Univerzita Pardubice
Fakulta elektrotechniky a informatiky

# Webová aplikace pro tvorbu a správu aktivit s využitím geolokace

Bc. Tomáš Bartošek

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2017/2018**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Tomáš Bartošek** |
| Osobní číslo: | **I16204** |
| Studijní program: | **N2646 Informační technologie** |
| Studijní obor: | **Informační technologie** |
| Název tématu: | **Webová aplikace pro tvorbu a správu aktivit s využitím geolokace** |
| Zadávající katedra: | **Katedra softwarových technologií** |

Zásady pro vypracování:

Cílem diplomové práce je realizovat webovou aplikaci, která bude sloužit k vytváření a správě výzev (na zápas v určité hře) od jednotlivých uživatelů.
V teoretické části práce bude provedena rešerše podobných systémů a popis klíčových technologií použitých při tvorbě samotné aplikace s důrazem na jazyk Java a framework Spring. Dále zde bude představeno rozhraní Google Maps API od společnosti Google pracující s geolokačními údaji.
V praktické části bude provedena analýza, návrh a popis zadané aplikace využívající Java technologie a framework Spring. Výzvy se budou vytvářet pro určité souřadnice (např. vytvoření výzvy na hru šachy na souřadnicích xx.yy). Ostatní uživatelé si potom budou moci, na základě geolokace, tyto výzvy zobrazit a zareagovat na ně. Pro jednotlivé uživatele budou vedeny statistiky s jejich bilancí v jednotlivých hrách, žebříčky, na jaké pozici se nacházejí oproti ostatním hráčům.

| | |
|---|---|
| Rozsah grafických prací: | **10** |
| Rozsah pracovní zprávy: | **60** |
| Forma zpracování diplomové práce: | **tištěná** |

Seznam odborné literatury:


**1. WALLS, Craig. Spring in action. Fourth Edition. Shelter Island, NY: Manning, 2015. ISBN 9781617291203. DEINUM, Marten., Koen. SERNEELS, Colin. YATES, Seth. LADD a Christophe.**
**2. VANFLETEREN. Pro Spring MVC: with Web Flow. New York: Distributed to the book trade worldwide by Springer Science+Business Media, c2012. ISBN 1430241551.**
**3. BAUER, Christian a Gavin. KING. Hibernate in action. Greenwich, CT: Manning, c2005. ISBN 193239415x.**
**4. SVENNERBERG, Gabriel. Beginning Google Maps API 3. New York, NY: Apress, c2010. Expert's voice in Web development. ISBN 1430228024.**


| | |
|---|---|
| Vedoucí diplomové práce: | **doc. Ing. Michael Bažant, Ph.D.**<br>Katedra softwarových technologií |
| Datum zadání diplomové práce: | **30. října 2017** |
| Termín odevzdání diplomové práce: | **18. května 2018** |

Ing. Zdeněk Němec, Ph.D.
děkan

L.S.

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2017

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 15. 5. 2019

Bc. Tomáš Bartošek

# PODĚKOVÁNÍ

Mé poděkování patří doc. Ing. Michaelu Bažantovi, Ph.D. za jeho profesionální vedení, vstřícnost a ochotu, kterou mi během zpracování diplomové práce poskytoval.

Dále bych zde rád poděkoval mé rodině a přátelům, kteří mě během celého studia podporovali.

**ANOTACE**

Cílem diplomové práce je navrhnout a realizovat webovou aplikaci na správu aktivit s využitím geolokace. Aplikace je realizována pomocí frameworku Spring. Pro geolokaci je využito Google maps API. V teoretické části je provedena rešerše a srovnání obdobných aplikací, představen framework Spring a proveden popis technologií a nástrojů, které byly při vývoji využity. V praktické části je čtenář proveden analýzou a implementací vytvářené aplikace. V závěru je k dispozici uživatelská příručka obsahující popis obsluhy a funkcionality výsledného produktu.

**KLÍČOVÁ SLOVA**

Java, Spring, DI, Google maps API, MVC, webová aplikace, aktivita

**TITLE**

Web application for creation and management activities with geolocation usage

**ANNOTATION**

The aim of this master thesis is to design and implement a web application for managing activities using geolocation. The application is implemented using framework Spring. For geolocation is used Google maps API. At theoretical part is done comparison of similar applications, introduction of Spring framework and a description of technologies and tools used in the development. At practical part is reader introduced to analysis and implementation of the created application. At the end there is a user manual containing a description of operation and functionality of the final product.

**KEYWORDS**

Java, Spring, DI, Google maps API, MVC, web application, activity

# OBSAH

# SEZNAM OBRÁZKŮ

# SEZNAM TABULEK

# SEZNAM ZDROJOVÝCH KÓDŮ

# SEZNAM ZKRATEK

| | |
|---|---|
| AOP | Aspect Oriented Programming |
| API | Application Programming Interface |
| CSS | Cascading Style Sheets |
| DI | Dependency Injection |
| HTML | HyperText Markup Language |
| HTTP | HyperText Transfer Protocol |
| IDE | Integrated Development Environment |
| IoC | Inversion of Control |
| JSP | JavaServer Pages |
| MVC | Model-View-Controller |
| OOP | Object Oriented programming |
| SQL | Structured Query Language |
| UML | Unified Modeling Language |
| URL | Uniform Resource Locator |

# ÚVOD

V dnešní době je velice složité udržet si aktivní způsob života. Není mnoho času a velice často nejsou ani v našich sociálních kruzích lidé, s kterými bychom mohli nějaké aktivity provozovat. Diplomová práce se zabývá právě tímto tématem, a to návrhem aplikace, která by lidem pomohla k nalezení partnerů na nejrůznější aktivity.

V teoretické části práce je nejprve proveden popis již existujících aplikací zabývajících se tímto tématem. V druhé kapitole je krátce představen framework Spring, jakožto nejpopulárnější Java framework této doby. Úvodem jsou zde popsány moduly, které ve Springu můžeme nalézt. V další kapitole je kladen důraz na pochopení základního kamene frameworku Spring a to na princip označován jako vkládání závislostí. Kapitola je také věnována podpoře aspektově orientovaného programování. Následně je podrobněji rozebrán modul sloužící pro podporu vývoje webových aplikací – Spring MVC. V této kapitole je také proveden popis architektonického vzoru MVC. Dále zde nalezneme vysvětlení, co je to Dispatcher Servlet, jaké komponenty Spring MVC nabízí či jak se zde mapují požadavky. V závěru této kapitoly je stručně představen framework Spring Boot, který bude také použit při implementaci praktické části.

V další části práce je popsána samotná realizace vytvářené aplikace. Jako první je provedena analýza aplikace, včetně funkčních a nefunkčních požadavků. Dále jsou zde představeny UML diagramy a návrh samotné databáze. Následující kapitola shrnuje výčet vybraných technologií a nástrojů použitých při vývoji, kde kromě jiného nalezneme krátký popis Google Maps API, které bylo použito pro geolokaci. V předposlední kapitole je popsána implementace dílčích částí vytvářené aplikace podle MVC architektury. Dále je zde představena grafická podoba aplikace. Věnována je také kapitola týkající se zabezpečení a podpory internacionalizace. V poslední kapitole je popsána funkcionalita a obsluha výsledné aplikace. Nalezneme zde podrobně popsány všechny sekce vytvářené aplikace spolu s jejich nabízenými funkcionalitami. Po přečtení této kapitoly by uživatel měl být bez problému schopný uvedenou aplikaci obsluhovat.

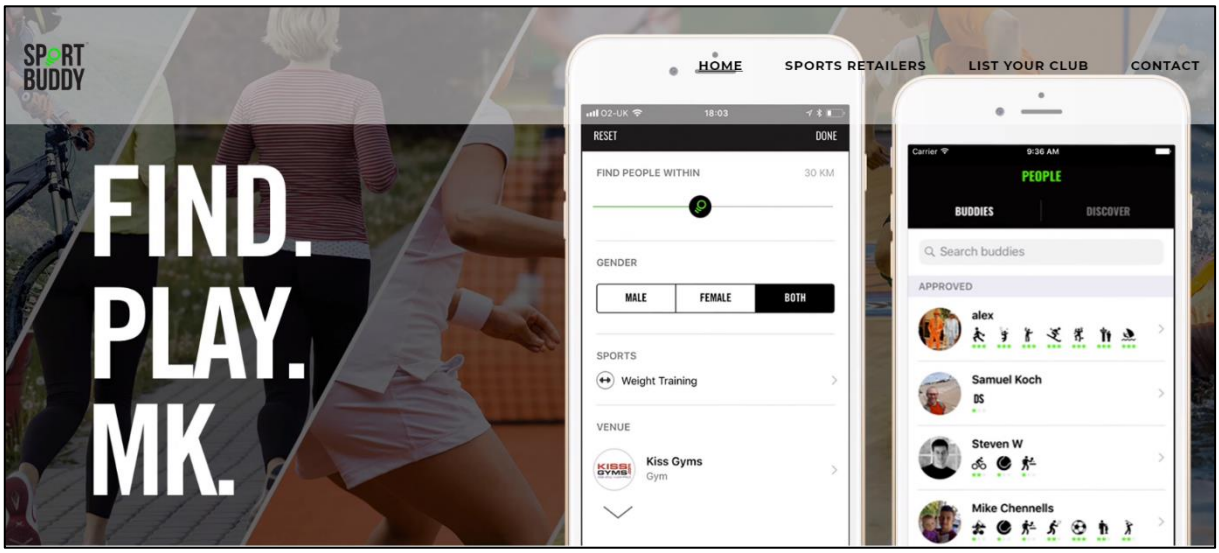
Od čtenáře se očekávají alespoň základní znalosti jazyka Java, zkušenosti s programováním a vědomosti týkající se vývoje webových aplikací obecně. K snadnějšímu porozumění některých kapitol je také vhodná znalost návrhových vzorů, běžných programátorských principů a databázových systémů.

# 1 VYBRANÉ EXISTUJÍCÍ APLIKACE

V nadcházejících podkapitolách je proveden popis vybraných, již existujících aplikací, které se funkcionalitou podobají vytvářené aplikaci v praktické části. Každá aplikace má svá specifika a přednosti, z toho důvodu je v poslední podkapitole provedeno srovnání na základě vybraných dílčích funkcionalit.

## 1.1 Sport buddy

Sport buddy[1] je mobilní aplikace sloužící k nalezení sportovních partnerů v okolí. Dostupná je pro zařízení s operačním systémem iOS i Android. Pochází z Anglie a je zcela zdarma. Tato aplikace byla marketingově uvedena jako Tinder[2] pro fanoušky sportovních aktivit, to znamená především jako seznamovací aplikace. [1], [2]




Obrázek 1 – ukázka aplikace Sport buddy, zdroj [1]

Funkcionalita aplikace spočívá v zadání parametrů, jako je pohlaví, typ sportu, místo apod. Na základě těchto parametrů se uživateli najde nejvhodnější kandidát. V případě, že se mu vybraný kandidát zamlouvá, odešle žádost o navázání kontaktu. Následně uživatel může či nemusí tuto žádost přijmout. Pokud oba uživatelé vzájemně navázání kontaktu potvrdili, dostanou

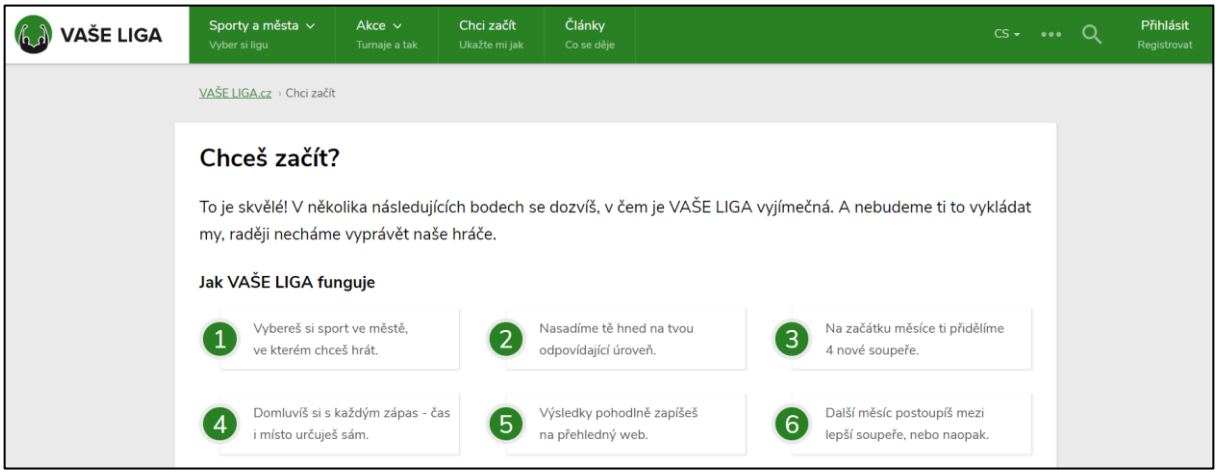
[1] http://sportbuddy.io/
[2] https://tinder.com/

notifikaci, že stojí o seznámení obě strany a mohou se na vybrané aktivitě domluvit, resp. seznámit. [1], [2]

Aplikace nepodporuje soutěživou stránku věci, jako je určení vítěze, poraženého či výsledné skóre v domluvené aktivitě. Jde čistě o nalezení partnera na nějakou aktivitu, resp. o alternativní způsob seznamování. [1], [2]

## 1.2 Vaše liga

Vaše liga[3] je česká webová aplikace založena v roce 2007. Slouží jako amatérská liga především v raketových sportech. Dříve vystupovala pod názvem *VŠEliga*. Účelem této aplikace je nalezení spoluhráčů na vybraný sport. Aktuálně aplikace podporuje šest různých sportů: squash, badminton, tenis, stolní tenis, nově také plážový volejbal a běh. [3]




Obrázek 2 – ukázka aplikace Vaše liga, zdroj [3]

Aplikace funguje tak, že se uživatel přihlásí na vybraný sport a tím pádem je přihlášen do ligy tohoto sportu. Tato liga poté trvá čtyři měsíce. Účastnící ligy jsou podle jejich výkonnosti rozděleny do skupin. Každý měsíc uživatel odehraje čtyři zápasy, každý týden jeden. Po odehrání zápasu uživatelé zadají do aplikace výsledek. Jednou za měsíc probíhá na základě dosažených výsledků posun hráčů do vyšší (nižší) úrovně v lize. [3]

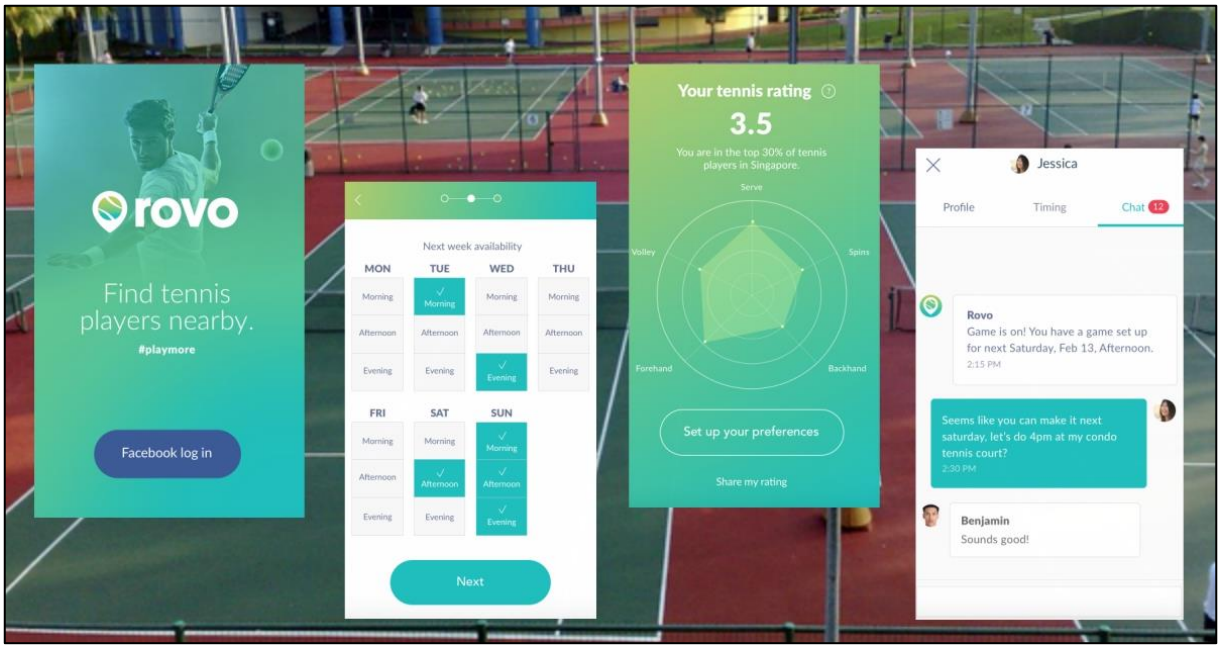
Vaše liga je bohužel placená aplikace, v době psaní této práce je cena za čtyři měsíce 555 Kč. Další nevýhodou je, že se ligy aktuálně vyskytují pouze ve velkých městech: v Praze, Brně, Olomouci, Plzni a Pardubicích. [3]

[3] https://www.vaseliga.cz/

## 1.3 Rovo

Rovo[4] je mobilní aplikace, pocházející ze Singapuru, která byla vytvořena v roce 2016. Aplikace vznikla jako univerzitní projekt dvou studentů, kteří hledali partnery na tenis. Dostupná je pro mobilní zařízení s operačním systémem iOS i Android. Aplikace je zcela zdarma. [4], [5]

Stejně jako předchozí aplikace, i Rovo slouží k hledání partnerů na sportovní aktivity. Aktuálně podporuje čtyři sporty: tenis, badminton, stolní tenis a squash, včetně jejich modifikací, jako jsou např. čtyř hry. V budoucnu je plánována podpora i dalších sportů. [4], [5]




Obrázek 3 – ukázka aplikace Rovo, zdroj [5]

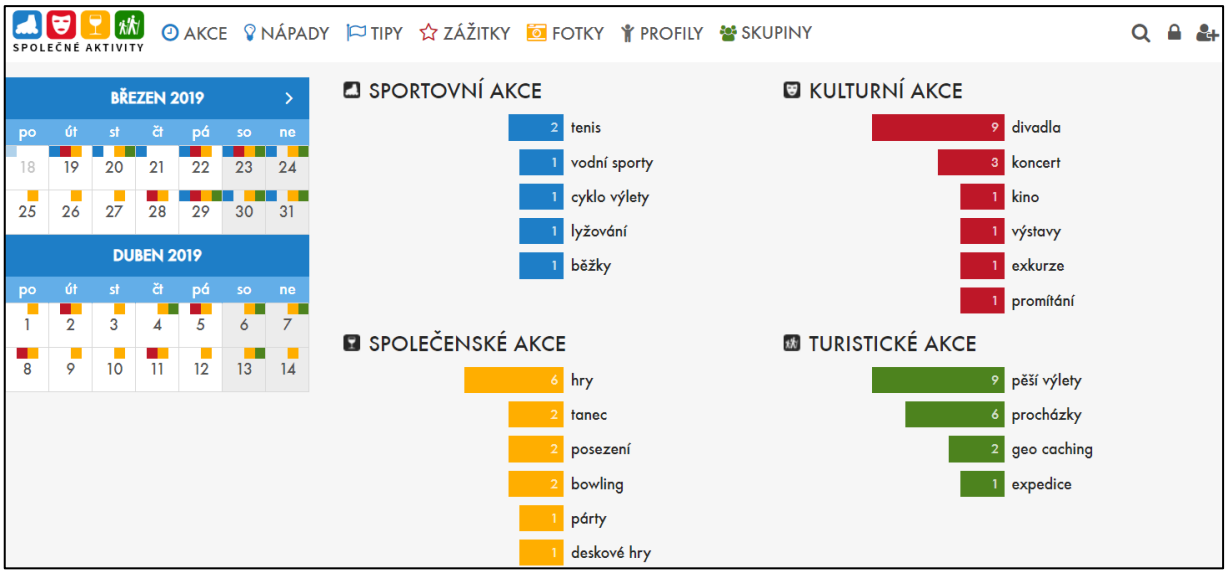
Po přihlášení do aplikace uživatel vyplní krátký dotazník, týkající se se jeho schopností, ve kterém se sám ohodnotí – např. jak dobrý má v tenise *forehand*, *backhand* apod. Poté v kalendáři vyplní datum a čas, kdy má o provozování vybraných aktivit zájem. Podle těchto parametrů si ho ostatní uživatelé mohou vyhledat a na dané aktivitě se poté domluvit. [4], [5]

Aplikace aktuálně nepodporuje hodnocení hráčů, žebříčky a obecně soutěživou stránku věci, nicméně tuto funkcionalitu plánují v blízké době dohotovit. Stejně tak se plánuje rozšíření o další sportovní aktivity. [4], [5]

[4] https://rovo.co/

## 1.4 Společné aktivity

Společné aktivity[5] je česká webová aplikace sloužící k vyhledávání volnočasových aktivit. Její myšlenkou je motivace lidí k aktivnímu životu, ať už to je sport, zábava, kultura, turistika či vzdělávání. V roce 2017 se tento portál dostal dokonce do finále soutěže Křišťálová lupa v kategorii zájmové weby. Užívání aplikace je částečně zdarma. [6]




Obrázek 4 – ukázka aplikace Společné aktivity, zdroj [6]

Jako přednosti webu tvůrci uvádějí možnost nalézt na každý den program ve společnosti příjemných lidí, potkávání nových přátel, nalezení spoluhráčů na nejrůznější sportovní aktivity, kamarádů na turistiku nebo se jen přirozenou cestou seznámit. [6]

Akce na tomto portálu organizují a vytvářejí sami uživatelé. Publikované aktivity procházejí pečlivým schvalovacím procesem, aby se do hlavní nabídky dostaly pouze kvalitní a připravené akce. Organizátorem se může stát každý, pokud přijde se zajímavou myšlenkou a plánem co společně s ostatními podniknout. Společné aktivity tedy nejsou určeny pouze pro sportovní akce, ale pro aktivity všeho druhu. [6]

Ačkoliv Společné aktivity nejsou primárně seznamovacím portálem, seznamkou či profilovou databází, přibližně polovina uživatelů používá tuto aplikaci k nalezení partnera, protože se jedná o snadný způsob, jak se s ostatními lidmi nenuceně a bezpečně seznámit na schválených akcích. [6]

[5] https://www.spolecneaktivity.cz/

## 1.5 Srovnání aplikací

V následující tabulce je shrnuta vybraná funkcionalita aplikací, které byly představeny v předešlých kapitolách.

Tabulka 1 – srovnání aplikací, zdroj vlastní

| | Sport buddy | Vaše liga | Rovo | Společné aktivity | Let's play folks (vytvářená aplikace) |
|---|---|---|---|---|---|
| **Typ aplikace** | Mobilní | Webová | Mobilní | Webová | Webová |
| **Čeština** | Ne | Ano | Ne | Ano | Ano |
| **Zdarma** | Ano | Ne | Ano | Částečně | Ano |
| **Počet aktivit** | 7 | 6 | 4 | Neomezeně | Neomezeně |
| **Hodnocení uživatelů** | Ne | Ano | Ne | Ne | Ano |
| **Aktivita kdekoliv** | Částečně | Ne | Ne | Ano | Ano |
| **Přátelé a zprávy** | Ano | Ne | Ano | Ano | Ano |
| **Statistiky, Historie her** | Ne | Částečně | Ne | Ne | Ano |

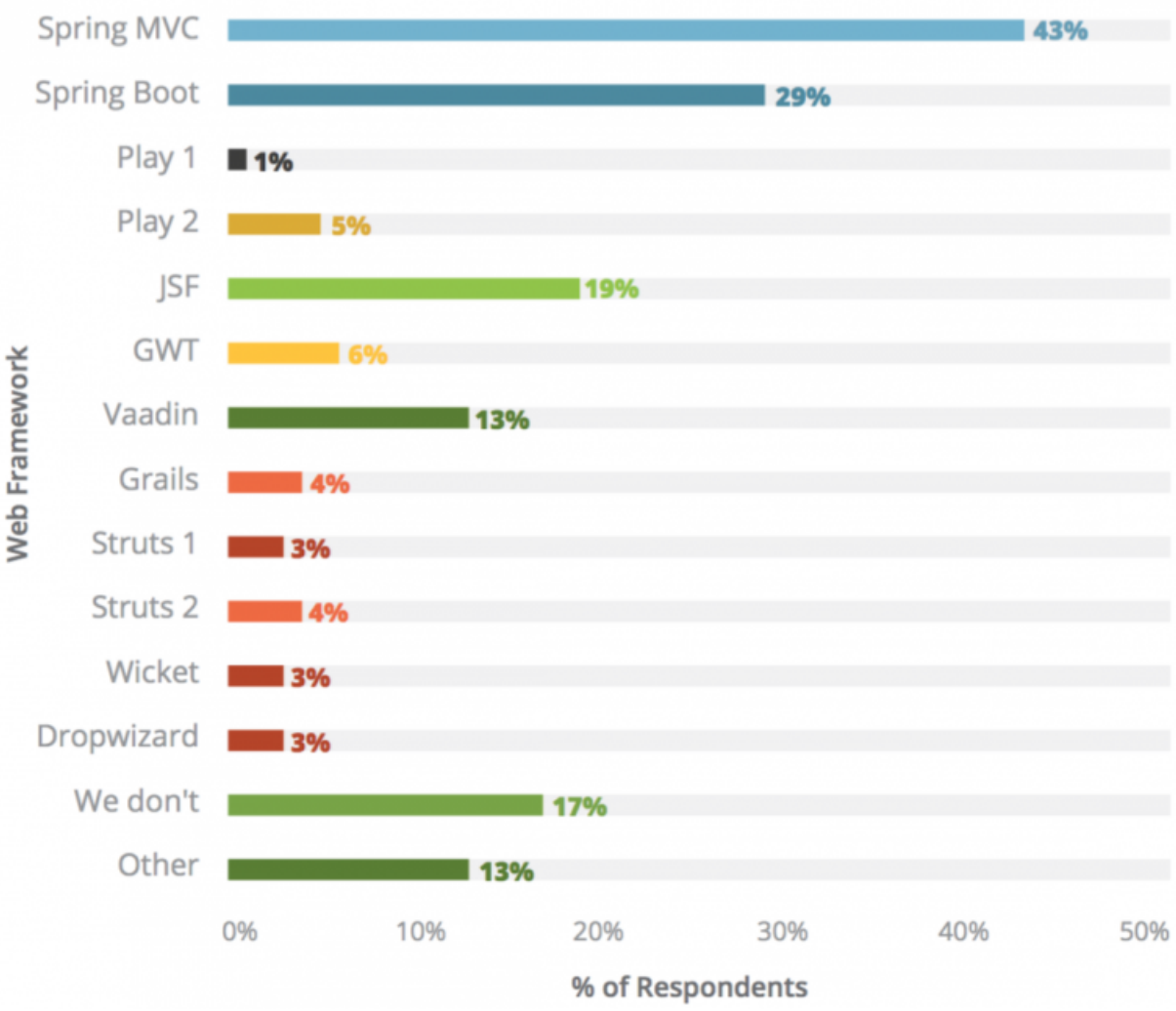
Mobilní aplikace jsou podporovány jak na operačním systému iOS, tak i na Androidu. Webové aplikace jsou funkční ve všech běžných internetových prohlížečích. Čeština je podporována pouze u českých aplikací.

Jako největší nevýhoda porovnávaných aplikací je považována absence soutěživé stránky jednotlivých aktivit, tzn. nepřítomnost hodnocení uživatelů v daných aktivitách, jejich skóre či žebříčky. Ve valné většině případů totiž uživatel sám vyplní své schopnosti a prakticky nezáleží na výsledku odehraného zápasu. Další nevýhodou je omezení sportů. Aplikace jsou směřovány především na raketové sporty. Díky tomu také dost často nelze založit konání aktivity kdekoliv, ale pouze na vybraných kurtech či v tělocvičnách.

Čtenáři je jednoznačně doporučena aplikace Rovo. Tato aplikace plánuje v blízké době napravit všechny nevýhody, které byly zmíněné v minulém odstavci a má ambice stát se jedničkou na trhu, co se týče aplikací určených k vyhledávání sportovních partnerů.

# 2 SPRING FRAMEWORK

Spring je značně populární, robustní, open-source framework[6], podporující různé oblasti vývoje Java EE aplikací. Jeho využití lze nalézt od drobných až po velice rozsáhle aplikace. Největšího uplatnění nachází v korporátních projektech, především pak v oblastech, jako je bankovnictví, pojišťovnictví nebo telekomunikace. Spring je v nynější době patrně nejpoužívanější Java framework, využívaný nejen pro vývoj webových aplikací. Jeho popularitu, v porovnání k ostatním Java frameworkům, lze vidět na následujícím grafu. [7], [8]




Obrázek 5 – popularita Java frameworků, zdroj [8]

Graf byl vytvořen na základě odpovědi účastníků ankety na otázku, jaký Java framework využívají ve svém projektu.

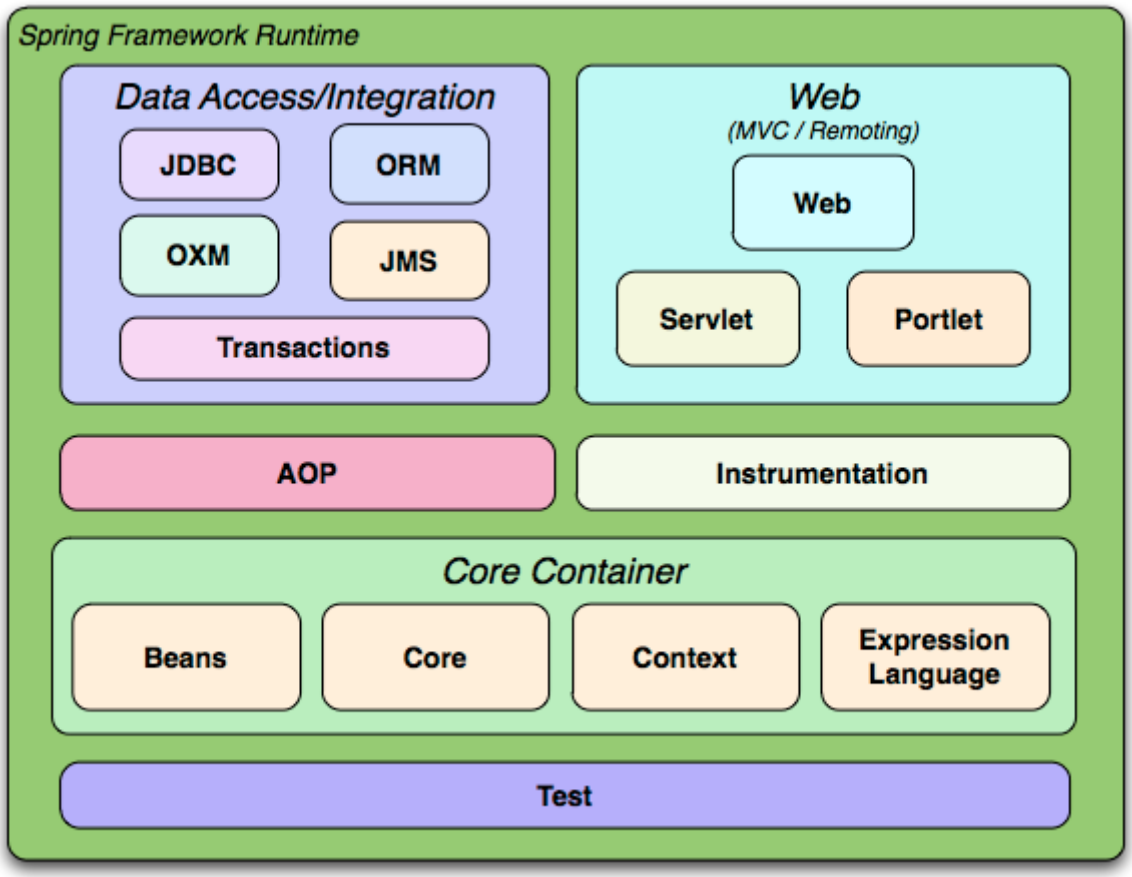
[6] Framework – softwarová struktura sloužící k podpoře programování, vývoje a organizaci projektu.

## 2.1 Moduly

Framework Spring obsahuje mnoho funkcionalit, které jsou pro přehlednost a snadnost použití uspořádány v přibližně 20 modulech. Tyto moduly jsou na základě jejich primární funkce seskupeny do jednotlivých kontejnerů, viz následující výpis: [7]

- Core Container – jádro frameworku, obsahuje IoC, DI, Application context apod.
- Data Access/Integration – přístup a práce s daty z DB, XML atd.
- Web – moduly obsahující dodatečné funkcionality při vývoji webových aplikací.
- AOP – podpora aspektově orientovaného programování.
- Instrumentation – modul pro dynamické načítání tříd do aplikačních serverů.
- Test – podpora testování za použití JUnit nebo TestNG.

Rozdělení modulů do konkrétních kontejnerů je znázorněno na následujícím obrázku.

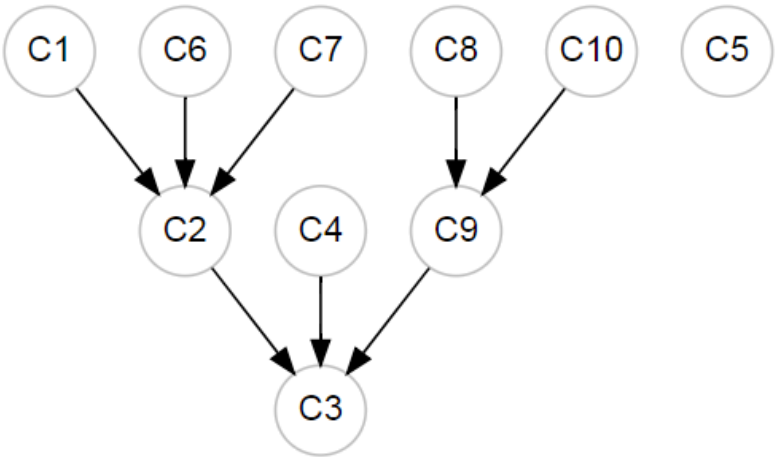



Obrázek 6 – moduly ve Springu, zdroj [7]

## 2.2 Dependency Injection

Zásadní myšlenka a jedna z největších výhod frameworku Spring je princip označován jako *Inversion of Control (IoC)* – převrácené řízení. IoC je obecným principem, který lze uplatnit mnoha způsoby. Konkrétní využití tohoto principu se nejen ve frameworku Spring nazývá *Dependency Injection (DI)* – vkládání závislostí. DI je mechanismus, který do třídy vkládá (injektuje) instanci jiné třídy. [9]

V objektově orientovaném programování se celý program skládá z řady vzájemně propojených objektů. Za předpokladu, že objekt *A* vyžaduje pro svou funkčnost objekt *B*, lze tvrdit, že objekt *A* je na objektu *B* závislý. Neboli objekt *B* tvoří závislost objektu *A*. Tuto situaci si lze představit jako orientovaný graf, ve kterém uzly představují objekty a závislosti mezi těmito objekty jsou znázorněny hranami. Ukázka orientovaného grafu znázorňujícího závislosti mezi třídami pro nějaký program může vypadat například tak, jako na obrázku 7. [9]




Obrázek 7 – ukázka programu jako orientovaného grafu, zdroj [9]

Jakým způsobem si jednotlivé instance určují, které závislosti se použijí a kdo nese zodpovědnost za jejich vytváření? Existují dvě možnosti.

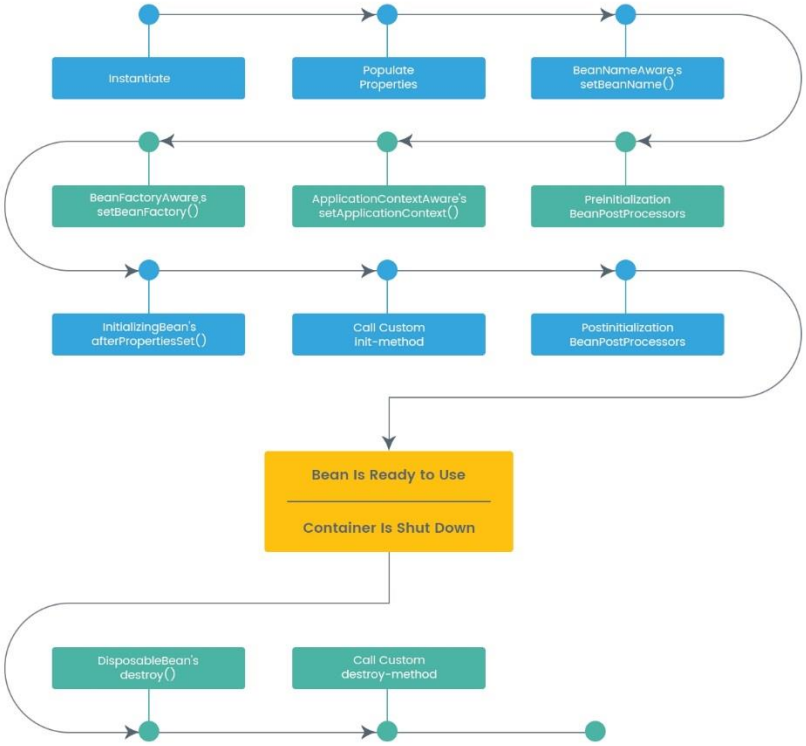
Jednou možností je, že objekt rozhoduje o svých závislostech sám. Ve svém konstruktoru si vytvoří vše potřebné. Výhodou tohoto způsobu je to, že takové objekty lze využívat jako samostatné komponenty. Nevýhodou je, že závislosti tohoto objektu nelze použít jinde nebo je nějak změnit. [9]

Druhý způsob je takový, že se závislosti do objektu vkládají z vnějšku, například pomocí hodnoty v konstruktoru. Při této variantě objekt důvěřuje tomu, že závislosti budou implementovat určité rozhraní a už pro něj není tolik podstatné, jaké implementující instance

to přesně budou. Díky tomu lze takovému objektu například při testování podvrhnout *mocky*[7] místo jeho skutečných instancí. Takovýto způsob se právě nazývá *dependency injection*. O toto vložení závislostí se stará část Springu, která se nazývá *IoC kontejner*. [9]

### 2.2.1 IoC kontejner

Tento kontejner je nejdůležitější částí Spring frameworku. Jeho zodpovědností je vytváření objektů, konfigurace, provázanost a řízení jejich celého životního cyklu, od vytvoření až po zničení. IoC kontejner je zaveden do paměti ihned při startu aplikace. Konkrétní implementací tohoto kontejneru je nejčastěji rozhraní *ApplicationContext* (aplikační kontext) a v celé aplikaci se vyskytuje pouze jeden. Objekty, které aplikační kontext spravuje, se ve Springu nazývají *beany* (fazole).




Obrázek 8 – životní cyklus Spring beany, zdroj [10]

*Beany* jsou objekty, které „žijí“ a jsou obsluhovány uvnitř kontejneru po celou dobu jejich životního cyklu. Ten je pro ilustraci znázorněn na obrázku 8.

[7] Mock – simulační objekt, který napodobuje chování reálných objektů, nejčastěji využíván pro testování.

Konfigurovat aplikační kontext je možné pomocí XML[8] souborů nebo pomocí obyčejných Java souborů (tříd). Obě možnosti jsou funkčně totožné. Způsob konfigurace pomocí XML zde byl od prvních verzí Springu, způsob konfigurace pomocí Javy byl představen až v novějších verzích. Velice často se oba způsoby kombinují a část konfigurace je poté v Javě a část v XML, záleží pouze na preferenci programátora nebo konvenci zavedené v projektu. V praktické části práce je dána přednost psaní konfiguračních souborů novějším, modernějším způsobem, tedy pomocí Java tříd. [11]

Ukázku Java konfigurace lze vidět v následující ukázce kódu.

```
// jedná se o konfiguraci
@Configuration
// naimportuje konfiguraci z třídy StorageConfig
@Import({StorageConfig.class})
// skenuje cz.itnetwork a tvoří beany (@Component, @Service...)
@ComponentScan("cz.itnetwork")
public class ContextConfig {
        // vytvoří beanu typu CarDao a názvem carRepository
        @Bean(name="carRepository")
        public CarDao carDao() {
                return new CarDaoImpl();
        }

        // vytvoří beanu CarService a injektuje ji CarDao (CarRepository)
        @Bean
        @Autowired
        public CarService carService(CarDao carDao) {
                return new CarServiceImpl(carDao);
        }
}
```

Zdrojový kód 1 – ukázka konfigurace pomocí Javy, zdroj [11]

Na předchozí ukázce kódu lze vidět, že pro injektování objektů se ve Springu používá anotace *@Autowired*. Díky tomu je tato anotace pravděpodobně nejčastěji využívanou anotací ve Springu vůbec. Mechanismy pro vkládání (injektování) objektů používáme ve Springu tři. Vkládání konstruktorem (constructor injection), *setterem* (setter injection) a *fieldem* (field injection). [9], [11], [12]

[8] XML – značkovací jazyk určený především pro výměnu dat mezi aplikacemi nebo konfigurační soubory.

### 2.2.2 Vkládání závislostí pomocí konstruktoru

Při využití *constructor injection* jsou závislosti třídy definovány rovnou v konstruktoru. Výhodou tohoto přístupu je to, že veškeré závislosti jsou definovány společně. Díky tomu je zajištěno, že objektu je předáno vše, co potřebuje ke svému životu a bude po svém vytvoření rovnou funkční. [9], [13], [14]

```
@Component
public class ConstructorBasedInjection {

    private final InjectedBean injectedBean;

    @Autowired
    public ConstructorBasedInjection(InjectedBean injectedBean) {
        this.injectedBean = injectedBean;
    }

}
```

Zdrojový kód 2 – ukázka DI pomocí konstruktoru, zdroj vlastní

Hlavním plusem DI pomocí konstruktoru je možnost deklarovat závislosti jako *final*[9], jelikož budou inicializovány až během vytváření instance třídy. Při použití *setter injection* i *field injection* je takto deklarovat totiž nelze. Jako další výhodu lze zmínit to, že na závislosti nelze zapomenout, jelikož na chybějící či nesprávné parametry je upozorněno již kompilátorem.

Nevýhodou může být nepřehledný zápis v případě velkého množství závislostí, konstruktor se stává se zvětšujícím množstvím parametrů nepřehledný, obtížně čitelný a špatně použitelný. Tento problém lze řešit sjednocováním závislostí do nového objektu, který je pak v konstruktoru použit. Díky tomu je pak počet parametrů konstruktorů zkrácen na rozumný počet. Dobrá praktika je, aby počet parametrů nebyl více než čtyři. [9], [13], [14]

[9] Final – po inicializaci objektu označeného tímto klíčovým slovem ho nelze měnit.

### 2.2.3 Vkládání závislostí pomocí setteru

V případě použití *setter injection* je pro vkládání závislostí použito metody *set*. Každá závislost má takovouto obsluhující metodu, která je poté zavolána před prvním použitím daného objektu. [9], [13], [14]

```
@Component
public class SetterBasedInjection {

    private InjectedBean injectedBean;

    @Autowired
    public void setInjectedBean(InjectedBean injectedBean) {
        this.injectedBean = injectedBean;
    }

}
```

Zdrojový kód 3 – ukázka DI pomocí setteru, zdroj vlastní

Nevýhodou tohoto přístupu je, že není zaručeno, zdali bude tato metoda před použitím opravdu zavolána a díky tomu je možné, že vykonávání programu padne na výjimce, protože bude chybět potřebná závislost. V případě Springu se o toto však starat nemusíme. Vkládání závislostí si řídí framework, resp. IoC kontejner sám[10]. [9], [13], [14]

Další vlastností použití *setterů* pro vkládání závislostí, ať už výhodou či nevýhodou je to, že lze závislosti za běhu dynamicky měnit. [9], [13], [14]

[10] Aplikační kontext si tuto metodu zavolá sám, kdykoli bude tato závislost chybět.

### 2.2.4 Vkládání závislostí pomocí fieldu

Poslední možností, která je ve Springu možná použít pro vkládání závislostí je *field injection*. Díky tomu dochází ke vkládání závislostí přímo do atributu třídy. Tento přístup je oficiálně nedoporučován, nicméně ze subjektivního hlediska autora je i přesto nejvíce rozšířený. Tento způsob vkládání závislostí je také použit v praktické části práce. [9], [13], [14]

```
@Component
public class FieldBasedInjection {

    @Autowired
    private InjectedBean injectedBean;

}
```

Zdrojový kód 4 – ukázka DI pomocí fieldu, zdroj vlastní

Hlavní výhoda je na první pohled zřejmá – tento způsob je ze všech zmíněných jasně nejkratší. Není potřeba psát konstruktor nebo další metody, které poté zbytečně třídu zahlcují „nepotřebným" kódem. Třída poté vypadá čistě, jednoduše a přímočaře. Jak bylo ale zmíněno v minulém odstavci, tento způsob není oficiálně doporučován, a to hned z několika důvodů.

Nelze závislosti deklarovat jako *final*, jako tomu bylo možné u *construction injection*. Je velice jednoduché porušit princip jedné zodpovědnosti, protože přidání další závislosti jsou pouze dva řádky navíc. V případě použití *construction injection* na první pohled vidíme, kdy začíná být konstruktor nepřehledný a v případě *setter injection*, je také zřejmé, kdy už třída obsahuje více *setterů* než je rozumné. *Field injection* není přímo zodpovědný za porušení zmíněného principu, ale nepřímo tomu napomáhá a skrývá náznaky toho, že kód není napsán úplně správně. Další nevýhodou je skrytí závislostí vkládaných objektů. V případě použití konstruktoru je hned zřejmé, co objekt požaduje, taktéž je snadné vidět, které závislosti daný objekt potřebuje v případě použití *setterů*. V případě *field injection* jsou tyto závislosti na první pohled skryty. Poslední, pravděpodobně největší nevýhodou je to, že objekty jsou těsně svázány se samotným Springovským IoC kontejnerem. To znamená, že bez něho nemohou být tyto třídy inicializovány, protože by závislosti byly samozřejmě *null*. Při použití *field injection* zde totiž chybí nějaký způsob, jak je inicializovat ručně – ať už pomoci konstruktoru nebo pomocí metod jako jsou například *settery*. [9], [13], [14]

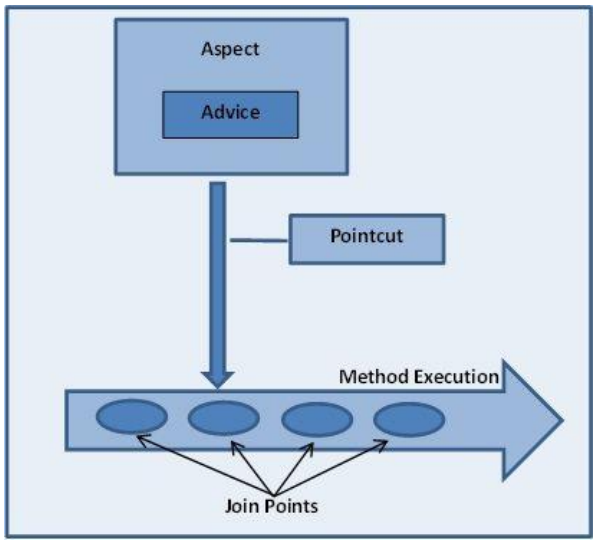
I přes veškeré nevýhody se tento způsob jeví jako nejjednodušší, nejpřímočařejší a v případě obezřetného psaní kódu se většina nevýhod eliminuje a zbyde především výhoda – čistota kódu.

## 2.3 Spring AOP

Aspektově orientované programování (AOP) je programovací paradigma doplňující OOP[11]. Primárním cílem tohoto přístupu je eliminace často se opakujících částí kódu. Typickým příkladem může být logování, *exception handling*, řízení přístupu pomocí uživatelských práv, *cachování* apod. [15]

Pro lepší pochopení následujících odstavců jsou zde představeny základní termíny z AOP, nicméně terminologie, která se k aspektově orientovanému programování váže, je opravdu rozsáhlá a její výčet by dal na několik stran. [15]

- Aspekt (aspect) – v OOP je základním stavebním kamenem třída, v AOP je tomu ekvivalentní aspekt. Je to objekt, který zapouzdřuje často se opakující části kódu.
- Bod vstupu (joinpoint) – místo ve vykonávání programu, ve kterém se daný *aspect* uplatňuje. Typicky se jedná o nějakou metodu, inicializaci objektu apod.
- Pokyn, rada (advice) – určení v jaké situaci se má *aspect* na daném *joinpointu* uplatnit. Typicky se jedná o situace *before* (před) nebo *after* (po) – to znamená např. před zavoláním nějaké metody, po vytvoření nějakého objektu apod.
- Bod vykonání (pointcut) – množina *joinpointů*, pro které platí stejné *advice*.




Obrázek 9 – Spring AOP, zdroj [15]

Na předešlém obrázku je graficky znázorněna komunikace popsaných prvků AOP.

[11] Objektově orientované programování – programovací paradigma využívající abstrakce reálného světa.

Principem AOP je tedy „zabalení“ často se opakujících částí kódu do *aspektů*. Tyto aspekty jsou poté navázány na určitý *joinpoint*, na kterém je provedena jejich činnost v situaci, kterou určuje *advice*.

Framework Spring obsahuje bohatou podporu pro AOP, a to pomocí modulu, který se nazývá Spring AOP. Tento modul je velice důležitou součástí tohoto frameworku a poskytuje prostor pro řešení valné většiny běžných problémů týkajících se AOP. Jednoduchost použití tohoto modulu je demonstrována na následující ukázce zdrojového kódu.

```
@Aspect
public class ExampleAspect {

    @Before("execution(public String setScore())")
    public void setScoreAdvice(){
        //...some logic which should process before method setScore() is called.
    }

    @After("execution(* com.bartosektom.letsplayfolks.service.*.get*())")
    public void getAllAdvice(){
        //...some logic which should process after any getter in service package is called.
    }
}
```

Zdrojový kód 5 – ukázka Spring AOP, zdroj vlastní

První metoda se zavolá vždy před zavoláním metody *setScore()*. Druhá metoda se zavolá vždy po zavolání jakéhokoliv *getteru*, který se nachází v balíčku *service*. Do těchto metod poté můžeme dát libovolnou logiku, ať už to je zmíněné logování nebo omezení podle uživatelských práv.

Spring AOP podporuje kompletní implementaci v jazyku Java a lze ho konfigurovat pomocí XML. Díky tomu není třeba využívat dalších technologií, které slouží k podpoře AOP a použití je velice snadné. Nevýhodou je podpora *joinpointů* pouze na metody, tím pádem nelze použít aspekty na konstruktory apod. Nicméně i přes to by dosavadní řešení mělo být dostatečné pro většinu požadavků. V případě, že si programátor nevystačí pouze s poskytovanými funkcionalitami, poskytuje Spring AOP také výbornou integraci s velice známým AOP frameworkem AspectJ[12]. Na závěr je vhodné zmínit, že pokud aspektově orientované programování v aplikaci nevyužijeme, Spring nám nic nevnucuje a využívat ho vůbec nemusíme nebo ho lze využívat pouze nepřímo, a to například díky Spring Security, jako je tomu i v případě praktické části této práce. [16], [17]

[12] AspectJ – Populární framework poskytující podporu AOP pro jazyk Java.

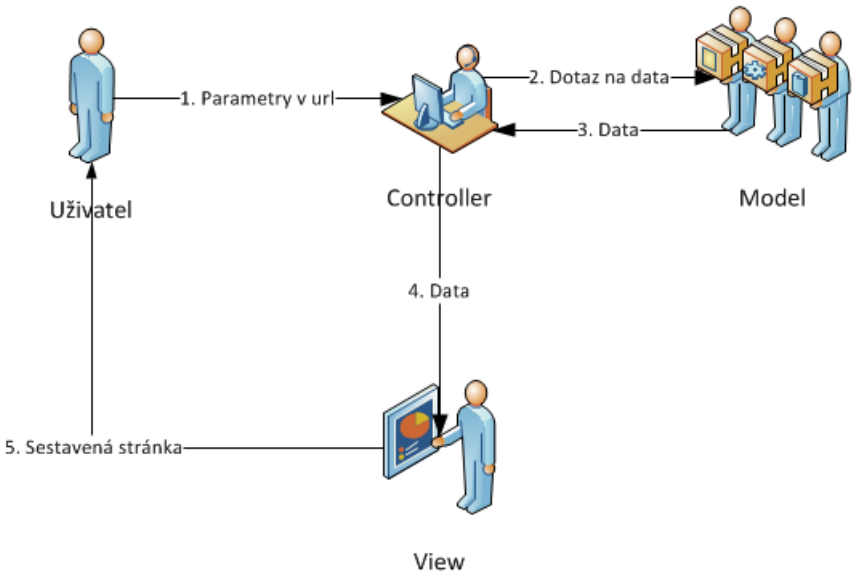
## 2.4 Spring MVC

V této části práce je popsána podpora architektonického vzoru MVC, kterou framework Spring poskytuje.

### 2.4.1 MVC Architektura

Model-View-Controller (MVC) je stále populárním architektonickým vzorem, používaným především pro vývoj webových aplikací, přestože původně vznikl pro desktopové aplikace. Na jeho principu je založeno nemalé množství frameworků, jako je například české Nette či v dnešní době velice populární Laravel (PHP), ASP.NET (C#) nebo již zmíněný Spring. [18]

Stěžejní myšlenkou této architektury je oddělení logiky od výstupu. Toho je dosaženo pomocí rozdělení aplikace na komponenty, z nichž každá zastřešuje určitou logickou část aplikace. Tento přístup umožňuje poté psát čistější kód a komponenty dodržující princip jedné odpovědnosti[13]. [18]




Obrázek 10 – princip MVC architektury, zdroj [18]

Tyto komponenty jsou běžně označovány jako *Model*, *View*, *Controller*.

[13] Princip jedné odpovědnosti – Každý modul, třída či funkce by měla mít pouze jednu zodpovědnost.

- *Model* má za úkol obsluhovat logickou část aplikace. Tím může být například dotaz do databáze, validace, různé výpočty apod. Zásadní je, že model neví nic o výstupu, jeho účelem je pouze na vstup nějaká data dostat, tyto data určitým způsobem zpracovat a poté nějaké data vrátit. Jak s těmito daty bude následně naloženo, model nezajímá.
- *View* neboli česky *pohled*, slouží k zobrazení dat, které jsou získány právě z modelu. Poté co jsou tyto data získána je nezbytné je také uživateli ve vhodné podobě zobrazit. Tato komponenta by měla obsahovat pouze minimální množství logiky, a to pouze nezbytnou pro vhodné reprezentování dat. Tím může být například cyklus, podmínka, validace apod. Pohled, stejně jako model, nezajímá, odkud k němu data přišla, jeho úkolem je pouze tyto data uživateli zobrazit.
- *Controller* je jakýmsi prostředníkem mezi předešlými prvky, se kterým komunikuje *model, view*, ale i uživatel. Jeho úkolem je zajištění vzájemné spolupráce jednotlivých komponent.

Díky návrhu aplikace podle MVC architektury je také ulehčeno myšlení při vývoji částí aplikace. V případě, že je potřeba naprogramovat nějakou logickou část, již dopředu víme, že bude patřit do modelu. Směrování a zpracování parametrů z URL adresy se bude řešit v *controlleru* a ke stylování a formátování výstupu poslouží *view*. Každý z těchto problému je tedy oddělený na jiném místě tak, aby si vzájemně do své činnosti nezasahovaly a nedělaly samotný vývoj složitějším více než je třeba. [18]

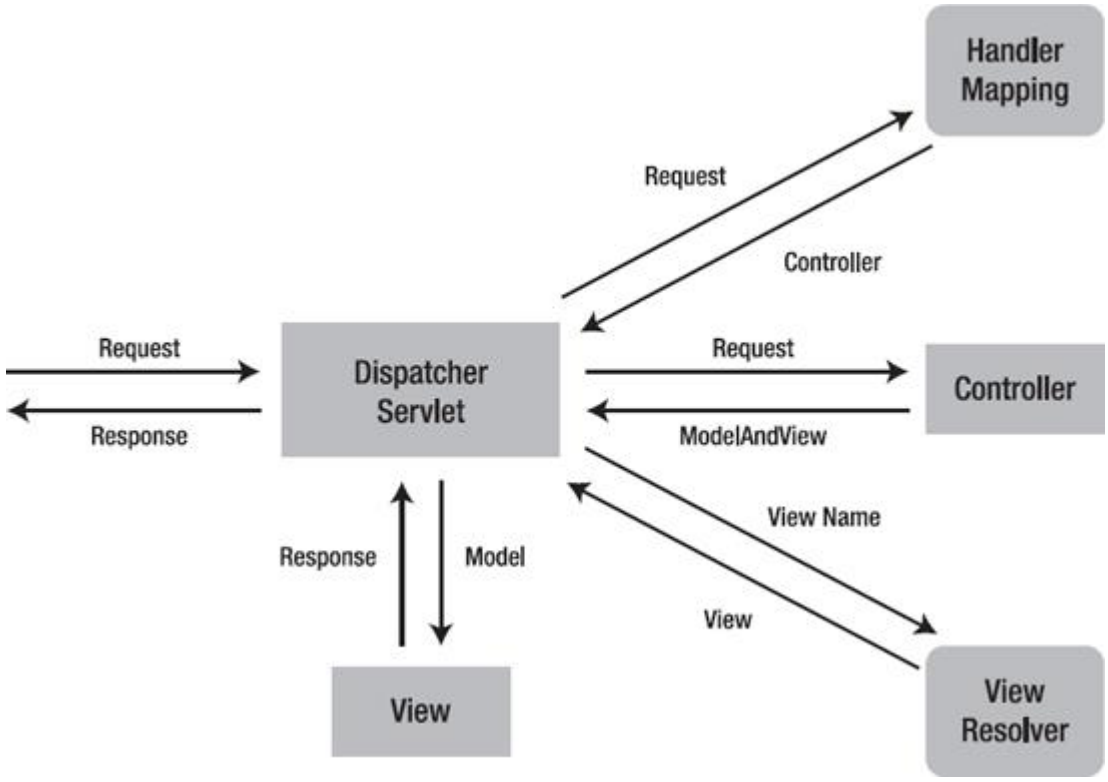
### 2.4.2 Dispatcher Servlet

Spring MVC, stejně jako velké množství ostatních webových frameworků, je navržen okolo návrhového vzoru *front controller*[14]. Dá se říci, že tento vzor doplňuje klasickou MVC architekturu o další část, která centralizuje zpracování jednotlivých požadavků a tím pádem přispívá k celkovému zjednodušení kódu.

Konkrétní implementací tohoto návrhového vzoru ve frameworku Spring je *Dispatcher Servlet* (servlet v pozici dispečera). Jak je již z názvu patrné, tento objekt tvoří jediné vstupní i výstupní místo, přes které procházejí veškeré HTTP *requesty* (požadavky), resp. *response* (odpovědi).

[14] Front controller – návrhový vzor sloužící k zapouzdření zpracovávaných požadavků na jedno místo.

Na následujícím obrázku můžeme vidět schématické zobrazení práce *dispatcher servletu*. Na první pohled je zřejmé, že jeho primární úlohou je delegace činností, ostatním částem aplikace, při zpracování HTTP požadavku.




Obrázek 11 – dispatcher servlet, zdroj [19]

Při obdržení *requestu* je nejdříve nutné zjistit, který *controller* má na starosti tento požadavek zpracovat. K tomuto účelu slouží objekt typu *HandlerMapping* (tento objekt bude ještě popsán v kapitole *2.4.3 Mapování požadavků*).

Po nalezení příslušného *controlleru* mu je požadavek předán ke zpracování. Ve většině případů není potřeba předávat celý požadavek, ale obsluhující metoda v *controlleru* si vystačí pouze s určitými parametry tohoto requestu. Po zpracování požadavku je *Dispatcher Servletu* vrácen objekt *ModelAndView*. Tento objekt obsahuje název *view* (pohledu) a naplněný objekt typu *Model*. Tento objekt je typu *Map<String, Object>* a představuje pouze mapu naplněnou hodnotami, které se poté předají pohledu k zobrazení.

Jak již bylo naznačeno v předchozím odstavci, účelem *ViewResolveru* je poté na základě názvu pohledu nalézt příslušný *view*, který se má zobrazit a předá tuto informaci *Dispatcher Servletu*. Ten již pouze tomuto pohledu předá daty naplněný *model* a nic mu nebrání odeslat klientovi odpověď.

### 2.4.3 Mapování požadavků

V předešlé kapitole bylo řečeno, že objekt *HandlerMapping* určí *controller*, který příslušný požadavek obslouží. Určení správného *controlleru* se děje na základě anotace *@RequestMapping*. Anotaci je možné umístit jak před definici třídy, tak před definice jednotlivých metod. Také je samozřejmě možné oba přístupy kombinovat a díky tomu zpřehlednit zpracování konkrétních požadavků.

Tato anotace obsahuje velké množství parametrů, nicméně ve většině případů si vystačíme pouze s následujícími dvěma:

- Parametr *value* – konkrétní hodnota adresy požadavku přijatého z URL adresy.
- Parametr *method* – stanovuje druh HTTP požadavku, který daná metoda obsluhuje, nejběžnějšími zástupci jsou: GET a POST.

```
@Controller
@RequestMapping("/friends")
public class FriendController {

    // some class attributes

    @RequestMapping(value = "/list", method = RequestMethod.GET)
    public ModelAndView showFriendsList(Map<String, Object> model) {
        // some logic there
        return new ModelAndView( viewName: "friend/list", model);
    }

    @RequestMapping(value = "/addToFriend", method = RequestMethod.POST)
    public ModelAndView addToFriend(Map<String, Object> model) {
        // some logic there
        return new ModelAndView( viewName: "friend/addToFriend", model);
    }
}
```

Zdrojový kód 6 – mapování požadavků, zdroj vlastní

Na předchozím příkladu lze vidět použití anotace *@RequestMapping* na celém *controlleru* a zároveň také na jeho jednotlivých metodách.

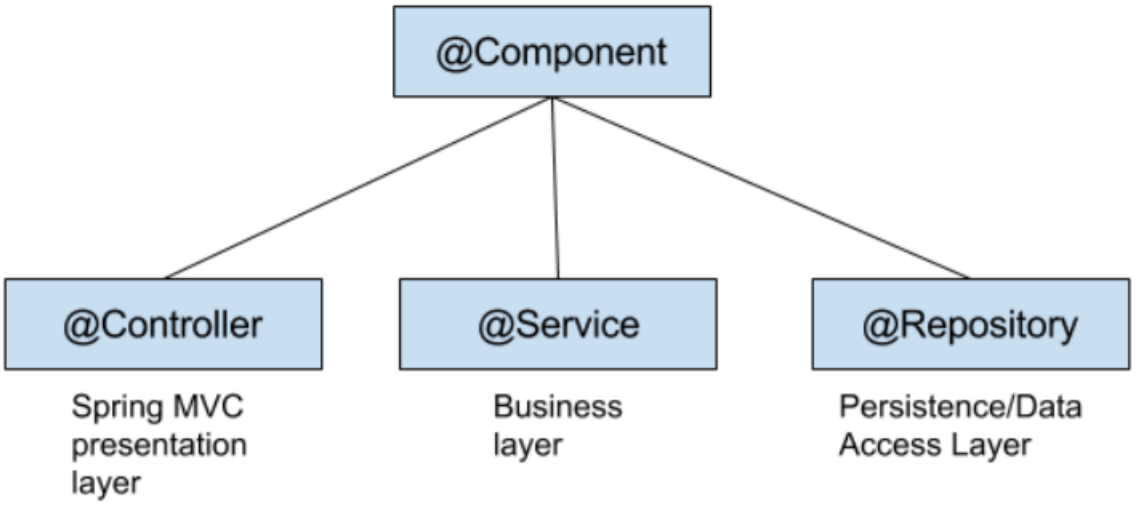
V případě URL adresy: *http://diplomovaprace/friends/list* nejdříve *HandlerMapping* nalezne příslušný *controller*, který má na starosti zpracování */friends* požadavku a poté v něm najde a zavolá metodu, která má na starosti obsluhovat adresu požadavku */list*.

### 2.4.4 Komponenty

Komponenta je ve Springu označení třídy, která slouží ke specifickému účelu. Jako příklad lze uvést *controller*, pro který je vyhrazena anotace *@Controller*. Při skenování anotací v jednotlivých třídách se poté podle těchto anotací určí, která třída je jakou komponentou. V následujícím výčtu jsou tyto komponenty krátce představeny: [20]

- *@Component* – je obecnou anotací určenou pro komponenty, ze které ostatní anotace dědí. V případě, že nejsme schopni určit jakého typu komponenty třída je, použijeme tuto anotaci.
- *@Controller* – jak již bylo naznačeno výše, tato anotace označuje komponentu typu *controller*.
- *@Repository* – tato anotace je použita pro třídy, jejichž účelem je zprostředkování přístupu k datům.
- *@Service* – označuje třídu představující určitou službu, resp. má úlohu obsluhy.

Hierarchie komponent ve Springu vypadá následovně:




Obrázek 12 – hierarchie komponent, zdroj [20]

Aby bylo možné využívat Spring MVC komponent je nezbytné v konfiguračním souboru tuto funkcionalitu nakonfigurovat. Toho je docíleno XML tagem *<mvc:annotation-driven />*. Tím je aplikaci povoleno používání anotací. Dále je nutné určit, v jakém balíčku, resp. třídě se mají komponenty vyhledávat. To je zprostředkováno pomocí XML tagu, *<context:component-scan/>*. Alternativou je poté samozřejmě použití Java konfigurace, a to pomocí anotace *@EnableWebMvc*, resp. *@ComponentScan*. Příklad použití pak vypadá následovně:

```
@Configuration
@EnableWebMvc // equivalent to XML <mvc:annotation-driven />
@ComponentScan({ "com.bartosektom.letsplayfolks.*"}) // equivalent to XML <context:component-scan/>
public class ApplicationConfig {
```

Zdrojový kód 7 – konfigurace komponent

Je vhodné také zmínit, že anotace *@Configuration*, která se používá pro konfiguraci aplikačního kontextu pomocí Java tříd, je sama o sobě na pozadí označena jako *@Component*. Díky tomu lze i konfigurační třídy automaticky vyhledávat pomocí *@ComponentScan*.

### 2.4.5 Spring Boot

Jelikož projekty ve Springu obsahují spousty konfiguračních souborů, které jsou plné různých konkrétních konfigurací, je pro nováčka značně obtížné se v těchto nastaveních vyznat a zprovoznit vůbec klasický *Hello world*. Tento problém se snaží řešit framework Spring Boot. Spring Boot je framework jenž je nadstavbou Springu. Poskytuje rychlejší a jednodušší způsob konfigurace projektu. Díky tomu není např. potřeba konfigurovat věci, jako tomu bylo v předešlé kapitole pomocí *@EnableWebMvc a @ComponentScan*, protože se o toto postará automaticky sám Spring Boot. Kromě jiného, v sobě také obsahuje nakonfigurovaný aplikační server, takže programátorovi odpadá i tato starost. [33]

```
@SpringBootApplication
public class Application extends SpringBootServletInitializer {
```

Zdrojový kód 8 – konfigurace pomocí Spring Bootu, zdroj vlastní

Samotné použití Spring Bootu nemůže být pak jednodušší, jediné, co je potřeba k jeho zprovoznění, je použití anotace *@SpringBootApplication*. Jednoduše řečeno se dá říci, že tato anotace zapouzdřuje několik ostatních, již *defaultně* nakonfigurovaných, nastavení. V případě potřeby jdou samozřejmě upravit a přenastavit podle potřeb programátora.

# 3 REALIZACE APLIKACE PRO TVORBU AKTIVIT

V následujících kapitolách jsou popsány jednotlivé kroky uskutečněné při vývoji aplikace. Od analýzy, přes výběr technologií, díky kterým bude aplikace realizována, až po její samotnou implementaci.

## 3.1 Analýza aplikace

Ještě předtím, nežli je bezmyšlenkovitě započat vývoj aplikace, je velice rozumné provést analýzu. To znamená získat od klienta jeho představu o výsledné aplikaci, jak má graficky vypadat, jakou má mít funkcionalitu, popř. další speciální požadavky, které klient na aplikaci nárokuje. V neposlední řadě je více než dobré, ujasnit si finanční strop a časové termíny.

Poté nad těmito klientskými požadavky následuje zamyšlení, díky kterému získáme určitý směr, jak danou aplikaci vyvíjet. Na tomto základě jsme pak schopni provést prvotní návrh databáze, ujasnit si, jak se má výsledná aplikace chovat a toto chování přenést například do UML diagramů.

Důležité je si uvědomit, že prvotní analýza aplikace nebude téměř nikdy bez chyb, protože je skoro jisté, že se během implementace narazí na problém, o kterém se při analýze nepřemýšlelo. Analýza tedy není proces, který je proveden pouze jednou (i když takový je samozřejmě záměr), ale postupně se k ní vracíme i při vývoji aplikace.

### 3.1.1 Funkční požadavky

Jsou požadavky určující funkcionalitu aplikace. Definují, co má aplikace dělat. Pro drtivou většinu klientů jsou požadavky jedinými, protože je technická stránka věci moc nezajímá a raději tuto starost přenechají na nás. V případě této aplikace má klient požadavky následující.

Vytvoření internetové aplikace na tvorbu a správu jednotlivých aktivit. Obsluha bude rozdělena na čtyři uživatelské role, a to: nepřihlášený uživatel, přihlášený uživatel, operátor aktivit a administrátor celé aplikace.

Nepřihlášený uživatel bude mít možnost zobrazení mapy s jednotlivými aktivitami konajícími se v jeho okolí a bude mu umožněno zobrazení základních informací o těchto aktivitách. Dále bude mít samozřejmě možnost registrace, přihlášení či zobrazení aktuálních novinek.

Přihlášenému uživateli náleží veškerá oprávnění jako uživateli nepřihlášenému. Kromě toho si bude moci zobrazit plný detail jednotlivých aktivit, které uvidí na mapě, přihlásit se na ně

a po odehrání zadat výsledek. Dále bude mít možnost přidat si přátelé, psát zprávy, editovat svůj profil, zobrazovat historii jednotlivých her či kompletní žebříčky.

Role operátor má veškeré možnosti jako přihlášený uživatel, nicméně účelem uživatele s touto rolí je rozhodování sporných her – v případě, že někdo na aktivitu nedorazí i přes svoje potvrzení nebo zadá-li odlišný výsledek od svého soupeře. Tyto uživatelé zastávají roli soudce nerozhodných výzev.

Administrátor může využívat veškerou funkcionalitu, kterou aplikace nabízí. Oproti předešlým rolím má možnost schvalovat nové aktivity navržené jednotlivými uživateli. Také má přístup do administrátorské konzole, kde může upravovat jakékoliv data.

Posledním požadavkem je podpora internacionalizace aplikace, tedy její vícejazyčnost. Jelikož dnes není problém být za pár hodin kdekoliv na světě, po kterém se téměř všude domluvíme anglicky, tak je požadováno, aby aplikace byla dvoujazyčná – tedy aby podporovala jazyk český a jazyk anglický. Samozřejmě se počítá v budoucnu i s podporou ostatních světových jazyků.

### 3.1.2 Nefunkční požadavky

Jak je již z názvu patrné, nefunkční požadavky jsou takové, které nemají vliv na uživatelskou funkcionalitu aplikace. Lze říci, že jsou doplňkem k funkčním požadavkům. Popisují další požadované vlastnosti aplikace vzhledem k jejímu prostředí. Jedná se například o nároky na bezpečnost, podporu během provozu, spolehlivost, výkonnost aj.

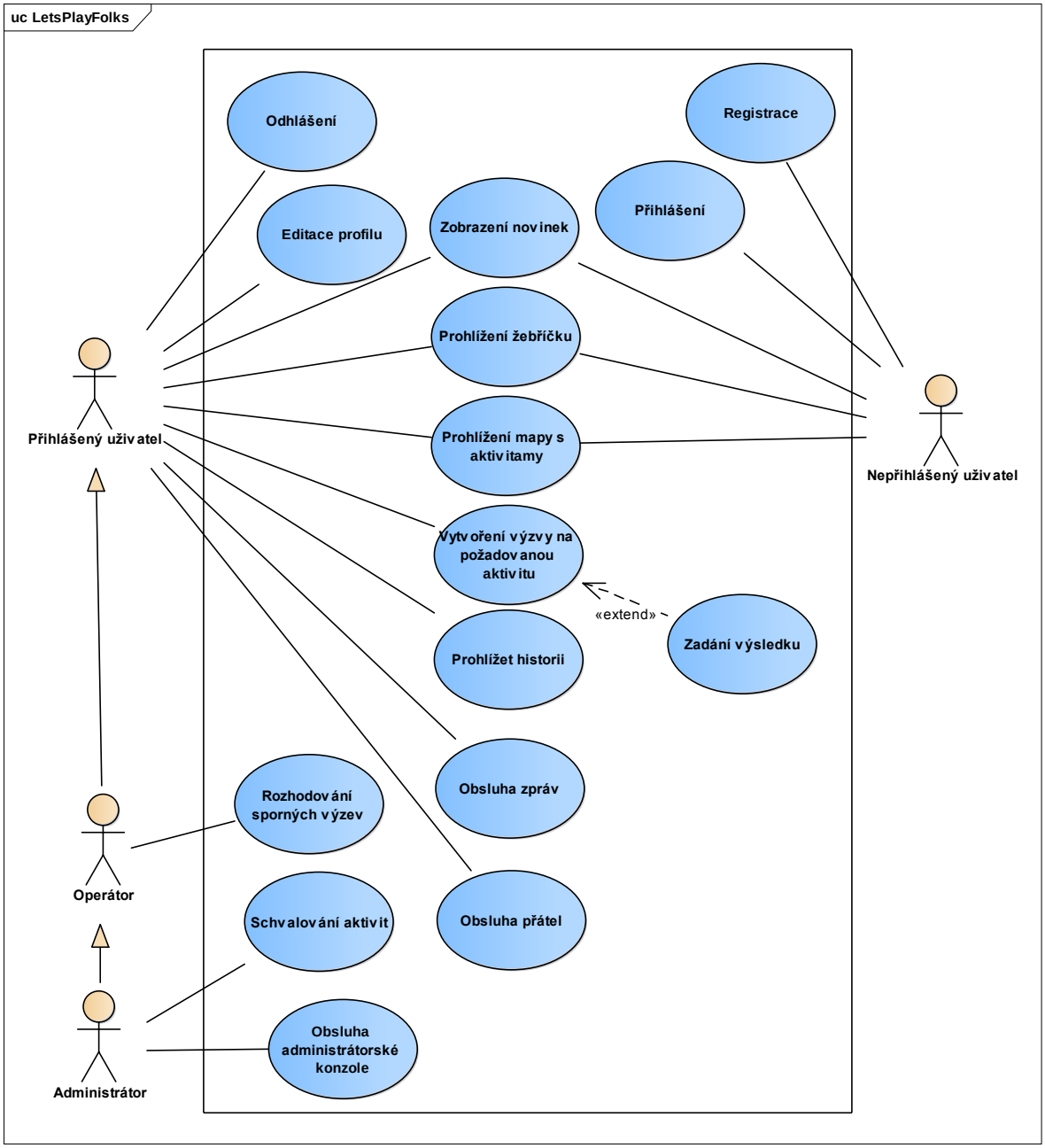
Aplikace by měla běžet na dedikovaném serveru z důvodu spolehlivosti při větším náporu uživatelů. Na tomto serveru by měl být nainstalovaný operační systém Linux libovolné distribuce a požadované technologie potřebné pro běh aplikace, jakými jsou například Java, Maven apod. Z důvodu velkého množství webových prohlížečů, které jsou dnes k dispozici je snad již samozřejmostí zajistit nezávislost aplikace na použitém webovém prohlížeči.

Návrh aplikace by měl dodržovat dnes zavedené standardy a principy. Tyto principy určují vodítka, které nám užitečně pomůžou v jednotlivých fázích návrhu aplikace. Jedním z takových principů může být např. SOLID[15] nebo DRY[16]. Také je požadováno použití responsivního designu, jelikož v dnešní době je běžné přistupovat k internetu z mobilních či jiných zařízeních.

[15] SOLID – Single responsibility, Open/Closed, Liskov subst., Interface segregation, Dependency inversion.
[16] DRY – Don't Repeat Yourself (česky „Neopakuj se").

### 3.1.3 UML use case diagram

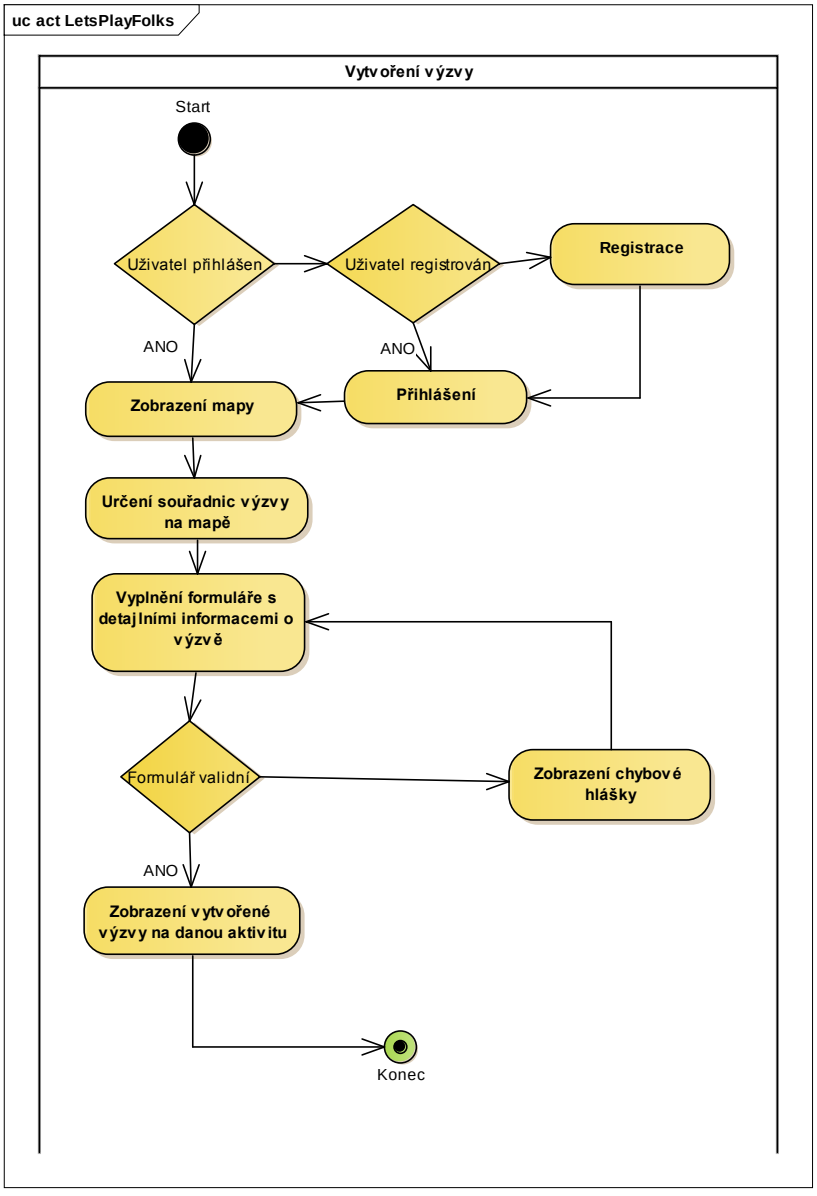
Use case diagram (diagram případů užití) znázorňuje možnosti jednotlivých aktérů (uživatelů) v systému (aplikaci), dle jim přiřazených rolí. Hlavním účelem tohoto diagramu je zachycení chování těchto aktérů, a to pomocí vizuální i textové podoby, která je poté srozumitelná jak zákazníkovi, programátorům, tak koncovým uživatelům. Diagram je vytvořen na základě funkčních požadavků. Use case vytvářené aplikace je znázorněn na obrázku 13.




Obrázek 13 – Use case diagram, zdroj vlastní

### 3.1.4 UML activity diagram

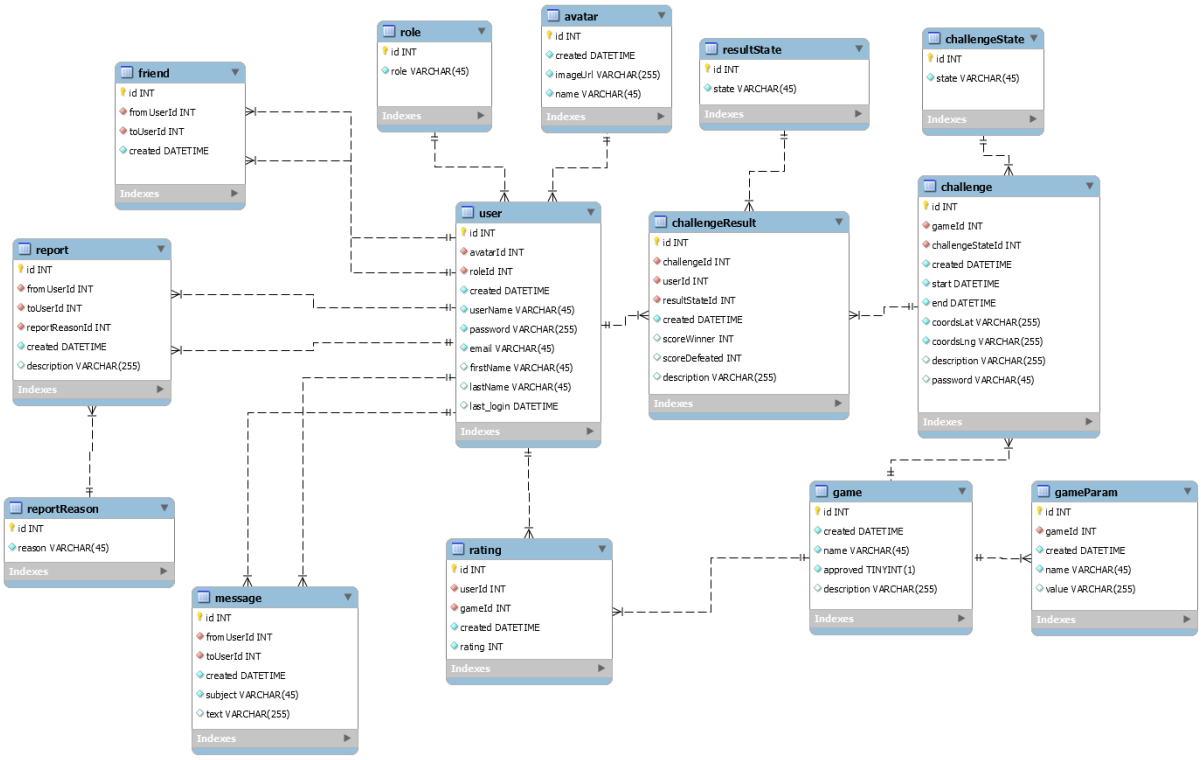
Activity diagram (diagram aktivit) popisuje chování akcí probíhajících v aplikaci. Jeho účelem je znázornit tok řízení aktivit. Oproti předchozímu (use case) diagramu se liší především v tom, že se soustředí přímo na proces a logiku jednotlivých akcí, které uživatelé v systému mohou vykonávat. Lze říci, že pro každý případ užití z předešlého diagramu lze vytvořit příslušný diagram aktivit, popisující konkrétní vykonávání tohoto případu užití. Na následujícím obrázku je pro ukázku vytvořen diagram aktivit pro vytvoření určité výzvy.




Obrázek 14 – UML activity diagram, zdroj vlastní

### 3.1.5 Návrh databáze

Pro uložení dat byla použita databáze H2. Tato databáze bude představena v kapitole popisující technologie a nástroje využité při vývoji. Na následujícím obrázku lze vidět schéma databáze použité ve vytvářené aplikaci. Toto schéma obsahuje 14 tabulek. Účel těchto tabulek se dá ve většině případů bez problému odvodit z jejich názvu. Nejvýznamnějšími tabulkami, které tvoří centrální logiku aplikace, jsou bezpochyby *user*, *challenge* a *challengeResult*. Naproti tomu tabulka *avatar* v aplikaci není aktuálně využita vůbec, nicméně vzhledem k plánovanému budoucímu rozšíření aplikace je tu již zahrnuta.




Obrázek 15 – schéma databáze, zdroj vlastní

Každá tabulka obsahuje umělý primární klíč – ID, který se automaticky navyšuje díky nastavené vlastnosti *auto inkrementace*. Díky tomu není potřeba vytvářet sekvence či jiné způsoby, které se o tuto funkcionalitu starají. Dále nalezneme v každé tabulce atribut *created*, což je časové razítko udávající vytvoření konkrétního záznamu. Hesla jsou z důvodu bezpečnosti v databázi uložena v šifrované podobě pomocí algoritmu *Bcrypt*, tím pádem je jejich lidsky čitelná podoba skryta.

## 3.2 Technologie použité pro vývoj

Nezbytným krokem, ještě před samotnou implementací aplikace, je zvolení příslušných technologií, ve kterých má být výsledná aplikace implementována. Základním stavebním kamenem aplikace je framework Spring, který byl dostatečně představen v předešlých kapitolách a jazyk Java, který není důležité popisovat, protože se tato znalost od čtenáře očekává.

Další využité technologie a nástroje jsou velmi stručně popsány v následujících kapitolách. Pro jejich úplné pochopení je doporučeno navštívit příslušný zdroj nebo odbornou literaturu.

### 3.2.1 HTML

HTML (Hypertext Markup Language) je značkovací jazyk používaný pro kódování webových stránek. Za pomoci tohoto jazyka se navrhuje základní struktura stránky a rozložení jednotlivých prvků. HTML nalezneme na každé webové stránce a tím pádem je to základní dovednost, kterou musí každý webový vývojář ovládat.

```
<!DOCTYPE html>
<html lang="cz">
<head>
    <meta charset="UTF-8">
    <title>Moje první stránka</title>
</head>
<body>
    <h1>Ahoj světe!</h1>
    <p>HTML značky mohou dát <b><i>různý</i></b>
    <u>vzhled a formátování</u> obsahu webové stránky.</p>
</body>
</html>
```

Obrázek 16 – ukázka HTML kódu, zdroj vlastní

Základem tohoto jazyka jsou značky, typicky párové. Tyto značky obklopují text podle jeho typu. Účelem webových prohlížečů je pak tyto dokumenty číst a podle těchto značek je i zobrazovat. HTML dokument je také výstupem při použití veškerých serverových technologií jako je PHP nebo v případě vytvářené aplikace JSP. [21]

### 3.2.2 CSS

Kaskádové styly neboli CSS (Cascading Style Sheets) jsou stylovým jazykem vytvořeným pro grafickou podobu a formátování dokumentů psaných značkovacími jazyky, jakými jsou například (X)HTML či XML. Důvodem vytvoření tohoto jazyka je oddělení struktury dokumentu od jeho samotného obsahu. Využití těchto stylů je alternativou k formátování stránek pomoci HTML tagů. Tyto styly jsou dnes velice běžné a jejich použití najdeme téměř na každé internetové stránce.

Syntaxe jazyka je tvořena z tzv. selektoru a deklaračního bloku. Selektor označuje, pro jaký HTML element daná deklarovaná pravidla platí. Deklarační blok poté pro tyto elementy definuje konkrétní hodnoty, jakými mohou například být barva pozadí, velikost textu, odsazení a další nepřeberné množství možností. [22]

```
.offline-doc .page-header:after {
  background-color: rgba(0, 0, 0, 0.5);
  content: "";
  display: block;
  height: 100%;
  left: 0;
  position: absolute;
  top: 0;
  width: 100%;
  z-index: 2;
}
```

Obrázek 17 – ukázka CSS stylů, zdroj vlastní

Vzhledem k tomu, že v dnešní době již nejsme při vývoji aplikací odkázáni pouze sami na sebe, ale máme k dispozici široké spektrum podpůrných nástrojů, tak tomu není jinak ani v případě této technologie. Takovým nejznámějším nástrojem je Bootstrap od společnosti Twitter. Jedná se o volně dostupnou knihovnu pro tvorbu webových aplikací. Nalezneme zde obrovské množství předpřipravených šablon, tlačítek, formulářů či jiných prvků vyskytujících se na stránce. Díky tomu je velice jednoduché vytvořit vzhledově přívětivou stránku za opravdu krátkou dobu. Samotné použití Bootstrapu tedy spočívá v přiřazení předpřipravených stylů konkrétním prvkům v dokumentu. Jako nevýhodu by mohl někdo zmínit to, že vzhledem k popularitě tohoto frameworku může docházet k podobě vzhledu jednotlivých webových aplikací, které Bootstrap využívají. Otázkou však zůstává, jestli obdobný vzhled webů napříč internetem není spíše výhodou. Dalšími populárními CSS frameworky jsou například Foundation, Bulma nebo Materialize od společnosti Google. [22]

### 3.2.3 Javascript

Javascript je skriptovací jazyk patřící mezi objektově orientované programovací jazyky. Hned na začátek je dobře upozornit na běžný omyl, a to spojování Javascriptu s programovacím jazykem Java. Tyto dva jazyky toho kromě názvu nemají moc společného.

V dnešní době nalezneme využití Javascriptu téměř na všech webových dynamických stránkách. Jeho účelem je ovládání interaktivních prvků, kterými stránka disponuje, jako jsou např. různá tlačítka, formuláře, animace, vyskakovací okna apod. Ke spuštění Javascriptu dochází na straně klienta, a nikoliv na straně serveru, jak tomu je například v případě jazyka PHP či JSP. To je velice zásadní z důvodu, že si klient skripty v tomto jazyce může libovolně upravovat či tento skriptovací jazyk v prohlížeči úplně zakázat. S trochou nadsázky lze tvrdit, že se z těchto důvodů na tento jazyk nelze s jistotou spoléhat.

Velice populárním rozšířením Javascriptu je knihovna jQuery. Tato knihovna klade důraz na rychlost a jednoduchost tvorby dynamických části webu. Obsahuje nespočet rozmanitých způsobů, jak jednotlivé prvky na stránce obsluhovat či upravovat. Tuto knihovnu lze v bohaté míře nalézt na moderních stránkách ve formě různých animací či pohyblivých prvků vyskytujících se na webu. [23]

### 3.2.4 Google Maps API

API (Application Programming Interface) je označení pro rozhraní, které programátor využívá pro komunikaci se softwarem. Jedná se o balík funkcí, protokolů, tříd či knihoven, které může vývojář určitým způsobem využívat či ovládat.

Google Maps (Mapy Google) je technologie a webová mapová aplikace od společnosti Google, na základě které je založeno mnoho mapových služeb po celém světě, včetně nejznámějších Map Google[17]. Tyto mapy nabízejí satelitní snímky, ve velice vysokém rozlišení, téměř celého obydleného světa. Díky tomu si některé vlády stěžují na potenciální hrozbu použití těchto map ke kriminálním účelům, na základě které byla některá místa účelně rozmazána. Kromě satelitních snímků lze na mapách od Googlu přepínat mezi obrovským množstvím zobrazení, z nichž určitě stojí za zmínku Google Street View[18]. Kromě jiného se zde dají plánovat trasy různými způsoby dopravy, na základě kterých dostaneme jejich časovou

[17] https://www.google.com/maps
[18] Google Street View – jedná se o virtuální vyobrazení okolí, skládajícího se z panoramatických snímků.

náročnost, převýšení a další zajímavé informace o naplánované cestě. Další velice používanou funkcionalitou této služby je vyhledávání zařízení v nejbližším okolí, jako jsou restaurace, nemocnice, bankomaty apod.

Jak by mělo být z předešlých odstavců patrné, Google Maps API je tedy rozhraní, které dovoluje programátorům přistupovat k službě Google Maps a využívat všechny její možnosti. Pro nekomerční využití je toto API zdarma, nicméně pro komerční využití je to již zpoplatněno. Pro používání tohoto API je nezbytné vlastnit přístupový klíč, který lze obstarat na oficiálních stránkách[19]. Pro komunikaci a obsluhu API od Google Maps se používá jazyku Javascript popsaného v předešlé kapitole. [24], [25]

### 3.2.5 Maven

Nástroj Maven od společnosti Apache je *buildovací* nástroj sloužící pro správu, automatizaci a řízení *buildů*[20] aplikace. Navzdory tomu, že lze tento nástroj použít pro různé programovací jazyky, je nejvíce podporován především jazyk Java. [26]

Maven je navržen pro zjednodušení práce při sestavování aplikace a nejčastější agentu s tím spojenou. Tím může být například:

- řízení závislostí
- tvorba dokumentace
- release
- tvorba reportů
- distribuce

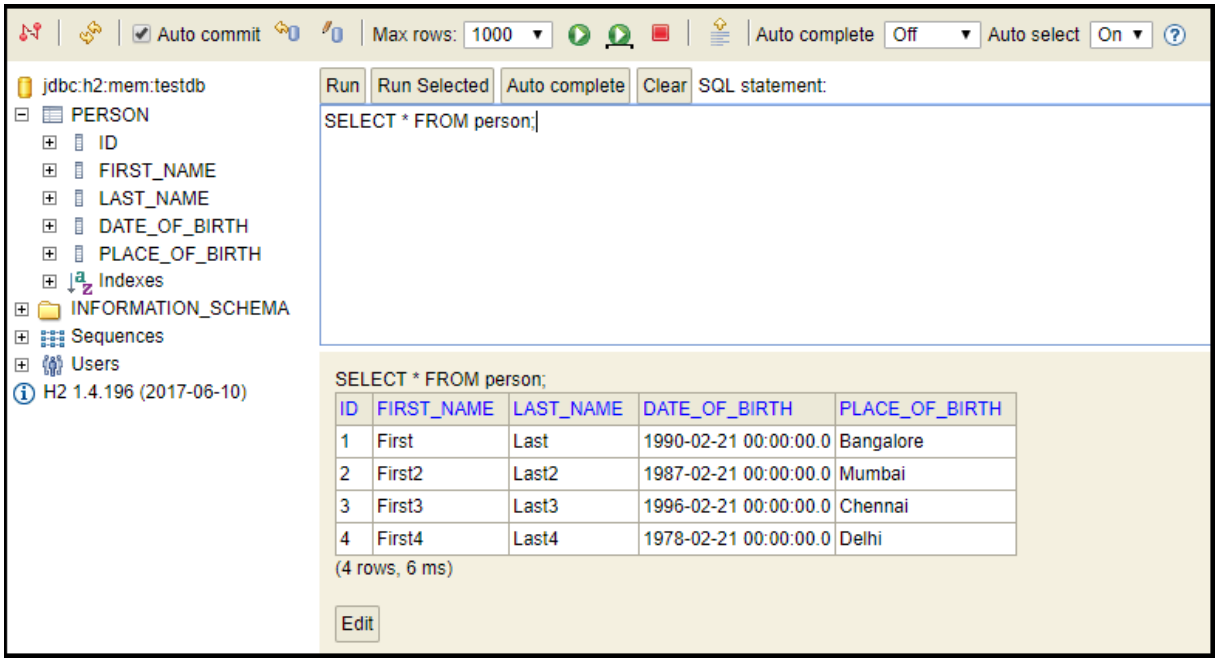
Výhodou Mavenu je podpora ve všech nejrozšířenějších vývojářských prostředích, kterými jsou NetBeans, Eclipse nebo IntelliJ IDEA.

Tento nástroj využívá konvence nad konfigurací, což je přístup výchozího nastavení, které není třeba dále konfigurovat. Nicméně pokud je to nezbytné, lze bez problému výchozí hodnoty změnit. Příkladem této konvence může být např. standardní adresářová struktura, pojmenování jednotlivých souborů apod. [26]

[19] https://cloud.google.com/maps-platform/
[20] Build – převedení zdrojového kódu do vhodné podoby pro zpracování počítačem.

### 3.2.6 H2 Database

H2 je jednoduchá, odlehčená, volně šiřitelná databáze, určená pro aplikace běžící na Javě. Největším plusem této databáze je možnost její konfigurace k tomu, aby běžela pouze v paměti počítače, což také samozřejmě znamená, že data nejsou reálně ukládána na disk. Z toho lze odvodit, že je vhodná především pro vývojové či testovací účely, a ne pro reálně použití na produkčním prostředí. Nicméně právě při vývoji je její jednoduchost, rychlost, a především přenositelnost mezi stroji velkým plusem, protože k jejímu běhu není potřeba nic instalovat ani konfigurovat, jako je tomu například u databáze Oracle, MySQL a dalších. Závislosti potřebné k jejímu běhu se stáhnou automaticky po sestavení aplikace pomoci Maven *buildu* a poté je databáze připravena k běhu. H2 databáze navíc již obsahuje vestavěnou konzoly, pomocí které ji lze bez problému obsluhovat a tím pádem není potřeba instalovat další programy, jako jsou např. pro výše zmíněné databáze Oracle SQL developer či MySQLWorkbench. [27]



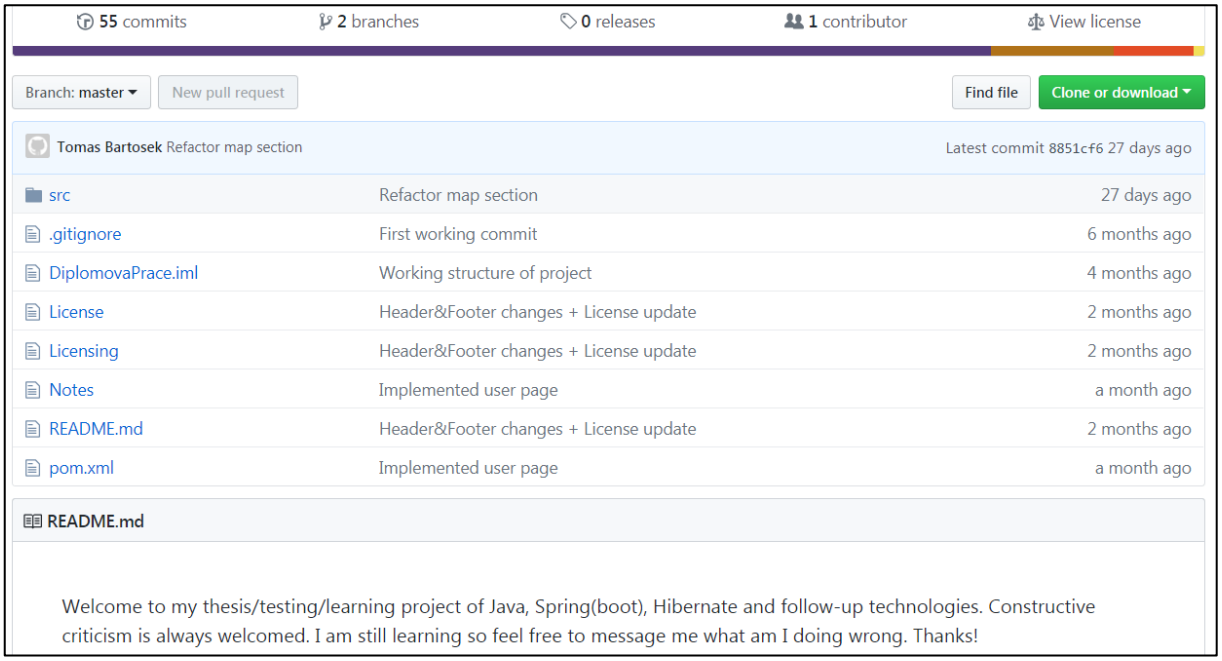

Obrázek 18 – H2 konzole, zdroj vlastní

Největší výhodou při vývoji aplikace, kterou nám tato databáze nabízí, je již výše zmíněné, ukládání dat *in-memory* (v paměti). Jelikož při testování aplikace dochází téměř neustále k nekonzistentním stavům v databázi, což má samozřejmě za následek nefunkční aplikaci, je velice snadné tento problém řešit. Místo složitého hledání a opravování konkrétních dat v databázi stačí aplikaci pouze restartovat a je opět v provozu-schopném stavu a chybu lze jednoduše znovu nasimulovat a otestovat. [27]

### 3.2.7 Git

Nástroj Git je distribuovaný systém správy verzí. Díky němu je možné jednoduše vytvářenou aplikaci publikovat ve verzích a dává nám jednoduchý způsob přístupu k celkové historii projektu. V historii je možné se libovolně přesouvat a lze zde přehledně zobrazit kdo a kdy jakou funkcionalitu či část kódu upravil a co přesně se změnilo. Díky větvení je možné vytvářet novou funkcionalitu, aniž by byl zbytek projektu nějakým způsobem ovlivněn. V případě že tato funkcionalita nebude podle představ, tak se celá tato větev smaže bez nějakého dopadu na projekt.

S nástrojem Git se pojí i další nástroj – GitHub, což je cloud platforma vytvořena okolo Gitu. Jednoduše řečeno lze říci, že nástroj Git ukládá veškeré změny, historii apod. pouze lokálně na počítači uživatele. Nicméně v případě, že pracujeme na projektu s více lidmi, je potřeba tyto změny promítnout i do jejich projektu. Právě k tomuto slouží GitHub. Jinými slovy to tedy je online služba pro sdílení projektu mezi více vývojáři. [28]




Obrázek 19 – zdrojové kódy projektu na GitHubu, zdroj vlastní

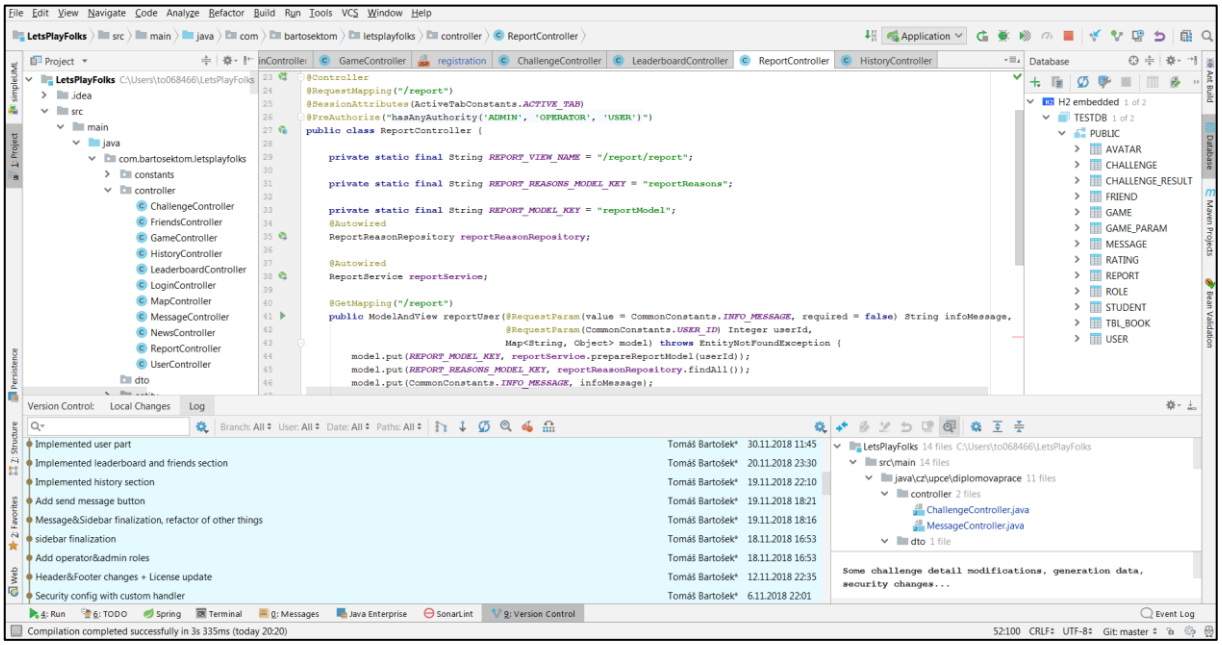
Zdrojové kódy vytvořené aplikace jsou veřejně přístupné v repositáři na GitHubu pod názvem *LetsPlayFolks*[21].

[21] https://github.com/TomasBartosek/LetsPlayFolks

### 3.2.8 IntelliJ IDEA

Pro vývoj aplikace bylo použito vývojové prostředí IntelliJ IDEA od společnosti JetBrains. Toto vývojové prostředí je komerční a jedná se o jediný placený produkt, který byl při vývoji použit. S tímto nástrojem nesmírně roste produktivita a ulehčuje práci téměř ve všech ohledech. Pro programátora, který vyvíjí v jazyku Java, je toto IDE[22] jednoznačně tím nejlepším, co je v dnešní době k dispozici.

Nesmírným plusem tohoto vývojového prostředí, oproti ostatním, je indexace projektu. Díky tomu je možné vyhledávat nad celým projektem velice rychle. Toto není ovšem na první pohled zřejmé na vytvářené aplikaci, nicméně v projektech o stovkách či tisících souborech je to nedocenitelná výhoda. [29]



Obrázek 20 – vývojové prostředí IntelliJ IDEA, zdroj vlastní

Další předností je velice dobrá podpora s verzovacími systémy, konkrétně při vývoji této aplikace s VCS Git, a to v takové míře, že obsahuje veškeré funkce potřebné pro obsluhu v grafickém režimu a není tedy potřeba obsluhovat Git přes konzoly či instalovat další nástroj. V neposlední řadě má Idea tak propracovanou podporu pro práci s databází, že se tím téměř dají nahradit ostatní takové nástroje, jakými jsou např. SQL Developer nebo MySQL Workbench. [29]

[22] IDE (Integrated Development Env.) – vývojové prostředí je software usnadňující práci programátorů.

## 3.3 Implementace vytvořené aplikace

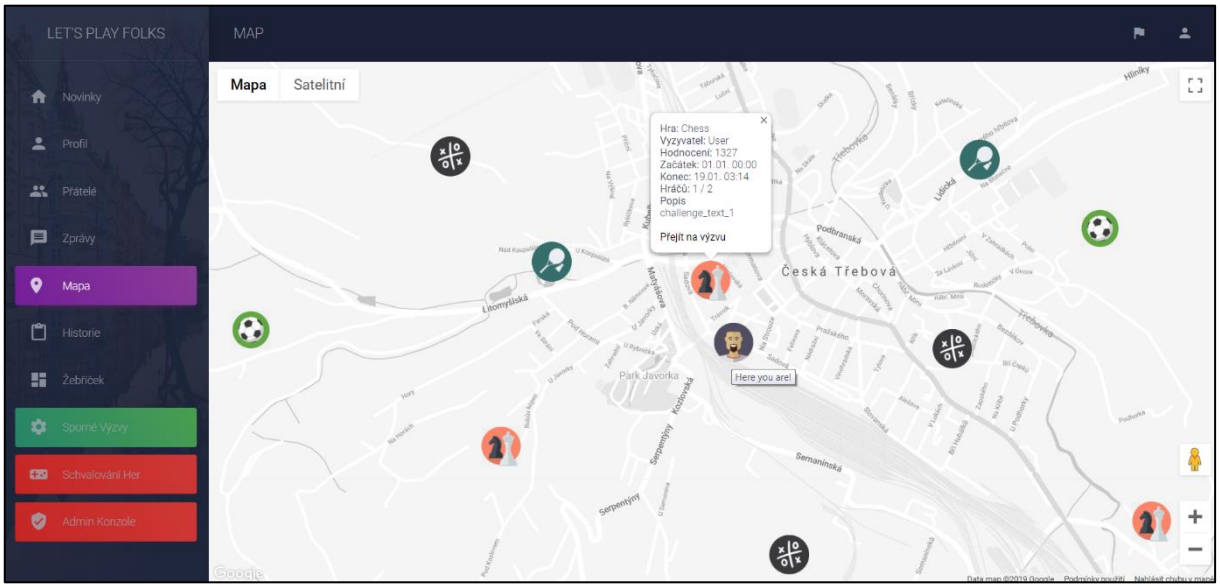
V této kapitole je stručně objasněn návrh a implementace vytvářené aplikace. Je zde popsána grafická podoba a implementace dílčích částí aplikace dle architektonického vzoru MVC. Mimo jiné je věnována kapitola i zabezpečení a internacionalizaci aplikace.

### 3.3.1 Grafická podoba aplikace

Grafická podoba webu není sice pro požadovanou funkcionalitu aplikace nezbytná, avšak v posledním desetiletí se na tuto vlastnost začíná čím dál více hledět a uživatelé internetu jsou již zvyklí na nějaké standardy.

Úprava aplikace je realizována především kaskádovými styly, frameworkem Bootstrap, jazykem JavaScript a jeho knihovnou jQuery. Zmíněné technologie byly již popsány v dřívějších kapitolách. Příklad toho, jaký vzhled má aplikace bez grafické úpravy, je znázorněn v příloze C. Celková funkcionalita aplikace zůstává stále stejná, nicméně uživatel by bezpochyby nebyl s takovýmto vzhledem spokojený.

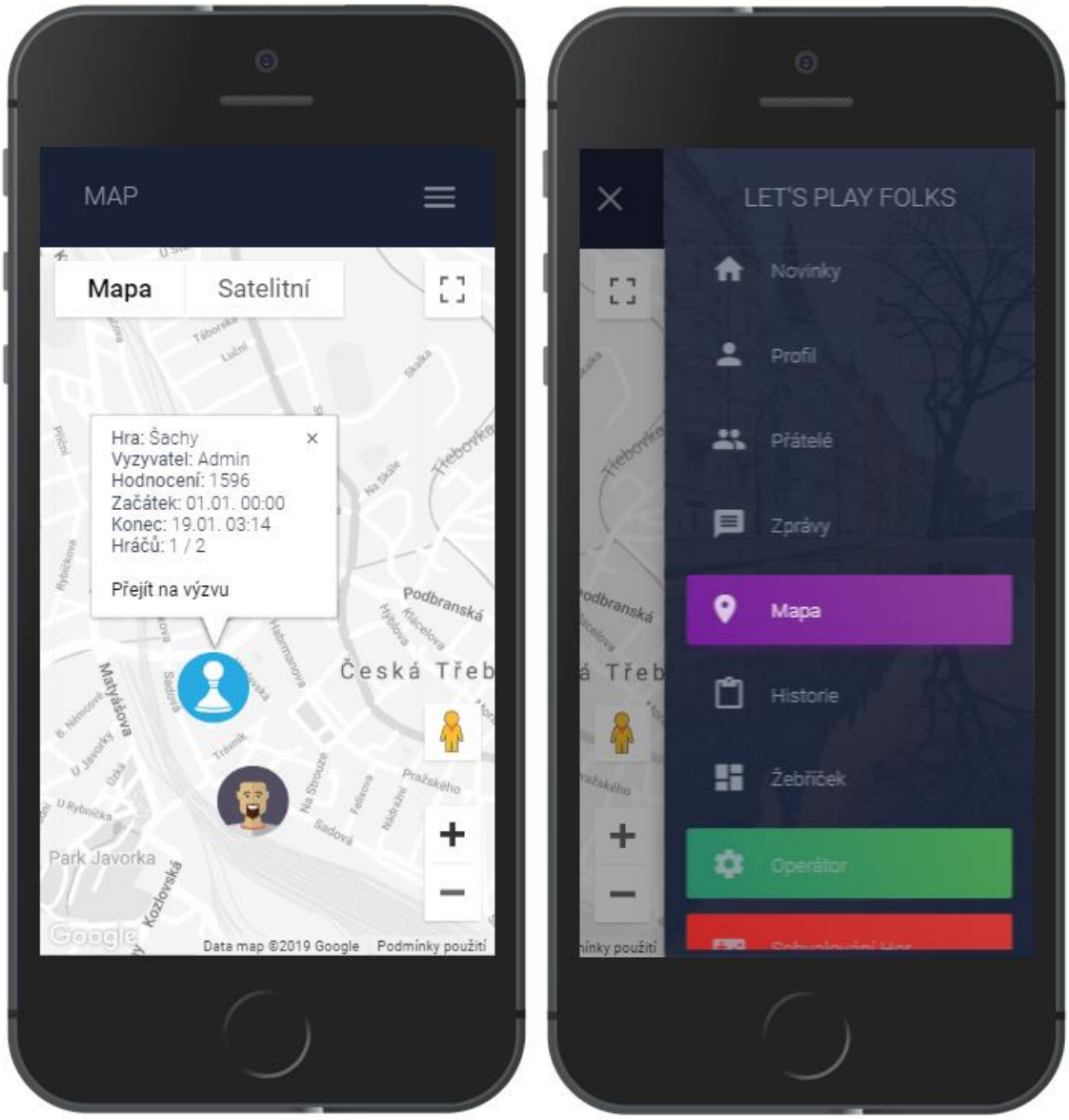
Finální grafickou podobu aplikace lze názorně vidět na následujícím obrázku.




Obrázek 21 – grafická podoba aplikace, zdroj vlastní

Veškeré stránky, vyskytující se v aplikaci, mají obdobnou podobu a totožné rozestavení dílčích prvků. Tato myšlenka se v dnešní době dodržuje v drtivé většině webových aplikací a díky tomu stránky vypadají jednotně, jsou přehledné a snadno udržovatelné. Díky tomu se také uživatel velice snadno orientuje. Jednotlivé stránky jsou tedy rozdílné jenom patřičným obsahem.

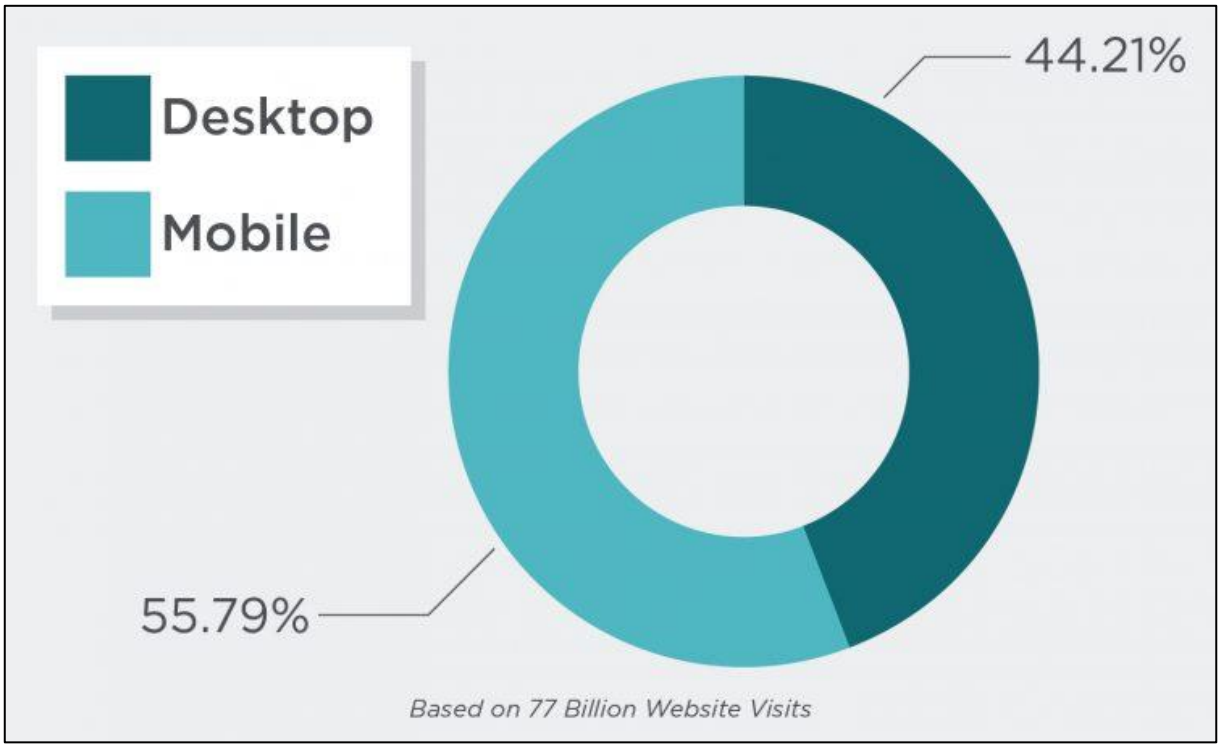
Jak bylo znázorněno na předchozím obrázku, aplikace využívá dvousloupcový layout. Na levé straně se vyskytuje vertikální menu, sloužící pro pohybování mezi jednotlivými částmi aplikace. V pravém horním rohu je horizontální menu, které slouží pro přihlášení či odhlášení uživatele a poskytuje možnost změny jazyka zobrazujícího se na jednotlivých stránkách. Mimo jiné aplikace obsahuje také samozřejmě *header* a *footer*.




Obrázek 22 – zobrazení aplikace na mobilním zařízení, zdroj vlastní

Na obrázku 23 lze vidět zobrazení aplikace na telefonním zařízení (konkrétně se jedná o mobilní zařízení iPhone 5S). V dnešní době, kdy valná většina uživatelů vlastní telefon s připojením k internetu, je podpora zobrazení webových aplikací na mobilních či jiných zařízeních doslova nezbytná.

Na následujícím grafu je znázorněn počet přístupů na webové stránky podle zařízení. Jak je z grafu patrné, v dnešní době již začínají v tomto směru vítězit mobilní zařízení na úkor klasických desktopových počítačů.




Graf 1 – Porovnání přístupu na web podle zařízení, zdroj [30]

Vlastnosti webových aplikací, díky které jsou zobrazitelné i na mobilních zařízení, se říká responsivní web design. Tento pojem se poprvé objevil v roce 2010 a to díky americkému programátorovi jménem Ethan Marcotte. Responsivní design je tedy způsob stylování HTML dokumentu, který zajistí, že zobrazení aplikace bude optimalizováno pro veškeré druhy různých zařízení, ať už se jedná o mobilní telefony, tablety nebo samozřejmě klasické počítače. [31]

V případě, že uživatel aplikace bude potřebovat nějakou konkrétní stránku vytisknout, je rozumné mít pro tuto situaci nastavené jiné styly a formátování, než se uživateli opravdu zobrazují v samotném dokumentu. Uživatel aplikace nechce tisknout patičku, menu, hlavičku apod. Jde mu především o obsah samotné stránky, o samotný text bez těchto „zbytečných“ grafických součástí. Jak takové formátování vypadá, je k nahlédnutí v příloze B.

### 3.3.2 Datová vrstva

Datová vrstva aplikace neboli podle MVC architektury model, je zajištěn pomocí ORM[23] frameworku Hibernate. Pro jednotlivé tabulky z databáze jsou vytvořeny třídy *entit*. Tyto třídy reprezentují data uložené v těchto tabulkách a informace kterých se jich týkají (datový typ, primární/cizí klíč, velikost, relace atd.). Tyto informace jsou do těchto tříd vkládány pomocí anotací. Pro jejich plný výčet je doporučeno navštívit oficiální dokumentaci.

```
@Entity
public class Challenge {
    private Integer id;
    private Timestamp created;
    // Another class attributes

    @Id
    @Column(name = "id", nullable = false)
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    public Integer getId() { return id; }

    public void setId(Integer id) { this.id = id; }

    @Basic
    @Column(name = "created", nullable = false)
    @CreationTimestamp
    public Timestamp getCreated() { return created; }

    public void setCreated(Timestamp created) { this.created = created; }
    // Another getters/setters
```

Zdrojový kód 9 – entity třída, zdroj vlastní

Pro přístup či modifikaci dat, které tyto entity reprezentují, slouží takzvané repositáře. Repositář je třída anotována anotací *@Repository*.

```
@Repository
public interface ChallengeRepository extends CrudRepository<Challenge, Integer> {
    List<Challenge> findByGameByGameId(Game game);
    /// Another repository methods
}
```

Zdrojový kód 10 – rozhraní CrudRepository, zdroj vlastní

[23] ORM – Objektově relační mapování.

Účelem takovýchto tříd je, jak již bylo řečeno, přístup a modifikace dat v databázi. Toto je díky Springu velice jednoduché a je k tomu využito rozhraní *CrudRepository*.

Na předchozím příkladu je názorně vidět jednoduchost práce s tímto rozhraním. Toto rozhraní má povinné dva parametry. První parametr je třída entity, reprezentující požadovanou databázovou tabulku. Druhý parametr je datový typ primárního klíče této tabulky, resp. entity. Poté jediné, co pro vyhledání záznamu v tabulce *Challenge* je potřeba, je správný název metody a její implementaci si již zařídí samo toto rozhraní. Předešlí příklad tedy konkrétně slouží pro: „nalezení všech záznamů v tabulce *Challenge*, ve které se vyhledává podle entity *Game*, resp. podle jejího primárního klíče".

Stejně jednoduché je pomocí tohoto rozhraní i vytvoření nového záznamu a uložení ho do databáze.

```
Challenge challenge = new Challenge();
challenge.setStart(Timestamp.valueOf(challengeModel.getStart()));
challenge.setEnd(Timestamp.valueOf(challengeModel.getEnd()));
challenge.setCoordsLat(challengeModel.getLatCoords());
challenge.setCoordsLng(challengeModel.getLngCoords());
// Another setters for challenge attributes
challengeRepository.save(challenge);
```

Zdrojový kód 11 – vytvoření nové výzvy, zdroj vlastní

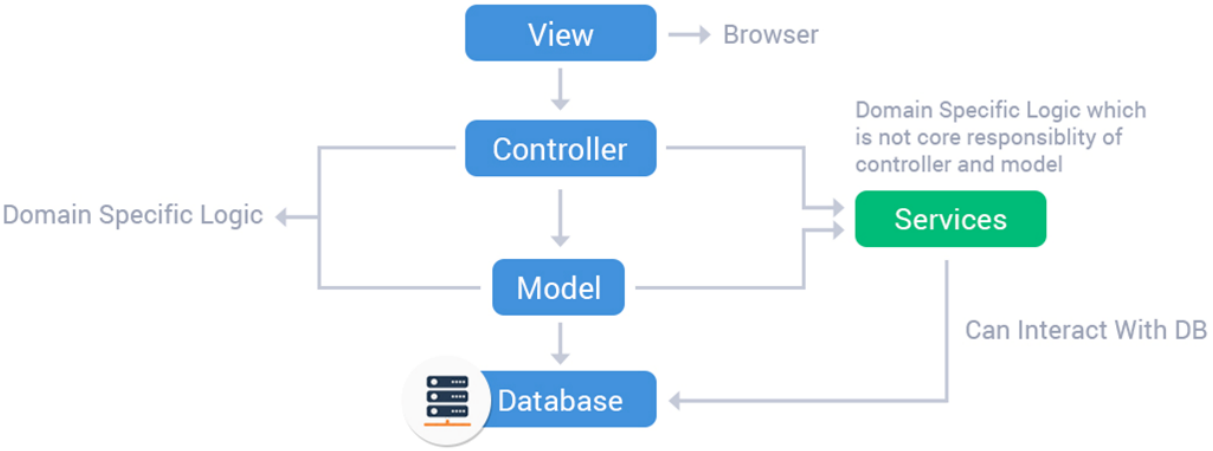
Na prvním řádku je vytvořena nová entita výzvy pomocí výchozího bezparametrického konstruktoru. Na dalších řádcích jsou této entitě nastaveny jednotlivé atributy (v tomto případě pomocí hodnot získaných z formuláře). Na posledním řádku je konečně entita společně s nastavenými atributy uložena do databáze pomocí metody *save*. Do této chvíle byl objekt *challenge* stále pouze třídou (entitou), která neměla na změnu databáze žádný vliv. Analogickým způsobem probíhá i úprava (editace) záznamů v databázi s tím rozdílem, že na prvním řádku se nevytváří entita nová, ale nalezne se konkrétní záznam, který je potřeba editovat. Stejně jako při vytváření pak stačí pouze zavolat metodu save, která nám změnu promítne i do databáze.

Posledním příkladem CRUD[24] operace, která nebyla zmíněna je odstraňování. Stejně jako ostatní operace, díky rozhraní *CrudRepository* je tato funkcionalita velice jednoduchá. Záznam stačí pouze vyhledat a poté na něm zavolat funkci *delete*.

[24] CRUD – označení pro čtyři základní operace používané v databázi (Create, Read, Update, Delete).

### 3.3.3 Vrstva business logiky

Na této vrstvě se nachází hlavní logika aplikace. Architektura MVC je běžně rozšířena ještě o jednu vrstvu, která se označuje jako *servisní*, viz znázornění na následujícím obrázku.




Obrázek 23 – MVC architektura se servisní vrstvou [32]

Důvodem tohoto rozšíření je dodržování správných programátorských principů, v tomto případě především „principu jedné odpovědnosti“. Ve většině případů totiž aplikace obsahuje spoustu skryté logiky, která není na první pohled vidět. Ale do jaké vrstvy tedy logiku implementovat? *Controller* by měl totiž v nejlepším možném případě obsahovat pouze směrování požadavků a minimum samotné logiky, stejně tak model by měl sloužit především pro obsluhu databáze. Odpovědí je tedy již zmíněná servisní vrstva. Třídy této vrstvy jsou označeni anotací *@Service* a nalezneme v nich implementaci samotné funkcionality aplikace.

```
@Service
public class ChallengeServiceImpl implements ChallengeService {

    @Autowired
    UserRepository userRepository;
    // Another autowired beans

    @Override
    public boolean isUserAlreadyInChallenge(Challenge challenge) {
        User user = userRepository.findById(sessionManager.getUserId()).orElse( other: null);
        return user != null && userRepositoryCustom.findAllChallengeUsersByChallenge(challenge).contains(user);
    }

    @Override
    public boolean isChallengeFinished(Challenge challenge) {
        return ChallengeStateConstants.FINISHED.equals(challenge.getChallengeStateByChallengeStateId().getState());
    }
    // Another business logic
```

Zdrojový kód 12 – ukázka servisní třídy, zdroj vlastní

Když už je logika implementována na servisní vrstvě, nic nebrání *controlleru* sloužit pouze ke směrování. Třídy *controlleru* jsou označeny anotací *@Controller* a v kódu se nacházejí v balíčku *controller*.

```
@Controller
@RequestMapping("/challenge")
@SessionAttributes(ActiveTabConstants.ACTIVE_TAB)
public class ChallengeController {

    private static final String CHALLENGE_ID_REQUEST_PARAM = "challengeId";
    // Another request params
    private static final String CHALLENGE_MODEL_ATTRIBUTE = "challengeModel";
    // Another model attributes
    private static final String CHALLENGE_MODEL_KEY = "challenge";
    // Another model keys
    private static final String CHALLENGE_DETAIL_VIEW_NAME = "challenge/detail";
    // Another view names

    @Autowired
    ChallengeService challengeService;
    // Another autowired beans

    // Controller calls service. Service returns an object (be it a DTO, domain model or something else)
    @GetMapping("/detail")
    public ModelAndView challengeDetail(@RequestParam(CHALLENGE_ID_REQUEST_PARAM) int challengeId,
                                        @RequestParam(value = INFO_MESSAGE, required = false) String infoMessage,
                                        Map<String, Object> model) throws EntityNotFoundException {
        Challenge challenge = challengeRepository.findById(challengeId).orElseThrow(EntityNotFoundException::new);
        ChallengeDetailModel challengeDetailModel = challengeService.prepareChallengeDetailDto(challenge);
        boolean isUserAlreadyInChallenge = challengeService.isUserAlreadyInChallenge(challenge);
        boolean isChallengeFinished = challengeService.isChallengeFinished(challenge);
        boolean canUserEnterResult = challengeService.canUserEnterResult(challenge);

        model.put(CHALLENGE_DETAIL_MODEL_KEY, challengeDetailModel);
        model.put(CHALLENGE_MODEL_KEY, challenge);
        model.put(IS_USER_ALREADY_IN_CHALLENGE_MODEL_KEY, isUserAlreadyInChallenge);
        model.put(IS_CHALLENGE_FINISHED_MODEL_KEY, isChallengeFinished);
        model.put(CAN_USER_ENTER_RESULT, canUserEnterResult);
        model.put(CommonConstants.INFO_MESSAGE, infoMessage);

        return new ModelAndView(CHALLENGE_DETAIL_VIEW_NAME, model);
    }
```

Zdrojový kód 13 – ukázka controlleru, zdroj vlastní

K tomu, aby aplikace věděla, na který *controller* má určitý URL požadavek poslat, slouží anotace *@RequestMapping*, která byla již popsána v kapitole 2.4.3.

Předchozí ukázka kódu se tedy v případě URL adresy **/challenge/detail* zachová tak, že aplikace nejprve nalezne příslušný *controller*, který obslouží požadavek */challenge* a poté v tomto controlleru hledá metodu */detail*. Poté co je nalezena se provede její logika, resp. se v ní pouze provolají metody na servisní vrstvě a poté dojde k přesměrování na prezenční vrstvu s naplněným modelem. Anotace *@GetMapping* je pouze zkrácením pro již zmíněnou anotaci *@RequestMapping* a jejím účelem je upřesnění jaký http požadavek obsluhuje (GET). Pro POST je alternativou *@PostMapping*. Anotace *@RequestParam* slouží k definování parametrů, které se očekávají v *requestu*. Parametry takto definované jsou povinné a v případě, že nějaký parametr má být dobrovolný, je nutné ho označit jako *required = false*.

### 3.3.4 Prezenční vrstva

Prezenční vrstva neboli podle MVC *view*, slouží k zobrazování výstupních dokumentů uživateli. Tato vrstva je v aplikaci implementována pomocí JSP (Java Server Pages), což je nadstavba Java servletů. Účelem JSP je zjednodušení zápisu dynamických částí kódu do HTML souboru a tím vytvoření dynamických stránek. JSP je v aplikaci možné nalézt v balíčku *view*.

```
<table class="table">
    <thead class="text-primary">
    <tr>
        <th>ID</th>
        <th>Game name</th>
        <th>Created</th>
        <th>Reason</th>
    </tr>
    </thead>
    <tbody>
    <c:forEach items="${questionableChallenges}" var="questionableChallenge">
        <tr>
            <td> ${questionableChallenge.id} </td>
            <td> ${questionableChallenge.gameName} </td>
            <td>
                <fmt:parseDate value="${questionableChallenge.created}"
                               pattern="yyyy-M-d'T'HH:mm"
                               var="parsedDateTime" type="both"/>
                <fmt:formatDate pattern="d.M.yyyy HH:mm" value="${parsedDateTime}"/>
            </td>
            <td>
                <c:set var="reason" value="${questionableChallenge.reason}"/>
                    ${reason}
            </td>
            <td>
                <c:url value="../detail.jsp" var="challengeDetailUrl">
                    <spring:param name="challengeId" value="${questionableChallenge.id}"/>
                </c:url>
                <a href="${challengeDetailUrl}">
                    Detail výzvy
                </a>
            </td>
        </tr>
    </c:forEach>
    </tbody>
</table>
```

Zdrojový kód 14 – JSP stránka, zdroj vlastní

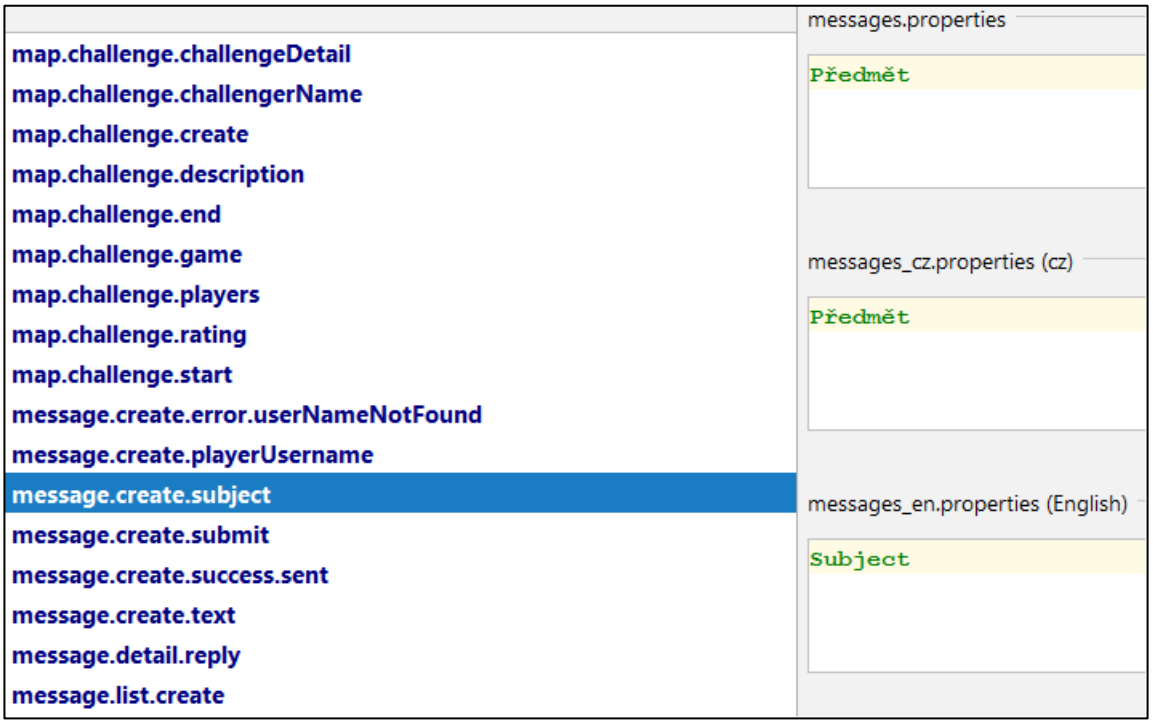
Na předešlé ukázce stránky lze vidět, že valná část kódu je napsána v obyčejném HTML a pouze pro dynamickou část je použit jazyk JSP, resp. knihovna JSTL[25], resp. EL[26].

[25] JSTL (JavaServer Pages Standard Tag Library) – knihovna JSP tagů obsahujících běžnou funkcionalitu.
[26] EL (Expression Language) – jazyk umožňující snadný přístup k uloženým datům jako jsou beany či session.

### 3.3.5 Internacionalizace

Vytvářená aplikace obsahuje překlady textů do dvou jazyků – českého a anglického. Jako výchozí jazyk je nastavena čeština, nicméně je uživatelům dána možnost mezi nabízenými jazyky libovolně přepínat.

K internacionalizaci aplikace je využito tzv. *properties* souborů. Tyto soubory představují slovník do určitého jazyka. Ukázku takového souboru lze vidět na následujícím obrázku.




Zdrojový kód 15 – resource bundle, zdroj vlastní

Texty v ukázce JSP stránky v minulé kapitole jsou do ní napřímo nakódovány. S takovým přístupem stránka samozřejmě vícejazyčná nebude a je tedy nutné tyto texty uložit do určitých proměnných. Názvy těchto proměnných můžeme na předešlé ukázce vidět v levém sloupci. V pravém sloupci lze poté nastavit těmto proměnným jejich překlad do požadovaných jazyků. V samotném JSP se pak tato proměnná vloží do tagu *spring:message* a její hodnota se podle nastavené lokalizace přeloží na požadovaný text, viz následující ukázka.

```
<label for="subject">
    <spring:message code="message.create.subject"/>
</label>
```

Zdrojový kód 16 – spring:message tag, zdroj vlastní

### 3.3.6 Zabezpečení

K zabezpečení aplikace je využito Springovské knihovny – *Spring security*. Tato knihovna poskytuje jednoduchý a účinný způsob autentizace a autorizace aplikace. Jako ostatní části Springu je potřeba funkcionalitu tohoto zabezpečení nakonfigurovat, abychom ji mohli poté využívat. Tato konfigurace může být jak pomocí XML, tak pomocí Java souborů. Ve vytvářené aplikaci je dán přednost konfiguraci v Javě a lze ji najít v souboru *SecurityConfig.java*.

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private MyUserDetailsService myAppUserDetailsService;
    // Another autowired beans

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http
                .authorizeRequests()
                .antMatchers( ...antPatterns: "/user/detail*").authenticated()
                .antMatchers( ...antPatterns: "/h2-console").hasAuthority("ADMIN")
                .and()
                .formLogin()
                .loginPage("/login").failureUrl("/login?error").successHandler(myLoginHandlerCustom())
                .and()
                .logout().logoutUrl("/logout").logoutSuccessUrl("/news?fromLogout=true").deleteCookies("JSESSIONID")
                .invalidateHttpSession(true).clearAuthentication(true).permitAll();
                // And many more security configurations
    }
```

Zdrojový kód 17 – ukázka konfigurace zabezpečení, zdroj vlastní

Na předešlém zdrojovém kódu můžeme vidět ukázku toho, jak takovýto konfigurační soubor vypadá. Nalezneme zde informace o tom, na které URL adresy má jaké uživatelské oprávnění přístup, co se stane po úspěšném či neúspěšném přihlášení, odhlášení a další konfigurace a nastavení přístupu jednotlivých uživatelů, resp. uživatelských rolí.

```
<sec:authorize access="hasAnyAuthority('OPERATOR','ADMIN')">
    <button type="submit" class="btn btn-danger" onclick="form.action='/game/approval';">
        <spring:message code="game.create.approveGame"/>
    </button>
</sec:authorize>
```

Zdrojový kód 18 – ukázka tagu sec:authorize, zdroj vlastní

Pomocí Spring security můžeme autorizovat přístup k jakékoliv funkci, a tudíž povolit volání určitých funkcí výlučně požadovaným rolím. Také můžeme podle autorizace vybrané prvky rovnou skrývat. K tomuto slouží JSP tag *sec:authorize*, který lze vidět na ukázce zdrojového kódu 19. Pokud uživatel nebude povolené role, kód uvnitř tohoto tagu bude skryt.

### 3.3.7 Customizované chybové stránky

Aplikace obsahuje *customizované* chybové stránky, to znamená, že v případě určité chyby se nezobrazí pouze klasická výchozí chyba, ale taková, jakou jsme si pro daný typ chyby definovali. Tato funkcionalita není nepochybně důležitá, ale jedná se o příjemné zpestření chybových stavů. Méně zkušeného uživatele také může strohá chybová stránka pouze s textem vystrašit, avšak když se mu místo toho objeví barevná nastylována chyba, tak z něj obavy nepochybně alespoň trochu opadnou.

Tyto stránky jsou implementovány pro tyto tři HTTP chybové stavy:

- 403 *(Forbidden)* – chyba pro případ, že požadujeme obsah, na který nemáme oprávnění.
- 404 *(Not Found)* – vrácený stav pro případ, že požadovaný soubor nebyl nalezen.
- 500 *(Internal server error)* – obecná chyba, došlo k blíže nespecifikované chybě.

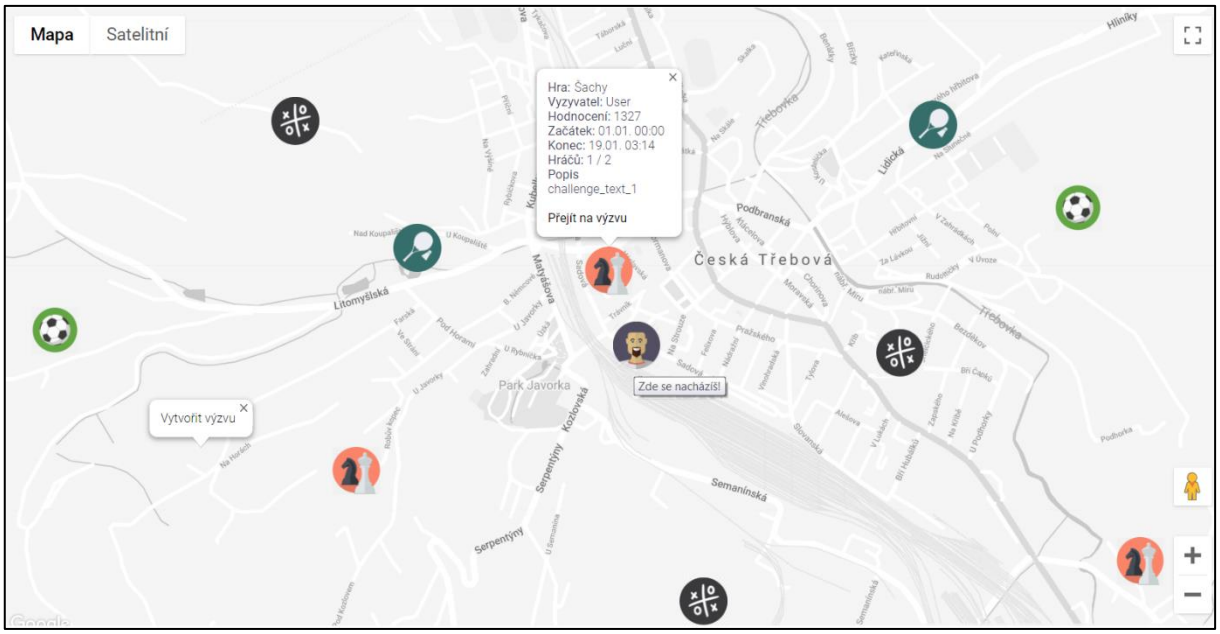

Obrázek 24 – customizovaná chybová stránka pro chybu 403, zdroj vlastní

# 4 FUNKCIONALITA A OBSLUHA VYTVOŘENÉ APLIKACE

Tato kapitola popisuje funkcionalitu a obsluhu vytvořené aplikace. Popis jednotlivých částí je koncipován tak, aby s touto „uživatelskou příručkou“ zvládl aplikaci obsluhovat i méně zdatný uživatel PC.

## 4.1 Mapa s výzvami

Mezi hlavní uživatelskou část aplikace patří možnost zobrazení mapy s výzvami na konkrétní aktivity. Pro tuto možnost vybereme z hlavního menu volbu *Mapa*. Na této mapě lze vidět uživatelovu polohu a konané aktivity v jeho blízkosti, viz následující obrázek.

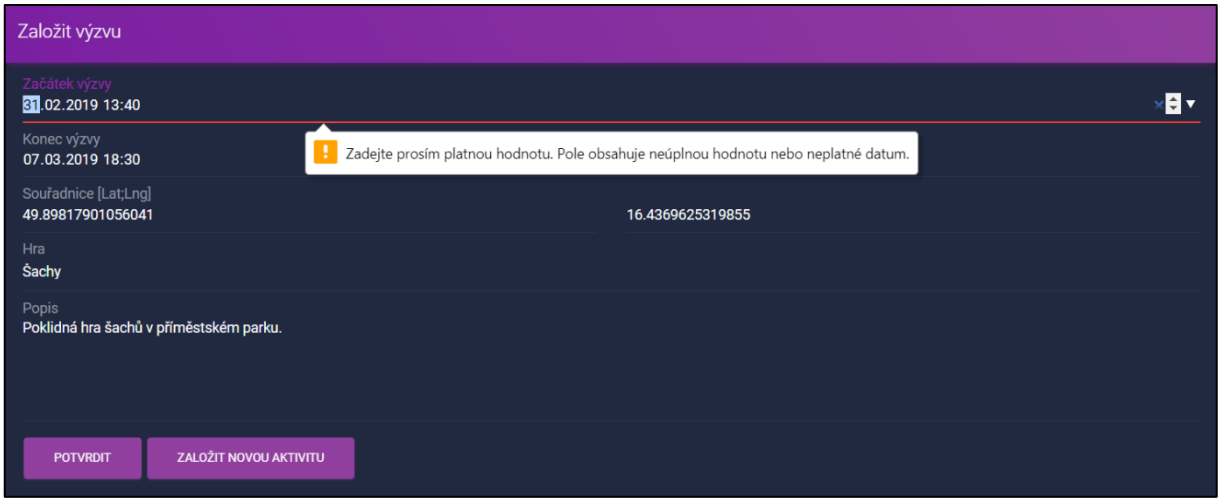



Obrázek 25 – zobrazení mapy výzev, zdroj vlastní

Tuto mapu lze libovolně přepínat mezi satelitním či klasickým zobrazením (možnosti vlevo nahoře), přibližovat nebo oddalovat (volby vpravo dole) a také využívat možnosti zobrazení *Street view*, díky čemuž lze nahlédnout na konkrétní místo vybrané výzvy v „reálném světě“ a ne pouze na mapě. Ukázka tohoto zobrazení je k nahlédnutí v příloze A.

V případě, že uživatel chce založit novou výzvu, stačí na požadovaných souřadnicích stisknout na půl vteřiny myš, resp. v případě responsivního zobrazení přidržet prst. Poté je zobrazeno okno dotazující se, zdali chcete na těchto souřadnicích *„Vytvořit výzvu“*. Po potvrzení této možnosti je uživatel přesměrován na formulář s potřebnými údaji k tomuto kroku. Jedná se o čas začátku a konce výzvy, souřadnice (vyplněné automaticky podle určeného

místa na mapě), zvolení konkrétní aktivity a případně popisu. Veškeré vstupy formuláře jsou validovány, aby bylo zabráněno vložení neočekávaných hodnot a tím pádem předejito neočekávanému chování aplikace. Ukázku formuláře zakládající novou výzvu lze vidět na následujícím obrázku.




Obrázek 26 – vytvoření nové výzvy, zdroj vlastní

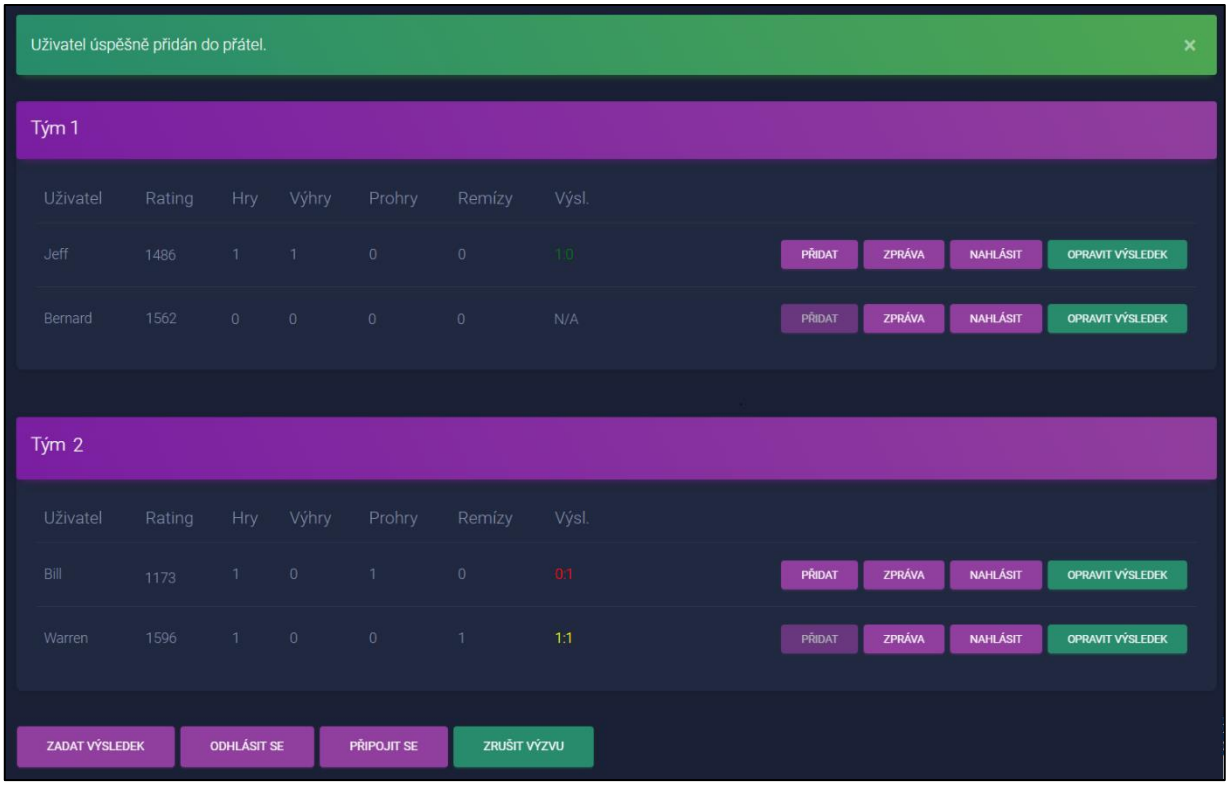
Po vyplnění formuláře validními hodnotami stačí pouze potvrdit stejnojmenným tlačítkem. Výzva je tím vytvořena a zobrazena pro ostatní uživatele na mapě.

Posledním nepopsaným prvkem je tlačítko *Založit novou aktivitu*. To slouží pro případ, že uživatel bude chtít založit výzvu na aktivitu, kterou aplikace ještě momentálně nepodporuje. Více o této funkcionalitě bude ještě řečeno v kapitole *4.9 Schvalování nových aktivit*.

## 4.2 Detail výzvy

Určité informace o výzvě jsou k zjištění ihned při zobrazení na mapě, jak bylo možné vidět na obrázku 25 v minulé kapitole. Stačí pouze na vybranou výzvu kliknout a je zobrazeno okno se základními informacemi. V tomto okně zjistíme, na kterou aktivitu je výzva založena, který uživatel ji založil, jeho aktuální hodnocení v této aktivitě, začátek či konec výzvy, aktuální počet přihlášených hráčů a krátký popis s dodatečnými informacemi, pokud jsou třeba. Poslední možností tohoto okna je volba *Přejít na výzvu*, která uživatele přesměruje na detail výzvy. Jak detail výzvy vypadá, můžeme vidět na následující stránce.

Na této stránce uživatel nalezne další informace a funkcionalitu týkající se vybrané výzvy. Především zde najde možnost se k výzvě připojit – tlačítkem *Připojit se*, popř. *Odhlásit*, v případě, že je již připojen. Na výzvu, kde je již maximální počet hráčů se samozřejmě připojit nelze.

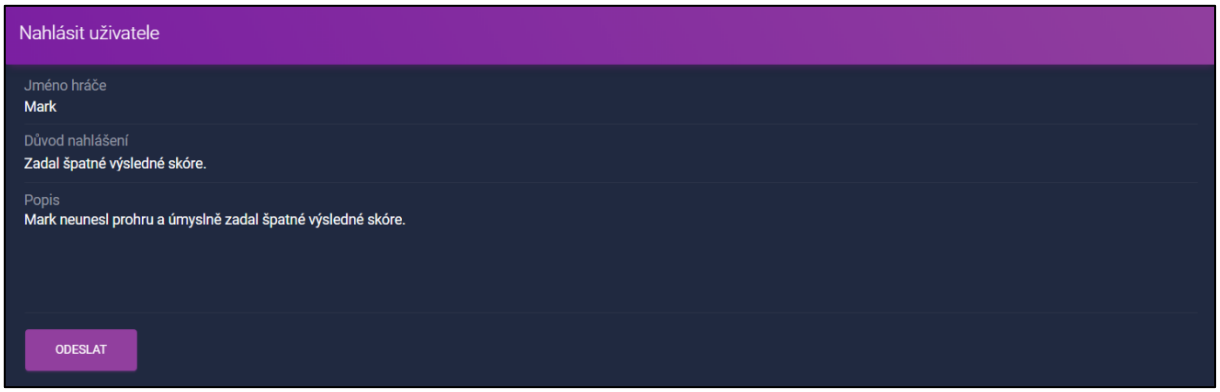



Obrázek 27 – detail výzvy, zdroj vlastní

Nejpodstatnější informací je zde rozdělení hráčů do jednotlivých týmů. Rozdělení probíhá na základě jejich *ratingu* (hodnocení) v dané aktivitě, kvůli tomu, aby byly týmy vyváženy podle schopností hráčů. O každém hráči jsou zde k nalezení informace o jeho hodnocení, počtu odehraných her v dané aktivitě, počet výher, proher a remíz. Také zde nalezneme informaci o tom, jaký výsledek uživatel nakonec po skončení výzvy zadal, tzn. skóre jednotlivých týmů.

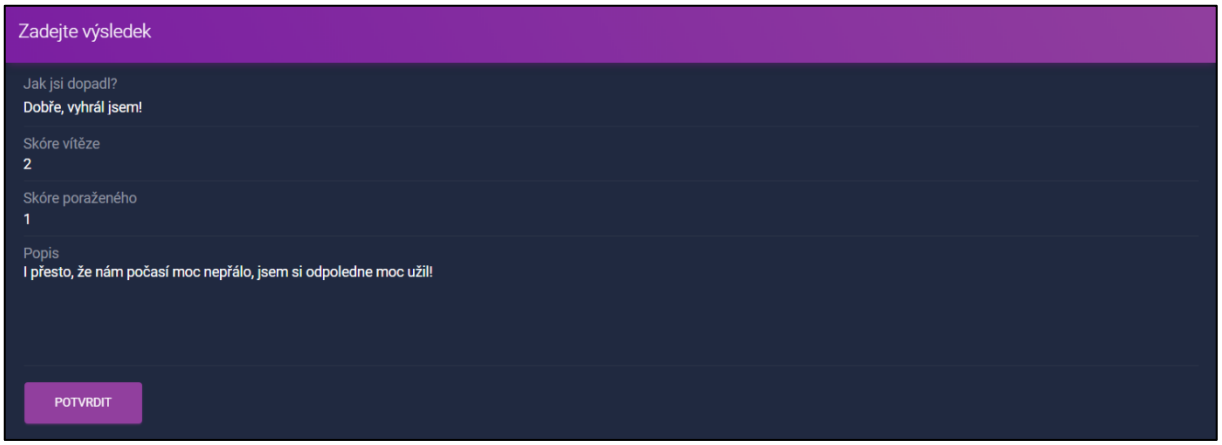
Uživatel tu má dále možnost vybraného hráče *přidat do přátel* (více v kapitole *4.3 přátelé*), *poslat zprávu* (více v kapitole 4.4 zprávy), *opravit zadaný výsledek* (více v kapitole *4.8 Rozhodování sporných výzev*) nebo uživatele *nahlásit za špatné chování*. V případě, že se uživatel choval nevhodně, zadal špatné výsledné skóre, na výzvu vůbec nedorazil nebo se jinak provinil proti pravidlům slušného chování je dána uživatelům možnost toto postihovat a dané hráče pro tyto neřesti nahlásit. Formulář pro nahlášení uživatele lze vidět na následující stránce.

V tomto formuláři uživatel vyplní pouze jméno hráče, důvod jeho nahlášení, a pokud je třeba, tak nějaký doplňující popis incidentu. V případě častého porušování pravidel a podle závažnosti provinění by se poté mohl hráč určitým způsobem penalizovat.




Obrázek 28 – nahlášení uživatele, zdroj vlastní

Pokud proběhlo všechno v pořádku a výzva se odehrála, je povinností každého hráče zadat její výsledné skóre. K tomu slouží tlačítko *Zadat výsledek* (obrázek 27). Po jeho stisknutí je uživatel přesměrován na následující formulář. Zde zadá informace o tom, jestli vyhrál, prohrál či remízoval a výsledné skóre vítěze, resp. poraženého. Jako i v ostatních formulářích zde může napsat libovolný popis. Formulářové pole jsou samozřejmě validována.



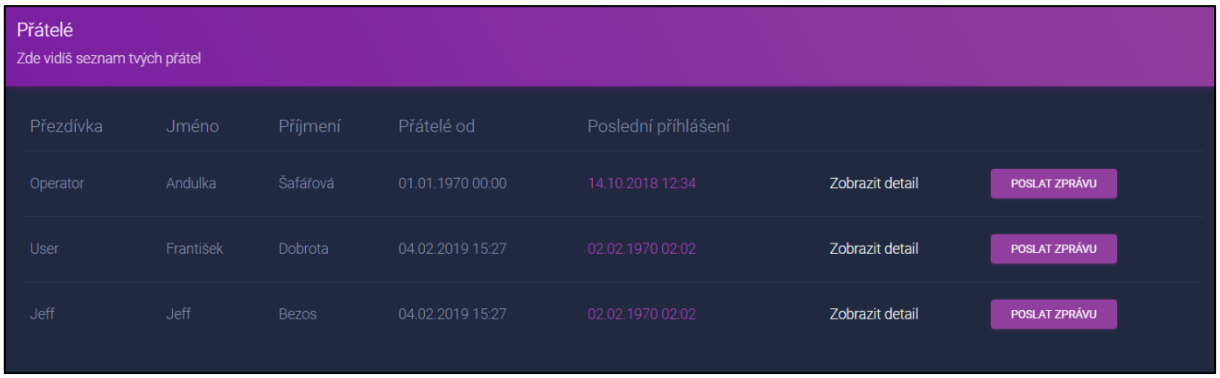

Obrázek 29 – zadávání výsledku, zdroj vlastní

Po tom, co všichni hráči dané výzvy zadají shodné výsledné skóre, je výzva ukončena. Podle výsledku je pak přepočítán rating hráčů pro danou aktivitu. Tento výpočet probíhá na základě toho, zdali uživatel hrál proti zkušenějším hráčům (měli větší *rating*) nebo naopak zdali hrál proti méně zkušeným (měli menší *rating*).

## 4.3 Přátelé

Uživatelům je dána možnost přidat si ostatní hráče do svých přátel. Jedna z možností, jak toto provést, byla popsána v minulé kapitole, další možností je poté přímo na profilu daného hráče.

Na seznam přátel se je možné dostat po zvolení možnosti *Přátelé*, nacházející se v hlavním menu. Uživatel je přesměrován na následující stránku.

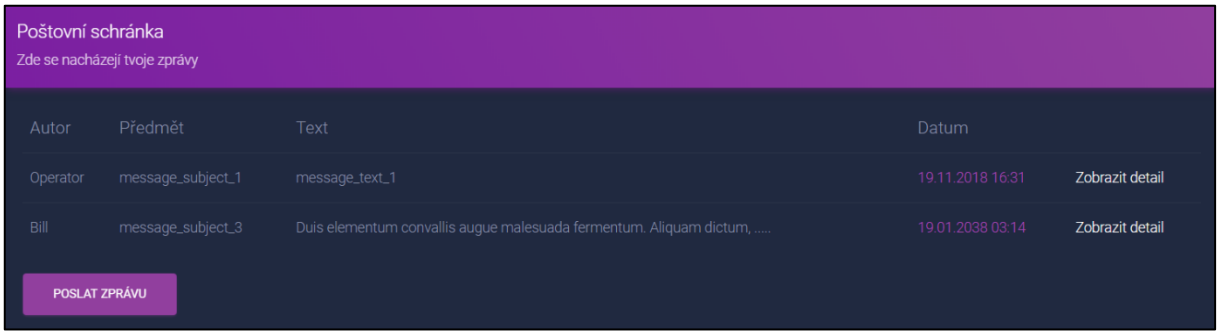



Obrázek 30 – přátelé, zdroj vlastní

Na této stránce můžeme nalézt seznam přátel s jejich základními informacemi, jako je přezdívka, jméno, příjmení či datum s jejich poslední aktivitou. Kromě jiného se z této stránky lze přesměrovat na uživatelský profil vybraného přítele, a to pomocí tlačítka *Zobrazit detail* či konkrétnímu uživateli poslat zprávu pomocí tlačítka *Poslat zprávu*.

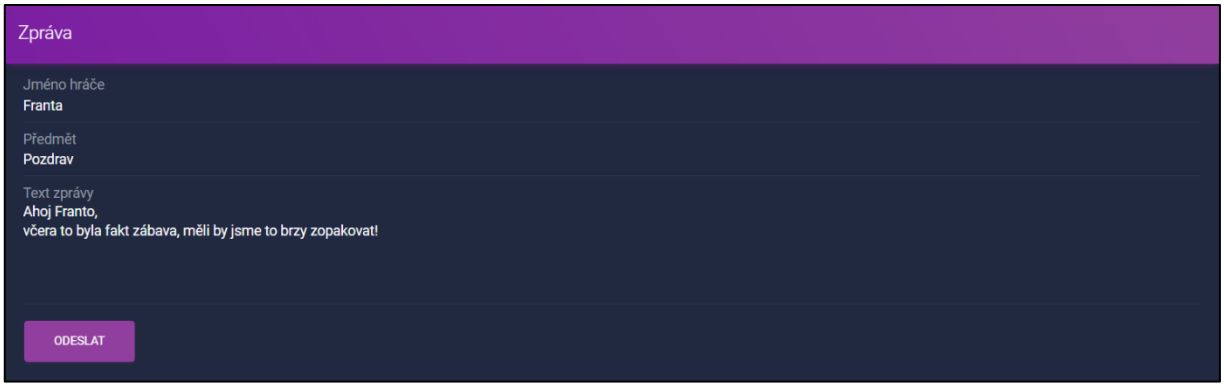
## 4.4 Zprávy

Uživatelé mají mezi sebou možnost komunikace pomocí zpráv. Do této nabídky se lze dostat způsoby popsanými v předchozích kapitolách, z profilu uživatele nebo volbou z menu: *Zprávy*.




Obrázek 31 – poštovní schránka, zdroj vlastní

Zde nalezneme informace o odesilateli, předmětu, náhled samotného textu zprávy a datu odeslání. Volbou *Zobrazit detail* je zobrazen detail zprávy s celým textem a možností rovnou na vybranou zprávu odpovědět. Tlačítko *Poslat zprávu* přesměruje uživatele na formulář určený k vytvoření zprávy, jak lze vidět na následujícím obrázku.

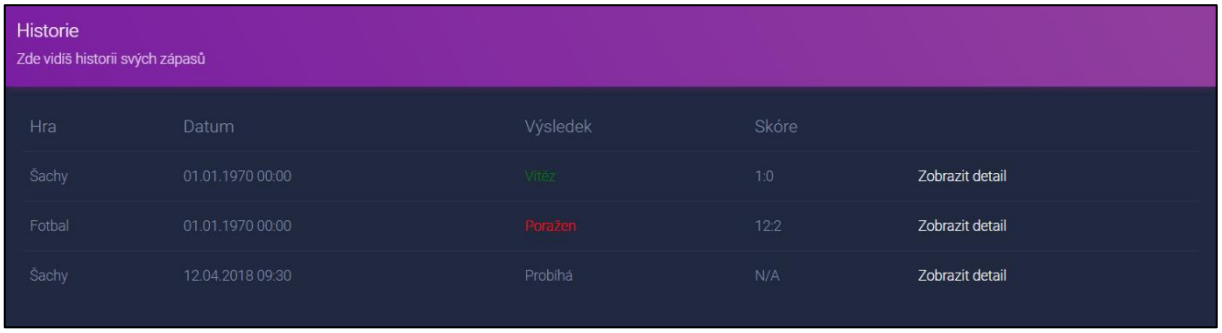



Obrázek 32 – vytvoření zprávy, zdroj vlastní

Formulář očekává informace o jménu hráče, předmětu a samotného textu zprávy. Zpráva je odeslána tlačítkem *Odeslat*. Formulářová pole jsou validována, tudíž je zamezeno odeslání zprávy neexistujícímu uživateli, popř. zprávy bez předmětu.

## 4.5 Historie

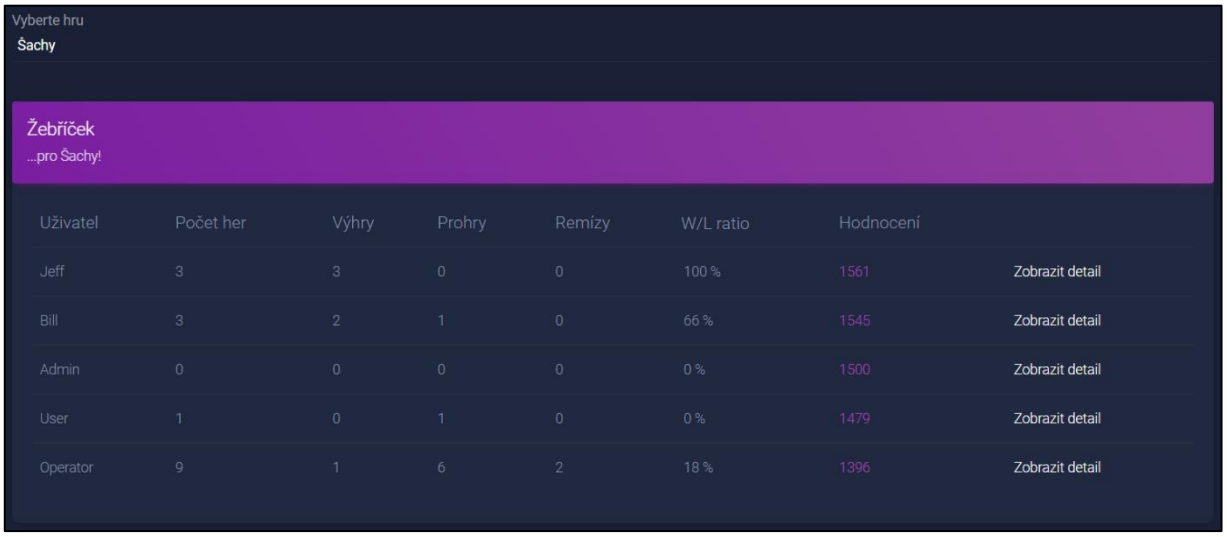
Volbou *Historie* je uživateli zobrazena historie jeho odehraných her. Zde nalezne informace o tom, jak si v jednotlivých výzvách vedl, včetně informací o výsledném skóre. Možností *Zobrazit detail* dojde k přesměrování na detail výzvy, která již byla znázorněna na obrázku 27. Na následujícím obrázku je znázorněna ukázka historie hráče.




Obrázek 33 – historie, zdroj vlastní

## 4.6 Žebříček

Žebříček slouží k zobrazení informací o hodnocení uživatelů ve vybrané hře. Na následujícím obrázku můžeme vidět žebříček pro hru šachy.




Obrázek 34 – žebříček, zdroj vlastní

Pro zobrazení žebříčku konkrétní aktivity uživatel vybere požadovanou volbu z nabídky *Hra*. Na této stránce jsou k dispozici informace o tom, jak si jednotlivé uživatelé ve vybrané aktivitě vedou oproti ostatním. Mají zde možnost vidět údaje jako je počet odehraných her dané aktivity, kolikrát vyhráli, prohráli či remízovali. Dále je zde zobrazena hodnota W/L ratio[27] udávající procentuální úspěšnost hráče v dané aktivitě. V neposlední řadě je zde informace o samotném hodnocení uživatele. Tlačítkem *Zobrazit detail* se lze přesměrovat na profil vybraného hráče.

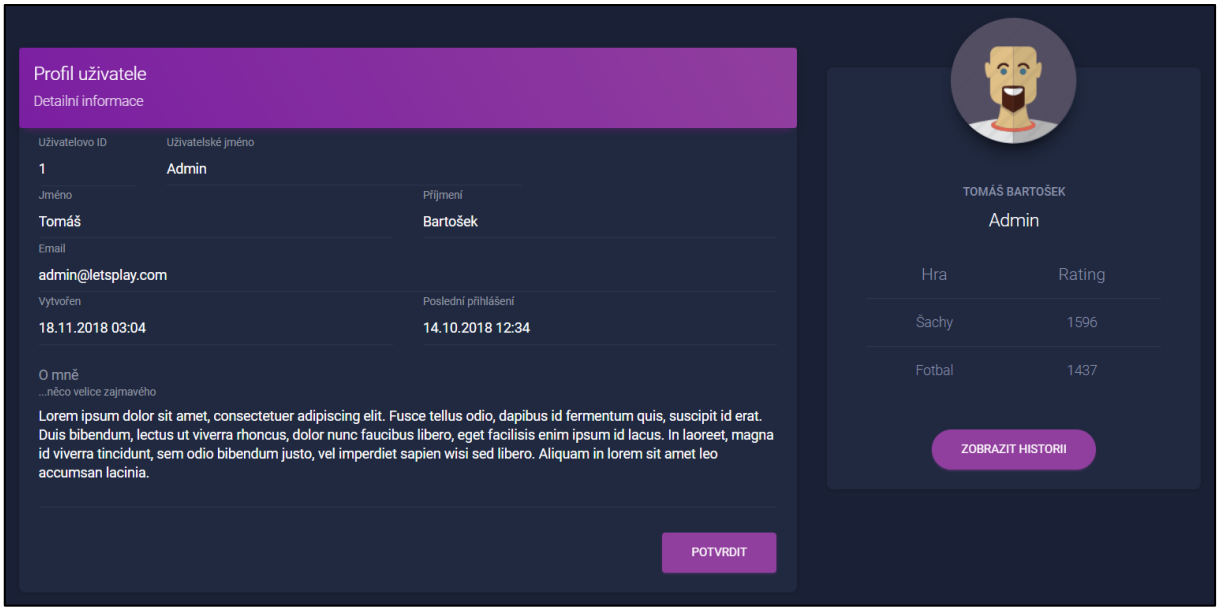
## 4.7 Profil

Na profil uživatele se lze dostat pomocí volby z hlavního menu: *Profil uživatele* nebo i z ostatních částí aplikace, jak bylo znázorněno v předešlých kapitolách.

Na této stránce nalezneme osobní informace, které se týkají vybraného uživatele, tím je např. uživatelovo reálné jméno, příjmení, kdy byl jeho účet vytvořen, datum posledního přihlášení či jeho krátké představení. Ukázku lze vidět na obrázku na následující stránce.

[27] W/L ratio (Win-Loss ratio) – poměr vyhraných her k prohranným.

Pokud je uživatel vlastníkem prohlíženého profilu, má možnost svoje údaje měnit. Tlačítkem *Potvrdit* provedené změny uložíme. Profil ostatních hráčů samozřejmě upravovat nelze. V případě, že se jedná o profil jiného uživatele, je tlačítko *Potvrdit* nahrazeno tlačítkem *Poslat zprávu*. Tato funkcionalita byla popsána již v předchozí kapitole.

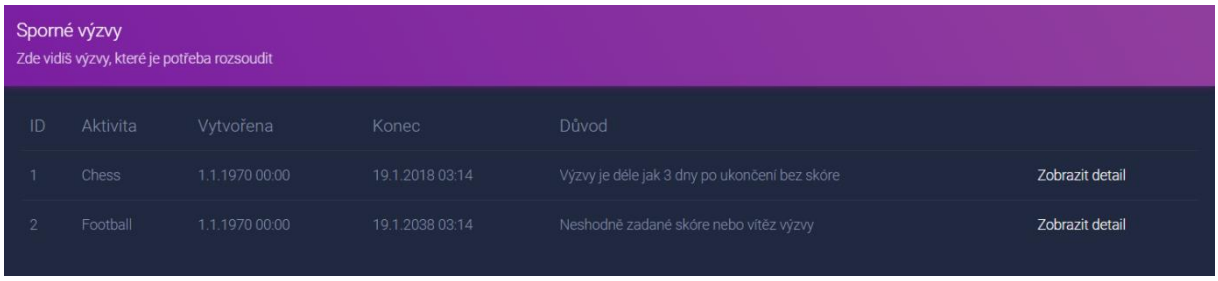



Obrázek 35 – profil uživatele, zdroj vlastní

Kromě osobních údajů, tu lze najít i hodnocení hráče v jednotlivých aktivitách. Možností *Zobrazit historii* si lze zobrazit uživatelovu historii, viz kapitola *4.5 Historie*.

## 4.8 Rozhodování sporných výzev

Po zvolení možnosti *Sporné výzvy*, která je přístupná pouze uživateli s rolí operátor, je takovýto uživatel přesměrován na stránku, jakou lze vidět na následujícím obrázku.




Obrázek 36 – sporné výzvy, zdroj vlastní

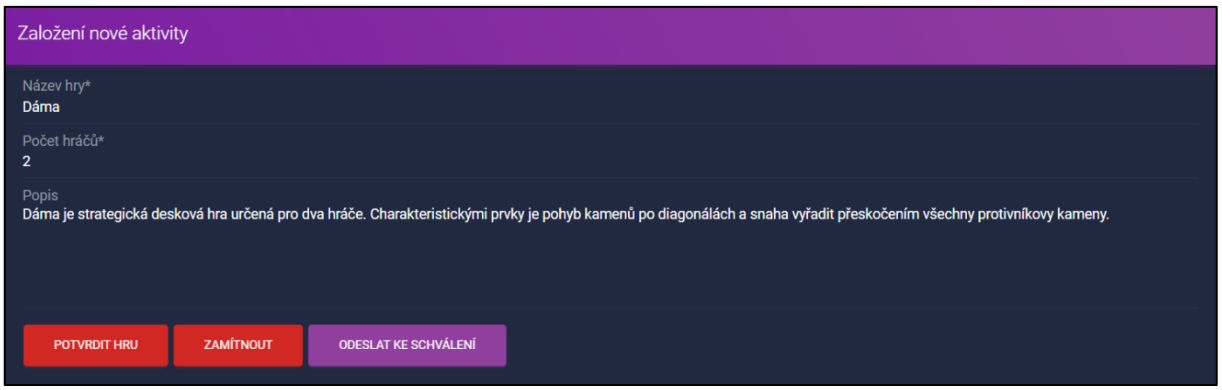
Pro případ, že se uživatelé neshodnou na zadaném výsledném skóre slouží role operátor. Uživatel s touto rolí má možnost těmto sporným výzvám výsledné skóre změnit podle svého uvážení. Díky tomu se výzvy konečně ukončí a přepočítá se *rating* jednotlivých hráčů.

Na této stránce lze nalézt výpis všech výzev, kde není zadané výsledné skóre jednotné od všech uživatelů nebo daná výzva nemá více jak tři dny po ukončení zadané skóre. Po zvolení možnosti *Zobrazit detail* je zobrazen detail výzvy, jak bylo znázorněno na obrázku 27. Na této stránce poté uživatel s rolí operátor, resp. administrátor použije tlačítko *Opravit výsledek* a vybranému uživateli tak upravit jeho zadané skóre. Obrazovka na úpravu skóre je totožná se stránkou, která byla znázorněna na obrázku 29.

## 4.9 Schvalování nových aktivit

Aplikace samozřejmě neobsahuje veškeré aktivity, které se na světě vyskytují. V situacích, že by uživatel rád aplikaci využíval, ale ještě neobsahuje jeho oblíbenou aktivitu, má možnost tuto aktivitu navrhnout.

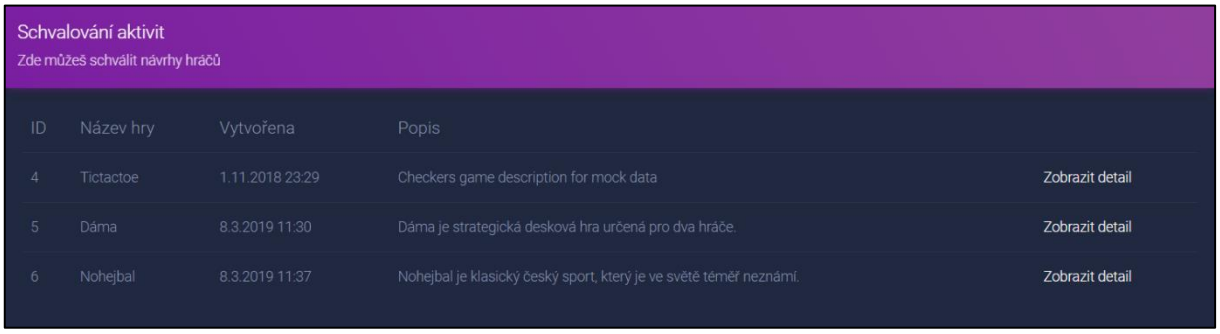
Jak bylo zmíněno v kapitole *4.2 Detail výzvy*, k založení nové aktivity je možné se dostat pomocí tlačítka *Založit novou aktivitu*, viz obrázek 26. Tato volba uživatele přesměruje na následující formulář.




Obrázek 37 – založení/schvalování nové aktivity, zdroj vlastní

Na této stránce je poté nutné vyplnit název navrhované aktivity, pro kolik hráčů je a případně popis. Poté uživatel formulář potvrdí tlačítkem *Odeslat ke schválení* a daná aktivita je vytvořena a čeká na potvrzení. Schvalovat nové aktivity může pouze uživatel s rolí administrátor, a to pomocí tlačítka *Potvrdit hru*. Před potvrzením může případně parametry vytvářené aktivity ještě upravit. Samozřejmě má také možnost na základě svého zvážení navrhovanou aktivitu zamítnout, a to pomocí tlačítka *Zamítnout*.

K tomu, aby administrátor věděl, které aktivity čekají na schválení, slouží nabídka z hlavního menu: *Schvalování aktivit*. Po jejím zvolení jsme přesměrování na následující stránku.

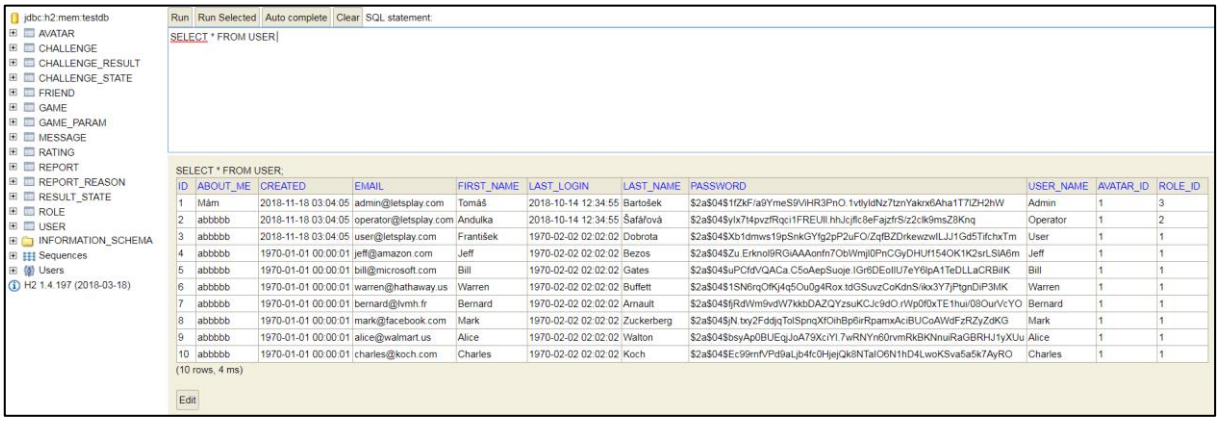



Obrázek 38 – schvalování nových aktivit, zdroj vlastní

Na této stránce nalezneme seznam všech aktivit čekající na potvrzení. Jsou zde základní informace, jako je název aktivity, kdy byla vytvořena a její popis. Tlačítkem *Zobrazit detail* je uživatel přesměrován na formulář, který byl již popsán u obrázku 37, kde může danou aktivitu finálně potvrdit, resp. zamítnout.

## 4.10 Administrátorská konzole

Administrátorská konzole je přístupná pouze uživatelům s přiřazenou rolí administrátor. Pomocí této konzole je možné zobrazovat a modifikovat veškerá data použitá v aplikaci. Jedná se tedy o přímý přístup k databázi. Do této konzole se dostaneme pomocí červeného tlačítka z hlavního menu: *Administrátorská konzole*.



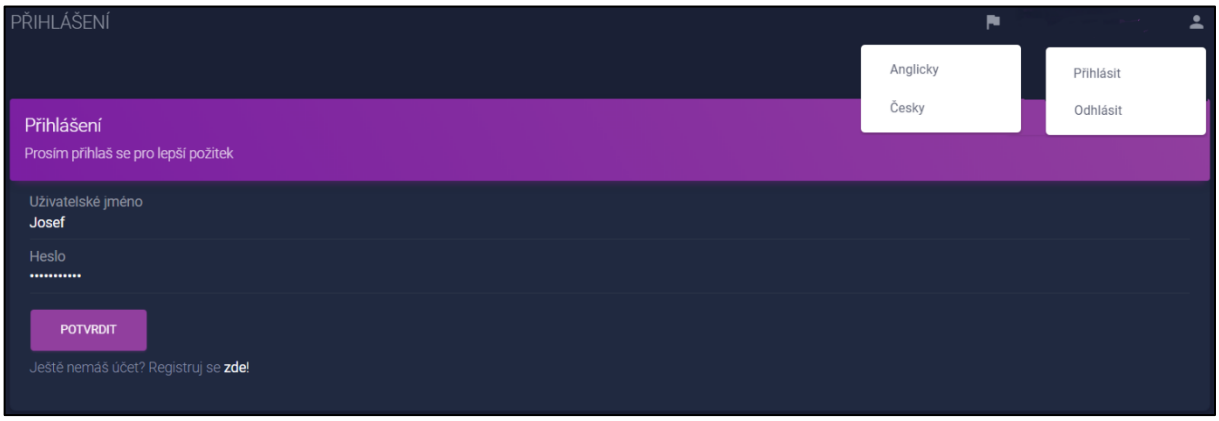

Obrázek 39 – administrátorská konzole, zdroj vlastní

V levém sloupci je výpis tabulek, které aplikace využívá. V horní části lze spouštět libovolné SQL dotazy a ve spodní části se nám k těmto dotazům dostává odpovědi.

## 4.11 Obecná funkcionalita

Jako každá aplikace, i tato obsahuje určitou běžnou funkcionalitu, jako je například přihlášení, odhlášení, změna jazyka, informační hlášky apod.

Většina funkcionalit, které byly popsány v předešlých kapitolách, je přístupná pouze registrovaným, resp. přihlášeným uživatelům. Tyto funkcionality jsou podle nastavených práv uživatelům zobrazeny, resp. skryty. Přihlašovací formulář je možné vidět na následujícím obrázku.



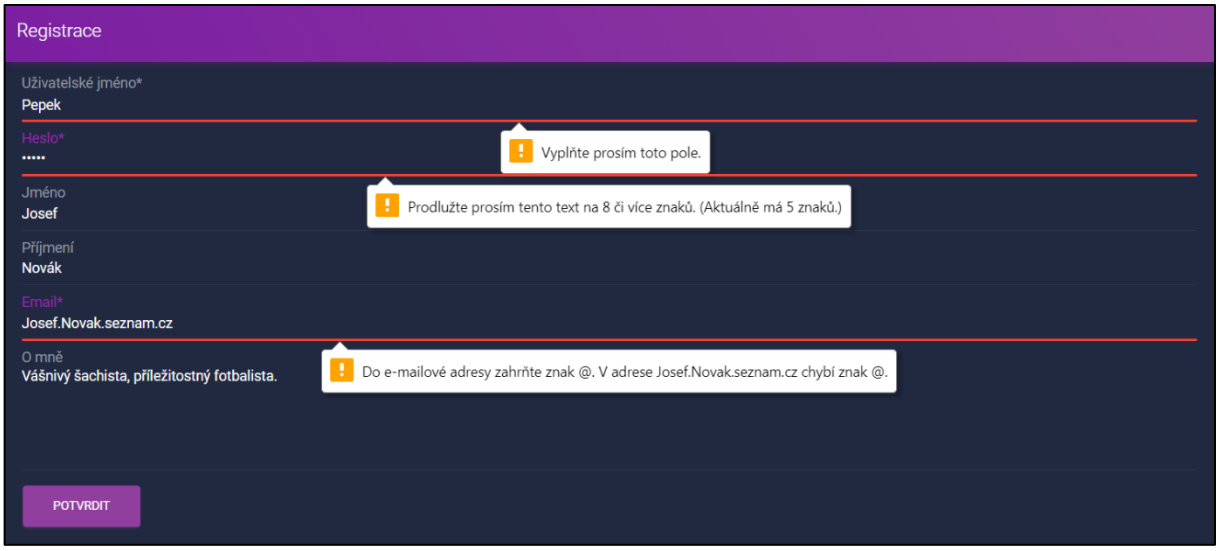

Obrázek 40 – přihlášení, zdroj vlastní

Obrazovka s přihlášením je výchozí stránkou při otevření aplikace, popř. je přístupná přes volbu *Přihlásit*, která je vidět v pravém horním rohu na předešlém obrázku. Stejně tak je zde možnost k odhlášení uživatele. Vedle těchto možností je zde ještě nabídka na změnu jazyka aplikace. Na výběr je aktuálně pouze ze dvou jazyků, a to českého a anglického. Po výběru jsou veškeré texty v aplikaci přeloženy do zvoleného jazyka.

Samotný formulář k přihlášení obsahuje klasicky pouze dvě položky, a to uživatelské jméno a heslo. V případě, že zadané hodnoty jsou korektní, tlačítkem *Potvrdit* dojde k přihlášení uživatele.

Pokud uživatel ještě není registrovaný, přihlásit se samozřejmě nemůže. K registraci se dostaneme přes odkaz v textu *„Ještě nemá účet? Registruj se zde!“*. Tento formulář je znázorněn na následující stránce.

V registračním formuláři jsou povinná pole označena hvězdičkou. Těmito poli je uživatelské jméno, heslo a email uživatele. Nepovinné údaje jsou jméno, příjmení a popis samotného hráče. Veškerá pole jsou stejně jako v ostatních formulářích validována proti zadání nevalidních vstupů.




Obrázek 41 – registrace, zdroj vlastní

Pro větší srozumitelnost a zjednodušení obsluhy se v aplikaci vyskytují informační hlášky, které uživatelovi „komentují“ jeho činnost. Typicky se jedná o potvrzení nebo zamítnutí uživatelova konání. Na následujícím příkladu lze vidět, jak tyto hlášky vypadají v případě, že se uživatel přihlásí korektně a v případě, kdy zadá špatné přihlašovací údaje.


Obrázek 42 – informační hlášky, zdroj vlastní

# ZÁVĚR

Cíle diplomové práce se podařilo splnit. V teoretické části byla nejprve provedena rešerše obdobných, alespoň myšlenkou podobných, již existujících aplikací. Vybrány byly aplikace: Sport buddy, Vaše liga, Rovo a Společné aktivity. Na závěr bylo provedeno jejich srovnání, s vytvářenou aplikací, na základě vybraných parametrů. V další části byl stručně představen framework Spring. Bylo zde popsáno k čemu tento framework slouží, z čeho se skládá a co nabízí. Podrobněji bylo rozebráno, co je to inversion of control a dependency injection. Kapitoly byly také věnovány aspektově orientovanému programování, a především pak podpoře vývoje webových aplikací, kterou Spring nabízí. Spring je opravdu mohutný framework, o kterém by bylo možné napsat desítky prací. Popis na pár stran, jako je tomu v případě této práce, určitě není k pochopení dostačující. Účelem bylo pouze tento framework představit, protože je základním kamenem, na němž je postavena celá vytvářená aplikace. Pro jeho plné pochopení je doporučeno navštívit oficiální dokumentaci nebo přečíst některou z desítek knih, které se mu věnují.

V praktické části byla úspěšně vytvořena aplikace na správu aktivit. Nejprve byla provedena analýza funkčních a nefunkčních požadavků. Následně zde byly vytvořeny ukázky UML diagramů a čtenář byl proveden vývojem aplikace od grafické podoby, přes návrh databáze, až po samotnou implementaci podle architektury MVC. Dozvěděli jsme se zde také něco o zabezpečení a způsobu podpory vícejazyčnosti aplikace. Kapitola byla také věnována popisu technologií a nástrojů, které k tomuto vývoji byly využity. Jejich výčet není samozřejmě kompletní, nicméně smyslem práce nebylo popsat úplně všechny. V poslední kapitole byla představena veškerá funkcionalita vytvořené aplikace, včetně způsobu její obsluhy, a to takovým způsobem, aby i počítačově negramotní uživatelé byli schopni výslednou aplikaci obsluhovat.

Vytvořená aplikace je funkčním prototypem, na kterém se bude pracovat i po odevzdání této práce a bude snaha ji dovést do takového stavu, aby byla opravdu schopná k ostrému nasazení mezi reálné uživatele. V průběhu zpracování vzešlo nemalé množství otázek, zdali by nešly nějaké věci udělat lépe, popř. zdali jsou opravdu uživatelsky přívětivé. Dalším plánovaným krokem je její přepsání do podoby SPA (single-page application), čehož bude docíleno využitím některého javascriptového frameworku, jako je např. Angular, React či Vue. Některé použité technologie jsou totiž již relativně zastaralé (JSP) a není důvod je nenahradit modernějšími. V budoucnu je také v plánu vytvořit mobilní aplikaci s totožnou funkcionalitou.

# POUŽITÁ LITERATURA


[1] *SportBuddy* [online]. London, ©2017 [cit. 2019-03-24]. Dostupné z: http://www.sportbuddy.io/

[2] PATKO, Brittany. *SportsBuddy Aims To Be Tinder For Sports And Fitness Fans* [online]. 2015 [cit. 2019-03-24]. Dostupné z: https://www.sporttechie.com/sportsbuddy-is-aims-to-be-tinder-for-sports-and-fitness-fans/

[3] *Chci začít* [online]. Praha, ©2018 [cit. 2019-03-24]. Dostupné z: https://www.vaseliga.cz/help/want-start

[4] *New Sports Platform - Rovo App | The Tennis Foodie* [online]. Makati, ©2019 [cit. 2019-03-24]. Dostupné z: http://thetennisfoodie.com/new-sports-platform-rovo-app/

[5] JEFFREY, Annabelle. *Because Tennis Buddies Aren't Easy To Find* [online]. 2016 [cit. 2019-03-24]. Dostupné z: https://vulcanpost.com/590876/rovo-singaporean-app-finding-tennis-buddies/

[6] POUČEK, Jiří. *O Společných aktivitách* [online]. ©2008-2017 [cit. 2019-03-24]. Dostupné z: https://www.spolecneaktivity.cz/projekt

[7] JOHNSON, Rod et al. *Introduction to Spring Framework* [online]. ©2004-2014 [cit. 2019-03-24]. Dostupné z: https://docs.spring.io/spring/docs/4.0.x/spring-framework-reference/html/overview.html

[8] MAPLE, Simon. *Java Development Tools and Technologies Landscape 2016* [online]. 2016 [cit. 2019-03-24]. Dostupné z: https://jrebel.com/rebellabs/java-tools-and-technologies-landscape-2016/

[9] HORDĚJČUK, Vojtěch. *Vkládání závislostí (dependency injection)* [online]. ©2008-2019 [cit. 2019-03-24]. Dostupné z: http://voho.eu/wiki/vkladani-zavislosti/

[10] *Spring bean life cycle* [online]. ©2019 [cit. 2019-03-24]. Dostupné z: https://www.oreilly.com/library/view/hands-on-high-performance/9781788838382/249b1941-f09c-4b7f-b6c8-fc7ebde09b01.xhtml

[11] KUNČAR, Petr. *Spring - IoC Kontejner* [online]. 2016 [cit. 2019-03-24]. Dostupné z: https://www.itnetwork.cz/java/pokrocile/spring-ioc-kontejner

[12] FOWLER, Martin. *Inversion of Control Containers and the Dependency Injection pattern* [online]. 2004 [cit. 2019-03-24]. Dostupné z: https://martinfowler.com/articles/injection.html

[13] NURI, Marc. *Field injection is not recommended – Spring IOC* [online]. 2018 [cit. 2019-03-24]. Dostupné z: https://blog.marcnuri.com/field-injection-is-not-recommended/

[14] PURCHART, Vašek. *Dependency Injection: předávání závislostí* [online]. 2011 [cit. 2019-03-24]. Dostupné z: https://www.zdrojak.cz/clanky/dependency-injection-predavani-zavislosti/

[15] *Spring AOP Tutorial Example* [online]. ©2016 [cit. 2019-03-24]. Dostupné z: https://howtodoinjava.com/spring-aop-tutorial/

[16] DYBAL, Martin. *Aspektově orientované programování - úvod* [online]. 2015 [cit. 2019-03-24]. Dostupné z: https://www.dotnetportal.cz/blogy/16/Martin-Dybal/6393/Aspektove-orientovane-programovani-Uvod

[17] ROD, Johnson et al. *Aspect Oriented Programming with Spring* [online]. ©2004-2014 [cit. 2019-03-24]. Dostupné z: https://docs.spring.io/spring/docs/2.5.x/reference/aop.html

[18] ČÁPKA, David. *MVC architektura* [online]. 2013 [cit. 2019-03-24]. Dostupné z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor

[19] GUPTA, Lokesh. *Spring MVC Hello World Example* [online]. ©2016 [cit. 2019-03-24]. Dostupné z: https://howtodoinjava.com/spring-mvc/spring-mvc-hello-world-example/

[20] TENNAKOON, Chathuranga. *Spring framework components* [online]. 2017 [cit. 2019-03-24]. Dostupné z: https://springbootdev.com/2017/07/31/spring-framework-component-service-repository-and-controller/

[21] Gediminas, B. *What is HTML? The Basics of Hypertext Markup Language Explained* [online]. 2018 [cit. 2019-03-24]. Dostupné z: https://www.hostinger.com/tutorials/what-is-html

[22] MORRIS, Scott. *What is HTML? The Basics of Hypertext Markup Language Explained* [online]. 2012 [cit. 2019-03-24]. Dostupné z: https://skillcrush.com/2012/04/03/css/

[23] MORRIS, Scott. *Tech 101: What is JavaScript?* [online]. 2012 [cit. 2019-03-24]. Dostupné z: https://skillcrush.com/2012/04/05/javascript/

[24] SVENNERBERG, Gabriel. *Beginning Google Maps API 3*. New York, NY: Apress, c2010. Expert's voice in Web development. ISBN 14-302-2802-4.

[25] KLÍMA, Jan. *Využití Google Maps API pro mapovou aplikaci sportovních aktivit v Orlických horách*. Praha, 2014. Bakalářská práce. České vysoké učení technické v Praze, Fakulta stavební.

[26] *Maven – Introduction* [online]. 2019 [cit. 2019-03-24]. Dostupné z: https://maven.apache.org/what-is-maven.html

[27] *H2 Database - features* [online]. ©2004 [cit. 2019-03-24]. Dostupné z: http://www.h2database.com/html/features.html

[28] KANAPARTHI, Vineeth. *Git and GitHub in a NutShell* [online]. ©2018 [cit. 2019-03-24]. Dostupné z: https://codeburst.io/git-and-github-in-a-nutshell-b0a3cc06458f

[29] *Features - IntelliJ IDEA* [online]. 2019 [cit. 2019-03-24]. Dostupné z: https://www.jetbrains.com/idea/features/

[30] ENGE, Eric. *Mobile vs Desktop Usage: Mobile Growing But Desktop Still Has Wins* [online]. 2017 [cit. 2019-03-24]. Dostupné z: https://www.stonetemple.com/mobile-vs-desktop-usage-mobile-grows-but-desktop-still-a-big-player-in-2017/

[31] *Responzivní design – co je a jak funguje* [online]. ©2019 [cit. 2019-03-24]. Dostupné z: https://www.websites.cz/blog/responzivni-design-co-je-a-jak-funguje/

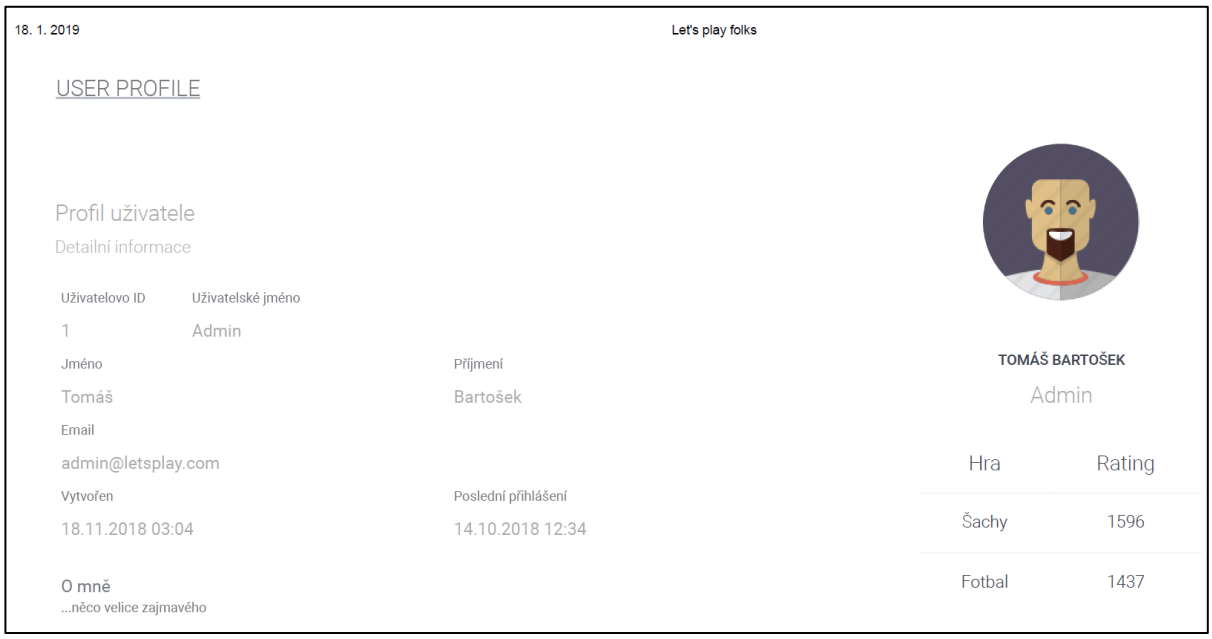
[32] JOSHI, Anand. *Service Layer in Rails Application* [online]. 2015 [cit. 2019-03-24]. Dostupné z: https://blog.harbinger-systems.com/2015/08/service-layer-in-rails-application

[33] RAFFAI, Zoltan. *What Is Spring Boot?* [online]. 2018 [cit. 2019-03-24]. Dostupné z: https://dzone.com/articles/what-is-spring-boot

# PŘÍLOHY

# PŘÍLOHA A – UKÁZKA ZOBRAZENÍ VE STREET VIEW

# PŘÍLOHA B – UKÁZKA APLIKACE V PŘÍPADĚ TISKU



# PŘÍLOHA C – UKÁZKA APLIKACE BEZ CSS

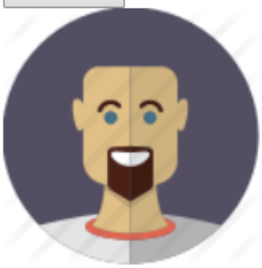
**Profil uživatele**

Detailní informace

Uživatelovo ID 1
Uživatelské jméno Admin
Jméno Tomáš
Příjmení Bartošek
Email admin@letsplay.com
Vytvořen 18.11.2018 03:04
Poslední přihlášení 14.10.2018 12:34
O mně

...něco velice zajmavého abbbbb

Potvrdit



**Tomáš Bartošek**

**Admin**

| Hra | Rating |
|---|---|
| Šachy | 1596 |
| Fotbal | 1437 |

Zobrazit historii