



Dokumentace k semestrální práci z KIV/PIA

# KIVBOOK

**Student:** Martin Kružej  
**St. číslo:** A17N0079P  
**E-mail:** kruzej@students.zcu.cz  
**Datum:** 6. ledna 2018

# Obsah

<b>1</b>	<b>Zadání</b>	<b>1</b>
<b>2</b>	<b>Architektura aplikace</b>	<b>2</b>
2.1	Obecná architektura . . . . .	2
2.2	Architektura servletů . . . . .	3
2.3	Architektura entit . . . . .	4
2.4	Architektura Manažerů . . . . .	5
2.5	Architektura Dao . . . . .	6
<b>3</b>	<b>Aplikace</b>	<b>8</b>
3.1	Use case . . . . .	8
3.2	Prohlížení stránek . . . . .	9
3.3	Registrace . . . . .	9
3.4	Přihlášení . . . . .	9
3.5	Prohlížení uživatelů . . . . .	10
3.6	Úprava profilu . . . . .	10
3.7	Přátelství . . . . .	10
3.8	Statusy . . . . .	11
3.9	Zeď . . . . .	11
<b>4</b>	<b>Moduly</b>	<b>12</b>
4.1	Servlety . . . . .	12
4.1.1	Index . . . . .	12
4.1.2	Information . . . . .	12
4.1.3	Reaction . . . . .	12
4.1.4	Users . . . . .	13
4.1.5	Profile . . . . .	13
4.1.6	Register . . . . .	13
4.1.7	Login . . . . .	14
4.1.8	Welcome . . . . .	14
4.1.9	Logout . . . . .	14
4.1.10	Friends . . . . .	15
4.1.11	Friend . . . . .	15
4.1.12	FriendApprove . . . . .	15
4.1.13	FriendDelete . . . . .	16
4.1.14	Wall . . . . .	16
4.1.15	Like . . . . .	17

4.1.16	Hate . . . . .	17
4.1.17	Upload . . . . .	18
<b>5</b>	<b>Testovací data</b>	<b>19</b>
<b>6</b>	<b>Uživatelská dokumentace</b>	<b>20</b>

# 1 Zadání

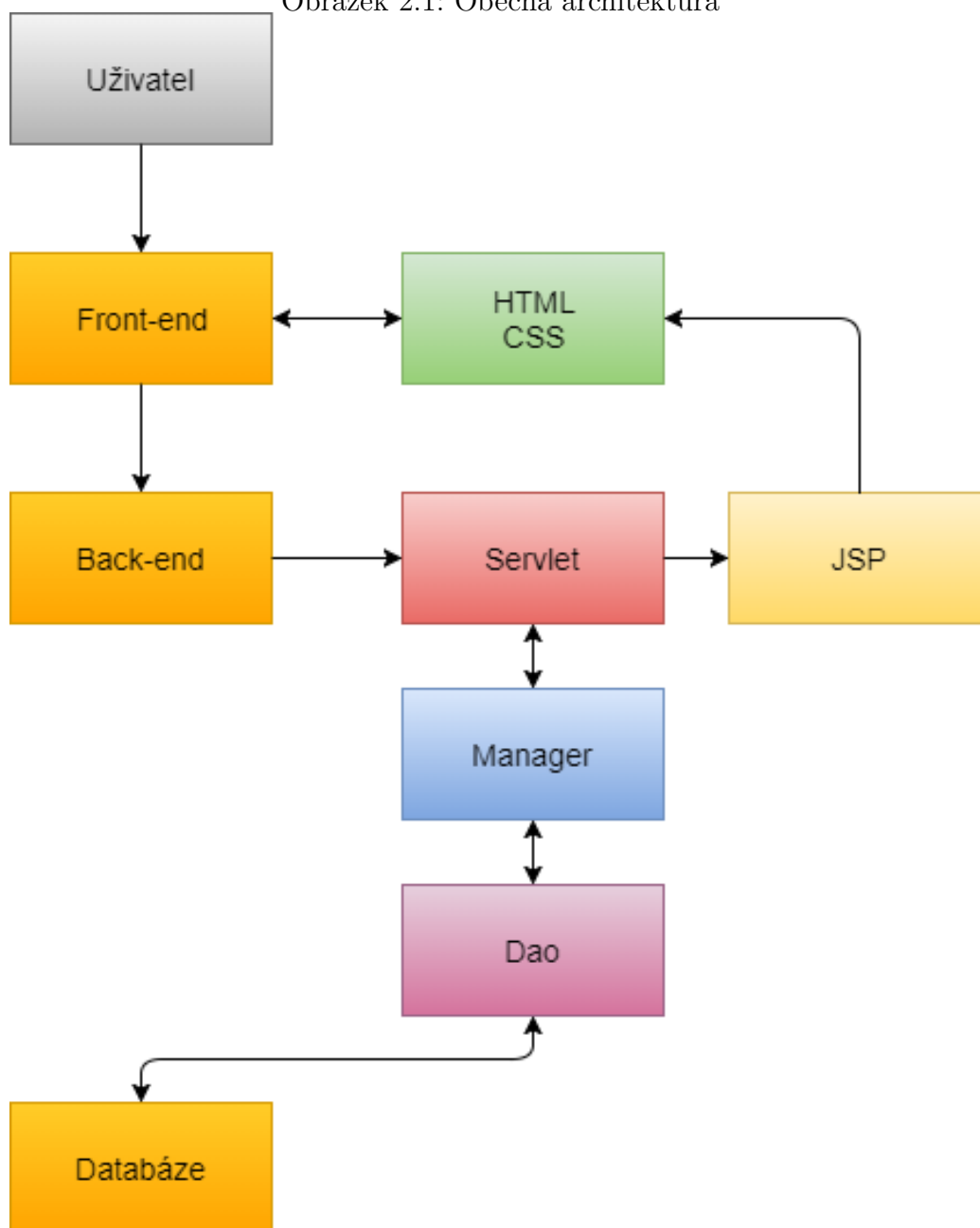
Standardní téma KIV/PIA, viz [KIV/PIA courseware](#).

## 2 Architektura aplikace

### 2.1 Obecná architektura

Aplikace je postavená na technologii Java EE. Jednoduchý diagram ukazující základní obecný chod aplikace je uveden na diagramu 2.1. Uživatel pracuje s tzv. front-endem, který je na klientovi ve formě HTML a CSS. Při zaslání požadavku je tento požadavek předán tzv. back-endu. Zde je zpracován jedním ze servletů, ti v případě potřeby mohou volat databázi přes třídy Manažerů a Dao. Poté vyberou některý z JSP souborů dle požadavku a ten vrací zpět na front-end, kde je zobrazen již ve formátu HTML a CSS. K architektuře bude řečeno více detailů v následujících kapitolách.

Obrázek 2.1: Obecná architektura



## 2.2 Architektura servletů

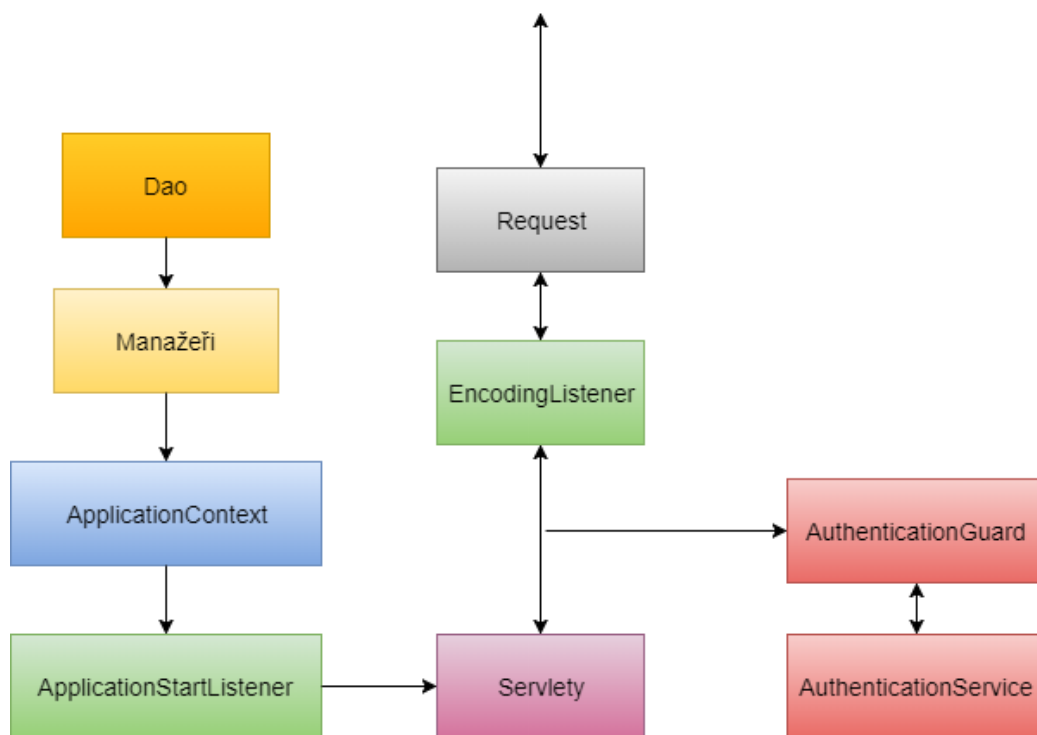
Servlety jsou základní kámen serveru - tyto servlety obsluhují veškeré požadavky, které na server chodí. Pro přístup k databázi využívají různé

manažery a tyto manažery jim musí někdo poskytnout. Máme zde třídu `ApplicationContext`, která vytvoří všechny potřebné třídy Manažerů a Dao. Při inicializaci je spuštěn `StartupListener`, který třídy z `ApplicationContext` injekčně dopraví do servletů, které to potřebují. Tento styl se často označuje jako *Dependency injection*.

Při vyslání požadavku na server se tento požadavek nedostane k servletům hned. Nejdříve projde `EncodingListener`, jenž zajišťuje kódování charakterů ve formátu UTF-8. Dále jsou určité servlety za jakousi bránou, která je brání před neautorizovaným přístupem. Tato brána je `AuthenticationGuard` využívající `AuthenticationService`. Tyto dvě třídy pouze kontrolují, že pro přístup k některým servletům je třeba být přihlášen.

Diagram znázorňující tuto architekturu je uveden na obrázku 2.2.

Obrázek 2.2: Architektura servletů



## 2.3 Architektura entit

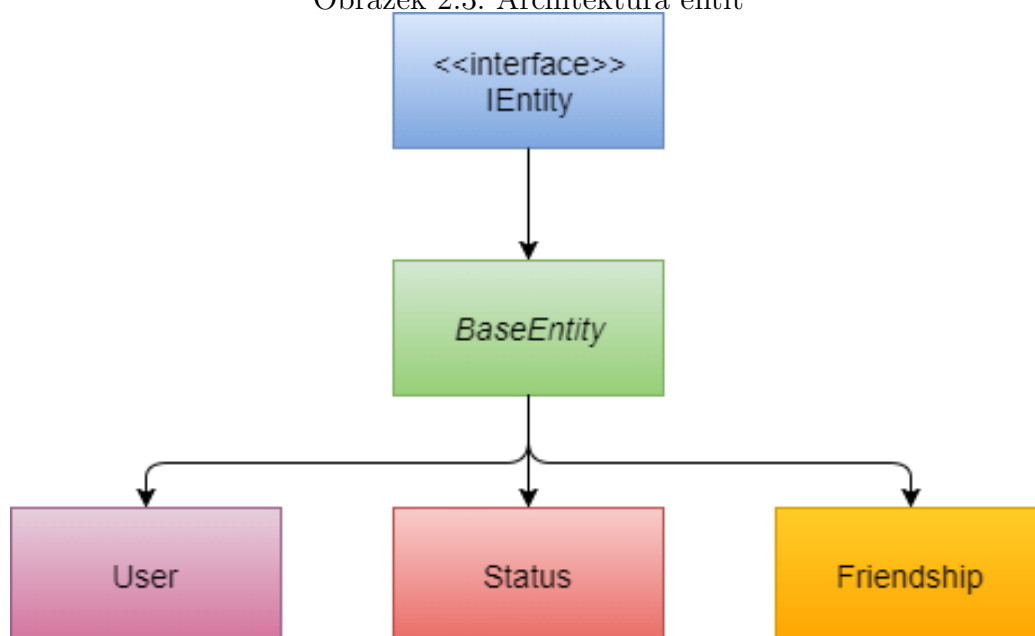
Před podíváním o Manažerech je třeba se podívat na nejnižší úroveň této architektury a to jsou entity. Tyto entity jsou v podstatě tabulky, které se v databázi používají. Nejdříve je zde rozhraní `IEntity`, které dědí abstraktní

třída BaseEntity. Tyto dvě třídy poskytují jednotné *id* pro všechny entity. Ty existují následující:

- **User** - uživatel v databázi
- **Status** - status/tweet v databázi
- **Friendship** - přátelství / požadavek na přátelství v databázi

Následuje dědění abstraktní třídy entitami. Tento vztah je vyobrazen na obrázku 2.3.

Obrázek 2.3: Architektura entit



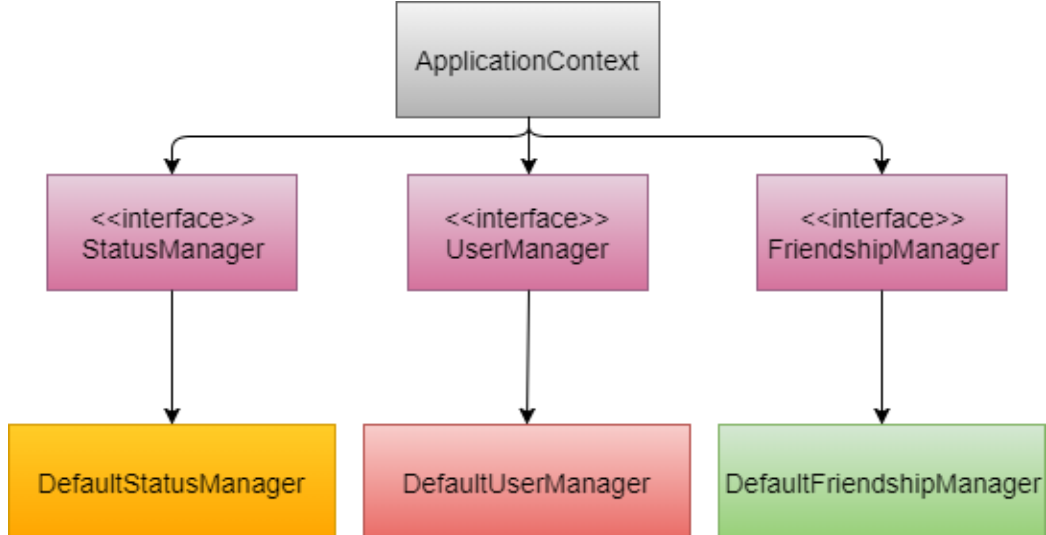
## 2.4 Architektura Manažerů

Manažeři mají za úkol poskytnout prostředníka mezi servlety a databází, potažmo Dao. Poksyťují různé metody pro práci s databází či entitami. Opět se zde setkáváme s rozhraními, ze kterých jsou potom odvozeny samotné třídy. Pro každou jednotlivou entitu existuje jednotliví Manažer. Servlety obecně pracují právě s rozhráním, které je jim injektováno a již je jim jedno, jakou implementaci Manažerů používají.

Vyobrazeno na obrázku 2.4.



Obrázek 2.4: Architektura Manažerů



## 2.5 Architektura Dao

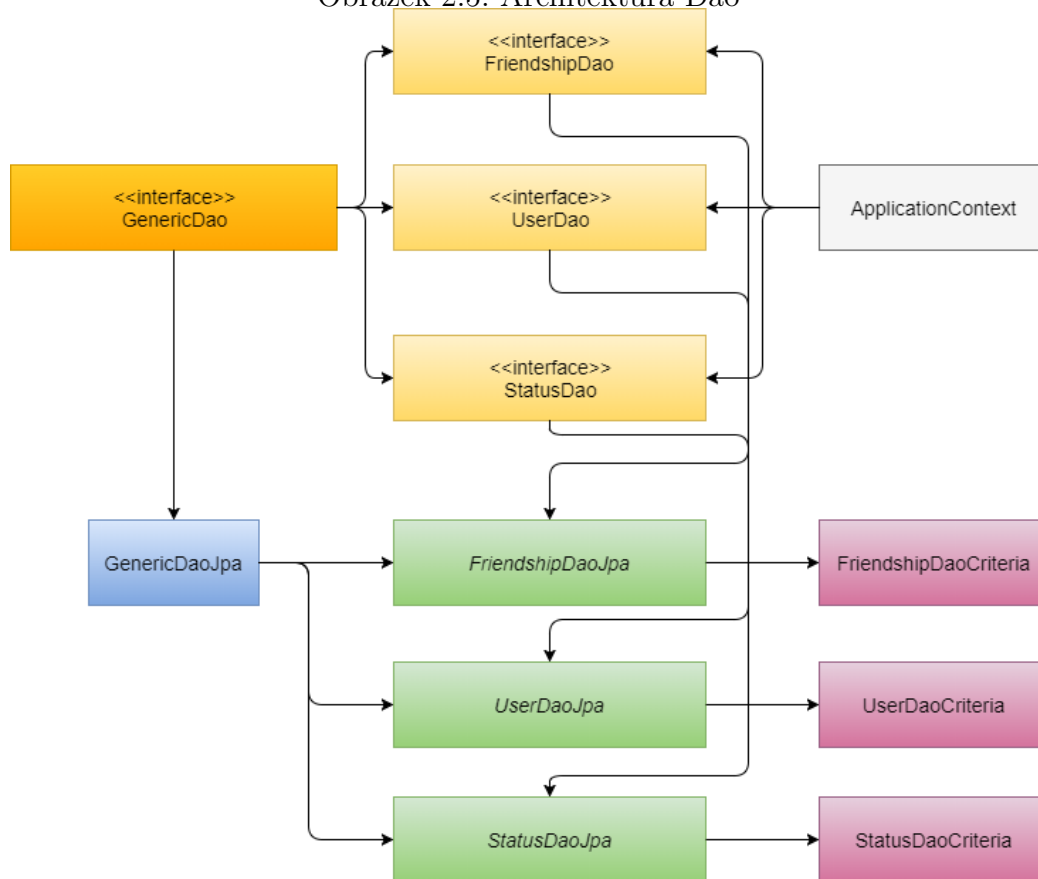
Tato úroveň pracuje s databází. Přes rozhraní slibuje různé metody pro tuto práci a poté je i implementuje. Vzhledem k tomu, jak je tato architektura navržena, je možno udělat několik různých implementací těchto Dao tříd. V aplikaci je použito Criteria, ale je možno udělat třeba JPQL pouhým přidáním tříd a změnou injektování.

Základní rozhraní je GenericDao, kde se definují ty nejzákladnější metody, které každé Dao musí implementovat - jako najít instanci, uložit instanci, smazat instanci. Toto rozhraní dědí rozhraní deklarující již specifické metody pro různé entity - FriendshipDao, UserDao a StatusDao. Tyto třídy jsou to rozhraní, které je injektováno třídou ApplicationContext.

Následuje specifikace JPA. Nejdříve obecná třída GenericDaoJpa s velice obecnými metodami. Následují abstraktní třídy pro jednotlivé entity. Tyto třídy dědí jak GenericDaoJpa tak jednotlivé Dao rozhraní. Poslední částí architektury jsou samotné implementace jednotlivých metod. Jak bylo řečeno v předešlých odstavcích, tato aplikace využívá Criteria, ale zde by mohly být další třídy, např. FriendshipDaoJPQL využívající JPQL.

Vyobrazeno na obrázku 2.5.

Obrázek 2.5: Architektura Dao



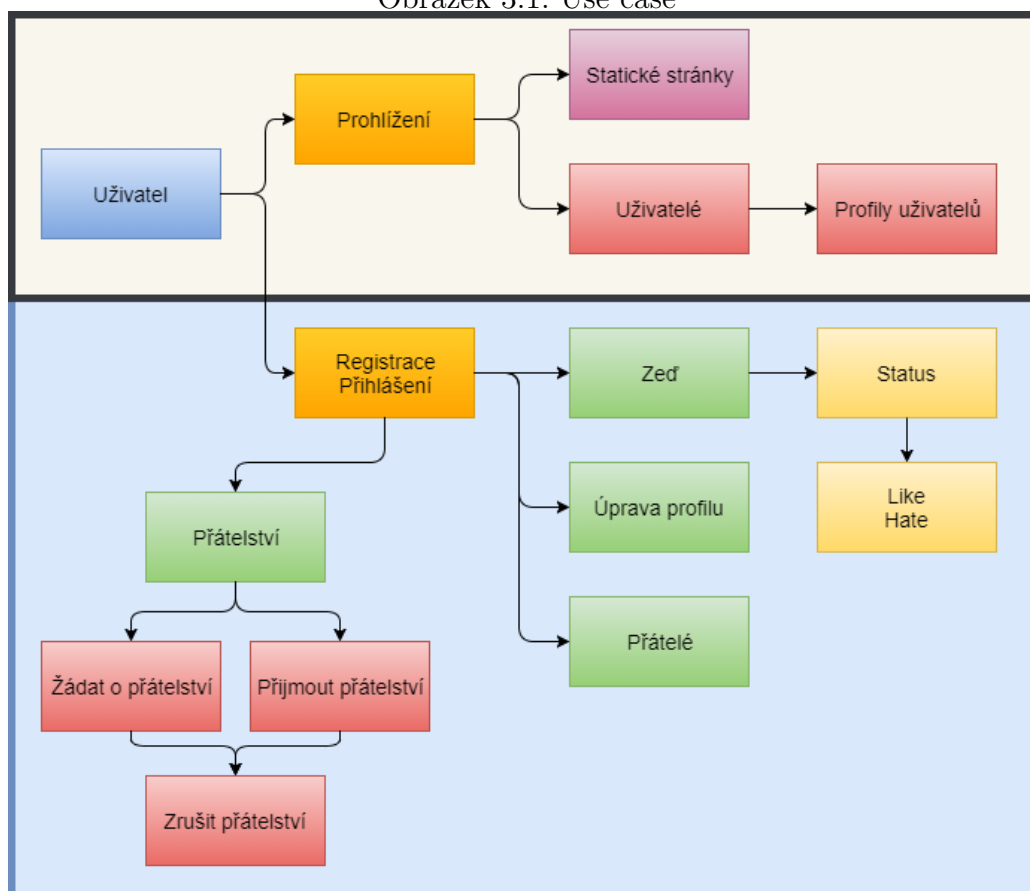
## 3 Aplikace

### 3.1 Use case

Aplikace imituje některé známé sociální sítě. Use case aplikace je vyobrazen na obrázku 3.1. Horní část diagramu ve světlé žluté je ta část, která je přístupná nepřihlášenému uživateli. Tento uživatel si může prohlížet některé statické stránky, seznam registrovaných uživatelů a profily těchto uživatelů. Přístup ke zbytku aplikace mu bude zamítnut dokud se nezaregistruje/nepřihlásí.

Po přihlášení do systému je uživateli zpřístupněn zbytek aplikace. Dostává nyní vlastní zeď, může upravovat svůj profil, dívat se na své přátele, žádat o přátelství, zamítat přátelství a také vytvářet statusy a lajkovat/hejtovat je.

Obrázek 3.1: Use case



## 3.2 Prohlížení stránek

Aplikace dovoluje nepřihlášenému uživateli prohlédnout si některé stránky. Může se podívat na názory některých uživatelů či informace o aplikaci. Je mu dovoleno zhlédnout seznam registrovaných uživatelů a může si prohlížet jejich profily. Sám nikde nenajde nikdy odkaz, který by ho zavedl do privilegované části aplikace, ale chytrý uživatel může samozřejmě měnit URL. Proto je v aplikaci AuthenticationGuard, který nepřihlášeného uživatele nikam dál nepustí a vybídne ho k přihlášení.

## 3.3 Registrace

Uživatel má možnost se zaregistrovat. Bude mu nabídnut registrační formulář s následujícími položkami:

- **Uživatelské jméno** - uživatelské jméno, kterým se bude přihlašovat - **povinné**
- **Heslo** - heslo k uživatelskému účtu - **povinné**
- **Kontrola hesla** - znovu zadání hesla - **povinné**
- **Pohlaví** - jestli je uživatel muž, či žena - **nepovinné**
- **Datum narození** - kdy se uživatel narodil - **nepovinné**
- **Souhlas s podmínkami** - uživatel souhlasí s podmínkami - **povinné**
- **Kontrola robotů** - uživatel je nucen napsat číslici 4 - **povinné**

Povinné položky formuláře nepovolí odeslat, dokud nejsou vyplněny. Uživatelské jméno je povoleno pouze písmena a číslice. Hesla se musejí shodovat. Datum narození nemusí být vyplněn vůbec ale pokud uživatel vyplní pouze část data, je mu vynadáno. Heslo je dále hashováno, než je uloženo do databáze. V případě chybných údajů, například že se neshodují hesla, je formulář předvyplněn některými údaji, jmenovitě uživatelské jméno, pohlaví, souhlas s podmínkami a kontrola robotů.

## 3.4 Přihlášení

Při přihlášení uživatel vyplňuje svoje uživatelské jméno a heslo. V případě neshody je uživatel nepřihlášen. Po úspěšném přihlášení se schovávají

položky pro přihlášení/registraci a jsou nahrazeny profilem a odhlášením. Nyní je uživatel volný kamkoliv do aplikace. Žádný odkaz nevede zpátky do statických stránek, ale chytrý uživatel si samozřejmě poradí. Pokud se přihlášený uživatel dostane zpět na stránky registrace a přihlášení, jednoduše mu nebude dovoleno žádnou z forem vyplňovat, dokud bude přihlášen.

### 3.5 Prohlížení uživatelů

Přihlášený uživatel si také může prohlédnout ostatní uživatele. Po kliknutí na jejich uživatelská jména vidí jejich profily. Může si také prohlédnout své přátele, pokud nějaké má, v záložce *přátelé*. U profilů na rozdíl od nepřihlášených uživatelů vidí další tlačítka. Jde o:

- Pokud není s uživatelem přítel, uvidí *Poslat žádost o přátelství*
- Pokud jsou přátelé, uvidí *Zrušit přátelství*
- Pokud jeden z nich zažádal o přátelství, neuvidí nic

Detaily k přátelství budou popsány v dalších kapitolách.

### 3.6 Úprava profilu

Pokud se uživatel dostane k prohlížení svého profilu, nebude si prohlížet ale má možnost ho změnit. Z klasických je podpora pro: heslo, email, pohlaví. Uživatel zde má možnost také nahrát nový profilový obrázek. Stačí jej vybrat a kliknout na *Nahrát*.

### 3.7 Přátelství

Uživatelé mohou mezi sebou vytvářet přátelství. Jde o oboustrannou dohodu. Jeden z uživatelů musí vždy požádat o přátelství - přes profily, jak bylo řečeno. Druhý musí tuto žádost potvrdit. Všechny žádosti jsou zobrazeny na Zdi. Zde uživatel vidí odchozí žádosti, které podal a příchozí žádost, které mu ostatní posílají. U svých žádostí má možnost je stáhnout, čímž je zruší. U příchozích je buď přijme, čímž vzniká přátelství, nebo bude ignorovat, čímž se zruší.

Přátelství zajišťuje přidání uživatele do záložky *Přátelé* a jeho statusy se budou zobrazovat na zdi uživatele. Přátelství se dá kdykoliv zrušit - stačí jít na profil přítele a stisknout *Zrušit přátelství*. Uživatel je odebrán ze seznamu přátel a jeho statusy se již nebudou zobrazovat.

## 3.8 Statusy

Uživatel může vytvářet statusy. Na stránce Zeď je možnost vyplnit text statusu a poté kliknout na *Statusovat*, čímž se vytvoří nový status. Automaticky je zobrazen čas, kdy byl status napsán. Tento status je viditelný pro samotného uživatele a jeho přátele.

Každý status lze lajkovat či hejtovat. Stačí kliknout na tlačítko *Lajkovat* či *Hejtovat*. Seznam těch, kteří určitý status lajkovali/hejtovali je umístěn pod statusem stejně jako celkový počet těchto lajků/hejtů. Uživatel může jeden status:

- Lajkovat
- Hejtovat
- Lajkovat poté, co lajkoval, tím zruší lajk
- Hejtovat poté, co hejtoval, tím zruší hejt
- Pokud se pokusí mixovat, tedy lajkovat poté co hejtoval, aplikace mu vynadá

Lajky a Hejty nezanikají se zánikem přátelství.

## 3.9 Zeď

Zeď je hlavní středisko aplikace. V pravé části jsou notifikace - příchozí a odchozí žádosti a lze s nimi manipulovat. Uprostřed je možnost psát statusy. Pod tím jsou zobrazeny statusy přátel a uživatele. Vždy se zobrazují po deseti a je možno mezi nimi procházet, pokud jich je hodně. Platí:

- Pokud je statusů méně než 10, žádné tlačítko pro stránkování není zobrazeno
- Pokud je statusů více než 10, objeví se tlačítko *Další*. To zobrazí statusy 10+ (obecně x+)
- Při zobrazení statusů x aplikace řeší, jestli existuje více statusů než x+10. Pokud ne, není uvedeno tlačítko *Další* (vzhledem k tomu že není kam dál jít). Pokud jich je víc, tlačítko *Další* bude přítomno.
- Pokud se zobrazují statusy x, kde x je aspoň 10, vždy je tlačítko *Předchozí*. To zobrazí statusy x - 10

## 4 Moduly

### 4.1 Servlety

Každý servlet má dvě hlavní metody - `doGet` a `doPost`. Ne každý servlet je definován pro práci s oběma těmito metodami, ale i nepodporovaná metoda neshodí server. Existují následující servlety:

#### 4.1.1 Index

Úvodní index. Dokud nebude řečeno jinak, následující servlety nejsou chráněny `AuthenticationGuard`, tedy jsou přístupné nepřihlášeným uživatelům. Úkolem tohoto servletu je vrátit uživateli *index.jsp*.

Metody:

- **doPost** - předává **doGet**
- **doGet** - vrací *index.jsp*

#### 4.1.2 Information

Tento servlet vrací statickou stránku o informacích o aplikaci.

Metody:

- **doPost** - předává **doGet**
- **doGet** - vrací *information.jsp*

#### 4.1.3 Reaction

Tento servlet vrací statickou stránku o reakcích uživatelů na aplikaci.

Metody:

- **doPost** - předává **doGet**
- **doGet** - vrací *reaction.jsp*

#### 4.1.4 Users

Tento servlet vypisuje stránku se všemi registrovanými uživateli. K tomu využívá *UserManager*.

Metody:

- **doPost** - předává **doGet**
- **doGet** - připraví seznam všech registrovaných uživatelů a vrátí *users.jsp*

#### 4.1.5 Profile

Tento servlet mění chování dle metod a přihlášenosti uživatele. Pokud je uživatel nepřihlášen, zobrazuje profil zadaného uživatele (přes parametr *username*). Pokud je přihlášen, zobrazuje také profil uživatele a dodává tlačítka podle toho, jaký je vztah mezi uživateli. Pokud je člověk přihlášen a klikl na sebe, tento servlet umožňuje úpravu profilu. Výše popsáno je vše **doGet**, kde **doPost** slouží právě pro úpravu profilu uživatele.

Pro svou práci vyžaduje servlet jak *UserManager* tak *FriendshipManager*.  
Metody:

- **doGet** - vyžaduje parametr *username*. Když:
  - parametr není zadán nebo je špatně - vrátí *profile.jsp* s chybovou hláškou
  - parametr značí uživatele, jenž je zároveň přihlášený - vrátí *profile.jsp* jako úpravu profilu
  - parametr značí nějakého uživatele - vrátí *profile.jsp* a nabízí tlačítka podle toho, jaký je mezi uživateli vztah
- **doPost** - obsluha úpravy profilu. Samotnou úpravu řeší *UserManager*.  
Vrací *welcome.jsp* s hláškou o úpravě profilu

#### 4.1.6 Register

Servlet starající se o registraci nových uživatelů. Využívá *UserManager*. Ten se stará o registraci samotnou, ale kontrolu některých parametrů obstarává již servlet.

Metody:

- **doGet** - vrátí *register.jsp*
- **doPost** - kontroluje parametry, než řízení předává *UserManager*



- hesla se neshodují - vrací *register.jsp* a chybovou hlášku
- test na robota je nesprávně - vrací *register.jsp* a chybovou hlášku
- špatně zadané datum narození - vrací *register.jsp* a chybovou hlášku
- *UserManager* vyhodí výjimku - vrací *register.jsp* a zprávu výjimky
- pokud nedojde k výjimce, vrací *login.jsp* a zprávu o úspěšné registraci

#### 4.1.7 Login

Servlet starající se o přihlášení uživatelů. Využívá *AuthenticationService*. Ten se stará o samotné přihlášení.

Metody:

- **doGet** - vrací *login.jsp*
- **doPost** - předá parametry *AuthenticationService*. Pokud vše v pořádku, vrací *welcome.jsp*, jinak *login.jsp* a chybovou hlášku

#### 4.1.8 Welcome

Následující servlety jsou chráněny *AuthenticationGuard*.

Tento servlet vrací statickou stránku s textem.

Metody:

- **doPost** - předává **doGet**
- **doGet** - vrací *welcome.jsp*

Za zmínku stojí, že tato stránka je často využívána pro komunikaci s uživatelem. V závislosti na parametrech vypisuje buď vítání uživatele, chybovou hlášku nebo hlášku o úspěchu.

#### 4.1.9 Logout

Tento servlet řeší odhlášení z aplikace. Využívá *AuthenticationService*, která samotné dohlášení řeší.

Metody:

- **doPost** - předává **doGet**
- **doGet** - požádá *AuthenticationService* o odhlášení, poté vrací *index.jsp*

#### 4.1.10 Friends

Tento servlet vypisuje stránku s přáteli přihlášeného uživatele. Využívá *UserManager* a *FriendshipManager*.

Metody:

- **doPost** - předává **doGet**
- **doGet** - připraví seznam přátel přihlášeného uživatele, načtež vrátí *friends.jsp*

#### 4.1.11 Friend

Tento servlet vyřizuje žádost o přátelství. Využívá *UserManager* a *FriendshipManager*.

Metody:

- **doPost** - předává **doGet**
- **doGet** - vyžaduje parametr *username*. Z toho získá uživatele, kterého se nové přátelství má týkat. Je kontrolováno, že existuje. Následně je zkontrolováno, že mezi uživateli již nějaký vztah není. Pokud vše proběhne správně, vrátí *welcome.jsp*

Za zmínku stojí, že správně by mělo proběhnout vždy, pokud se uživatel nebude do URL míchat.

#### 4.1.12 FriendApprove

Tento servlet vyřizuje potvrzení žádosti o přátelství. Využívá *FriendshipManager*. Samotné potvrzení provádí právě manažer.

Metody:

- **doPost** - předává **doGet**
- **doGet** - vyžaduje parametr *id*. Předává řízení manažerovi se zadaným *id*. V případě chyb vrátí *welcome.jsp* s chybovou hláškou, jinak s úspěšnou hláškou.

K chybě v *id* by nemělo dojít, pokud se uživatel nebude do URL míchat.

### 4.1.13 FriendDelete

Tento servlet vyřizuje zrušení žádosti o přátelství. Využívá *FriendshipManager*. Samotné zrušení provádí právě manažer.

Metody:

- **doPost** - předává **doGet**
- **doGet** - vyžaduje parametr *id*. Předává řízení manažerovi se zadaným *id*. V případě chyby vrátí *welcome.jsp* s chybovou hláškou, jinak s úspěšnou hláškou.

K chybě v *id* by nemělo dojít, pokud se uživatel nebude do URL míchat.

### 4.1.14 Wall

Servlet obsluhující zeď aplikace. Využívá všechny tři manažery v aplikaci. Jeho účel se liší podle metody.

- **doPost** - obstarává vytvoření statusu. Pokud není žádný text poslán, předává **doGet**. Pokud je text prázdný, předává **doGet**. Jinak předává *StatusManager* a poté předá nakonec opět **doGet**
- **doGet** - očekává parametr *stream*. Má několik úkolů:
  - **Notifikace** - připraví seznam všech nepotvrzených žádostí o přátelství, kde je nějak zainteresován přihlášený uživatel a předává je JSP
  - **Statusy** - připraví seznam všech statusů uživatele a jeho přátel, seřadí je podle data a předává je JSP
  - **Stream** - daný seznam statusů, pokud není nulový, je měněn v závislosti na *streamu*
    - \* *stream* nebyl zadán - přiřadí se 0
    - \* **stream** byl zadán a nelze parsovat do Long - přiřadí se 0
    - \* **stream** byl zadán a není dělitelný deseti - přiřadí se 0
    - \* **stream** byl zadán a je dělitelný deseti - nechá se být
  - V této chvíli se *stream* nastaví dle těchto pravidel a následuje kontrola dalších parametrů:
  - *stream* je 0
    - \* **endIndex** se nastaví na 10

- \* pokud je **endIndex** větší než velikost seznamu, přiřadí se mu velikost seznamu
- \* vezme se `subList (0,endIndex)` a ten se předá JSP
- *stream* není 0
  - \* **startIndex** je *stream*
  - \* **endIndex** je *stream* + 10
  - \* dokud je **startIndex** větší než velikost seznamu, ubereme o 10
  - \* pokud je **endIndex** větší než velikost seznamu, přiřadíme velikost seznamu
  - \* vezmeme `subList (startIndex, endIndex)` a ten se předá JSP

Tímto dosáhneme správného sub-seznamu a předejdeme chybnému parametru *stream*, který by tam nezbedný uživatel mohl zadat

#### 4.1.15 Like

Servlet řešící lajkování statusu. Využívá *StatusManager* a *UserManager*. Samotné lajkování provádí *StatusManager*.

Metody:

- **doPost** - předává **doGet**
- **doGet** - předává řízení *StatusManager* s parametrem *id*. V případě chyb vrátí *welcome.jsp* s chybovou hláškou, jinak s úspěšnou hláškou

K chybě v *id* by nemělo dojít, pokud se uživatel nebude do URL míchat.

#### 4.1.16 Hate

Servlet řešící hejtování statusu. Využívá *StatusManager* a *UserManager*. Samotné hejtování provádí *StatusManager*.

Metody:

- **doPost** - předává **doGet**
- **doGet** - předává řízení *StatusManager* s parametrem *id*. V případě chyb vrátí *welcome.jsp* s chybovou hláškou, jinak s úspěšnou hláškou

K chybě v *id* by nemělo dojít, pokud se uživatel nebude do URL míchat.

### 4.1.17 Upload

Servlet řešící nahrání souboru na server. V aplikaci výhradně použito pro profilový obrázek. Využívá *UserManager*.

Metody:

- **doGet** - nemá příliš smysl volat nahrání serveru jako GET, servlet vrátí *index.jsp*
- **doPost** - pokud nebyl obrázek vybrán, servlet vrací *welcome.jsp* s chybovou hláškou, jinak nahraje soubor na server a upraví uživatelskou položku *avatar*

## 4.2 Manažeri

## 4.3 Dao

## 4.4 Entity

## 5 Testovací data

K aplikaci je připojen SQL skript, jenž načte do databáze testovací data. Tato data poskytují ukázkou toho, co lze v aplikaci dělat. Jsou k dispozici tři testovací uživatelé:

Uživatelské jméno	Heslo
user1	User@1234
user2	User@1234
user3	User@1234

Navíc platí, že:

- **user1** byl narozen 29.6.1987, je muž a jeho email je user1@example.com
- **user2** byl narozen 8.3.1981, je žena a nemá uvedený email
- **user3** nemá uvedené žádné z těchto vlastností

Co se týče přátelství, tak platí, že **user1** je přítelem s **user2**, **user1** požádal o přátelství **user3** a **user3** požádal o přátelství **user2**. Tato přátelství ještě nebyla potvrzena. **user3** má napsané dva statusy, **user1** a **user2** společně 11, aby byla ukázána stránkování po 10. Některé statusy byly lajkovány/hejtovány.

Po nahrání těchto dat by měla být viditelná funkčnost aplikace a co obsahuje a umožňuje.

## 6 Uživatelská dokumentace