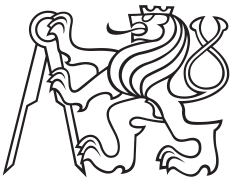
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Katedra počítačové grafiky a interakce

# Rezervační systém pro fitness studio

**Martin Šubrt**

Softwarové technologie a management, Web & multimedia

Leden 2018

Vedoucí práce: Ing. Karel Frajták

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Šubrt** Jméno: **Martin** Osobní číslo: **394334**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Softwarové technologie a management**

Studijní obor: **Web a multimedia**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Rezervační systém pro fitness studio**

Název bakalářské práce anglicky:

**Fitness studio reservation system**

Pokyny pro vypracování:

Navrhněte a implementujte rezervační systém pro fitness studio. Systém bude umožňovat registrovaným a přihlášeným uživatelům výpis seznamu lekcí podle definovaných kritérií a přihlašování na dané lekce. V případě naplnění kapacity lekce bude systém umožňovat registraci náhradníků. Součástí systému bude administrativní část, kde mohou uživatelé s oprávněním administrátora lekce zadávat. Systém bude uživatelům systému poskytovat statistiky a přehledy proběhlých a absolvovaných lekcí, stejně administrátorům statistiky týkající se účasti uživatelů na lekcích.
Systém bude implementován jako Single Page Aplikace v JavaScriptu za použití aktuálních technologií a knihoven pro implementaci webových aplikací s důrazem na udržitelnost, rozšiřitelnost, testovatelnost a intuitivní ovládání uživatelského rozhraní. Součástí bakalářské práce bude rešerše vhodných technologií a knihoven pro implementaci a návrh vhodných jednotkových testů systému.

Seznam doporučené literatury:

Michael S. Mikowski and Josh C. Powell - Single Page Web Applications: JavaScript end-to-end (ISBN 9781617290756)

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**Ing. Karel Frajták, katedra počítačů FEL**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **22.11.2017** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2019**

Karel Frajták
Ing. Karel Frajták
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Ing. Pavel Ripka, CSc.
podpis děkana(ky)

## III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

14. 12. 2017
Datum převzetí zadání

Podpis studenta

# Poděkování / Prohlášení

Děkuji všem, kteří mě podporovali při psaní této práce. Zejména vedoucímu práce Ing. Karlu Frajtákovi za odbornou pomoc, čas a trpělivost. Děkuji také rodině a přátelům za podporu psychickou, gramatickou a stylistickou.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval sám s přispěním vedoucího práce a konzultanta a že jsem v práci uvedl veškeré použité informační zdroje.

Dále prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé bakalářské práce nebo její části se souhlasem katedry Počítačové grafiky a interakce na FEL.

V Praze dne 9. 1. 2018

.....

# Abstrakt / Abstract

Tato práce se zabývá vývojem rezervačního systému fitness studia. Cílem této práce je vytvořit rezervační systém pro fitness studio. Webové stránky budou moci procházet i nepřihlášení uživatelé, kde uvidí seznam cvičení. Po registraci se klient bude moci na cvičení přihlásit. Při zaplnění kapacity systém nabídne možnost přihlásit se jako náhradník a bude přihlášen při uvolnění. Klient bude moci zobrazit svá cvičení, na kterých v historii byl. Uživatelské rozhraní bude napsáno v programovacím jazyku JavaScript, aplikace typu Single page application, to přinese rychlost a multiplatformnost aplikaci. Na vývoj budou použity nejnovější technologie. Výsledná aplikace bude otestována.

**Klíčová slova:** fitness studio, cvičení, posilování, masáž

The goal of this thesis is to design and develop reservation system for fitness studio. Web page will be open for unregistered people to browse future exercises. It can bring new ones to the studio. Logged users will be able to sign for the exercises they like. If there is no space in the exercise, user will have opportunity to sign as an alternate. He will be sign for regulary if someone resigns. Client will be able to see his exercises history. User interface will be coded in JavaScript as Single page application. It speeds up the application and enables it be run on many todays platforms.

**Keywords:** fitness studio, workout, exercising, massage

**Title translation:** Bachelor thesis – Reservation system for fitness studio

# Obsah /

# Obrázky

# Kapitola 1
# Úvod

Pohodlí zákazníka je dnes prioritou pro většinu poskytovatelů různých služeb. Proto vznikají inteligentní aplikace, skrze které obchodníci a poskytovatelé služeb nabízejí své produkty a služby.

Jako mnoho dalších služeb, i fitness studia chtějí klientům nabídnout možnost přihlásit se na cvičení jejich oblíbených trenérů na dálku.

Vzniká rezervační systém pro fitness studio. Aplikace bude webová, ta je automaticky multiplatformní. Díky tomu se aplikace dostane k více uživatelům.

Klienti se budou moci přihlásit na cvičení na dálku. Zobrazit historii svých cvičení. Zobrazit kontakty trenérů.

Naopak trenéři menšího fitness studia a fitness studio jako celek se zviditelní. Nadcházející cvičení budou moci zobrazit i nepřihlášení návštěvníci stránek.

Důraz u takovéto aplikace klademe na jednoduchost, rychlost a použitelnost například v metru, kde bývá jen velmi slabý signál nebo vůbec žádný.

V první kapitole této práce navrhujeme aplikaci. Definujeme funkční a nefunkční požadavky a vytváříme doménový model.

V druhé kapitole vybíráme vhodné technologie pro každou ze dvou částí aplikace. Programovací jazyk části serverové a framework části klientské.

Ve třetí kapitole popisujeme a seznamujeme s dalšími použitými technologiemi. Vysvětlujeme komunikaci mezi částí klientskou a serverovou.

Ve čtvrté kapitole popisujeme implementaci systému. Popisujeme validaci formulářů, registraci uživatelů, zakládání nového cvičení.

V poslední kapitole zpětně hodnotíme práci a její výsledek.

# Kapitola 2
# Analýza aplikace

Aplikace vzniká pro lepší a pohodlnější možnost rezervace na cvičení a další služby, jež fitness studio nabízí.

## 2.1 Zainteresované osoby a instituce (stakeholders)

- Fitness studio, které nabízí cvičení s trenéry, skupinové cvičení a další služby
  - Další služby mohou být například masáže, poradenství trenérů a podobně
- Klient fitness studia
- Trenér fitness studia
- Manažer fitness studia

## 2.2 Uživatelé

- Klient fitness studia
  - chce cvičit pod vedením trenéra
  - chce přijít na konkrétní cvičení
  - chce využít další služby, například přihlásit se na masáž
- Trenér fitness studia
  - zakládá nová cvičení
  - mění vypsaná cvičení
- Administrátor
  - spravuje ostatní uživatelské účty
  - spravuje všechna cvičení
  - spravuje typy cvičení
  - spravuje všechny místnosti fitness studia
  - spravuje typy místností fitness studia

## 2.3 Požadavky na funkcionalitu

- Uživatelský profil
- Zobrazení volných cvičení
- Přihlášení se na cvičení, odhlášení se z cvičení
- Vypsání cvičení, úprava cvičení (čas, místo a délku trvání), zrušení cvičení
- Přihlášení se na cvičení jako náhradník
- Úprava profilu
- Email notifikace
- Hromadné notifikace (zrušení cvičení)

## 2.4 Funkční požadavky

- **FR01 Zaregistrovat se**
  Systém umožní návštěvníkovi registrovat se do systému.
- **FR02 Přihlásit se**
  Systém umožní zaregistrovanému uživateli se do systému přihlásit.
- **FR03 Zobrazit přehled cvičení**
  Systém umožní klientovi fitness studia zobrazit přehled cvičení, na které se může přihlásit.
- **FR04 Zobrazit detail cvičení**
  Systém umožní uživateli zobrazit detail cvičení, kde vidí více informací.
- **FR05 Zadat nové cvičení do systému**
  Systém umožní trenérovi zadat nové cvičení do systému.
- **FR06 Upravit vlastní cvičení v systému**
  Systém umožní trenérovi upravit jím zadané cvičení.
- **FR07 Upravit cvičení v systému**
  Systém umožní manažerovi upravit cvičení v systému.
- **FR08 Přihlásit se na cvičení**
  Systém umožní klientovi přihlásit se vybrané cvičení.
- **FR09 Přihlásit se na cvičení jako náhradník**
  Při zaplnění kapacity cvičení systém umožní klientovi přihlásit se vybrané cvičení jako náhradník.
- **FR10 Odhlásit se z cvičení**
  Systém umožní klientovi odhlásit se z cvičení před datem uzavření přihlášek.
- **FR11 Otevření profilu uživatele**
  Systém umožní uživateli otevření vlastního profilu a profilů cvičitelů.
- **FR12 Upravení profilu**
  Systém umožní uživateli upravit vlastní profil.
- **FR13 Notifikace klienta**
  Systém automaticky notifikuje klienta v případě změny času nebo místa konání cvičení nebo při jeho zrušení.
- **FR14 Notifikace cvičitele**
  Systém automaticky notifikuje cvičitele pokud se zaplní minimální nebo maximální kapacita jeho cvičení.
- **FR15 Otevření profilu uživatele**
  Systém umožní klientovi otevření vlastního profilu a profilů cvičitelů.
- **FR16 Spravovat cvičení**
  Systém umožní manažerovi spravovat cvičení, které následně cvičitel může vypsat.

## 2.5 Nefunkční požadavky

- **NFR01 Webová aplikace**
  - Systém bude fungovat jako webová aplikace.
- **NFR02 Grafické prostředí**
  - Systém bude ovládán přes grafické uživatelské rozhraní ve webovém prohlížeči pomocí standardních vstupů.

- Grafické prostředí bude mít čistý, jednoduchý a přehledný design, aby uživatelé neměli problém s orientací.
- Uživatelé budou různě technicky gramotní.

#### NFR03 Rozšiřitelnost

- Systém bude navržen tak, aby mohl být jednoduše rozšířen.
- Na stejném principu může fungovat mobilní aplikace.

#### NFR04 Univerzálnost

- Systém bude navržen multiplatformě. Půjde spustit na kterémkoliv systému používaném v dnešní době.

#### NFR05 Zabezpečené přihlašování

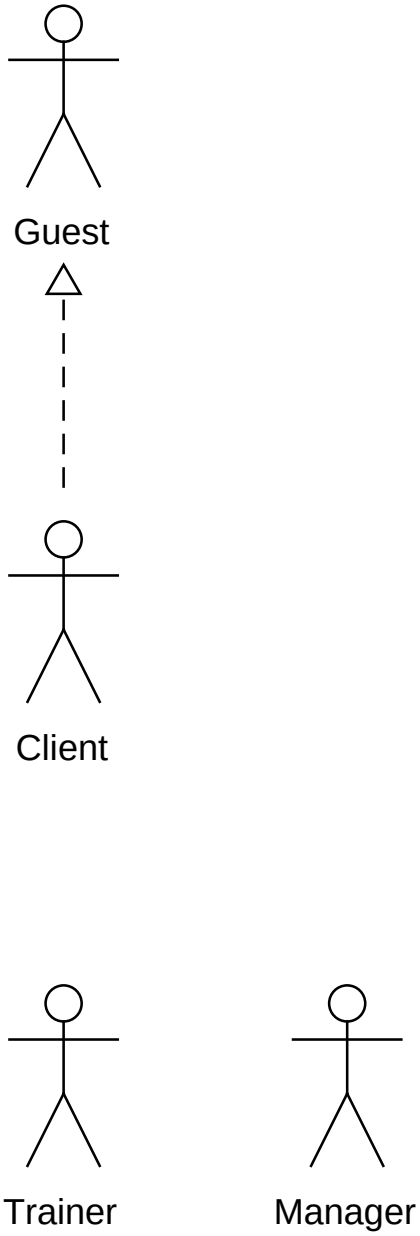
- Uživatel se bude přihlašovat pomocí protokolu HTTPS.

#### NFR06 Spolehlivost přístupu

- Náš systém nepotřebuje nejvyšší úroveň spolehlivosti. Aplikace je přístupná bez připojení k internetu po prvním stažení, proto uživatel může aplikaci ovládat například v metru.

## 2.6 Případy užití

### 2.6.1 Uživatelé

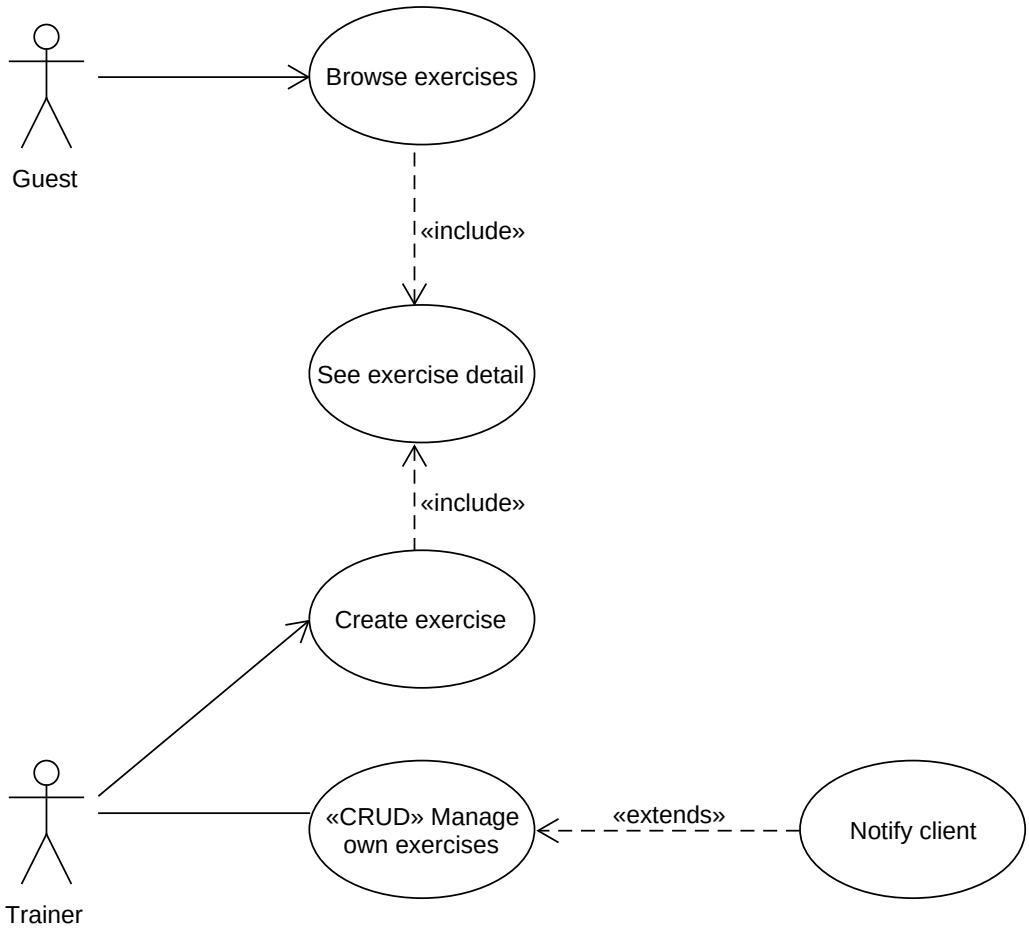



**Obrázek 2.1.** Uživatelé

- **Guest**
  Host si může procházet vypsaná cvičení. Může se zaregistrovat a přihlásit do systému.
- **Client**
  Klient se může přihlásit na cvičení. Může se přihlásit jako náhradník. Nebo se může z cvičení odhlásit.
- **Trainer**
  Trenér spravuje svá cvičení, zakládá nová, upravuje stávající.

- **Manager**
  Manažer má absolutní práva, může přidávat, upravovat nebo mazat nové místnosti, cvičení, typy místností nebo cvičení do systému. Manažer také může upravovat profily ostatních zaměstnanců nebo klientů.

### 2.6.2 Správa cvičení




**Obrázek 2.2.** Správa cvičení

- **Vytvořit nové cvičení**
  Systém umožní trenérovi vytvořit nové cvičení v systému.

1. Přihlášený trenér vybere možnost zadata nové cvičení.
2. Zadá typ cvičení, místnost, kapacitu, datum a čas konání.
3. Cvičení uloží.
4. Cvičení se zobrazí v přehledu všech cvičení, na které se klienti mohou přihlašovat.

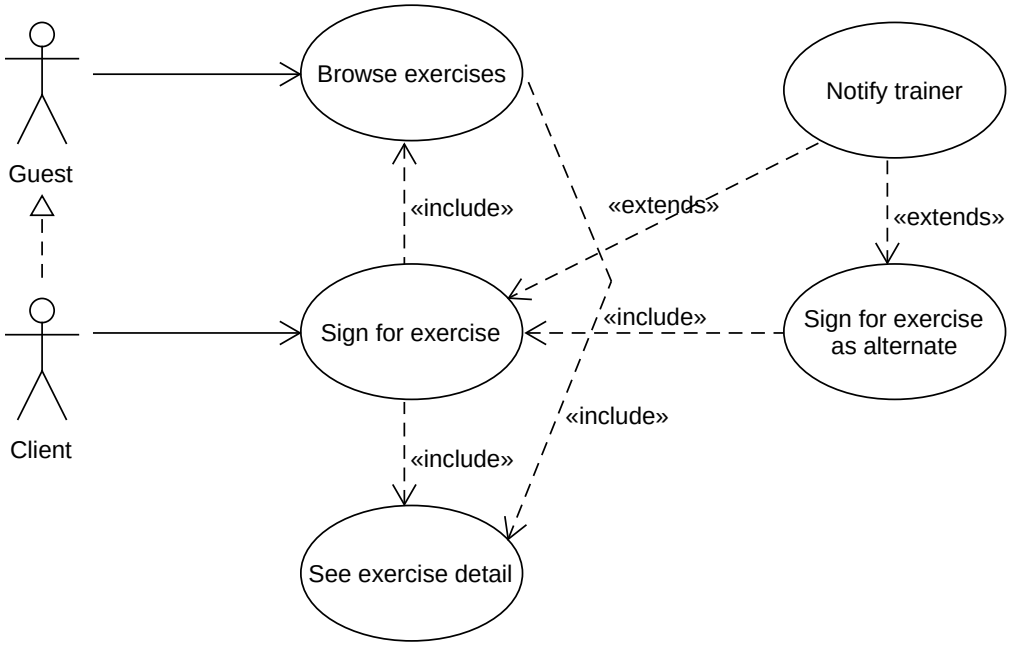
- **Upravit cvičení**
  Systém umožní trenérovi upravit jím zadané cvičení do systému.

1. Přihlášený trenér vybere dané cvičení.
2. Upraví kapacitu, místo nebo čas konání cvičení.

3. Při úpravě místa nebo času konání budou přihlášení klienti na toto cvičení notifikováni emailem.
4. Při snížení kapacity budou přihlášení klienti, kteří mají pořadové číslo větší než je nová kapacita, notifikováni, že jsou nyní na cvičení přihlášeni pouze jako náhradníci. Při navýšení kapacity, budou klienti, kteří se přihlásili jako náhradníci nejdříve notifikováni, že byli na cvičení přihlášeni.

- **Procházet cvičení**
  Systém umožní klientovi procházet všechna zadaná cvičení.

### 2.6.3 Přihlásit se na cvičení



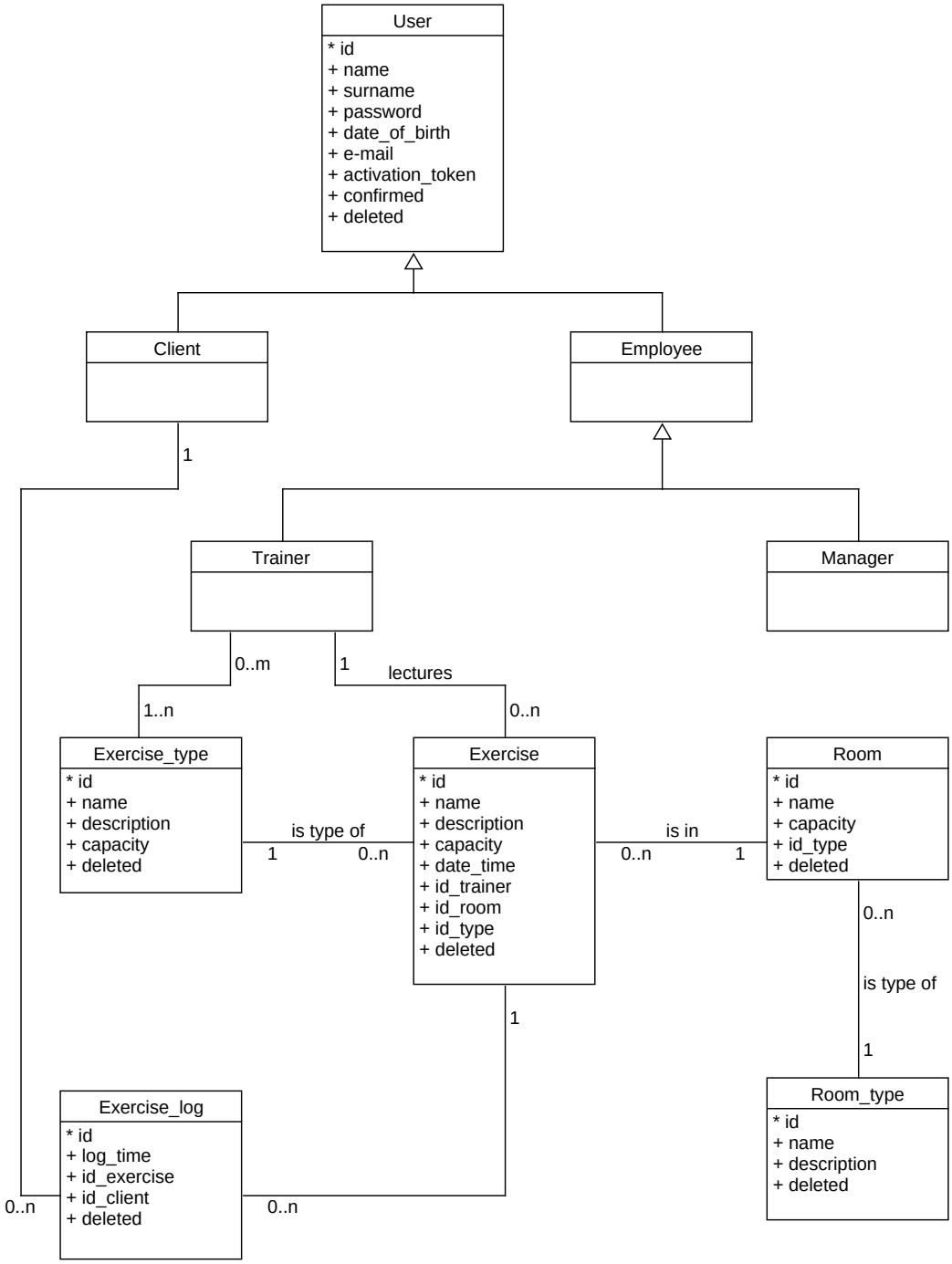

**Obrázek 2.3.** Správa cvičení

- **Přihlásit se na cvičení**
  Systém umožní klientovi přihlásit se na cvičení.

1. Klient se v systému přihlásí na vybrané cvičení.
2. V případě že je kapacita cvičení již vyčerpána, je systémem nabídnuta možnost přihlásit se jako náhradník.
3. Cvičení uloží.
4. Cvičení se zobrazí v přehledu všech cvičení, na které se klienti mohou přihlašovat.

## 2.7 Doménový model

Doménový model ukazuje zjednodušené třídy výsledného programu a vztahy mezi nimi. Doménový model V této sekci popisuji třídy, které vznikly při prvotním návrhu aplikace.




**Obrázek 2.4.** Doménový model

### 2.7.1 Uživatel

Registrovaný uživatel, který se může do systému přihlásit. Podle role, může vykonávat další akce. Akce jednotlivých rolí jsou popsány v kapitole 2.2.

Atributy

- **id** - jednoznačný identifikátor záznamu v databázi
- **name** - křestní jméno uživatele

- **surname** - příjmení uživatele
- **password** - heslo
- **date_of_birth** - datum narození uživatele
- **e-mail** - e-mailová adresa uživatele
- **activation_token** - pseudo náhodná kombinace znaků sloužící pro potvrzení e-mailové adresy uživatele
- **confirmed** - příznak zda uživatelova e-mailová adresa již byla ověřena
- **deleted** - příznak zda byl záznam vymazán

### 2.7.2 Klient

Uživatel role klienta. Má vazbu `1 - 0..n` na log cvičení. Může se přihlásit na `n` cvičení.

### 2.7.3 Zaměstnanec

Uživatel role zaměstnance. Zaměstnanec jako takový nemá žádnou vazbu.

### 2.7.4 Trenér

Uživatel role trenéra. Má vazby

- `1 - 1..n` na typ cvičení. Může trénovat (cvičit) 1 až n typů cvičení.
- `0..n - 1..m` na cvičení. Může cvičit `m` cvičení.

### 2.7.5 Manažer

Uživatel role manažera.

### 2.7.6 Typ místnosti

Typ místnosti fitness studia. Nejdůležitější atribut je zde popis. Může existovat typ posilovny, ta bude zpravidla jedna, dále místnost na protahování, masérna nebo místnost na skupinová cvičení, těch může být ve fitness studiu více, proto zakládám typ místnosti. Má vazbu `1 - 0..n` na místnost. Více místností může být stejného typu.

Atributy

- **id** - jednoznačný identifikátor záznamu v databázi
- **name** - jméno typu nebo krátký popis
- **description** - delší popis typu místnosti
- **deleted** - příznak zda byl záznam vymazán

### 2.7.7 Místnost

Ve fitness studiu máme několik místností, ve kterých se provádí různá cvičení. Má vazby

- `0..n - 1` na typ cvičení. Každá místnost je určitého typu.
- `1 - 0..n` na cvičení. V místnosti může probíhat `n` cvičení.

Atributy

- **id** - jednoznačný identifikátor záznamu v databázi
- **name** - jméno typu nebo krátký popis místnosti
- **capacity** - maximální kapacita místnosti
- **deleted** - příznak zda byl záznam vymazán

### 2.7.8 Typ cvičení

Trénovaná cvičení mohou být různých typů (např.: trénink jednotlivce s trenérem, cvičení jógy apod.) Má vazby

- `1..m - 1..n` na trenéra. Typ cvičení může cvičit `n` trenérů.
- `1 - 0..n` na cvičení. Určitého typu může být `n` cvičení.

### 2.7.9 Cvičení

Má vazby

- `0..n - 1` na typ cvičení. Každé cvičení musí být určitého typu.
- `0..n - 1` na trenéra. Každé cvičení musí předcvičovat trenér.
- `0..n - 1` na místnost. Každé cvičení musí probíhat v místnosti fitness studia.
- `1 - 0..n` na log cvičení. Na cvičení se může přihlásit `n` klientů.

# Kapitola 3
# Analýza technologií

Před implementací analyzuji technologie (programovací jazyky a knihovny). Zde jsou popsány další použité technologie4. Ve výběru se budu rozhodovat podle následujících parametrů.

- **Přínosnost aplikaci**: Splňuje technologie alespoň některý nefunkční požadavek2.5?
- **Popularita**: Jak moc je technologie ve světě využívaná? Vyšší popularita přináší širší komunitu, snadnější řešení potíží i snadnější nasazování aplikace.
- **Vlastní zkušenost**: Mám s danou technologií zkušenosti?

## 3.1 Klientská část

V této sekci rozebírám základní technologie dvou částí aplikace. U první (klientské) části je předem stanoveno, že bude napsána v programovacím jazyce JavaScript[1] a že tato část aplikace bude typu *Single page application*[2]. Zde se tedy zabývám výběrem nejvhodnější technologie (frameworku A) pro psaní právě *Single page application*.

### 3.1.1 Analýza

Jelikož nemám žádné předchozí zkušenosti s JavaScript frameworky, konzultoval jsem výběr frameworku s vedoucím práce. Byly mi doporučeny dva frameworky, a to Vue.js[3] a React[4].

- **Přínosnost aplikaci**: Oba frameworky zaručují chod aplikace jako Single page application4.4 a rychlost.
- **Popularita**: V popularitě zde vede Ract.
- **Vlastní zkušenost**: Osobní zkušenost nemám ani s jedním frameworkem.

### 3.1.2 Rozhodnutí

Po pročtení dokumentací, jsem se rozhodl pro Vue.js.

## 3.2 Serverová část

Ze zadání není nijak stanoveno, v jakém programovacím jazyce má být tato část napsána. Zde vybírám vhodný jazyk pro programování serverové části.

[1] `https://www.javascript.com/`
[2] Jednostránková aplikace, která je načtena hned. Další obsah, který je na této stránce měněn právě JavaScriptem, je donačítán postupně. viz sekci4.4.
[3] `https://vuejs.org/`
[4] `https://reactjs.org/`

### 3.2.1 Analýza

Vzhledem k rošířenosti jsem se rozhodoval mezi těmito třemi programovacími jazyky: PHP[1], Java[2] a ASP.NET[3].

- **Přínosnost aplikaci**: Všechny 3 jazyky nabízí možnosti pro vývoj serverové části aplikace.
- **Popularita**: Aktuální statistika popularity podle W3Techs[4]

| jazyk | využití | změna od 1. prosince 2017 |
|---|---|---|
| PHP | 83.1% | +0.1% |
| ASP.NET | 14.1% | -0.1% |
| Java | 2.5% | |
| static files | 1.4% | |
| ColdFusion | 0.6% | |

**Obrázek 3.1.** Popularita programovacích jazyků pro servery

- **Vlastní zkušenost**: Vlastní zkušenost školní i pracovní mám s jazyky PHP a Java EE. S jazykem ASP.NET nemám zkušenosti žádné.

### 3.2.2 Rozhodnutí

Po zvážení výše zmíněných jsem se jednoznačně rozhodl pro jazyk PHP.

[1] http://php.net/
[2] http://www.oracle.com/technetwork/java/javaee/overview/index.html
[3] https://www.asp.net/
[4] https://w3techs.com/

# Kapitola 4
# Použité technologie

V této kapitole popisuji tenchnologie, které jsem vybral pro implementaci aplikace **Gymbook**.

Moderní aplikace dbají na rychlost a dostupnost. Proto je naše aplikace webová. Díky tomu je možné ji využít na platformách osobního počítače, mobilního telefonu či tabletu.

## 4.1 Framework Symfony

Webová aplikace je rozdělena do serverové a klientské části. Serverová část je napsána v jazyce PHP, využit framework Symfony 3.4[1].

Tento framework je jedním z nejrozšířenějších a celosvětově nejpoužívánějším frameworkem pro tvorbu webových stránek.

## 4.2 Databáze

Pro jednoduchost instalace a zkušenostmi s technologií jsem pro ukládání do databáze zvolil MySQL[2]

Framework Symfony používá ORM Doctrine 2, je to objektově relační mapování. Což ulehčuje práci s databází. V programu se pracuje s objekty, které reprezentují záznamy v databázi. Díky tomuto principu je jednodušší a čitelnější práce se záznamy v databázi. Pro persistenci objektů do databáze se využívá entity manager.

## 4.3 REST api

Technologie REST api je dnes velmi rozšířená a používáná technologie z několika důvodů. Uvedeme si tu alespoň některé z nich.

- Dá se využít na více platformách, jeden server poskytující api může sloužit pro webovou aplikaci i pro mobilní aplikaci.

[1] `https://symfony.com/doc/3.4//index.html`
[2] `https://www.mysql.com/`

### 4.3.1 REST

REST (Representational State Transfer) je architektura rozhraní, navržená pro distribuované prostředí. Pojem REST byl představen v roce 2000 v disertační práci Roye Fieldinga, jednoho ze spoluautorů protokolu HTTP. Proto nepřekvapí, že REST má s HTTP hodně společného.

REST je, na rozdíl od známějších XML-RPC či SOAP, orientován datově, nikoli procedurálně. Webové služby definují vzdálené procedury a protokol pro jejich volání, REST určuje, jak se přistupuje k datům.

Rozhraní REST je použitelné pro jednotný a snadný přístup ke zdrojům (resources). Zdrojem mohou být data, stejně jako stavy aplikace (pokud je lze popsat konkrétními daty). Všechny zdroje mají vlastní identifikátor URI a REST definuje čtyři základní metody pro přístup k nim.

### 4.3.2 Metody pro přístup ke zdrojům

REST implementuje čtyři základní metody, které jsou známé pod označením CRUD, tedy vytvoření dat (Create), získání požadovaných dat (Retrieve), změnu (Update) a smazání (Delete). Tyto metody jsou implementovány pomocí odpovídajících metod HTTP protokolu.

### 4.3.3 GET (Retrieve)

Základní metodou pro přístup ke zdrojům je **získání zdroje** – metoda GET. Setkává se s ní každý uživatel webu dnes a denně – není to nic jiného než starý dobrý požadavek na stránku.

```
GET /api/user/pepa
Host: www.server.cz
```

Jak jsme si už řekli, má každý zdroj (resource) podle rozhraní REST vlastní identifikátor (URI). Pomocí HTTP GET požadavku získáme data konkrétního zdroje. Ukažme si praktický příklad – získání posledních zpráv konkrétního uživatele (bude to `lupacz`) na Twitteru. Twitter poskytuje pro přístup k datům právě rozhraní REST (jeho API je, řečeno terminologií, RESTful – myslím ale, že se volný překlad *RESTovací* pravděpodobně neujme, což je škoda, pozn. autora).

Podle dokumentace REST API Twitteru lze získat zprávy konkrétního uživatele jako zdroj `/statuses/usertimeline`. Přesný tvar je

`http://twitter.com/statuses/usertimeline/uživatel.formát`. Data uživatele „Lupacz“ ve formátu XML získáme tedy takovýmto HTTP požadavkem:

```
GET /statuses/usertimeline/lupacz.xml
Host: twitter.com
```

Což odpovídá téměř naprosto přesně požadavku, který webový prohlížeč odešle, když klikneme na následující odkaz: http://twitter.com/statuses/usertimeline/lupacz.xml[1] – můžeme si tedy sami velmi jednoduše vyzkoušet, jak vypadá výsledek. Můžeme zkusit

[1] `http://twitter.com/statuses/usertimeline/lupacz.xml`

změnit formát dat – máme na výběr XML, JSON, RSS a ATOM. Takto například vypadá tentýž zdroj, když je formátován jako RSS[1]

API služby Twitter je poměrně dobrým příkladem rozhraní, které je postaveno na architektuře REST, a tak jeho studium může sloužit jako dobrý odrazový můstek pro prozkoumání možností podobných RESTful API.

Pokud potřebujeme získaná data nějak upravit či filtrovat, můžeme je předat jako součást požadavku tak, jak jsme zvyklí – jako tzv. „Query parametry“, tedy za otazníkem. Například http://twitter.com/statuses/usertimeline/lupacz.xmlcount=100[2]

### 4.3.4 POST (Create)

Získání dat je jednoduché a přímočaré. Stejně tak i vytvoření dat bude většině webových vývojářů připadat povědomé. Pro vytvoření dat slouží totiž metoda POST, známá (minimálně) z HTML formulářů.

U metody POST není ve chvíli volání známý přesný identifikátor zdroje (logicky, protože zdroj ještě neexistuje). Proto se pro POST používá domluvený společný identifikátor („endpoint“). Znovu se podíváme na ilustraci na konkrétním příkladu služby Twitter.

K vytvoření dat (zprávy) je potřeba zavolat zdroj s URI `/statuses/update` pomocí HTTP metody POST. V parametru „status“ se předává text pro nově vytvořenou zprávu. Vzhledem k tomu, že vytvoření nové zprávy ovlivňuje uživatelská data, je třeba, aby volání bylo autorizováno. Twitter podporuje dvě metody autentizace: prostá HTTP autentizace nebo OAuth. OAuth poskytuje mnohem větší úroveň zabezpečení než prostá HTTP autentizace, při níž se posílá jméno a heslo v otevřeném textu, navíc umožňuje vytvářet aplikace, které využívají API, aniž by potřebovaly znát uživatelovo jméno a heslo. Implementace OAuth však není triviální, proto si pro zjednodušení ukážeme použití POST metody s jednoduchou HTTP autentizací.

```
curl -u user:password -d "status=Zkousime REST API"
http://twitter.com/statuses/update.xml
```

Po odeslání by měl server vrátit patřičný návratový kód – HTTP má k tomu účelu stavový kód 201 – Created, v němž lze předat URI nově vytvořeného zdroje. Pokud došlo k chybě, měl by server vrátit chybový kód.

### 4.3.5 DELETE

Zdroj lze smazat pomocí volání URI HTTP metodou DELETE. Volání je obdobné volání metody GET:

```
DELETE /api/user/pepa
Host: www.server.cz
```

V praxi bývá někdy problematické vyvolat HTTP metodu DELETE – spousta HTTP nástrojů či HTML formuláře jsou omezeny pouze na metody POST a GET. Proto se v praxi u REST rozhraní používají náhradní způsoby – např. volání pomocí POST

[1] http://twitter.com/statuses/usertimeline/lupacz.rss
[2] http://twitter.com/statuses/usertimeline/lupacz.xmlcount=100

s parametrem, který sděluje, že má být ve skutečnosti použita metoda DELETE, nebo speciální URI: `http://twitter.com/statuses/destroy/číslo.formát`

Pokud máme nástroj, který umožňuje vyvolat metodu DELETE, můžeme ji použít:

```
curl -u user:password --http-request DELETE
http://twitter.com/statuses/destroy/1472669360.xml
```

### 4.3.6 PUT (Update)

Operace změny je podobná operaci vytvoření (create, metoda POST), s tím rozdílem, že voláme konkrétní URI konkrétního zdroje, který chceme změnit, a v těle předáme novou hodnotu (jako u metody POST). Na rozdíl od POST je u úprav zdroje jeho URI už známá, takže ji lze zadat.

U metody PUT platí totéž, co bylo napsáno výše o metodě DELETE: Ne každý nástroj ji podporuje, a proto některá RESTful API používají různé náhradní metody, jak bylo zmíněno výše. Například:

```
curl -u user:password -d "url=http://zdrojak.root.cz"
http://twitter.com/account/updateprofile.xml
```

### 4.3.7 Shrnutí

REST je architektura, která umožňuje přistupovat k datům na určitém místě pomocí standardních metod HTTP. Pomocí REST lze ovládat i stav aplikace, pokud jej dokážeme popsat takovým způsobem, že si vystačí s modelem „zdroje – CRUD akce“.

REST nabývá na významu a stává se spolu s JSON defacto standardem pro API webových služeb. Jeho rozšíření napomáhá i technika AJAX, které REST vychází vstříc. Jeho rozšíření napomohlo i to, že se nijak zásadně neliší od standardního volání a získávání dat pomocí HTTP, pouze je zobecňuje. Dá se říct, že REST je architektura, která umožňuje CRUD operace pomocí standardních HTTP dotazů. (Pokud vám pojem CRUD stále evokuje databázové tabulky, tak jste na správné stopě – a ano, existuje nástroj[1], který funguje jako REST rozhraní k databázi.)

Moderní frameworky pro vývoj server-side aplikací pomáhají vytváření REST rozhraní tím, že dokáží nadefinovat patřičné procedury pro všechny potřebné metody, takže vytvoření vlastního RESTful rozhraní je opravdu snadné. Čím dál tím častěji je vidět v přehledech vlastností frameworků kromě (např.) „Podporuje MVC“ i „Podporuje REST“.

REST svou bezestavovostí vychází vstříc moderním metodám vývoje webových aplikací, které jsou založené na paralelním zpracování distribuovaného obsahu. Tato jeho vlastnost, spolu s výše zmíněnými výhodami (jednoduchost a nenáročnost) naznačuje, že se s rozhraními, postavenými na modelu REST, budeme do budoucna setkávat stále častěji.[1]

[1] `http://phprestsql.sourceforge.net/`

## 4.4 Single page applicatiion

Klasické webové aplikace jsou sice zatím nejlepší volbou pro mnoho projektů, ovšem přinášejí s sebou určité nevýhody a to hlavně v oblasti použitelnosti. Jedná se zejména o nutnost připojení k internetu, zhoršení uživatelského požitku z aplikace z důvodu přenačítání stránek, což má za následek i vyšší zátěž pro server.

SPA si klade za cíl tyto problémy vyřešit a představuje technologii pro tvorbu aplikací nové generace.

### 4.4.1 Klient

Single Page Application minimalizuje server a klade důraz na klientskou část. Je to vlastně aplikace napsaná v JavaScriptu, která se serverem komunikuje pomocí Web API.

Celou aplikaci představuje jediná stránka (od toho název technologie), která se stáhne ze serveru a v tu chvíli je u klienta přítomná kompletní aplikace a již nikdy nemusí přejít na jinou stránku. Jsou v ní přítomny všechny podstránky, které JavaScript skrývá a zobrazuje jak se uživatel po stránce naviguje.

Samotná data aplikace jsou ukládána lokálně, aby byla práce s aplikací co nejrychlejší a nejpohodlnější. Uživatelská přívětivost se tím blíží aplikacím desktopovým. Změny v datech aplikace se synchronizují se serverem a to buď v reálném čase AJAXem nebo zkrátka když si to klient přeje.[2]

Pro psaní klientské části můžeme využít několik Javasciptových frameworků. AngularJS, backbone.js nebo v poslední době čím dál více používaný Vue.js.

### 4.4.2 Server

Na server se tedy posílá buď XML nebo JSON a to samé ze serveru odchází.

### 4.4.3 Výhody

Ačkoli jsme již nějaké výhody technologie zmínili, uveďme si ucelený seznam těch nejdůležitějších:

- SPA kombinuje dohromady to nejlepší z desktopu a webu
- Protože je GUI u klienta, aplikace mohou být bohatší
- Aplikace okamžitě reagují
- Aplikace jsou multiplatformní a zároveň působí nativním dojmem
- Aplikace mohou fungovat offline
- Díky tlustému klientovi se šetří čas serveru a přenos dat
- Aplikace jednoduše distribuujeme

[2]

### 4.4.4 Nevýhody

Zásadní nevýhodou je zesložitění aplikace, do které kromě serverových kontrolerů, modelů a šablon přibude ještě další javascriptová vrstva, ve které v podstatě programujeme to samé znovu (modely a data).

## 4.5 Vue.js

Vue.js je Open-source progresivní JavaScriptový framework pro vytváření uživatelských rozhraní. Integrace do projektů, které využívají jiných knihoven JavaScriptu je snadné díky Vue, protože je navržen tak, aby byl postupně přijatelný. Vue může také fungovat jako webový aplikační framework, který je schopen pohánět pokročilé „jednostránkové aplikace“ (**Single page application** 4.4).[3]

## 4.6 Bootstrap

Bootstrap je framework, který ulehčuje psaní uživatelského rozhraní webových stránek. Obsahuje několik formátování a stylování textu a všech možných HTML tagů. Obsahuje šablony založené na HTML, CSS a JavaScriptu.

Framework je „mobile-first“, je to postup tvorby uživatelského rozhraní od nejmenších obrazovek (mobilní telefony) po větší (osobní počítač). U takto rozšířeného frameworku se můžeme spolehnout, že uživatelské rozhraní bude přívětivé na malých obrazovkách, pokud je přívětivé na těch větších.

# Kapitola 5
# Implementace

V následující kapitole si popíšeme implementaci systému **Gymbook**. Naše aplikace je rozdělena na dvě části, na serverovou část poskytující REST api. Tato část také komunikuje s databází. Veškerá funkcionalita aplikace je držena v této části.

Druhá část je uživatelské rozhraní. Tato část též obsahuje funkcionality, avšak ty jsou pouze pro uživatelskou přívětivost.

Nejdříve si popíšeme serverovou část, kde se zabýváme poskytováním REST api a zabezpečením systému.

Ve druhé části si poté popíšeme klientskou část, uživatelské rozhraní webové aplikace.

Komunikace mezi serverem a klientskou částí probíhá na bázi HTTP požadavků a odpovědí. Data jsou posílána ve formátu JSON. Jiný formát není podporován, pokud přijde požadavek s daty v jiném formátu, je odeslána chybová hláška.

## 5.1 Serverová část

### 5.1.1 Registrace uživatele

Uživatelé při registraci zadávají:

- E-mail
- Jméno
- Příjmení
- Datum narození
- Heslo
- Potvrzení hesla

E-mail je jednoznačný identifikátor uživatele. E-mail je dále použit při odesílání notifikací klienta (např.: při zrušení cvičení nebo při přihlášení klienta na cvičení, na které byl přihlášen jako náhradník).

Heslo musí mít minimálně 8 znaků a maximálně 20. Další omezení na heslo nejsou aplikována. Do databáze je heslo ukládáno zašifrováno algoritmem **Bcrypt**.

Bcrypt je hašovací funkce pro odvození klíče (key derivation function) navržená Nielsem Provosem a Davidem Mazieresem. Je založena na šifře Blowfish a byla prezentována v USENIXu v roce 1999. Bcrypt v sobě zahrnuje kryptografickou sůl, která chrání proti útokům pomocí duhové tabulky a mimo jiné se jedná o adaptivní funkci. To znamená, že můžeme záměrně zvýšit počet iterací, čímž dojde k jejímu zpomalení. Tím je zajištěna ochrana proti útokům hrubou silou.[4]

### 5.1.2 Autentizace uživatele

Pro naši aplikaci jsem vybral autentizaci pomocí api klíče. Symfony framework nabízí hned několik balíčků k autentizaci uživatelů. Pro naši aplikaci jsem zvolil Guar authentication system[1]

Princip autentizace:

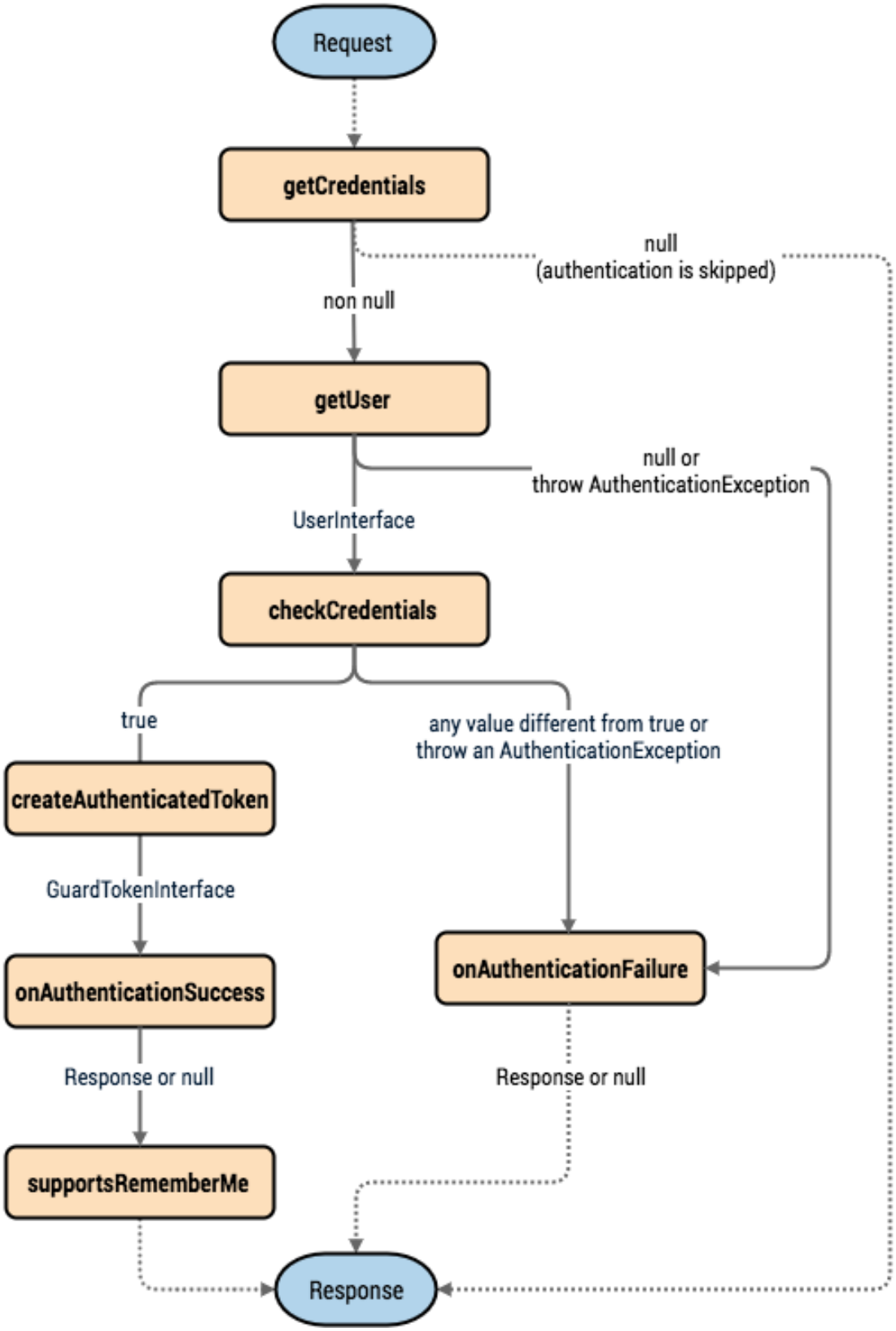
- Přihlášení

1. Klient odešle HTTP požadavek na stránku
2. Pokud není přítomen api klíč v požadavku, který server přijme, klient je vždy přesměrován na stránku s přihlašovacím formulářem
3. Klient odešle přihlašovací údaje
4. Server ověří uživatele podle e-mailu a hesla
5. Pokud jsou uživatelské údaje zadány správně, server odešle klientovi api klíč. Jinak odešle odpověď s chybovou hláškou.

- Dotazování stránek s omezeným přístupem

1. Klient odešle HTTP požadavek na stránku
2. Server ověří platnost api klíče
3. Vyhledá podle něho klienta
4. Ověří, jestli má klient na danou url přístup
5. Pošle požadovaná data, pokud klient přístup má. Jinak odešle chybovou hlášku.

[1] `https://symfony.com/doc/current/security/guard_authentication.html`




**Obrázek 5.1.** Využití Guard authentication metod

Přihlašovací údaje jsou odesílány v plain textu. Komunikace mezi serverem a klientem je šifrovaná, pomocí HTTPS.

### 5.1.3 Uživatelské role

Přístupy k jednotlivým stránkám jsou řešeny přes uživatelské role. Tyto role jsou v systému 3. Zde uvádíme k jakým stránkám mají jednotlivé role přístup:

1. Klient:

- vlastní profil
- všem vypsaným cvičením
- svým cvičením
- kontaktům trenérů

2. Trenér

- všem stránkám jako klient
- založení, editaci a zrušení jím vypsaným cvičením

3. Manažer fitcentra

- všem stránkám jako trenér
- založení, editaci a zrušení typů místností fitcentra
- založení, editaci a zrušení místností fitcentra
- založení, editaci a zrušení typů cvičení
- založení, editaci a zrušení všech cvičení
- editaci a smazání ostatních uživatelů

### 5.1.4 Validace formulářů

Pro efektivitu se používá ověřování formulářů ještě před odesláním od klienta. Gymbook také kontroluje formuláře na straně klienta, avšak z důvodu bezpečnosti je nutné všechna formulářová data kontrolovat ještě na straně serveru. Server přijímá HTTP požadavky s JSON daty. Takovéto požadavky může potencionální útočník odeslat se špatnými daty. Pokud bychom spoléhali na správnost přijatých dat a nevalidovali data na straně serveru, útočník by mohl například registrovat uživatele s e-mailovou adresou, která neobsahuje `@`. Při odesílání e-mailu na špatnou adresu se e-mail vrací s tím, že nemohl být doručen. Takovéto problémy se snažíme eliminovat.

### 5.1.5 Persistence databáze

Systém Gymbook je malá aplikace, u které se počítá s nevelkým množstvím dat. Řádově stovky uživatelů a tisíce cvičení během prvního roku.

Veškeré mazání dat v systému je *soft delete*. Data nejsou ve skutečnosti vymazána, u konkrétního záznamu je pouze nastaven příznak, že je vymazán. To přináší výhodu možnosti znovuobnovení dat. Databáze zůstává persistentní.

## 5.2 Klientská část

### 5.2.1 Validace formulářů

Validace formulářů na straně klienta šetří čas a síťové prostředky. Srovnání validace bez použití a s použitím strany serveru

- Validace pouze na straně serveru

  1. Vyplnění formuláře
  2. Odeslání dat na server
  3. Validace dat serverem
  4. Přijetí a zpracování odpovědi od serveru. Podání zpětné vazby uživateli.

- Validace na straně klienta i na straně serveru

  1. Vyplnění formuláře
  2. Validace dat na straně klienta
  3. Odeslání dat na server
  4. Validace dat serverem
  5. Přijetí a zpracování odpovědi od serveru. Podání zpětné vazby uživateli (pokud jsou validace naprogramovány správně a jsou konzistentní, zpětná vazba by měla být vždy kladná).

Oba procesy se opakuje do té doby, než uživatel zadá správná data. Nežádoucí vliv validace pouze na straně serveru je opakované odesílání a přijímání dat.

Pro validaci formulářových dat na straně klienta máme dvě možnosti.

**HTML5** validace probíhá najednou u všech polí při pokusu o odeslání formuláře. Tato validace je ze základu u tagů `input`, které jsou uvnitř tagu `form` zapnuta. Toto chování přináší výhodu v tom, že nemusíme psát vlastní JavaScriptovou validaci. Některé atributy tagu `input` zajišťují zpětnou vazbu, respektive při nesplnění daného pravidla nelze formulář odeslat. Některé z nich:

- required: Pole je povinné, nesmí být odesláno prázdné.
- type=ëmail¨: V poli musí být vyplněna platná e-mailová adresa.
- pattern: Pole musí splňovat daný regulární výraz[1].

HTML5 validaci je možné vypnout atributem `novalidate` u tagu `form`.

U **JavaScript** validace záleží, jak je naprogramována. JavaScript dokáže reagovat na změny polí webových formulářů a tlačítek. Je tedy možné validovat každé pole formuláře zvlášť, když jej opustí kurzor.

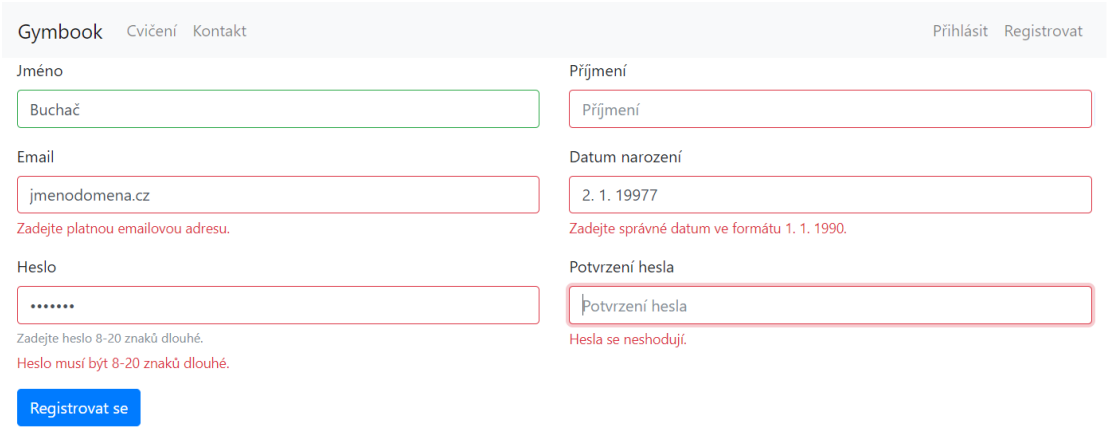
Formuláře u systému Gymbook jsou na straně klienta validovány oběma způsoby. U převážné většiny polí u všech formulářů je HTML5 validace dostačující.

První validace probíhá při prvním pokusu o odeslání dat. Dále je pak každé pole validováno při změně.

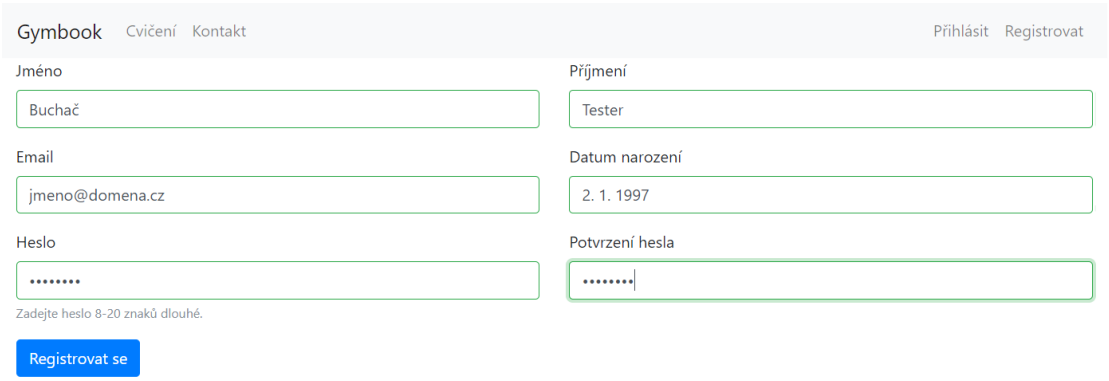
Na první heslo stačilo použít HTML5 pattern `[^ \t]{8,20}`. Ten zamezuje použití mezery a tabulátoru a omezuje délku na 8 až 20 včetně.

Speciální kontrola JavaScriptem probíhá u pole *Potvrzení hesla*. Zde je kontrolována rovnost dvou řetězců.

[1] Regulární výraz slouží k vyhledání části řetězce, kterou předem (úplně) neznáme nebo která může mít více podob. Používá se v programovacích a skriptovacích jazycích.[2]




**Obrázek 5.2.** Nevalidní formulář registrace




**Obrázek 5.3.** Validní formulář registrace

# Kapitola 6
# Testování

## 6.1 Klientská část

### 6.1.1 Použité testování

K testování uživatelského rozhraní používáme progresně regresními testy.

**Progresní a regresní testy**

Progresní testy využíváme při kontrole nových funkcí nebo vlastností aplikace. K jejich správnému provedení je nutná dokumentace, která přesně popisuje nově implementované oblasti. Používáme je ve všech etapách testování.

Regresní testy se využívají při opětovném testování funkcí a vlastností aplikace. Jejich smyslem je ověření, že provedené změny či implementace nových vlastností v aplikaci nemělo žádný vliv na stávající funkce a vlastnosti. Tedy především na oblasti, které zůstaly v programovém kódu nezměněny. Oblasti, které byly předmětem úprav, by správně již měly být otestovány funkčními testy. Takovéto situace z pravidla nastávají po opravení chyb či po novém release. Regresní testy je velmi vhodné automatizovat.

V praxi jsou regresní testy velmi rozšířené. V různých formách se používají prakticky u většiny projektů. Využívají se hlavně při retestech opravených chyb. Oproti tomu progresní testy nejsou příliš rozšířeny a často je tato kategorie testů rozložena do jiných kategorií.[5]

### 6.1.2 Návrh jednotkových testů

K jednotkovému testování uživatelského rozhraní navrhuji použití Selenium WebDriveru[1] a napsání testů v JavaScript. Selenium umožňuje vytvoření Selenium-WebDriver projektu v JavaScriptu[2].

## 6.2 Serverová část

### 6.2.1 Použité testování

K-testování serverové části využíváme aplikaci PostMan. Tato aplikace umí odesílat HTTP požadavky všech použitých metod. Přijatá odpověď je porovnána s očekávanou, tím je proveden test.

### 6.2.2 Návrh jednotkových testů

Podle dokumentace frameworku Symfony[3] vytvořím testovací třídu pro každý kontroler a pro všechny metody vytvořím testovací metodu (jeden jednotkový test).

[1] `http://www.seleniumhq.org/projects/webdriver/`
[2] `http://www.seleniumhq.org/docs/03_webdriver.jsp`
[3] `https://symfony.com/doc/current/create_framework/unit_testing.html`

# Kapitola 7
# Závěr

Cílem této práce bylo navrhnout a implementovat rezervační systém pro fitness studio. Systém implementovat tak, aby přihlášený uživatel mohl zobrazit seznam lekcí, mohl se na ně přihlásit, případně se přihlásit jako náhradník při zaplnění kapacity. Uživatel s rolí Trenér aby mohl zadávat nová cvičení do systému. Aby systém poskytoval statistiky klientovi o jeho účasti na cvičeních a trénérovi o účasti na jeho cvičeních. Systém měl být implementován jako Single page application za použití aktuálních knihoven.

Dalším cílem této práce bylo analyzovat vhodné technologie a knihovny a navrhnout jednotkové testy.

Návrh aplikace se vydařil. Detailně je popsán v kapitole 2.

V době odevzdávání této práce se na implementaci stále pracuje. Serverová část je funkční, komunikuje s databází. Přijimá HTTP požadavky a odpovídá na ně. Klientská část je ve vývoje a zatím nekomunikuje se serverem. Implementace je popsán 5

Aplikace je implementována jako Single page application, za použití nejmodernějších knihoven a technologií popsaných v kapitole 4.

Analýzou a popisem použitých technologií se zabýváme v kapitolách 2 a 4.

# Literatura


[1] *REST: architektura pro webové API*. 2010.
https://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/.

[2] *Single page application*. 2014.
https://www.itnetwork.cz/csharp/asp-net/single-page-application/tutorial-uvod-do-asp-net-single-page-application.

[3] *Co je Vue js*. 2014.
https://vuejs.org/v2/guide/#What-is-Vue-js.

[4] *Bcrypt*. 2002.
http://bcrypt.sourceforge.net/.

[5] *Progresní a regresní testy*. 2017.
http://testovanisoftwaru.cz/tag/regresni-testy/.

# Příloha A
# Zkratky a pojmy

| | |
|---|---|
| HTTP | HyperText Transfer Protocol slouží pro výměnu HTML dokumentů. |
| Framework | Sada knihoven, které ulehčují práci při programování. Výhody: méně psaní, přehlednější kód a rychlejší vývoj. |

# Příloha B
# Obsah přiloženého CD

- Text bakalářské práce (soubor BP-Subrt.pdf)
- Zdrojový kód serverové části, project IntelliJ PHPStorm (adresář server)
- Zdrojový kód klientské části, project IntelliJ PHPStorm (adresář client)