České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Jaroslav Bambas

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: Analýza a implementace software pro organizaci tréninků

Pokyny pro vypracování:

Cílem práce je vytvoření software pro organizaci tréninků, nabízení časů, přihlašování účastníků a evidenci jejich účasti.

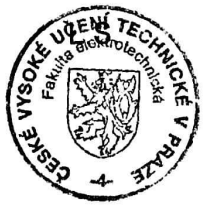
- proveďte sběr a specifikaci požadavků
- proveďte analýzu existujících software pro podobné účely
- analyzujte dostupné technologie odpovídající specifikovaným požadavkům a navrhněte vhodné technologie pro implementaci
- navrhněte architekturu aplikace a adekvátní testovací scénáře
- implementujte navržené řešení pomocí vybraných technologií a vyhodnoťte splnění jednotlivých požadavků dle jejich specifikace
- proveďte uživatelské testy podle definovaných scénářů, otestujte kompatibilitu uživatelského rozhraní s vybranými internetovými prohlížeči

Seznam odborné literatury:

[1] UML 2 a unifikovaný proces vývoje aplikací - Ila Neustadt, Jim Arlow
[2] Elisabetta Zibetti, Aline Chevalier, Robin Eyraud, What type of information displayed on digital scheduling software facilitates reflective planning tasks for students? Contributions to the design of a school task management tool, Computers in Human Behavior, Volume 28, Issue 2, March 2012, Pages 591-607, ISSN 0747-5632
[3] Bart van den Hooff, Electronic coordination and collective action: use and effects of electronic calendaring and scheduling, Information & Management, Volume 42, Issue 1, December 2004, Pages 103-114, ISSN 0378-7206

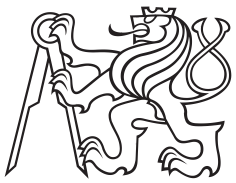
Vedoucí: Ing. Pavel Šedek

Platnost zadání do konce letního semestru 2017/2018



V Praze dne 7.3.2017

Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Analýza a implementace software pro organizaci tréninků

Bc. Jaroslav Bambas
Otevřená informatika - Softwarové inženýrství

Květen 2017
Vedoucí práce: Ing. Pavel Šedek

Poděkování / Prohlášení

Tímto bych chtěl poděkovat vedoucímu diplomové práce, panu Ing. Pavlu Šedkovi, za cenné rady a připomínky týkající se výběru technologií a literárních zdrojů. Dále bych chtěl poděkovat své partnerce a rodině za jejich podporu během celé doby studia.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných pracích.

V Praze, dne 22.5.2017

Abstrakt / Abstract

Tato závěrečná práce popisuje proces tvorby webové aplikace umožňující organizaci tréninků a přihlašování na události za pomoci Open Source technologií. Hlavním cílem práce je provedení sběru a analýzy požadavků na software, analýza existujících řešení a poskytovaných funkcí, návrh architektury a implementace webové aplikace pro podporu konání tréninkových akcí. Práce je zakončena reportem z procesu jednotkového a uživatelského testování aplikace a provedením zhodnocení splnění specifikace softwarového produktu.

Klíčová slova: Organizace událostí, Webová aplikace, Elektronický kalendář, Grails, Groovy, Marionette, Backbone

This diploma thesis describes process of creating web application that allows training organization and logging to events. The web application is based on Open Source technologies. The main goal of the thesis is to collect and analyze requirements to software product, analyze existing solutions and their functionalities. The next step is to perform the suggestion of solution and web application implementation. The thesis ends with report from application unit and user testing and evaluation of the software product specification.

Keywords: Event organization, Web application, Electronic Calendar, Grails, Groovy, Marionette, Backbone

Title translation: Software for workout management - analysis and implementation

Obsah /

1 Úvod 1
1.1 Motivace a cíle práce 1
1.2 Zadavatelé 1
1.3 Struktura práce 1
2 Analýza existujících řešení 3
2.1 Úvod do problematiky 3
2.2 Způsoby prezentace informací 5
2.2.1 Seznam událostí 6
2.2.2 Detail události 7
2.2.3 Shrnutí 8
2.3 Existující řešení 8
2.3.1 Funkce existujících řešení 9
2.3.2 Nabízené podmínky existujících řešení 10
3 Analýza a sběr požadavků 13
3.1 Funkční požadavky 13
3.1.1 F1 - Registrace uživatelů 14
3.1.2 F2 - Obnova zapomenutého hesla 14
3.1.3 F3 - Autentizace a autorizace 14
3.1.4 F4 - Vytváření a editace události 14
3.1.5 F5 - Zrušení události 15
3.1.6 F6 - Zasílání emailů 15
3.1.7 F7 - Zaznamenávání docházky 15
3.1.8 F8 - Zobrazení docházky 15
3.1.9 F9 - Přihlašování a odhlašování z událostí 16
3.1.10 F10 - Zobrazení přihlášených uživatelů 16
3.1.11 F11 - Zobrazení vytvořených událostí 16
3.1.12 F12 - Zobrazení přihlášených událostí uživatele 16
3.1.13 F13 - Vyhledávání a filtrování 16
3.1.14 F14 - Administrace 17
3.2 Nefunkční požadavky 17
3.2.1 N1 - RESTful API 17
3.2.2 N2 - Česká lokalizace 17
3.2.3 N3 - Licenční požadavky 17
3.2.4 N4 - Responzivní uživatelské rozhraní 17
3.2.5 N5 - Zneplatnění uživatelských účtů 17
3.2.6 N6 - Konfigurace aplikace 18
3.2.7 N7 - Logování 18
3.2.8 N8 - Výkonnostní požadavky 18
3.3 Business pravidla 18
3.3.1 B1 - Plánování událostí 18
3.3.2 B2 - Přihlašování na události 18
3.3.3 B3 - Kontrola počtu přihlášených uživatelů 18
3.4 Případy užití 19
3.4.1 Aktéři 19
3.4.2 Use cases 20
3.4.3 Mapování případů užití na funkční požadavky 23
4 Návrh architektury a výběr technologií 24
4.1 Výběr technologií 24
4.1.1 Serverová část 24
4.1.2 Klientská část 28
4.2 Doménový model 30
4.3 Architektura aplikace 32
4.3.1 Serverová část 33
4.3.2 Klientská část 35
4.4 Diagram nasazení 38
4.5 Licenční podmínky 38
4.5.1 Apache Licence 40
4.5.2 BSD Licence 40
4.5.3 GNU General Public Licence 40
4.5.4 MIT Licence 40
4.5.5 Použité frameworky, pluginy a knihovny 40
5 Implementace 42
5.1 Vývojové prostředí 42
5.2 Implementace backendu 42
5.2.1 Doménové třídy 42
5.2.2 Controller 43
5.2.3 View 45

5.2.4 Služby (Service) 45
5.2.5 Implementační problémy 45
5.3 Implementace frontendu 46
5.3.1 Backbone.Model 46
5.3.2 Backbone.Router 46
5.3.3 Marionette.View 47
5.3.4 Šablony 47
5.3.5 Uživatelské rozhraní 47
5.3.6 Implementační problémy 48
6 Testování 49
6.1 Jednotkové testy 49
6.1.1 Výsledky testování 49
6.2 Uživatelské testování 50
6.2.1 Testovací data 51
6.2.2 Testovací scénáře 51
6.2.3 Výsledky testování 55
6.3 Testování výkonnostních požadavků 56
6.4 Testování kompatibility prohlížečů 56
7 Zhodnocení splnění specifikace . 59
8 Závěr 61
8.1 Budoucí práce 61
Literatura 62
A Zkratky 65
B Instalační příručka 66
B.1 Vývoj a build aplikace 66
B.2 Deployment war souboru 66
C Obsah přiloženého CD 68

Tabulky / Obrázky

3.1. Mapování případů užití na funkční požadavky 23
4.1. Popis endpointu User 34
4.2. Popis endpointu Record 34
4.3. Popis endpointu Event 34
4.4. Popis endpointu Label 34
4.5. Popis endpointu Room 35
4.6. Popis endpointu Place 35
4.7. Popis endpointu pro autentizaci 35
4.8. Mapování URL Backbone.Router 36
4.9. Mapování Marionette.View na URL 37
4.10. Licenční podmínky použitých komponent 41
5.1. Zabezpečení controllerů 44
5.2. Šablony pro odesílání emailů .. 45
6.1. Report Unit testů 50
6.2. Účastníci uživatelského testování 50
6.3. Konfigurace testovacího serveru 56

2.1. Timeline 6
2.2. Timetable 7
3.1. Funkční požadavky 13
3.2. Nefunkční požadavky - UML .. 17
3.3. Business pravidla - UML 18
3.4. Případy užití - UML 19
4.1. Struktura Grails 25
4.2. Dynamic Finders 28
4.3. Doménový model - UML 31
4.4. Stavový UML diagram entity Event 32
4.5. Diagram nasazení - UML 38
6.1. Kompatibilita s prohlížeči 57

Kapitola 1
Úvod

Pro zajištění optimálního využívání místností při pořádání a plánování akcí je potřeba využívat nástrojů podporující koordinaci jednotlivých událostí. Komplexnost těchto nástrojů může být od primitivního řešení, jakým je například ručně psaný deník, až po ERP informační systémy umožňující například prodeje vstupenek. Tyto nástroje pomohou předcházet možným konfliktům při obsazování místností, překročení kapacity možných účastníků akce, překrývajícím se přihláškám uživatelů a dalším problémům, které mohou nastat při plánování a rozvrhování.

Důležitým prvkem v konání událostí je také poskytnout účastníkům události pohodlnou možnost k přihlášení na událost. Pokud je potenciálnímu účastníkovi umožněno zobrazit informace o všech událostech, na které se může potenciálně přihlásit, je možné dosáhnout většího počtu účastníků na událostech.

1.1 Motivace a cíle práce

Zadaná práce je navržena pro budoucí praktické využití zadavateli, viz 1.2. Zadavatel v současné době tráví velmi času režijní činností okolo organizace tréninků a dalších událostí. Očekávaným přínosem je snížení časové náročnosti těchto režijních činností a tím usnadnění organizace, jak po stránce časové, tak co se týká chyb v docházce, plánování a obsazenosti tréninků.

Hlavním této cílem práce je vytvoření software umožňující organizaci tréninků a událostí, přihlašování účastníků na tyto akce a evidenci jejich účasti. Software bude vytvořen na základě sběru a specifikace požadavků, získaných při schůzkách s vedoucím této závěrečné práce a zadavatelem. Součástí návrhu softwaru bude i analýza možností na budoucí vytvoření mobilní aplikace jako doplněk k implementovanému softwaru. Vývoj mobilní aplikace již není obsažen v cílech této práce.

1.2 Zadavatelé

Zadavatelé projektu jsou Mgr. Adam Zdobinský a PhDr. Radim Pavelka, Ph.D. z Katedry technických a úpolových sportů Fakulty tělesné výchovy a sportu na Univerzitě Karlova. Implementovaný software chtějí využívat pro podporu výuky a konání kurzů sebeobrany.

1.3 Struktura práce

Tato práce je strukturována na následujících 5 částí:

1. Analýza existujících řešení rezervačních a organizačních systémů, jejich možné využití pro potřeby zadavatele. Tato část je popsána v kapitole 2.

2. Provedení sběru a specifikace požadavků na software pro organizaci tréninků, viz. kapitola 3.
3. Navržení architektury aplikace a výběr technologií pro serverovou a klientskou část splňující požadované licenční podmínky, popsáno v kapitole 4.
4. Popis procesu implementace navrženého řešení z předchozí kapitoly 4, popsáno v kapitole 5.
5. Otestování aplikace a uživatelského rozhraní pomocí uživatelského a jednotkového testování, zhodnocení splnění specifikace na software z kapitoly 3. Touto částí se zabývají kapitoly 6 a 7.

Kapitola 2
Analýza existujících řešení

Následující kapitola se zabývá úvodem do problematiky aplikací sloužících k organizaci událostí, analýze existujících řešení a jejich možného použití pro účely zadavatele.

2.1 Úvod do problematiky

První část této kapitoly je věnována problematice rezervačních a organizačních systémů a benefitům spojených s využíváním těchto elektronických nástrojů.

Rezervační systémy, elektronické kalendáře nebo nástroje pro plánování událostí jsou softwarová řešení, která můžeme označovat jako ICT aplikace. Cílem těchto aplikací je podporovat koordinaci hromadných aktivit. Využití těchto nástrojů zastává důležitou roli v řízení organizací a umožňuje efektivní využití drahocenného času. ICT je velmi obecné označení pro informační technologie, používané pro komunikaci a práci s informacemi[1]. Článek [2] tyto nástroje označuje také jako ECS - Electronic Calendaring and Scheduling. Toto označení bude použito i dále v této práci. Charakteristickým znakem těchto nástrojů je umožnění uživatelům plánovat schůzky (události), rezervovat místnosti, případně si vytvářet poznámky k jednotlivým událostem [2] - například seznam účastníků a docházku přihlášených účastníků. Tyto řídící aktivity lze zaznamenávat i pomocí tradičních papírových poznámek a kalendářů. Hlavním problémem tohoto způsobu organizace je ovšem obtížnost sdílení informací o konání události mezi (potenciálními) účastníky - informace jsou centralizované u pořadatele. Použití elektronické organizace umožňuje odstranit tento problém a případnou ztrátu papírových materiálů. S.F.Ehrlich ve svém článku [3] podporuje pozorováním a měřením tvrzení, že ICT (ECS) nástroje zlepšují koordinaci aktivit a snižují časovou náročnost věnovanou řídícím činnostem. Metriky, na které se autor v článku zaměřuje, jsou zejména časové údaje spojené s režijní činností a možnosti sdílení informací mezi uživateli.

Literatura [4] rozlišuje tři dimenze pohledu na systém, které je nutné začlenit do studia ECS systémů.

1. individuální uživatel - osobní kalendář (události) uživatele
2. socio-organizační prostředí - skupinové funkcionality
3. technologie - technologie aplikace, funkcionality, snadnost použití

Dále jsou identifikovány interakce mezi těmito dimenzemi, které jsou představovány[4]:

1. kalendář pro jednoho uživatele - interakce mezi dimenzemi *individuální uživatel* a *technologie*
2. komunikace mezi uživateli - koordinace a interakce mezi dimenzemi *individuální uživatel* a *socio-organizační prostředí*

3. socio-technická evoluce - interakce mezi dimenzemi *socio-organizační prostředí* a *technologie*

První dvě popsané interakce lze popsat jako rozdílné pohledy na používání ECS[2]:

- *Individuální použití* - individuální kalendář uživatele a naplánované aktivity, kterých se účastní
- *Kolektivní použití* - koordinace událostí za účelem zabránění konfliktů termínů, seznam přihlášených účastníků

Tento obecný koncept je možné využít pro software sloužící podpoře organizaci tréninků. Na software pro podporu organizace událostí je možné pohlížet jako na integraci použití tří různých elektronických kalendářů a způsobů použití:

- *Individuální použití*
  - *Použití běžného individuálního uživatele* - individuální kalendář uživatele zobrazující přihlášené události, kterých se chce zúčastnit.
  - *Použití řídící organizace (trenéra)* - individuální kalendář trenéra obsahující vytvořené události tímto uživatelem
- *Kolektivní použití* - koordinace událostí za účelem zabránění konfliktů termínů, seznam přihlášených účastníků. Alternativní interpretace může být kalendář místnosti pro konání událostí.

Hlavní rozdíl proti předchozím způsobům použití popsaném v článku [2] je v rozdělení role individuálního uživatele na dvě. Způsoby použití systému uvedené v [2] předpokládají, že události mohou být vytvořené libovolným uživatelem (každý uživatel má možnost vytvořit událost). V případě nástroje pro podporu organizace tréninků je nutné rozlišovat uživatele, kteří mají právo vytvářet nové události (trenéři) a uživatele, kteří se na vypsané události přihlašují. Zkráceně, software pro podporu organizace tréninkových událostí je reprezentován systémem, který umožňuje integraci 3 elektronických kalendářů a způsobů jejich použití - kalendář uživatele s přihlášenými událostmi, kalendář trenéra s vytvořenými událostmi a kalendář místnosti popisující volné a obsazené termíny událostmi.

Dále práce [2] uvádí celkem 12 hypotéz týkajících se pozitivního ovlivňování výkonu práce při kolektivním a individuálním používání ECS systému. Hypotézy jsou následně empiricky ověřeny v praxi při plánování aktivit uvnitř reálné organizace. Těchto 12 hypotéz lze aplikovat na software pro podporu organizace tréninků, jakožto na ECS systém. Ověřované hypotézy jsou následující[2]:

1. **H1** - Kolektivní použití ECS má pozitivní vliv na koordinaci - ve smyslu zjednodušení pořádání událostí se spolupracovníky (trenéři, účastníci událostí).
2. **H2** - Kolektivní použití ECS má pozitivní vliv na koordinaci - ve smyslu lepší informovanosti spolupracovníků o termínu konání a umístění.
3. **H3** - Učenlivost (adaptace na používání) uživatele ECS systému má pozitivní vliv na individuální a kolektivní používání.
4. **H4** - Sdílení uživatelských aktivit má pozitivní vliv na možnosti kolektivního použití ECS.

5. **H5** - Zkušenost uživatele s používáním ECS systému má pozitivní vliv na individuální a kolektivní používání.
6. **H6** - Individuální způsob používání ECS příznivě ovlivňuje koordinaci událostí a jejich termínů (kolektivní používání ECS).
7. **H7** - Jednoduchost používání ECS má pozitivní vliv na individuální a kolektivní práci (čím jednodušší ovládání, tím větší vliv benefitů z používání ECS).
8. **H8** - Počet naplánovaných událostí má vztah k individuální a kolektivní práci (čím více událostí je organizováno skrze ECS, tím větší je příznivý vliv ECS na fungování organizace).
9. **H9** - Frekvence, se kterou uživatel vytváří události, má pozitivní vliv na individuální a kolektivní používání (souvislost s hypotézou H8).
10. **H10** - Úroveň vnější orientace uživatelských událostí má pozitivní vliv na individuální a kolektivní užívání ECS - vnější orientací je myšleno, zda se jedná o událost jednotlivce (soukromá) nebo kolektivu (tréninková událost pro více osob)
11. **H11** - Rozsah používání ECS spolupracovníky ovlivňuje individuální a kolektivní používání uživatele (čím více uživatelů v organizaci využívá ECS, tím větší je jeho pozitivní vliv).
12. **H12** - Tlak spolupracovníků a nadřízených na využívání ECS systému pozitivně ovlivňuje individuální a kolektivní způsob použití ECS.

Výsledky měření a pozorování plně podpořilo správnost hypotéz **H1**, **H2**, **H3**, **H4**, **H6**, **H7**, **H8**, **H9**, **H10** a **H12**. Platnost hypotézy **H5** byla podpořena pouze částečně. Zkušenost uživatele s používáním ECS systému má pozitivní vliv pouze na individuální způsob používání a neovlivňuje koordinaci událostí (kolektivní používání)[2]. Poslední zbývající hypotéza **H11** nebyla podpořena. Důvodem je nenalezení korelace mezi počtem spolupracovníků používajících ECS a zlepšením úrovně individuálního používání uživatelem nebo kolektivního používání (zabránění konfliktům při plánování, obsazenost místností, seznamy účastníků). Po implementaci a nasazení ECS systému sloužící k podpoře organizování sportovních událostí je očekáváno, že se pozitivně projeví benefity popsané podpořenými hypotézami.

2.2 Způsoby prezentace informací

Druhá část této kapitoly je věnována možným způsobům zobrazení informací týkajících se plánování událostí a jejich použití v ECS nástrojích. Způsob prezentace informací má přímý dopad na výsledné uživatelské prostředí a pohodlnost ovládání (hypotézy H3 a H7, viz. předchozí podkapitola 2.1). Nejméně tři aspekty prezentace informací mají významný vliv na kognitivní vnímání: styl prezentace (včetně způsobu), načasování prezentace a množství obsahu[5]. Autoři práce [6] se zaměřují na tyto aspekty u školního systému pro plánování studentských úkolů a událostí.

Výsledky této práce poukazují na fakt, že snížení počtu prezentovaných informačních elementů o události může mít pro uživatele pozitivní efekt na plánování událostí a rozvrhování (snížení potřebného času na vytvoření rozvrhu a nalezení požadované informace - např. termín konání, místo konání). Následující sekce této kapitoly se zabývají možnými způsoby prezentace:

- seznamu událostí (např. nabídka událostí k přihlašování, přihlášené události uživatele), viz. 2.2.1
- detailních informací o vypsané události, viz. 2.2.2

2.2.1 Seznam událostí

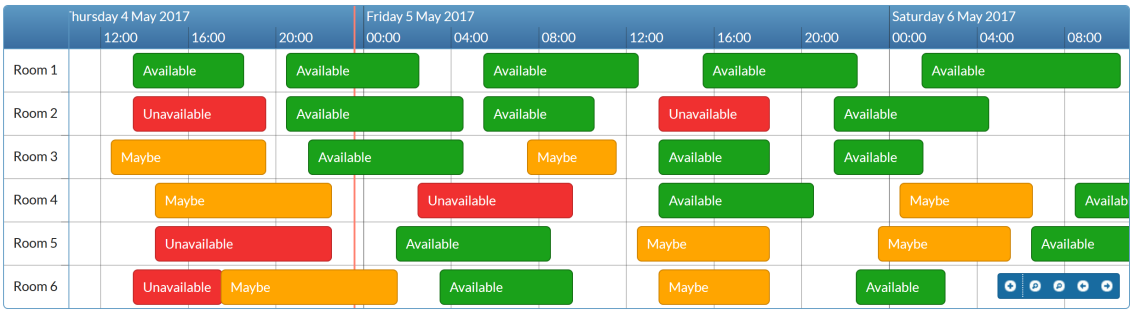
Prezentace seznamu konaných událostí umožňuje uživateli základní přehled o událostech a jejich termínů konání. Práce [6] pro účely testování vzorové aplikace používá pro výpis informací o seznamu událostí tyto atributy úkolu: název úkolu, jméno předmětu a termín pro dokončení úkolu. Pro prezentaci informací o množině vypsaných událostech (úkolů) lze použít různé elementy uživatelského rozhraní. Tyto možné způsoby prezentace dat jsou předmětem zájmu této sekce.

- **Tabulkový výpis**

  Tabulkový výpis úkolů (událostí) je reprezentován tabulkou, kde jeden naplánovaný úkol (událost) představuje řádek této tabulky. Skrze řádek tabulky je umožněno zobrazit doplňující atributy, které identifikují úkol (událost). Popis způsobů prezentace těchto atributů je obsahem sekce 2.2.2. Výhodou tabulkového výpisu je možnost integrace s filtrovacím formulářem, s možností řadit záznamy v tabulce a se stránkováním záznamů.

- **Timeline**

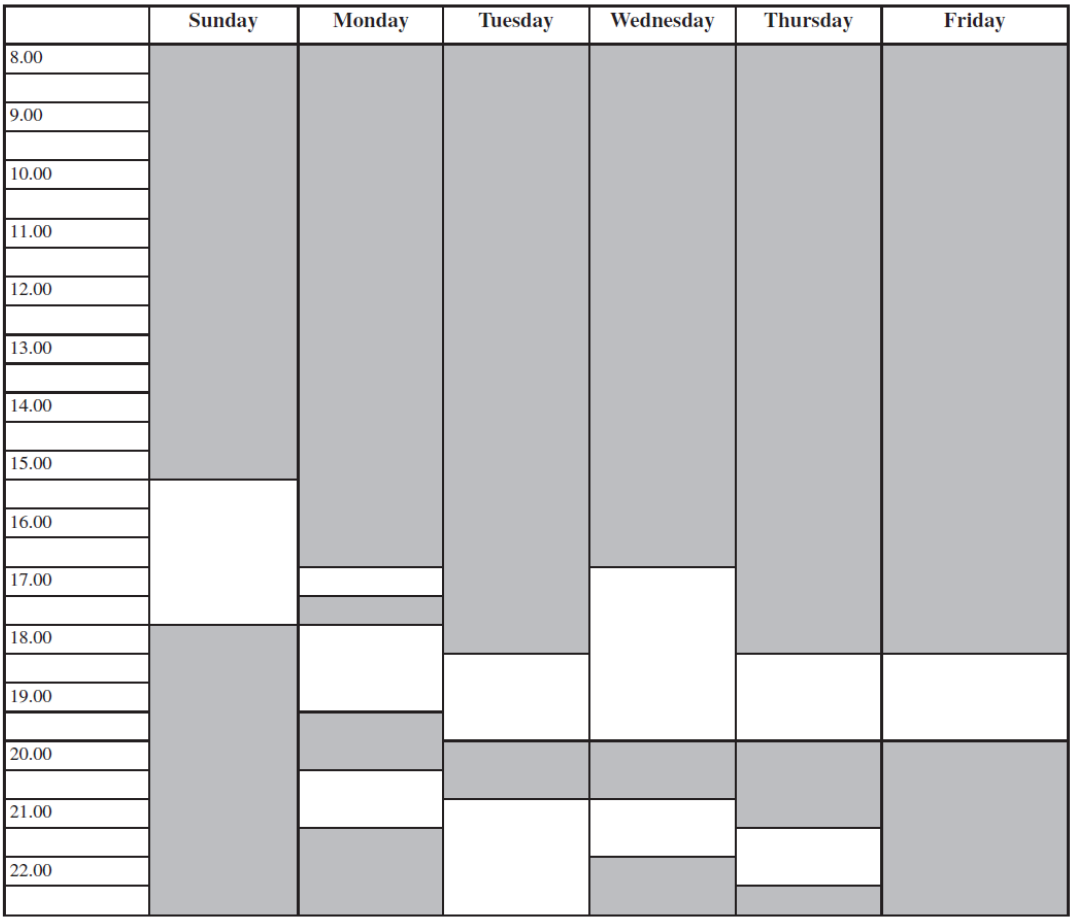
  Timeline, neboli časová osa, je prvek vhodný pro zobrazení obsazenosti místnosti a zobrazení volných termínů. Jedná se o interaktivní vizualizaci časového plánu. Jednotlivé události jsou zaneseny do časové osy dle času konání události. Ukázka zobrazení naplánovaných událostí pomocí Timeline je znázorněna na obrázku 2.1. Možnou vlastností prvku Timeline je změna zobrazeného časového úseku pomocí scrollování (efekt zoomu) - od řádu minut až po měsíce [7]. Existující knihovny pro prezentaci událostí pomocí prvku Timeline na webu jsou například vis.js - Timeline nebo PrimeFaces Showcase Timeline.



Obrázek 2.1. Příklad elementu Timeline, převzato z [8]

- **Timetable**

  Timetable (rozvrh), je způsob prezentace naplánovaných úkolů a aktivit uživatele během dne v daném časovém období [6]. Ukázka způsobu prezentace událostí uživatele je zobrazena na obrázku 2.2. Tento způsob prezentace informací je možné popsat jako kalendář, obsahující údaje o termínu konání naplánovaných událostí (aktivit) uživatele. Události jsou v jednotlivých dnech zobrazeny obdobným způsobem



Obrázek 2.2. Příklad elementu Timetable pro zobrazení týdenního seznamu událostí, převzato z [6]

jako u Timeline. Interval zobrazovaných dnů rozvrhu může být představován měsíčním nebo týdenním kalendářem. Možný způsob implementace této formy prezentace informací pro využití na webové stránce je pomocí HTML elementu `table`.

2.2.2 Detail události

Součástí práce [6] je porovnání tří způsobů prezentace detailních informací o zadaných úkolech. Detailními informacemi jsou myšleny veškeré údaje o úkolu (případně události), včetně informací nezobrazených v prezentaci informací seznamu událostí. Mezi tyto údaje patří v rámci domácích úkolů například popis se zadáním úkolu. V rámci experimentální aplikace byly otestovány 3 způsoby prezentace informací o zadaném úkolu, kdy byly některé atributy o události přesunuty do externího elementu uživatelského rozhraní [6]. Těmito externími elementy byly:

- Tooltip zobrazující se po najetí myši na element představující úkol v seznamu - varianta **G1**
- Pop-up okno zobrazující se po kliknutí na element představující úkol v seznamu - varianta **G2**
- Prezentace všech záznamů o události jako součást prezentace seznamu úkolů - varianta **G3**

Uživatelské testy zkoumaly průměrnou dobu odezvy uživatelů, průměrný počet úspěšně dokončených problémů, průměrný čas potřebný k vyřešení zadaného úkolu a průměrný čas strávený na čekání zobrazení vyskakovacích oken. V testech dosáhly nejlepších výsledků varianty **G2** a **G3**.

Další možností pro zobrazení detailních informací o úkolu (události), která není v práci [6] popsána a implementovaná v experimentální aplikaci, je pomocí samostatné obrazovky.

- **Obrazovka detailu události**

  Veškeré informace o události (úkolu), jsou zobrazeny na samostatné obrazovce. Výhodou tohoto řešení v případě použití u webové aplikace je možnost sdílení s ostatními uživateli skrze URL adresu (odkaz).

- **Pop-up okno**

  Pop-up okno, případně modální okno, představuje vizuální element grafického uživatelského rozhraní, který překrývá původní obsah stránky [9]. Pop-up okno je otevřeno po kliknutí na element představující položku v prezentaci seznamu událostí. Termín pop-up okno je poměrně obecný a může představovat různé elementy uživatelského rozhraní, například dialogová nebo modální okna. Obsah modálního okna následně může být shodný se zobrazovanými údaji v předchozím bodu - Obrazovka detailu události.

- **Tooltip**

  Tooltip je pomocný element uživatelského rozhraní, sloužící k zobrazení významů zkratek a dalších doplňujících informací o elementu uživatelského rozhraní (například odkazu nebo tlačítka). Tooltip je zobrazený po najetí kurzoru na daný element. Nevýhodou použití tooltipu pro prezentaci detailních informací o události je nutné čekání uživatele po najetí kurzoru na element, než se tooltip zobrazí. Další nevýhodou je pouze omezené použití - slouží pouze k zobrazení doplňujících informací. Obsah tooltipu nemůže nést formulářové prvky a odkazy.

2.2.3 Shrnutí

Cílem předchozích sekcí 2.2.1 a 2.2.2 bylo provedení analýzy způsobů prezentace informací uživateli skrze elementy GUI. Od provedení analýzy způsobů prezentace informací je očekáván návrh a implementace přívětivějšího GUI umožňující pohodlné ovládání implementované aplikace. Článek [6] uvádí tvrzení, že snížení množství prezentovaných informací uživateli může mít pozitivní vliv na kognitivní vnímání uživatelů. Stále je však nutné umožnit uživateli zobrazit veškeré informace. Pro zajištění tohoto požadavku je vhodným řešením přesun části informací do externího elementu, který je zobrazen po vybrání konkrétní události z prezentovaného seznamu událostí.

2.3 Existující řešení

Třetí část této kapitoly je věnována analýze existujících řešení v podobě komerčních rezervačních systémů. Cílem analýzy je zjištění, zda některé existující řešení odpovídá požadavkům a potřebám zadavatele, popsaných v následující kapitole 3.

2.3.1 Funkce existujících řešení

Následující část kapitoly je věnována popisu nabízených funkcí, které poskytují existující řešení. Tato sekce je v práci zahrnuta z důvodu výzkumu existujících funkcí a způsobů prezentace dat uživateli používaných v již existujících řešení ověřených v praxi. Tyto poznatky následně budou moci být využiti při návrhu a implementaci vlastního řešení dle požadavků zadavatele. Existující služby a řešení použité k analýze (včetně webových odkazů na tyto služby) jsou součástí sekce 2.3.2. Poznatky o existujících řešení jsou čerpány z dostupných demo aplikací jednotlivých služeb, jejich ceníků a z dokumentace popisující vlastnosti a funkce dané varianty produktu.

Nalezené komerční nabídky umožňují zabezpečenou komunikaci skrze HTTPS protokol a zajištění hostingu pro běh webové aplikace. V případě Open Source řešení je nutné zajištění běhového prostředí a SSL certifikátu - produkt je distribuován pouze v podobě zdrojových kódů bez podpory.

Existující řešení poskytují uživatelům několik způsobů prezentace informací o seznamu událostí. Popis konkrétních použitých způsobů prezentace je předmětem předchozí části 2.2.1. Případy užití prezentace informací o seznamu událostí je možné rozdělit na následující tři skupiny:

- **Vypsané události k přihlášení/události konané v konkrétní místnosti**

  Představuje katalog vypsaných událostí, na které je možné se přihlásit. V existujících nástrojích jsou využity způsoby prezentace pomocí tabulkového výpisu, který je možné filtrovat a určit časové rozpětí zobrazovaných událostí (určení termínu začátku a konce od, do). Druhým nalezeným řešením je prezentace dat pomocí rozvrhu (timetable).

- **Nabídka volných termínů**

  Při vytváření nové události je nutné, aby aplikace uživateli poskytovala způsob pro zjištění neobsazených termínů konkrétní místnosti. Jediným způsobem, který je v analyzovaných řešení používaný, je zobrazení vypsaných událostí na časové ose (timeline).

- **Přihlášené události uživatele/vytvořené události trenéra**

  Pro zobrazení přehledu přihlášených událostí uživatele, případně vytvořených událostí organizátora (trenéra), nabízí existující řešení všechny způsoby prezentace popsané v sekci 2.2.1. Existující řešení v tomto nejsou jednotná - každé řešení používá z popsaných způsobů určitou podmnožinu, případně je umožněno mezi těmito různými formami prezentace informací přepínat, čímž se doplňují. Používány jsou všechny způsoby - časová osa (timeline), rozvrh (timetable) i tabulkový výpis.

Popis konkrétních použitých způsobů prezentace detailních informací o události je předmětem předchozí části 2.2.2. Prezentace informací je v existujících řešeních realizována následujícími způsoby:

- **Součást prezentace seznamu událostí**

  Uživateli jsou poskytnuty všechny údaje o události, včetně možnosti pro přihlášení na událost, jako součást prezentace seznamu událostí.

- **Samostatná stránka**

- **Pop-up okno (modálního okno)**

Nalezené způsoby prezentace informací odpovídají způsobům popsaných v kapitolách 2.2.1 a 2.2.2. Další formy prezentace nebyly v existujících řešení nalezeny. Naopak zobrazení detailních informací formou tooltipu nebylo u žádných nalezených existujících řešení nalezeno.

Pro komerční organizace jsou součástí placených variant funkce pro podporu jejich podnikání v podobě napojení rezervací místností na platební bránu, správa provedených plateb a účetnictví, možnost permanentních karet pro klienty nebo slevové klientské programy. Tyto funkce nejsou potřebné pro plánované využití zadavateli - nasazení aplikace pro usnadnění organizace tréninků v akademické sféře.

Velké množství existujících řešení umožňuje posílání notifikací uživatelům pomocí zasílání emailů nebo SMS zpráv. Tato funkce je často závislá na placené variantě produktu. Notifikace se mohou být zasílány uživatelům v následujících případech:

- potvrzení přihlášky na událost
- změna atributů události (termín a místo konání)
- zrušení události
- upozornění na událost před začátek akce přihlášeným uživatelům

Další prémiovou funkcí některých existujících řešení je možnost synchronizace s externími ECS službami - například Google kalendář, iCal nebo Outlook kalendář. Přihlášené události uživatele jsou importovány do jeho osobního elektronického kalendáře. Tato funkce není součástí všech nalezených řešení a to ani placených variant.

2.3.2 Nabízené podmínky existujících řešení

Na trhu existuje mnoho softwarových řešení v podobě rezervačních a kalendářních systémů. Při hledání optimálního řešení podle požadavků zadavatele, viz. kapitola 3, bylo nalezeno 6 možných komerčních řešení a 1 open source řešení. Každé z těchto řešení nabízí různé varianty, které se liší úrovní nabízených funkcionalit. Z důvodu požadavku na bezplatné provozování jsou zde popsány klady a zápory nejméně finančně náročné varianty daného produktu. Výhody a nevýhody jednotlivých řešení jsou čerpány z nabídek platných ke dni 5.5.2017.

- **Bizzy**[1]

Klady	Zápory
česká lokalizace	placená služba - od 600,- Kč měsíčně
statistiky událostí (docházka)	omezený počet míst pro konání událostí na 6
responzivní uživatelské rozhraní	
správa uživatelů a uživatelských skupin	

- **SuperSaaS**[2]

Klady	Zápory
česká lokalizace	umístěné reklamy v uživatelském rozhraní
možnost bezplatného provozu	omezený počet uživatelů na 50
responzivní uživatelské rozhraní	omezený počet rezervací na měsíc na 50
správa uživatelů a uživatelských skupin	

[1] http://www.e-rezervace.cz/
[2] http://www.supersaas.cz/

- **Reservanto**[1]

Klady	Zápory
česká lokalizace	chybějící emailové notifikace
možnost bezplatného provozu	chybějící statistiky událostí (ve free verzi)
bez omezení počtu událostí a uživatelů	chybějící správa uživatelských účtů
responzivní uživatelské rozhraní	

- **Reenio**[2]

Klady	Zápory
česká lokalizace	omezený počet rezervací za měsíc na 50
možnost bezplatného provozu	omezený počet míst pro konání událostí na 5
bez omezení počtu uživatelů	omezený počet účtů zaměstnanců (trenérů) na 5
responzivní uživatelské rozhraní	

- **TeamUp**[3]

Klady	Zápory
mobilní aplikace pro Android a iOS	chybějící česká lokalizace
možnost bezplatného provozu	chybějící správa uživatelů (ve free verzi)
bez omezení počtu uživatelů	
notifikace uživatelům před událostí	
synchronizace s Google, Outlook, iCal	

- **JdemeNaTo**[4]

Klady	Zápory
česká lokalizace	placená služba - jednorázově 7900,- Kč
bez omezení počtu rezervací	chybějící statistiky o účastnících (docházka)
bez omezení počtu událostí	
bez omezení počtu míst pro události	

- **Reservio**[5]

Klady	Zápory
česká lokalizace	omezení na 40 rezervací za měsíc
možnost bezplatného provozu	zobrazované reklamy součástí GUI
synchronizace s Google, Outlook, iCal	pouze 1 zaměstnanecký (trenérský) účet

- **Easy!Appointments**[6]

Klady	Zápory
Open source řešení	chybějící česká lokalizace
možnost komerčního použití	zajištění běhového prostředí (doména, server)
modifikace konkrétním potřebám	

Všechny výše uvedené produkty umožňují pořádání hromadných událostí. Tato fukcionalita je nutná pro pořádání tréninkových událostí a následné přihlašování jednotlivých uživatelů. Dalšími službami, sloužící k organizování sportovních událostí, jsou

[1] https://www.reservanto.cz/
[2] https://reenio.cz/
[3] http://www.teamup.com/
[4] http://rezervacnisystem.jdemenato.cz/
[5] https://www.reservio.com/
[6] http://easyappointments.org/

například Rezervačník[1] nebo Calendly[2]. Problémem těchto produktů v bezplatné variantě je absence pořádání hromadných událostí. Tyto služby jsou vhodné pouze pro rezervaci konkrétního sportoviště jednou osobou.

Pro potřeby zadavatele není vhodná žádná z výše popsaných variant existujícího řešení. Řešení, která by odpovídala požadavkům zadavatele jsou zpoplatněná a řešení umožňující bezplatné provozování mají následující nedostatky:

- omezení počtu uživatelů a jejich rezervací (přihlášek) za měsíc
- omezení počtu událostí za měsíc
- omezení počtu umístění pro konání událostí
- chybějící česká lokalizace
- vkládání reklam do uživatelského rozhraní

[1] http://www.rezervacnik.cz/
[2] https://calendly.com/

Kapitola 3
Analýza a sběr požadavků

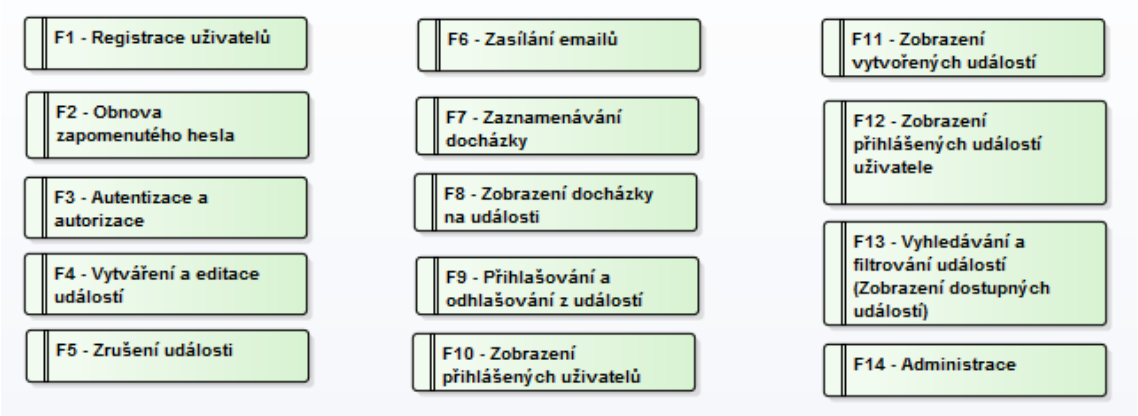
Analýza a sběr požadavků na software byl prováděn v průběhu pravidelných schůzek s vedoucím této práce a zadavatelem. Sběr požadavků probíhal formou rozhovoru o potřebách a očekáváních na požadovaný software.

Získané požadavky jsou rozděleny do tří kategorií.

1. **Funkční požadavky** - definují nezbytné úkoly nebo akce, které musí aplikace nezbytně vykonávat [10], v následujících sekcích jsou označeny *Fn*, kde *n* je číslo funkčního požadavku.
2. **Nefunkční požadavky** - definují požadavky na architekturu aplikace, uživatelské rozhraní a prostředí pro běh aplikace, v následujících sekcích jsou označeny *Nn*, kde *n* je číslo nefunkčního požadavku.
3. **Business požadavky** - definují pravidla a zásady, která jsou klíčová pro dané business odvětví či konkrétní podnikání, business pravidla definují nebo omezují určitý aspekt dané domény [11]. Literatura [12] uvádí business požadavky jako součást nefunkčních požadavků. V následujících sekcích jsou označeny *Bn*, kde *n* je číslo business požadavku.

3.1 Funkční požadavky

Celkem bylo identifikováno 14 hlavních funkčních požadavků. Seznam požadavků je zobrazen na obrázku 3.1. Podrobný popis jednotlivých požadavků je v následujících podkapitolách.



Obrázek 3.1. Funkční požadavky

3.1.1 F1 - Registrace uživatelů

Systém bude umožňovat neautentizovaným uživatelům registraci nového uživatelského účtu. O uživateli budou zaznamenávány následující údaje: jméno, příjmení, unikátní username, heslo, email a telefonní číslo.

Heslo musí splňovat délku minimálně 8 znaků a musí obsahovat alespoň 3 zástupce z různých kategorií znaků - [a-z], [A-Z], [0-9] a nealfanumerické znaky. Telefonní číslo může obsahovat pouze znaky [0-9] a '+', jeho délka bude v rozmezí 9 až 15 znaků (z důvodu možného vložení mezinárodní předvolby před samotné telefonní číslo). Zadaný email odpovídá vzoru `xxx@xxx.xxx` a je délky v rozmezí 5-50 znaků.

Registraci účtu bude nutné potvrdit přistoupením na odkaz, který bude uživateli zaslán na emailovou adresu uvedenou v registračním formuláři. Platnost tohoto odkazu bude časově omezena na dobu 24 hodin od přijetí registračního formuláře. Po této době nebude již možné účet aktivovat. Takto registrovaný uživatel bude mít nastavená práva běžného uživatele. Systém nebude umožňovat neautentizovaným uživatelům registraci účtů s jinými právy než běžného uživatele. Uživatelské účty s právy trenérských a administrátorských účtů bude možné vytvořit pomocí administrátorského účtu a funkčního požadavku F14 - Administrace - 3.1.14.

3.1.2 F2 - Obnova zapomenutého hesla

Aplikace bude registrovanému uživateli umožňovat obnovu zapomenutého hesla. Na email uvedený v registraci bude uživateli po vložení emailové adresy zaslán odkaz na stránku umožňující zadání nového hesla. Platnost odkazu umožňující změnu hesla bude časově omezena na dobu 24 hodin od přijetí požadavku na obnovu hesla. Po uplynutí této doby již nebude možné pomocí tohoto odkazu změnit heslo a uživatel si bude muset vytvořit novou žádost o obnovu hesla.

3.1.3 F3 - Autentizace a autorizace

Aplikace bude umožňovat neautentizovanému uživateli přihlášení k aktivovanému účtu. Účtem, který není aktivován, nebude možné se přihlásit. Přihlašování bude realizováno zadáním svého unikátního username a hesla. Tyto údaje jsou zadány uživatelem v registračním formuláři požadavku F1 - 3.1.1.

3.1.4 F4 - Vytváření a editace události

Aplikace bude umožňovat přihlášeným uživatelům, autorizovaným jako trenér, vytvářet nové události. O každé události budou zaznamenány následující atributy: minimální a maximální počet účastníků, místo konání, termín konce přihlašování uživatelů na událost, začátek a konec akce. Aplikace bude umožňovat vytvářet jednodenní i několika denní události (termín začátku a konce události je v různé kalendářní dny). Každá událost bude mít možnost nastavení jednoho ze stavů:

- Nepřístupné přihlašování
- Odemčeno k přihlašování
- Přihlašování uzamčeno
- Proběhnutá událost

Dále bude aplikací umožněno vytvářet tzv. opakovatelné události v zadaném termínu od a do s možnostmi:

- Každý den
- Každý týden
- Každý sudý týden
- Každý lichý týden

Takto vytvořené události se následně chovají jako jednotlivé události - tzn. každá událost vystupuje pro přihlašování a editaci jako samostatně vytvořená událost. V případě, že nebude možné vytvořit některou z událostí v tomto časovém intervalu (např. z důvodu obsazenosti místnosti), nebudou vytvořeny ani žádné ostatní události. Termín konce přihlašování bude u opakovaných událostí defaultně nastaven na 24 hodin před jejím začátkem. Vytvořené opakovatelné události budou pouze jednodenní. V případě potřeby je možné změnit události na několika denní pomocí editace atributů události.

V případě změny (editace) atributů události (termín konání, místo konání), bude již přihlášeným uživatelům na událost odeslán email s informací o této změně na adresu uvedenou v jejich uživatelském účtu.

3.1.5 F5 - Zrušení události

Přihlášeným uživatelům, autorizovaným jako trenér bude, aplikace umožňovat zrušit svou vytvořenou událost. V případě, že na události budou již zapsaní někteří uživatelé, bude jim odeslán automatický email s informací o zrušení události. Rušit události bude možné vždy samostatně (po jedné), tzn. aplikace nebude umožňovat hromadné rušení událostí pomocí jednoho požadavku.

3.1.6 F6 - Zasílání emailů

Aplikace bude umožňovat odesílat uživatelům emaily v následujících případech:

- Potvrzení nové registrace
- Žádost o obnovu zapomenutého hesla
- Potvrzení o změně hesla
- Změny atributů události - informační email přihlášeným uživatelům na událost
- Zrušení události - informační email přihlášeným uživatelům na událost

3.1.7 F7 - Zaznamenávání docházky

Aplikace bude přihlášeným uživatelům, autorizovaným jako trenér, umožňovat zaznamenávat docházku uživatelů, kteří jsou přihlášeni na jejich vytvořené události. Ke každému přihlášenému uživateli na událost bude možno přiřadit jednu ze dvou možností - přítomen/nepřítomen. Zaznamenávání docházky bude umožněnou pouze u událostí, které již proběhly.

3.1.8 F8 - Zobrazení docházky

Přihlášeným uživatelům, autorizovaným jako trenér, bude aplikace zobrazovat údaje o docházce na svých událostech. Data o docházce budou reprezentována ve formě předvyplněného formuláře pro zaznamenávání docházky (funkční požadavek F7 - 3.1.7). Dále bude umožněno trenérům zobrazit docházku konkrétního uživatele na událostech, které vytvořil.

Běžným uživatelům, kteří se přihlašují na události, budou záznamy o docházce přístupny v rámci zobrazení přehledu událostí, na které byli přihlášeni. U každé proběhnuté události bude zobrazen záznam o docházce přihlášeného uživatele.

3.1.9 F9 - Přihlašování a odhlašování z událostí

Aplikace bude umožňovat přihlašování uživatelů na vypsané události. Přihlašování na vytvořenou událost bude umožněno pouze autentizovaným uživatelům, kteří nemají práva na editaci události (tzn. kromě administrátorských účtů a trenéra, který událost vytvořil).

Po přihlášení bude mít uživatel možnost se z události odhlásit. Možnost přihlašování a odhlašování událostí bude přístupná pouze na události, které mají nastavený stav *Přihlašování odemčeno* a ještě neproběhl termín na přihlašování/odhlašování z události (časový termín nastavený při vytváření události - 3.1.4). Po tomto termínu již nebude uživateli umožněno přihlášku měnit.

Aplikace bude umožňovat autorovi události přihlásit na událost jiné uživatele, případně již přihlášeným uživatelům přihlášku zrušit.

3.1.10 F10 - Zobrazení přihlášených uživatelů

Aplikace bude umožňovat uživateli, který vytvořil událost, zobrazit seznam aktuálně přihlášených uživatelů na jeho události. Pro každého přihlášeného uživatele bude zobrazeno jeho jméno, příjmení, username a odkaz na detail jeho účtu obsahující kontaktní údaje (telefonní číslo a email).

Aplikace nebude optimalizovat formát tabulky s přihlášenými uživateli k tisku.

3.1.11 F11 - Zobrazení vytvořených událostí

Aplikace bude umožňovat přihlášeným uživatelům autorizovaným jako trenér zobrazit přehled všech svých v minulosti vytvořených událostí. V tomto přehledu bude možno události třídit a vyhledávat dle data konání (od, do), typu události a místa konání události. Z tohoto zobrazení bude možné přejít do sekce editace údajů konkrétní události se sekcí zobrazení přihlášených uživatelů na událost.

3.1.12 F12 - Zobrazení přihlášených událostí uživatele

Autentizovanému uživateli bude aplikace umožňovat zobrazit seznam událostí, na které se v minulosti přihlásil. Události, na které se v minulosti přihlásil a následně odhlásil, v tomto seznamu nebudou uvedeny. V seznamu přihlášených událostí budou také zobrazeny údaje o docházce.

3.1.13 F13 - Vyhledávání a filtrování

Uživatelům bude aplikace umožňovat vyhledávání ve vypsaných událostech. Filtrování událostí bude možné pomocí následujících atributů: termín konání, trenér (uživatel) který vytvořil událost, kategorie (trénink, seminář, atd.), místo konání a stav události. Filtrovat záznamy bude možné pomocí několika atributů najednou, např. podle termínu konání a trenéra zároveň. Záznamy budou stránkovány po 20 záznamech na stránku.

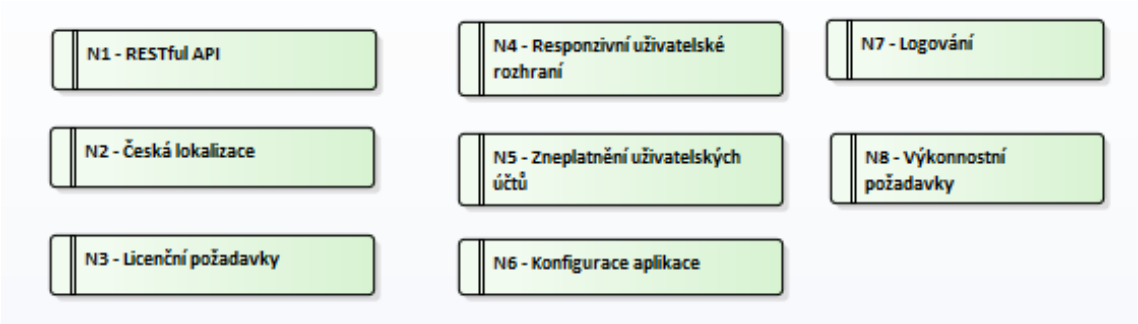
Dále bude možné filtrovat a vyhledávat pomocí administrátorského účtu v umístěních (podle názvu umístění a adresy), uživatelských účtech (pomocí jména, příjmení, uživatelského jména, emailu a telefonního čísla) a kategoriích (podle názvu kategorie).

3.1.14 F14 - Administrace

Aplikace bude umožňovat administraci aplikace pomocí administrátorského uživatelského účtu. Administrací se rozumí vytváření, editace, prohlížení a mazání záznamů uložených v databázi, neboli veškeré CRUD operace.

3.2 Nefunkční požadavky

Celkem bylo identifikováno 7 nefunkčních požadavků. Seznam požadavků je zobrazen na obrázku 3.2. Podrobný popis jednotlivých požadavků je v následujících podkapitolách.



Obrázek 3.2. Nefunkční požadavky

3.2.1 N1 - RESTful API

Aplikace bude vystavovat RESTful API umožňující veškeré CRUD operace, včetně vyhledávání a filtrování záznamů. Přihlášený uživatel bude identifikován pomocí tokenu, který klientská část získá po úspěšném přihlášení. Klientská část bude reprezentována one page webovou stránkou, která bude komunikovat se serverovou částí pomocí tohoto REST API. Existence REST API bude umožňovat v budoucnu případné vytvoření mobilní aplikace.

Zabezpečení komunikace skrze API bude realizováno HTTPS protokolem. SSL certifikát bude zajištěn provozovatelem po implementaci a nasazení do produkčního prostředí.

3.2.2 N2 - Česká lokalizace

Veškeré uživatelské prostředí aplikace bude lokalizováno do českého jazyka.

3.2.3 N3 - Licenční požadavky

Veškeré použité komponenty systému budou splňovat licenční podmínky pro bezplatné vytvoření a následné provozování aplikace. S finanční náklady spojené s hostingem aplikace zadavatel akceptuje.

3.2.4 N4 - Responzivní uživatelské rozhraní

Aplikace bude nabízet responzivní uživatelské rozhraní přizpůsobené pro zobrazení na mobilních zařízeních i stolních počítačích. Uživatelské rozhraní bude kompatibilní s prohlížeči Mozilla Firefox verze 52, Chrome verze 56, Internet Explorer 11, Microsoft Edge a Safari na iOS.

3.2.5 N5 - Zneplatnění uživatelských účtů

Odstranění uživatelských účtů z databáze nebude realizováno jeho přímým vymazáním, ale za pomoci příznaku o zneplatnění záznamu. Databázové záznamy ostatních entit bude možné trvale odstraňovat z databáze.

3.2.6 N6 - Konfigurace aplikace

Konfigurační údaje pro připojení k databázi, konfiguraci emailového serveru a adresářové cesty k logování budou získávány pomocí JNDI resources aplikačního serveru. Případné další potřebné konfigurační údaje budou umístěny v externím konfiguračním souboru.

3.2.7 N7 - Logování

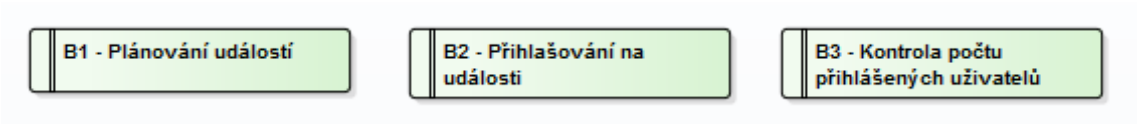
Aplikace bude ukládat logování do souboru logu aplikačního serveru. Logovány budou CRUD operace s entitami a případné neočekávané stavy aplikace.

3.2.8 N8 - Výkonnostní požadavky

Předpokládané zatížení aplikace odpovídá 10 umístěním, každé průměrně o 5 místnostech. Počet kategorií (labelů) je očekáván v počtu 10 položek. Dále je předpokládáno 30 trenérských uživatelských účtů a jednotky stovek ostatních uživatelských účtů (v součtu do 1 tisíce). Průměrná doba odezvy při 10 přihlášených uživatelských účtech v testovacím prostředí bude na rozsahu výše uvedených dat do 500 ms.

3.3 Business pravidla

Celkem bylo identifikováno 7 business pravidel pro software. Seznam pravidel je zobrazen na obrázku 3.3. Podrobný popis jednotlivých pravidel je v následujících podkapitolách.



Obrázek 3.3. Business rules

3.3.1 B1 - Plánování událostí

Při vytváření nové události bude aplikace kontrolovat, zda ve stejný termín není na stejném místě již naplánovaná jiná událost. Pokud ano, nebude uživateli umožněno vytvořit tuto událost a bude nutná změna termínu či místa konání.

3.3.2 B2 - Přihlašování na události

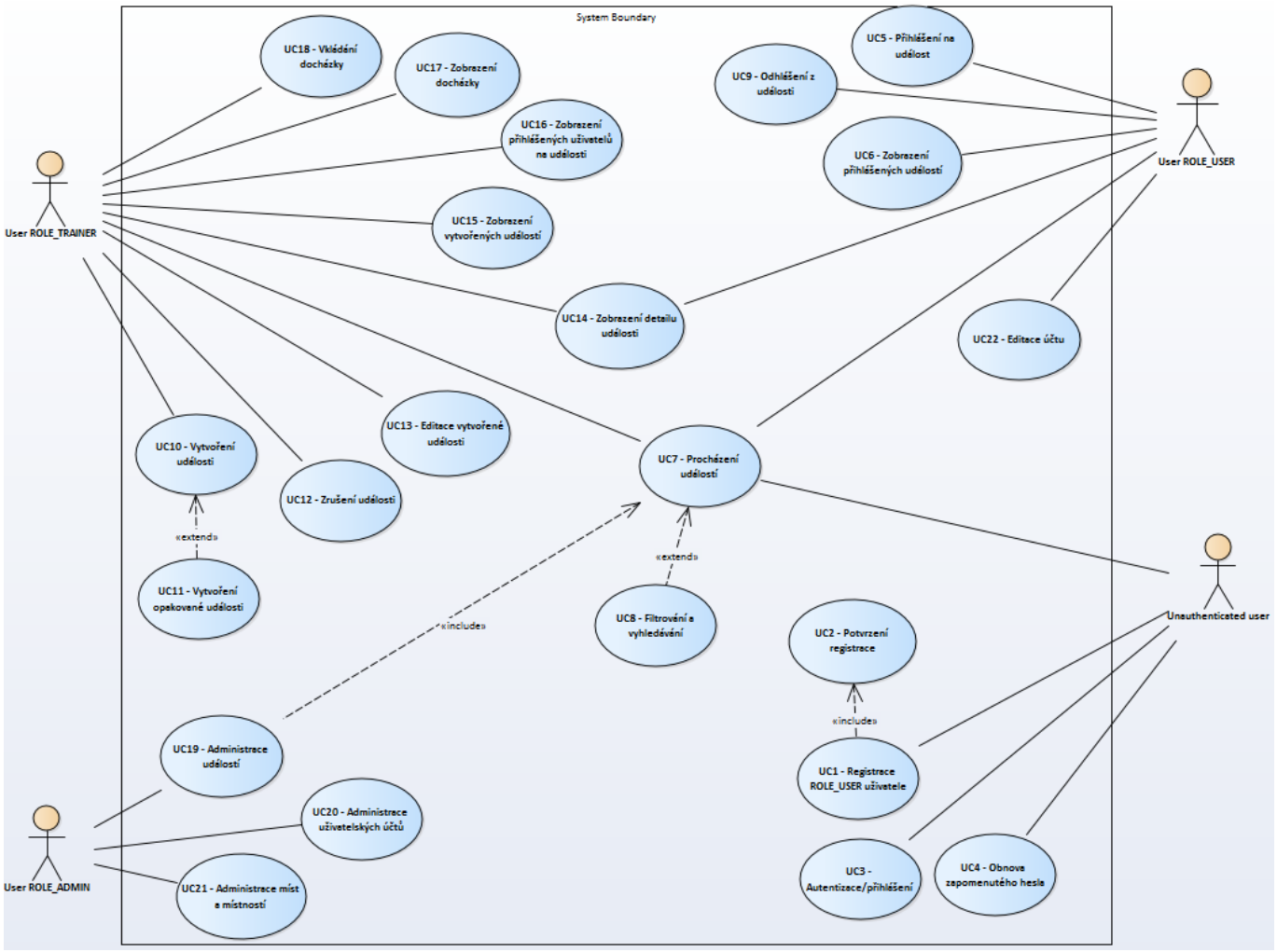
Aplikace bude při přihlašování uživatele na událost kontrolovat, zda již uživatel není v čase konání události přihlášený na jinou paralelně konanou událost. V případě kolize nebude uživateli umožněno se na tuto událost přihlásit.

3.3.3 B3 - Kontrola počtu přihlášených uživatelů

Aplikace bude kontrolovat kapacitu přihlášených účastníků události s již přihlášeným počtem uživatelů. Pokud počet přihlášených účastníků dosáhne hodnoty kapacity, dalším uživatelům již není zpřístupněno přihlašování na tuto událost. Přihlašování je pro nepřihlášené uživatele opět zpřístupněno v případech zvýšení kapacity události nebo odhlášení jiného uživatele z události.

3.4 Případy užití

Následující kapitola představuje identifikované aktéry interagující s navrhovaným systémem a jednotlivé případy užití (use cases), které mohou aktéři provádět. Vztahy mezi aktéry a případy užití jsou zobrazeny na následujícím use case diagramu na obrázku 3.4.



Obrázek 3.4. Use case diagram

3.4.1 Aktéři

V průběhu procesu sběru a analýzy požadavků byli identifikováni čtyři různí aktéři, kteří mohou pracovat s navrhovaným softwarem. Konkrétní aktéři jsou následující:

1. **Unauthenticated user** - Aktér představující anonymního návštěvníka webového portálu.
2. **User ROLE_USER** - Aktér představující autentizovaného uživatele, který se přihlašuje na vypsané události a následně tyto události navštěvuje.
3. **User ROLE_TRAINER** - Aktér představující trenéra, případně osobu, které je umožněno vytvářet a spravovat nové události. Tento uživatel je aplikací autentizován.
4. **User ROLE_ADMIN** - Aktér představující autentizovaného uživatele s právy umožňující kompletní administraci portálu.

3.4.2 Use cases

Tato podkapitola popisuje identifikované případy užití jednotlivých aktérů identifikovaných v předchozí podkapitole 3.4.1. Přehled případů užití je znázorněn v use case diagramu 3.4.

- **UC1 - Registrace ROLE_USER uživatele** - Neautentizovaný uživatel vyplní registrační formulář obsahující uživatelské jméno, heslo, jméno, příjmení, email a telefonní číslo.

  Před odesláním údajů zkontroluje klientská část aplikace platnost vstupů pro heslo, email a telefonní číslo. Uživatelské jméno je složeno z alfanumerických znaků o délce 3-30 znaků a v rámci aplikace je pro každého uživatele unikátní. Unikátní pro každého registrovaného uživatele je v rámci aplikace také email.

  Po úspěšné validaci zadaných informací je uživateli zobrazena informace o úspěšné registraci a odeslán aktivační email na uvedenou adresu. V případě nedodržení některého z požadavků na vstupní data je uživatel vyzván k opravě údajů.

- **UC2 - Potvrzení registrace** - Uživatel potvrdí registraci účtu přistoupením na odkaz, který je součástí emailu potvrzující registraci. Po přistoupení na odkaz je uživateli zobrazena informace o aktivaci účtu. Uživatel musí aktivaci účtu provést do 24 hodin od registrace.

- **UC3 - Autentizace/přihlášení** - Pokud uživatel chce provést operaci, na kterou musí být autentizován, aplikace zobrazí formulář pro zadání přihlašovacích údajů - uživatelské jméno a heslo. Po odeslání formuláře aplikace zkontroluje přihlašovací údaje. Po úspěšném přihlášení je uživatel přesměrován na úvodní stránku. Pokud přihlašovací údaje nejsou správné, aplikace zobrazí informaci o chybných přihlašovacích údajích.

- **UC4 - Obnova zapomenutého hesla** - Součástí přihlašovacího formuláře bude odkaz na formulář umožňující obnovu zapomenutého hesla. Uživatel zadá do formuláře svůj email uvedený při procesu registrace. Po odeslání formuláře bude zkontrolováno, zda existuje účet s tímto emailem. Následně aplikace odešle na tuto adresu email s odkazem, pomocí kterého bude uživatel přesměrován na stránku umožňující zadání nového hesla. Po zadání nového hesla bude uživatel přihlášen a přesměrován na úvodní stránku aplikace.

  Pokud pro zadaný email neexistuje žádný vytvořený účet, aplikace uživateli také zobrazí potvrzení o odeslání emailu s odkazem na obnovu hesla, ale žádný email se neodešle. Toto znemožní zjištění, zda pro daný email je vytvořený uživatelský účet.

- **UC5 - Přihlášení na událost** - Uživatel si vybere v seznamu událostí událost, které se chce zúčastnit, je na ní zpřístupněné přihlašování a kapacita události není naplněna (UC7). Po zobrazení detailu události (UC14) vybere možnost přihlásit se na událost. Po této akci bude vytvořena nová přihláška na událost a uživatel bude informován o úspěšném přihlášení. Před uložením záznamu o přihlášení provede aplikace kontrolu, zda uživatel již není ve stejný termín přihlášen na jinou událost. Po přihlášení na událost je možnost *Přihlásit na událost* změněna na *Odhlásit z události*.

- **UC6 - Zobrazení přihlášených událostí** - Uživatel se rozhodne zobrazit všechny události, na které se v minulosti přihlásil pomocí případu užití UC5. Uživatel v menu vybere možnost *Přihlášené události*, aplikace poté zobrazí seznam všech událostí,

na které vytvořil přihlášky. Uživatel bude dále moci tento seznam událostí filtrovat podle termínu konání (interval od, do).

- **UC7 - Procházení událostí** - Uživatel v případě zájmu prohlížet vypsané události vybere v hlavním menu volbu *Události*. Aplikace následně uživateli zobrazí tabulku se seznamem vypsaných událostí.
- **UC8 - Filtrování a vyhledávání** - Jedná se o rozšíření případu užití UC7. Na obrazovce se seznamem vypsaných událostí vybere v podmenu položku *Filtrovat události*. Aplikace následně zobrazí formulář, který slouží k filtrování událostí dle zadaných parametrů. Aplikovatelné filtry jsou název události, kategorie (label), místa konání, termínu konání a jméno trenéra, který vytvořil událost. Po odeslání tohoto formuláře jsou zobrazeny pouze události, které odpovídají nastavení filtru.
- **UC9 - Odhlášení z události** - Uživatel vybere událost, na kterou je přihlášen a ze které se chce odhlásit. Toto provede pomocí případu užití UC6, UC7, UC8 nebo UC14. Po zobrazení detailu události vybere možnost *Odhlásit z události*. Vybráním této možnosti se provede zrušení přihlášky. O úspěšném odhlášení je uživatel informován potvrzením. Po odhlášení je možnost *Odhlásit z události* nahrazena možností *Přihlásit na událost*.
- **UC10 - Vytvoření události** - Přihlášený trenér naplánuje novou událost. Po zvolení volby *Přidat jednorázovou událost* v podmenu obrazovky *Události* zobrazí aplikace uživateli formulář pro vyplnění údajů o nové události. Po vyplnění údajů (jméno události, termín konání, termín pro přihlašování, místo konání, stav události, minimální a maximální počet účastníků) uživatel formulář odešle a je vytvořena nová událost. Před uložením události provede aplikace kontrolu, zda na místě konání již není ve stejný termín naplánovaná jiná událost. O úspěšném vytvoření události je uživatel aplikací informován potvrzujícím sdělením. V případě kolize s jinou událostí konanou na stejném místě ve stejném čase je uživatel vyzván k úpravě času nebo místa konání.
- **UC11 - Vytvoření opakované události** - Jedná se o rozšíření případu užití UC10. Přihlášený trenér zvolí v podmenu obrazovky *Události* možnost *Vytvořit opakovanou událost*. Trenér, který událost zakládá, následně zvolí interval a čas, v jakém má být událost opakovaně naplánovaná a jestli denně, týdně nebo každý sudý/lichý týden. Aplikace následně vytvoří na tyto dny události se stejným místem konání, kapacitou a názvem.
- **UC12 - Zrušení události** - Trenér, který je autorem události, nalezne událost k odstranění v seznamu vytvořených událostí - viz. use case UC15. V seznamu uživatel u vybrané události zvolí možnost *Zrušit událost* a potvrdí svou volbu, že tak chce skutečně učinit. Aplikace následně událost odstraní společně se všemi vytvořenými přihláškami. Přihlášeným uživatelům na zrušenou událost bude odeslán email s informací o zrušení události.
- **UC13 - Editace vytvořené události** - Přihlášený trenér si pomocí use case UC15 vybere událost k editaci. Po zobrazení detailu události může uživatel editovat jednotlivé atributy události a zpřístupnit událost pro přihlašování. Po stisknutí tlačítka pro uložení změn aplikace provede aktualizaci události a odešle email přihlášeným uživatelům, pokud došlo ke změně některého z následujících atributů: název události, termín události nebo umístění události.
- **UC14 - Zobrazení detailu události** - Uživatel si ve výpisu událostí (use case UC6, UC7 nebo UC15) vybere v tabulce událost, kterou chce zobrazit detailněji. Po vybrání

řádku s událostí je zobrazena obrazovka obsahující detailní informace o události, včetně informací o místě konání, počtu aktuálně přihlášených uživatelů a trenérovi, který událost vypsal. Z této obrazovky je dále možné provést přihlášení/odhlášení z události pomocí UC5 a UC9.

- **UC15 - Zobrazení vytvořených událostí** - Přihlášený trenér si zobrazí veškeré jeho dříve vytvořené události. K zobrazení využije use case UC8, kde po nastavení filtru *Tvůrce události* na své jméno a následném odeslání formuláře aplikace zobrazí pouze události, které vytvořil. Události mohou být následně dále filtrovány dle stavu, termínu konání apod.
- **UC16 - Zobrazení přihlášených uživatelů na události** - Přihlášený uživatel s účtem trenéra si vybere v seznamu svých vytvořených událostí konkrétní položku a zobrazí její detail (UC14). V obrazovce detailu události aplikace zobrazí seznam aktuálně přihlášených osob. Seznam osob bude obsahovat následující údaje o uživateli: jméno, příjmení, username a odkaz na detail uživatele obsahující další kontaktní údaje (email, telefonní číslo).
- **UC17 - Zobrazení docházky** - Přihlášený uživatel s účtem trenéra si vybere v seznamu svých vytvořených událostí konkrétní položku a zobrazí její detail (UC14). Aplikace zobrazí vyplněnou docházku ve formě předvyplněného formuláře pro vkládání docházky (UC18). Pokud docházka ještě nebyla zadána, je u všech přihlášených zobrazený stav nepřítomen.
- **UC18 - Vkládání docházky** - Přihlášený uživatel s účtem trenéra si vybere v seznamu svých vytvořených událostí konkrétní položku a zobrazí její detail (UC14). Součástí zobrazení detailu je formulář pro vyplnění docházky u jednotlivých účastníků události (přihlášených uživatelů na událost). Formulář obsahuje jména přihlášených uživatelů a checkboxy. Uživatel vyplní formulář a odešle docházku k uložení. V případě předchozího vyplnění docházky je formulář předvyplněn.
- **UC19 - Administrace událostí** - Administrátor po přihlášení vybere možnost *Události*. Aplikace zobrazí seznam všech událostí vytvořených v rámci aplikace (UC7). Administrátor vybere položku ze seznamu a zobrazí její detail, ve kterém je možné editovat její atributy, případně událost odstranit. Administrátor může editovat i seznam přihlášených uživatelů na událost. Po odeslání formuláře s editovanými údaji o události je změna aplikací uložena.

  Administrátor je dále oprávněn vytvořit novou událost, včetně možnosti přiřazení zodpovědné osoby za událost (tzn. trenéra, který má následně možnosti, jako kdyby událost vytvořil sám).
- **UC20 - Administrace uživatelských účtů** - Administrátor po přihlášení vybere možnost *Uživatelé*. Aplikace zobrazí seznam všech uživatelských účtů vytvořených v rámci aplikace. Administrátor vybere položku ze seznamu a zobrazí její detail, ve kterém je možné editovat všechny atributy, včetně aktivace účtu. Součástí administrace je také možnost odstranění (zneplatnění) záznamu. Dále je možné definovat role uživatele a veškeré provedené přihlášky uživatele na události. Administrátor je dále oprávněn vytvořit nový uživatelský účet, včetně možnosti jeho aktivace bez nutné aktivace pomocí emailu. Po odeslání formuláře s editovanými údaji o uživateli je změna aplikací uložena.
- **UC21 - Administrace míst a místností** - Administrátor po přihlášení vybere možnost *Umístění* pro administraci míst, případně *Místnosti* pro administraci jednotlivých

místností). Aplikace zobrazí seznam všech míst a místností, ve kterých se mohou konat události. Administrátor vybere záznam a zobrazí si jeho detail, ve kterém je možné editovat všechny atributy. Součástí editace bude mít administrátor i možnost odstranění položky z databáze a to pouze v případě, že v daném umístění (místnosti) není naplánována žádná událost. Pouze administrátor je oprávněn vytvářet nové místa a místnosti pro konání událostí. Po odeslání formuláře s editovanými údaji je změna aplikací uložena.

- **UC22 - Editace účtu** - Přihlášený uživatel se rozhodne upravit některý z údajů, který uvedl v registračním formuláři. Ve správě svého účtu bude mít možnost upravit údaje - jméno, příjmení, telefonní číslo a heslo. Pro potvrzení změn uživatel zadá své současné heslo. Aplikace po odeslání formuláře aktualizuje údaje uživatelského účtu a informuje uživatele o úspěšné změně údajů. Username a emailovou adresu nebude možné editovat.

3.4.3 Mapování případů užití na funkční požadavky

Následující tabulka 3.1 zobrazuje realizaci jednotlivých funkčních požadavků identifikovanými případy užití. Tuto tabulku lze využít ke kontrole splnění všech funkčních požadavků vzhledem k identifikovaným případům užití. Pokud by se v tabulce vyskytl funkční požadavek, který není pokrytý případem užití, může být tento funkční požadavek definován nesprávně a zbytečně, případně nejsou identifikovány všechny potřebné případy užití. Z výsledné tabulky lze poznat, že všechny identifikované funkční požadavky z kapitoly 3.1 jsou ve vztahu s alespoň jedním případem užití.

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14
UC1	X					X								
UC2	X					X								
UC3			X	X	X	X	X	X	X	X	X	X	X	X
UC4		X				X								
UC5									X					
UC6								X				X		
UC7									X		X		X	
UC8									X		X		X	
UC9									X					
UC10				X										
UC11				X										
UC12					X	X								
UC13				X		X								
UC14									X		X			
UC15							X				X			
UC16										X				
UC17								X						
UC18							X							
UC19									X	X	X		X	X
UC20	X	X							X			X		X
UC21														X
UC22	X													X

Tabulka 3.1. Realizace funkčních požadavků jednotlivými případy užití

Kapitola 4
Návrh architektury a výběr technologií

Následující kapitola popisuje průběh a důvody výběru použitých technologií k implementaci a seznámení čtenáře se specifickými vlastnostmi těchto technologií. Dále je proveden návrh architektury pro následnou implementaci aplikace vybranými technologiemi.

Vzhledem k získaným požadavkům na software bylo rozhodnuto, že bude vytvořena webová aplikace. Výhoda tohoto řešení proti desktopové aplikaci je absence nutnosti instalovat desktopovou aplikaci a použitelnost webové aplikace na všech platformách obsahující webový prohlížeč podporující JavaScript.

4.1 Výběr technologií

Podkapitola *Výběr technologií* je rozdělena na dvě části. První se zabývá technologiemi využitých při implementaci serverové částí (backendu). Druhá část je zaměřena na technologie použité pro klientskou část aplikace (frontend). Oddělení backendu a frontendu aplikace a jejich komunikace pomocí REST rozhraní je navrženo s ohledem na nefunkční požadavek N5, viz. 3.2.1, a případnou budoucí realizaci mobilní aplikace.

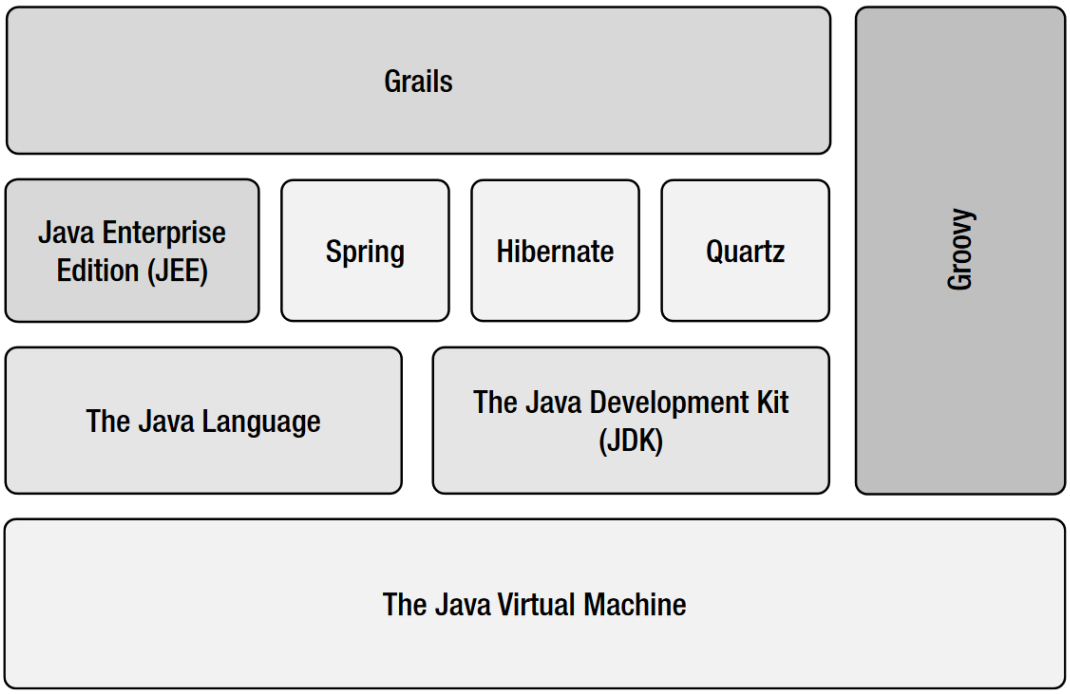
4.1.1 Serverová část

Z možných technologií a programovacích jazyků, které lze využít pro implementaci webové aplikace zprostředkovávající REST rozhraní, byla preferována Java EE. V případě použití čisté Javy EE je nutné implementovat spoustu komponent, které jsou již součástí různých frameworků a pluginů. Použití frameworku je také vhodné pro vytvoření robustní architektury aplikace a její udržitelnost v průběhu dalšího vývoje a údržby.

Z tohoto důvodu se další směr výběru vhodných technologií zaměřil na webové frameworky, které jsou na Java EE založeny a výsledná aplikace je nasaditelná na aplikační server (JSP container), jako jsou například Tomcat nebo Glassfish. Frameworky zahrnuté do výběru byly Grails, JavaServer Faces, Spring MVC, Vaadin. Součástí požadavků není nasazení aplikace do cloudu, z tohoto důvodu nebyl u těchto frameworků prováděn výzkum možností nasazení aplikace do cloudu. Pro konečnou implementaci byl zvolen framework Grails ve verzi 3.2.4. Důvody tohoto rozhodnutí jsou popsány dále.

Framework Grails je inspirován webovým frameworkem Rails pro programovací jazyk Ruby [13]. První verze Grails byla vydána v roce 2005 a aktuálně je vyvíjena třetí generace tohoto frameworku. Struktura Grails a použité komponenty uvnitř frameworku jsou znázorněny na obrázku 4.1.

Webový framework Grails využívá programovací jazyk Groovy. Jedná se o objektově orientovaný jazyk pro Java platformu (je dynamicky kompilovaný do JVM bytecode). Groovy čerpal inspiraci z objektových a skriptovacích jazyků Ruby, Python, Perl nebo Smalltalk [13]. Hlavní odlišnosti a rozšíření oproti Javě jsou následující (příklady jsou inspirovány dokumentací jazyka Groovy, viz. [14]):



Obrázek 4.1. Vnitřní struktura frameworku Grails, převzato z [13]

- **Použití operátoru** `==` - Operátor `==` v Groovy na rozdíl od Javy znamená logickou rovnost pro všechny typy atributů dvou porovnávaných objektů (v Javě nutné předefinovat a použít metody `hashcode()` a `equals()`)
- **Typová kontrola** - Groovy umožňuje vytvářet proměnné bez určení datového typu, pomocí klíčového slova `def` místo názvu datového typu před jménem proměnné, např. `def number = 5`. Dále je možné vytvářet pole obsahující různé datové typy - `def array = [4, 2.5, 'pět']`. Groovy umožňuje vytvoření switch deklarace napříč různé datové typy včetně pole:

```
switch (x) {
    case 1.23:
    result = "double found"
    break
    case "bar":
    result = "bar"
    break
    case [4, 5, 6, 'inList']:
    result = "list"
    break
    case 12..30:  //tento příkaz odpovídá poli [12, 13, ..., 29, 30]
    result = "range"
    break
    default:
    result = "default"
}
```

- **Closure, klíčové slovo it** - Dalším rozšiřujícím prvkem Groovy jsou tzv. Closures. Jedná se o konstrukt, obdobný lambda výrazům, který je součástí Java 8. Lze jím

nahradit for cykly nebo celé metody. Jedná se o blok kódu uzavřený ve složených závorkách, který je vykonán se vstupním objektem. Klíčové slovo `it` uvnitř bloku kódu představuje proměnnou, která obsahuje vstupní parametr tohoto konstruktu, viz. příklad:

```
//druhá mocnina čísla
def mocnina = {it * it}
println mocnina (3)     //výstup 9

//vynásobení každého prvku pole číslem 2
def arr = [1,2,3]
arr.each { it * 2 }     //arr = [2,4,6]
```

- **Stringy** - Řetězce je možné v Groovy definovat dvěma způsoby - pomocí uvozovek nebo apostrofů (obdobně jako v JavaScriptu). Groovy dále umožňuje skládání řetězců pomocí tzv. Groovy stringů[13]. Jedná se o vykonání metody `toString()` nad atributem definovaným uvnitř bloku `${}`. Oba zápisy níže mají shodný výsledek:

```
println "Dnešní datum je ${new Date()}."
println "Dnešní datum je" + new Date().toString() + "."
```

- **Tvorba map** - Groovy umožňuje tvorbu map obdobnou syntaxí jako JSON, pomocí dvojic `klíč: hodnota`. Deklarace prázdné mapy je realizováno pomocí příkazu `[:]`. Deklarace jednoduché mapy je po té následující:

```
def map = ['klic1': 'hodnota1', 'klic2': 2]

//přístup k prvkům v mapě:
println map.klic1
println map['klic1']
```

- **Ukončení řádek kódu** - Groovy nevyžaduje na rozdíl od Javy ukončování řádek kódu pomocí středníku.

Framework Grails je navržen podle MVC návrhového vzoru, který odděluje doménový model aplikace, uživatelské rozhraní (view) a aplikační logiku v controlleru [13].

- **Model** - Model představuje jádro aplikace, kde jsou definovány business (datové) entity, omezení jejich atributů (např. přípustné mezní hodnoty) a vztahy (relace) mezi entitami. Tyto entity je možné ukládat perzistentně do databáze a opět je z ní načítat pomocí ORM (GORM).
- **View** - Část View je zdopovědná za vykreslování uživatelského rozhraní, u webové aplikace je to typicky HTML stránka. View jsou ve frameworku Grails reprezentovány `.gsp` soubory, které často potřebují parametry z části Model. K hodnotě parametru je v .gsp souboru přistupováno pomocí Groovy stringů (konstrukt `${}`).
- **Controller** - Controllery jsou třídy, které zpracovávají požadavky a akce uživatelů [13]. Přijímají HTTP požadavky a delegují je na jednotlivé části modelu nebo na části view. Parametry jsou jednotlivým view předány pomocí mapy příslušnou akcí controlleru pomocí return konstruktu. Příklad předání parametrů z akce `bar` controlleru `Foo` příslušnému BarView.gsp:

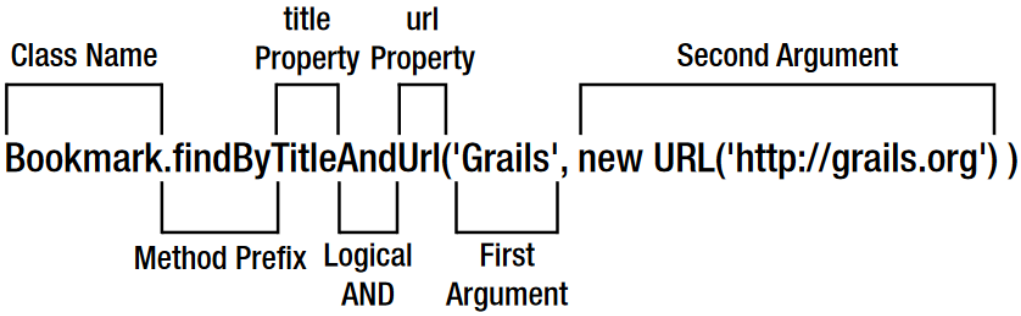
```
class FooController{
   def bar(){
      return [parameter: "Hello Bob"]
   }
}
```

URL pro volání konkrétní akce controlleru je defaultně nastaveno následovně: `http://applicationURL/controller/action`, pro příklad výše je URL požadavku `http://applicationURL/foo/bar`. Controllery také obstarávají komunikaci REST rozhraní. Jedná se o rozšíření servletů z Java EE. Pro umožnění vytvoření komplexních aplikací, kde veškerou logiku neřídí pouze controllery, umožňuje Grails také definování služeb (service) [13]. Jedná se o třídy, které mohou sloužit například k parsování XML konfigurace, zasílání emailů, REST klient pro komunikaci se vzdálenou službou apod.

Grails využívá k ORM mapování tzv. Grails ORM, zkráceně GORM. Jedná se o nadstavbu nad Hibernate. Použít je možné stejné relační databáze, které jsou podporovány Hibernate. Hlavní přínosy použití GORM místo samotného Hibernate jsou následující:

- **Validace entit (domény)** - GORM poskytuje mechanismy pro validaci definovaných entit doménového modelu. Tyto konstrukce usnadňují vývojáři implementaci integritních omezení doménového modelu. Pro jednotlivé atributy je možné v definici třídy například říct, v jakém intervalu musí být hodnota atributu, minimální a maximální délka řetězce, zda má být některý atribut v rámci domény unikátní a další. Konkrétní možnosti validací jsou uvedeny v dokumentaci [15] v sekci *Constraints*. Validace je prováděna před prvním uložením nové entity do databáze a při každém jejím dalším updatu.
- **Dotazování pomocí Dynamic Finders** - Grails nevyužívá k manipulaci s objekty DAO, ale tzv. Dynamic domain class method [13]. Tyto metody jsou volány z doménových objektů, konkrétně se jedná o metody:
  - `save()` - uložení nebo update entity
  - `delete()` - trvalé odstranění entity
  - `get()` - získání entity podle id
  - `list()` - získání seznamu všech entit
  - `count()` - získání počtu perzistuovaných entit
  - `findBy*()`
  - `findAllBy*()`

  Poslední dva zmíněné body jsou dynamické metody, které umožňují dotazování. Výslední dotaz odpovídá názvu dané metody a vstupním parametrům. Tvorba takového dotazu je znázorněna na obrázku 4.2. Po názvu metody následuje místo symbolu `*` název atributu, jehož hodnota ovlivňuje dotazování. Vstupním parametrem metody je následně hodnota argumentu. Návratová hodnota metody je entita (případně seznam entit u metody `findAllBy*()`), která odpovídá danému dotazu. Názvy atributů pro dotazování je možné za sebe zřetězit pomocí logických spojek `And` a `Or`. Dotazování je možné provádět také pomocí SQL dotazů, HQL nebo pomocí vytvoření kritérií (Criteria).



Obrázek 4.2. Dotazování pomocí Dynamic Finders, převzato z [13]

- **Řazení a stránkování** - Získání množiny prvků z databáze lze doplnit možností seřazení záznamů vzestupně či sestupně podle určeného atributu a stránkováním (získání pouze určité části ze všech záznamů). Tyto funkcionality jsou realizovatelné nastavením parametrů `max` (maximální počet zaslaných záznamů), `offset` (od kolikátého záznamu), `sort` (název atributu pro řazení) a `order` (vzestupně, sestupně).

Pro zajištění bezpečnosti aplikace, přihlašování uživatelů a uchovávání přihlašovacích údajů bylo vybráno řešení pomocí Spring Security Core pluginu pro Grails framework. Toto řešení již obsahuje možnost Ajax autentizace, autorizace a ochranu proti možným útokům na bezpečnost uživatele (např. Session hijacking, session stealing) [16]. Tento plugin provádí před uložením hashování hesel pomocí *bcrypt* algoritmu s možností změny hashovacího algoritmu na *pbkdf2*. Díky tomu aplikace neukládá hesla uživatelů v plain textu a není umožněno případnému útočníkovi zcizit uložená uživatelská hesla z databáze.

V poslední velké aktualizaci Grails na verzi 3 došlo k několika velmi důležitým změnám oproti předchozím verzím. V této verzi je nově framework založen na Spring boot a buildovacím nástroji Gradle [17]. Gradle nahradil zastaralý Gant (Grails Ant) [17]. Dále byl aktualizován Spring na verzi 4.1. GORM nově podporuje i NoSQL databáze, konkrétně MongoDB, Neo4J, Cassandra a Redis.

Perzistentním úložištěm v průběhu vývoje byla zvolena relační databáze PostgreSQL. Pro produkční prostředí je možné použít libovolnou relační databázi podporovanou aplikačním serverem a komponentou GORM (resp. Hibernate).

Volbou frameworku Grails pro backend webové aplikace jsou získány možnosti Javy EE (použitelnost existujících repozitářů), frameworku Spring a kompatibilita s existujícími aplikačními servery a navíc možnost využití síly programovacího jazyka groovy a komponenty GORM.

4.1.2 Klientská část

Na doporučení vedoucího práce, pana Ing. Pavla Šedka, bylo rozhodnuto pro klientskou část aplikace použít JavaScriptové knihovny BackboneJS a MarionetteJS. Klientská část aplikace bude reprezentována JavaScriptovou aplikací, založenou na těchto knihovnách. Tato aplikace bude komunikovat skrze REST API se serverovou částí, popsanou v předchozí kapitole 4.1.1. Řešení frontendu pomocí JavaScriptové aplikace, místo použití Grails View, byl zvoleno z důvodu nefunkčního požadavku N1 - 3.2.1 a plánovaného budoucího vytvoření mobilní aplikace. Vytvořenou JavaScriptovou aplikaci je možné po mírných úpravách kódu a konfigurace (úpravy URL cest) použít k vytvoření mobilní aplikace - například za pomoci frameworku Cordova nebo web view. JavaScriptovou aplikací je myšlena single-page webová stránka, která je načtena do prohlížeče a ná-

sledně vyměňuje data se serverovou částí bez potřeby úplného znovu načítání stránek ze serveru.

Backbone je lightweight JavaScriptová knihovna poskytující vyvíjeným aplikacím MVC architekturu [18]. Rozdíl mezi tradičním MVC (např. Grails) a jeho implementací pomocí Backbone je v roli Controlleru. Dokumentace Backbone [19] a literatura [18] uvádí, že na View může být pohlíženo jako na Controller - zpracovává události pocházející z uživatelského rozhraní (např. submit formuláře), přičemž šablona HTML slouží jako pravé View. Proto je architektura Backbone v literatuře [18] označována jako MV*. Výsledná JavaScriptová aplikace komunikuje skrze existující RESTful JSON rozhraní backendu.

Základními stavebními prvky Backbone jsou třídy Model, Collection a View. Charakteristika těchto tříd je následující [19]:

- **Backbone.Model** - Získává data ze serveru a posílá požadavek k uložení/aktualizace záznamu - provádí tzv. synchronizaci s backendem. Obaluje JSON objekt přenášený přes REST rozhraní.
- **Backbone.View** - Renderuje uživatelské rozhraní pomocí HTML šablon, zpracovává uživatelské vstupy a interakci, zobrazuje uživateli položku (Model nebo Collection) pomocí HTML šablony.
- **Backbone.Collection** - Obsahuje množinu objektů Backbone.Model, umožňuje řazení a filtrování záznamů pomocí parametrů.

Instance tříd Backbone.View a Backbone.Collection umožňují provádět CRUD operace pomocí REST rozhraní. Volání HTTP požadavků představující CRUD operace RESTful rozhraní je prováděno následovně:

```
GET  /books/ .... collection.fetch();
POST /books/ .... collection.create();
GET  /books/1 ... model.fetch();
PUT  /books/1 ... model.save();
DEL  /books/1 ... model.destroy();
```

Ačkoliv Backbone aplikace je single-page, je možné směrování pomocí URL. Tato funkcionalita je realizována třídou Backbone.Router a umožňuje například uživateli sdílení odkazu nebo vrácení na předchozí stránku pomocí stisku tlačítka zpět v prohlížeči. Router provádí směrování pomocí části URL za symbolem `/#`. V případě změny URL zavolá příslušnou funkci a je možné renderovat potřebné view.

Hlavní předností Backbone je malá velikost, čímž umožňuje použití s pomalým internetovým připojením či na mobilních zařízeních. Produkční zkompilovaný soubor (minified a gzip) knihovny Backbone 1.3.3 má velikost 7,6kB. Pro srovnání framework AngularJS 1.6.4 má velikost 144kB. Backbone vyžaduje JavaScriptové knihovny jQuery a Underscore.js

Pro vývoj rozsáhlých aplikací je samotný Backbone nevhodný, hodí se na malé single-page stránky nebo stránky využívající manipulaci s jquery DOM [18]. Marionette zavádí funkcionality a návrhové vzory potřebné pro vývoj netriviálních JavaScriptových aplikací [18]. Cílem Marionette je zjednodušení rozsáhlých konstrukcí a aplikací. Klíčové benefity použití Marionette popsané v [18] jsou následující:

- **Vnořené view** - Marionette umožňuje definování regionů a vytváření vnořených view (tzv. Child view). V HTML šabloně View lze definovat region pomocí jQuery selec-

toru [20]. V definovaném regionu lze následně dynamicky renderovat jiné View (tzv. ChildView).

- **Správa paměti** - Marionette obsahuje memory management, který zajišťuje uvolňování paměti po neaktuálních view a mapovaných událostech [18]. Jeho cílem je snížení náročnosti na výkon a zamezení vzniku tzv. zombie view. Tomuto problému se věnuje část v kapitole 5.3.6 - implementace frontendu, implementační problémy.
- **CollectionView** - CollectionView umožňuje vytvářet view pro množinu modelů. Pro každý element z Backbone.Collection provede renderování childView [20]. Tento koncept je užitečný při renderingu více položek stejné třídy modelu - např. renderování řádků tabulky nebo možností select boxu.

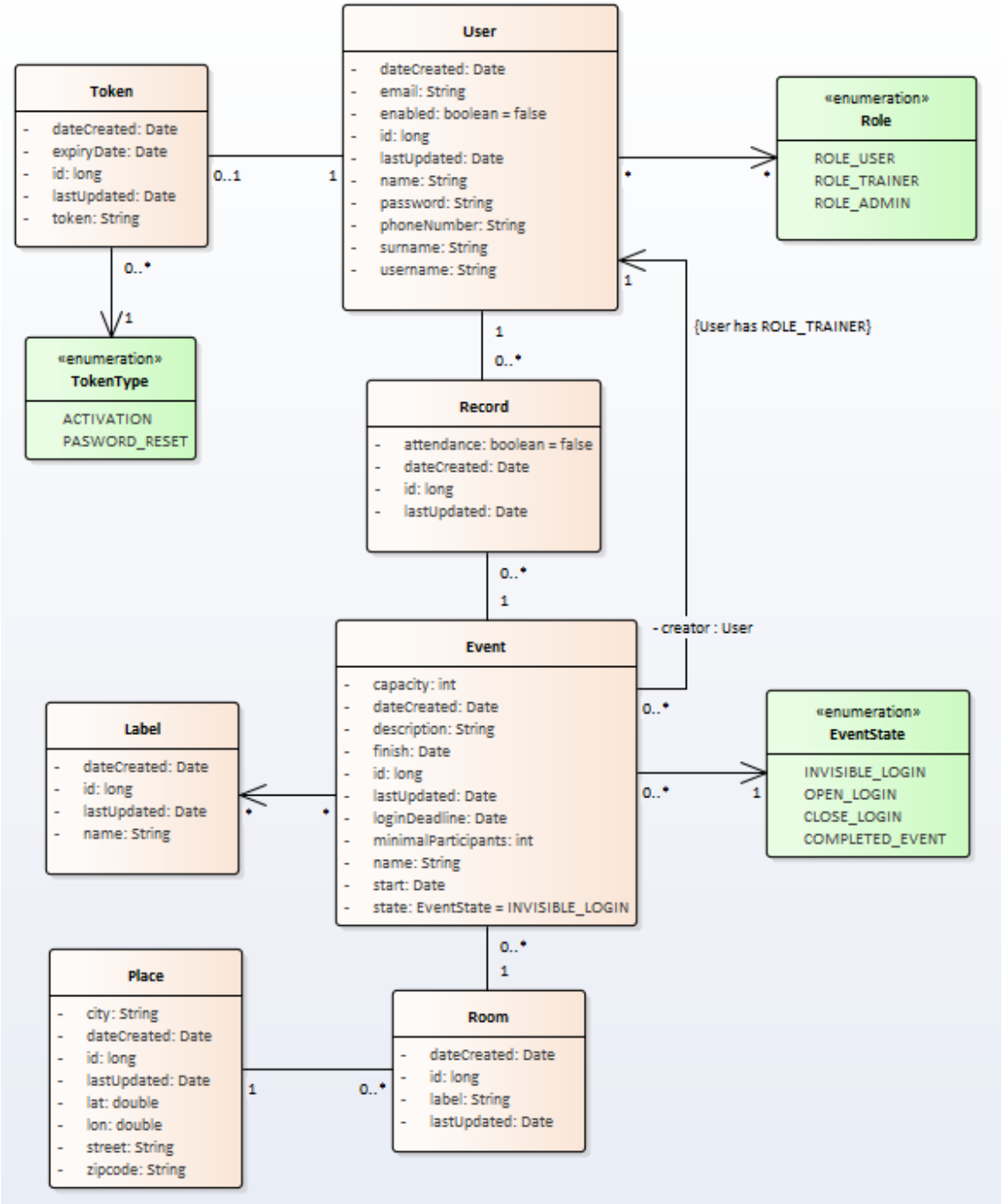
4.2 Doménový model

Následující podkapitola seznámí čtenáře s procesem tvorby doménového modelu. Identifikace entit a tvorba doménového modelu probíhala před výběrem implementačních technologií uvedené v kapitole 4.1 a snažila se tak být co nejvíce implementačně nezávislá na platformě. Výsledný UML diagram je zachycen na obrázku 4.3. Význam identifikovaných doménových objektů je následující:

- **User** - představuje registrovaný uživatelský účet uživatele
- **Token** - objekt definující unikátní identifikátor, který je použitý v odkazu URL k aktivaci účtu nebo obnově zapomenutého hesla a dobu jeho platnosti
- **TokenType** - výčtový datový typ, definující typ tokenu - token sloužící k aktivaci účtu, token sloužící k obnově zapomenutého hesla
- **Role** - výčtový datový typ, který definuje možné uživatelské role pro uživatelské účty
- **Event** - objekt představující konanou událost, vytvořenou uživatelským účtem s přiřazenou rolí trenéra
- **Record** - jednotlivé přihlášky uživatelů na konkrétní událost, obsahující záznam o docházce - z pohledu budoucí implementace a databázového úložiště se jedná o převod vazby M:N mezi User a Event na 2 vazby 1:N
- **Label** - tzv. štítky představující kategorie jednotlivých událostí - slouží k organizaci událostí dle druhu a následnému filtrování
- **EventState** - výčtový datový typ, definující stav události a možnosti přihlašování na událost a zadávání docházky trenérem
- **Room** - definuje místnost, ve které je konkrétní událost naplánována a konána
- **Place** - konkrétní umístění s adresou a GPS souřadnicemi - představuje budovu ve které jsou místnosti pro konání událostí

Stav entity Event definuje operace, které mohou uživatelé s entitou provádět. Popisy stavů a možných akcí je definován následně:

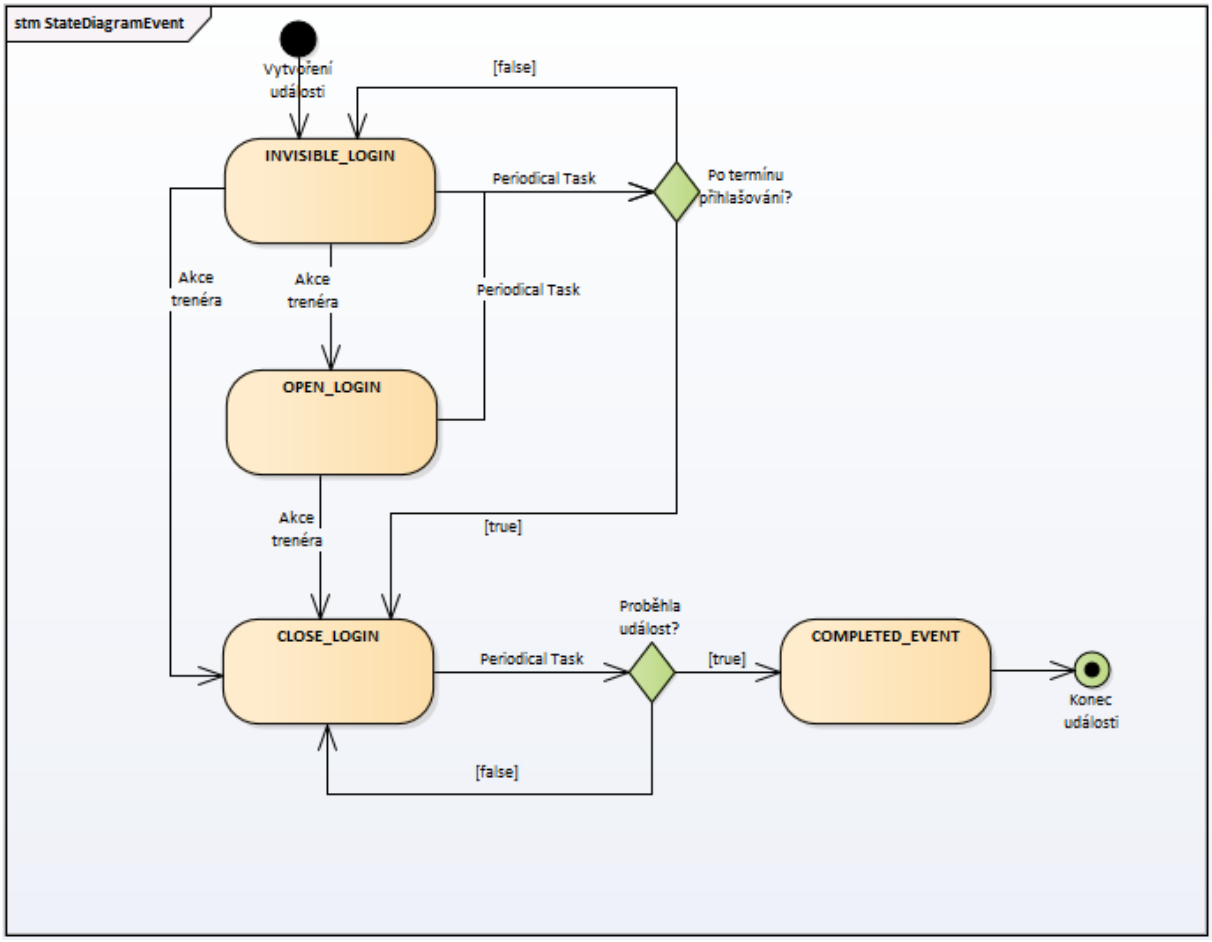
- **INVISIBLE_LOGIN** - nová událost je vytvořena, přihlašování není uživatelům zpřístupněno, jedná se o defaultní hodnotu stavu nově vytvořené události
- **OPEN_LOGIN** - přihlašování uživatelů na událost je zpřístupněno, přihlašování je možné, než proběhne termín konce přihlašování (loginDeadline)



Obrázek 4.3. UML diagram doménového modelu

- **CLOSE_LOGIN** - přihlašování na událost již není umožněno a již realizované přihlášky nemohou uživatelé měnit (odhlásit se)
- **COMPLETED_EVENT** - tento stav je nastaven po konci události, trenérovi (pouze tvůrci události) je umožněno vkládání docházky přihlášených uživatelů

Změny stavu může měnit v rámci editace údajů události trenér, který událost založil. Další změny stavů jsou měněny v průběhu času automaticky pomocí periodického tasku, který kontroluje stavy událostí vůči termínu konání a termínu konce přihlašování.



Obrázek 4.4. Stavový UML diagram entity Event

Změny stavů jsou dány stavovým diagramem 4.4. Uživatelskému účetu s administrátorskými právy je umožněno měnit stav události libovolně mezi 4 definovanými volbami.

Z důvodu definování termínu platnosti u entity Token je nutná periodická kontrola jeho překročení a odstraňění těchto záznamů. Společně s odstraněním expirovaných aktivačních tokenů je také potřeba provést odstranění neaktivovaných uživatelských účtů s expirovaným aktivačním tokenem. Důvodem pro odstranění expirovaných tokenů je kromě nepatrného zmenšení potřebného úložného prostoru hlavně odstranění neaktivovaných uživatelských registrací. V případě nerealizace této funkcionality by mohlo dojít k tomu, že uživatelský účet, který není aktivován a je expirovaný aktivační token, znemožňuje provedení nové registrace se stejnou emailovou adresou.

Součástí úspěšného provedení aktivace uživatelského účtu je odstranění příslušného aktivačního tokenu. Totéž platí u obnovy zapomenutého hesla - v případě úspěšné změny hesla pomocí tokenu je následně token odstraněn.

4.3 Architektura aplikace

Navržená architektura aplikace je postavena na principu *server - tenký klient*. Vzájemná komunikace mezi serverovou a klientskou částí je navržena skrze RESTful rozhraní pomocí HTTP požadavků. Formát přenášených dat je zvolen JSON. Umístěním jednotlivých částí na fyzická zařízení se zabývá podkapitola 4.4 - Diagram nasazení. Následující sekce seznámí čtenáře s architekturou serverové a klientské části aplikace.

4.3.1 Serverová část

K implementaci serverové části byl použitý webový framework Grails. Důvody a výhody plynoucí z volby této technologie jsou popsány v sekci 4.1.1. Volba použití tohoto frameworku ovlivňuje možnost výsledné architektury. Softwarová architektura odpovídá vzoru MVC.

- **Model (datová vrstva)**

  Část Model je reprezentována doménovým modelem uvedeným v předchozí podkapitole 4.2. ORM mapování a funkcionalita DAO objektů bude zajištěna pomocí komponenty GORM frameworku Grails a Dynamic Finders metod doménových objektů. Součástí doménových objektů je i definování podmínek pro validaci entit - viz. 4.1.1.

- **Controller (business logika, REST rozhraní)**

  Část Controller bude sloužit převážně k vystavování REST rozhraní, kterým bude aplikace komunikovat s klientskou částí. Dále část Controller bude zajišťovat poskytnutí klientské části aplikace (JavaScriptové aplikace) při přistoupení na URL `/`. Endpointy RESTového rozhraní budou mapovány na URL `/api/v1/$resource_name`. Odpovědi (response) na HTTP požadavky budou mít nastavenou *Content Type* hlavičku na hodnotu `application/json`.

  Navržené RESTové rozhraní je složeno celkem ze 7 endpointů, 6 endpointů slouží k CRUD operacím (resources) a 1 endpoint k autentizaci a odhlášení uživatele. Pro entitu *Token* není endpoint navržen. Důvodem je, že se jedná o objekt, který se vytváří při registraci nového uživatelského účtu nebo žádosti o obnovu hesla. Odstranění tokenu je následně provedeno po jeho expiraci, případně po úspěšné aktivaci účtu či obnově hesla. Tyto akce jsou prováděny automaticky aplikací bez nutnosti zásahu uživatele (administrátora).

  Pro každý endpoint rozhraní resourců je navrženo filtrování, vyhledávání a stránkování záznamů dle parametrů odeslaných v rámci GET požadavku na daný resource:

  - *offset* - pořadí, od kterého mají být záznamy odeslány, implicitně nastaveno na `0`
  - *max* - maximální počet záznamů k odeslání (velikost stránky), implicitně nastaveno na `100`
  - *sortBy* - název atributu podle kterého mají být atributy seřazeny, implicitně nastaveno na `id` entity
  - *order* - pro řazení vzestupně = `asc`, pro řazení sestupně = `desc`, implicitně nastaveno na `asc`

  Pro filtrování a vyhledávání záznamů parametr vždy odpovídá názvu atributu filtrované entity, implicitně není nastaven žádný filtr. Jednotlivé parametry pro filtrování a řazení je možné řetězit a filtrovat podle více atributů. Navržené endpointy pro vystavované REST rozhraní jsou následující:

- **User**

URL	Metoda	Popis
`/api/v1/user`	GET	endpoint vrací informace o uživatelich
	POST	vytvoření nového uživatelského účtu
`/api/v1/user/$id`	GET	získání detailních údajích o uživateli
	PUT	aktualizace atributů uživatele
	DELETE	zneplatnění uživatelského účtu
`/api/v1/user/passwordreset`	POST	odeslání žádosti o obnovu hesla (email)
`/api/v1/user/changepassword`	POST	změna uživatelského hesla
`/api/v1/user/whoami`	GET	získání údajů o autentizovaném uživateli

Tabulka 4.1. Popis endpointu pro entitu User

- **Record**

URL	Metoda	Popis
`/api/v1/record`	GET	získání údajů o přihláškách na události
	POST	vytvoření nové přihlášky na událost
`/api/v1/record/$id`	GET	získání detailních údajích o přihlášce na událost
	PUT	aktualizace atributů přihlášky - zápis docházky
	DELETE	odstranění přihlášky na událost - odhlášení z události

Tabulka 4.2. Popis endpointu pro entitu Record.

- **Event**

URL	Metoda	Popis
`/api/v1/event`	GET	získání údajů o vytvořených událostech
	POST	vytvoření nové události
`/api/v1/event/$id`	GET	získání detailních údajů o události a přihlášky
	PUT	aktualizace atributů události
	DELETE	odstranění události, zrušení události
`/api/v1/event/repeatable`	POST	vytvoření opakovatelné události

Tabulka 4.3. Popis endpintu pro entitu User.

- **Label**

URL	Metoda	Popis
`/api/v1/label`	GET	získání údajů uložených štítků (kategorií)
	POST	vytvoření (přidání) nové kategorie
`/api/v1/label/$id`	GET	získání údajů o konkrétním štítku (kategorii)
	PUT	aktualizace atributů štítku (kategorie) - změna názvu
	DELETE	odstranění štítku (kategorie)

Tabulka 4.4. Popis endpintu pro entitu Label.

- **Room**

URL	Metoda	Popis
`/api/v1/room`	GET	získání údajů uložených místností
	POST	vytvoření nové místnosti
`/api/v1/room/$id`	GET	získání údajů o konkrétní místnosti
	PUT	aktualizace atributů místnosti - změna názvu, umístění
	DELETE	odstranění místnosti

Tabulka 4.5. Popis endpintu pro entitu Room.

- **Place**

URL	Metoda	Popis
`/api/v1/place`	GET	získání údajů uložených umístění
	POST	vytvoření nového umístění
`/api/v1/place/$id`	GET	získání detailních údajů o umístění - včetně místností
	PUT	aktualizace atributů umístění
	DELETE	odstranění umístění

Tabulka 4.6. Popis endpintu pro entitu place.

- **Autentizace**

URL	Metoda	Popis
`/login/authenticate`	POST	autentizace uživatele - přihlášení
`/logout/index`	POST	odhlášení autentizovaného uživatele

Tabulka 4.7. Popis endpintu pro autentizaci (přihlášení) uživatele.

- **View (prezentační vrstva)**

  Část View není u serverové části aplikace navržena pomocí `.gsp` souborů, ale je realizována klientskou JavaScriptovou aplikací. JavaScriptová aplikace bude poskytnuta Main controllerem na URL `/`.

- **Grails Service**

  Dále je mimo MVC architekturu frameworku Grails navržena implementace následujících servis. Jedná se o třídy zajišťující business logiku, která není součástí Controllerů a doménových objektů Modelu.

  - *MailingService* - služba zajišťující odesílání emailů dle funkčního požadavku F6, viz. 3.1.6.
  - *EventStateWatcher* - služba, která periodicky provádí kontrolu stavu událostí dle stavového diagramu entity Event uvedeném v předchozí podkapitole 4.2.
  - *TokenWatcher* - služba, která periodicky provádí kontrolu expirace aktivačních tokenů a tokenů pro obnovu zapomenutého hesla

4.3.2 Klientská část

Pro implementaci klientské části aplikace v podobě JavaScriptové aplikace byly využity JavaScriptové knihovny BackboneJS a MarionetteJS. Výběru technologie a její charakteristice je věnována sekce 4.1.2.

- **Backbone.Model (datová vrstva)**

  Pro každou entitu doménového modelu 4.3 s vystaveným REST endpointem je navržen potomek třídy Backbone.Model. Každá třída má specifikovaný atribut `urlRoot`. Hodnota tohoto atributu odpovídá cestě k hlavnímu endpointu dle návrhu vystavovaného API, viz. sekce 4.3.1 - např. pro entitu User je atribut `urlRoot` roven hodnotě `/api/v1/user`. Na tuto URL jsou následně odesílány HTTP požadavky volané pomocí metod `fetch()`, `create()`, `save()` a `destroy()` třídy Backbone.Model. Tyto třídy zajišťují synchronizaci s datovou vrstvou na straně serveru.

  Odesílané HTTP požadavky budou mít nastavenu hlavičku `Accept` na hodnotu `application/json`. Požadavek odeslaný metodou POST bude mít dále nastavenou hlavičku `Content-type` také na hodnotu `application/json`. Data odesílaná metodami POST a PUT ve formátu JSON budou v případě primitivních datových typů obsahovat hodnotu určenou k uložení do perzistentního úložiště. V případě komplexního datového typu parametru (např. umístění pro místnost) bude přenášeno pouze jeho id (pouze id umístění).

- **Backbone.Router (směrování URL)**

  Pro splnění funkčních požadavků byly identifikovány obrazovky, definované v tabulce 4.8. Dále jsou v tabulce definovány relativní URL adresy, na které jsou následující obrazovky mapovány.

#	URL	Popis
1	`/#`	úvodní stránka
2	`/#login`	stránka s přihlašovacím formulářem
3	`/#logout`	provede odhlášení a přesměrování na úvodní stránku
4	`/#password_reset`	stránka s formulářem pro zadání emailu
5	`/#registration`	stránka obsahující formulář pro registraci uživatele
6	`/#reset_token/$token`	stránka umožňující nastavení nového hesla
7	`/#myaccount`	editace uživatelských údajů přihlášeného uživatele
8	`/#myaccount/password`	změna hesla přihlášeného uživatele
9	`/#event`	zobrazení vytvořených událostí
10	`/#event/$id`	zobrazení detailu události dle zadaného id události
11	`/#event/create`	stránka s formulářem pro vytvoření jednorázové události
12	`/#event/createrepeatable`	stránka s formulářem pro vytvoření opakovatelné události
13	`/#place`	zobrazení vytvořených umístění
14	`/#place/$id`	zobrazení detailu umístění dle zadaného id umístění
15	`/#place/create`	stránka s formulářem pro vytvoření nového umístění
16	`/#room`	zobrazení vytvořených místností
17	`/#room/$id`	zobrazení detailu (editace) místnosti dle zadaného id
18	`/#room/create`	stránka s formulářem pro vytvoření nové místnosti
19	`/#user`	zobrazení registrovaných uživatelů
20	`/#user/$id`	zobrazení detailu registrovaného uživatele
21	`/#user/create`	vytvoření nového uživatelského účtu - administrace
22	`/#label`	zobrazení vytvořených labelů (kategorií) pro události
23	`/#label/$id`	zobrazení detailu (editace) místnosti dle zadaného id
24	`/#label/create`	stránka s formulářem pro vytvoření labelu (kategorie)

Tabulka 4.8. Mapování URL pomocí Backbone.Router

- **Marionette.View (prezentační a business logika)**

  Návrh potomků třídy Marionette.View je proveden způsobem popsaným tabulkou 4.9 - ve většině případů je 1 obrazovka renderována 1 třídou View. Výjimka je v případě, že 1 obrazovka musí realizovat více různých způsobů zobrazení dat (např. detail umístění - administrátor možnost editace, ostatní uživatelé pouze readonly náhled), je pro tuto obrazovku vytvořeno více tříd pro renderování View a podle uživatelské role aktuálně přihlášeného uživatele je vybráno příslušné View. Toto rozhodování dle uživatelské role je součástí mapování v Backbone.Router. Čísla v prvním sloupci tabulky odpovídají číslům URL definovaným v tabulce 4.8.

#	Role uživatele	Popis obrazovky
1	■	úvodní stránka obsahující popis portálu
2	neautentizovaný	přihlašovací formulář pro zadání username a hesla
3	autentizovaný	odhlášení a přesměrování na úvodní stránku
4	neautentizovaný	formulář pro zadání registrovaného emailu
5	neautentizovaný	formulář pro registraci uživatele s rolí USER
6	neautentizovaný	formulář pro zadání nového hesla po žádosti o obnovu
7	autentizovaný	předvyplněný formulář s údaji přihlášeného uživatele
8	autentizovaný	formulář pro zadání nového hesla přihlášeného uživatele
9	[neautentizovaný, USER]	tabulka s vytvořenými událostmi a filtrovací formulář
	[TRAINER, ADMIN]	odkazy pro přidání jednorázové a opakovatelné události
10	neautentizovaný	údaje o události, výzva k přihlášení do aplikace
	[USER, TRAINER]	údaje o události, tlačítka na login a logout na událost
	[TRAINER-creator, ADMIN]	formulář s údaji o události, formulář pro zadání docházky
11	[TRAINER, ADMIN]	formulář pro vytvoření jednorázové události
12	[TRAINER, ADMIN]	formulář pro vytvoření opakovatelné události
13	[neauten., USER, TRAINER]	zobrazení vytvořených umístění
	ADMIN	přidán odkaz pro přidání nového umístění
14	[neauten., USER, TRAINER]	definiční seznam s údaji o umístění
	ADMIN	předvyplněný formulář s údaji o úmístění
15	ADMIN	formulář pro vytvoření nového umístění
16	ADMIN	tabulka s vytvořenými místnostmi a filtrovací formulář
17	ADMIN	předvyplněný formulář s údaji o místnosti
18	ADMIN	formulář pro vytvoření nové místnosti
19	[TRAINER, ADMIN]	tabulka s registrovanými uživateli a filtrovací formulář
20	[neauten., USER, TRAINER]	údaje o registrovaném uživateli
	ADMIN	předvyplněný formulář s údaji o registrovaném uživateli
21	ADMIN	formulář pro vytvoření nového účtu - administrace
22	ADMIN	tabulka s vytvořenými labely (kategorie) pro události
23	ADMIN	předvyplněný formulář s údaji o štítku (kategorii) - editace
24	ADMIN	formulář pro vytvoření nového štítku (kategorie)

Tabulka 4.9. Mapování Marionette.View na URL z tabulky 4.8 a roli uživatele

V případě, kdy je potřeba sdílet část View mezi několika různými View, budou tyto části refaktorovány do samostatných View, které bude možné renderovat jako Child View. Tento koncept bude použitý například u renderování menu a filtrovacích formulářů.

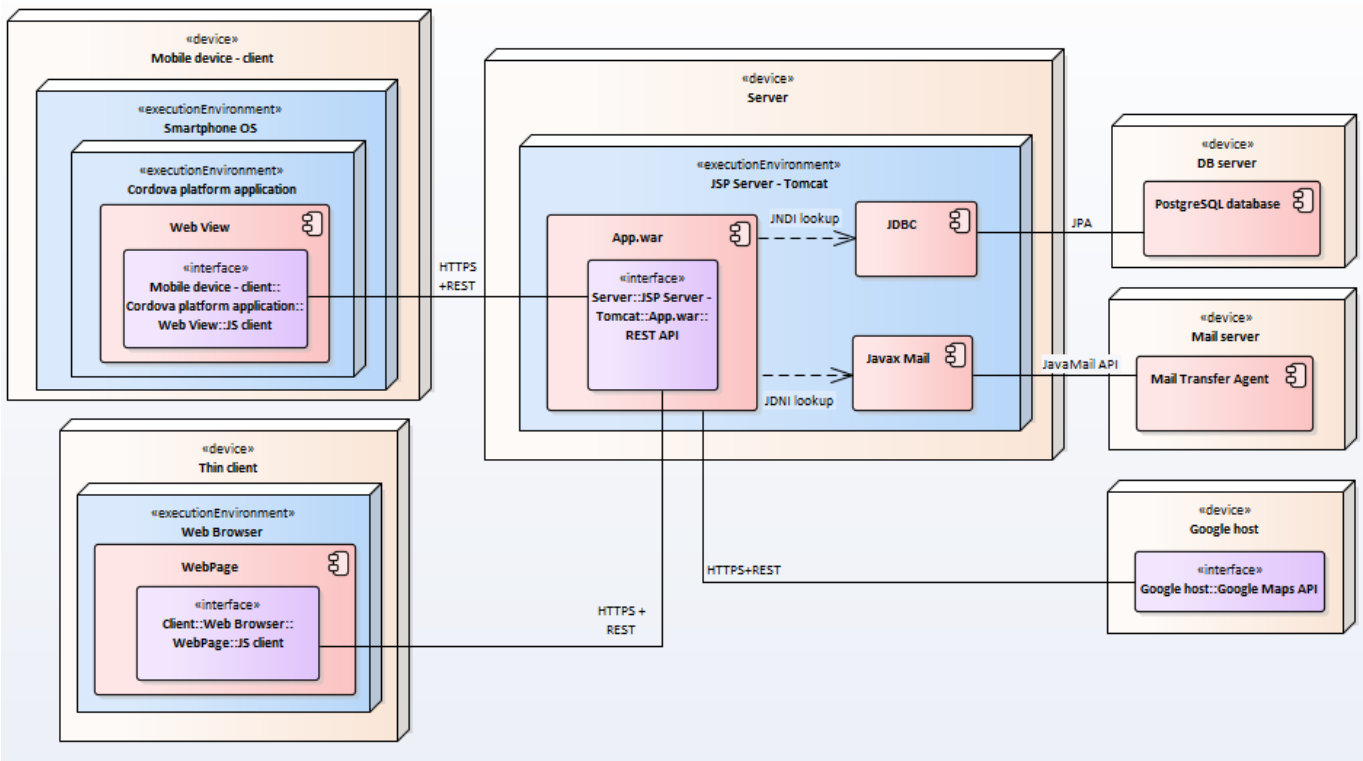
4.4 Diagram nasazení

Následující podkapitola popisuje nasazení aplikace na fyzická zařízení a způsob jejich komunikace. Diagram 4.5 zobrazuje rozdělení jednotlivých komponent systému a popisuje prostředí (platformu), ve kterém je komponenta provozována.

Serverová část je distribuována ve formě `.war` souboru, který je nasazený na aplikační server (JSP/servlet container). Webová aplikace vystavuje REST rozhraní pro komunikaci s klienty. Dalšími komponentami, které webová aplikace využívá jsou JNDI resource aplikačního serveru pro přístup k relačnímu databázovému úložišti a mailovému serveru. Poslední komponentou komunikující s webovou aplikací je Google maps api[1], pro zjištění GPS souřadnic z adresy nového umístění.

Tenký klient je reprezentován zařízením s běžícím webovým prohlížečem podporující HTML5 - viz. podporované prohlížeče nefunkčního požadavku N4 - 3.2.4. JavaScriptová aplikace komunikuje s webovou aplikací na straně serveru skrze REST rozhraní. Přenášená data jsou ve formátu JSON.

Diagram zobrazuje také část s mobilní aplikací. Vytvoření samotné mobilní aplikace není předmětem této práce, v diagramu nasazení aplikace je uveden pro úplnost kompletního softwarového řešení. Jako návrh možného řešení je uvedeno využití vývojového frameworku (platformy) Cordova a původní JavaScriptové aplikace jako u tenkého klienta.



Obrázek 4.5. UML diagram nasazení aplikace

4.5 Licenční podmínky

V následující podkapitole je provedena rešerše 4 nejčastěji používaných Open Source licenčních podmínek pro používání software a pojmů v nich využívaných. Tato část

[1] Dokumentace - `https://developers.google.com/maps/documentation/geocoding/start`

je v této práci zahrnuta z důvodu nefunkčního požadavku N3 uvedené v sekci 3.2.3. Z důvodu požadavku na bezplatné vytvoření a provozování aplikace byl směr zájmu zaměřen na technologie licencované jako Svobodný (Free) a Open Source software.

Svobodný software je dle [21] software, ke kterému lze svobodně přistupovat a používat. Oficiální definicí svobodného software je podle organizace Free Software Foundation splnění následujících podmínek [22]:

- Svoboda spustit program za jakýmkoliv účelem (svoboda 0).
- Svoboda studovat, jak program pracuje a přizpůsobit ho svým potřebám (svoboda 1). Předpokladem k výše uvedenému je přístup ke zdrojovému kódu.
- Svoboda redistribuovat kopie, abyste pomohli vašemu kolegovi (svoboda 2).
- Svoboda vylepšovat program a zveřejňovat zlepšení, aby z nich mohla mít prospěch celá komunita. (svoboda 3). Předpokladem k výše uvedenému je přístup ke zdrojovému kódu.

K uplatnění těchto základních svobod nesmí být stanovena povinnost žádat o povolení ani povinnost za něj platit [23]. Pro zajištění svobody svobodného software je využíván pojem Copyleft. Princip copyleft spočívá v takovém ustanovení licence, které zavazuje jejího nabyvatele, aby se šířeným software, ať už je pozměněn nebo nikoliv, poskytoval svobodu jej dále upravovat a šířit [23]. Tato povinnost se vztahuje i na díla, která jsou od tohoto díla odvozena [24]. Z této povinnosti plyne, že pokud jedna licence z použitých komponent obsahuje copyleft klauzuli, musí celé dílo poskytovat tyto svobody.

Pojmy Open Source a Svobodný software jsou často zaměňovány. Některé články uvádí jako rozdíl odlišnosti Open Source software od free software, že pro to, aby mohl být počítačový program považován za svobodný, musí umožňovat nabyvateli změny zdrojového kódu počítačového programu bez nutnosti jejich následného zveřejnění, zatímco definice Open Source software tento požadavek neobsahuje [25]. Definici pro Open Source spravuje organizace Open Source Initiative (OSI). Tato organizace na žádost provádí hodnocení licencí, zda splňují kritéria Open Source [23]. Pro splnění kritérií musí licence splňovat následujících 10 kritérií [26]:

1. Volná další distribuce
2. Distribuce zdrojového kódu
3. Umožnění vytváření odvozených děl
4. Integrita zdrojového kódu autora
5. Zákaz diskriminace osob nebo skupin
6. Zákaz diskriminace oblasti činnosti
7. Šířená licence platná pro všechny bez potřeby další licence
8. Licence nesmí záviset na určitém produktu (např. aby byl program součástí určité distribuce software)
9. Licence nesmí omezovat ostatní software
10. Licence musí být technologicky neutrální

Po schválení licence organizací OSI je znění licence přidáno na webové stránky organizace mezi ostatní Open Source licence.

4.5.1 Apache Licence

Jedná se o název pro licenci sloužící k licencování Open Source software potvrzenou organizací OSI. Licence neobsahuje copyleft klauzuli [27], tzn. odvozené dílo nemusí být celé svobodné. Apache licence umožňuje uživateli svobodně využívat software k různým účelům: distribuce původního díla, úprava díla, distribuce odvozeného díla - za dodržení podmínek z bodu 4 uvedených v [28]. Zatím poslední vydanou verzí je Apache 2 licence. Oproti předchozí verzi upravuje kompatibilitu s licenční smlouvou GPLv3.

4.5.2 BSD Licence

BSD licence je označení skupiny velmi jednoduchých permisivních softwarových licencí, původně určených pro účely unixového operačního systému BSD [25]. Jedná se o velmi volnou licenci - neomezuje šíření díla ani neomezuje omezení šíření díla autorem. Na začátku BSD licence je nutné uvést jméno/název autora a rok vytvoření programu, tzv. copyright doložku. Následují informace o licenci a zřeknutí se odpovědnosti za dílo, viz. znění licence dle OSI [29]. Licence neobsahuje copyleft klauzuli.

4.5.3 GNU General Public Licence

General Public Licence (zkráceně GPL) je licence pro svobodný software, původně určená pro projekt GNU. Licence je schválena organizací OSI pro licencování Open Source software. Postupem času byla licence upravována z důvodu reflektování aktuálně platných právních norem. Aktuálně používanou verzí je GPLv3, vydána v roce 2007 [23]. Z hlediska obsahu jsou licenční podmínky GNU GPL považovány za jedny z nejpřísnějších licenčních podmínek (z pohledu nabyvatele) používaných v oblasti free software či Open Source software [25]. Jedná se totiž o copyleftovou licenci [23], která vynucuje šíření této licence i na díla odvozená.

4.5.4 MIT Licence

Jak napovídá název licence, její text vznikl na Massachusetts Institute of Technology [25]. Software licencovaný touto licencí je možné použít jak v software s uzavřeným kódem tak i s GPL licencí. Obsah je velice podobný BSD licenci, hlavním rozdílem je, že licenční podmínky se vztahují i na dokumentaci softwaru a ne pouze na samotný software [25].

4.5.5 Použité frameworky, pluginy a knihovny

V uvedené tabulce 4.10 jsou definovány všechny použité komponenty implementované webové aplikace, jejich verze a použité licenční podmínky. Tyto knihovny a frameworky nebyly v průběhu implementace nijak upravovány a jedná se o původní produkty bez modifikací.

U všech použitých komponent vyvinuté aplikace jsou použity licence schválené organizací OSI pro Open Source software, konkrétně Apache 2 a MIT licence. Žádná z licenčních podmínek komponent systému neobsahuje copyleft klauzuli, tzn. není vynucené licencování výsledné aplikace pod stejnou licencí a dodržení svobody softwaru výsledného díla a je tedy možné výsledný produkt provozovat bez licenčních poplatků a bez nutnosti otevřít zdrojové kódy.

Jméno a verze frameworku/knihovny	Použitá licence
Grails 3.2.4	Apache 2
Spring Security Core Plugin 3.1.1	Apache 2
Backbone 1.3.3	MIT Licence
Backbone Syphon 0.6.3	MIT Licence
Marionette 3.1.0	MIT Licence
jQuery 3.1.1	MIT Licence
jQuery Chosen 1.4.2	MIT Licence
jQuery Simple pagination 1.6	MIT Licence
Bootstrap 3.3.7	MIT Licence
Bootstrap datepicker 1.6.4	Apache 2
Bootstrap datetimepicker 4.17	MIT Licence
MomentJS 2.18	MIT Licence
Underscore 1.8.3	MIT Licence
Vis.js 4.18	MIT Licence
RequireJS 2.3.2	MIT Licence

Tabulka 4.10. Licenční podmínky použitých komponent v implementované webové aplikaci.

Kapitola 5
Implementace

Tato kapitola popisuje použité vývojové prostředí a postup, jakým probíhala implementace jednotlivých částí softwaru a jaké implementační problémy byly řešeny.

5.1 Vývojové prostředí

Serverová i klientská část aplikace byla vyvíjená ve vývojovém prostředí *IntelliJ IDEA*. Pro správu zdrojových kódů byl využit verzovací nástroj *GIT* s repozitáři umístěnými na *GitLab* serveru FEL ČVUT. Databázové úložiště pro účely vývoje a následného uživatelského testování bylo zvoleno *PostgreSQL* ve verzi 9.5. Aplikace byla vyvíjena a testována na aplikačním serveru (servlet kontejneru) Tomcat 8.5.

5.2 Implementace backendu

Serverová část aplikace je založena na frameworku Grails ve verzi 3.2.4. Volbou tohoto frameworku je dána architektura aplikace a struktura balíčků s jednotlivými třídami. Následující sekce se věnují popisu struktury balíčkům a jednotlivým třídám, které definují doménový model - 5.2.1, controllery - 5.2.2, view - 5.2.3 a service - 5.2.4.

5.2.1 Doménové třídy

Implementovaný doménový model je součástí balíčku `cvut.fel.organizer` a obsahuje následující třídy a výčtové typy:

- Třídy, reprezentující stejně pojmenované entity doménového modelu - `Event`, `Label`, `Place`, `Record`, `Room`, `Token`, `User`
- Výčtové datové typy (enum), reprezentující stejně pojmenované entity doménového modelu - `EventState`, `Roles`, `TokenType`
- Třídy vzniklé dekompozicí doménového modelu
  - `Role` - Třída vyžadovaná Spring Security Pluginem pro definování entity Role - plugin vyžaduje definování role pomocí třídy, výčtový datový typ může být použitý pouze jako atribut pro název role.
  - `UserRole` - Třída vzniklá dekompozicí M:N relace mezi třídami `User` a `Role`. Dekompozice na dvě relace 1:N je vyžadována použitím Spring Security Pluginu.

Součástí implementace doménových tříd je i definování podmínek (*Constraints*) pro validaci doménových entit. Implementace doménového modelu byla provedena pomocí přístupu Test Driven Development (TDD), kdy jsou nejprve definovány jednotkové testy a následuje implementace jednotlivých tříd za účelem splnění již napsaných testů. Popisu provedeným jednotkovým testům je věnována kapitola 6.1.

5.2.2 Controller

Třídy controllerů jsou rozděleny do dvou balíčků - `cvut.fel.organizer.api.v1` a `cvut.fel.organizer`.

- Abstraktní třída `RestfulController` poskytující obecný předpis pro controllery z balíčku `cvut.fel.organizer.api.v1` - CRUD operace skrze vystavené REST rozhraní. Definované metody jsou:
  - `save()` - Provede uložení nového záznamu. V případě úspěšného provedení operace je návratový kód 201 (`Created`) a klientovi je vrácena uložená entita v podobě dat v JSON formátu. Pokud provedení operace není možné úspěšně dokončit z důvodu chybného nastavení některého z atributů (např. při registraci uživatele je zadáno již použité uživatelské jméno), je klientovi vrácen návratový kód 422 (`Unprocessable Entity`) a pole objektů `Error` ve formátu JSON. Objekt typu `Error` obsahuje 2 atributy - název chybně validovaného atributu, řetězec popisující chybu při validace entity.
  - `update()` - Provede aktualizaci údajů již uloženého záznamu s id odeslaném jako součást URL. Pokud záznam s daným id není nalezen, je klientovi vrácen návratový kód 404 (`Not Found`). V případě úspěšného provedení operace je návratový kód 200 (`OK`) a klientovi je vrácena uložená entita v podobě dat v JSON formátu. V případě neúspěšné validace aktualizovaných atributů je postupováno stejným způsobem jako u metody `save()`.
  - `delete()` - Provede odstranění záznamu dle id, které je součástí URL požadavku. Po úspěšném odstranění záznamu je klientské straně odeslána odpověď s návratovým kódem 204 (`No Content`). Stejný návratový kód je vrácen i v případě volání požadavku na záznam s neexistujícím id.
  - `index()` - Vrátí odpověď klientské straně se seznamem všech záznamů dané entity (resource) v rozmezí od (`offset`) do (`offset + max`) ve formátu JSON.
  - `show()` - Vrátí záznam s id dle URL požadavku ve formátu JSON. Pokud záznam s daný id není nalezen, je klientské straně odeslána odpověď s návratovým kódem 404 (`Not Found`).
- Třídy balíčku `cvut.fel.organizer.api.v1` - `EventController`, `LabelController`, `PlaceController`, `RecordController`, `RoomController`, `UserController`. Jedná se o potomky abstraktní třídy `RestfulController`. Uvedené třídy poskytují endpointy pro RESTové rozhraní, popsané v části 4.3.1. Controllery `EventController` a `UserController` obsahují kromě implementace obecného předpisu metod controlleru `RestfulController` další metody pro vytvoření endpointů dle 4.3.1:
  - `EventController` - metoda `repeatable()`, zajišťující vytváření opakovatelných událostí.
  - `UserController` - metody `activation()` pro aktivaci uživatelského účtu po registraci, `passwordReset()` pro přijetí požadavku o obnovu zapomenutého hesla, `changePassword()` pro změnu zapomenutého hesla a `whoami()` pro zjištění identity klientské strany.
- Třída `MainController` - controller sloužící k poskytnutí JavaScriptové aplikace na relativní URL `/`.

Controller	Metoda	Přístupné role uživatelů
EventController	`index()`	neautentizovaný
	`show()`	*
	`save()`	[ROLE_TRAINER, ROLE_ADMIN]
	`update()`	[ROLE_TRAINER - creator, ROLE_ADMIN]
	`delete()`	[ROLE_TRAINER - creator, ROLE_ADMIN]
	`repeatable()`	[ROLE_TRAINER, ROLE_ADMIN]
LabelController	`index()`	*
	`show()`	*
	`save()`	[ROLE_ADMIN]
	`update()`	[ROLE_ADMIN]
	`delete()`	[ROLE_ADMIN]
PlaceController	`index()`	*
	`show()`	*
	`save()`	[ROLE_ADMIN]
	`update()`	[ROLE_ADMIN]
	`delete()`	[ROLE_ADMIN]
RecordController	`index()`	autentizovaný
	`show()`	[record creator, event creator, ROLE_ADMIN]
	`save()`	[record creator, event creator, ROLE_ADMIN]
	`update()`	[ROLE_TRAINER, ROLE_ADMIN]
	`delete()`	[record creator, event creator, ROLE_ADMIN]
RoomController	`index()`	*
	`show()`	*
	`save()`	[ROLE_ADMIN]
	`update()`	[ROLE_ADMIN]
	`delete()`	[ROLE_ADMIN]
UserController	`index()`	[ROLE_TRAINER, ROLE_ADMIN]
	`show()`	*
	`save()`	neautentizovaný
	`update()`	[updated user, ROLE_ADMIN]
	`delete()`	[ROLE_ADMIN]
	`activation()`	neautentizovaný
	`passwordReset()`	neautentizovaný
	`changePassword()`	[updated user, neautentizovaný + token]
	`whoami()`	*
MainController	`index()`	*

Tabulka 5.1. Zabezpečení metod controllerů

Zabezpečení jednotlivých endpointů controllerů je provedeno dle tabulky 5.1. Označení povoleného přístupu pomocí `*` znamená autentizovaného i neautentizovaného uživatele.

V případě zaslání HTTP požadavku na zabezpečený endpoint (vyžadováno přihlášení) neautentizovaným uživatelem, je klientské straně odeslána odpověď s návratovým kódem 401 (`Unauthorized`) a přesměrování na přihlašovací stránku. Je-li uživatel autentizován a nemá dostatečná přístupová práva pro vykonání požadované operace, je klientské straně odeslána odpověď s návratovým kódem 403 (`Forbidden`).

5.2.3 View

Soubory `.gsp` představující View architektury frameworku Grails jsou využity ke dvěma účelům:

1. šablony pro emaily

Název souboru	Popis použití
AccountActivation.gsp	odeslání aktivačního odkazu po registraci uživatele
EventCanceled.gsp	email o zrušení události přihlášeným uživatelům
EventUpdated.gsp	email o změně údajů události přihlášeným uživatelům
PasswordReset.gsp	odeslání odkazu umožňující nastavení nového hesla
PasswordResetConfirm.gsp	potvrzovací email o úspěšné změně hesla

Tabulka 5.2. Šablony pro odesílání emailů

2. úvodní HTML stránka zajišťující načtení klientské JavaScriptové aplikace

5.2.4 Služby (Service)

Implementované třídy Grails Service navržené v části 4.3.1 jsou implementované v balíčku `cvut.fel.organizer.services`.

- `EventStateWatcherService` - Služba pro provádění kontrolu stavu událostí. Periodické spouštění je definováno pomocí Java EE anotace `@Scheduled` - periodicita spouštění tasku každých 5 minut.
- `TokenWatcherService` - Služba pro provádění periodické kontroly expirace tokenů. Periodické spouštění je definováno pomocí Java EE anotace `@Scheduled` - periodicita spouštění tasku každých 5 minut.
- `MailingService` - Služba zajišťující odesílání emailů definovaných v `.gsp` souborech, viz. sekce 5.2.3.

5.2.5 Implementační problémy

V průběhu návrhu a implementace aplikační logiky byly nalezeny dvě možné alternativy pro implementaci stavu události (doménová entita `Event`) - pomocí návrhového vzoru *state* nebo pomocí výčtového datového typu (enum). Po konzultaci s vedoucím práce bylo zvoleno řešení pomocí výčtového typu. Návrhový vzor *state* nebyl vybrán z důvodu požadovaného administrátorského účtu, pomocí kterého je umožněno spravovat přihlášky uživatelů na události nehledě na stav, ve kterém se entita události aktuálně nachází. V každé třídě konkrétního stavu by bylo nutné provádět kontrolu uživatelských rolí uživatele.

Během implementace tříd controllerů byla nalezena chyba ve frameworku Grails, konkrétně v metodě `getObjectToBind()`. Tato metoda umožňuje automatické mapování přijatých parametrů objektu `request` na atributy objektu doménové třídy. V původní implementaci této metody byl mapován na atributy objektu doménové třídy celý objekt `request`. Tento problém byl vyřešen předefinováním metody `getObjectToBind()`. Předefinovaná implementace této metody mapuje na atributy objektu doménové třídy JSON data přijatá v objektu `request` (konkrétně `request.JSON`).

5.3 Implementace frontendu

Klientská část aplikace je založena na JavaScriptových knihovnách BackboneJS a MarionetteJS. Následující sekce popisují členění, vzhled uživatelského rozhraní a adresářovou strukturu implementované klientské JavaScriptové aplikace. Struktura klientské aplikace odpovídá návrhu z kapitoly 4.3.2. JavaScriptové knihovny použité při implementaci klientské části aplikace jsou umístěny ve složce `/vendor`. Seznam použitých knihoven je součástí tabulky 4.10.

5.3.1 Backbone.Model

Potomci třídy Backbone.Model pro definici entit umožňující synchronizaci se serverovou částí skrze REST rozhraní jsou umístěny ve složce `/src/entity`. Soubory s definovanými třídami odpovídají poskytovaným resourcům na straně serveru: `Event.js`, `Label.js`, `Place.js`, `Record.js`, `Room.js` a `User.js`. Dále je implementována pomocná třída `MenuItem` v souboru `MenuItem.js`. Tato třída slouží k renderování položek menu dle konkrétní role přihlášeného uživatele.

5.3.2 Backbone.Router

Třída zajišťující směrování akcí dle URL je součástí souboru `OrganizerApp.js` a je umístěna ve složce `/src`. Pro přehlednost kódu této třídy je směrování refaktorováno do tříd controllerů umístěných ve složce `/src/controller`. Třída uvnitř souboru `OrganizerApp.js` obstarává pouze směrování dle první úrovně URL. Směrování akcí konkrétního zdroje (resource) je definováno uvnitř příslušného controlleru - například pro resource `Event` a URL `/#event/*` definuje směrování části URL označené symbolem `*` na jednotlivá View soubor `EventController.js`. Složka `/src/controller` obsahuje následující soubory:

- `EventController.js` - controller zajišťující směrování URL `/#event/*`
- `LabelController.js` - controller zajišťující směrování URL `/#label/*`
- `MainPageController.js` - controller zajišťující směrování URL `/#` na zobrazení úvodní stránky (MainPageView)
- `PlaceController.js` - controller zajišťující směrování URL `/#place/*`
- `RoomController.js` - controller zajišťující směrování URL `/#room/*`
- `UserController.js` - controller zajišťující směrování URL:
  - `/#user/*`
  - `/#myaccount/*`
  - `/#activation_token/*`
  - `/#login`
  - `/#registration`
  - `/#password_reset`
  - `/#reset_token/*`

5.3.3 Marionette.View

Soubory jednotlivých tříd View, dle návrhu v tabulce 4.9, jsou umístěny ve složce `/src/view`. Jedná se o potomky třídy Marionette.View, zajišťující renderování entit (viz. část 5.3.1) pomocí definovaných HTML šablon a obsluhu událostí (například submit formuláře).

5.3.4 Šablony

Šablony pro renderování View jsou refaktorovány do samostatných souborů s koncovkou `.jst`. Tyto soubory jsou umístěny ve složce `/src/template`. Umístěním HTML šablon do samostatných souborů umožňuje zpřehlednění kódu jednotlivých View - dlouhé HTML struktury nejsou součástí View.

5.3.5 Uživatelské rozhraní

Zadavatel neměl žádné požadavky na design uživatelského rozhraní. Jediným požadavkem bylo dodržení nefunkčního požadavku N4 - Responzivní uživatelské rozhraní 3.2.4. Při návrhu vzhledu GUI bylo vycházeno z CSS stylů uživatelského rozhraní Hello World aplikace frameworku Grails. Použité CSS styly a elementy využívají knihovny Bootstrap 3.

Rozložení stránky je provedeno do horizontálních bloků (jedno-sloupcový vertikální layout). Prvním horizontálním blokem je hlavička stránky, obsahující logo s názvem aplikace a uživatelské menu aplikace. Druhým blok obsahuje informace sdělované konkrétní stránkou (např. tabulka, formulář). Tento blok je v případě potřeby členěn na další horizontální bloky, konkrétní příklad - stránka se seznamem událostí: formulář pro filtrování záznamů, množina záznamů ve formě tabulky, elementy pro stránkování záznamů. Výhoda použití horizontálního členění se projeví při používání aplikace na mobilních zařízeních. Na malých obrazovkách je mnohem přirozenější scrollovat vertikálně než horizontálně [30]. Třetím blokem je patička stránky obsahující údaje o roku vytvoření stránky a jméno autora.

Použité barevné schéma kombinuje barevný odstín zelené (konkrétní hexadecimální hodnota #4D8618) s bílou. Pro font písma bylo zvoleno bezpatkové písmo Helvetica Neue. Pro zvolené písmo byly zvolené barvy černé (na bílém pozadí) a bílé (na zeleném pozadí).

Zvolený způsob prezentace dat pro implementaci vychází ze způsobu používaných v existujících řešeních, viz. kapitola 2. Pro zobrazení informací o seznamu událostí byly vybrány pro následující případy tyto možnosti:

- Pro obrazovku s vypsanými událostmi byl zvolen tabulkový výpis, který lze filtrovat pomocí formuláře a položky seznamu řadit vzestupně či sestupně dle termínu začátku a konce události, termínu konce přihlašování nebo názvu události. Stejný způsob byl vybrán i pro zobrazení přihlášených událostí uživatele a vytvořené události trenéra.
- Pro zobrazení volných termínů v jednotlivých místnostech při vytváření nových událostí byl vybrán způsob prezentace informací pomocí časové osy (Timeline). Časová osa je uživateli vytvářející novou událost zobrazena po stisku tlačítka *Zobrazit obsazenost místností* na spodní části stránky pod formulářem. Aplikace umožňuje zobrazení obsazenosti místností na konkrétním umístění v zadaném termínu.

Pro zobrazení detailních informací o konání události byla zvolena možnost prezentace pomocí samostatné obrazovky, obsahující definiční seznam s údaji události. Součástí této obrazovky je i možnost přihlášení (odhlášení) na událost.

5.3.6 Implementační problémy

Při implementaci jednotlivých Marionette.View byl nalezen problém při odesílání formulářů - formuláře byly odesílány vícenásobně. Problém vznikal následujícím postupem kroků:

1. Vyplnit formulář pro vytvoření nové události
2. Odeslat formulář
3. Po úspěšném vytvoření události vybrat možnost pro přidání další události
4. Opět vyplnit formulář pro vytvoření další události
5. Odeslat formulář
6. Odesílaný formulář byl odeslán dvakrát
7. Při opakování přidání další události byl formulář odeslán třikrát, atd.

Příčinou tohoto problému byl vznik tzv. zombie view [31, 18]. Při renderování nového *Marionette.View* zůstávaly registrovány události, které byly namapovány na akci submit formuláře. Řešení tohoto problému spočívalo v odregistrování těchto událostí před renderingem nového View. Toto odregistrování událostí bylo provedeno v metodě `initialize()`, která je defaultně volána mezi konstruktorem objektu a metody `render()`.

Při implementaci uživatelského rozhraní byl nalezen problém s responzivním designem tabulek. Pro korektní způsob zobrazení tabulek na mobilních zařízení byly nalezeny 3 možné řešení [32]. Jedná se o:

1. zjednodušení tabulky - vynechání některých sloupců
2. horizontální posuvník - šířka tabulky je zachována a je přidán posuvník pro horizontální scrollování
3. rozlámání tabulky - buňky řádku tabulky jsou uspořádány pod sebou (řádek tabulky je transformován do jedné buňky)

Z navrhovaných možností bylo vybráno rozlámání tabulky. Výhodou tohoto řešení je zachování informační hodnoty původního informačního zdroje a zachování konceptu vertikálního scrollování.

Kapitola 6
Testování

Následující kapitola popisuje proces testování implementované aplikace za účelem nalezení chyb při implementaci a kontrolu splnění požadavků specifikace. Proces testování je rozdělen na 4 části: jednotkové testy (podkapitola 6.1), uživatelské testování na testovacích scénářích (podkapitola 6.2), testování výkonnostních požadavků (podkapitola 6.3) a testování kompatibility s prohlížeči (podkapitola 6.4).

6.1 Jednotkové testy

Implementované jednotkové testy jsou rozděleny na dvě skupiny - testy doménových tříd a testy tříd controllerů. Testy doménových tříd jsou zaměřeny na implementaci integritních omezení. Celkem bylo realizováno 109 unit testů pro ověření definic *Constraint* bloků jednotlivých entit. Implementace těchto unit testů integritních omezení proběhla před samotnou implementací doménových tříd a dle požadavků specifikace. Následné *Constraint* bloky pro validaci entit byly implementovány vůči unit testům.

Jednotkové testy controllerů jsou zaměřeny na testování povolených metod HTTP požadavků odesílaných na jednotlivé přístupové body (endpointy) REST rozhraní. Testování povolených metod je nemožné provést pomocí uživatelského testování za pomoci klientské JavaScriptové aplikace. Alternativní možnost otestování controllerů je pomocí RESTového klienta (například v podobě pluginu RESTEasy do prohlížeče Firefox). V případě použití nedovolené metody HTTP požadavku je očekávána odpověď s návratovým kódem 405 (Method not allowed). Celkem bylo vytvořeno 99 unit testů pro controllery.

Celkem bylo pro účely testování serverové části aplikace vytvořeno 208 jednotkových testů.

6.1.1 Výsledky testování

Všechny implementované jednotkové testy skončily úspěšně. Report z průběhu jednotkových testů je obsahem tabulky 6.1.

Testy třídy	Testů	Chybných	Ignorováno	Trvání
cvut.fel.organizer.Event	15	0	0	2.191s
cvut.fel.organizer.Label	5	0	0	1.021s
cvut.fel.organizer.Place	35	0	0	1.255s
cvut.fel.organizer.Record	2	0	0	0.749s
cvut.fel.organizer.Role	8	0	0	1.373s
cvut.fel.organizer.Room	6	0	0	1.023s
cvut.fel.organizer.Token	7	0	0	0.985s
cvut.fel.organizer.User	31	0	0	1.073s
cvut.fel.organizer.api.v1.EventController	17	0	0	5.832s
cvut.fel.organizer.api.v1.LabelController	14	0	0	1.868s
cvut.fel.organizer.api.v1.PlaceController	14	0	0	1.770s
cvut.fel.organizer.api.v1.RecordController	14	0	0	1.269s
cvut.fel.organizer.api.v1.RoomController	14	0	0	1.056s
cvut.fel.organizer.api.v1.UserController	23	0	0	1.314s
cvut.fel.organizer.MainController	3	0	0	1.049s

Tabulka 6.1. Report z průběhu jednotkových testů

6.2 Uživatelské testování

Uživatelského testování aplikace se zúčastnili 3 participanti. Profily participantů jsou popsány v tabulce 6.2.

Participant 1	
Věk	25 let
Pohlaví	Muž
Povolání	Student informatiky
Setup	Notebook s Windows 10 (prohlížeče Edge, Internet Explorer 11, Chrome 56, Firefox 52), iPhone SE - iOS 10.3, Safari
Participant 2	
Věk	20 let
Pohlaví	Žena
Povolání	Studentka zdravotnických studií
Setup	Stolní počítač s Windows 10 (prohlížeče Edge, Internet Explorer 11, Chrome 56, Firefox 52), iPhone 6s - iOS 10.3, Safari
Participant 3	
Věk	23 let
Pohlaví	Muž
Povolání	Softwarový vývojář
Setup	Notebook s Windows 10 (Chrome 57, Firefox 53)

Tabulka 6.2. Profily účastníků uživatelského testování

Cílem uživatelského testování bylo ověření použitelnosti návrhu uživatelského rozhraní a nalezení chyb v implementaci aplikace. Testování probíhalo pomocí definovaných testovacích scénářů (viz. sekce 6.2.2). Testovací procedura probíhala na zařízeních a webových prohlížečích participantů - viz. tabulka 6.2.

6.2.1 Testovací data

Testovací data byla vytvořena pomocí JavaScriptové knihovny Faker.js[1]. Jedná se o knihovnu sloužící ke generování falešných údajů (např. jména uživatelů, adresy) použitelných pro účely testování aplikací. Údaje byly ve smyčce generovány do SQL `INSERT INTO` skriptů, které byly následně vloženy do databázového úložiště. Skripty obsahovaly celkem záznamy pro 1000 uživatelů (30 trenérských účtů, 970 běžných uživatelů), 10 umístění, 5 místností pro každé umístění a 10 kategorií. Rozsah importovaných testovacích záznamů odpovídá předpokládanému zatížení aplikace a nefunkčnímu požadavku N8 - Výkonnostní požadavky 3.2.8.

6.2.2 Testovací scénáře

Následující testovací scénáře jsou navrženy dle identifikovaných případů užití, kapitola 3.4.2. Definované testovací scénáře zajišťují pokrytí těchto případů užití. Pro každý testovací scénář jsou uvedeny počáteční podmínky (stav aplikace), kroky testovací procedury s očekávaným výsledkem a případy užití, které jsou testovacím scénářem pokryty.

- **TC1 - Registrace uživatele**
  - Počáteční stav: úvodní stránka, nepřihlášený uživatel
  - Kroky:
    1. vybrat možnost *Přihlášení a registrace - Registrace uživatele*
    2. vyplnit registrační formulář (ověřit chybné vstupy - chybný formát emailu, telefonního čísla, slabé heslo), po úspěšné registraci odeslán email na uvedenou adresu s odkazem pro aktivaci účtu
    3. pokusit se přihlásit - účet není aktivován, přihlášení se nezdařilo
    4. přistoupit na odkaz zaslaný emailem - uživatelský účet úspěšně aktivován (při opětovném přistoupení na tento odkaz zobrazena chybová hláška)
    5. přihlásit se zadanými údaji, po úspěšném přihlášení se odhlásit
  - Pokryty případy užití: UC1, UC2, UC3
- **TC2 - Obnova zapomenutého hesla**
  - Počáteční stav: úvodní stránka, nepřihlášený uživatel
  - Kroky:
    1. vybrat možnost *Přihlášení a registrace - Zapomenuté heslo*
    2. zadat do formuláře falešný email - `xxxxx@xxxxx.cz`, odeslat formulář, zobrazena potvrzující hláška o odeslání emailu (reálně žádný email není odeslaný)
    3. opakovat postup pro email uvedený při registraci uživatelského účtu, odeslán email s odkazem pro zadání nového hesla
    4. přistoupit na zaslaný odkaz a zadat do formuláře nové heslo

[1] `https://github.com/marak/faker.js/`

5. při pokusu o opětovnou změnu hesla stejným odkazem je zobrazena chybová hláška

- Pokryty případy užití: UC4

TC3 - Přihlášení na událost, odhlášení z události

- Počáteční stav: přihlášený uživatel s rolí `ROLE_USER`
- Kroky:

1. vybrat v menu možnost *Události*
2. vybrat ze seznamu událostí jednu, na kterou je zpřístupněné přihlašování a zobrazit její detail
3. stisknout tlačítko *Přihlásit se na událost* - zobrazeno potvrzení o přihlášení na událost, tlačítko nahrazeno tlačítkem *Odhlásit z události*, zvýšen počet přihlášených účastníků
4. vybrat v menu položku *Můj účet* - *Přihlášené události* - v seznamu událostí je zobrazena přihlášená událost, vrátit se na detail události stiskem tlačítka *Zobrazit detail*
5. stisknout tlačítko *Odhlásit z události* a potvrdit odhlášení v dialogovém okně - zobrazeno potvrzení o odhlášení z události, tlačítko *Odhlásit z události* nahrazeno tlačítkem *Přihlásit se na událost*, snížen počet přihlášených účastníků
6. přejít zpět do seznamu událostí (bod 1.)
7. zobrazit detail události, na kterou není zpřístupněné přihlašování - místo tlačítek k přihlašování/odhlašování je zobrazeno *Přihlašování na událost není zpřístupněno*
8. zobrazit detail události s obsazenou kapacitou - místo tlačítek k přihlašování/odhlašování je zobrazeno *Kapacita události je plně obsazena*

- Pokryty případy užití: UC5, UC6, UC7, UC9, UC14

TC4 - Filtrování a vyhledávání záznamů

Následující testovací scénář je postupně proveden na obrazovkách se seznamem záznamů o událostech, umístěních, místnostech, uživatelích a kategoriích (položky v menu *Události, Umístění, Místnosti, Uživatelé a Kategorie*).

- Počáteční stav: uživatel přihlášený administrátorským účtem
- Kroky:

1. vybrat v menu příslušný druh záznamů k testování (*Události, Umístění, Místnosti, Uživatelé, Kategorie*)
2. vybrat v podmenu možnost *Filtrovat* - aplikace zobrazí formulář pro filtrování záznamů dané entity
3. vyplnit vždy jeden atribut pro filtrování záznamů a odeslat formulář stiskem tlačítka *Filtrovat* (opakovat pro každý filtrovací atribut) - aplikace zobrazí záznamy odpovídající danému nastavení filtrů

4. vyplnit všechny filtrovací atributy a odeslat formulář stiskem tlačítka *Filtrovat* - aplikace zobrazí záznamy odpovídající danému
5. k vyplněnému filtrovacímu formuláři z bodu 4. přidat možnost řazení výsledků (ověřit všechny možnosti řazení), tzn. všechny filtrovací atributy + řazení výsledků - aplikace zobrazí záznamy odpovídající danému nastavení filtrů a řazení

- Pokryty případy užití: UC7, UC8

TC5 - Tvorba událostí

- Počáteční stav: přihlášený uživatel s rolí `ROLE_TRAINER`
- Kroky:

1. vybrat v menu možnost *Události*
2. vybrat v podmenu možnost *Přidat jednorázovou událost*
3. vyplnit formulář nové události (ověřit chybné vstupy - záporná kapacita, vyšší počet minimálních účastníků než kapacita, konec akce před začátkem akce, nevybrání umístění a místnosti konání) - po úspěšném odeslání formuláře a vytvoření události je uživatel přesměrován na stránku se seznamem událostí
4. vybrat v podmenu možnost *Přidat jednorázovou událost*
5. vyplnit formulář nové události, nastavit čas a místo konání na stejnou hodnotu jako v bodě 3. - aplikace zobrazí chybovou hlášku o obsazenosti místnosti v daném čase
6. změnit čas nebo místo konání a znovu odeslat formulář - po úspěšném odeslání formuláře a vytvoření události je uživatel přesměrován na stránku se seznamem událostí
7. vybrat v podmenu možnost *Přidat opakovanou událost*
8. vyplnit formulář nové opakovatelné události (ověřit chybné vstupy jako v budě 2.), pravidelnost událostí zvolit *Denně*, mezi atributy *Plánovat od* a *Plánovat do* nastavit alespoň 2 dny - po úspěšném odeslání formuláře a vytvoření událostí v daném intervalu je uživatel přesměrován na stránku se seznamem událostí
9. opakovat bod 8. pro pravidelnosti *Každý týden, Sudý týden, Lichý týden*

- Pokryty případy užití: UC7, UC10, UC11

TC6 - Zrušení a editace událostí

- Počáteční stav: přihlášený uživatel s rolí `ROLE_TRAINER`
- Kroky:

1. vybrat v menu možnost *Události*
2. vybrat v podmenu možnost *Filtrovat události* - aplikace zobrazí formulář pro filtrování událostí
3. vyfiltrovat události dle tvůrce události (nastavit atribut na přihlášeného uživatele)
4. vybrat ze seznamu událost před datem konání, stisknout tlačítko *Zobrazit detail*

5. upravit v předvyplněném formuláři údaje o konání události, odeslat formulář stiskem tlačítka *Upravit událost* - při vyplnění korektních údajů je zobrazeno oznámení o úspěšné aktualizaci údajů a odeslán email účastníkům přihlášených na událost
6. vybrat v menu možnost *Události*
7. vybrat v podmenu možnost *Filtrovat události* - aplikace zobrazí formulář pro filtrování událostí
8. vyfiltrovat události dle tvůrce události (nastavit atribut na přihlášeného uživatele)
9. vybrat ze seznamu událost před datem konání, stisknout tlačítko *Zobrazit detail*
10. stisknout tlačítko *Zrušit událost* - po potvrzení dialogového okna je událost odstraněna a poté odeslán email přihlášeným účastníkům o zrušení události a následuje přesměrování na obrazovku se seznamem událostí
11. vybrat ze seznamu událost, jejíž tvůrce je jiný uživatel než aktuálně přihlášený, stisknout tlačítko *Zobrazit detail* - aplikace zobrazí údaje o události bez možnosti editace

- Pokryty případy užití: UC7, UC8, UC12, UC13, UC14, UC15, UC16

TC7 - Zápis docházky

- Počáteční stav: přihlášený uživatel s rolí `ROLE_TRAINER`
- Kroky:

1. vybrat v menu možnost *Události*
2. vybrat v podmenu možnost *Filtrovat události* - aplikace zobrazí formulář pro filtrování událostí
3. vyfiltrovat události dle tvůrce události (nastavit atribut na přihlášeného uživatele)
4. vybrat ze seznamu událost po datu konání, stisknout tlačítko *Zobrazit detail* - v případě předchozího vyplnění docházky je docházkový formulář v detailu události předvyplněn, jinak jsou všichni přihlášení účastníci označeni jako nepřítomní
5. vyplnit údaje o docházce a odeslat formulář - po úspěšném uložení docházky je přihlášený uživatel informován potvrzující hláškou

- Pokryty případy užití: UC7, UC14, UC16, UC17, UC18

TC8 - Administrace

Následující testovací scénář je postupně proveden na obrazovkách se seznamem záznamů o událostech, umístěních, místnostech, uživatelích a kategoriích (položky v menu *Události, Umístění, Místnosti, Uživatelé a Kategorie*). Pro každou entitu doménového modelu je ověřena možnost veškerých CRUD operací. Součástí tohoto testovacího scénáře jsou předchozí testovací scénáře TC3 až TC7.

- Počáteční stav: uživatel přihlášený administrátorským účtem

- Kroky:

  1. vybrat v menu příslušný druh záznamů k testování (*Události, Umístění, Místnosti, Uživatelé, Kategorie*)
  2. pro každou administrovanou entitu vytvořit nový záznam, tzn. novou událost, umístění, místnost, uživatele a kategorii (u každé entity ověřit kontrolu chybných vstupů)
  3. pro každou administrovanou entitu zobrazit detail umožňující editaci údajů
  4. pro každou administrovanou entitu provést úpravu údajů (editaci), u události navíc přidat a odebrat přihlášku uživatele na událost, u uživatele provést úpravu přiřazených uživatelských rolí (uživateli s rolí `ROLE_USER` přidat a následně odebrat roli `ROLE_TRAINER`)
  5. pro každou administrovanou entitu provést smazání záznamu vytvořeného v kroku 2, v případě mazání uživatelského účtu provést kontrolu zneplatnění záznamu v tabulce databáze

- Pokryty případy užití: UC19, UC20, UC21

TC9 - Editace uživatelského účtu

- Počáteční stav: autentizovaný uživatel
- Kroky:

  1. vybrat v menu možnost *Můj účet - Změnit údaje*
  2. upravit v předvyplněném formuláři údaje o jménu, příjmení a telefonním čísle, potvrdit změnu zadáním hesla a odeslání formuláře - při zadání korektních vstupů je zobrazeno oznámení o úspěšné aktualizaci údajů

- Pokryty případy užití: UC22

6.2.3 Výsledky testování

V průběhu testování nenastal po úvodním seznámení s aplikací u žádného participanta problém s uživatelským rozhraním a jeho designem. Při testování dle testovacích scénářů byly nalezeny 2 chyby:

1. Filtrování a řazení uživatelských účtů

   Při filtrování uživatelů dle role a požadavku na seřazení záznamů (dle jména, příjmení nebo uživatelského jména) nebyly výsledné záznamy seřazeny a jejich pořadí bylo náhodné (testovací scénář TC4).

2. Chybějící překlad chybové hlášky

   Při nevybrání místa konání a následného odeslání formuláře pro vytvoření nové události byla vypisována generická chybová hláška o neúspěšné validaci entity pomocí GORM (testovací scénář TC5 a TC6).

Na podnět neúspěšných testů byly tyto chyby v implementaci opraveny. Po této opravě všechny testovací scénáře byly provedeny úspěšně.

6.3 Testování výkonnostních požadavků

Testování výkonnostních požadavků bylo provedeno z důvodu ověření splnění nefunkčního požadavku N8 - Výkonnostní požadavky 3.2.8. Požadavkem je průměrná doba odezvy backendové části aplikace s testovacími daty do 500 ms. Doba odezvy byla měřena pomocí konzole webového prohlížeče. Měření průměrné odezvy probíhalo s přihlášeným administrátorským účtem a se záznamy o uživatelských účtech (důvodem je nejvyšší počet aplikovatelných filtrů a počet záznamů v testovacích datech). Hardwarová a softwarová konfigurace testovacího serveru je popsána v tabulce 6.3.

Operační systém	Windows 10
Java	JDK 1.8
Aplikační server	Apache Tomcat 8.5
Databáze	PostgreSQL 9.5
Procesor	Intel Core i7 860, 2.8 GHz
Operační paměť RAM	12 GB
Úložiště	SSD disk, 256 GB

Tabulka 6.3. Konfigurace serveru pro účely výkonnostních testů backendu

Průměrná doba odezvy byla měřena z deseti opakovaných pokusů každého testovacího případu. Měření průměrné doby odezvy probíhalo na následujících případech:

- Nefiltrovaný výpis a stránkování - 150 ms
- Řazení - 150 ms
- Vyhledávání a filtrování - dle osobních údajů uživatele 50 ms, dle role uživatele 470 ms
- Kombinace stránkování, vyhledávání a řazení - včetně filtrování dle role 520 ms
- Celkový průměr - 325 ms

Výsledné doby odezvy pro jednotlivé testovací případy splňují požadovanou dobu odezvy, vyjma posledního testovacího případu. Tento případ přesahuje požadavek o 20 ms. Průměrná doba odezvy všech testovacích případů je pod požadovanou hodnotou.

6.4 Testování kompatibility prohlížečů

Testování kompatibility klientské JavaScriptové aplikace bylo provedeno s prohlížeči Mozilla Firefox 52, Chrome 56, Internet Explorer 11, Edge a na mobilních zařízeních iPhone prohlížeč Safari. Podpora těchto prohlížečů je součástí nefunkčního požadavku N4 - 3.2.4. Testování kompatibility bylo zajištěno provedením testovacích scénářů ze sekce 6.2.2 se všemi popsanými prohlížeči. Úspěšnost testů byla posuzována vizuální podobou stránek, očekávaným chováním aplikace na podněty uživatele a absencí chybových hlášek ve vývojové konzoli desktopových prohlížečů. Obrázek 6.1 zobrazuje jednotný vzhled webové aplikace v desktopových prohlížečích a responzivitu vzhledu při používání mobilních zařízeních.

Obrázek 6.1. Zobrazení stránky ve webových prohlížečích

Kapitola 7
Zhodnocení splnění specifikace

Splnění jednotlivých bodů specifikace implementované aplikace je ověřeno úspěšným provedením všech unit testů a uživatelských testů. Ověření splnění jednotlivých bodů požadavků je zajištěno:

- **F1 - Registrace uživatelů** - uživatelské testování (testovací scénáře TC1, TC8, TC9) a unit testy ověřující implementaci constraint bloku u třídy User
- **F2 - Obnova zapomenutého hesla** - uživatelské testování (testovací scénář TC2)
- **F3 - Autentizace a autorizace** - uživatelské testování (testovací scénář TC1)
- **F4 - Vytváření a editace události** - uživatelské testování (testovací scénáře TC5, TC6, TC8) a unit testy ověřující implementaci constraint bloku u třídy Event
- **F5 - Zrušení události** - uživatelské testování (testovací scénář TC6)
- **F6 - Zasílání emailů** - uživatelské testování (testovací scénáře TC1, TC2, TC6)
- **F7 - Zaznamenávání docházky** - uživatelské testování (testovací scénář TC7)
- **F8 - Zobrazení docházky** - uživatelské testování (testovací scénáře TC3, TC7)
- **F9 - Přihlašování a odhlašování z událostí** - uživatelské testování (testovací scénáře TC3, TC8)
- **F10 - Zobrazení přihlášených uživatelů** - uživatelské testování (testovací scénáře TC6, TC7)
- **F11 - Zobrazení vytvořených událostí** - uživatelské testování (testovací scénáře TC4, TC6, TC7)
- **F12 - Zobrazení přihlášených událostí uživatele** - uživatelské testování (testovací scénář TC3)
- **F13 - Vyhledávání a filtrování** - uživatelské testování (testovací scénář TC4, TC6, TC7, TC8)
- **F14 - Administrace** - uživatelské testování (testovací scénář TC8) a unit testy ověřující implementace constraint bloků u doménových tříd
- **N1 - RESTful API** - návrh architektury aplikace popsaný v kapitole 4 a unit testy ověřující povolené HTTP metody jednotlivých controllerů
- **N2 - Česká lokalizace** - uživatelské testování (v průběhu všech testovacích scénářů)
- **N3 - Licenční požadavky** - podkapitola 4.5 popisující licenční podmínky jednotlivých komponent použitých v průběhu implementace
- **N4 - Responzivní uživatelské rozhraní** - uživatelské testování a testování kompatibility prohlížečů 6.4
- **N5 - Zneplatnění uživatelských účtů** - uživatelské testování (testovací scénář TC8)

- **N6 - Konfigurace aplikace** - návrh architektury aplikace popsaný v kapitole 4 a instalační příručka uvedená v příloze B
- **N7 - Logování** - nastavení logování aplikačního serveru
- **N8 - Výkonnostní požadavky** - výkonnostní testy (sekce 6.3)
- **B1 - Plánování událostí** - uživatelské testování (testovací scénáře TC5, TC6)
- **B2 - Přihlašování na události** - uživatelské testování (testovací scénář TC3)
- **B3 - Kontrola počtu přihlášených uživatelů** - uživatelské testování (testovací scénář TC3)

Všechna ověření jednotlivých požadavků byla úspěšně provedena. Implementovaná aplikace odpovídá identifikovaným požadavkům a specifikaci.

Kapitola 8
Závěr

Cíle této práce byly úspěšně splněny následujícím způsobem. Na začátku práce byl proveden úvod do ECS aplikací a analyzována existující řešení pro organizování hromadných událostí a rezervace místností. Tato analýza pomohla definovat očekávané přínosy aplikace po uvedení do provozu. Dále byly identifikovány funkčnosti existujících řešení a používané způsoby prezentace informací o událostech. Tyto poznatky byly využity při návrhu aplikace a uživatelského rozhraní.

Následně byl proveden sběr a analýza požadavků na software. Identifikované požadavky byly rozděleny do tří kategorií (funkční, nefunkční, business). Při analýze funkčních požadavků byly dále identifikováni jednotliví aktéři pracující s navrhovanou aplikací a případy užití (use case). Případy užití následně poskytly základ pro definování testovacích scénářů uživatelského testování implementované aplikace.

V následující kapitole 4 byl proveden návrh architektury aplikace dle analýzy požadavků. Navrženým řešením je webová aplikace s frontendem prezentovaným JavaScriptovou aplikací, která komunikuje skrze RESTové rozhraní se serverovou částí. Pro implementaci aplikace byly vybrány Open Source technologie, umožňující provozování aplikace bez licenčních poplatků. Pro serverovou část aplikace (backend) byl vybrán webový MVC framework Grails založený na programovacím jazyce Groovy. Klientská část aplikace (frontend) je navržena jako JavaScriptová aplikace, založená na knihovnách BackboneJS a MarionetteJS. Implementace aplikace byla provedena pomocí vybraných technologií.

Aplikace byla otestována jednotkovým a uživatelským testováním. Uživatelského testování se zúčastnili 3 participanti. Při uživatelském testování byly nalezeny 2 chyby v implementaci, které byly následně opraveny. Testovací scénáře pro uživatelské testování byly navrženy dle identifikovaných případů užití v kapitole 3.4.2. Dále byly provedeny výkonnostní testy serverové části aplikace a testy kompatibility s prohlížeči Firefox 52, Chrome 56, Internet Explorer 11, Edge a mobilních zařízeních iPhone SE a iPhone 6s.

Na závěr práce bylo provedeno zhodnocení splnění požadavků specifikace. Všechna ověření splnění požadavků byla úspěšně provedena.

8.1 Budoucí práce

V době dokončení této práce nebyla implementovaná aplikace nasazená a provozovaná v produkčním prostředí. Cílem další práce je nasazení aplikace do produkčního prostředí a zpřístupnění aplikace zadavateli pro požadované účely.

Směr dalšího vývoje a údržby aplikace je zaměřen na implementaci mobilní aplikace. Navrženým řešením pro nejméně náročnou implementaci aplikace je použití platformy Cordova a existující klientské JavaScriptové aplikace pro vytvoření mobilních aplikací založených na web view. O realizaci mobilní aplikace bude rozhodnuto až po nasazení do produkčního prostředí a zjištění, zda současný responzivní design webových stránek a přístup pomocí webového prohlížeče není přijatelným řešením.

Literatura

[1] Wikipedia [online]. *Information and communications technology*. 2017. https://en.wikipedia.org/wiki/Information_and_communications_technology. 2017-04-27.

[2] Bart vanden Hooff. Electronic coordination and collective action: use and effects of electronic calendaring and scheduling. *Information & management 42*. 2004, 103–114.

[3] Susan F.Ehrlich. Strategies for encouraging successful adoption of office communication systems. *ACM Transactions on Office Information Systems 5*. 1987, 340–357.

[4] Leysia A.Palen. Social individual & technological issues for groupware calendar systems. *Proceedings of the ACM Computer-Human Interaction '99 Conference*. 1999, 17–24.

[5] Julie Thomas Claudia Roda. Attention aware systems: Theories, applications, and research agenda. *Computers in Human Behavior, 22*. 2006, 557–587.

[6] Robin Eyraud Elisabetta Zibetti, Aline Chevalier. What type of information displayed on digital scheduling software facilitates reflective planning tasks for students? Contributions to the design of a school task management tool. *Computers in Human Behavior 28*. 2012, 591–607.

[7] *Documentation vis.js Timeline*. http://visjs.org/docs/timeline/index.html. 2017-05-07.

[8] *Timeline PrimeFaces*. https://www.primefaces.org/showcase/ui/data/timeline/custom.xhtml. 2017-05-04.

[9] Wikipedia [online]. *Modal window*. 2016. https://en.wikipedia.org/wiki/Modal_window. 2017-05-10.

[10] Defense Acquisition University Press. *System engineering fundamentals*. https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide_01_01.pdf. 2017-04-15.

[11] EA team. *Business Rule Model*. 2016. http://www.enterprise-architect.cz/blog/135. 2017-04-15.

[12] Jim Arlow Ila Neustadt. *UML2 a unifikovaný proces vývoje aplikací*. Computer Press, 2011. ISBN 978-80-251-1503-9.

[13] Graeme Keith Rocher. *The Definitive Guide to Grails*. Apress, 2006. ISBN 978-1-59059-758-3. http://dbmanagement.info/Books/Others/The_Definitive_Guide_to_Grails_Dec_2006.pdf.

[14] *Groovy Language Documentation, Version 2.4.10*. 2017. http://groovy-lang.org/single-page-documentation.html. 2017-04-21.

[15] Marc Palmer Graeme Rocher, Peter Ledbrook. *The Grails Framework Documentation, Version 3.2.4*. 2017.
http://docs.grails.org/3.2.4/.

[16] Burt Beckwith. *Spring Security Core Plugin - Reference Documentation*. 2016.
http://grails-plugins.github.io/grails-spring-security-core/v3/index.html. 2017-04-21.

[17] Cristian Olaru. *Grails 3 - Step By Step, preview*. Apress, 2017.
https://grailsthreebook.com/assets/pdf/grails3book-preview.pdf. 2017-04-26.

[18] Addy Osmani. *Developing Backbone.js Application*. O'Reilly, 2013. ISBN 978-1-449-32825-2.

[19] Jeremy Ashkenas. *Backbone.js Documentation*.
http://backbonejs.org/. 2017-04-25.

[20] *Backbone.Marionette v3.1.0 Documentation*.
https://marionettejs.com/docs/v3.1.0/. 2017-04-25.

[21] Open Source Initiative. *Frequently Answered Questions - Basics of Open Source*.
https://opensource.org/faq. 2017-04-17.

[22] Free Software Foundation Inc. *The Free Software Definition*. 2001.
http://www.gnu.org/philosophy/free-sw.html. 2017-04-17.

[23] Radek Dvořák. *Fenomén licencí Open Source*. 2010.
https://is.muni.cz/th/134784/pravf_m/DIPLOMKA-FINAL.pdf. Právnická fakulta Masarykovy univerzity v Brně, 2017-04-17.

[24] Free Software Foundation Inc. *What is Copyleft?* 2002.
https://www.gnu.org/copyleft. 2017-04-17.

[25] Josef Aujezdský. *Open source software*. 2012.
https://www.root.cz/specialy/licence/open-source-software/. 2017-04-17.

[26] Open Source Initiative. *The Open Source Definition*. 2007.
https://opensource.org/docs/osd. 2017-04-17.

[27] Wikipedia [online]. *Apache Licence*. 2017.
https://www.gnu.org/copyleft. 2017-04-17.

[28] Apache Software Foundation. *Apache Licence, version 2*. 2004.
https://www.apache.org/licenses/LICENSE-2.0.html. 2017-04-17.

[29] Apache Software Foundation. *The 3-Clause BSD License*.
https://opensource.org/licenses/BSD-3-Clause. 2017-04-18.

[30] Martin Malý. *Mobilizujeme web v HTML5*. 2011.
https://www.zdrojak.cz/clanky/mobilizujeme-web-v-html5/. 2017-05-08.

[31] Ben McCormick. *The Case For Marionette.js*. 2014.
https://benmccormick.org/2014/12/02/the-case-for-marionette-js/.

[32] Bohumil Jahoda. *Responsivní tabulky*. 2013.
http://jecas.cz/responsivni-tabulky/. 2017-05-09.

Příloha A
Zkratky

API	Application Programming Interface
ASF	nezisková organizace Apache Software Foundation
BSD	softwarová licence pojmenovaná podle Unixového systému, Berkeley Software Distribution
CRUD	základní operace nad záznamy v perzistentním úložišti (Create, Read, Update, Delete)
CSS	Cascading Style Sheets
ČVUT	České Vysoké Učení Technické v Praze
DAO	Data access object
ECS	Electronic Calendaring and Scheduling
ERP	Enterprise Resource Planning - informační systém pro plánování podnikových zdrojů
FEL	Fakulta Elektrotechnická
GUI	Graphical User Interface, grafické uživatelské rozhraní
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technologies - informační a komunikační technologie
JNDI	Java Naming and Directory Interface
JSON	JavaScript Object Notation
JSP	JavaServer Pages
JVM	Java Virtual Machine
MVC	architektura Model-View-Controller
ORM	Object-relational mapping - objektově relační mapování
OSI	nezisková organizace Open Source Initiative
REST	Representational State Transfer
SSL	Secure Sockets Layer
UML	Unified Modeling Language
URL	Uniform Resource Locator - řetězec znaků sloužící ke specifikaci umístění zdroje
XML	eXtensible Markup Language

Příloha B
Instalační příručka

B.1 Vývoj a build aplikace

1. nainstalovat Java JDK 1.7 nebo vyšší, nastavit systémovou proměnou `JAVA_HOME`, přidat `JAVA_HOME/bin` do systémové proměnné `PATH`
2. stáhnout Grails 3.2.4[1]
3. nastavit systémovou proměnnou `GRAILS_HOME` - umístění extrahovaného zip souboru, přidat `GRAILS_HOME/bin` do systémové proměnné `PATH`
4. pokud je vše korektně nastaveno, výpis příkazu `grails -version` zobrazí verzi frameworku - `Grails version: 3.2.4`
5. následné akce je nutné spouštět ve složce projektu

```
grails run-app      - spuštění aplikace ve vývojovém prostředí
grails test-app     - spuštění unit testů
grails war          - vytvoření *.war souboru, umístění v ./build/lib
```

B.2 Deployment war souboru

1. vytvořit novou databázi pro aplikaci
2. aplikační server
   - přidat JNDI resource pro databázi - název `jdbc/training`
   - přidat JNDI resource pro mailového klienta - název `mail/training`
   - přidat MIME type mapping - extension: jst, mime-type: text/plain
3. umístit do složky aplikačního serveru, ve které bude aplikace nasazená, konfigurační soubor `app-config.yml` - příklad konfigurace:

```
grails:
   serverURL: http://...../       #url serveru
   googleMaps:
      apiKey: ....                #api klíč pro google maps api
      baseURL: https://maps.googleapis.com/maps/api/geocode/json
                                  #url pro volání google maps api
```

[1] `https://grails.org/download.html`

4. vložit war soubor k deploymentu, restartovat aplikační server/doménu
5. při prvním úspěšném spuštění je automaticky vytvořen administrátorský účet - username: admin, heslo: admin, přihlašovací údaje po prvním spuštění změnit!

Aplikace je provozovatelná na Java EE kontejnerech, které podporují Servlet 3.0 a vyšší, konkrétně se jedná o:

- Tomcat 7 a vyšší
- Glassfish 3 a vyšší
- Resin 4 a vyšší
- JBoss 6 a vyšší
- Jetty 8 a vyšší
- Oracle Weblogic 12c a vyšší
- IBM WebSphere 8.0 a vyšší

Případná další nastavení buildu aplikace vzhledem k použitému aplikačnímu serveru jsou popsána v dokumentaci frameworku Grails[1].

[1] `http://docs.grails.org/3.2.4/guide/gettingStarted.html#deployingAnApplication`

Příloha C
Obsah přiloženého CD

- `/`
  - `app`
    - `configs`
      - `app-config.yml` ..... externí konfigurační soubor aplikace
    - `grails-3.2.4` ..... Grails framework
    - `training_organizer` ..... adresář obsahující zdrojové kódy implementované aplikace
    - `war` ..... adresář obsahující .war soubor určený k nasazení na aplikační server
    - `testData.sql` ..... SQL soubor pro import testovacích dat
    - `testData.js` ..... script pro vytvoření obsahu testData.sql pomocí Faker.js
  - `documents`
    - `enterpriseArchitect`
      - `ea.eap` ..... projekt Enterprise Architect obsahující UML diagramy
    - `tex` ..... adresář se zdrojovými kódy textu práce ve formátu LaTeX
      - `fig` ..... adresář s použitými obrázky v textu práce
      - `logo` ..... adresář obsahující logo ČVUT
    - `DP_Bambas_Jaroslav_2017.pdf` ..... text práce v PDF formátu
  - `readme.txt` ..... soubor s popisem obsahu CD