjadriť, či sa tréningu zúčastnia alebo nie. Túto informáciu uvidia aj ostatní členovia klubu, pretože to má množstvo výhod napríklad pre trénerov, alebo iných členov (jeden rodič môže pomôcť odviezť deti druhému, možnosť spoločného auta najmä v zime keď sa dochádza na kopce).

Ďalším cieľom tejto aplikácie je poskytnúť používateľom aktuálne informácie o zmenách, ako môže byť napríklad náhle zrušenie tréningu kvôli poveternostným podmienkam a pod.

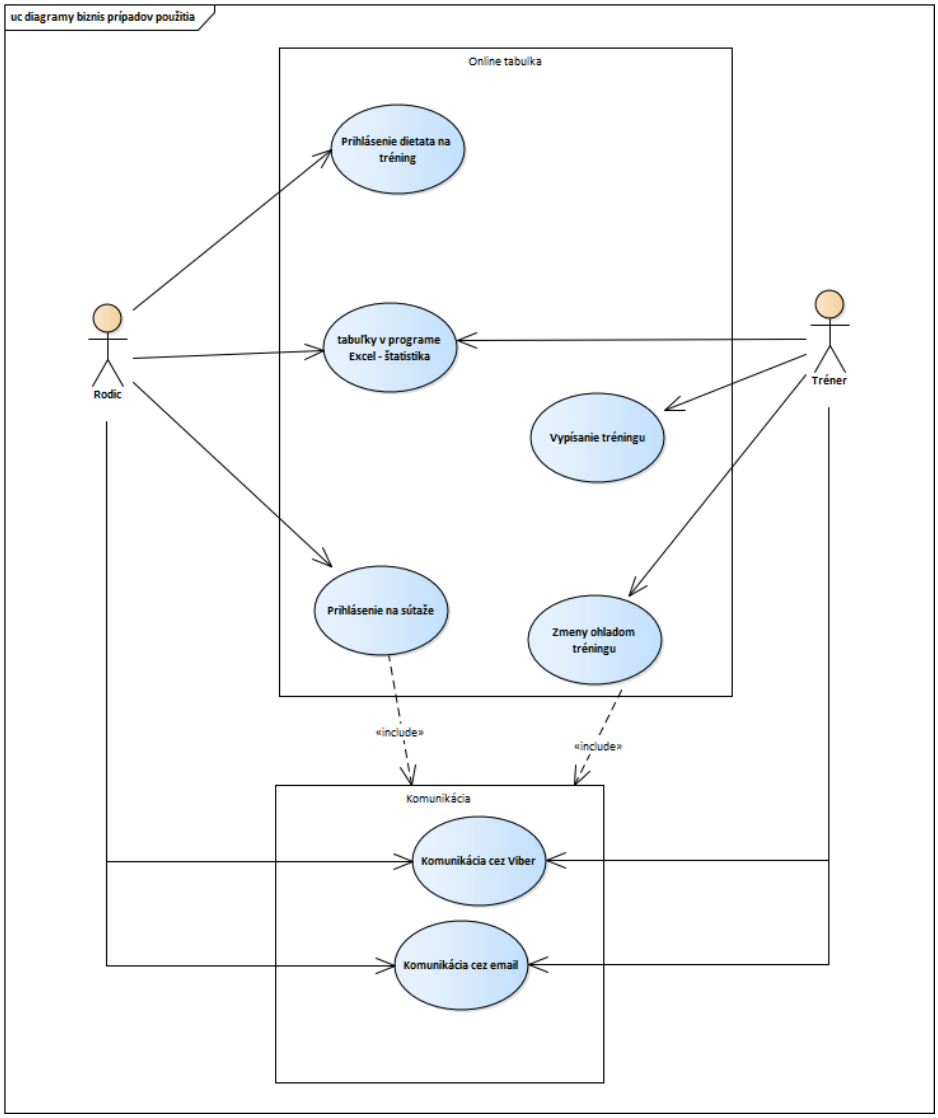
Dôležitým bodom je aj zber a poskytovanie štatistík o športovcoch a ich výkonoch, účastiach na tréningoch a všetkých potrebných dát, najmä pre zlepšenie športovej výkonnosti. Tieto dáta ďalej umožnia našim trénerom lepšie analyzovať tréningový proces a jeho výsledky.

V ďalších kapitolách práce sa budeme venovať analýze existujúcich aplikácií, analýze samotnej aplikácie z pohľadu softvérového návrhu, implementácii programu s ukážkami grafického rozhrania a popíšeme výsledok testovania aplikácie.

2 SÚČASNÝ STAV RIEŠENEJ PROBLEMATIKY

Aktuálne tento klub rieši situáciu pomocou tabuľky v Google Sheets, teda nástroja od Googlu podobnému programu Microsoft Excel. Táto tabuľka je zverejnená na internetovej adrese a každý, kto má túto adresu k dispozícií, sa k tejto tabuľke dostane a môže vypísať potrebné informácie. Toto riešenie je síce funkčné, no ani zďaleka nie je ideálne a dlhodobo udržateľné.

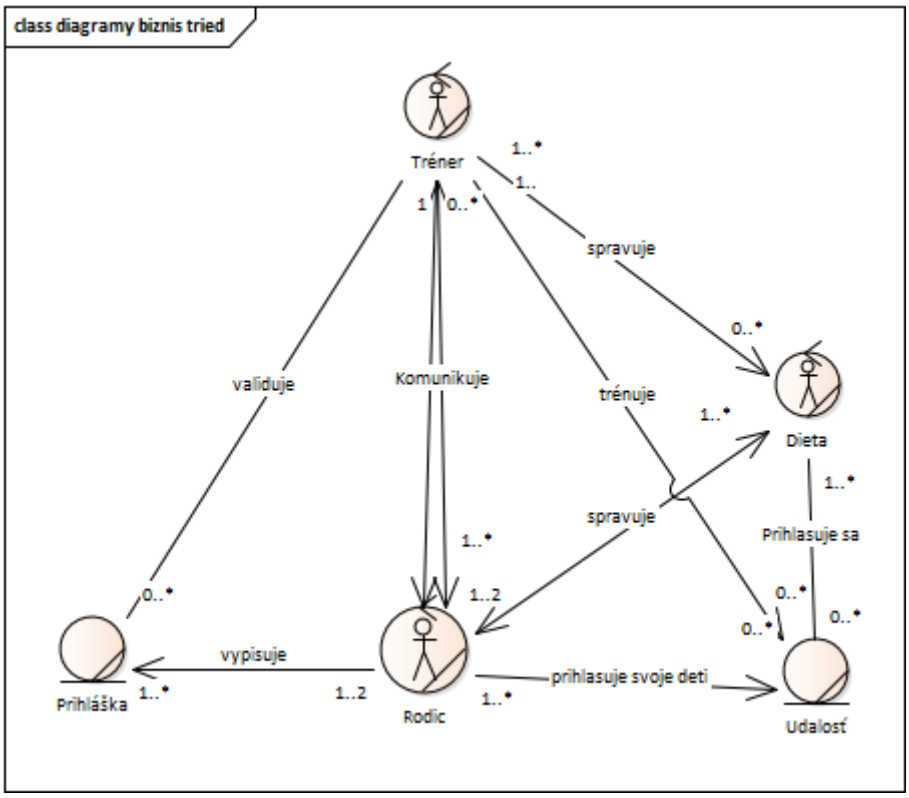
Na lepšie vysvetlenie celého fungovania tohto stavu si pomôžeme diagramom biznis prípadov použitia viď. Obrázok 2.1. Na tomto diagrame môžeme vidieť, ako prebiehajú procesy v klube pomocou online tabuľky. Ako môžeme vidieť, jeden z trénerov (v tomto klube som to konkrétne ja), má na starosti najmä vypisovanie udalostí, na ktoré potom rodičia prihlasujú svoje deti.



Obrázok 2.1 - Diagram biznis prípadov použitia

2.1 Diagram biznis tried

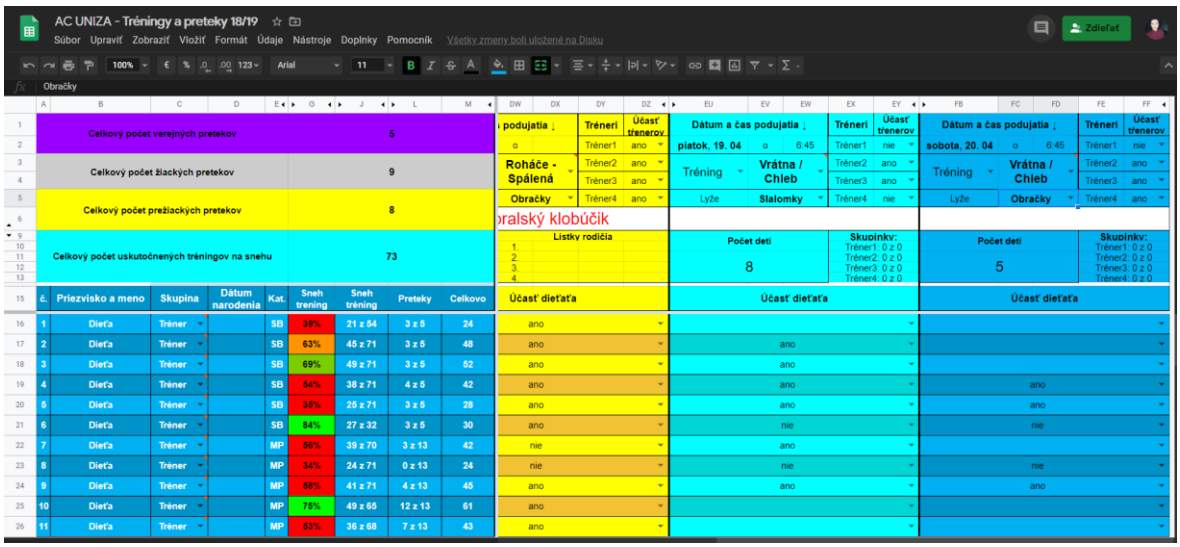
Teraz, keď už máme vysvetlený diagram biznis prípadov použitia, môžeme sa pozrieť na diagram biznis tried (Obrázok 2.1). Tento diagram nám popisuje štruktúru systému zobrazením tried a vzťahov medzi nimi. Za povšimnutie stojí napríklad fakt, že prihlášky na špecifické udalosti, napríklad súťaže, validuje tréner podľa výkonnosti.



Obrázok 2.2 - Diagram biznis tried

2.2 Ukážky aktuálneho riešenia

Na záver by sme pre lepšiu predstavu dizajnu momentálneho stavu ešte mohli uviesť jednu ukážku z počítača a jednu z mobilu. Cieľom týchto ukážok je poukázať na nie príliš prívetivé a intuitívne prostredie, ktoré je na počítači (Obrázok 2.3) ešte pomerne prehľadné, ale na mobile (Obrázok 2.4) je toto prostredie neintuitívne a mierne chaotické. Ako môžeme vidieť, na mobile sa dá zobraziť len jedna udalosť, užívateľ tak môže ľahko omylom vyplniť bunku nad, či pod žiadanou bunkou.



Obrázok 2.3 - Pohľad na súčasne riešenie na PC

Obrázok 2.4 - Pohľad na súčasne riešenie na mobile

2.3 Výhody aktuálneho stavu

Toto riešenie sme zaviedli už v minulosti, kedy klub nenavštevoval taký veľký počet detí. Tabuľka je vždy aktuálna, vzhľadom na to, že je v nej dokonca v reálnom čase vidieť, kto čo píše. Výhodou je multiplatformová podpora, aj keď nie v pravom slova zmysle. Na počítači je dostupná pomocou webového prehliadača, na mobil si však musí užívateľ stiahnuť aplikáciu.

2.4 Nevýhody aktuálneho stavu

Medzi hlavné nevýhody takéhoto riešenia patrí najmä potreba stiahnutia aplikácie pre mobily. Taktiež aj bezpečnosť, vzhľadom na to, že k tejto tabuľke má prístup každý, kto má odkaz. Rovnako sa veľa používateľov sťažuje na to, že tabuľka je neprehľadná. Častokrát sa stáva, že veľa nových členov nevie, čo majú vyplniť. Rovnako aj situácia, že užívateľ vyplní bunku, ktorú nechcel, bez toho, aby si to všimol.

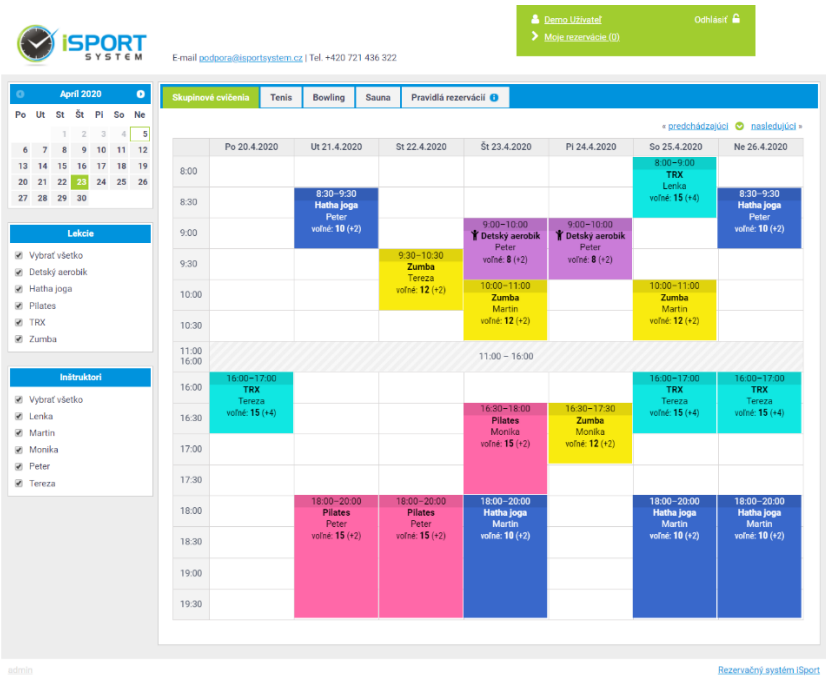
3 ANALÝZA EXISTUJÚCICH RIEŠENÍ

Vzhľadom na to, že tento klub určite nie je jediný, ktorý sa stretáva s takýmito problémami a má podobné požiadavky, budeme sa venovať najskôr analýze existujúcich riešení. V rámci tejto analýzy sme zistili, že podobných aplikácií je dostupných na trhu viacero, ale hlavná funkcionalita, ktorá im väčšinou chýba, je vzťah rodič-dieťa. To znamená, že jeden používateľ môže prihlásiť sám seba, ale pre požiadavky klubu je dôležité, aby rodič vedel prihlásiť seba a aj svoje deti na jednotlivé udalosti. Tieto aplikácie sú taktiež primárne len na online objednávanie, či zarezervovanie sa, ale nie na celkovú správu klubu.

3.1 Rezervačný systém iSport

https://www.isportsystem.sk/

Tento rezervačný systém od českej spoločnosti vyšiel z našej analýzy najlepšie. Poskytuje vysokú mieru prispôsobenia pre rôzne typy podujatí. Systém môže užívateľom zaslať e-mail alebo SMS správu, napríklad pri zmene termínu, či zrušení udalosti. Je vysoko komplexný a dá sa použiť na správu a evidenciu rezervácií pre široké množstvo športov, skupinových cvičení, ale aj rezervácií rôznych nešportových aktivít. Medzi jeho najväčšie výhody patrí to, že systém umožňuje založiť detské lekcie. Na tieto lekcie rezervujú rodičia svoje deti. Výhodou je taktiež posielanie upozornení, permanentky, viaceré úrovne používateľských práv a veľké množstvo klientov (viac ako 400). Celkovo je tento systém veľmi robustný a má množstvo funkcionalít, ktoré klub nevyužije, no napriek tomu by ich musel platiť, keďže sú súčasťou balíčka. Na druhej strane, špecifické požiadavky, ktoré by sme potrebovali, sa platia naviac, lebo by bolo nutné ich doprogramovať. To sa pochopiteľne odráža aj na vysokej cene za počiatočné spustenie a nastavenie celého systému.

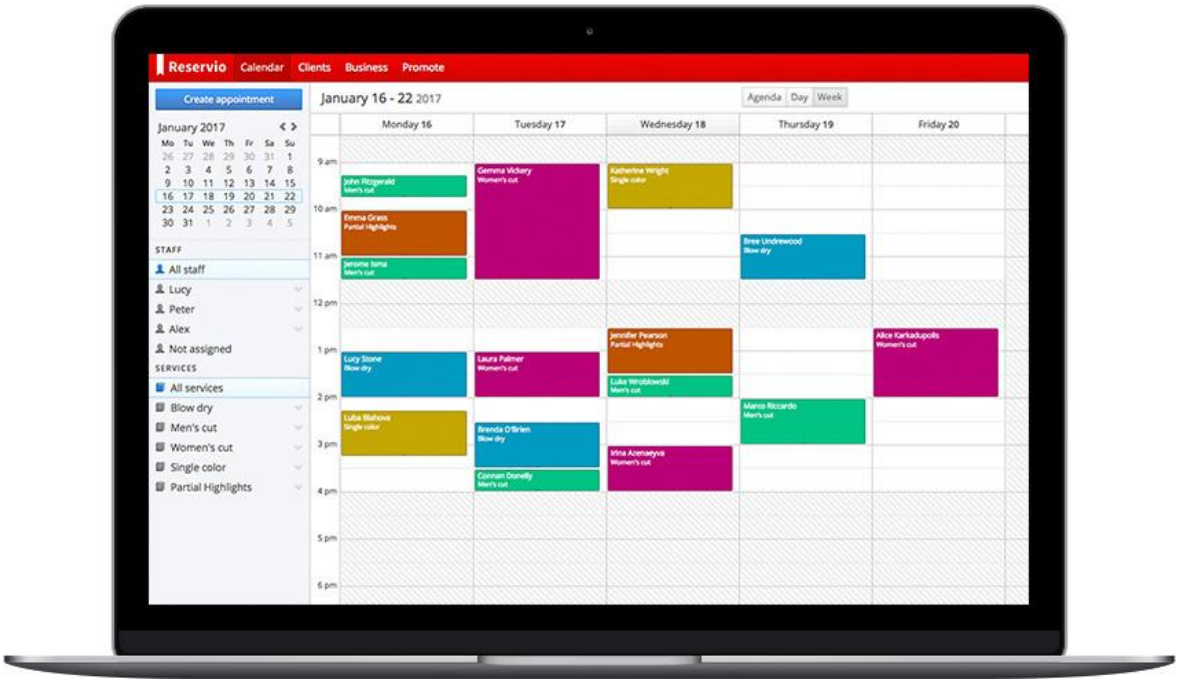


Obrázok 3.1 - Ukážka systému iSport

3.2 Reservio

https://www.reservio.com/sk/

Služba Reservio tiež ponúka online rezervácie. Aplikácia ponúka prehľadný kalendár udalostí, vernostný program, či možnosť pripomenutia rezervácií pomocou SMS, alebo emailov. Navyše ešte poskytuje integráciu s Google Kalendárom, čo sa vedeniu klubu veľmi páčilo, vzhľadom na to, že potom by si každý vedel pridať tento zdieľaný kalendár do svojho kalendára. Má viac ako 150 000 spokojných zákazníkov v 35 krajinách po celom svete. Služba ponúka aj riešenia na mieru, ktoré vedia prispôsobiť softvér pre každý typ podnikania. Nevýhodou je vysoká cena, ktorá sa platí na mesačnej báze v 3 rôznych typoch balíčkov. Registrácia detských lekcií momentálne nie je možná v univerzálnom riešení, čo by znamenalo ešte vyššie finančné náklady.

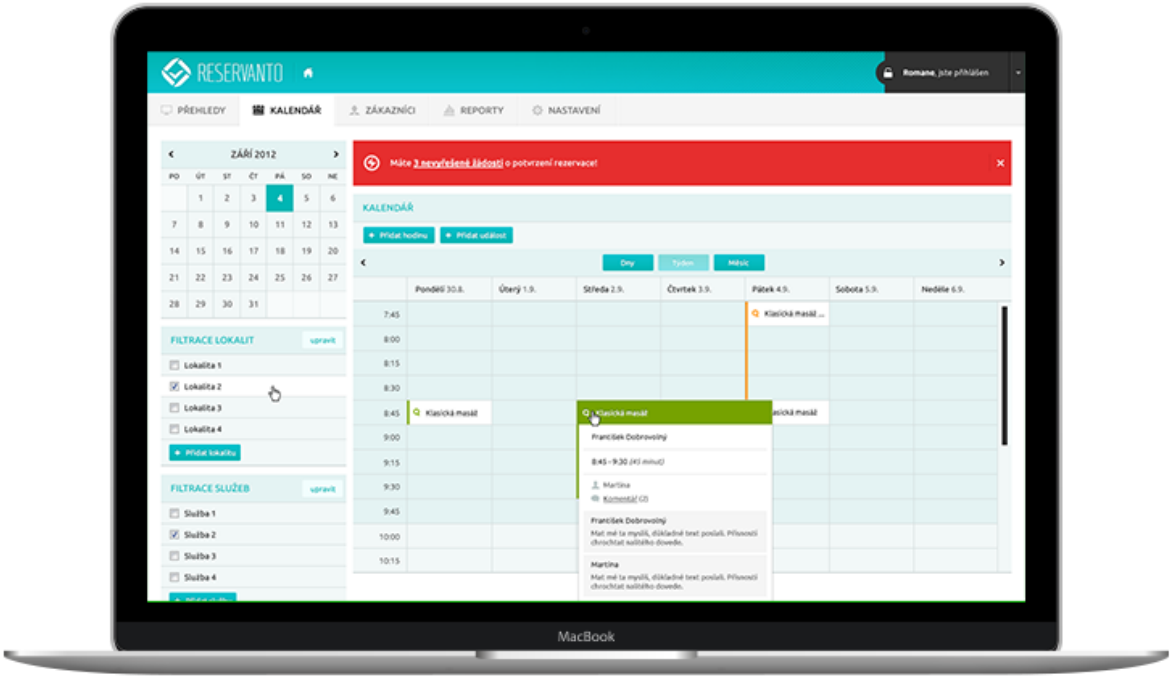


Obrázok 3.2 - Ukážka systému Reservio

3.3 Reservanto

https://www.reservanto.sk/

Ďalšou aplikáciou na trhu je rezervačný systém Reservanto. Tento prináša jednoduché a prehľadné ovládanie. Výhodou sú opäť automaticky odosielané pripomenutia pomocou SMS zákazníkom, ktorí vytvorili rezerváciu. Príjemnou funkcionalitou sú online platby, ktoré sú ale bohužiaľ v našom klube zatiaľ nepoužiteľné. Tak, ako predošlé aplikácie, aj Reservanto je viac zameraný len na rezervovanie nejakého časového bloku jedným používateľom, bez možnosti detských rezervácií. Tento systém obsahuje 4 základné balíčky, z ktorého je jeden dokonca zadarmo, neobsahuje však potrebnú funkcionalitu. Ďalšie balíčky sú obohatené o extra funkcionalitu. Avšak SMS upomienky sú spoplatnené samostatne, a taktiež už spomínané online platby sú účtované percentami z prevedených platieb.

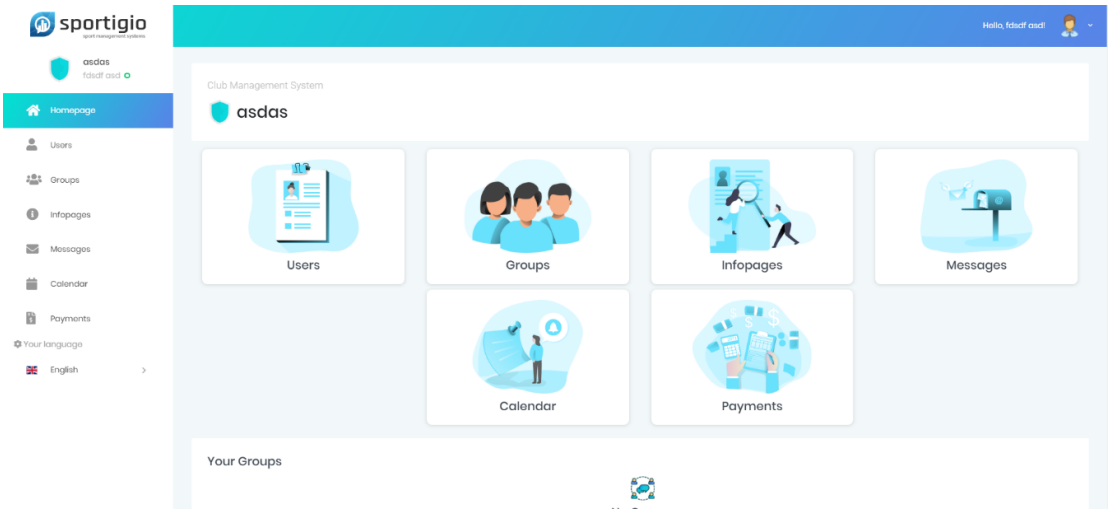


Obrázok 3.3 - Ukážka systému Reservanto

3.4 Sportigio

https://sportigio.com/management-system/

Tento zahraničný systém má pekný dizajn a veľmi intuitívne ovládanie. Systém umožňuje vytváranie rôznych skupín, čo by sa pri väčšej námahe teoreticky dalo použiť na naše kategórie. Napriek tomu z celkového hľadiska je jeho funkcionalita značne obmedzená, nakoľko neponúka žiaden typ upozornenia, či už SMS alebo emailom. Taktiež sme tu nenašli žiadne detailné štatistiky z minulosti, ani možnosť registrácie dieťaťa. Je tu tiež veľmi malá možnosť rozšíriteľnosti o funkcionality do budúcna, nakoľko autori nemali záujem o doprogramovanie nejakého vlastného riešenia.



Obrázok 3.4 - Ukážka systému Sportigio

3.5 Zhodnotenie

Vo všeobecnosti sa väčšina aplikácií sústreďuje skôr na online objednávanie, rezervovanie časového bloku, či služieb a udalosti. V našom prípade ale potrebujeme trochu inú funkcionalitu. Veľkým špecifikom je napríklad to, že rodič (ako jeden užívateľ) vie prihlásiť svoje dieťa (ako druhého užívateľa). Takúto funkcionalitu nám vedel poskytnúť v univerzálnom riešení jedine rezervačný systém iSport. Vlastné riešenie umožní vytvoriť systém na mieru, ktorý uľahčí všetkým stranám ich vzájomnú interakciu. Systém na mieru ušetrí klubu čas a peniaze, ale pomôže aj trénerom zlepšiť prípravu a výsledky jednotlivých športovcov. Rovnako tak uľahčí rodičom nahlasovanie a komunikáciu s klubom či trénermi, tiež budú mať vždy aktuálne informácie o situácii v klube. Ďalším, nemenej dôležitým faktorom pre vytvorenie vlastnej aplikácie, je požiadavka na rozvoj aplikácie do budúcnosti a rozšírenie o ďalšie funkcionality.

4 ANALÝZA APLIKÁCIE

V tejto časti práce sa budeme venovať analýze danej problematiky, prečo a za akým účelom aplikáciu vyvíjame a ktoré všetky vlastnosti má daná aplikácie spĺňať. Vzhľadom na to, že vytvárame aplikáciu pre konkrétny lyžiarsky klub, najskôr sa musíme pozrieť ako tento klub funguje a aké sú jeho procesy. Na ich základe a rozprávania sa s vedením klubu a rodičov, ktorí momentálne v tomto klube pôsobia, môžeme následne identifikovať požiadavky na systém.

4.1 Popis procesov v klube

Ako som už spomínal, aplikácia je vyvíjaná pre stredne veľký lyžiarsky klub zo Žiliny, zameraný na alpské závodné lyžovanie. Klub je oficiálnym klubom registrovaným v Slovenskej lyžiarskej asociácií, ktorý sa zúčastňuje na regionálnych, štátnych, aj medzištátnych súťažiach.

Vzhľadom na zvyšujúcu sa popularitu zjazdového lyžovania za poslednú dobu, klub prijal pomerne veľa nových členov. Problémom ale začalo byť, že nie vždy prídu všetky deti na tréning, a tak sa medzi počtom detí na jednotlivých udalostiach objavujú veľké odchýlky. Tento fakt je veľmi nežiadúci, keďže klub musí nahlásiť stredisku, aké chce vleky (záleží od kategórie), a koľko športovcov bude trénovať. Ďalším dôvodom, prečo je potrebné vedieť počet a detaily športovcov, je prihlasovanie na preteky, ktoré je podmienkou pre zúčastnenie sa a realizuje sa vždy elektronicky dopredu. Nemenej dôležitým dôvodom je aj potreba dostatočného množstva trénerov, ktoré musí klub zabezpečiť na danú udalosť.

Okrem tohto naši tréneri môžu tieto dáta využiť na lepšiu analýzu športovca. Môžu sa pozrieť napríklad na to, koľko tréningov strávil športovec na jednotlivých typoch udalostí v ľubovoľných časových úsekoch.

Toto celé manažovanie momentálne nie je vôbec jednoduché. Dáta mimo aktuálnej sezóny sú uložené vo viacerých súboroch a nie sú prístupné online. Prístup na prihlásenie cez Google Tabuľky cez mobil vyžaduje aplikáciu, ktorá nie je dostupná na všetky mobilné operačné systémy. Tabuľka s pribúdajúcim množnom dát a exponenciálne narastajúcou komplexnosťou začína byť pomalá.

4.2 Mimofunkčné požiadavky

Jednou z najvýznamnejších mimofunkčných požiadaviek zo strany klubu je intuitívne používateľské rozhranie, ktoré zabezpečí príjemný pocit z používania aplikácie ako súčasným, tak aj novým požívateľom. Samozrejmosťou v dnešnej dobe je responzívny dizajn, ktorý vie prispôsobiť zobrazenie veľkosti obrazovky. Naša aplikácia by taktiež mala byť nasadená na serveri a prístupná na verejne dostupnej adrese, na ktorú sa vie každý používateľ jednoducho dostať. Vzhľadom na to, že aplikácia bude obsahovať citlivé údaje o jednotlivých používateľoch, tieto údaje by mali byť prístupné len používateľovi so správnymi autorizačnými právami.

4.3 Multiplatformovosť

Ako sme už vyššie podotkli, je potrebné navrhnúť systém, s ktorým žiadny z rodičov nebude mať problém, a bude sa k nemu môcť dostať odkiaľkoľvek z hocijakého zariadenia. Musíme preto brať ohľad na to, že niektoré aktivity sú vykonávané primárne na mobile. Takéto činnosti môžu byť napríklad prihlásenie, či odhlásenie z tréningu. Naopak existujú aktivity primárne vykonávané na počítači, ako môže byť napríklad analýza dát trénerom. Potrebovali by sme niečo platformovo nezávislé s možnosť používania aplikácie odkiaľkoľvek, kedykoľvek a na ľubovoľnej platforme.

4.4 Vzťah rodič - dieťa

Potrebnou a nutnou súčasťou tejto aplikácie bolo to, aby každý jej užívateľ, či už rodič, dieťa, alebo v budúcnosti napríklad tréner, mal samostatné konto a vedel sa ním do aplikácie prihlásiť. Vzhľadom na to, že v klube sú aj neplnoleté osoby, v aplikácií by mali byť používateľské práva vytvorené tak, aby mal rodič plnú správu nad svojím dieťaťom, ktorému po prihlásení môže vytvoriť konto.

4.5 Udalosti

Systém by mal obsahovať aktuálne vypísané tréningy (trénermi), na ktoré si rodičia budú vedieť prihlásiť svoje deti do určitého času pred danou udalosťou. K týmto údajom bude mať prístup vedenie klubu. Rodič by mal vedieť prihlásiť či odhlásiť dieťa na udalosť

(tréning, preteky, sústredenie..). Dieťa by sa malo vedieť samé prihlásiť do aplikácie a vedieť sa prihlásiť na tréning.

4.6 Upozornenia

Ďalšou funkcionalitou by mali byť upozornenia. ktoré by systém, napríklad v prípade náhleho zrušenia tréningu či inej neočakávanej udalosti, automaticky poslal. Častokrát sa stáva, že napríklad v prípade zlého počasia sa tréning zruší, či presunie na iné miesto pár minút pred tréningom. Nie všetci rodičia si však takúto vec pozerajú v tabuľke tesne pred tréningom.

4.7 Štatistiky

Aplikácia by mala všetky údaje zbierať a ukladať si ich do databázy, odkiaľ k nim, okrem samotných rodičov a detí, bude mať prístup aj tréner. Je to z toho dôvodu, aby tréner mal potrebné dáta. Tieto dáta budú slúžiť na to, aby vedel napríklad, akú má športovec pohybovú zdatnosť, alebo či má dosť dní na snehu na to, aby išiel na preteky. Tieto dáta následne môžu pomôcť trénerom v dosahovaní lepších športových výkonov u svojich zverencov.

4.7.1 Sezóna a kategórie

Dôležitým faktorom a údajom pre tieto štatistiky sú časové údaje a kategórie, pre ktoré boli tieto údaje zaznamenávané. V rôznych sezónach môžu byť deti zaradzované rôzne dlhú dobu. Vysvetlením je, že tieto kategórie sú závislé na súťažnom poriadku SLA. To sa môže stať napríklad tak, že v sezóne 2017-2018 bola kategória mladší predžiaci pre ročníky 2000 a 2001. SLA ale po tejto sezóne zmenila pravidlá a tak došlo k tomu, že v nasledujúcej sezóne 2018-2019 sa táto kategória rozšírila o ešte jeden ročník. Takáto situácia teda znamená, že táto sezóna (2018-2019) zahŕňa len ročníky narodenia 2002 a 2003, ako by tomu bolo bez zmeny pravidiel, ale aj ročník 2001.

4.8 Dodatočné funkcionality

Z rozhovoru vyplynulo, že klub by mal záujem aj o doplňujúce funkcionality v budúcnosti. V práci preto uvádzame aj dodatočné funkcionality, ktoré síce aplikácia nebude momentálne poskytovať, ale je dobré na ne myslieť v prípade ďalšieho vývoja, aby

nebolo nutné niektoré veci prerábať. Klub mal záujem o evidenciu nielen detí, ale aj trénerov na danom tréningu pre následné výplaty, taktiež o evidenciu platieb členského poplatku, alebo generovanie rôznych dokumentov, ako napríklad generovanie prihlášky do klubu, či do lyžiarskeho zväzu.

4.9 Aktori a role v systéme

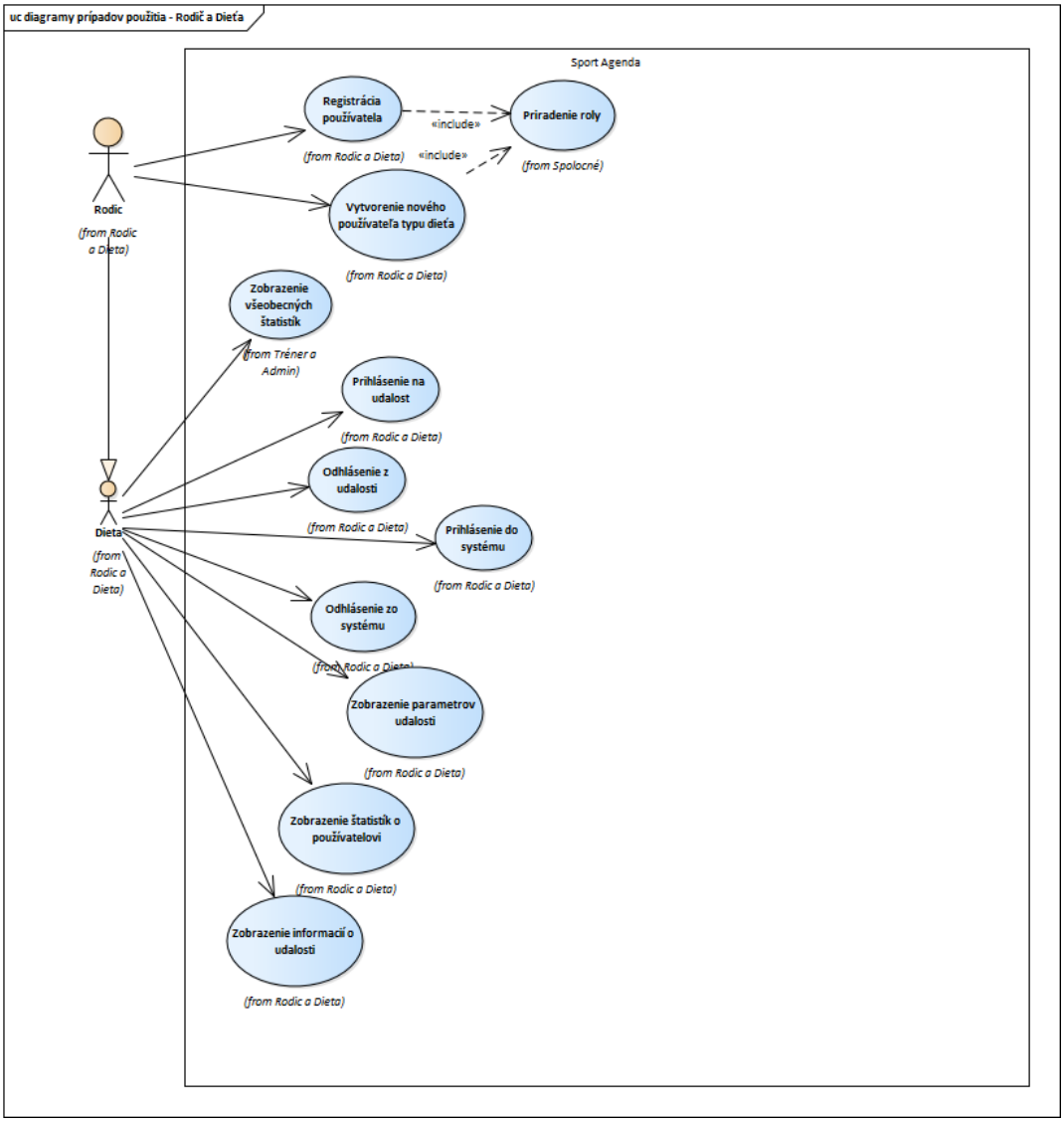
V systéme budú 4 typy aktorov: bežný používateľ, dieťa, tréner a administrátor. Bežný používateľ – **rodič** má možnosť vytvoriť ďalšieho používateľa. Takýto používateľ sa stáva automaticky dieťaťom patriacim do rodiny spolu s rodičom. **Dieťa** má obmedzené práva len na prihlásenie sa na tréning a na zobrazenie svojich štatistík. **Tréner**, ktorého rola je založená na rodičovi, má právo aj na editovanie a zmazanie udalosti. Ako si uvedieme v ďalšom odstavci, najväčšie možnosti má **administrátor**. Týchto aktorov si postupne predstavíme s pomocou diagramu prípadov použitia.

4.10 Pohľad dieťaťa

Takýto typ používateľa vie vytvoriť rodič až po prihlásení sa do aplikácie. Týmto návrhom vieme priradiť korektné vzťahy v rodine. Zároveň nám takýto prístup umožní vytvoriť používateľa podľa používateľského mena, cez ktoré sa môže dieťa prihlásiť, a teda aj bez toho, aby malo email.

4.11 Pohľad rodiča

Rodičom sa stávajú všetci používatelia, zaregistrovaní pomocou registračného formulára, ešte pred prihlásením sa do aplikácie. Rodič sa registruje pomocou emailu, aby sa zabránilo zbytočným spamom, či nevyžiadaným osobám v aplikácií. Toto riešenie nám zabezpečí aj to, že keď budeme chcieť posielať emaily, máme istotu, že tento email bude správny. Na nasledujúcom diagrame si uvedieme prípady použitia z pohľadu tohto typu aktora. Ako môžeme vidieť na Obrázok 4.1, rodič má všetky funkcie dieťaťa, navyše je mu aj umožnené vytvorenie nového špecifického užívateľa v systéme (dieťaťa) a zobrazenie detailných štatistík o tomto užívateľovi.



Obrázok 4.1 - Diagram prípadov použitia - rodič a dieťa

4.12 Pohľad trénera

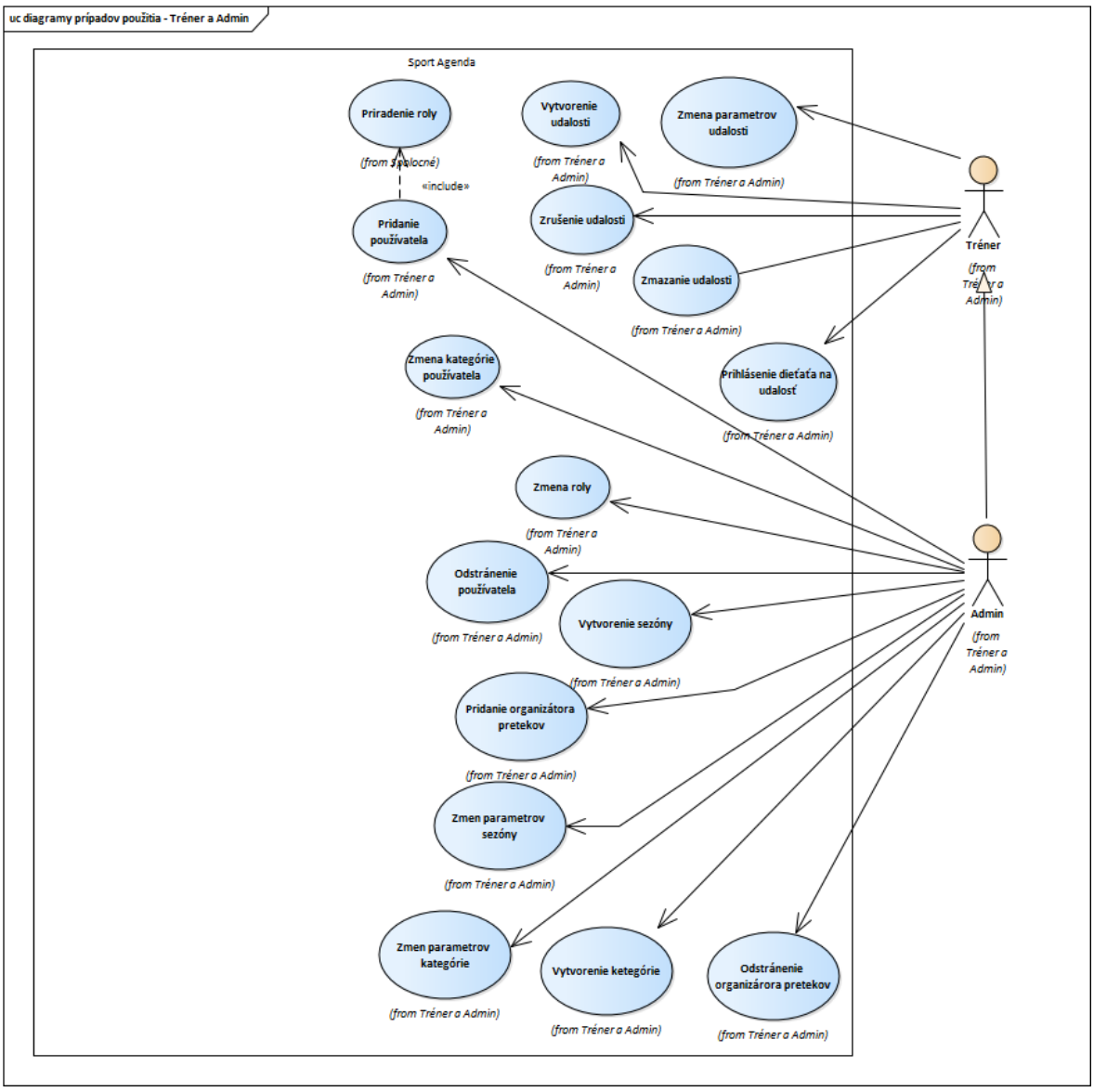
Pohľad trénera je určitým prienikom funkcií a práv rodiča a administrátora. V našej aplikácií nechceme, aby tréner mohol napríklad zmeniť email inému používateľovi. Na druhej strane, určite bude potrebné priradiť mu práva navyše oproti rodičovi, napríklad na editovanie udalostí. Pre prehľadnosť budeme vychádzať z toho, že má všetky funkcie rodiča. Tie, ktoré má naviac sú:

- Pridať/Editovať/Zmazať udalosť
- Zrušiť udalosť
- Pridať/Odstrániť ktorékoľvek dieťa z udalosti

4.13 Pohľad admina

Admin pristupuje do systému špeciálnym spôsobom. Nepristupuje bežným prihlásením ako všetci ostatní, ale extra adresou a extra prihlasovacími údajmi. Tento prístup sme zvolili kvôli tomu, že framework, ktorý používame pre serverovú časť, má toto administrátorské rozhranie čiastočne implementované. Takéhoto používateľa môže vytvoriť len iný admin, alebo môže priradiť administratívne práva už existujúcemu užívateľovi. Toto administratívne rozhranie sa používa väčšinou pri menších až stredne veľkých projektoch, takže je v našom stave projektu dostačujúce riešenie. Ako môžeme vidieť na Obrázok 4.2 z tohto rozhrania vie upraviť úplne všetko, od pridania tréningu po zmenu hesla a užívateľského mena, či vzťahov v rodine. Okrem iného má funkcie:

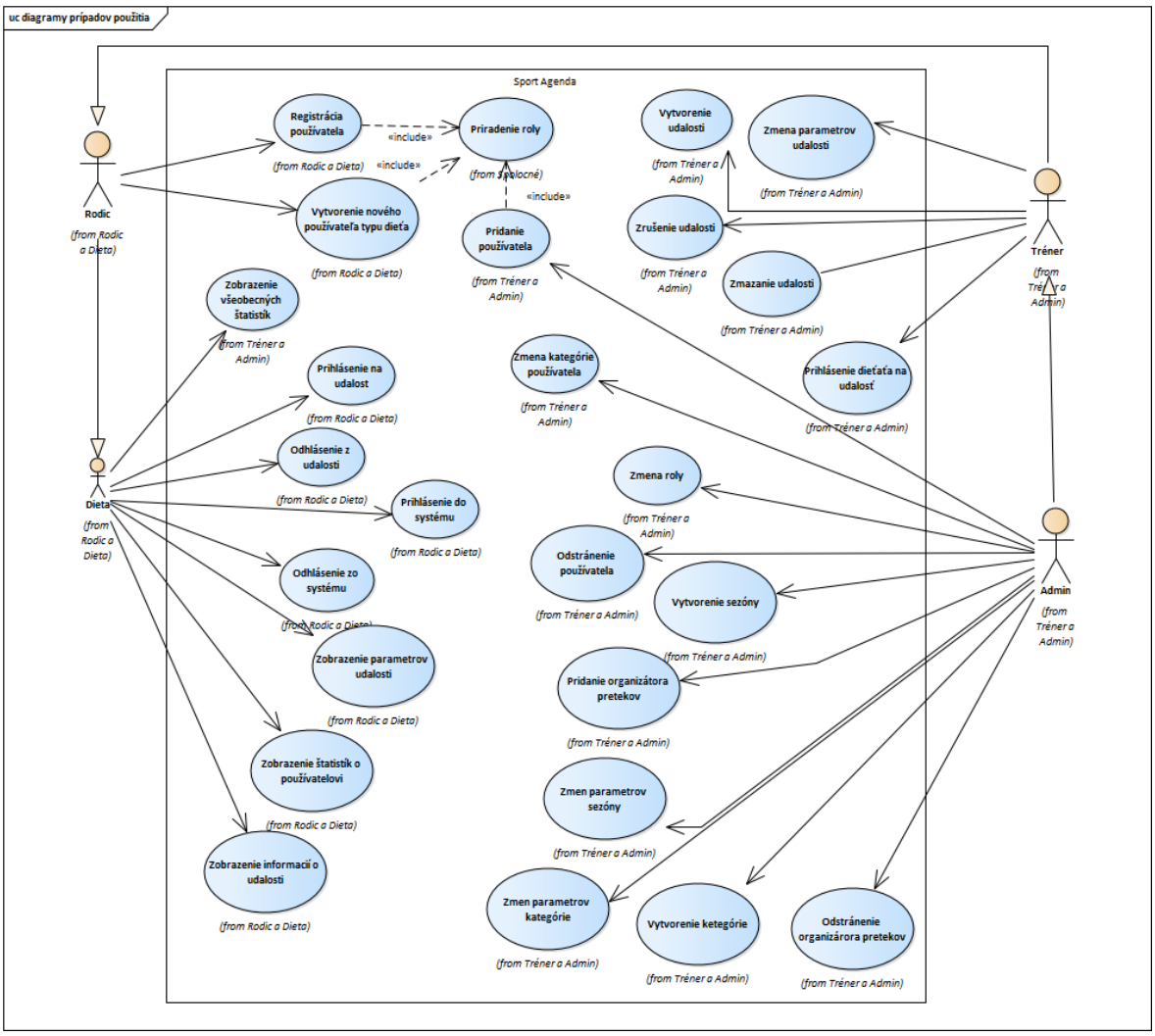
- Vytvoriť/Editovať/Zmazať sezónu
- Vytvoriť/Editovať/Zmazať kategórie
- Vytvoriť/Editovať/Zmazať organizátora pretekov
- Vytvoriť/Editovať/Zmazať udalosť
- Pridať/Editovať/Odstrániť kategóriu z udalosti
- Pridať/Editovať/Odstrániť dieťa z udalosti
- Zrušiť udalosť



Obrázok 4.2 - Diagram prípadov použitia – tréner a admin

4.14 Celkový diagram prípadov použitia

Hlavným cieľom pri návrhu tohto systému je samostatnosť rodiča a dieťaťa, čo sa týka hlavne prihlasovania na udalosti. Pre lepšie pochopenie tu uvedieme príklad z praxe. V prípade, že dieťa ešte nie je schopné sa prihlásiť (pre nízky vek), prihlasovanie za neho rieši jeho rodič. U starších detí toto prihlasovanie na udalosti väčšinou prechádza už na dieťa samotné. Rodič má samozrejme stále kontrolu nad všetkým, čo dieťa v systéme robí. Nemusí však, pre správne fungovanie logiky a potrieb aplikácie, nijako zasahovať, potvrdzovať, či dokonca používať aplikáciu, ak nemá záujem. Takýmto návrhom môžeme aspoň čiastočne zvýšiť správnosť údajov, a tak zabezpečiť jednoduchší rozvoj funkcionality do budúcna.



Obrázok 4.3 - Diagram prípadov použitia pre všetkých aktorov

5 NÁVRH APLIKÁCIE

V tejto kapitole sa budeme venovať celkovému návrhu aplikácie. Tento návrh bude vychádzať najmä z predošlej kapitoly, ale taktiež aj z osobných skúseností autora a z rozhovorov s jednotlivými členmi klubu. Aplikáciu bude vytváraná v anglickom jazyku, nakoľko je programovanie v tomto jazyku v dnešnej dobe prirodzenejšie a obnáša to viacero výhod. Napríklad, nie všetci členovia klubu sú zo Slovenska, resp. rozumejú po slovensky. Z programátorského hľadiska tiež pri Open Source projekte vie tento projekt použiť či pochopiť prakticky celý svet. Samozrejme, vzhľadom na to, že zameranie aplikácie je primárne pre Slovensko, aplikácia bude obsahovať prekladový súbor v angličtine, ktorý sa dá jednoducho preložiť do ktoréhokoľvek jazyka.

5.1 Architektúra aplikácie

Od našej aplikácie sa vyžaduje, aby používateľovi poskytovala relevantné dáta v prehľadnej forme. Tieto dáta získa užívateľská aplikácia z centralizovaného **servera**, ktorý bude prístupný pomocou viacerých inštancií webového prehliadača - **klientov**. Webová aplikácia je typu klient-server vytvorená pre prostredie internetu. To, akým spôsobom bude klient žiadať o dáta a akým spôsobom mu ich server poskytne, rozoberieme v ďalších kapitolách.

5.1.1 Klient-server architektúra

Klient-server je architektúra počítačových sietí, v ktorej klienti (procesy aplikácie) žiadajú a získavajú servis od centralizovaného servera (hostiteľský počítač). Klientske počítače poskytujú rozhranie, ktoré umožňuje užívateľovi počítača požadovať služby servera a zobraziť výsledky, ktoré server vráti. Server čaká na žiadosť od klienta a odpovedá na ňu. V ideálnom prípade server poskytuje známe rozhranie, aby klienti nemuseli poznať hardvérové a softvérové špecifiká serveru, a teda klient aj server sa stávajú nezávislí na platforme. Klienti sa často nachádzajú na počítačoch či mobiloch, zatiaľ čo servery sú zvyčajne umiestnené na výkonnejších počítačoch inde na sieti. Nevylučuje sa ale, že server a klient sú spustené ako dva procesy na tom istom zariadení. Tento výpočtový model je zvlášť efektívny vtedy, keď klienti aj server majú odlišné úlohy, ktoré bežne vykonávajú. Klientske aplikácie zvyčajne nevykonávajú pamäťovo alebo časovo náročné úlohy, ale poskytujú užívateľovi len rozhranie pre zadávanie alebo zobrazovanie dát, kým na serveri beží iná aplikácia riadiaca prístup k databáze, kde sú dáta trvalo uložené. Dôležitou

charakteristikou tejto architektúry je to, že na server môžu pristupovať **naraz viacerí klienti,** zároveň však klient môže vykonávať aj iné operácie. [1]

5.2 Komunikácia so vzdialenými službami

Komunikácia prostredníctvom API po mnoho rokov spočívala na dvoch hlavných pilieroch - SOAP a REST. SOAP bol vytvorený v roku 1998 a REST v roku 2000, pričom mnoho rokov dominovali všetkému súvisiacemu s rozhraniami programovania aplikácií. V roku 2015 však Facebook vydal GraphQL, ktorý vyhlásil za alternatívu k REST a SOAP. V nasledujúcej časti sa teda pozrieme na jednotlivé technológie a vyberieme takú, ktorá pre našu aplikáciu vyhovuje najviac.

5.2.1 SOAP

Pri webových službách sa požadované dáta získavajú volaním vzdialených metód. Vyžíva sa protokol SOAP na výmenu správ pomocou XML prostredníctvom protokolu HTTP. Keďže SOAP je sám o sebe protokol, vyžaduje silnú väzbu klienta a servera, a preto je zložitejší a často zbytočne komplikovaný pre náš typ aplikácie.

5.2.2 RESTfull API

RESTful API na rozdiel od SOAP nie je protokol, ale architektonický štýl pre komunikáciu klienta a servera. Je postavený na princípe zdrojov (resources), na rozdiel od SOAP, kde hovoríme o akciách (metódach). Každý zdroj, ktorý môže byť čokoľvek od používateľov, autorov alebo článkov, 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. [2]

Reprezentácie zdrojov sa posielajú medzi klientom a serverom vo formáte XML alebo JSON (môžu však byť aj iné formáty ako CSV, HTML alebo čistý text). Na komunikáciu klienta a servera využíva REST bezstavový protokol, ktorým je zvyčajne HTTP. Používajú sa HTTP metódy (GET, POST, PUT, DELETE) a spolu s reprezentáciou zdrojov v URI vytvoria zmysluplnú požiadavku, na ktorú server odpovie HTTP odpoveďou. Fakt, že REST je bezstavový, znamená to, že na strane servera sa neukladá žiadny stav

klienta, každá žiadosť dostatočne opisuje samú seba a stav sa ukladá (ak je treba) na strane klienta. [3]

5.2.3 GraphQL

Ako už bolo spomenuté, tento model bol vyvinutý spoločnosťou Facebook a predstavený v roku 2015. Je vyhľadávacím jazykom s otvoreným zdrojovým kódom. Jeho cieľom je poskytnúť alternatívu k tradičnej architektúre REST API. GraphQL vo svojom jadre poskytuje syntax, ktorá popisuje, ako požiadať o údaje (nazývané schéma). GraphQL je databázová agnostika a pracuje tak, že vytvára jediný koncový bod zodpovedný za prijímanie dotazov. Nespolieha sa na prístup REST API, ktorý má pre každú službu samostatné koncové body. [4]

5.2.4 Výber technológie

Výber správnej technológie teda nie je jednoduchý. V prípade, že porovnávame tieto technológie, nachádzame ich výhody aj nevýhody. V našej aplikácii využijeme RESTful API na komunikáciu klienta so serverom, keďže je omnoho jednoduchšia než iné webové služby, ako napríklad SOAP, a zároveň je dlhodobo používaná oproti GraphQL. Je ideálnou voľbou pre aplikácie založené na JavaScripte. Okrem toho dokáže zvládnuť vyššie zaťaženie a spracovávať dáta rýchlejšie ako SOAP. Nevýhodami GraphQL je napríklad nepodporovanie nahrávania súborov, či zbytočne veľká komplexnosť pre našu aplikáciu. [2]

5.3 Návrh serverovej časti aplikácie

Väčšiu časť logiky aplikácie bude mať na starosti serverová aplikácia, ktorá je pre fungovanie klientskej aplikácie nevyhnutná, pretože jej poskytuje potrebné dáta. Hlavnými charakteristikami serverovej aplikácie budú:

- čaká na požiadavky od klienta
- prijíma požiadavky, spracúva ich a následne odpovedá
- zvyčajne akceptuje pripojenia od väčšieho množstva klientov
- typicky koncoví používatelia neprichádzajú s ňou priamo do styku

5.3.1 MVC (MTV)

Model-View-Controller je jeden z najpoužívanejších architektonických vzorov pri vývoji webových aplikácií. MVC je veľmi obľúbený architektonický vzor, ktorý sa uchytil najmä na webe, hoci pôvodne vznikol na desktopoch. Je súčasťou populárnych webových frameworkov.

- **Model** reprezentuje dátovú vrstvu zodpovednú za riadenie logiky a prístup k dátam.
- **View** je vrstva užívateľského rozhrania zobrazujúca dáta z vrstvy Model.
- **Controller** je logická vrstva, ktorej je oznamované užívateľovo správanie a na základe tohto správania obnovuje Model.

Kontroler (*Controller*) reaguje na udalosť, ktorú vyvolá užívateľská interakcia. Zavolá model, ktorý dáta vyhľadá v databáze a vráti jeho údaje. Tieto údaje si controller ukladá do premenných. Nakoniec vyrenderuje view (pohľad). View sú odovzdané premenné s príslušnými dátami. View prijme dáta od Controlleru a zobrazí ich.

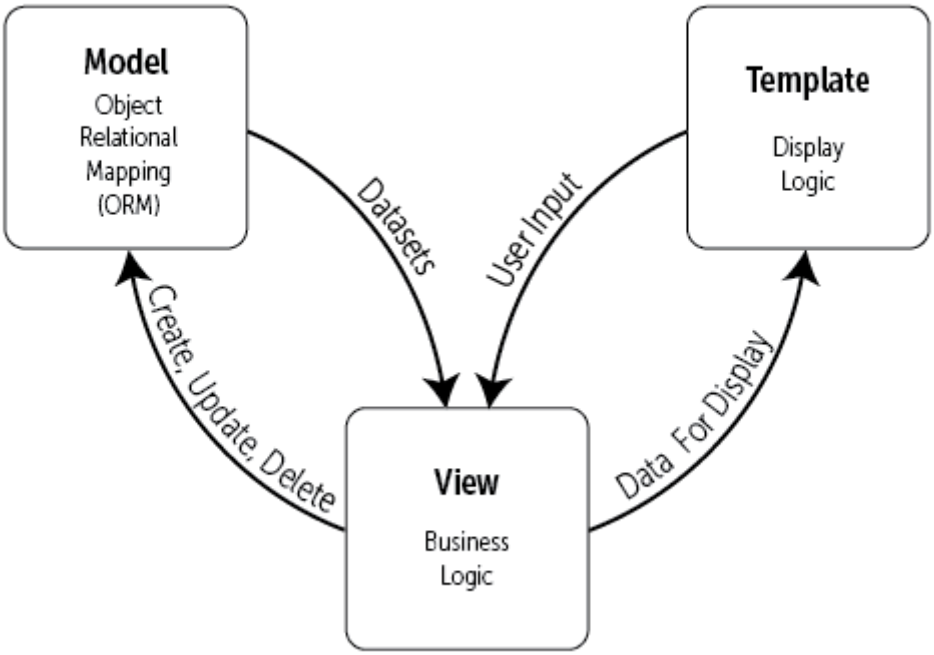
Framework Django implementuje MVC architektúru presne tak, ako sme si ju popísali. Jednotlivé komponenty ale nazýva po svojom. Bohužiaľ názov jedného komponentu prehodil, čo môže byť mätúce.

Model - Modelom hovorí Django ***Model,*** v preklade model.

Views - pohľadom hovorí Django ***Template***, v preklade šablóna.

Controller - kontrolérom hovorí Django ***View***.

V takomto prípade, keď v našej Django aplikácii tvoríme nový view, nie je to šablóna (template), ale prostredník medzi modelom a šablónou, inak povedané Conroller. [5] Odpoveď je vo forme HTTP odpovede. Súčasťou odpovede je aj stavový kód, ktorý indikuje správnosť spracovania požiadaviek. Kvôli tomuto býva tento model v Djangu nazývaný aj MTV (Model-Template-View). Pre priblíženie celého modelu a jeho fungovanie uvádzam Obrázok 5.1.



Obrázok 5.1 - MTV model Django [6]

5.3.2 Zabezpečenie zdrojov

Naša aplikácia obsahuje citlivé dáta a klub si neželá, aby ktokoľvek mimo tohto systému mal k nim prístup. Preto musia byť API endpointy zabezpečené a prístup budú mať len autorizované žiadosti. To znamená, že len klientovi so správnymi autorizačnými údajmi bude poskytnutá očakávaná odpoveď na žiadosť. Klient, ktorý nemá správne autorizačné údaje alebo práva, nedostane prístup ku zdrojom, čo sa prejaví v stavovom kóde HTTP odpovede.

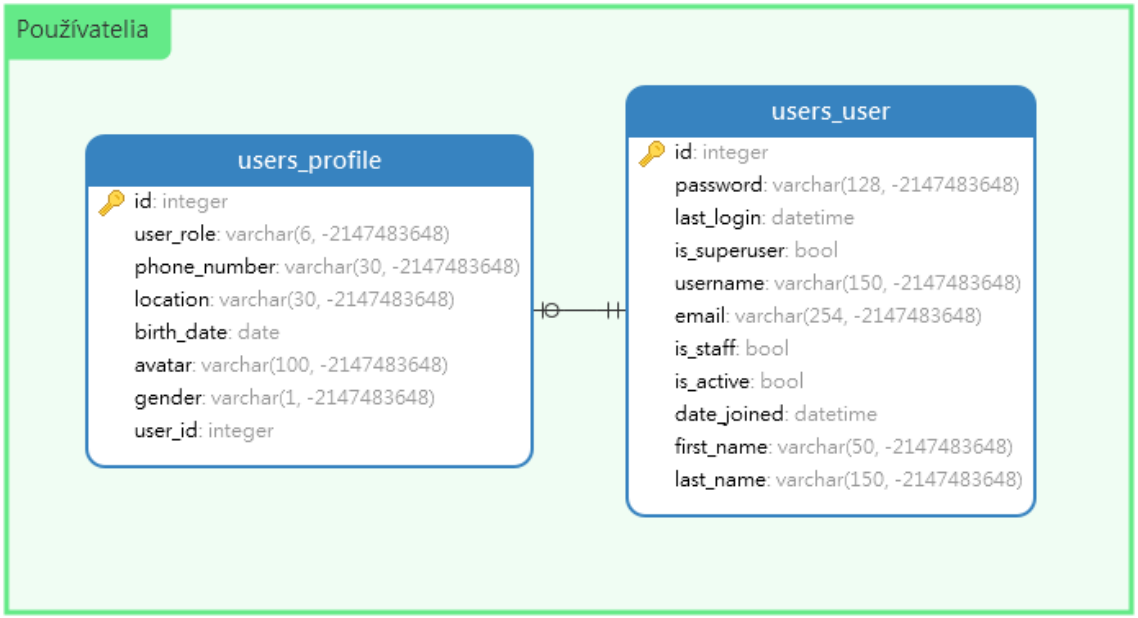
5.4 Návrh databázy

Pre našu aplikáciu sme zvolili databázový systém SQLite, ktorý je súčasťou Djanga. Django podporuje integráciu aj s inými databázovými systémami, ako napríklad PostgreSQL, momentálne to však nie je potrebné. Pri návrhu databázy sme vychádzali z funkcionalít aplikácie, ktorými sú správa používateľov, poskytovanie informácií o udalostiach, posielanie upozornení a vedenie štatistík. Databázu sme si rozdelili na 4 hlavné moduly, pre ktoré ale nebudeme opisovať všetky tabuľky databázy podrobne. Priblížime si hlavne na logiku týchto modulov a ozrejmime si časti, ktoré sú pre aplikáciu špecifické.

5.4.1 Používatelia

V tejto časti sa nachádzajú dve tabuľky, ako je možné vidieť z Obrázok 5.2. Sú to konkrétne tabuľky s názvom ***User*** a ***Profile***. Tabuľka ***User*** je zodpovedná len pre prihlasovacie účely. Všetky ostatné informácie, ktoré nie sú potrebné pre prihlasovanie, ale sú súčasťou informácií o používateľovi, sú uložené v tabuľke ***Profile***.

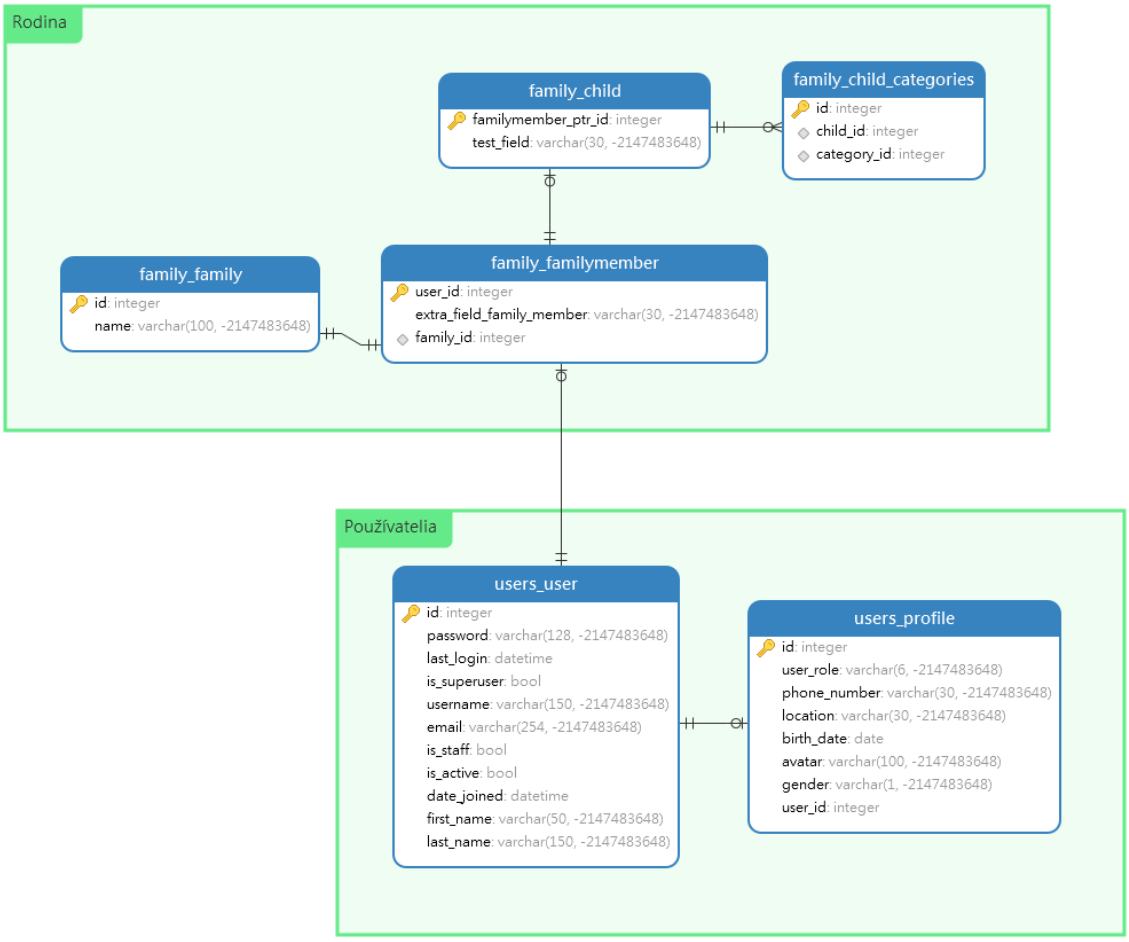
Týmto návrhom ešte budeme rozširovať aj zabudovaného administrátora v Djangu. To znamená, že náš admin bude tiež používateľ a bude mať potrebné tabuľky, ale už nebude obsahovať žiadne iné nižšie spomenuté databázové tabuľky z nášho návrhu.



Obrázok 5.2 - Databázový diagram - používatelia

5.4.2 Rodina

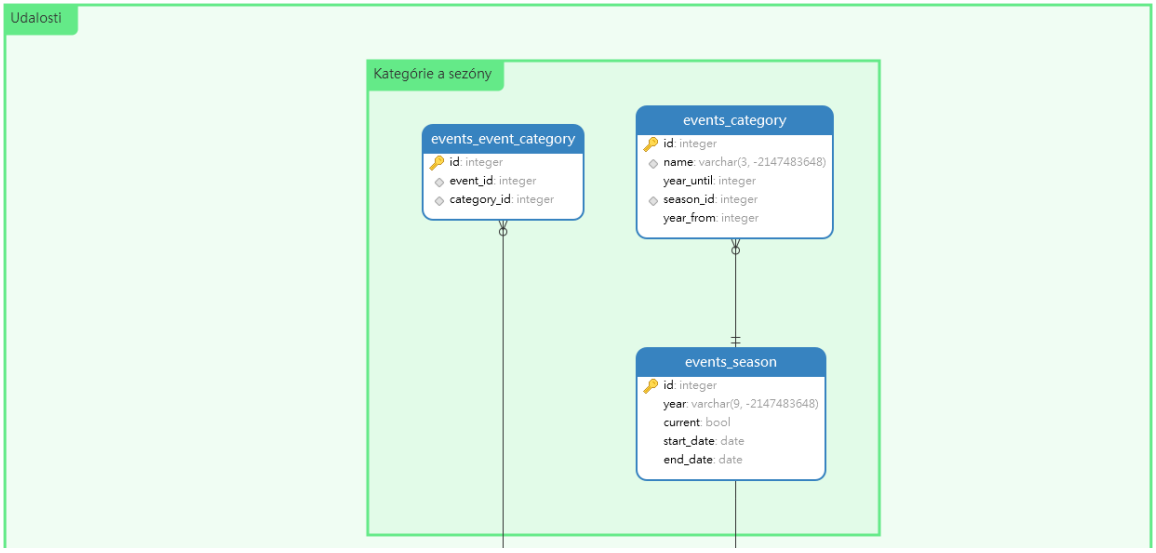
Ako som už spomínal, každý používateľ bude mať svoju rodinu, kde je zastúpený tabuľkou ***Member***. Výnimkou je len admin, ktorého databáza, resp. tabuľky sú zabudované priamo v Djangu a teda takýto používateľ nemá žiadnu z uvedených databázových tabuliek. Táto tabuľka môže byť ešte rozšírená o tabuľku ***Child*** v tom prípade, že užívateľ je typu „***Child***“ a potrebujeme evidovať extra informácie o dieťati vid Obrázok 5.3. Aby tieto informácie zbytočne neobsahoval aj rodič, budú uložené v samostatnej tabuľke. Tabuľka ***Family*** slúži na identifikáciu, že títo užívatelia patria do rovnakej rodiny, a v budúcnosti môže byť táto funkcionalita jednoducho rozšírená.



Obrázok 5.3 - Databázový diagram - rodina

5.4.3 Kategórie a sezóny

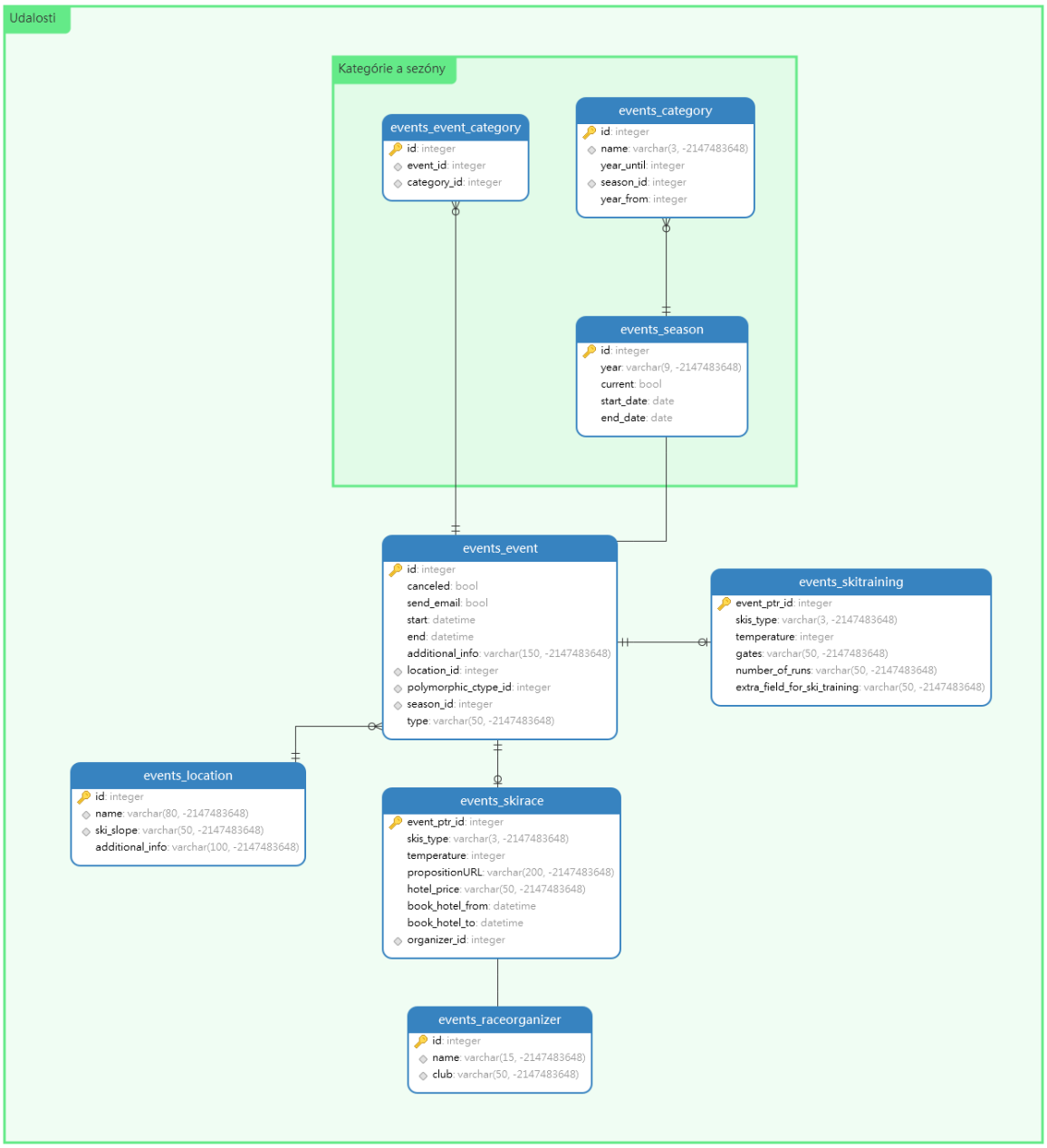
Každá sezóna obsahuje kategórie, definované ročníkmi narodenia účastníkov v tejto kategórií. Táto skutočnosť je zastúpená v tabuľke ***Category***, ktorá má cudzí kľúč z tabuľky ***Season***. Každá sezóna sa vyhodnocuje samostatne, preto sme zvolili takýto návrh, aby sme aj v budúcnosti vedeli, aká kategória bolo dieťa v konkrétnom roku či sezóne. Za povšimnutie stojí tiež fakt, že sezóna obsahuje stĺpec s názvom *current*, ktorý značí, či ide o momentálne prebiehajúcu sezónu.



Obrázok 5.4 - Databázový diagram - kategórie a sezóny

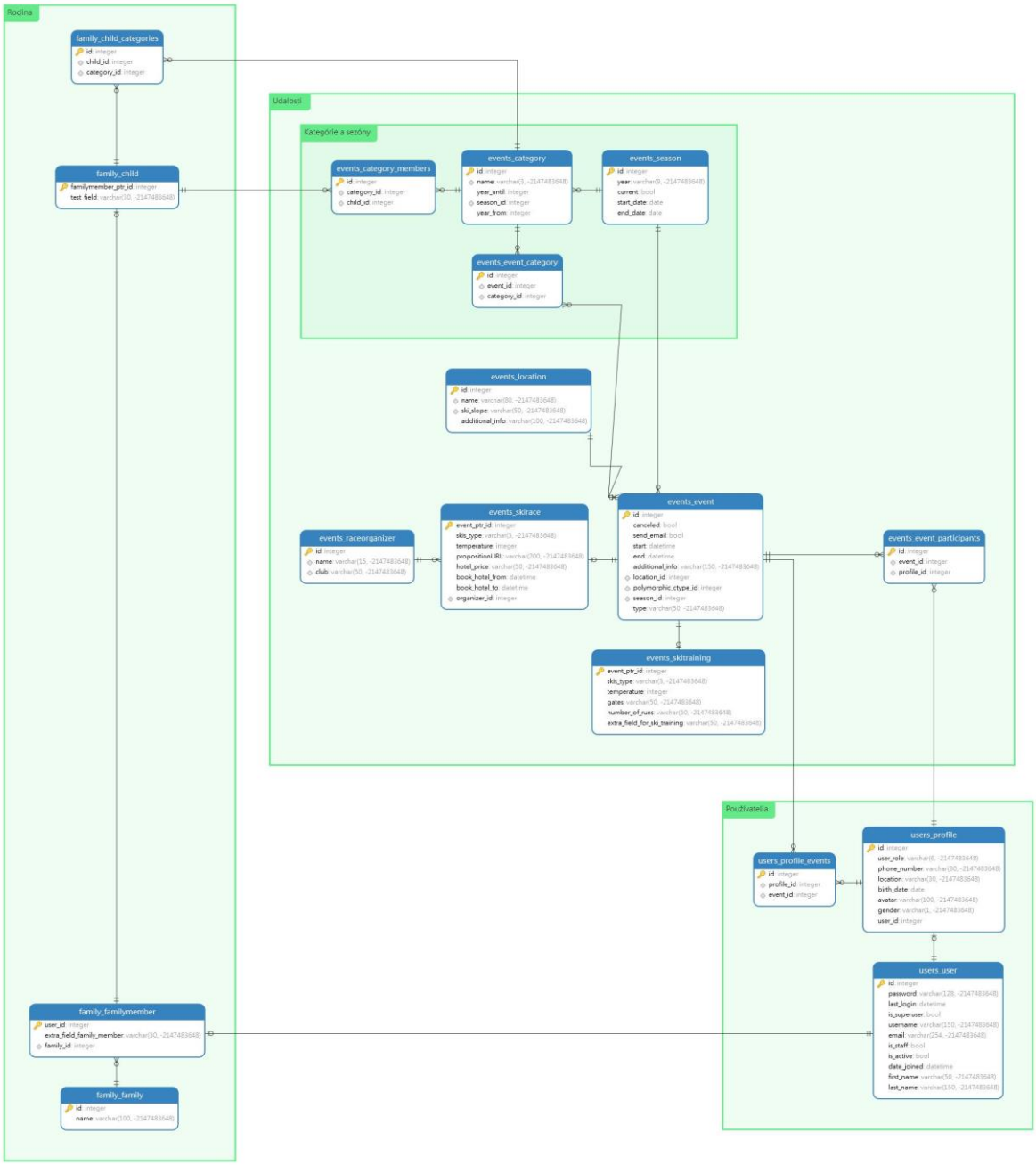
5.4.4 Udalosti

Samotné udalosti sú navrhnuté pomocou tabuľky ***Evens***. Táto tabuľka slúži pre čo najuniverzálnejšie riešenie typu jednotlivých eventov. Ako vidíme na Obrázok 5.5, tabuľka obsahuje stĺpec s názvom ***type***, kde si budeme ukladať preddefinované typy udalostí. Často sa stáva, že udalosť potrebuje len miesto, čas začatia, čas konca a pod. Takéto udalosti sú uložené iba v tejto tabuľke. Špeciálnym prípadom je napríklad ***Ski Training*** či ***Ski Race***. V týchto prípadoch si potrebujeme ukladať množstvo ďalších dodatočných dát. Tieto typy preto majú ešte dodatočné samostatné tabuľky, ktoré ale vychádzajú z tabuľky ***Event.***



Obrázok 5.5 - Databázový diagram - udalosti

Celkový pohľad na databázový model, nám zobrazuje Obrázok 5.6. Tu môžeme vidieť ako sú prepojené jednotlivé časti, ktoré sme si opísali.



Obrázok 5.6 - Databázový diagram

5.5 Návrh klientskej časti aplikácie

Klientom bude webová aplikácia bežiaca vo webovom prehliadači užívateľa, poskytujúca žiadanú funkcionalitu. Hoci framework Django vie poskytnúť aj frontend pomocou spomínaných šablón, v našej aplikácií túto funkcionalitu nevyužijeme. Napriek tomu, že neexistuje žiadne pravidlo, ktoré by určovalo, či potrebujeme alebo nepotrebujeme

framework na klientsku časť aplikácie, pozrieme sa na benefity, ktoré to môže mať, a prečo by sme sa tým mali zaoberať.

5.5.1 Single-page aplikácie

Single-page aplikácia (SPA) je aplikácia, ktorá funguje vo vnútri prehliadača a počas používania nevyžaduje opätovné načítanie stránky. Tento typ aplikácií je používaný každý deň. Sú to aplikácie ako napríklad: Gmail, Mapy Google, Facebook alebo GitHub.

SPA sa snažia napodobniť „prirodzené“ prostredie v prehliadači - žiadne načítanie stránky, žiadne ďalšie čakanie. Je to len jedna webová stránka, ktorú navštevujeme a ktorá potom načíta všetok ostatný obsah pomocou JavaScriptu, od ktorého veľmi závisí. [7]

Táto technológia je častokrát označovaná ako technológia pre tvorbu aplikácií novej generácie. S prístupom SPA sa frontend a backend stávajú takmer samostatnými projektmi. Komunikujú len prostredníctvom API za účelom výmeny údajov.

Hoci sme už nejaké výhody technológie spomenuli, uveďme si ucelený zoznam tých najdôležitejších [8]:

- SPA kombinuje dohromady to najlepšie z desktopu a webu
- Aplikácie okamžite reagujú
- Aplikácie sú multiplatformové a zároveň pôsobia natívnym dojmom
- Aplikácie môžu fungovať čiastočne offline
- Jasné rozhranie medzi frontendom a backendom
- Vývojár klientskeho rozhrania sa nemusí učiť Django na to, aby načítal údaje prostredníctvom rozhrania REST API.
- Príjemnejší používateľský zážitok s kratšími časmi načítania a menej vnímanými prechodmi - údaje sa načítajú na pozadí a znova sa načíta iba časť stránky
- Ak máme viac typov klientov (web, mobil), na backende stále potrebujeme iba jedno API. To šetrí prácu na serverovej strane aplikácie.

Tieto výhody ale nie sú zadarmo. Ak sa rozhodneme pre stavbu SPA, sú tu niektoré veci, ktoré nám prácu môžu naopak skomplikovať:

- Tím/Jednotlivec teraz pracuje na troch projektoch namiesto jedného. Musíme implementovať backend Django, API a nový frontend projekt. To zvyšuje zložitosť.
- Jednoduché veci sa môžu skomplikovať - napríklad ako autentifikovať svoju frontendovú reláciu pomocou backendu a ako správne posielať tokeny CSRF.
- Stratíme niektoré zo vstavaných výhod Djanga
- Budeme mať duplicitné úsilie. Niektorú logiku bude potrebné implementovať do backendu aj frontendu.
- V závislosti od projektu môžu vzniknúť obavy týkajúce sa SEO. Vzniká tu ale práca navyše, aby SPA fungovali bezproblémovo s vyhľadávacími nástrojmi.
- Časy načítania sa môžu zvýšiť, ak si nedáme pozor na veľkosť balíčka frontendovej aplikácie.

5.6 PWA

Progresívna webová aplikácia (PWA) je typ aplikačného softvéru dodávaného prostredníctvom webu, zostavená pomocou bežných webových technológií vrátane HTML, CSS a JavaScript. Je určená na prácu na akejkoľvek platforme, ktorá používa prehliadač kompatibilný s normami. Medzi významné funkcie patrí práca v režime offline, push oznámenia a prístup k hardvéru zariadenia, čo umožňuje vytváranie používateľských zážitkov podobných natívnym aplikáciám na stolných a mobilných zariadeniach.

Keďže progresívna webová aplikácia je typom webovej stránky, alebo webovej stránky známej ako webová aplikácia, vývojári ani používatelia nemusia inštalovať webové aplikácie prostredníctvom digitálnych distribučných systémov, ako sú Apple App Store alebo Google Play.

Takýto typ aplikácie prináša možnosť pracovať v offline režime, ktorá bola predtým k dispozícii iba pre natívne aplikácie, PWA môžu fungovať omnoho rýchlejšie a poskytovať viac funkcií, čím sa zmenšujú rozdiely s natívnymi aplikáciami.

PWA nevyžadujú samostatné zoskupovanie alebo distribúciu. Publikácia progresívnej webovej aplikácie je rovnaká ako v prípade akejkoľvek inej webovej stránky. PWA fungujú v akomkoľvek prehliadači, ale funkcie ako nezávislosť od pripojenia,

inštalácia na domácu obrazovku a zasielanie push notifikácií, závisia od podpory prehliadača. [9]

Naša aplikácia v čase písania práce spĺňa požiadavky potrebné na inštaláciu v prípade, že bude prístupná pomocou HTTPS protokolu. Takýto typ aplikácie sme používali hlavne preto, že jej implementácia je relatívne jednoduchá a nijako nezasahuje do už existujúcej aplikácie. Naopak, takýto krok nám vie v budúcnosti poskytnúť viaceré výhody.

6 IMPLEMENTÁCIA

V tejto kapitole sa pozrieme na použité technológie pri vývoji systému, či už serverovej alebo klientskej časti aplikácie a následne si opíšeme základné knižnice a frameworky využívané pri vývoji. Nakoniec si uvedieme problémy, ktoré počas implementácie nastali, či ukážky jednotlivých častí aplikácie.

6.1 Vývoj serverovej aplikácie

Prvou časťou práce bolo vyvinúť serverovú aplikáciu, ktorá bude poskytovať údaje klientovi. Tieto údaje budú uložené v databáze. Úlohou tejto aplikácie bude okrem iného prístup k týmto dátam a ich úprava, ako aj zabezpečenie celej logiky či administrátorského rozhrania.

6.1.1 Django

https://www.djangoproject.com/

Django je webový framework s otvoreným zdrojovým kódom s jazykom Python, ktorého cieľom je flexibilita, jednoduchosť, škálovateľnosť a spoľahlivosť. Framework je zbierka modulov a predpísaného kódu, ktorý pomáha vývojárom s konkrétnymi úlohami.

Tento nástroj sme použili kvôli jeho jednoduchosti a narastajúcej popularite. Vďaka jazyku Python je pomerne jednoduchý na učenie a poskytuje nám užitočné zabudované funkcionality, ktoré nemusíme programovať nanovo, ako napríklad administrátorské rozhranie, ORM modul 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, Mozzila, Dropbox, Disqus či Spotify. [8]

6.1.2 Štruktúra projektu

Projekt bol vytvorený pomocou IDE PyCharm, kde je vytvorenie takéhoto projektu jednoduché a zaberie pár klikov. Stačí si vybrať Django projekt a PyCharm sa postará o všetko ostatné. V našej aplikácií sme nepoužívali virtuálne prostredie a nástroj pip na inštalovanie knižníc, ale použili sme jeho pravdepodobnú budúcu náhradu pipenv.

Tento nástroj (ako jeho názov napovedá) spája funkcionalitu virtuálneho prostredia a inštalácie knižníc. Funguje však presne na rovnakom princípe, a teda vytvára virtuálne prostredie pre program a inštaluje do tohto prostredia knižnice.

Štruktúra našej serverovej aplikácie sa skladá z celkovo troch „aplikácií“ (názov Djanga, dal by sa prirovnať k balíčkom), pričom každá z nich sa snaží o čo najväčšiu samostatnosť. Cieľom takéhoto nastavenia je jednoduchá škálovateľnosť aplikácie, rovnako ako aj prehľadnosť celého projektu. Každá aplikácia je zodpovedná za jednu logickú časť projektu, a v prípade potreby tieto aplikácie spolu komunikujú. Výhodou je, že každá z týchto aplikácií obsahuje vlastné modely, smerovania a náhľady, ktoré sú pre tento celok špecifické. Rovnako aj fakt, že ich môžeme znovu použiť v inom projekte, alebo naopak jednoducho pridať ďalšiu aplikáciu.

Súbor manage.py

Dôležitou súčasťou Django projektu je špecifický súbor *manage.py*, pomocou ktorého napríklad dokážeme na aplikáciu vytvoriť, spustiť alebo vytvoriť migrácie databázy. Po vytvorení samotného projektu sme pomocou príkazu „*python manage.py startapp users*“ vytvorili aplikáciu „users“, teda používatelia. Následne sme v súbore *models.py* špecifikovali databázové modely potrebné pre danú aplikáciu. Pre lepšiu prehľadnosť sme vytvorili dodatočný priečinok s názvom api, kde sme umiestnili dodatočné súbory potrebné pre api rozhranie aplikácie. Pre lepšie priblíženie, štruktúra aplikácie po prvom kroku vyzerala nasledovne.

```
[SportAgenda]                <- koreňový priečinok aplikácie
+---backend
|   |   db.sqlite3           <- databázový súbor
|   |   manage.py
|   |   Pipfile              <- súbor obsahujúci všetky potrebné knižnice
|   +---apps
|   |   +---users
|   |   |   |   admin.py
|   |   |   |   apps.py
|   |   |   |   models.py
|   |   |   +---api
|   |   |   |   |   permissions.py
|   |   |   |   |   serializers.py
|   |   |   |   |   urls.py
|   |   |   |   |   views.py
|   |   |   +---migrations   <- priečinok s generovanými migráciami pre databázu
|   |   |   +---tests        <- priečinok pripravený pre unit test súbory
|   +---config
|   |   |   api.py
|   |   |   settings.py
|   |   |   urls.py
|   |   |   wsgi.py
```

Rovnakým postupom sme následne vytvorili aplikácie „*family*“ a „*events*“. Takýmito krokmi nám vznikli tri samostatné časti serverovej aplikácie. Každá z týchto častí obsahuje minimálne 5 základných súborov a to konkrétne models.py a priečinku *api* sú to *urls.py*, *views.py*, *serializers.py* a *permissions.py*.

6.1.3 Modely

Každá z už spomínaných aplikácií má vlastný súbor *models.py*, kde sa nachádzajú všetky potrebné modely. Takéto rozdelenie sme zvolili za účelom lepšej prehľadnosti. V týchto modeloch vytvárame triedy, ktoré tvoria základ pre ORM.

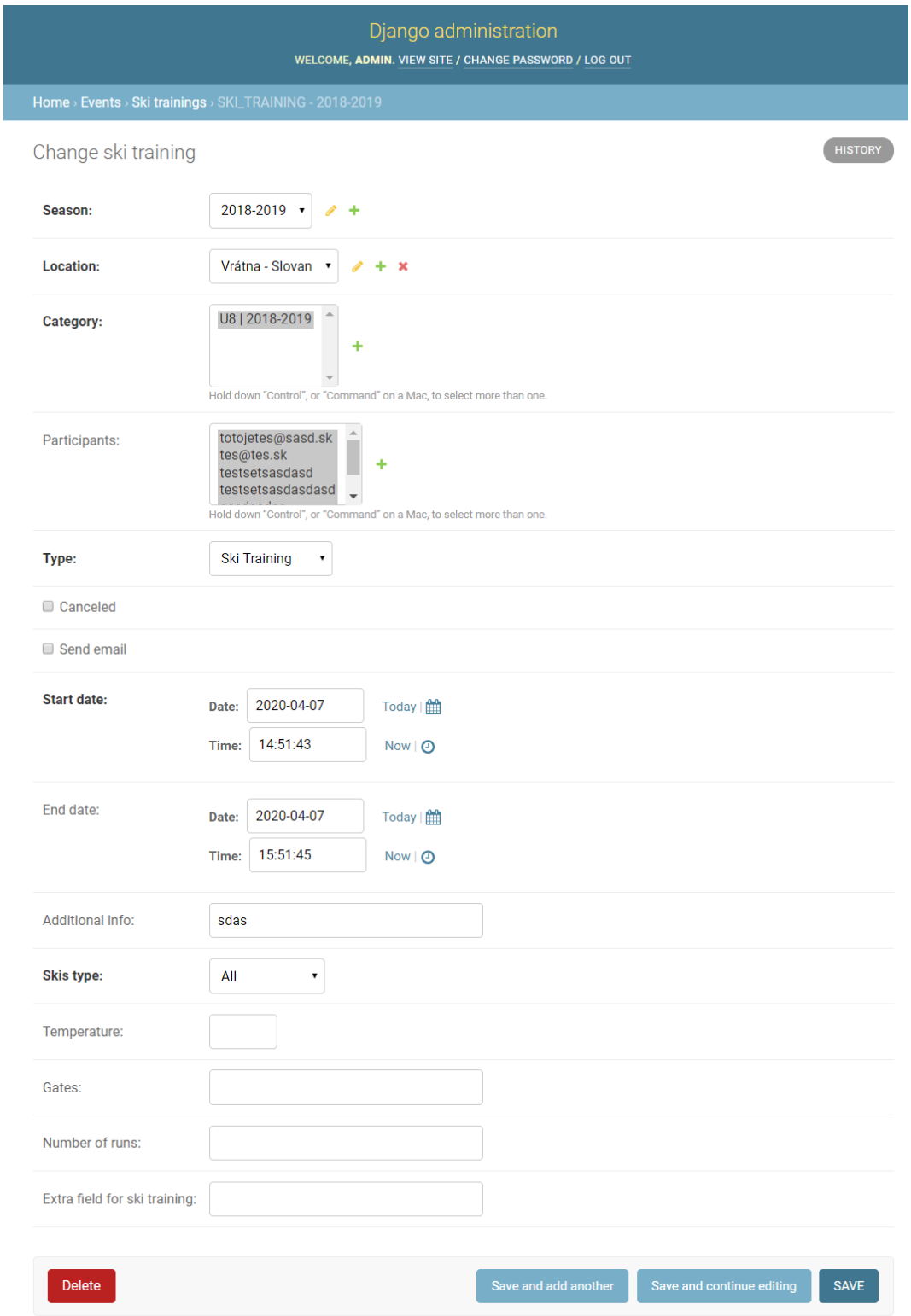
ORM

ORM je objektovo relačné mapovanie, ktoré je spravidla súčasťou frameworkov. V našom prípade ho využijeme ako prekladač medzi databázou a objektovo orientovaným programovacím jazykom. Výhodou takéhoto mapovania je, že vývojári nemusia mať tak veľkú znalosť práce s databázou, no napriek tomu ju môžu ľahko a efektívne využívať.

V našom prípade sa tento ORM používa nasledujúcim spôsobom. Vývojár vytvorí model, teda triedu, kde zadefinuje všetky potrebné vlastnosti a vzťahy. Následne si vie príkazom „*python manage.py makemigrations*“ vytvoriť súbory s obsahujúcimi migráciami. Tieto migrácie aplikuje spustením príkazu „*python manage.py migrate*“. V prípade, že je všetko v poriadku, sa tieto migrácie aplikujú na databázu.

6.1.4 Administrátorské rozhranie

Po vytvorení jednotlivých migrácií pre aplikácie, je potrebné tieto modely ešte zaregistrovať do administrátorského rozhrania. Tým sa umožní následná interakcia s nimi. Po tomto zaregistrovaní môžeme prejsť do rozhrania pre administrátorov, kde môže administrátor pridať, editovať či vymazať jednotlivé záznamy v databáze. Ako môžeme vidieť na príklade Obrázok 6.1, kde je zobrazené vytváranie nového lyžiarskeho tréningu na snehu pomocou spomínaného administrátorského rozhrania.



Obrázok 6.1 - Ukážka administrátorského rozhrania - Event

6.1.5 URL adresy

Základné rozdelenie URL adries je rozdelené na 3 časti. Sú to:

- ***„/admin/“*** - má na starosti adresovanie spomínaného Django admin modulu a obsahuje teda balík preddefinovaných URL adries Djanga.
- **„/api/rest-auth/“**- táto časť adries slúži na autentifikáciu pomocou api rozhrania
- **„/api/“** - obsahujú adresy pre api rozhranie uložené vo vyššie spomínaných aplikáciách.

Na príklade kódu nižšie môžeme vidieť, ako takéto časti vyzerajú priamo v aplikácií.

```
1. urlpatterns = [
2.     path('admin/', admin.site.urls),
3.     path("api/rest-auth/", include('core.auth')),
4.     path('api/', include('config.api')),
5. ]
```

6.1.6 Django Rest Framework

https://www.django-rest-framework.org/

Django Rest Framework je výkonná a flexibilná sada nástrojov na vytváranie webových rozhraní API. Je to najrozšírenejší spôsob vytvárania REST API spolu s Djangom. Je to hlavne z toho dôvodu, že vytváranie takýchto koncových bodov a zároveň ich spracovávanie, je omnoho jednoduchšie a rýchlejšie.

Implementácia

Ako všetky sady nástrojov, aj tento má svoje pravidlá, ktoré je potrebné dodržiavať. Pri jeho implementácii je v zásade nutné vykonanie nasledujúcich krokov:

1. Definícia serializovania dát
2. Vytvorenie kontroléra
3. Pridanie smerovania kontroléra

Pri vytváraní URL adries používame zabudovaný router, ktorý nám automaticky vytvorí základné koncové body, ktoré je možné upraviť podľa našej potreby. V zásade používame dva základné typy:

- ListView - používaný pri metódach vracajúcich zoznam objektov
- DetailView - používaný pri volaniach konkrétneho objektu, napríklad jeho upravovanie

6.1.7 Views

Pohľady (Views) tvoria samostatné funkcie, volané a vykonávane v prípade požiadavky na URL adresu. Všetky náhľady sú opäť v jednotlivých aplikáciách v priečinku api v súbore *views.py*. Tieto náhľady musia pre fungovanie obsahovať dve základné veci, a to *queryset* a *serializer_class*. *Queryset* nám označuje, aké objekty z databázy tento pohľad bude volať, a *serializer_class* označuje serializačnú triedu, ktorú chceme použiť. Na ukážke nižšie môžeme vidieť ešte *permission_classes*, tento atribút nám bude zabezpečovať oprávnenia na takomto pohľade.

```
1. class EventViewSet(viewsets.ModelViewSet):
2.     queryset = Events.objects.all()
3.     serializer_class = EventPolymorphicSerializer
4.     permission_classes = [IsAuthenticated, IsCoachOrReadOnly]
```

6.1.8 Serializers

Z predchádzajúcej ukážky zdrojového kódu vidíme jednoduchosť takéhoto modelu. V nasledujúcej ukážke si uvedieme príklad pre serializačnú triedu. Táto trieda vytvára akúsi šablónu či vzor pre API. Jednoznačne definuje, aké dáta budeme používať a aké budú mať parametre. V prípade, že tieto parametre nedefinujeme, primárne sa používajú polia z modelov. Táto ukážka ukazuje veľmi základnú serializáciu

```
1. class EventSerializer(BaseEventSerializer):
2.
3.     class Meta:
4.         model = Event
5.         fields = "__all__"
```

V našom projekte používame, ale aj omnoho zložitejšie kontroléry. Napríklad pri udalostiach, kde pre všetky udalosti existuje jeden kontrlér, aby sme ich mali všetky v jednom liste, bez ohľadu na typ. Toto môže byť problém, vzhľadom na to, že lyžiarsky tréning a lyžiarske preteky majú samostatné databázové tabuľky. Pri takomto riešení je

potrebné myslieť aj na to, že v budúcnosti by mohli takéto tabuľky ešte pribudnúť pre iné typy udalostí.

Takto pripravený kontróler s použitím serializovaných dát je funkčný a pripravený na použitie. Vzhľadom na to, že naša aplikácia ale obsahuje určité role a nie všetci užívatelia budú mať možnosť upravovať dáta, je nutné definovanie používateľských práv.

Oprávnenia (permissions) nám definujú, kto daný pohľad môže upravovať, resp. aké metódy môže nad pohľadom zavolať. Sú uložené v samostatnom súbore s názvom *permissions.py*. V tomto súbore je vytvorená trieda, v ktorej vyhodnocujeme žiadosti na základe žiadanej funkcionality. Na príklade nižšie môžeme vidieť, že porovnávame či žiadosť prišla je jedna zo SAFE_METHODS. Tieto sú určené ako žiadosti typu 'GET', 'HEAD', 'OPTIONS', pretože pri takýchto žiadostiach nie je možná zmena dát. Ďalej sa v kóde overuje, či používateľ, ktorý žiada tento pohľad je typu tréner. V takom prípade pohľad zobrazíme, v opačnom prípade vrátime HTTP odpoveď s kódom 403, čo značí, že tento užívateľ nemá právo na tento pohľad. Teraz je pohľad zabezpečený a teda plne pripravený akceptovať požiadavky.

```
1.  class IsCoachOrReadOnly(permissions.BasePermission):
2.      def has_object_permission(self, request, view, obj):
3.          if request.method in permissions.SAFE_METHODS:
4.              return True
5.          return request.user.profile.user_type == choices.COACH
```

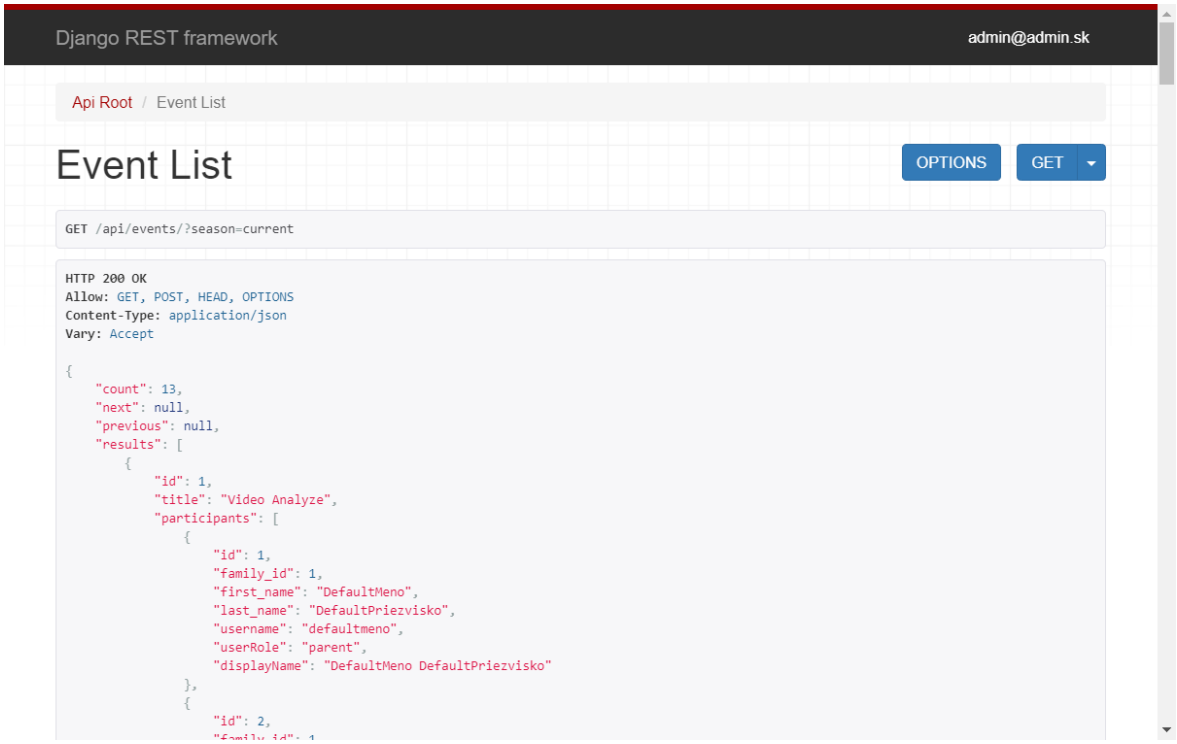
6.1.9 Koncový bod pre udalosti

Keď sme si prešli základné vytváranie prístupových bodov, pozrime sa na jeden konkrétny príklad a to udalosti (events). Je dobré sa pozrieť ešte na to, ako sú uvedené URL adresy pre túto časť serverovej aplikácie. Na ukážke nižšie vidíme, že máme dva koncové body, a síce events, kde s je nepovinné, to znamená, že táto adresa funguje rovnako pri zadaní *„/api/event/“* tak aj pri zadaní *„/api/event**s**/“*. Rovnako je to aj v prípade sezón.

```
1.  router = DefaultRouter()
2.
3.  router.register(r"events?", EventViewSet, basename="event")
4.  router.register(r"seasons?", SeasonViewSet, basename="season")
5.
6.  urlpatterns = [
7.      path("", include(router.urls)),
8.  ]
```

Zoznam udalostí

Udalosti si môžeme zobraziť na adrese „*/api/events/*“. V prípade zadania takejto adresy sa nám zobrazia všetky udalosti v systéme. Vzhľadom na to, že na klientskej strane sa bude vyhodnocovať najmä aktuálna sezóna, môžeme na takýto koncový bod použiť filter. V tomto konkrétnom prípade môžeme filter nastaviť ako „*season=current*“ alebo rok danej sezóny, t.z. napríklad „*season=2018-2019*“. V našom prípade je adresa pre všetky udalosti konajúce sa v aktuálnej sezóne „*<IP alebo doména><:port>/api/events/?season=current*“. Ako takýto endpoint vyzerá v prehliadači, môžeme vidieť na Obrázok 6.2.



Obrázok 6.2 – Ukážka api zoznamu udalostí

Detail udalosti

Pre zobrazenie konkrétnej udalosti je potrebné vedieť id tejto udalosti. Následne vieme vyvolať koncový bod našej serverovej aplikácie, a to s použitím adresy *„<IP alebo doména><:port>/api/event/1/“*, kde číslo 1 označuje id konkrétnej udalosti.

Na takomto koncovom bode môže podľa práv, ktoré sme si definovali vyššie, posielať užívateľ jednotlivé HTTP žiadosti. To znamená, tréner vie udalosť zmazať a teda poslať „DETELE“ požiadavku, rovnako ako aj upraviť (PATCH žiadosť). V prípade, že sa o takúto požiadavku pokúsi užívateľ, ktorý nemá dostatočné práva, požiadavka bude zamietnutá.

Pridanie používateľov na udalosť

Tento koncový bod je špecifický ešte jednou funkcionalitou, ktorá stojí za povšimnutie. Na udalosti sa prihlasujú a odhlasujú deti zo všetkých rodín, ale rodič môže prihlásiť/odhlásiť len svoje dieťa. Zároveň mu však nedávame možnosť editovať túto udalosť. Z toho dôvodu sme vytvorili nový koncový bod *„<IP alebo doména><:port>/api/event/1/**change**/“*, kde 1 stále označuje id udalosti.

Na tejto adrese umožňujeme užívateľom posielať POST žiadosti, spolu s údajmi o používateľoch, ktorých chcú na udalosť pridať alebo odstrániť.

6.2 Vývoj klientskej aplikácie

Pred začatím implementácie klientskej aplikácie je dôležité si uvedomiť fakt, že aplikácia má slúžiť všetkým členom športového klubu na lepšie organizovanie ich činností. Je teda veľmi podstatné, aby ju mali možnosť používať všetci členovia klubu. Poďme sa pozrieť na proces vytvorenia klientskej aplikácie, ktorá bude mať na starosti najmä zobrazovanie dát používateľom a možnosť s nimi intuitívne narábať pomocou grafických prvkov.

6.2.1 Vue.js

https://vuejs.org/

Vue.js patrí medzi najpopulárnejšie frameworky pre tvorbu SPA. Spolu s Angularom a Reactom kraľujú rebríčku najpoužívanejších Javascript frameworkov. Veľká výhoda Vue spočíva v tom, že je minimalistický a rýchly. Samotné jadro Vue.js slúži, podobne ako v prípade Reactu, hlavne na dynamickú prácu s UI. Pokročilejšie veci ako smerovanie (*routing*) a podobne je potrebné doplniť cez dodatočné knižnice. Framework je tiež veľmi jednoduchý na naučenie.

Vue.js je inšpirovaný dvoma skvelými webovými fameworkami, t.j. React and Angular. Berie tie najlepšie časti z nich oboch. Rovnako ako React, je založený na komponentoch a používa Virtual DOM, vďaka čomu je mimoriadne rýchly. Nejde o knižnicu (na rozdiel od React.js), a nie o úplnú rámcovú štruktúru (na rozdiel od Angularu). Má skvelú rovnováhu funkcií, ktoré sú zabudované a je neuveriteľne ľahké pridať ďalšie funkcionality. Jednou z nich je aj Vuex.

Jeho implementácia do nášho projektu bola pomerne priamočiara. Predpokladom nainštalovania je mať nainštalovaný node a nástroj npm. Následne môžeme zadať príkaz „*npm install -g @vue/cli*“, ktorým si nainštalujeme knižnicu pre vytváranie Vue aplikácie. Po úspešnej inštalácií knižnice si následne v našom koreňovom priečinku príkazom „*vue create frontend*“ a prejdením jednotlivých krokov vytvoríme novú Vue aplikáciu.

6.2.2 Vuexy

https://pixinvent.com/demo/vuexy-vuejs-admin-dashboard-template/landing/

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. Okrem toho, zvyčajne na takýchto veľkých šablónach, ako sme zvolili v našej aplikácií, pracujú celé tímy dizajnérov a programátorov, čo je predpoklad na kvalitu takejto šablóny..

Základ našej klientskej aplikácie preto tvorí šablóna Vuexy, ktorá obsahuje preddefinované usporiadania a komponenty, ktoré si následnej vieme doupravovať, podľa našich požiadaviek. Takýmto spôsobom zabezpečíme prívetivý a jednotný UX a UI naprieč celou našou aplikáciou. Jej neimplementovanie prebehlo bez problémov, následne bolo potrebné len doinštalovať knižnice, ktoré téma potrebuje a aplikácia sa dala spustiť.

6.2.3 Vuex

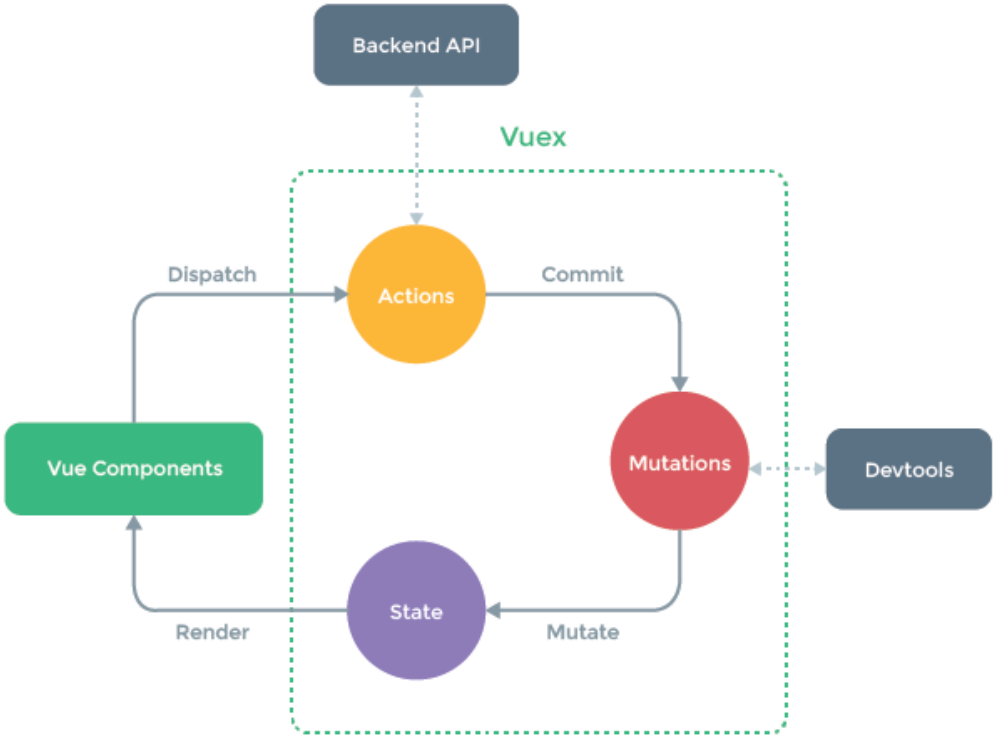
https://vuex.vuejs.org/

VUEX je knižnica, ktorá slúži na správu stavu aplikácie. V zložitejšej aplikácii s väčším množstvom komponentov môže byť výmena a úprava informácií medzi jednotlivými komponentmi dosť komplikovanou záležitosťou. VUEX nám preto poskytuje možnosť držať stav aplikácie na jednom mieste, do ktorého budú mať prístup všetky komponenty.

Vo VUEX sa robí priamo zmena hodnôt v existujúcom stave a to cez **mutácie**. Tie priamo menia hodnoty v stave. Mutácie sa vždy vykonávajú **synchrónne**.

Okrem mutácií tu existujú aj **akcie**. Akcie obsahujú logiku zmeny stavu a následne vyvolávajú mutácie, čím dochádza k zmene stavu aplikácie. Vykonávanie akcií prebieha **asynchrónne**. [10]

Celá táto funkcionalita je graficky znázornená na Obrázok 6.3.



Obrázok 6.3 - Pohľad na fungovanie Vuex

6.2.4 Fullcalendar

https://fullcalendar.io/

Vzhľadom na to, že naše udalosti chceme používateľom aplikácie zobrazovať nielen vo forme nejakej tabuľky či zoznamu, zvolili sme knižnicu Fullcalendar. Táto knižnica je najpoužívanejšia naprieč tromi najpopulárnejšími frontendovými frameworkami a má prehľadne spracovanú dokumentáciu. Z nej sa dá vyčítať všetko potrebné, navyše obsahuje aj praktické príklady a ukážky kódu.

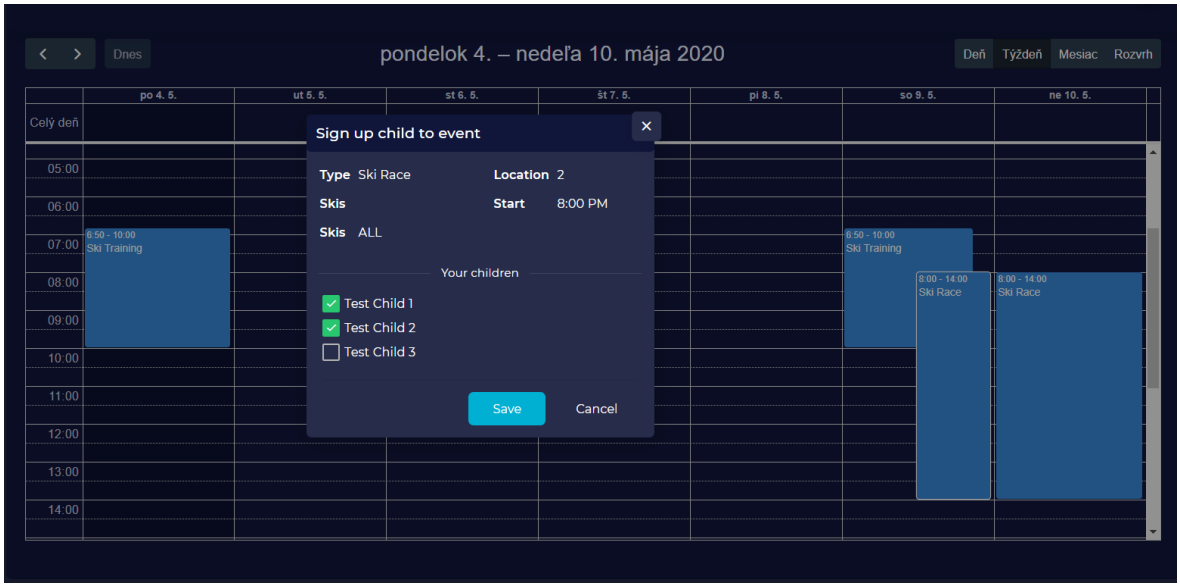
Knižnica dokáže zobrazovať prehľad dňa, týždňa, mesiaca či roka. Takisto dokáže prehľadne zobraziť list takýchto udalostí, spolu s dátami potrebnými pre používateľa aplikácie. Táto implementácia by mala používateľovi prehľadne zobraziť udalosti a umožniť mu tiež ďalšiu interakciu s nimi.

Intuitívny dizajn

Kalendár je komponent, na ktorom budú pravdepodobne užívatelia tráviť najviac času, preto je potrebné zabezpečiť čo najlepší užívateľský zážitok. Je tu zabezpečený prehľad udalostí, kedy užívateľ vidí presne, kedy sa ktorá udalosť koná. Napríklad prehľad týždňa je veľmi praktický, pretože sa môže stať, že cez víkendy sa preteky konajú paralelne s tréningami a je potrebné takúto situáciu prehľadne zobraziť. Netreba pritom zabúdať ani na responzívny dizajn, aby sme zabezpečili jednoduché používanie na všetkých zariadeniach.

Detail udalosti a prihlásenie

Po kliknutí na jednotlivé udalosti sa zobrazí okno, ktoré obsahuje základné detaily udalosti, ako napríklad typ lyží alebo miesto konania tejto udalosti. Čo je ale podstatnejšie, po kliknutí na udalosť môže užívateľ na túto udalosť prihlásiť všetky deti z jeho rodiny viď. Obrázok 6.4.



Obrázok 6.4 - Pohľad na kalendár udalostí

Tieto údaje sa ukladajú do pamäti aplikácie a pri kliknutí na tlačidlo uložiť sa odošle požiadavka na server na pridanie týchto užívateľov na danú udalosť. Ten požiadavku spracuje a v situácií, ak by sa v tomto procese vyskytla nejaká chyba, klientska časť chybu zachytí a vypíše pomocou upozornenia. V prípade, že pri užívateľovi svieti zaškrtnutý zelený štvorček, užívateľ je na udalosť prihlásený, naopak, ak je prázdny, tak prihlásený nie je.

6.3 Problémy spojené s vývojom webovej aplikácie

Pri vývoji akéhokoľvek softvéru sa takmer vždy vyskytne zopár problémov, ktoré je potrebné vyriešiť. Nižšie si uvedieme, akým problémom sme museli čeliť pri našom vývoji.

„*It works on my machine*“ alebo v preklade „*Na mojom zariadení to funguje*“ je problém, s ktorým sa určite stretol každý vývojár. Tento problém je výraznejší pri webových aplikáciách, kde hrá úlohu množstvo faktorov. Od klientskej strany, kedy má používateľ napríklad niečo uložené v pamäti cache v prehliadači, až po proces nasadenia aplikácie na server.

Častokrát sa stáva, a aj v našej aplikácií sme sa s tým počas vývoja stretli, že aplikácia na počítači na ktorom sme ju primárne vyvíjali fungovala bez problémov, ale pri nasadení na testovací server s iným operačným systémom, či pri vývoji na inom počítači sa vyskytovali rôzne chyby. Primárne sa jednalo o chyby jednotlivých knižníc (iná verzia pre Windows ako pre Linux), zlá verzia knižnice, či chýbajúce závislosti. Rovnako, pri každom zariadení na ktorom chceme túto aplikáciu spustiť, si najskôr musíme nakonfigurovať všetky potrebné veci. Až po úspešnom absolvovaní tohto kroku je možné aplikáciu spustiť.

Tento proces môže byť niekedy naozaj komplikovaný, a v prípade viacerých zariadení strávime veľa času nastavovaním a riešením problémov s týmito nastaveniami namiesto práce na aplikácií ako takej. S týmto problémom sa stretáva množstvo vývojárov, preto existujú rôzne nástroje, ktoré presne takéto situácie riešia. V našej aplikácií sme zvolili Docker, a v nasledujúcej časti sa pozrieme, prečo ho používame a ako nám dokáže uľahčiť prácu.

6.4 Docker

https://www.docker.com/

Docker je jedna z popredných otvorených platforiem na vývoj a kontajnerizáciu aplikácií. Poskytuje spôsob, ako zabaliť infraštruktúru aplikácie do kontajnera, kde môže bežať izolovane od zvyšku systému, a nástroje na kontrolu kontajnerov. Docker zjednodušuje vývojový proces softvéru využívaním už popísaných praktík kontajnerizácie a mikroservisov. V nasledujúcich častiach sa pokúsime v krátkosti zhrnúť, čo to Docker je a prečo je populárnym nástrojom vo svete vývojárov.

6.4.1 Kontajner

Najjednoduchšia definícia kontajnera ho popisuje ako samostatný balík softvéru, ktorý obsahuje všetko potrebné na spustenie a chod aplikácie. Dlhšia definícia by ho popísala ako samostatný logický balík softvéru, ktorý zapuzdruje a oddeľuje aplikáciu spolu so všetkými jej závislosťami tak, aby bola aplikácia schopná bežať nezávisle od prostredia, na ktorom bola vytvorená. Cieľom je oddeliť aplikáciu a všetky jej závislosti od operačného systému a jeho prostredia takým spôsobom, aby sa dala následne jednoducho, rýchlo a spoľahlivo zapnúť na hocijakom inom prostredí.

Môžeme si to predstaviť ako kontajner na lodi. Tak, ako sa na lodi prepravuje tovar v osobitných kontajneroch, je to aj s kontajnermi vo vývoji softvéru. Pri vytváraní kontajnera by malo platiť pravidlo, že jeden kontajner je jedna služba.

6.4.2 Docker Compose

Reálny softvér sa ale bude skladať z viacerých služieb. Ak by sme chceli vytvoriť viacero Docker kontajnerov naraz, tak s tým nám pomôže Docker Compose. Je to nástroj, ktorým môžeme definovať, vytvoriť a zapnúť viaceré kontajnery jedným príkazom. Jediné, čo na to potrebujeme, je vytvoriť yaml súbor, v ktorom si zadefinujeme jednotlivé služby. [11]

6.5 Štruktúra celého projektu

Projekt je rozdelený do dvoch samostatných priečinkov a tieto priečinky sú od seba v podstate nezávislé. Klientska aplikácia sa dá spustiť samostatne, napríklad pomocou NPM, rovnako ako aj serverová časť, napríklad pomocou jazyka Python. V prípade vývoja má každá aplikácia spustený vlastný server. Tieto aplikácie navzájom komunikujú. Existuje viac spôsobov ako komunikáciu medzi týmito aplikáciami implementovať, v našej aplikácií sme spočiatku zvolili klasický spôsob, pomocou jazyka Python. Takýmto spôsobom sa spustil backend a poskytoval potrebné dáta. Keď bol backend pripravený, spustili sme druhú konzolu, v ktorej bežal frontend a aplikácie spolu vedeli komunikovať. Tento spôsob sa ukázal ako dosť neefektívny a mal viacero nevýhod. Napríklad aplikácia nezaznamenávala všetky zmeny, ktoré sme urobili, v prípade, že sme aplikáciu chceli vyvíjať na inom zariadení, museli sme všetko nanovo nastaviť. Rovnako bol aj problém s knižnicami, ktoré sme na každom zariadení museli nainštalovať samostatne. Preto sme do našej aplikácie neskôr implementovali Docker, ktorý celú situáciu vyriešil. Obsahuje backend, frontend

a nginx. Nginx je webový server, ktorý v tomto našom prípade zabezpečuje komunikáciu medzi jednotlivými kontajnermi.

S takýmito nastaveniami stačí do konzoly zadať príkaz `docker-compose up -build` a celá aplikácia sa spustí vo vývojárskom režime. Tento režim je určený pre vývojárov a spúšťa developerský server pre Django aj pre Vue. Takáto konfigurácia funkcie pomáha vývojárom zjednodušiť ich prácu. Napríklad funkcia „hot reload“, ktorá spočíva v tom, že keď zmeníme niečo v kóde, automaticky sa to v aplikácií prejaví bez nutnosti jej reštartovania.

Ak chceme aplikáciu nasadiť do produkcie, musíme klientsku aplikáciu zostaviť (build). Týmto sa nám vytvoria súbory potrebné na beh aplikácie a následne môžeme spustiť serverovú časť, ktorá nám tieto súbory bude poskytovať. V takomto prípade nepotrebujeme server pre Vue, pretože Django už poskytuje klientovi tieto vygenerované statické súbory. Tento proces opäť riešime pomocou Docker, a to jednoduchým príkazom `docker-compose -f docker-compose-prod.yml up --build -d`

6.6 Nasadenie aplikácie na server

Aplikácia bude nasadená na súkromnom serveri. Pri vývoji sme používali lokálny server resp. Docker na vlastnom počítači, ktorý bol úplne postačujúci pre vývojárske účely. Aby sa ale aplikácia dala testovať aj bez prítomnosti vývojárovho počítača alebo na inej sieti, aplikáciu sme nasadili na verejne dostupný server. Tento server je nastavený tak, aby prijímal žiadosti na verejnú URL adresu stránky klubu, a teda stačí už len spustiť potrebný príkaz a aplikácia bude nasadená a prístupná.

V budúcnosti by bolo vhodné nájsť a implementovať riešenie, ktorý by pri každom vydaní novej verzie na GitHube túto aktualizáciu aplikovalo aj na serverové súbory.

6.7 Notifikácie

Jednou z mnohým funkcionalít, ktorú ponúka naša aplikácie, je posielanie email notifikácií užívateľom o nových udalostiach. Pri tvorbe aplikácie sme mali viacero možností, ako pracovať s notifikáciami, preto si v tejto časti ešte opíšeme, ako sme túto funcionalitu implementovali. Prvou z možností bolo posielanie SMS notifikácií, ktoré by sa dali vyriešiť pomocou nejakej externej služby, ktorá takéto upozornenie dokáže poslať na základe

poskytnutých údajov a následného volania API rozhrania tejto služby. Takéto riešenie je síce spoľahlivé, ale finančne nákladné, a preto po konzultácii s vedením klubu sme sa rozhodli pre kompromis.

Takýmto kompromisom sú emaily, ktoré sa momentálne odosielajú automaticky pri zrušení tréningu, alebo sa dajú poslať manuálne. V takom prípade sa zašle email o danej udalosti, bez ohľadu na to, či bola zrušená, alebo nie. Až testovanie ukáže, či sú takéto upozornenia dostatočné, a na ktoré udalosti by mali byť ešte používatelia upozornení.

V budúcnosti je možné implementovať aj push notifikácie, ktoré by užívateľov notifikovali priamo na ich zariadeniach. Momentálne ale toto riešenie nie je pre nás prioritou, nakoľko nemáme dáta z testovania aplikácie a implementácia tejto funkcionality by bola časovo náročná a je možné, že v konečnom dôsledku aj nevyužívaná.

7 TESTOVANIE A MOŽNOSTI ROZVOJA APLIKÁCIE

Počas vývoja bola aplikácia samozrejme neustále testovaná a ladená. Súčasťou vývoja aplikácie je ale aj samotné otestovanie aplikácie v reálnom použití. V prípade aplikácie slúžiacej športovému klubu je ideálne, ak aplikáciu otestujú aj niektorí členovia klubu. Vedenie klubu preto rozhodlo o vybraných členoch, ktorú túto aplikáciu otestujú. To bolo ale ešte predtým, ako svet ovládla pandémia. Nanešťastie, momentálne sú všetky udalosti zrušené, takže potrebná funkcionalita sa nedá prakticky otestovať s reálnymi dátami. Preto som zvolil aspoň testovacie dáta, ktoré som do aplikácie nahral a tak aspoň čiastočne otestoval aplikáciu. Po skončení núdzového stavu a opätovného organizovania udalostí bude táto aplikácia pripravená na plnohodnotné beta testovanie.

7.1 Github Actions

GitHub Actions uľahčuje automatizáciu všetkých softvérových pracovných postupov. Tento nástroj nám umožňuje primárne zostavenie (*build*), test a nasadenie (*deploy*) aplikácie priamo zo služby GitHub. Táto služba je pomerne nová, bola predstavená koncom októbra 2019. V našom projekte používame túto službu najmä na zautomatizovanie procesu zostavenia aplikácie a následného spustenia testov. V prípade, že by tieto testy neboli úspešné, nástroj nás upozorní emailom. Takýto životný cyklus aplikácie pre nás znamená, že máme istotu, že náš kód prechádza testami aj bez toho, aby sme ich zakaždým museli manuálne spustiť.

7.2 Možnosti rozvoja

Vzhľadom na to, že o aplikáciu bol veľký záujem nielen v našom klube, ale aj u viacerých trénerov či predsedov iných klubov, ktorým sme túto aplikáciu odprezentovali, jej vývoj bude pravdepodobne pokračovať aj ďalej.

Ako som spomínal v úvode, vývoj aplikácie môže ďalej pokračovať pridávaním nových funkcií, ako napríklad podávanie prihlášok pre nových členov, ktoré vedenie klubu potom posiela na príslušné inštitúcie, či zaznamenávanie a vyhodnocovanie výsledkov z pretekov, alebo z testovania pohybovej výkonnosti. Z aplikácie sa môže stať komplexný nástroj pre správu akéhokoľvek športového klubu v prípade, že o jeho vývoj budú mať záujem členovia klubu alebo niekto iný.

ZÁVER

Výsledkom tejto bakalárskej práce bolo vytvorenie webovej aplikácie, ktorá zjednoduší jednotlivé procesy v lyžiarskom klube. Aplikácia je rozdelená na dve samostatné časti, a to backend a frontend. Serverová aplikácie je postavená na frameworku Django, klientska aplikácia je vytvorená pomocou frameworku Vue.js. Ako základ dizajnu slúži šablóna Vuexy.

Základ tvorí serverová aplikácia, ktorá sa stará o hlavnú logiku aplikácie, spravuje a poskytuje potrebné dáta. Klientska aplikácia má na starosť komunikáciu so serverovou časťou, a to s použitím RESTful API. Hlavnou úlohou klientskej aplikácie je získať dáta, správne ich zobraziť a poskytnúť možnosť používateľom jednoduchým spôsobom interagovať s týmito dátami.

Aplikácia bola nasadená na server a čaká na testovanie. Je verejne prístupná, takže k nej má prístup každý zo znalosťou URL adresy, čo v konečnom dôsledku znamená, že je použiteľná v reálnej prevádzke.

Pri vývoji aplikácie som sa naučil pracovať s novými frameworkami, ktoré spočiatku vyžadujú veľa času na zvládnutie týchto technológií. Je to spôsobené najmä tým, že tieto frameworky sú komplexné a robustné, v konečnom dôsledku však zjednodušujú vývoj aplikácie. Taktiež som lepšie pochopil komunikáciu medzi serverovou a klientskou aplikáciou a spôsob výmeny informácií.

Najzaujímavejšia technológia, s ktorou som sa počas vývoja stretol, bol jednoznačne Docker. Tento nástroj má určite veľký potenciál do budúcna a určite ho rád použijem aj pri mojich budúcich projektoch. V procese vývoja som narazil aj na viacero problémov, z ktorých sa mi podarilo všetky úspešne vyriešiť, čo taktiež vnímam ako pozitívnu skúsenosť.

Aplikácia musí ešte prejsť testovaním, ktoré bude možné až po tom, ako skončí karanténa a budú opäť obnovené udalosti klubu. Predpokladám, že testovanie zrejme odhalí niektoré nedostatky aplikácie, ktoré budú následne opravené. Je to hlavne preto, že často sa správne fungovanie aplikácie overí až pri praktickom použití, a preto má testovanie v reálnej prevádzke podstatný význam. Po tomto testovaní môže aplikácia slúžiť ako veľmi dobrý základ pre správu klubu a všetci členovia klubu ju budú s obľubou využívať.

ZOZNAM POUŽITEJ LITERATÚRY

[1] „Webová stránka CIO Wiki,“ 2020. [Online]. Available: https://cio-wiki.org/wiki/Client_Server_Architecture. [Cit. 18 04 2020].

[2] „Softvare Planet Group,“ [Online]. Available: https://softwareplanetgroup.co.uk/graphql-vs-rest-what-to-choose-in-2020/. [Cit. 04 19 2020].

[3] „REST API Tutorial,“ 21 04 2020. [Online]. Available: https://www.restapitutorial.com/lessons/whatisrest.html.

[4] „Stable Kernel,“ [Online]. Available: https://stablekernel.com/article/advantages-and-disadvantages-of-graphql/. [Cit. 04 30 2020].

[5] D. ČÁPKA, „ITnetwork,“ [Online]. Available: https://www.itnetwork.sk/navrh/mvc-architektura-navrhovy-vzor. [Cit. 21 04 2020].

[6] „The Django Book,“ [Online]. Available: https://djangobook.com/mdj2-django-structure/. [Cit. 27 04 2020].

[7] „Medium,“ [Online]. Available: https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58. [Cit. 01 050 2020].

[8] „Netguru,“ [Online]. Available: https://www.netguru.com/blog/top-django-apps. [Cit. 01 05 2020].

[9] „Wikipedia,“ [Online]. Available: https://en.wikipedia.org/wiki/Progressive_web_application. [Cit. 2020 04 2020].

[10] „bart.sk,“ [Online]. Available: https://blog.bart.sk/uvod-vue-js/. [Cit. 01 05 2020].

[11] „Magazín KPI,“ [Online]. Available: https://magazin.kpi.fei.tuke.sk/2019/02/docker-a-jeho-pouzitie-pri-kontajnerizacii/. [Cit. 30 04 2020].

[12] „vsupalov,“ [Online]. Available: https://vsupalov.com/do-i-need-to-combine-django-with-vue-or-react/. [Cit. 30 04 2020].

[13] „maruti techlabs,“ [Online]. Available: https://marutitech.com/5-challenges-in-web-application-development/. [Cit. 02 05 2020].

[14] „mrc,“ [Online]. Available: https://www.mrc-productivity.com/blog/2020/04/7-web-application-development-challenges-you-cant-ignore/. [Cit. 02 05 2020].

[15] „Django REST framework,“ 03 05 2020. [Online]. Available: https://www.django-rest-framework.org/api-guide/routers/.