Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Návrh sociální sítě se sportovní tématikou s využitím technologií Java Enterprise Edition

Bakalářská práce

Petr Backstuber

Vedoucí práce: Ing. František Drdák, CSc.

České Budějovice 2014

Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

ZADÁVACÍ PROTOKOL BAKALÁŘSKÉ PRÁCE

Student: **Petr Backstuber**
(jméno, příjmení, tituly)

Obor – zaměření studia: Aplikovaná informatika

Katedra: Ústav aplikované informatiky

Školitel: ing. František Drdák, CSc.
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Garant z PřF: .....
(jméno, příjmení, tituly, katedra – jen v případě externího školitele)

Školitel – specialista, konzultant: .....
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Téma bakalářské práce: **Návrh sociální sítě se sportovní tématikou s využitím technologií Java Enterprise Edition**

Cíle práce:

1. Navrhnout způsob řešení sociální sítě se sportovní tématikou formou webové aplikace.
2. Provést výběr vhodných implementačních technologií a nástrojů vývojové platformy JEE a navazujících systémů.
3. Realizovat vývojový proces aplikace.
4. Kriticky zhodnotit vybrané implementační technologie z hlediska efektivity vývoje aplikace a s ohledem na její případný další rozvoj.

Základní doporučená literatura:

1. http://docs.oracle.com/javaee/7/tutorial/doc/

Financování práce :.....

Vedoucí práce : ing. František Drdák, CSc podpis : .....

U externích vedoucích fakultní garant práce.....podpis : .....

Garant oboru bak.. studia (nepožaduje se u zaměření „příprava na mag. studium biologie)

..... podpis : .....

Vedoucí katedry RNDr. Libor Dostálek podpis .....

Případný souhlas vedoucího ústavu AV .....podpis : .....

V Českých Budějovicích dne 16. 10. 2013

Převzal/a dne 4.11.2013 podpis : Backstuber P.

Bibliografické údaje

Backstuber P., 2014: Návrh sociální sítě se sportovní tématikou s využitím technologií Java Enterprise Edition.
[Sport oriented social network design using Java Enterprise Edition technology. Bc. Thesis, in Czech.] –37p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Tato bakalářská práce využívá technologie Java Enterprise Edition při vývoji aplikace na podporu sociální sítě se sportovní tématikou formou webové aplikace a kriticky hodnotí jejich použitelnost při vývoji aplikace. Zabývá se návrhem aplikace, výběrem implementačních technologií, samotnou implementací a zhodnocením nabytých zkušeností s technologiemi Java Enterprise Edition.

In English

This bachelor's thesis presents Java Enterprise Edition technologies in a development of a sport oriented social network designed in a form of a web application and critically evaluates their applicability in the development of the application. It concerns the design of the application, the selection technologies for implementation and the implementation itself, and evaluation of the experience acquired with the Java Enterprise Edition technologies.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích, dne 12. prosince 2014

Podpis

Poděkování

Rád bych poděkoval panu Ing. Františku Drdákovi, CSc. za pomoc, věnovaný čas a ochotu při odborných konzultacích, které mi pomohli k úspěšnému vytvoření bakalářské práce. Rád bych také poděkoval příbuzným a přátelům za jejich trpělivost během vytváření bakalářské práce.

Obsah

1 Úvod

1.1 Motivace

Využívání nástrojů pro utváření sociálních sítí je již v dnešní době běžnou součástí každodenního života mnoha uživatelů internetu a každým rokem jejich počet roste [1]. Proto je předmětem této práce návrh a realizace webové aplikace na podporu komunikace sociálních skupin se zájmem o sport.

Základním motivem této bakalářské práce je tedy pokusit se navrhnout a realizovat přidanou funkcionalitu, která by vycházela vstříc potřebám těchto sportovně zaměřených uživatelů.

Další motivací této práce je vyzkoušet a zhodnotit vybrané technologie platformy Java Enterprise Edition použité při vývoji této aplikace.

1.2 Cíle práce

- Navrhnout způsob řešení webové aplikace na podporu sociální sítě se sportovní tématikou.
- Provést výběr vhodných implementačních technologií a nástrojů vývojové platformy JEE a navazujících systémů.
- Realizovat vývojový proces aplikace.
- Kriticky zhodnotit využitelnost technologií při vývoji aplikace a možnosti jejího dalšího rozvoje s ohledem na vybrané implementační technologie.

1.3 Struktura práce

Práce se zabývá návrhem a realizací webové aplikace na podporu sociální sítě se sportovní tématikou, pro implementaci je využito vybraných technologií platformy Java Enterprise Editon.

V první kapitole je uvedena motivace, stanovené cíle a členění práce.

Druhá kapitola obsahuje analytickou fázi vývoje aplikace. Popisuje základní požadavky na aplikaci a formou UML diagramů popisuje základní funkcionalitu.

Ve třetí kapitole představuje přehled vybraných implementačních technologií, které platforma Java Enterprise Edition nabízí a popisuje architekturu realizované aplikace.

Dále se práce zabývá provedenou programátorskou realizací webové aplikace a jejích jednotlivých částí. Jedná se o webovou aplikaci pro podporu sociální sítě se základní

funkčností, jako je možnost uživatele přidávat a měnit na profilu své osobní údaje, měnit profilovou fotku, přidávat ostatní uživatele do seznamu přátel, psát komentáře jak na svůj profil, tak na profily přátel a sportů a hlavně sdílet informace o svých osobních výkonech na stránce dosažených výsledků v jednotlivých sportech.

V závěru je provedeno zhodnocení získaných zkušeností s použitými technologiemi. Je vyhodnocena i možnost dalšího rozvoje aplikace v závislosti na použitých technologiích.

2 Analytická část

2.1 Požadavky na aplikaci

Z důvodu, že výsledná aplikace by měla svým uživatelům poskytovat alespoň základní možnosti pro podporu sociální sítě, jsou základními požadavky:

- přidání ostatních uživatelů do seznamu přátel
- možnost přidávání komentářů
- přidávat a měnit osobní údaje

Jako přidaná funkcionalita pro začlenění sportovní tématiky do aplikace pro podporu sociální sítě se zaměřením na sportovně orientované uživatele byla navržena tzv. obrazovka výsledků, kde jsou zobrazeny uživatelovi sportovní výsledky v porovnání s ostatními uživateli registrovanými v aplikaci.

Vyvíjená aplikace je tedy využitelná pro podporu standardní komunikace v rámci sociální sítě a zároveň přidává funkcionalitu ve formě obrazovky výsledků a možnosti správy sportů na profilu uživatele.

Je však nutné si uvědomit, že cílem práce není navrhnout komplexní řešení takovéhoto typu aplikace. Jedná se pouze o aplikační námět, kde se autor snaží spojit zájem o sport s v dnešní době, velice využívanou problematikou sociální sítě a používá tento námět k praktickému ověření technologií poskytovaných platformou Java EE. Z pohledu funkcionality jde o vytvoření prototypu, který by umožnil získání připomínek a dalších námětů od potenciálních uživatelů.

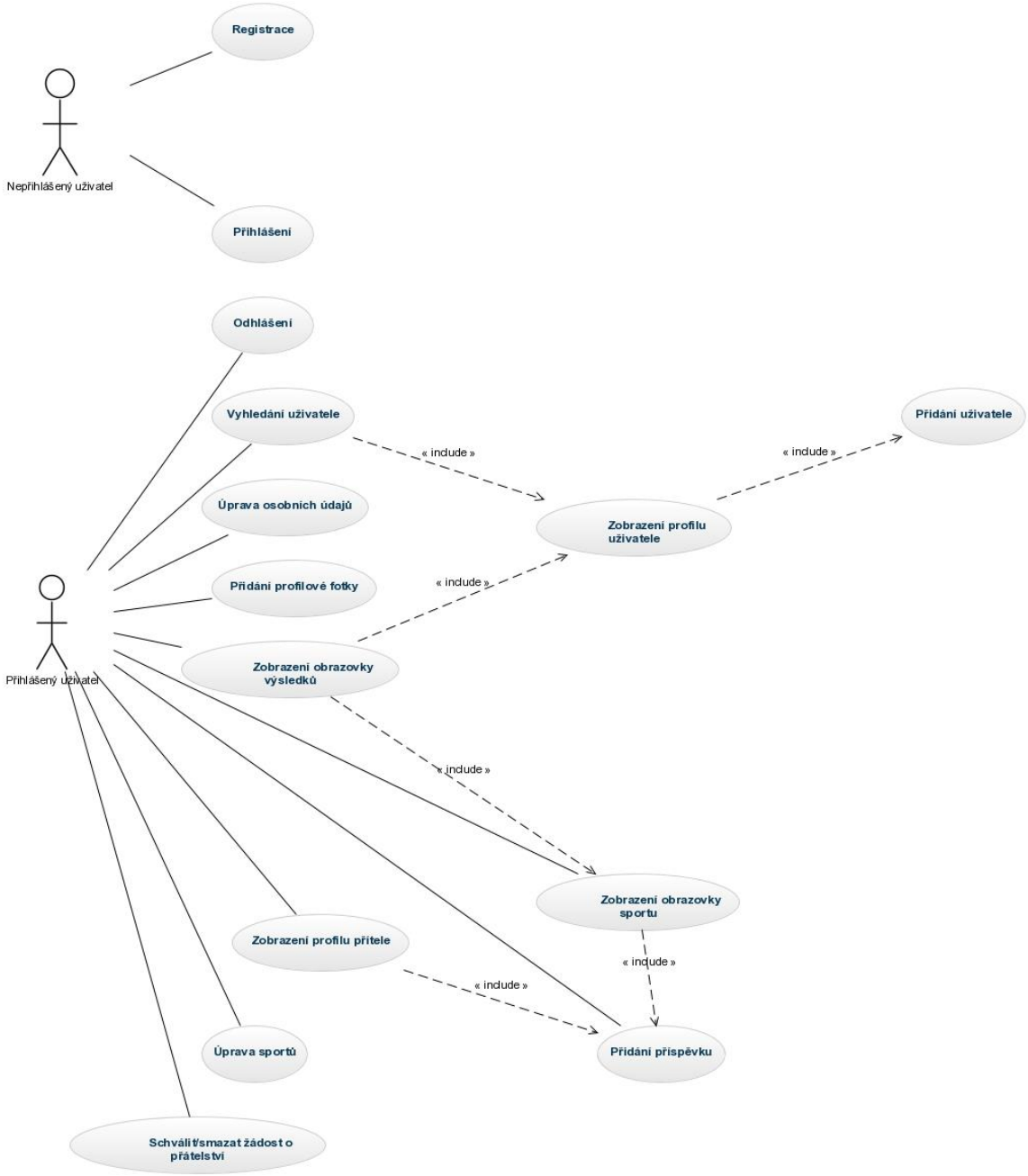
2.2 Funkcionalita

V rámci návrhu aplikace byl vytvořen pro definici interakce mezi uživatelem aplikace a samotnou aplikací use-case diagram. Výsledný diagram obsahuje dva aktéry (role) přistupující do systému:

- Nepřihlášený uživatel
- Přihlášený uživatel

Pokud uživatel není do aplikace přihlášený, vystupuje v roli nepřihlášeného uživatele, tento uživatel může provádět pouze dvě akce a to přihlásit se nebo registrovat.

Naopak, pokud je již uživatel do aplikace úspěšně přihlášený, vystupuje v roli přihlášeného uživatele a může provádět hned několik akcí. Podrobnější popis možných akcí, které smějí přihlášení uživatelé vykonávat je zobrazen na obrázku 1.



Obrázek 1: Use-case diagram

Při návrhu use-case diagramu byly vytvořeny scénáře pro případy užití, které bylo potřeba před implementací podrobněji rozebrat. V ostatních případech nebylo nutné scénáře vytvářet, neboť by se opakovaly, či by popisovaly triviální problematiku a během vývoje by nebyly využity.

Jeden use-case scénář byl zvolen pro případ užití s názvem **Registrace**, scénář popisuje, k čemu samotná akce registrace slouží a jaké kroky vedou až k uložení uživatele. Podle kroků popsaných na obrázku 2, byla následně provedena implementace potřebných metod pro uložení registrovaného uživatele do databáze.

Registrace

Popis

Umožňuje registrovat nového uživatele do aplikace.

Aktéři

- Uživatel
- Aplikace

Podmínky pro spuštění

Uživatel se musí nacházet na stránce registrace.

Hlavní tok

1. Aplikace vygeneruje formulář celkem se 6 vstupními poli, jedním radio button a dvěma tlačítky.
2. Uživatel vyplní povinná pole označená hvězdičkou a splní validační minima pro jednotlivá pole a odešle formulář.
3. Aplikace ověří zadání všech povinných polí a provede validaci vstupních polí.
4. Aplikace prověří, že zadaný uživatelský email již není v aplikaci registrovaný.
5. Aplikace uloží nového uživatele a provede přesměrování na plochu pro přihlášení.

Alternativní tok 1

3.1 Pokud uživatel nevyplní všechna povinná pole nebo nezadá správná data do vstupu/vstupů, zobrazí se mu zpráva o neúspěchu a nedojde k uložení.
3.2 Uživatel doplní chybějící údaje, případně upraví stávající podle chybové zprávy a tok pokračuje od 2. bodu.

Alternativní tok 2

4.1 Pokud už je zadaný email jednou v aplikaci registrovaný, zobrazí se uživateli zpráva o shodě zadávaného emailu a nedojde k uložení.
4.2 Uživatel upraví zadávaný email a tok pokračuje od 2. bodu.

Obrázek 2: Use-case specifikace- Registrace

Další scénář se týká případu užití s názvem **Přidání uživatele**. Tento scénář slouží k popisu procesu zaslání žádosti o přidání do seznamu přátel. Jak je vidět na obrázku 3, po stisku tlačítka pro přidání do přátel není ještě daný uživatel do seznamu přidán, ale čeká se na jeho schválení žádosti.

Na obrázku 4 je znázorněn průběh schválení, či smazání již zaslané žádosti o přátelství.

Přidání uživatele do seznamu přátel

Popis

Provede odeslání žádosti o přidání do přátel.

Aktéři

- Uživatel
- Aplikace

Podmínky pro spuštění

Uživatel musí být přihlášený v aplikaci a nacházet se na stránce uživatele, kterého nemá ještě v přátelích.

Hlavní tok

1. Uživatel klikne na tlačítko přidat do přátel.
2. Aplikace vytvoří v databázi nový záznam v tabulce relationships s atributem accepted nastaveným na hodnotu false.
3. Na profilu přidávaného uživatele se zobrazí žádost o přidání v seznamu: žádosti o přidání do přátel

Obrázek 3: Use-case specifikace- Přidání uživatele

Schválit/smazat žádost o přátelství

Popis

Slouží k potvrzení/smazání již vytvořené žádosti o přátelství.

Aktéři

- Uživatel
- Aplikace

Podmínky pro spuštění

Uživatel musí mít v databázi uloženou žádost o potvrzení přátelství.

Hlavní tok

1. Aplikace v seznamu žádostí o přátelství zobrazuje uživateli všechny ještě nepotvrzené, či nesmazané žádosti.
2. Uživatel vybere možnost potvrdit žádost.
3. V příslušném záznamu v databázi je změněn atribut accepted na true.
4. Uživatel nyní vidí v seznamu přátel nově přidaného uživatele.

Alternativní tok 1

2.1 Pokud uživatel zvolí možnost smazat žádost, dojde ke smazání žádosti.
2.2 Seznam přátel zůstane nezměněný.

Obrázek 4: Use-case specifikace- Schválit/smazat žádost o přátelství

3 Přehled vybraných implementačních technologií

Pro realizaci webové aplikace na podporu sociální sítě byly vybrány následující technologie:

- Aplikační server Glassfish verze 4.0
- JavaServer Faces
- Enterprise JavaBeans
- Java Persistence API
- Databáze MySQL

3.1 Architektura a prostředí

Z hlediska podpory vybraných technologií bylo zvoleno jako vývojové prostředí Netbeans, konkrétně ve verzi 7.3.1 [11].

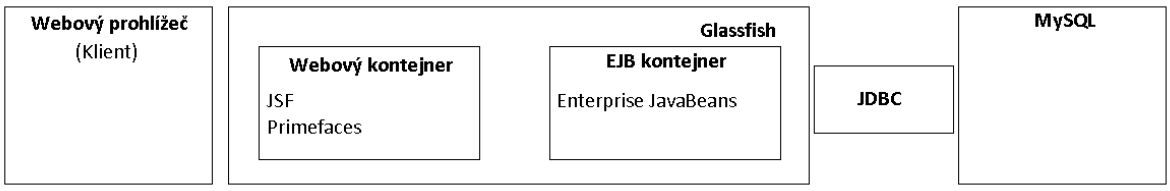
Architektura vychází ze skutečnosti, že se jedná o klasickou webovou aplikaci.

Datová vrstva využívá prostředí MySQL serveru, přičemž pro účely navrhované aplikace je server rovněž přístupný na lokálním počítači. Pro připojení Java aplikace k MySQL databázi je použito JDBC připojení.

Navrhovaná webová aplikace běží v prostředí aplikačního serveru Glassfish verze 4.0. Aplikační server zahrnuje několik typů tzv. kontejnerů a každý kontejner spravuje určité komponenty neboli základní stavební jednotky aplikace běžící na platformě Java EE [9]. Z hlediska realizované webové aplikace je potřeba využít **webové** a **EJB komponenty**, o které se starají stejnojmenné kontejnery, tedy **webový** a **EJB kontejner**.

Dále pro zobrazení získaných dat je potřeba využít na straně klienta webový prohlížeč, který zobrazí výsledný HTML kód přímo uživateli a zajišťuje komunikaci uživatele s aplikací.

Oba servery, jak databázový tak aplikační jsou umístěny v rámci jednoho stroje.



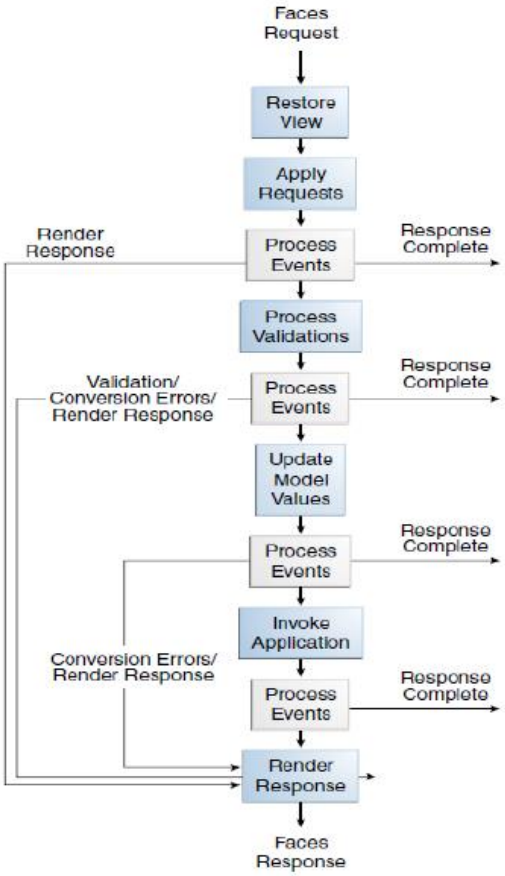
Obrázek 5: Architektura aplikace

3.2 JavaServerFaces

JavaServer Faces technologie je MVC webový Framework, skládá se z API pro reprezentaci komponent a knihovny tagů pro zobrazování komponent [3]. Hlavní výhodou JavaServer Faces, dále již JSF, je, že poskytuje oddělení aplikační vrstvy od prezentační vrstvy. Životní cyklus aplikace využívající JSF je následující:

- Fáze vykonávání:
  - Je sestaven vzhled aplikace
  - V parametrech jsou poslány hodnoty komponent
  - Provede se ověření správnosti (validace) a konverze získaných hodnot z parametrů
  - Atributy v Managed Bean jsou nastaveny na hodnoty z parametrů
  - Poté už je vykonána aplikační logika
- Fáze poskytovací:
  - Požadovaná stránka je vykreslena jako odpověď na původní žádost od klienta

Podrobnější popis životního cyklu JSF aplikace je zobrazen na obrázku 6.



Obrázek 6: Životní cyklus JavaServer Faces aplikace [3]

3.2.1 User Interface Component Model

Komponenty JavaServer Faces jsou stavebními bloky všech JavaServer Faces zobrazení. Tento model poskytuje:

- Sadu tříd **javax.faces.component.UIComponent**
- Vykreslovací model (Rendering model)
- Model konverze
- Event a listener model
- Validační model
- Navigační model

3.2.1.1 Třídy komponent

Základní abstraktní třída pro všechny komponenty je **javax.faces.component.UIComponent**. Každá třída reprezentuje nějakou komponentu v zobrazení a může být reprezentována různě, např. **UICommand**, lze zobrazit jako tlačítko nebo jako prostý odkaz [3]. Dalšími příklady tříd jsou:

- **UIForm**, reprezentuje vstupní formulář.
- **UIInput**, jedná se o vstup, přijímá vstupní data od uživatele.
- **UICommand**, reprezentuje tlačítko nebo odkaz.

3.2.1.2 Vykreslovací model

Třídy komponent mohou být klientovi předány v různém formátu, to udávají třídy **javax.faces.render.Renderer** [4]. Tento přístup je výhodný v tom, že je navržena jedna komponenta se všemi funkčnostmi, ale při rozdílné definici Renderer tříd může být zobrazena různými klienty.

Render kit udává, jak se budou třídy komponent mapovat na tagy, pro rozdílné klienty je potřeba vytvářet zobrazení v různých formátech, například pro webový prohlížeč v počítači (webový klient), je potřeba komponenty poskytovat ve formě HTML tagů, naopak pro mobilní zařízení ve formě XML tagů.

3.2.1.3 Model konverze

Používá se, pokud je potřeba převést získaná data z komponenty na jiný datový typ, než je standardní, jako int, long, … Proto JSF umožňuje připojit ke komponentě svůj vlastní **javax.faces.convert.Converter**.

3.2.1.4 Event a listener model

JSF definuje 3 různé typy událostí: aplikační, systémové a data-model [4].

Aplikační události jsou generovány některou z **UIComponent** a dále se v JSF rozdělují na dva druhy:

- **Action event**, k vyvolání události dochází při aktivaci komponenty, která implementuje **ActionSource**, jedná se o tlačítka, odkazy.
- **Value-changed event**, k vyvolání dojde v případě, že uživatel změní hodnotu některé komponenty, která je reprezentovaná třídou **UIInput**.

Dalším typem událostí definovaných v JSF jsou systémové události, které jsou prováděny během provádění aplikace v předdefinovaných časech. Data-model události se vykonají, pokud je vybrán nový řádek v komponentu **UIData**.

Pro reakci na action nebo value-changed události je možno použít vlastní tag **<f:valueChangeListener>** nebo uvnitř tagu komponenty **<f:actionListener>**, další možností je implementovat metodu uvnitř Managed Bean, která se o odchycení události postará.

3.2.1.5 Model validace

Validační model poskytuje sadu tříd [3], které je možno použít k validaci vstupních dat. K samotné validaci dochází před přiřazením hodnoty do proměnné v Managed Bean. Je možné použít validátory již definované v **javax.faces.validator.Validator** nebo využít validátorů, které obsahují jednotlivé **UIInput** komponenty. Kromě těchto dvou možností lze implementovat svůj vlastní validátor vytvořením metody v Managed Bean, která provede validaci.

3.2.1.6 Navigační model

Slouží ke zjednodušení navigace mezi stránkami, v rámci navigačního modelu jsou možné dva druhy navigace:

- **Implicitní navigace**- Tento typ navigace se provádí, pokud ve zdrojových souborech nejsou pro danou stránku definovaná navigační pravidla, pak defaultní navigační handler zkusí najít odpovídající stránku podle hodnoty atributu „action“ u komponenty [3].

```
<h:commandButton value="Submit" action="response"/>
```

Příklad 1: Implicitní navigace

- **Navigace definovaná uživatelem**- Takováto navigace se definuje ve zdrojových souborech, jako je **faces-config.xml**. Slouží k definování navigačních pravidel samotným uživatelem, lze v každém elementu **navigation-rule** definovat, kam se má provést přesměrování z jedné stránky (definované v elementu **from-view-id**) na jiné stránky v aplikaci. Libovolný počet elementů **navigation-case** definuje, na kterou stránku se má provést přesměrování (element **to-view-id**) na základě výstupu definovaného v elementu **from-outcome**.

```
<navigation-rule>
    <from-view-id>/greeting.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>response.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
</navigation-rule>

<h:commandButton value="Submit" action="success"/>
```

Příklad 2: Uživatelem definovaná navigace

V případě vývoje webové aplikace, kterou se zabývá tato práce, byla pro zobrazení použita Facelets technologie, která je doporučenou prezentační technologií v rámci JSF.

3.2.2 Facelets

Facelets je přední technologií pro vytváření view v rámci JSF [3]. Tato technologie podporuje několik knihoven tagů pro přidávání komponent do stránky. Pro podporu knihoven tagů JavaServer Faces, používá Facelets XML definici jmenných prostor [3]. Facelets využívá pro přístup k proměnným a metodám Managed Bean Expression Language (EL), který je popsán níže. Pokud používáme na svých stránkách technologii Facelets je potřeba definovat **FacesServlet** v souboru web deployment descriptor, například **web.xml**, který bude zpracovávat HTML žádosti a odpovědi.

V rámci technologie Facelets je možné používat ve stránkách šablony (templates), díky těmto šablonám je možné opakovaně používat stejné zobrazení ve více stránkách, k tomu slouží knihovna tagů *http://xmlns.jcp.org/jsf/facelets*, označována ve stránkách prefixem **ui:**.

3.2.3 Expression Language (EL)

Expression Language slouží pro přístup k proměnným, či metodám přímo ze stránek [3]. Ve stránkách tento jazyk vkládáme do ${}, tuto syntaxi lze ale použít pouze u atributů tagů, které podporují runtime expressions [3] a je potřeba jejich přímé vykonání. Tato syntaxe se používá vždy pro hodnoty, které jsou pro čtení, naopak za použití syntaxe #{} může být vykonání odloženo a lze ji použít pro čtení i nastavení proměnné v Managed Bean. Expression Language má využití i v případě Resource Bundle souborech, ve kterých můžeme definovat jazykovou lokalizaci pro stránky.

3.2.4 PrimeFaces

Jedná se o open source knihovnu [5], která nabízí plno součástí využívajících JSF pro tvorbu bohatého uživatelského rozhraní. Knihovna je dostupná volně na internetu ve formě jediného souboru typu .jar. Obsahem knihovny je spousta rozšíření, jako:

- Početná sada komponent (Dialog, AutoComplete, Grafy, HtmlEditor a další)
- Vestavěný Ajax postavený na standardu JSF 2.0 Ajax API
- Skládá se pouze z jednoho souboru .jar, nulová konfigurace a žádné požadavky na závislosti
- Push podpora přes Atmosphere Framework
- Mobilní UI kit pro vytváření mobilních webových aplikací
- Skinning Framework s více než 35 vestavěnými tématy
- Rozsáhlá dokumentace
- Velká uživatelská komunita

3.3 Enterprise JavaBeans (EJB)

Enterprise JavaBeans jsou základní částí platformy Java Enterprise Edition [6]. EJB kontejner řídí životní cyklus EJB instancí, takže se vývojář nemusí starat kdy vytvořit nebo smazat EJB objekty. Primárně jsou k dispozici 3 typy Enterprise JavaBeans.

3.3.1 Session Bean

Ukládá data konkrétního uživatele pro jednu relaci (session). Tato bean může být stavová i bezstavová a nebo singleton.

3.3.1.1 Stateless Session Bean (Bezstavová session bean)

Bezstavová session bean je typem Enterprise Bean a používá se pro operace, které na sobě nejsou závislé. Nemá žádný vztah se stavem klienta, ale může zachovávat stav své instance.

Při vytváření bean je doporučeno nejdříve vytvořit remote nebo local interface, ve kterém jsou zpřístupněny business metody. Typ rozhraní je definován použitou anotací:

- **@Local**- použijeme, jeli EJB klient ve stejném prostředí jako je zavedena EJB session bean.
- **@Remote**- anotaci použijeme, jestliže je EJB klient v jiném prostředí.

Samotný typ session bean, ať už stateless nebo stateful určuje anotace umístěná nad názvem třídy, v tomto případě **@Stateless**.

```
import javax.ejb.Remote;

@Remote
public interface LibrarySessionBeanRemote {
    //add business method declarations
}
```

Příklad 3: Remote(Vzdálený) interface [6]

```
@Stateless
public class LibrarySessionBean implements LibrarySessionBeanRemote {
    //implement business method
}
```

Příklad 4: Stateless(Bezstavová) EJB [6]

3.3.1.2 Stateful Session Bean (Stavová Session Bean)

Stateful session bean je typ Enterprise Bean, která udržuje stav s klientem. Je stále spojena se stavem klienta, v jeho případě instančními proměnnými. Jakmile request (žádost) vyprší, stateful bean je zničena. Stejně jako u stateless session bean je možné vytvořit remote nebo local interface. Pro definici bean jako stateful je použita anotace **@Stateful**.

```
@Stateful
public class LibraryStatefulBean implements LibraryStatefulBeanRemote {
    //implement business method
}
```

Příklad 5: Stateful(Stavová) EJB [6]

3.3.1.3 Singleton Session Bean

Singleton session bean poskytuje podobnou funkcionalitu jako stateless session bean, ale rozdíl je v tom, že je pouze jedna singleton session bean v rámci aplikace, na rozdíl od stateless session bean, kterých může být víc, umístěných v poolu, a každá z nich může odpovídat na požadavek klienta.

3.3.2 Entity Bean

Entity Bean představují persistentní doménové objekty. Uživatelská data jsou ukládána do databáze přes entity bean a později znovu získána z databáze v podobě entitních tříd. V EJB 3.0 je entitní bean jednoduchý POJO(Plain Old Java Object), který je namapovaný na databázovou tabulku [6]. Typicky entita reprezentuje tabulku v relační databázi a jednotlivé instance odpovídají řádkům v této tabulce. Základní představitelé Java Persistent API:

- **Entita**- Objekt reprezentující tabulku v datovém úložišti.

```
@Entity
@Table(name = "books")
public class Book implements Serializable {
    private int id;
    private String name;

    public Book() {}

    //mark id as primary key with autogenerated value
    //map databse column id with id field
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    public int getId() {
        return id;
    }
}
```

Příklad 6: Entity Bean [6]

- **EntityManager**- Interface pro operace s daty na entitě.
- **Persistence unit (persistence.xml)**- Popisuje vlastnosti persistentního mechanismu.

```
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema/instance"
xsi:schemaLocation="http://java.sun.com/xml/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
    <persistence-unit name="EjbComponentPU" transaction-type="JTA">
        <jta-data-source>java:/PostgresDS</jta-data-source>
        <exclude-unlisted-classes>false</exclude-unlisted-classes>
        <properties/>
    </persistence-unit>
    <persistence-unit name="EjbComponentPU2" transaction-type="JTA">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <jta-data-source>java:/PostgresDS</jta-data-source>
        <exclude-unlisted-classes>false</exclude-unlisted-classes>
        <properties>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
        </properties>
    </persistence-unit>
</persistence>
```

Příklad 7: Persistence unit(persistence.xml) [6]

- **Data Source (*ds.xml)**- Popisuje související vlastnosti s datovým úložištěm jako URL připojení k databázi, uživatelské jméno, heslo, atd.

```
<?xml version="1.0" encoding="UTF-8"?>
<datasources>
   <local-tx-datasource>
      <jndi-name>PostgresDS</jndi-name>
      <connection-url>jdbc:postgresql://localhost:5432/postgres</connection-url>
      <driver-class>org.postgresql.driver</driver-class>
      <driver-class>org.postgresql.driver</driver-class>
      <user-name>sa</user-name>
      <password>sa</password>
      <min-pool-size>5</min-pool-size>
      <max-pool-size>20</max-pool-size>
      <idle-timeout-minutes>5</idle-timeout-minutes>
   </local-tx-datasource>
</datasources>
```

Příklad 8: DataSource [6]

3.3.3 Message Driven Bean

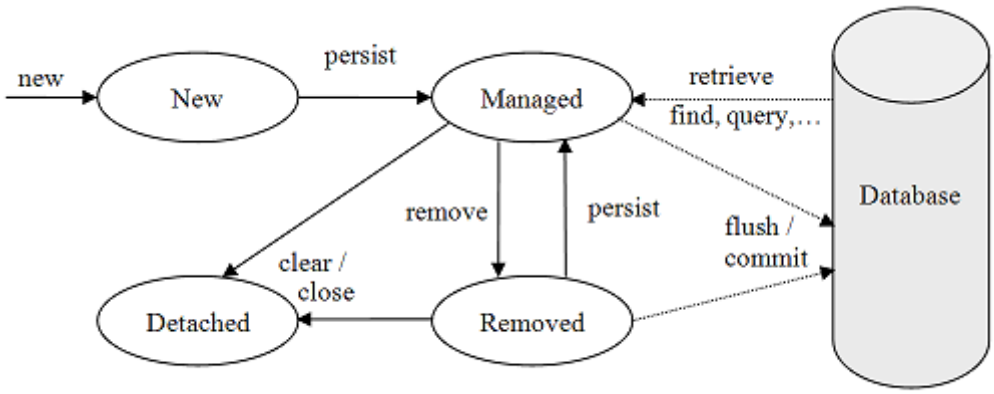
Jsou používány v souvislosti s JMS (Java Messaging Service). Message driven bean umožňuje Java EE aplikacím zpracovávat zprávy asynchronně [3].

3.4 Java Persistence API (JPA)

Java Persistence API je Framework pro programovací jazyk Java, který poskytuje objektově relační přístup pro práci s databází. Základním prvkem JPA jsou entitní třídy, které musejí splňovat některé požadavky [3]:

- Entitní třída musí být označena anotací **javax.persistence.Entity**.
- Třída musí mít public nebo protected konstruktor bez parametrů.
- Nesmí být deklarována jako final.
- Instanční proměnné musejí být deklarovány jako private, protected, nebo package-private a lze k nim přistupovat přímo pouze přes metody get a set.

Instanční proměnné takovéto třídy reprezentují jednotlivé sloupce v relační databázi a každý řádek je reprezentován jednou instancí třídy. Z toho vyplývá, že entita je objekt, reprezentující data v databázi. Každá entita má svůj životní cyklus, který se skládá ze 4 stavů: **new, managed, removed, detached**.



Obrázek 7: Životní cyklus entity [3]

Když je entitní objekt vytvořen nachází se ve stavu new a není reprezentován záznamem v databázi. Do stavu managed se entitní objekt dostane ve dvou případech, prvním z nich je, pokud je objekt uložen do databáze za použití metody persist interface (rozhraní) EntityManager [7] nebo jestliže je již uložený objekt z databáze získán. Další je stav deleted, do kterého se objekt dostane v případě, že byl získaný z databáze a označený k vymazání metodou remove v EntityManager, k fyzickému smazání objektu dojde až při vykonání příkazu commit v transakci. Poslední je detached stav, ve kterém se nacházejí všechny entitní objekty, které jsou odpojené od EntityManager, Pro příklad, všechny managed objekty se stanou detached, pokud je EntityManager uzavřen [7].

JPA poskytuje 4 druhy vztahů mezi entitami.

3.4.1 One To One (1:1)

Tento vztah slouží k vytvoření závislosti 1:1 mezi entitami, přesněji řečeno, každá instance je svázána pouze s jednou instancí jiné entity. Příklad 9 ukazuje použití anotace **@OneToOne** pro vytvoření vztahu 1:1.

```
@OneToOne
private Vehicle vehicle;
```

Příklad 9: Vytvoření vztahu One To One

3.4.2 One To Many (1:N)

Vztah One To Many se používá, pokud má instance jedné entitní třídy vazbu na více instancí jiné entity. Přesný opak vztahu One To Many je Many To One, který se vytváří zrcadlově úplně stejně. Spojení tabulek za pomoci anotace **@OneToMany** a **@ManyToOne** pro vytvoření cizího klíče je znázorněno na příkladu 10.

```
Soubor: User.java

@OneToMany(mappedBy = "user")
private List<Vehicle> vehicles;

Soubor: Vehicle.java

@ManyToOne
@JoinColumn(name = "user_id")
private User user;
```

Příklad 10: Vytvoření vztahu One To Many

3.4.3 Many To Many (M:N)

Tento vztah je nutný, pokud několik instancí jedné entity může mít vazbu na více instancí jiné entity. V obou zúčastněných modelech musejí být kolekce odkazující na druhý model. Pro vytvoření vztahu M:N se používá anotace **@ManyToMany**, kterou je nutné umístit nad dané kolekce odkazující na druhou entitu a to v případě obou entit. Příklad vytvoření takovéhoto vztahu je uveden na příkladu 11.

Soubor: Vehicle.*java*

```
@ManyToMany(mappedBy = "vehicles")
private List<User> users;
```

Soubor: User.*java*

```
@ManyToMany
@JoinTable(name = "users_vehicles",
        joinColumns = @JoinColumn(name = "user_id"),
        inverseJoinColumns = @JoinColumn(name = "vehicle_id"))
private List<Vehicle> vehicles;
```

Příklad 11: Vytvoření vztahu Many To Many

4 Implementační část

4.1 Postup vývoje

V analytické části práce byly za použití use-case diagramů navrženy možné činnosti vykonávané jednotlivými uživateli nad aplikací. V souvislosti s tímto návrhem byla navržena databázová struktura. Podrobný návrh je popsán v kapitole: 4.2 Návrh databázové struktury.

Po detailním návrhu funkčností aplikace bylo provedeno připojení k MySQL databázi běžící na lokálním stroji programátora a jednotlivé tabulky znázorněné v návrhu databázové struktury realizovány pomocí entitních tříd s využitím Frameworku JPA (Java Persistence API), následně automaticky vygenerovány tabulky v databázi.

Následovalo nastavení zabezpečení pro přístup ke zdrojům aplikace. Toto nastavení spočívá v konfiguraci Glassfish serveru a nadefinování uživatelských rolí a skupin, viz kapitola: 4.4.1 Definice uživatelů a rolí na serveru Glassfish. Po vytvoření bezpečnostní domény (Security Realm) byla tato doména připojena do projektu pomocí definice v souboru **web.xml**, viz kapitola: 4.4.2 Nastavení bezpečnosti pro přístup k souborům.

Poté přišlo na řadu vytvoření tříd potřebných pro vykonávání aplikační logiky, ze strany back-endu aplikace se jedná zejména o třídy z balíčku services, které se starají o operace prováděné nad databází, v rámci těchto tříd byly vytvořeny i potřebné databázové dotazy.

Po funkčním získávání dat z databáze byla provedena implementace tříd z balíčku controllers a všech potřebných zobrazení pro základní chod aplikace.

4.2 Návrh databázové struktury

Klíčovým prvkem databázové struktury projektu je tabulka **users**, která obsahuje data samotných uživatelů, včetně emailu a hesla, které jsou použity pro přihlašování uživatelů do aplikace, z tohoto důvodu bylo možné navrhnout strukturu databáze tak, aby byla tabulka **users** rozdělena do dvou tabulek, z níž jedna by obsahovala pouze uživatelský email a heslo pro přihlášení a další tabulka, pojmenovaná například **users_detail**, by obsahovala ostatní údaje daného uživatele jako je jméno, příjmení, atd. Tyto dvě tabulky by mezi sebou měly samozřejmě vazbu 1:1 (One To One), aby bylo zajištěno, že přihlašovací údaje z tabulky **users** patří pouze k jedné instanci v tabulce **users_detail**. Pro řešení, které je popsáno v této kapitole byla ale vytvořena pouze jedna tabulka obsahující všechna uživatelská data.

Další základní částí databázové struktury je tabulka **sports**, která v základní funkčnosti obsahuje pouze atributy id, jméno sportu a typ sportu, který slouží k tomu, aby bylo možné

zjistit, zdali je za lepší výkon považována menší hodnota, či vyšší. Tuto tabulku je možné v budoucnu rozšířit o další atributy, ale pro účel popisované aplikace je toto dostačující. Pro rozlišení, který sport používá jakou základní jednotku pro měření výkonů, je databáze rozšířena o tabulku **units**, která je vazbou 1:N (One To Many) propojena s tabulkou **sports**.

Jelikož by propojením tabulek **sports** a **users** vznikla vazba M:N (Many To Many), byla vytvořena asociační tabulka mezi těmito dvěma, její název vznikl spojením názvů obou tabulek (**users_sports**), tabulka obsahuje kromě cizích klíčů na obě zmiňované tabulky také další atributy, jako je max_score, který slouží pro záznam sportovního výkonu uživatele v daném sportu. Dalším atributem přidaným hlavně kvůli historizaci výkonů je atribut actual, který může nabývat pouze hodnot true nebo false a slouží jako jakýsi ukazatel na aktuální skóre. Další atribut scoredate je spíše informativní, aby bylo možné zjistit, k jakému datu je záznam o skóre aktuální.

Výše zmíněnou částí struktury je popsán návrh navázání jednotlivých uživatelů na konkrétní sporty, včetně připojení jejich aktuálního skóre skrze asociační tabulku. V další části jsou popsány ostatní části databázového modelu, které jsou nezbytné pro další funkčnosti aplikace.

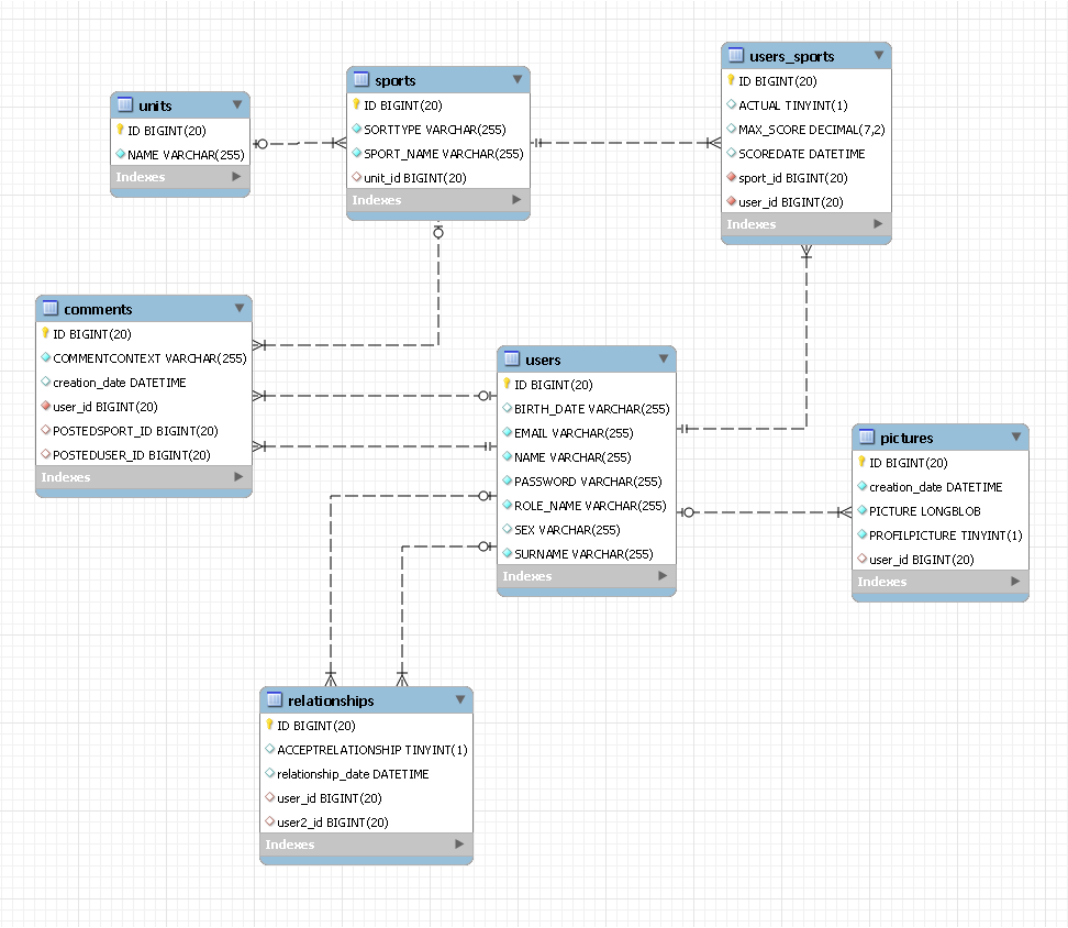
Jednou z nich je tabulka **comments**, která je přímo navázaná na tabulku **users** ve vazbě N:1 (Many To One) a slouží k ukládání jednotlivých komentářů, jak na profilech uživatelů, tak na stránkách sportů, k rozeznání, na jakém profilu jsou komentáře ukládány, slouží cizí klíče na tabulky **sports** a znovu **users**, oba zmiňované vztahy ve vazbě 1:1 a jedná se o nepovinné atributy.

Další důležitou částí databáze je tabulka **relationships**, sloužící pro ukládání vztahů mezi jednotlivými uživateli, obsahuje 5 atributů, prvním z nich je id, zabezpečující unikátnost záznamů, dalším atributem je cizí klíč na tabulku **users**, který slouží k uložení uživatele zakládajícího vztah, druhý cizí klíč na tabulku **users** obsahuje záznam se kterým uživatelem je vztah navázán. Pro identifikaci, zdali je vytvořený záznam v tabulce **relationships** schválen oběma uživateli, tabulka obsahuje atribut acceptRelationship nabývající pouze hodnot true nebo false, kde hodnota true u tohoto atributu je považována za známku schválení vytvořeného vztahu. Posledním atributem je datum vytvoření vztahu, který ale není pro naše účely tak důležitý.

Jako posledním prvkem struktury je tabulka **pictures**, sloužící k ukládání profilových fotek jednotlivých uživatelů, z tohoto důvodu obsahuje cizí klíč na centrální tabulku **users** ve

vazbě N:1, v popisované aplikaci je uživateli povoleno přidání pouze jedné profilové fotky, tudíž by zde stačila vazba 1:1, ale z hlediska další možnosti rozšiřitelnosti, je přidána vazba N:1, umožňující uživateli přidání více fotografií. S tímto souvisí další pravdivostní atribut, který nabývá hodnoty true, pokud je daný obrázek brán jako profilový. Samotný obrázek je ukládán do databáze ve formě pole jednotlivých bytů a kvůli velikosti tohoto pole je nutno, aby tento atribut byl v databázi typu LONGBLOB.

Na obrázku 8 je ve formě ER diagramu znázorněna celá popisovaná databázová struktura. Návrh byl proveden v nástroji MySQL Workbench 6.2 [8].



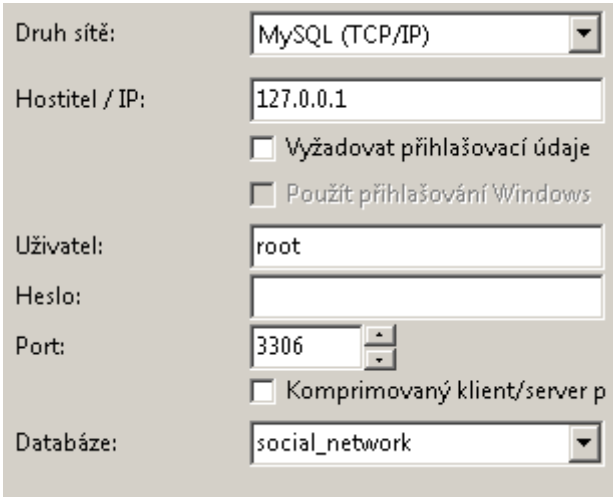
Obrázek 8: ER diagram databázové struktury

4.3 Implementace databázové struktury pomocí JPA

Po navržení databázové struktury byl tento návrh naimplementován pomocí Frameworku JPA, který využívá takzvané Entity Beans, což jsou klasické Java třídy, splňující některé náležitosti, jak již bylo popsáno v kapitole: 3.3.2 Entity Bean.

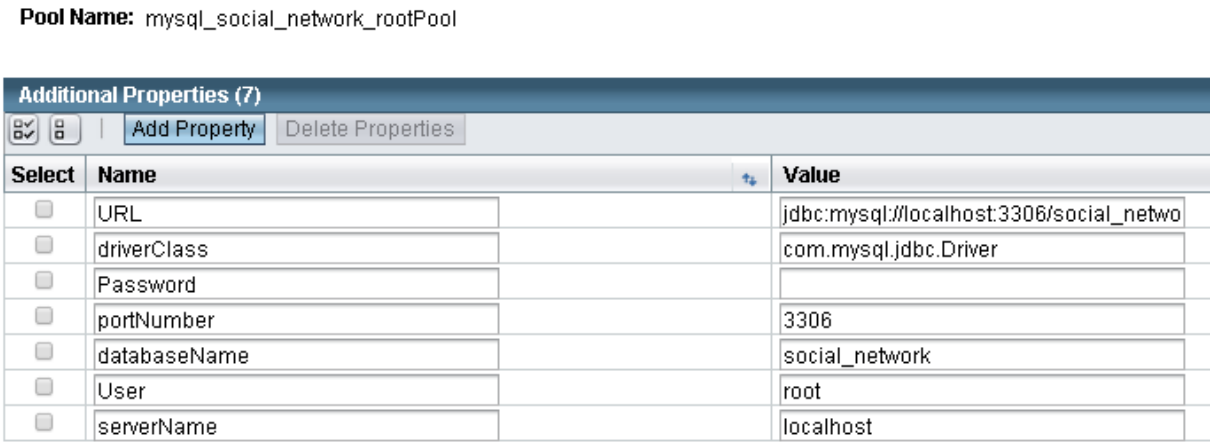
4.3.1 Napojení na MySQL databázi

V rámci projektu byla vytvořena MySQL databáze, pojmenovaná **social_network**, běžící na předem nainstalovaném MySQL serveru. Konfigurace databáze zobrazena na obrázku 9.



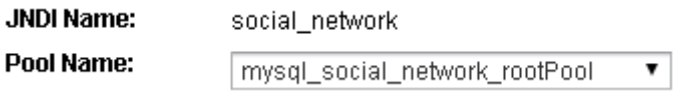
Obrázek 9: Konfigurace MySQL databáze

Aplikaci bylo potřeba připojit k vytvořené databázi. Z tohoto důvodu byl vytvořen **connection pool**. Tento pool byl nastaven v administrativní konzoli aplikačního serveru Glassfish pod záložkou **Resources -> JDBC -> JDBC Connection Pools**. Vlastnosti vytvořeného poolu musejí odpovídat konfiguraci MySQL databáze, jak je zobrazeno na obrázku 10.



Obrázek 10: Nastavení Connection poll na serveru GlassFish

Dalším krokem byla definice **JDBC Resource**, kde je nutno nastavit **JNDI Name**, tzn. název, pod kterým bude nakonfigurovaný pool přístupný, jednoduchá konfigurace zobrazena na obrázku 11.



Obrázek 11: Nastavení JDBC Resource

Všechno definované nastavení lze také vidět a editovat v souboru **glassfish-resources.xml** umístěného v adresáři **Server Resources** ve složce projektu.

Po nastavení připojení k vytvořené databázi bylo potřeba založit tzv. persistence unit, jejíž konfigurace je zaznamenána v souboru **persistence.xml**, umístěného v adresáři **Configuration Files** ve složce projektu, soubor může obsahovat jednu či více persistence unit, v případě popisované aplikace byla použita jedna persistence unit, konfigurace znázorněna na příkladu 12.

```
<persistence-unit name="social_networkPU" transaction-type="JTA">
  <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
  <jta-data-source>social_network</jta-data-source>
  <exclude-unlisted-classes>false</exclude-unlisted-classes>
  <properties>
    <property name="eclipselink.ddl-generation" value="create-tables"/>
  </properties>
</persistence-unit>
```

Příklad 12: : Konfigurace persistence unit v souboru persistence.xml

4.3.2 Implementace Entity Beans

V projektu bylo vytvořeno 7 entitních tříd, jmenovitě třídy: Comment, Picture, Relationship, Sport, Unit, User, UsersSports, každá tato třída reprezentuje jednu tabulku v databázi. Pro ukázku implementace Entity Beans byla vybrána třída User, která reprezentuje tabulku users v databázi. Částečná implementace třídy User zobrazena na příkladu 13.

```java
@Entity
@Table(name = "users")
public class User implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(nullable = false)
    private Long id;
    @Column(nullable = false)
    private String email;
    @Column(nullable = false)
    private String password;
```

Příklad 13: Entita User

V projektu byly při vytváření entitních tříd použity anotace popsané v kapitole: 4.3.3 Použité anotace. Takto vytvořená entitní třída splňující dané náležitosti a označena potřebnými anotacemi je Frameworkem JPA namapována do databáze jako klasická tabulka obsahující primární klíč a všechny nadefinované atributy, výsledkem výše uvedené implementace entitní třídy User je vytvoření v databázi tabulky se všemi atributy, jak je zobrazeno na obrázku 12.

1	ID	BIGINT	20	☐	☐
2	BIRTH_DATE	VARCHAR	255	☐	☑
3	EMAIL	VARCHAR	255	☐	☐
4	NAME	VARCHAR	255	☐	☐
5	PASSWORD	VARCHAR	255	☐	☐
6	ROLE_NAME	VARCHAR	255	☐	☐
7	SEX	VARCHAR	255	☐	☑
8	SURNAME	VARCHAR	255	☐	☐

Obrázek 12: Tabulka users vytvořená Frameworkem JPA z entitní třídy User

4.3.3 Použité anotace

Knihovna **javax.persistence** poskytuje několik anotací, které je možné použít při vytváření entitních tříd, níže jsou popsány jen ty z nich, které byly použity v popisované aplikaci [3].

@Entity- indikuje, že třída, nad kterou je tato anotace použita slouží jako persistentní entita

@NamedQueries- použijeme, pokud potřebujeme využít více pojmenovaných dotazů

@NamedQuery- slouží k definici jednoho konkrétního jmenného dotazu nad entitní třídou

@Table- slouží ke specifikaci detailů databázové tabulky, např. **@Table(name=”users”)**

@Id- anotace používaná pro definici primárního klíče

@GeneratedValue- slouží k výběru strategie pro generování primárního klíče

@Column- slouží ke specifikaci detailů u daného sloupce tabulky, např. **@Column(nullable=false)**

@OneToMany- slouží k vytvoření vztahu 1:N

@ManyToOne- slouží k vytvoření vztahu N:1

@OneToOne- slouží k vytvoření vztahu 1:1

@JoinColumn- specifikuje vlastnosti sloupce vytvořeného založením vztahu mezi entitami

4.4 Zabezpečení

V rámci volby vhodného typu zabezpečení byla brána zřetel na data, se kterými aplikace pracuje a jako vhodné řešení bylo zvoleno použití vlastností serveru Glassfish a vytvořena bezpečností doména.

Nevýhodou této volby je zejména přenositelnost na jiný aplikační server, neboť by bylo potřeba provést na patřičném serveru znovu konfiguraci bezpečnostní domény, ale jelikož byla aplikace vyvíjena pouze na jednom stroji, nebyl tento problém nijak omezující.

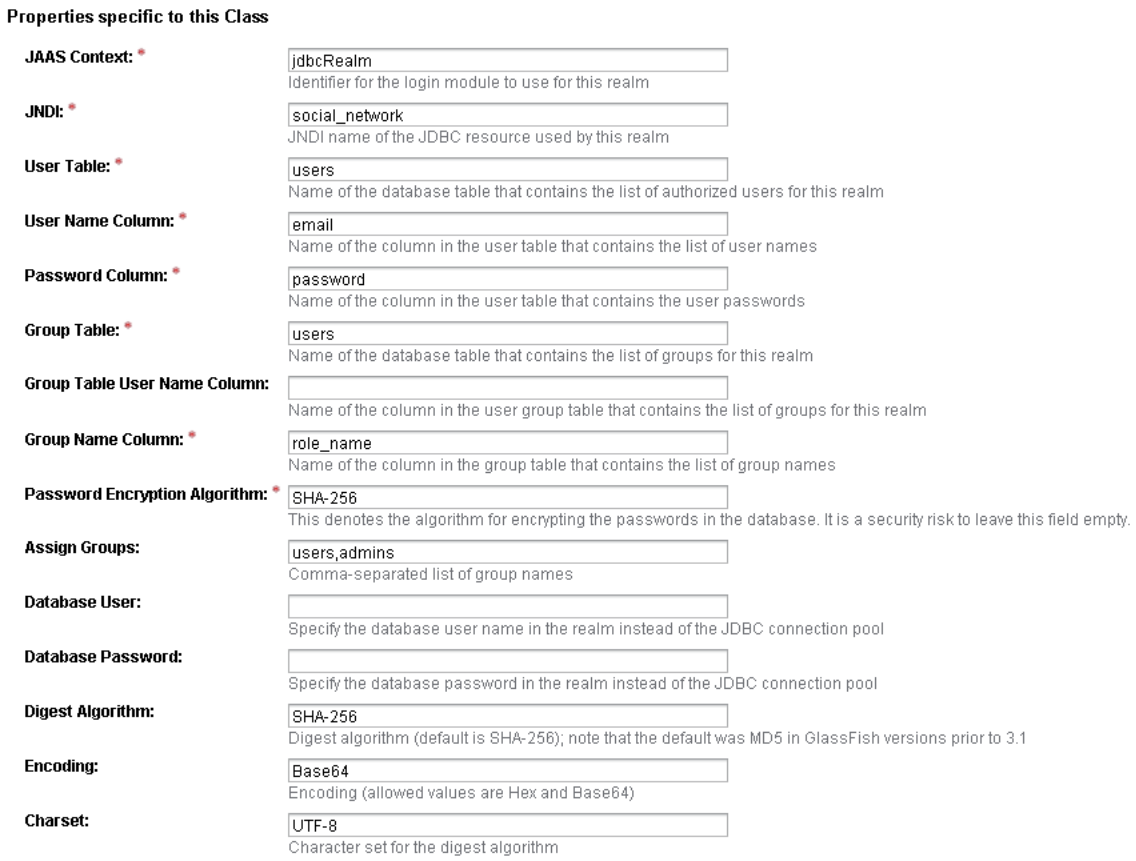
Pro samotné přihlašování je použit uživatelský email a heslo, kde pro přenos autentizačních údajů a obecně všech dotazů na server je využit pouze protokol http, který nepodporuje žádné šifrování, a tudíž jsou data přenášena v čitelné podobě. Jelikož aplikace neobsahuje citlivá data, která by byla náchylná na odcizení, je tento typ zabezpečení dostačující.

Byla zvažována i možnost použití protokolu https, ale pro účely vytvářené aplikace, jakožto prototypu, byl http protokol dostačující.

4.4.1 Definice uživatelů a rolí na serveru Glassfish

Pro funkční přihlašování uživatelů bylo nutné provést konfiguraci bezpečnostních domén a uživatelských rolí na serveru Glassfish. Prvním krokem bylo spustit administrativní konzoli serveru, v případě popisované aplikace byl aplikační server zprovozněn na adrese localhost za použití portu 4848, to znamená, že konzole Glassfish serveru je dostupná na adrese: *http://localhost:4848*, kde je možné provádět různá nastavení serveru.

Pro nastavení rolí byla vybrána položka **Security** a vytvořen nový **Realm** s nastavením povinných atributů, viz obrázek 13.



Obrázek 13: Nastavení Security realmu

V rámci projektu byly vytvořeny dvě skupiny rolí: users, admins. S těmito rolemi se v projektu dále pracuje, viz 4.4.2 Nastavení bezpečnosti pro přístup k souborům.

4.4.2 Nastavení bezpečnosti pro přístup k souborům

Upřesnění nastavení bezpečnosti se provádí v souboru **web.xml**. Lze definovat, které role, definované v rámci vytváření bezpečnostní domény (Security Realm), budou mít přístup, k jakým souborům v adresářové struktuře projektu. V rámci projektu se pracuje s rolemi users a admins, to znamená, pokud přihlášený uživatel má přiřazenou jednu z těchto rolí, je oprávněn přistupovat k souborům umístěným v adresáři **authenticated_users**. Definice přístupových práv v souboru ***web.xml*** je znázorněna na příkladu 14.

```
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Secure Pages</web-resource-name>
        <url-pattern>/faces/authenticated_users/*</url-pattern>
        <http-method>GET</http-method>
        <http-method>POST</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>USERS</role-name>
        <role-name>ADMINS</role-name>
    </auth-constraint>
</security-constraint>
<security-role>
    <role-name>USERS</role-name>
</security-role>
<security-role>
    <role-name>ADMINS</role-name>
</security-role>
```

Příklad 14: Definice Security Realm ve web.xml

Jak je vidět na obrázku nejprve je nadefinována adresa, která bude přístupná pouze s určitými právy, hvězdička za konečným lomítkem v adrese znamená, že budou zpřístupněny všechny podřazené soubory a adresáře.

Dalším důležitým prvkem při definici bezpečnostních omezení je definování rolí, které mají k dané adrese přístup, v případě aplikace se jedná o role users a admins.

4.4.3 Specifikace autentizačního mechanismu

V popisované aplikaci je použita form autentizace, která je jednou z řady možností, které platforma Java EE poskytuje pro bezpečné přihlašování uživatelů. Konfigurace tohoto typu autentizace se provádí v souboru **web.xml**. Nejprve je nutné v xml tagu **<auth-method>** definovat vybraný typ autentizačního mechanismu, v našem případě se jedná o form autentizaci, dále se v tagu **<realm-name>** udává název bezpečnostní domény, která se má při přihlašování použít, více o nastavení bezpečnostní domény, viz kapitola: 4.4.1 Definice uživatelů a rolí na serveru Glassfish. Dalším potřebným krokem je definice stránky, která se má zobrazit, pokud bude uživatel požadovat přístup ke stránce, která vyžaduje přihlášení. Tento krok se provádí uvnitř tagu **<form-login-page>**, v případě implementované aplikace byla pro tento účel vytvořena stránka **login.xhtml**, která obsahuje formulář pro přihlašování do aplikace. Tento formulář musí ještě splňovat některá další kritéria pro využití této metody autentizace, viz kapitola: 4.4.4 Náležitosti přihlašovacího formuláře. Stejně jako je definovaná stránka pro přihlašování do aplikace, je potřeba definovat stránku, která je

uživateli zobrazena při neúspěšném přihlášení. Tato stránka je definována v rámci tagu **<form-error-page>**, pro tento účel byla v projektu vytvořena stránka **error.xhtml**.

Příklad konfigurace form autentizace v aplikaci je znázorněn na příkladu 15.

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>jdbcRealm</realm-name>
    <form-login-config>
        <form-login-page>/faces/login.xhtml</form-login-page>
        <form-error-page>/faces/error.xhtml</form-error-page>
    </form-login-config>
</login-config>
```

Příklad 15: Konfigurace FORM autentizace

4.4.4 Náležitosti přihlašovacího formuláře

Pouhé definování form autentizace v souboru web.xml nestačí, dalším důležitým krokem je splnit náležitosti při vytváření samotného formuláře ve stránce. První věcí je, že formulář musí být prostý html **<form>** tag a jako hodnotu atributu „action“ je nutné zadat metodu **j_security_check**, která je definovaná v Java EE pro přihlašovací formuláře. Dalším krokem je nutnost pojmenovat vstupní pole pro přihlašovací jméno jako **j_username** a heslo jako **j_password**.

Pokud jsou splněny tyto náležitosti přihlašovacího formuláře a je korektně provedeno definování autentizačního mechanismu a bezpečnostní domény, je možné provádět bezpečnou autentizaci uživatelů před vstupem ke zdrojům v aplikaci.

4.5 Vytvoření dotazů do databáze

Pro vytváření dotazů do databáze poskytuje Framework JPA několik způsobů: použití nativního SQL dotazovacího jazyka, využití JPQL (Java Persistence Query Language), vytvoření jmenných dotazů, anebo použití Criteria Query API. Pro demonstraci byly v popisované aplikaci použity jak pojmenované dotazy (NamedQuery), tak využití čistého SQL dotazovacího jazyka za využití funkce **createNativeQuery()**.

4.5.1 Pojmenované SQL dotazy

Pro vytvoření jmenného dotazu je potřeba v doménové třídě použít anotaci **@NamedQuery** a vytvořit konkrétní dotaz do databáze, viz příklad 16, kde je použit SQL SELECT všech uživatelů, kteří mají atribut surname shodný s hodnotou vstupního parametru dotazu.

```
@NamedQuery(name="getUser.BySurname",
    query="SELECT u FROM User u WHERE u.surname LIKE ?1")
```

Příklad 16: Vytvoření pojmenovaného dotazu

Tato anotace má dva povinné parametry, kde prvním z nich je název dotazu (name), pomocí tohoto názvu se dotaz později volá a jelikož je rozsah pojmenovaných dotazů platný v rámci celé persistence unit, je potřeba volit název pečlivě, aby nedocházelo ke kolizím. Druhým parametrem je samotný dotaz (query). V popisované aplikace jsou v drtivé většině použity právě jmenné dotazy, hlavně kvůli svojí jednoduchosti a přehlednosti.

4.5.2 Native SQL dotazy

Pro vytvoření nativního SQL dotazu se v JPA využívá metoda **createNativeQuery()**, která má dva parametry, prvním je java.lang.String hodnota samotného dotazu a druhým entitní třída, na kterou je dotaz uplatňován. Definice nativního SQL dotazu je ukázána na příkladu 17.

```
Query query = em.createNativeQuery("Select * from comments", Comment.class);
```

Příklad 17: Vytvoření nativního SQL dotazu

4.6 Implementace Enterprise JavaBeans pro práci s databází

Pro práci s databází byl v projektu pro přehlednost vytvořen balíček s názvem **services**, obsahující všechny EJB třídy pracující s databází. Jmenovitě se jedná o třídy: CommentService, PictureService, RegistrationService, RelationshipService, SportService, UserService, UsersSportsService. Každá tato třída se stará o databázové úkony převážně jedné doménové třídy, například UserService provádí databázové operace nad entitou User. Základem práce s databází je vytvoření instance třídy EntityManager.

4.6.1 Práce s Entity Manager

EntityManager je klíčovým prvkem pro práci s databází, proto je v každé třídě z balíčku **services** nejprve vytvořena privátní proměnná EntityManager a označena anotací **@PersistenceContext**, tato anotace slouží k tomu, aby kontejner za běhu programu vytvořil instanci EntityManager a použil k tomu informace obsažené v souboru **persistence.xml** nebo jiné námi definované v parametrech anotace. V případě popisované aplikace je použita anotace bez parametrů a pro přístup k databázi se používají informace ze souboru **persistence.xml**. Definice EntityManager je ukázána na příkladu 18.

```
@PersistenceContext
private EntityManager em;
```

Příklad 18: Deklarace EntityManager

Všechny operace create, update a delete je vyžadováno provádět uvnitř transakce, v případě aplikace, která je zde popisována, bylo použito JavaTransaction API a EntityManager se stará o provádění transakce sám. Samozřejmě je možné obsluhovat transakce manuálně použitím instance třídy UserTransaction a jejích metod begin(), která vytvoří transakci, poté uživatel provede nějakou databázovou operaci a správnost změn je ověřena metodou commit(), která když není úspěšně provedena, provede se v databázi rollback a všechny prováděné změny jsou z databáze odstraněny.

Získávání dat z databáze za použití EntityManager je v aplikaci prováděno pomocí metod createNamedQuery() a createNativeQuery(), jak již bylo popsáno v kapitole: 4.5 Vytvoření dotazů do databáze. Na příkladu 19 je znázorněno použití jmenného dotazu za využití metody createNamedQuery().

```
User user = (User)em.createNamedQuery("getUser.ByEmail")
        .setParameter(1, userEmail).getSingleResult();
```

Příklad 19: Použití jmenného dotazu

Parametr metody je název již vytvořeného jmenného dotazu v entitní třídě, jelikož dotaz má definovaný vstupní parametr, je nutno použít metodu setParameter() a přidat požadovanou hodnotu parametru. A protože chceme získat jen jeden záznam z databáze, použijeme ještě metodu getSingleResult(), která vrátí jen jeden záznam.

Třída EntityManager neslouží pouze k získávání dat z databáze, ale i k operacím jako je create, update a delete. V rámci aplikace bylo využito několik těchto metod, například pro uložení právě registrovaného uživatele je zavolána metoda saveUser() ve třídě UserService, která vytvoří novou instanci třídy User, pomocí metod pro nastavení jejích privátních proměnných, metody s prefixem set, se provede nastavení atributů u nově vytvořeného objektu. Pro uložení nového objektu do databáze se na objektu třídy EntityManager zavolá metoda persist(), po které se objekt ze stavu **new** přesune do stavu **managed**, viz kapitola: 3.4 Java Persistence API (JPA). V tomto stavu už jsou objekty zpracovávány kontejnerem a ten se postará o uložení jejich aktuálního stavu do databáze. Jelikož může během operace dojít k výjimce, je třeba tyto metody umisťovat do bloku try-catch, jak je znázorněno na příkladu 20.

```
User user = new User();
user.setEmail(email);
user.setPassword(password);
 //some other required attributes
try {
    em.persist(user);
} catch(Exception e) {
    e.printStackTrace();
}
```

Příklad 20: Použití metody persist()

Stejným způsobem jsou v projekty prováděny i další databázové operace, jako je update a delete, jen za použití rozdílných metod EntityManager, pro update stávajícího objektu v databázi se používá metoda merge() a naopak pro delete metoda remove().

4.6.2 Použité typy Enterprise JavaBeans

I když existuje více typů Enterprise JavaBeans, viz kapitola: 3.3 Enterprise JavaBeans, pro implementaci tříd v balíčku services v projektu bylo využito pouze typu jednoho a to Stateless Session Bean, který je pro vykonávané operace dostačující.

4.7 Implementace kontrolerů ve formě Managed Beans

Stejně jako tomu bylo u balíčku services, tak i při tvorbě kontrolerů, byl v projektu vytvořen balíček controllers, který obsahuje jen Java třídy ve formě Managed Beans, které se starají o zpracování dat pro jednotlivá view (zobrazení). Jak již bylo napsáno, hlavní funkcí těchto tříd je zpracovat parametry requestů (žádostí), připravit data pro zobrazení ve view a zajistit přesměrování mezi stránkami.

4.7.1 Náležitosti Managed Bean

Managed Beans jsou klasické Java beans, tím že je tato třída registrována s JSF, stává se z ní Managed Bean. Managed beans jsou použity jako model pro UI komponenty, to znamená, že připravují data pro jednotlivá view. Registrace Java bean s JSF se při použití JSF 2 dá uskutečnit dvěma způsoby, prvním z nich, je použití konfiguračního souboru **faces-config.xml** a definování názvu Managed Bean a rozsahu platnosti v rámci tohoto souboru. Od verze JSF 2 je možné použít pro registraci anotace, což je značná výhoda hlavně z hlediska přehlednosti, neboť se anotace používají přímo uvnitř třídy [10]. Použité anotace:

- **@Named**- definuje jméno používané pro volání Managed Bean z komponenty

- **@SessionScoped, @RequestScoped, @NoneScoped, @ViewScoped, @ApplicationScoped, @CustomScoped**- použije se anotace podle potřeby rozsahu platnosti
- **@EJB**- používá se k dependenci injection jiné Enterprise JavaBean

Managed Bean má všechny atributy nastavená jako private a jednotlivé komponenty k nim přistupují přes metody s prefixem get a set.

4.7.2 Přesměrování mezi stránkami

Pro přesměrování mezi jednotlivými stránkami byly v aplikaci využity dva druhy navigace, které jsou k dispozici v rámci navigačního modelu, viz kapitola: 3.2.1.6 Navigační model. Pro využití uživatelem definované navigace byla nastavena pravidla pro přístup k jednotlivým zobrazením. Tato pravidla jsou definovaná v souboru **faces-config.xml** umístěného v adresáři **Configuration Files**.

```
<navigation-rule>
    <from-view-id>authenticated_users/*</from-view-id>
    <navigation-case>
        <from-outcome>login</from-outcome>
        <to-view-id>login.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
</navigation-rule>
```

Příklad 21: Navigační pravidlo definované uživatelem

Jednoduché navigační pravidlo znázorněné na příkladu 21 ukazuje uživatelem definovanou navigaci, kdy navigační handler provede přesměrování na stránku **login.xhtml**, pokud akce vyvolaná, například stiskem obyčejného tlačítka, má jako návratovou hodnotu řetězec „login“ a jestliže je akce vyvolána ze stránky umístěné v adresáři **authenticated_users**. Jednoduché použití tohoto pravidla ve třídě UserController je ukázáno na příkladu 22.

```
public String logout(){
    userService.logout();

    return "login";
}
```

Příklad 22: : Využití uživatelem definované navigace v kontroleru

Uživatel nemusí definovat žádná svá pravidla, pokud to není třeba, v tomto případě je využita implicitní navigace, tedy navigační handler provede přesměrování na stránku podle hodnoty uvedené v atributu „action“ u komponenty, jednoduchý příklad je uveden na

příkladu 23, kde je v atributu action vložen řetězec „user_profil“, tudíž při kliku dojde k přesměrování na view **user_profile.xhtml**.

```
<h:commandLink value="#{msg['button.profil']}" action="user_profil"
               styleClass="button" style="float: left;"/>
```

Příklad 23: Implicitní navigace

4.8 Vytvoření zobrazení (View)

4.8.1 Použité technologie

Při vytváření jednotlivých view v aplikaci bylo použito technologie JavaServer Faces, která striktně odděluje aplikační logiku od vytváření jednotlivých zobrazení, jež jsou postavena na využití xhtml, které je podporováno technologií Facelets. Facelets technologie poskytuje také možnost používání šablon ve stránkách a další nadstavby.

Pro vytváření zobrazení byly použity tagy z různých knihoven, včetně tagů z přidaného webového Frameworku Primefaces, který obsahuje některé komponenty, jak esteticky přitažlivější, tak s přidanou funkcionalitou.

4.8.2 Využité komponenty

Do aplikace byly zasazeny různé komponenty, od obyčejných tlačítek, které jsou reprezentovány tagy **<h:commandButton>**, po komplexnější komponenty, jako například pro zobrazení obrázků **<p:graphicImage>** z knihovny Primefaces. Ostatní použité komponenty je možné si prohlédnout ve zdrojovém kódu.

4.8.3 Použití šablon (templates)

V projektu byl vytvořen adresář **Web Pages/WEB-INF/templates**, který obsahuje šablony použité v aplikaci. Protože je v aplikaci definované menu, které je potřeba zobrazovat ve více zobrazeních, nabízí se možnost použití šablony. Z tohoto důvodu je v projektu vytvořen soubor **profiles.xhtml**, který v hlavičce importuje všechny potřebné soubory, jako jsou css styly a javascriptové soubory, dále v těle dokumentu definuje celou strukturu menu a popup dialogů používaných v rámci všech view využívajících tuto šablonu. Pro definici prostoru pro vložení obsahu jednotlivých view používá šablona tag ***<ui:insert>*** z knihovny ***http://java.sun.com/jsf/facelets***. Použití je zobrazeno na příkladu 24.

```
<ui:insert name="body" />
```

Příklad 24: Definice prostoru pro vložení obsahu do template

Samotná stránka vkládaná do šablony je implementována v rámci tagu <***ui:composition***> a uvádí název použité šablony, viz příklad 25.

```
template="/WEB-INF/templates/profiles.xhtml"
```

Příklad 25: : Použití template

Jakmile je ve stránce použita šablona, je nutné, aby byly definovány všechny její části označené tagem <***ui:insert***>. V našem případě, pokud chceme nějaký obsah vložit do šablony do části body, použijeme tag <***ui:define***> viz příklad 26.

```
<ui:define name="body">
```

Příklad 26: Tag pro vložení obsahu do template

4.8.4 Použití Resouce Bundles

V aplikacích využívajících platformu Java EE je možné použít, tzv. Resource Bundles, což jsou soubory využívající se převážně k lokalizaci aplikace do uživatelem zvoleného jazyka.

V případě popisované aplikace není použití lokalizovaných stránek nutné, neboť je použit jen český jazyk, nicméně kvůli přehlednosti v kódu a znovu použitelnosti některých názvů, například tlačítek, byly do projektu také přidány. Za tímto účelem, byl v projektu vytvořen balíček nazvaný **i18n**, který obsahuje soubor **messages.properties**, což značí, že se jedná o soubor používaný jako Resource Bundle.

Takovýto soubor je nutné ještě definovat v konfiguračním souboru faces-config.xml, aby mohl být použit ve stránkách. Definice souboru messages.properties použitá v projektu, je ukázána na příkladu 27.

```
<application>
      <resource-bundle>
            <base-name>i18n.messages</base-name>
            <var>msg</var>
       </resource-bundle>
</application>
```

Příklad 27: Definice Resource Bundle

Jak je vidět na obrázku, pro definici Resource Bundle je nutné zadat cestu k souboru s koncovkou **.properties** a proměnou, která bude využita pro volání ze stránek. Samotný soubor obsahuje unikátní klíčky, které odpovídají řetězcové hodnotě, která bude zobrazena.

Volání konkrétní hodnoty potřebné na stránce se provede za použití Expression Language , viz příklad 28.

```
<h:outputText value="#{msg['user.email']}: "
              class="outputText-medium black" />
```

Příklad 28: Použití Resource Bundle v komponentě

4.8.5 Import souborů css stylů a javascript

Pro importování souborů obsahujících, ať už css styly nebo javascriptové funkce existují v Java EE tagy, které se o to postarají, konkrétně pro import css stylů byl v aplikaci použit tag **<h:outputStylesheet>** a pro javascriptové soubory tag **<h:outputscript>** nebo je možné použít přímo v konkrétní stránce tag **<script></script>** uvnitř kterého lze definovat rovnou kód, který má být vykonán.

4.9 Doplňující implementace

4.9.1 Upload profilových fotek

V rámci aplikace byl naimplementován upload fotografií, které jsou využité jako profilové obrázky identifikující jednotlivé uživatele aplikace. Fotografie jsou ukládány do databáze, hlavně kvůli tomu je omezena velikost vkládané fotografie na 1MB, neboť MySQL databáze v defaultním nastavení podporuje maximální velikost paketu právě 1MB, pro účely aplikace je ale tato velikost dostačující, protože se jedná jen o vkládání profilových obrázků.

Při registraci je uživateli na profil uložen výchozí obrázek, který je vybrán podle pohlaví zvoleného při registraci, tento obrázek je možné ihned po úspěšném přihlášení změnit.

Vkládaná fotografie je nejprve převedena do byte pole (**byte[]**), takto převedený obrázek lze už jednoduše uložit do databáze. Je však nutné nad proměnou entitní třídy, která se bude mapovat do databáze jako atribut obsahující bytové záznamy jednotlivých obrázků, umístit anotaci **@Lob**, která zajistí, že v databázi bude pro uložení hodnoty vybrán největší možný datový typ (**longblob**).

4.9.2 Validace

Aby byla zajištěna správnost dat zadávaných do formulářů, zejména při registraci uživatelů, je potřeba u jednotlivých vstupních polí provádět validaci zadávaných dat. Java EE k tomuto účelu poskytuje několik možností, v aplikaci je využito rozhraní **javax.faces.validator.Validator** a přidány třídy implementující toto rozhraní. Rozhraní Validator definuje metodu validate, která je použita pro validaci vstupních dat.

Aby bylo možné volat již vytvořené validátory u jednotlivých vstupních polí, je použita anotace **@FacesValidator**, pomocí které se nastaví jedinečné jméno validátoru a tento název je následně použit při volání z JSF stránky za použití tagu **<f:validator/>** a v atributu tagu **validatorId** je uveden název konkrétního validátoru, který má být pro dané vstupní pole použit. Příklad takovéhoto volání validátoru je zobrazen na příkladu 29.

```
<h:inputSecret id="pwd1" value="#{registrationBean.password}"
               label="Password 1" styleClass="textbox" >
    <f:validator validatorId="passwordValidator" />
</h:inputSecret>
```

Příklad 29: Použití validátoru

4.10 Testování

4.10.1 Testování programátorem

Realizovaná aplikace byla v průběhu implementační fáze vývoje postupně testována samotným programátorem, jedná se o nejzákladnější testování, kdy programátor jednoduše ověří, jestli programový výstup odpovídá jeho implementované myšlence.

Z tohoto hlediska byl test programátorem proveden úspěšně a nebyly zjištěny žádné problémy.

4.10.2 Unit testy

Do projektu byly přidány i některé unit testy, ověřující, zda jednotlivé metody opravdu provádějí požadovanou akci správně a jestli se metoda chová správně při různých hraničních vstupních hodnotách.

Při testování pomocí unit testů byl použit testovací Framework JUnit ve verzi 4.11.

4.10.3 Testování uživateli

Aplikace byla poskytnuta několika uživatelům, aby zhodnotili funkcionalitu a upozornili na případné nedostatky z hlediska estetické stránky. Zpětná vazba, kterou uživatelé poskytli, byla přínosem pro odhalení drobných nesrovnalostí a ty mohly být následně opraveny.

4.10.4 Zátěžové testy

Zátěžové testy ověřují fungování aplikace v závislosti na zátěži, kterou bude vytvářet rostoucí počet uživatelů. Toto testování nebylo prováděno, neboť cílem práce bylo vytvořit prototyp aplikace k ověření funkcionality.

5 Závěr

Dle stanovených cílů práce bylo provedeno:

1. Návrh funkcionality aplikace pomocí use-case diagramů a příslušných specifikací, které byly následně využity při implementaci aplikace.
2. V souvislosti s navrženou funkcionalitou byly vybrány vhodné implementační technologie platformy Java EE, včetně databázového serveru a vývojového prostředí.
3. Následně byla provedena implementace navržené aplikace za použití vybraných technologií. Aplikace byla dopracována do stavu, kdy může sloužit jako prototyp k ověření způsobu a rozsahu nabízené funkcionality a získání zpětné vazby od potenciálních uživatelů užitečné pro eventuální další rozvoj aplikace.
4. Vybrané implementační technologie se v rámci vývoje ukázaly jako velmi dobře použitelné. Jejich vlastnosti a způsob využití jsou popsány v předchozích kapitolách. Platforma Java EE je však určena především pro vývoj rozsáhlých robustních aplikací. Řada standardizovaných rozhraní a nástrojů umožňuje realizovat vývojové práce souběžně specializovanými programátorskými týmy. Tzn. skupina programátorů se zabývá např. vývojem databázové vrstvy aplikace, jiná zase prezentační vrstvou nebo aplikační logikou. Tuto vlastnost nebylo možno využít v rámci této práce, neboť aplikace byla vyvíjena pouze jedním programátorem, bude však nespornou výhodou při eventuálním dalším rozvoji aplikace. Další výhodou zvolené platformy je tzv. škálovatelnost, jinak řečeno jednotlivé části navržené architektury aplikace je možné umístit na různé po síti spolupracující stroje a řešit tím např. zvyšující se zátěž aplikace, oddělenou správu atd.. Tato vlastnost v rámci práce, kdy všechny komponenty aplikace jsou umístěny na jednom stroji, opět nebyla využita. Její použití má smysl při zavedení aplikace do rutinního provozu.

Nastudování technologií použitých při vývoji aplikace přineslo autorovi mnoho cenných zkušeností, které je teď schopen uplatnit v praxi.

5.1 Možnosti dalšího rozvoje aplikace

Prototyp aplikace splňuje vytipované základní funkcionality, které by bylo možné po širším odzkoušení a získání zpětné odezvy od uživatelů upravit resp. rozšířit o další..

Nabízí se také možnost zvýšit bezpečnost aplikace a to komplexnějším řešením, které platforma Java EE poskytuje pro omezení přístupu k jednotlivým metodám aplikace pomocí anotací nebo např. možnost využití https komunikace, atd.

V další verzi aplikace by bylo taktéž vhodné vytvořit např. administrativní rozhraní aplikace.

6 Literatura

[1] Number of worldwide social network users. [online]. [cit. 2014-10-22]. Dostupné z: http://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/

[2] Programming languages used in most popular websites. [online]. [cit. 2014-10-22]. Dostupné z: http://learntogeek.com/websites/programming-languages-used-in-most-popular-websites/

[3] The Java EE 7 Tutorial. [online]. [cit. 2014-10-22]. Dostupné z: http://docs.oracle.com/javaee/7/tutorial/doc/

[4] JavaServer Faces Technology: Advanced Concepts. [online]. [cit. 2014-10-22]. Dostupné z: http://www.informit.com/articles/article.aspx?p=2015115&seqNum=4

[5] ÇAĞATAY, ÇIVICI. PrimeFaces Users Guide 4.0. [online]. [cit. 2014-10-22]. Dostupné z: http://www.primefaces.org/documentation

[6] Enterprise Java Beans. [online]. [cit. 2014-10-22]. Dostupné z: http://www.tutorialspoint.com/ejb/

[7] Working with JPA Entity Objects. [online]. [cit. 2014-10-22]. Dostupné z: http://www.objectdb.com/java/jpa/persistence/managed

[8] MySQL Workbench. [online]. [cit. 2014-10-09]. Dostupné z: http://www.mysql.com/products/workbench/

[9] Aplikační server. [online]. [cit. 2014-10-09]. Dostupné z: http://java.vse.cz/jsf/chunks/ch02s02.html

[10] Learning JSF2: Managed beans | Max Katz. [online]. [cit. 2014-10-09]. Dostupné z: http://maxkatz.org/2009/08/17/learning-jsf2-managed-beans/

[11] NetBeans IDE 7.3.1 Release Information. [online]. [cit. 2014-09-10]. Dostupné z: https://netbeans.org/community/releases/73/

7 Obrázky a příklady

8 Příloha

8.1 Instalační příručka

Požadavky:

- Glassfish server verze 4.0
- MySQL server
- JDK verze 7

Postup:

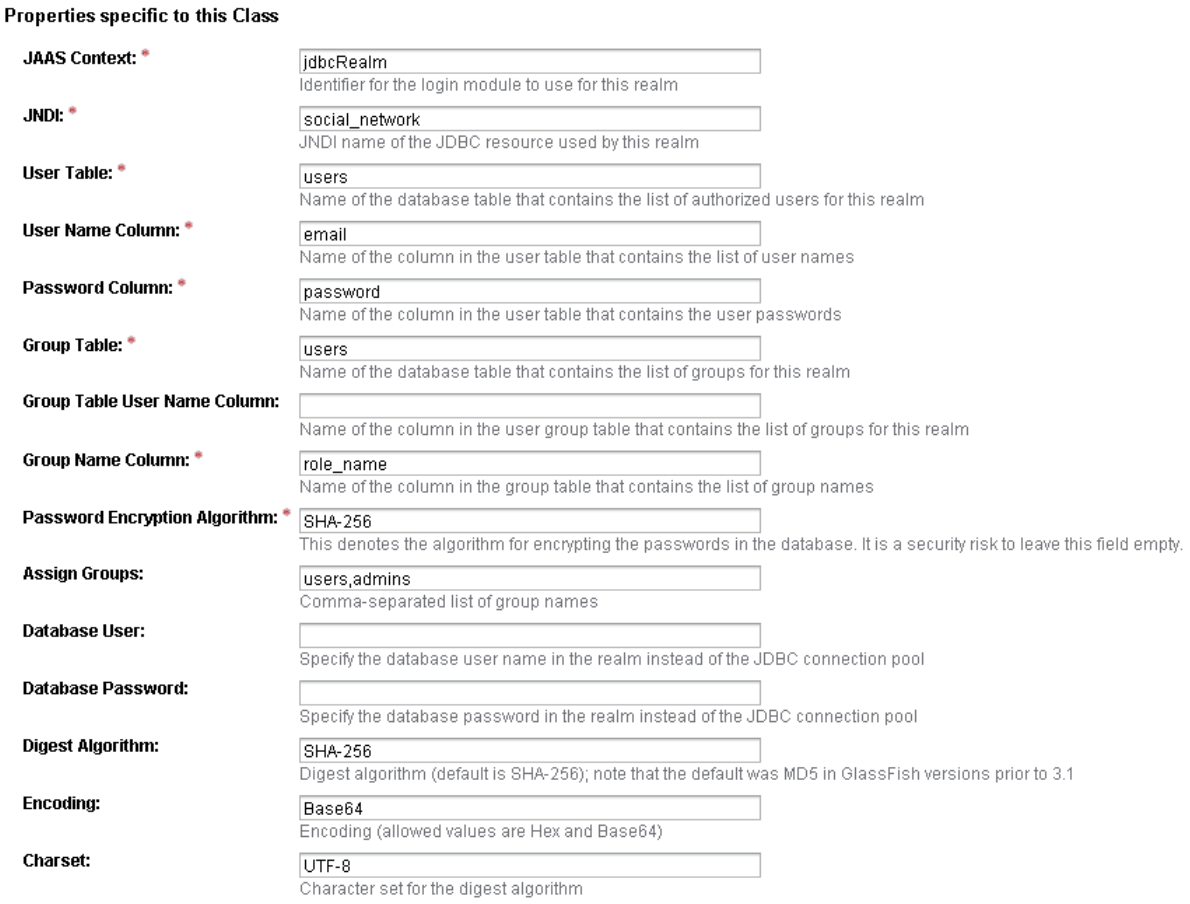
Nejprve je nutné importovat přiložený SQL skript (social_network.sql) na MySQL server, tím se vytvoří databáze s názvem **social_network**. Poté je dobré zkontrolovat konfiguraci MySQL serveru, která by měla být následující:

- Adresa serveru: **localhost**
- Port: **3306**
- Uživatel: **root**
- Heslo: bez hesla

Pokud je nastavení správné, aplikace by se měla po spuštění bez problému připojit k vytvořené databázi.

Dalším krokem je vytvoření bezpečnostní domény na serveru Glassfish, pokud je Glassfish server nainstalován na lokálním počítači měl by mít adresu **localhost** a v defaultním nastavení je přístupný na portu **4848**, tudíž administrativní konzole Glassfish serveru je přístupná na adrese *http://localhost:4848*.

Novou bezpečnostní doménu lze vytvořit pod záložkou **Security**, kde se nachází volba vytvořit nový **Realm**, tento Realm by měl být nastavený podle obrázku.



Po tomto nastavení je možné aplikaci normálně spustit a budou dostupné všechny její funkčnosti.

8.2 Příručka uživatele

8.2.1 Základní funkčnosti aplikace

Po provedení celkového návrhu a realizaci webové aplikace je možné provádět všechny navržené funkčnosti popsané v kapitole: 2.2 Funkcionalita.

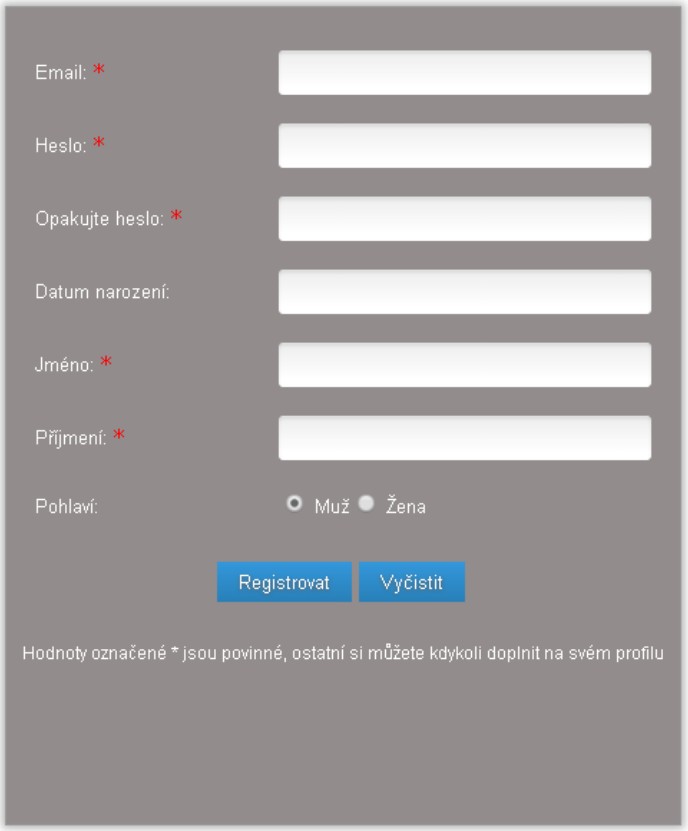
Uživatel je po spuštění aplikace vyzván k přihlášení, pokud však nemá ještě vytvořený účet, je nejprve nutné se zaregistrovat, při registraci je kontrolována:

- správná složitost hesla (alespoň jedno malé a velké písmeno, numerický znak)
- správnost a jedinečnost zadávaného emailu
- formát data narození
- správnost jména a příjmení (nezadány žádné numerické hodnoty)

Po registraci je uživatel znovu vyzván k přihlášení, pokud je autentizace úspěšná, zobrazí se uživateli jeho domovský profil, kde je možné změnit profilovou fotografii, vložit příspěvek a také využít možnosti nabízené na liště menu:

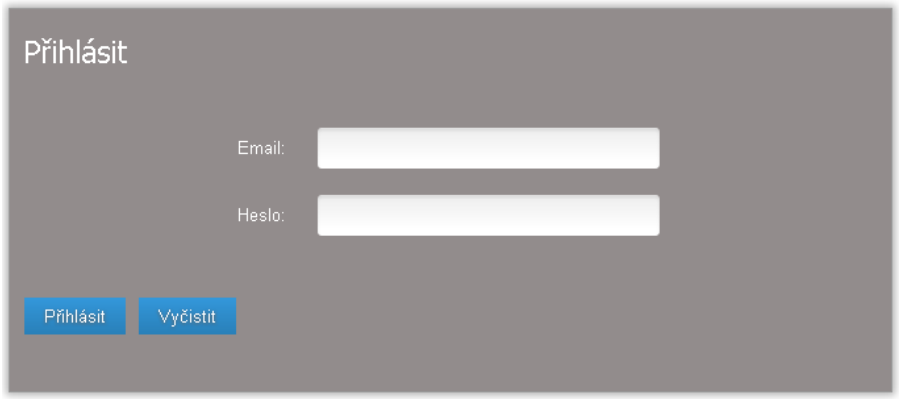
- Zobrazit plochu výsledků
- Upravit osobní údaje a sporty
- Zobrazit sporty přidané na profilu
- Zobrazit žádosti o přátelství
- Vyhledat uživatele
- Odhlásit se z aplikace

8.2.2 Registrační obrazovka



8.2.3 Přihlašovací obrazovka

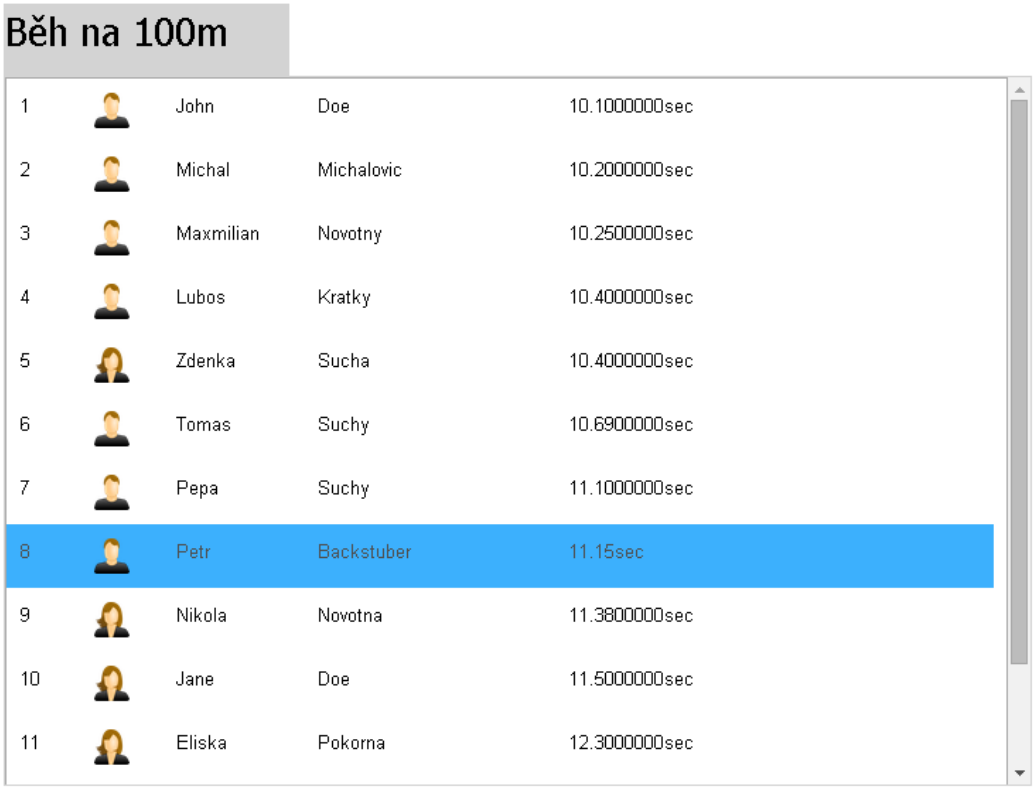
Po přistoupení do aplikace je uživateli zobrazena přihlašovací obrazovka, která obsahuje dvě vstupní pole potřebná pro přihlašování.



8.2.4 Obrazovka výsledků

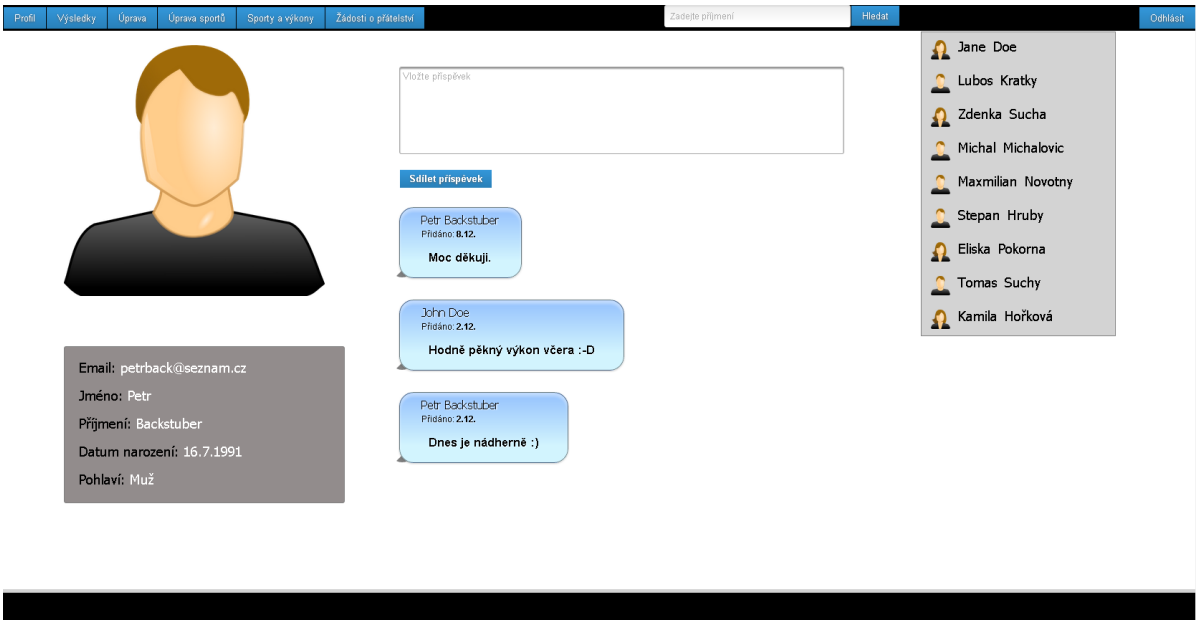
Hlavní motivací k využívání webové aplikace, kterou se zabývá tato práce, je zobrazení obrazovky výsledků, kde mohou uživatelé vidět, jak si vedou v jednotlivých sportech, které mají přidané na svém profilu v porovnání s ostatními registrovanými uživateli.

Výsledky jednotlivých sportů jsou zobrazovány v tabulkách, vždy seřazené od nejlepšího výkonu, tudíž je možné vidět, jaké pořadí zaujímá výkon přihlášeného uživatele. Náhled na některé sporty z plochy výsledků je zobrazen na obrázku.



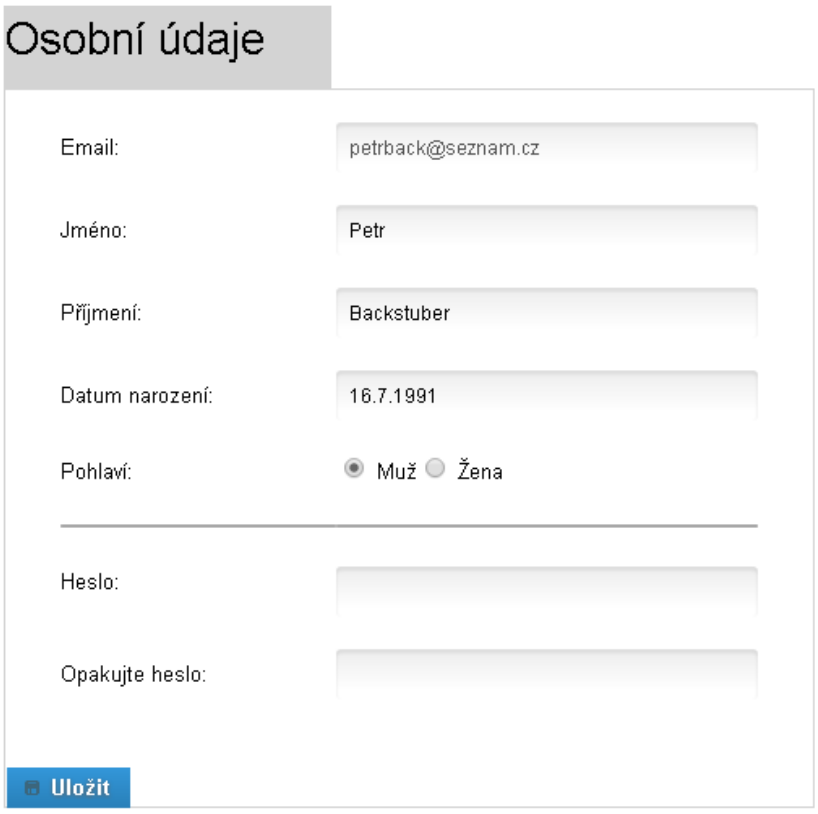
8.2.5 Profil uživatele

Po úspěšném přihlášení do aplikace je uživateli zobrazen jeho profil, na kterém se nachází některé osobní údaje spolu s profilovou fotkou, dále jsou zobrazeny příspěvky přidané k profilu s možností přidávat další. Na pravé straně je zobrazen seznam uživatelů zavedených v seznamu přátel právě přihlášeného uživatele.



8.2.6 Úprava osobních údajů

Na obrazovce pro úpravu osobních údajů jsou zobrazeny osobní údaje přihlášeného uživatele s možností jejich editace, změna osobních údajů obsahuje i možnost změny hesla používaného pro přihlašování do aplikace.



8.2.7 Správa sportů

V rámci správy sportů na uživatelském profilu je možné vlevo vidět sporty, které jsou v rámci aplikace k dispozici pro přidání na profil, a uživatel je ještě přidané nemá, naopak na straně pravé jsou zobrazeny sporty již uložené na profilu uživatele včetně aktuálního nejlepšího výsledku.

Sport je možné z profilu uživatele odebrat pomocí tlačítka **Odstranit** a následného potvrzení tlačítkem **Ano** nebo editovat skóre pomocí tlačítka **Upravit**.

8.2.8 Žádosti o přátelství

Tlačítko **Žádosti o přátelství** slouží k zobrazení všech nepotvrzených, či nesmazaných žádostí o přidání do seznamu přátel, v seznamu jsou zobrazeny profilové fotky jednotlivých uživatelů včetně jména a příjmení, uživatel může buďto žádost schválit, tímto krokem bude uživatel přidán do seznamu přátel nebo žádost zmazat v tom případě nedojde k přidání do seznamu přátel. Zároveň jsou jednotlivé záznamy také odkazy na profil žádající uživatele.