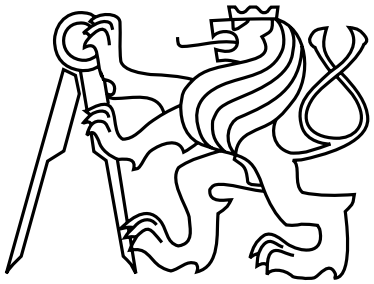
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

Implementace webové aplikace pro kompenzaci diabetes mellitus

Bc. Václav Dobeš

Vedoucí práce: Ing. Daniel Novák, Ph.D.

Studijní program: Otevřená informatika, Magisterský

Obor: Softwarové inženýrství

24. května 2016

Poděkování

Chtěl bych poděkovat především své rodině za podporu při studiu, vedoucímu mé diplomové práce panu Ing. Danielovi Novákovi, Ph.D. za vedení a pravidelné konzultace během psaní této práce a všem, kteří se podíleli na testování aplikace.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 8. 5. 2016 .....

Abstract

This thesis deals with analysis of requirements followed by implementation of web application to support compensation of diabetes mellitus. The thesis includes description of the main use cases followed by description of realization in Java EE language. The end of the thesis is devoted to testing and future improvements.

Abstrakt

Tato práce se zabývá analýzou požadavků a následnou implementací webové aplikace na podporu kompenzace diabetes mellitus. Práce zahrnuje popis klíčových funkcí aplikace následovaný popisem samotné realizace v programovacím jazyce Java EE. Konec práce je věnován testování a popisu možných rozšíření do budoucna.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Diabetes mellitus je souhrnný název pro onemocnění, která se projevují zvýšenou hladinou cukru v krvi (tzv. hyperglykémií). Vysokou hladinu cukru v krvi reguluje hormon zvaný inzulín. Ten umožňuje cukru k krvi vstoupit do buněk, kde slouží jako zdroj energie. Hladina cukru v krvi se zvyšuje především po jídle. Rozlišují se dva základní typy nemoci: diabetes mellitus 1. typu a diabetes mellitus 2. typu. Oba typy mají podobné příznaky, ale odlišné příčiny vzniku a léčby. Základem léčby diabetu 1. typu je aplikace inzulínu. Léčba diabetu 2. typu pak spočívá hlavně v dodržování diety a pravidelného pohybu. [14] [3] [7]

Jedním z projektů na ČVUT FEL je projekt Mobiab. Ten se zabývá kompenzací diabetes mellitus pomocí edukačního efektu mobilních technologií. V rámci projektu vznikla mobilní aplikace pro sledování kalorií, inzulínu, glykémie, krevního tlaku a hmotnosti. Data z aplikace se odesílají server. Cílem této práce byl návrh a implementace webového serveru pro uložení těchto dat. Důležitým požadavkem na aplikaci je správa dat přes webový prohlížeč, a proto je součástí řešení webový klient.

1.1 Struktura práce

Nejprve je popsán současný stav projektu. Dále představím aplikace, které se zabývají podobnou problematikou.

V kapitole Analýza (2) jsou popsány požadavky, které musí aplikace splňovat. Po výpisu požadavků následuje popis klíčových případů užití. Nakonec je představen doménový model aplikace a její vzhled.

Ze začátku kapitoly Realizace (3) popisuji technologie, které byly použity při implementaci. Poté následuje popis architektury aplikace, komunikace s databází a popis zajímavých částí z implementace. Konec kapitoly je věnován nasazení aplikace do provozu.

Kapitola Testování (4) Popisuje průběh testování aplikace. Nejprve je popsán způsob testování, poté testované případy užití. Na konci kapitoly je seznam objevených nálezů s návrhy na jejich odstranění.

V závěru (5) hodnotím výsledek práce a popisuji možná rozšíření do budoucna.

1.2 Současný stav

V rámci projektu Mobiab vznikla mobilní aplikace pro operační systém Android. Aplikace popisovaná v této práci obsahuje stejné funkce jako mobilní aplikace Mobiab a měla by sloužit jako centrální úložiště dat. Veškerá data z mobilní aplikace se již nyní synchronizují na server <`http://sledujeme-kalorie.cz/`>. Serverovou část je potřeba kompletně přepsat tak, aby byla lépe rozšiřitelná a škálovatelná. Současná aplikace je napsána v jazyce PHP víceméně procedurálním stylem.

Týmu projektu Mobiab se podařilo získat databázi téměř 7 tisíc jídel včetně výživových hodnot (sacharidů, tuků a bílkovin) na 100 g potraviny. Jídla jsou zařazena do kategorií. Každé jídlo může být ve více kategoriích. Kromě jídel Mobiab disponuje databází aktivit. U každé aktivity je evidováno množství vydané energie na 1 kg váhy po dobu 1 minuty. Předpokládejme aktivitu s hodnotou 0,54. Člověk s váhou 70 kg pak po dobu jedné hodiny vydá energii vypočítanou pomocí vzorce: 54 * 70 * 60. Obě tyto databáze posloužily jako základ nového systému.

1.3 Podobná řešení

1.3.1 Kalorické tabulky

Webová aplikace na adrese <`http://www.kalorictabulky.cz`> je zaměřená čistě na kalorický příjem a výdej. Jde o velmi propracovanou aplikaci s kvalitní databází jídel. Na rozdíl od databáze jídel poskytnuté Mobiabem obsahuje informace o množství vody, vápníku, vlákniny a jiných složkách. [6]

Aplikace je spíše určená lidem, kteří chtějí zhubnout. Z pohledu diabetika zde chybí správa glykémií a inzulínu.

1.3.2 DiaMonitor

DiaMonitor <`https://diamonitor.com/dm/`> je interaktivní deník, který se zaměřuje na sledování glykémie. Diabetikům pomáhá s rozhodováním o případných úpravách denního režimu tak, aby byl diabetes dobře kompenzovaný. [4]

Kapitola 2

Analýza

V této kapitole popíši, jak bude aplikace fungovat. Jako první se budu zabývat popisem uživatelů aplikace, dále výčtem funkčních a nefunkčních požadavků. Nakonec jsou zde popsány důležité případy užití a doménový model aplikace.

2.1 Uživatelské role

V systému jsou tři základní uživatelské role. Nepřihlášený uživatel, přihlášený uživatel a lékař.

2.1.1 Nepřihlášený uživatel

Jedná se o základní roli, kterou návštěvník získá ihned po načtení aplikace. Nepřihlášený uživatel se může pouze registrovat do systému. Tím se z něj stává přihlášený uživatel.

2.1.2 Přihlášený uživatel

Může zadávat a procházet data jako jsou aktivity (kalorický výdej), konzumace (kalorický příjem), glykémie, inzulín, krevní tlak a hmotnost. Rovněž může vybírat lékaře, se kterým si poté může psát zprávy přímo v aplikaci. Pokud se bude dále v textu hovořit o uživateli, vždy se tím myslí přihlášený uživatel.

2.1.3 Lékař

Lékař má přístup k datům svých pacientů (uživatelů, kteří si daného lékaře vybrali). Dále může pacientům posílat zprávy přímo z aplikace. Lékař nemá možnost sám se registrovat do systému. Vložení musí udělat programátor na žádost lékaře pomocí SQL příkazů přímo nad databází.

2.2 Požadavky na systém

Níže jsou uvedeny veškeré požadavky, které jsou kladeny na systém.

2.2.1 Funkční požadavky

2.2.1.1 Obecné

1. V aplikaci bude použita již hotová databáze jídel a aktivit, kterou má Mobiab k dispozici.
2. Veškerá data z původní aplikace budou k dispozici i v novém systému.
3. Aplikace bude poskytovat data skrze REST API a to ve formátu XML a JSON.
4. Uživatelé a lékaři se do systému přihlásí pomocí svého emailu a hesla.

2.2.1.2 Požadavky z pohledu uživatele

1. Nepřihlášený uživatel má možnost registrovat se do systému.
2. Uživatel může přidávat záznamy o jeho kalorickém příjmu, kalorickém výdeji, glykémii, inzulínu, krevním tlaku a hmotnosti. Tyto záznamy lze zpětně upravovat popřípadě úplně smazat.
3. K záznamům jednotlivých modulů lze vyplnit datum vložení záznamu do systému.
4. Veškeré záznamy lze zobrazit ve formě tabulky nebo grafu. Záznamy v tabulce budou pro lepší přehlednost stránkované.
5. Záznamy je možné filtrovat podle data v rozsahu od do. Minimální rozsah je jeden den.
6. Jídla a aktivity bude možné přidat do oblíbených.
7. Jídla a aktivity bude možné odebrat z oblíbených.
8. Uživatel bude mít možnost změnit své heslo a další osobní údaje.
9. Uživatel může vybírat ze seznamu lékařů a to opakovaně. K uživateli může být přiřazen vždy pouze jeden lékař.
10. Editace modulů (přidání, úprava a mazání) bude uložena do historie.

2.2.1.3 Požadavky z pohledu lékaře

1. Po přihlášení lékař uvidí seznam jeho pacientů. Tento seznam je možné filtrovat podle jména.
2. Záznamy pacienta budou k dispozici na jednom místě.
3. Záznamy pacienta bude možné globálně filtrovat podle data.

2.2.2 Nefunkční požadavky

1. Aplikace bude plně škálovatelná (vertikálně i horizontálně).
2. Aplikace bude implementována v jazyce Java.
3. Podpora lokalizace - aplikace bude připravena na budoucí rozšíření o další jazyky.
4. Bezpečnost - hesla nejsou nikdy uložena v otevřené formě, vždy jako hash otisk. Při nasazení aplikace je nutné počítat s použitím šifrovaného protokolu HTTPS.

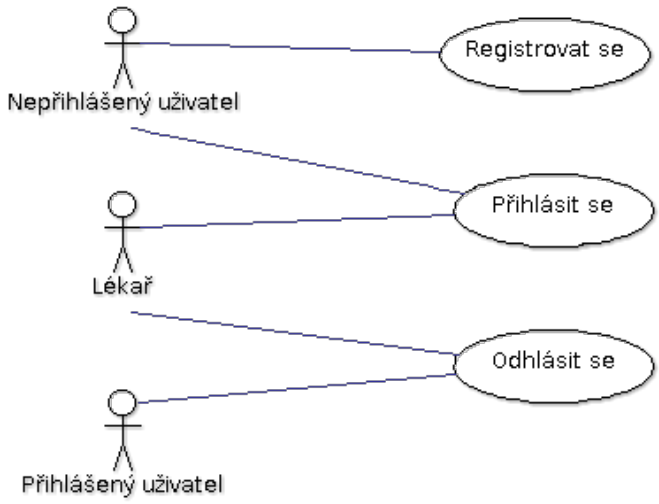
2.3 Případy užití

Následující text popisuje interakci uživatelů se systémem. Jelikož aplikace bude poskytovat veškerou funkcionalitu i ve formě REST API, u každého případu užití bude krátký popis této komunikace. Konkrétně půjde o vstupní bod (entry point), což je URL adresa, na které lze přistupovat ke konkrétním datům. Jako další je definována použitá HTTP metoda. Server pak společně s daty vrací i tzv. stavové kódy (HTTP statusy). Tyto kódy jsou základem komunikace skrze REST API a velmi usnadňují komunikaci mezi serverem a klientem. Dále jsou popsány důležité stavové kódy a jejich význam, které jsou použité v aplikaci.

- 200 Ok - Znamená, že operace proběhla v pořádku. Server jej typicky vrací, pokud klient žádá o nějaká data, nebo došlo k úspěšné změně dat.
- 201 Created - Vrací se při úspěšném vložení nového záznamu.
- 204 No Content - Smazání záznamu bylo úspěšné.
- 400 Bad Request - Server vrací tento kód, pokud data posílaná na server nejsou v pořádku. Obvykle se posílá pokud záznam, který se snažíme vložit do databáze, není validní.
- 401 Unauthorized - Server takto odpoví, pokud klient není oprávněn vykonat danou funkci, ať už jde o výpis dat nebo jejich vložení do databáze.
- 404 Not Found - Vrací se, pokud se klient snaží získat neexistující data.
- 500 Internal Server Error - Neočekávaná chyba na straně serveru.

2.3.1 Přihlášení a registrace

Veškeré funkce jsou dostupně až po registraci a následném přihlášení do aplikace.



Obrázek 2.1: Případy užití - registrace a přihlášení

Registrace

Registrace je volná pro každého návštěvníka aplikace. Jde o jednoduchý webový formulář, ve kterém je nutné zadat všechny následující údaje. Emailovou adresu, jméno, příjmení a heslo. Emailová adresa (společně s heslem) slouží k přihlášení do aplikace a musí být unikátní. Není tedy možné, aby dva uživatelé měli stejnou emailovou adresu. Do budoucna je počítáno s možností zasílat na tento email zapomenuté heslo nebo notifikace o obdržení nové zprávy od lékaře. Dále uživatel musí zadat jméno a příjmení, tento údaj slouží k identifikaci uživatele při psaní zpráv s lékařem.

Během registrace se počítá doporučený denní limit příjmu energie pomocí metody BMR. K tomu je zapotřebí informace o výšce v centimetrech, váze v kilogramech, pohlaví, věkem uživatele a typem aktivity. Vzorec pro výpočet je detailněji popsán v kapitole Realizace 3.5.2 [13]

Pro úspěšné odeslání registrace musí uživatel souhlasit s podmínkami užití. Registrační formulář bude validován na straně serveru i na straně klienta, stejně jako všechny ostatní formuláře.

Entry point	/rest/user/registration
HTTP status	POST
Stavové kódy	201, 400

Tabulka 2.1: Popis REST API - registrace

Přihlášení

Pro přihlášení musí uživatel zadat platnou kombinaci emailu, který vyplnil při registraci a hesla. Počet pokusů pro zadání hesla není jinak omezen.

Entry point	/rest/authenticate
HTTP status	POST
Stavové kódy	200, 401

Tabulka 2.2: Popis REST API - přihlášení

Odhlášení

Odhlásí uživatele ze systému.

Entry point	/rest/logout
HTTP status	GET
Stavové kódy	200

Tabulka 2.3: Popis REST API - odhlášení

2.3.2 Modul konzumace (Kalorický příjem)

Modul konzumace pomáhá uživateli vytvořit si přehled o jeho stravě a zvláště o tom, kolik sacharidů, tuků a bílkovin uživatel zkonzumuje. Historie konzumací je pak důležitá i pro samotného lékaře, který tak může uživatele upozornit na jeho špatnou životosprávu.

Přidat vlastní jídlo

Aplikace disponuje databází téměř sedmi tisíc jídel. Většinou jde o základní potraviny jako je mléko, chleba atd. V případě, že uživatel nenajde potravinu v databázi, může si ji sám vytvořit. Tato potravina je poté viditelná pouze pro uživatele, který potravinu vytvořil.

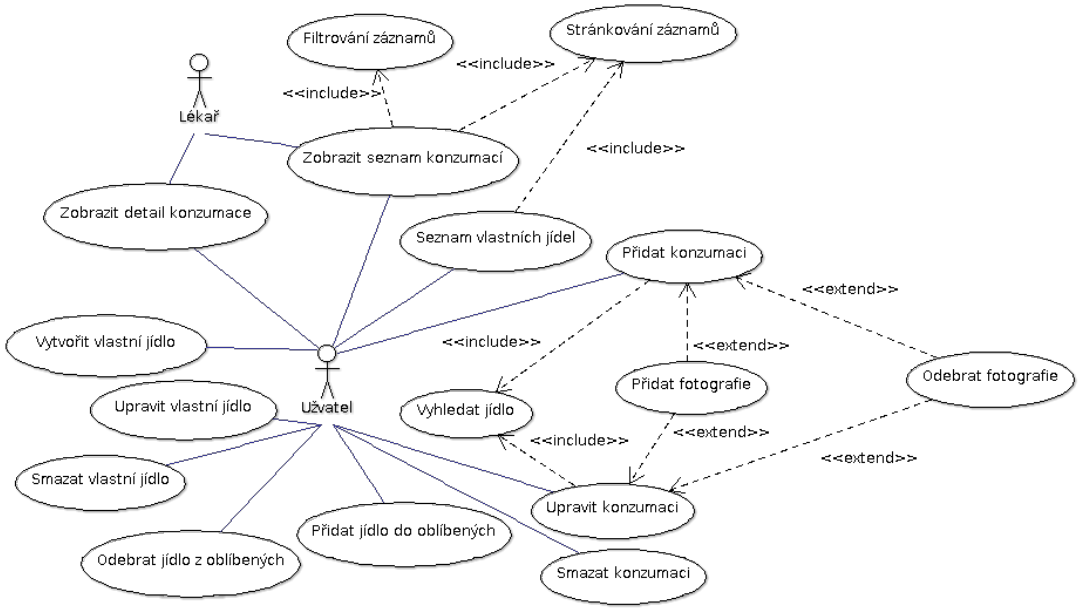
Stojí za úvahu, zda tuto funkcionalitu v budoucnu nerozšířit o možnost zviditelnit takto vytvořenou potravinu i ostatním uživatelům. To by ovšem vyžadovalo před samotným zviditelněním potravinu ručně zkontrolovat.

Entry point	/rest/food
HTTP status	POST
Stavové kódy	201, 400, 401

Tabulka 2.4: Popis REST API - přidání jídla

Upravit vlastní jídlo

Jídlo, které uživatel vytvořil může kdykoliv upravit.



Obrázek 2.2: Případy užití - kalorický příjem

Entry point	/rest/food/{foodId}
HTTP status	PUT
Stavové kódy	200, 400, 401

Tabulka 2.5: Popis REST API - úprava jídla

Smazat vlastní jídlo

Jídlo, které uživatel vytvořil může kdykoliv smazat. Jídlo není možné smazat úplně, protože by mohlo dojít ke ztrátě konzumací, které jsou spojeny s daným jídlem. Jídlo je tedy smazáno pouze na oko, ve skutečnosti dojde k jeho skrytí ve výpisu jídel.

Entry point	/rest/food/{foodId}
HTTP status	DELETE
Stavové kódy	204, 401

Tabulka 2.6: Popis REST API - smazání jídla

Seznam vlastních jídel

Uživatel svá jídla vidí v tabulce. Nepředpokládá se, že by uživatelé vytvářeli velké množství jídel. Z toho důvodu je tabulka pouze stránkovaná. V případě, že by uživatelé přeci jen vytvářeli velké množství jídel, není problém implementovat vyhledávání podle názvu jídla.

Entry point	/rest/food/{foodId}
HTTP status	GET
Stavové kódy	200, 401

Tabulka 2.7: Popis REST API - seznam vlastních jídel

Přidat konzumaci

Zadáváme datum konzumace. Předpokládá se, že někteří uživatelé budou veškeré měření a konzumace zadávat do systému zpětně. Například data z celého dne vyplní najednou až večer. Dále je potřeba vybrat jídlo, které uživatel zkonzumoval a typ jídla. Jídel je velké množství. Z toho důvodu lze jídlo vyhledat pomocí fulltextového vyhledávání. Jídla lze filtrovat podle kategorií, či oblíbených jídel. Typem jídla se myslí výběr z následujících možností: snídaně, svačina dopoledne, oběd, svačina odpoledne, večeře, pozdní večere a ostatní. Nakonec uživatel musí zadat zkonzumované množství a nepovinně poznámku.

V mobilní aplikaci lze ke konzumaci připojit libovolný počet fotografií. Vize je taková, že uživatel vyfotí svůj pokrm a ten nahraje na server. Webová aplikace poskytuje rozhraní v podobě REST API pro nahrávání a mazání fotografií. Samotné nahrávání fotografií z webového klienta není implementováno jelikož postrádá smysl.

Entry point	/rest/user/{id}/eating
HTTP status	POST
Stavové kódy	201, 400, 401

Tabulka 2.8: Popis REST API - přidání konzumace

Upravit konzumaci

U konzumace lze upravovat veškerá data, která byla vyplněna při přidání konzumace. Přes REST API lze ke konzumaci nahrávat nové fotografie a zároveň lze mazat již nahrané.

Entry point	/rest/user/{id}/eating
HTTP status	PUT
Stavové kódy	200, 400, 401

Tabulka 2.9: Popis REST API - úprava konzumace

Smazat konzumaci

Smazání konzumace je nevratné. Pokud byly ke konzumaci nahrány fotografie, jsou rovněž smazány.

Entry point	/rest/user/{id}/eating
HTTP status	DELETE
Stavové kódy	204, 401

Tabulka 2.10: Popis REST API - mazání konzumace

Přidat jídlo do oblíbených

Pro rychlejší přidání konzumace má uživatel možnost uložit jídlo do oblíbených. Kromě jídla se ukládá typ a množství. Uživatel tak u často zadávaných jídel nemusí vše znovu vyplňovat, stačí mu pouze vybrat jídlo z oblíbených a hodnoty se sami předvyplní do formuláře pro přidání konzumace.

Entry point	/rest/user/{userId}/food/favorite
HTTP status	POST
Stavové kódy	201, 400, 401

Tabulka 2.11: Popis REST API - přidání jídla do oblíbených

Odebrat jídlo z oblíbených

Oblíbená jídla lze smazat.

Entry point	/rest/user/{userId}/food/favorite/{favoriteId}
HTTP status	DELETE
Stavové kódy	204, 401

Tabulka 2.12: Popis REST API - odebrání jídla z oblíbených

Zobrazit seznam konzumací

Uživatel má na výběr ze dvou možností, jak data zobrazit. První možností je zobrazení v tabulce. Záznamy jsou řazeny sestupně podle data vložení, jsou stránkované a data je možné zobrazit pouze v určitému časovém období. Minimální časový rozsah je 1 den. Maximální rozsah je v podstatě neomezený.

Druhý způsob je zobrazení záznamů v grafu. Zde je rozdíl, zda zobrazujeme data v jednom dni, nebo data za delší časový úsek. Zatímco u záznamů přes více dnů se zobrazí koláčový graf s poměrem sacharidů, tuků a bílkovin, U záznamů z jednoho dne se zobrazí celkové zkonzumované množství jednotlivých složek. Uživatel rovnou vidí, jaké je jeho doporučené množství jednotlivých složek a kolik je jeho aktuální množství. Pro lepší představu si prohlédněte obrázek C.2, kde je znázorněn tento graf.

Entry point	/rest/user/{id}/eating
HTTP status	GET
Stavové kódy	200, 401, 404

Tabulka 2.13: Popis REST API - zobrazení seznamu konzumací

Zobrazit detail konzumace

U konzumace je poměrně velké množství informací a není tedy dobrý nápad všechny tyto informace zobrazovat v tabulce. Tabulka by se tím stala nepřehledná. Z toho důvodu ta-

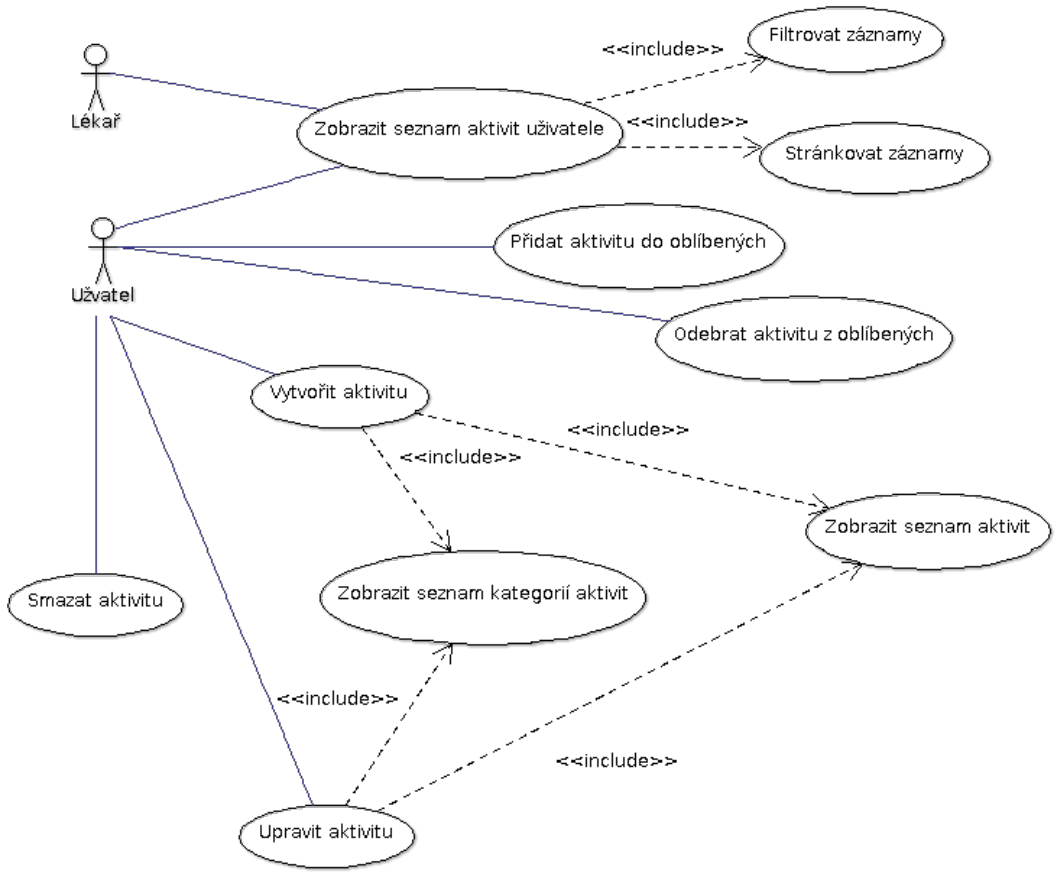
bulka obsahuje základní informace, jako je zkonzumované jídlo, typ a množství jídla. Zbytek informací se nachází v detailu konzumace.

Entry point	/rest/user/{userId}/eating/{eatingId}
HTTP status	GET
Stavové kódy	200, 401, 404

Tabulka 2.14: Popis REST API - zobrazení konzumace

2.3.3 Modul aktivit (Kalorický výdej)

Kalorický výdej slouží jako přehled aktivit uživatele a jeho kalorický výdej. Společně s ostatními moduly tak dává lékaři náhled na životní styl pacienta.



Obrázek 2.3: Případy užití - kalorický výdej

Přidání aktivity

Jak již bylo řečeno, aplikace obsahuje databázi aktivit rozdělených do kategorií. Uživatel tedy vybírá z těchto předdefinovaných kategorií. Pro lepší orientaci lze aktivity filtrovat právě podle kategorie. U aktivity se také zadává její délka, přesněji čas začátku a konce aktivity. Nakonec je možné vložit k aktivitě libovolnou poznámku.

Entry point	/rest/user/{id}/activity
HTTP status	POST
Stavové kódy	201, 400, 401

Tabulka 2.15: Popis REST API - přidání aktivity

Úprava aktivity

Zadanou aktivitu lze kdykoliv upravit.

Entry point	/rest/user/{id}/activity/{activityId}
HTTP status	PUT
Stavové kódy	200, 400, 401

Tabulka 2.16: Popis REST API - úprava aktivity

Smazání aktivity

Aktivitu lze natrvalo smazat.

Entry point	/rest/user/{id}/activity/{activityId}
HTTP status	DELETE
Stavové kódy	204, 401

Tabulka 2.17: Popis REST API - mazání aktivity

Přidání aktivity do oblíbených

Stejně jako u konzumace, i aktivity lze přidat mezi oblíbené. Zde se přidává pouze samotná aktivita. Uživatel pak může filtrovat jeho oblíbené aktivity.

Entry point	/rest/user/{id}/activity/favorite
HTTP status	POST
Stavové kódy	201, 400, 401

Tabulka 2.18: Popis REST API - přidání oblíbené aktivity

Odebrání aktivity z oblíbených

Oblíbené aktivity lze i odebírat.

Entry point	/rest/user/{userId}/activity/{activityId}/favorite
HTTP status	DELETE
Stavové kódy	204, 401

Tabulka 2.19: Popis REST API - odebrání oblíbené aktivity

Zobrazit seznam aktivit uživatele

Aktivity lze zobrazit v tabulce, která je opět stránkovaná. Kromě dat vyplněných při vkládání aktivity se v tabulce zobrazuje množství vydané energie (v kJ) po dobu trvání aktivity. Záznamy lze filtrovat podle data v rozsahu od - do. Na rozdíl od jiných modulů má aktivita začátek a konec. Filtrování bylo nutné nastavit tak, aby filtr "od"zohledňoval začátek aktivity a filtr "do"její konec.

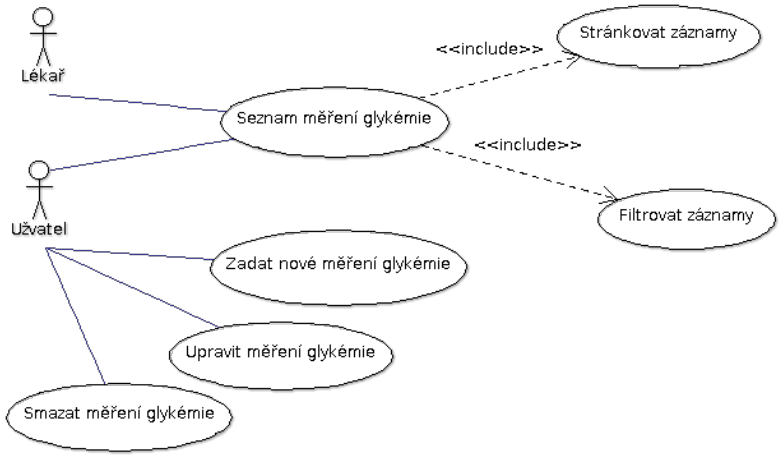
Data mohou být zobrazena i ve formě grafu. Jde o sloupcový graf, kde osa x představuje čas a osa y množství vydané energie v jedné hodině. Pokud uživatel jezdil na kole 3 hodiny a spálil 9 000 kJ. V grafu se tato skutečnost zobrazí jako tři sloupce s hodnotou 3 000 Kj. V případě, že v jedné hodině má uživatel více aktivit, hodnoty se sčítají.

Entry point	/rest/user/{id}/activity
HTTP status	GET
Stavové kódy	200, 401

Tabulka 2.20: Popis REST API - seznam aktivit uživatele

2.3.4 Modul glykémie

Jde o poměrně jednoduchý modul, jehož cílem je sledovat hladinu glykémie uživatele/pacienta. Glykémie udává množství cukru v krvi. Sledování glykémie u diabetiků je vzhledem k jejich nemoci velmi důležité. Podle naměřené hodnoty se určuje množství aplikovaného inzulínu, případně množství výměnných jednotek, které musí uživatel zkonzumovat. [5]



Obrázek 2.4: Případy užití - glykéme

Přidání měření glykémie

Uživatel vyplní datum měření, naměřenou hodnotu v jednotkách milimol na litr (mmol/l) a nepovinně poznámku. Měření typicky probíhá před jídlem a před spaním. Aplikace proto uživateli nabízí několik možností, jak předvyplnit poznámku. Jde o možnosti: na lačno, před jídlem, po jídle a před spaním.

Entry point	/rest/user/{id}/glycemia
HTTP status	POST
Stavové kódy	201, 400, 401

Tabulka 2.21: Popis REST API - přidání měření glykémie

Úprava měření glykémie

Vloženou hodnotu lze upravit.

Entry point	/rest/user/{userId}/glycemia/{glycemiaId}
HTTP status	PUT
Stavové kódy	200, 400, 401

Tabulka 2.22: Popis REST API - úprava měření glykémie

Úprava měření glykémie

Vloženou hodnotu lze smazat. Obecně u všech modulů platí, že aplikace se před samotným smazáním uživatele zeptá, zda chce záznam opravdu smazat. Stojí za úvahu, zda mazání záznamu po uplynutí určitého časového úseku nezakázat.

Entry point	/rest/user/{userId}/glycemia/{glycemiaId}
HTTP status	PUT
Stavové kódy	204, 401

Tabulka 2.23: Popis REST API - smazání měření glykémie

Seznam měření glykémie

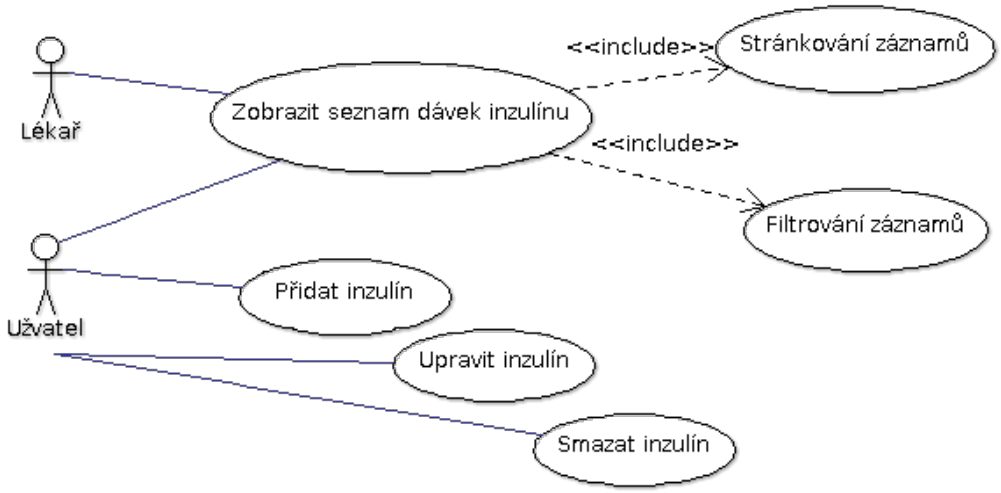
Záznamy v tabulce jsou seřazené podle data vložení. Jeden řádek tabulky obsahuje veškerá data o měření. Záznamy lze filtrovat podle data měření. K zobrazení glykémie v grafu se jako nejlepší volba jeví použít čárový graf, kde osa x představuje datum a čas měření, osa y pak naměřenou hodnotu.

Entry point	/rest/user/{userId}/glycemia
HTTP status	GET
Stavové kódy	200, 401

Tabulka 2.24: Popis REST API - seznam měření glykémie

2.3.5 Modul inzulínu

Slinivka břišní zdravého člověka produkuje inzulín. Inzulín pomáhá udržovat hodnotu glykémie přibližně v rozmezí 3,3 - 6 mmol/l. Slinivka diabetika 1. typu neprodukuje žádný inzulín, nebo jej produkuje velmi málo. Diabetici 1. typu si sami musí aplikovat inzulín. Tento modul pak slouží jako přehled aplikovaného inzulínu v průběhu času. [2]



Obrázek 2.5: Případy užití - inzulín

Přidání dávky inzulínu

Kromě data aplikace uživatel zadává typ inzulínu. Máme tři typy:

- bazální inzulín - Dlouhodobá dávka inzulínu na noc.
- bolusový inzulín - Nárazová dávka inzulínu podávaná před jídlem. [10]
- dopichy rychlým inzulínem - Dávky inzulínu, které se aplikují podle potřeby během dne, typicky při hyperglykémii.

Dále zadává konkrétní typ inzulínu. V dnešní době existuje několik typů inzulínu lišících se v rychlosti nástupu a době trvání. Nakonec zadává počet jednotek inzulínu a nepovinně popis.

Entry point	/rest/user/{id}/insulin
HTTP status	POST
Stavové kódy	201, 400, 401

Tabulka 2.25: Popis REST API - přidání inzulínu

Úprava dávky inzulínu

Dávku inzulínu lze upravit.

Entry point	/rest/user/{userId}/insulin/{insulinId}
HTTP status	PUT
Stavové kódy	200, 400, 401

Tabulka 2.26: Popis REST API - úprava inzulínu

Smazání dávky inzulínu

Dávku inzulínu lze smazat.

Entry point	/rest/user/{userId}/insulin/{insulinId}
HTTP status	DELETE
Stavové kódy	204, 401

Tabulka 2.27: Popis REST API - smazání inzulínu

Zobrazení dávek inzulínu

Dávky lze prohlížet v tabulce a grafu. Zobrazení v tabulce se nijak zásadně neliší od ostatních modulů. Data v grafu jsou opět reprezentována čárovým grafem. V grafu se zvlášť zobrazují jednotlivé typy inzulínu. Na ose x máme čas a na ose y aplikovanou hodnotu.

Entry point	/rest/user/{userId}/insulin
HTTP status	GET
Stavové kódy	200, 401

Tabulka 2.28: Popis REST API - zobrazení dávek inzulínu

2.3.6 Modul krevního tlaku

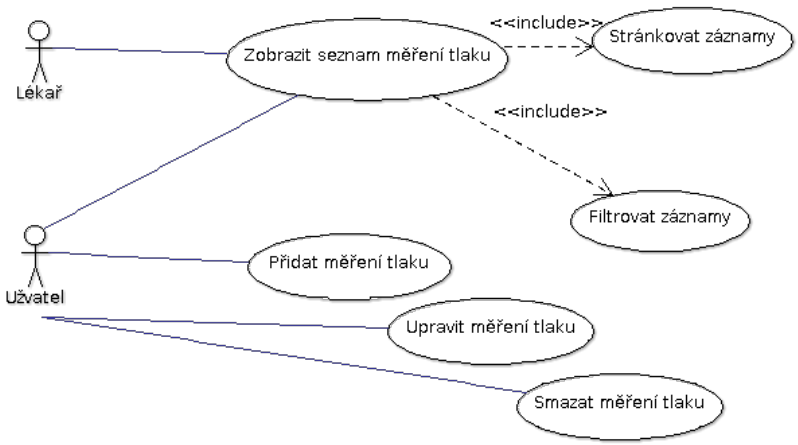
Krevní tlak může souviset s nadváhou, která bývá příčinou vzniku diabetu 2. typu. Hodnota krevního tlaku může mít vliv na pozdější rozvinutí komplikací diabetu.

Přidání měření krevního tlaku

Uživatel zadá datum a čas měření, systolický tlak, diastolický tlak, puls a nepovinně poznámku.

Entry point	/rest/user/{id}/pressure
HTTP status	POST
Stavové kódy	201, 400, 401

Tabulka 2.29: Popis REST API - přidání měření krevního tlaku



Obrázek 2.6: Případy užití - krevní tlak

Úprava měření krevního tlaku

Měření lze upravit. V mobilní aplikaci je možné spárovat telefon s tlakoměrem (pokud to tlakoměr umožňuje) a poté data posílat z tlakoměru do telefonu a na server. Stojí za úvahu, zda takto získané záznamy může uživatel upravovat. Úprava záznamů je určená k tomu, aby uživatel mohl opravit případnou chybu či překlep, který vznikl při zadávání, to u tlakoměru nehrozí.

Entry point	/rest/user/{userId}/pressure{pressureId}
HTTP status	PUT
Stavové kódy	200, 400, 401

Tabulka 2.30: Popis REST API - úprava měření krevního tlaku

Smazání měření krevního tlaku

Měření lze smazat.

Entry point	/rest/user/{userId}/pressure{pressureId}
HTTP status	DELETE
Stavové kódy	204, 401

Tabulka 2.31: Popis REST API - smazání měření krevního tlaku

Seznam měření krevního tlaku

Stejně jako u ostatních modulů lze měření zobrazit ve stránkované tabulce a v grafu. Jeden záznam v tabulce obsahuje veškeré zadané údaje. V grafu osa x představuje čas a na ose y

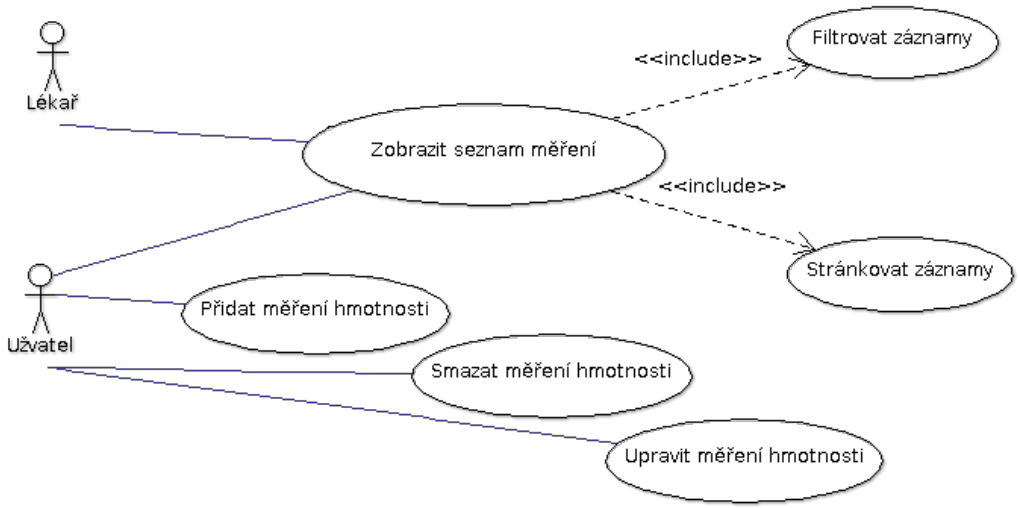
jsou naměřené hodnoty systolického tlaku, diastolického tlaku a naměřený puls. K zobrazení dat je použit čárový graf. Záznamy je možné filtrovat podle data vložení.

Entry point	/rest/user/{id}/pressure
HTTP status	GET
Stavové kódy	200, 401

Tabulka 2.32: Popis REST API - seznam měření krevního tlaku

2.3.7 Modul hmotnosti

Tento modul má smysl hlavně pro diabetiky 2. typu, kteří by měli dodržovat dietu.



Obrázek 2.7: Případy užití - hmotnost

Přidání měření hmotnosti

Kromě samotné hmotnosti v kilogramech musí uživatel vyplnit obvod pasu v centimetrech, datum a čas měření a nepovinně popis. Hmotnost zadaná v tomto modulu nemá vliv na hmotnost zadanou při registraci.

Entry point	/rest/user/{id}/weight
HTTP status	POST
Stavové kódy	201, 400, 401

Tabulka 2.33: Popis REST API - přidání měření hmotnosti

Úprava měření hmotnosti

Měření je možné upravit.

Entry point	/rest/user/{id}/weight
HTTP status	PUT
Stavové kódy	200, 400, 401

Tabulka 2.34: Popis REST API - úprava měření hmotnosti

Smazání měření hmotnosti

Měření je možné smazat.

Entry point	/rest/user/{id}/weight
HTTP status	DELETE
Stavové kódy	204, 401

Tabulka 2.35: Popis REST API - smazání měření hmotnosti

Seznam měření hmotnosti

Záznamy měření je možné procházet v tabulce nebo čárovém grafu. Osa x reprezentuje čas a osa y váhu a obvod pasu. Pro váhu a obvod máme dvě čáry v grafu.

Entry point	/rest/user/{id}/weight
HTTP status	DELETE
Stavové kódy	204, 401

Tabulka 2.36: Popis REST API - seznam měření hmotnosti

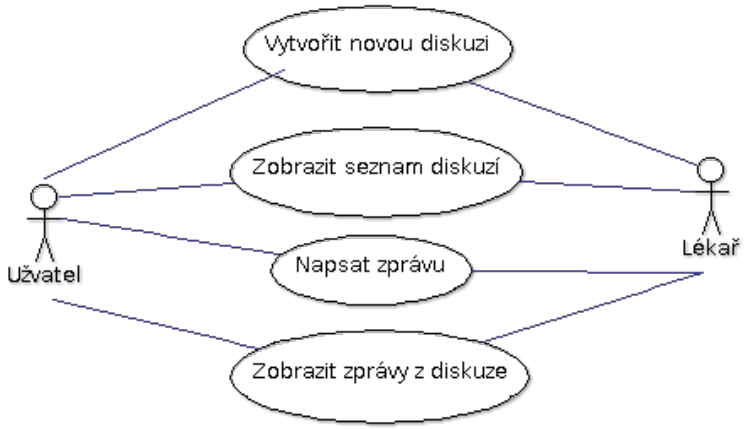
2.3.8 Zprávy

Pomocí zpráv může pacient komunikovat s lékařem. Může se lékaře ptát na různé otázky související s jeho nemocí nebo naopak lékař může pacienta skrze zprávu motivovat k vyplňování údajů.

Vytvořit novou diskuzi

Zprávy jsou členěny do samotných diskuzí. Konkrétní problém by se měl řešit v jedné diskuzi a pro další problémy a otázky je doporučeno vytvořit vlastní diskuze.

Při zakládání nové diskuze uživatel vyplní předmět, což je krátký popis toho, čeho se diskuze týká. Společně s vytvořením diskuze uživatel vkládá i první zprávu. Do uživatelem vytvořené diskuze se automaticky přidá jeho lékař. V případě, že diskuzi zakládá sám lékař, musí navíc do diskuze přidat konkrétního pacienta.



Obrázek 2.8: Případy užití - zprávy

Entry point	/rest/user/{id}/discussion
HTTP status	POST
Stavové kódy	201, 400, 401

Tabulka 2.37: Popis REST API - vytvoření nové diskuze

Zobrazit seznam diskuzí

Uživateli se zobrazují všechny diskuze, do kterých byl přidán. Kliknutím na diskuzi zobrazí všechny její zprávy.

Entry point	/rest/user/{id}/discussion
HTTP status	GET
Stavové kódy	200, 401

Tabulka 2.38: Popis REST API - zobrazení diskuzí

Napsat zprávu

Napsat zprávu do diskuze může pouze ten, kdo byl do diskuze přidán.

Entry point	/rest/user/{userId}/discussion/{discussionId}/message
HTTP status	POST
Stavové kódy	201, 400, 401

Tabulka 2.39: Popis REST API - napsat zprávu

Zobrazit zprávy z diskuze

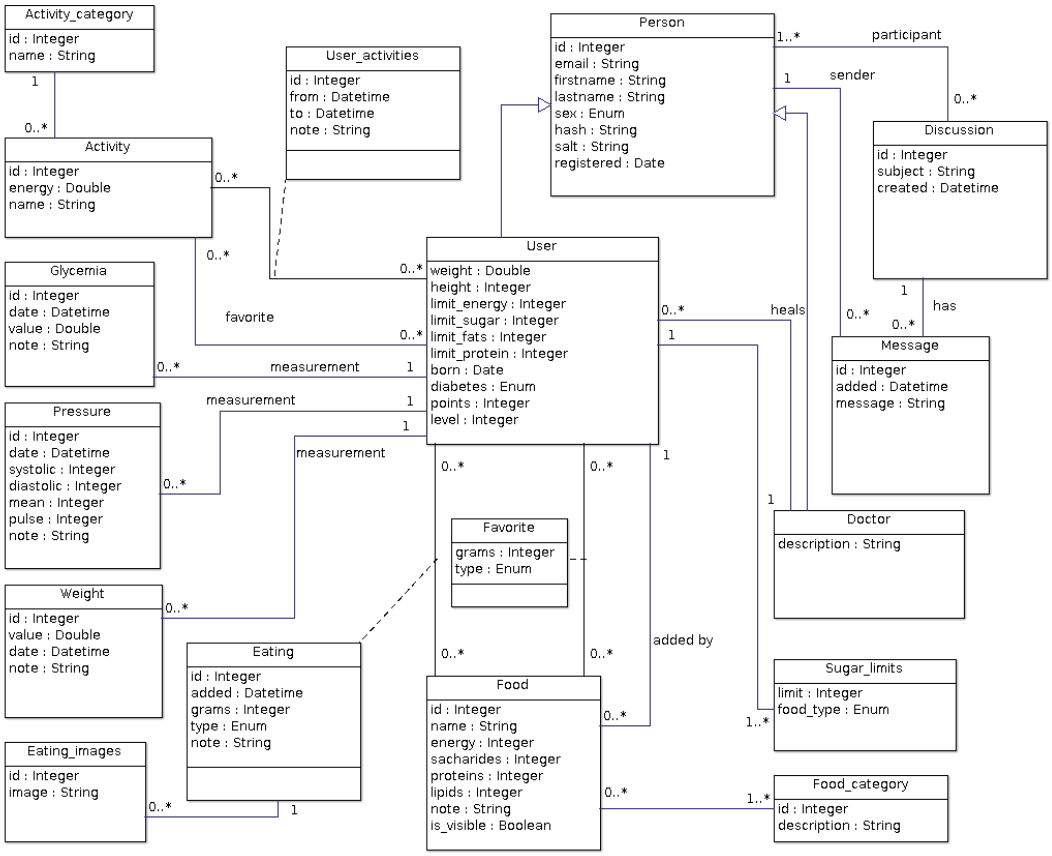
Detail diskuze obsahuje seznam zpráv seřazených vzestupně podle času vytvoření. Zprávy mohou vidět pouze účastníci diskuze.

Entry point	/rest/user/{userId}/discussion/{discussionId}/message
HTTP status	GET
Stavové kódy	200, 401

Tabulka 2.40: Popis REST API - zprávy z diskuze

2.4 Doménový model

Zde vidíme schéma aplikace. Toto schéma posloužilo jako návrh databáze.

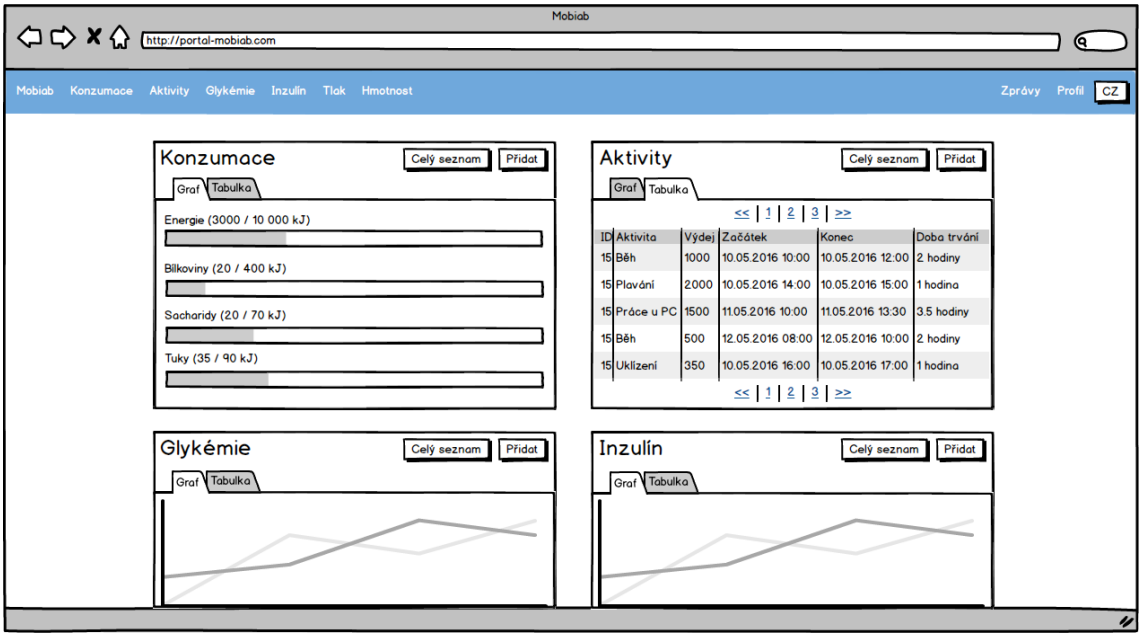


Obrázek 2.9: Doménový model

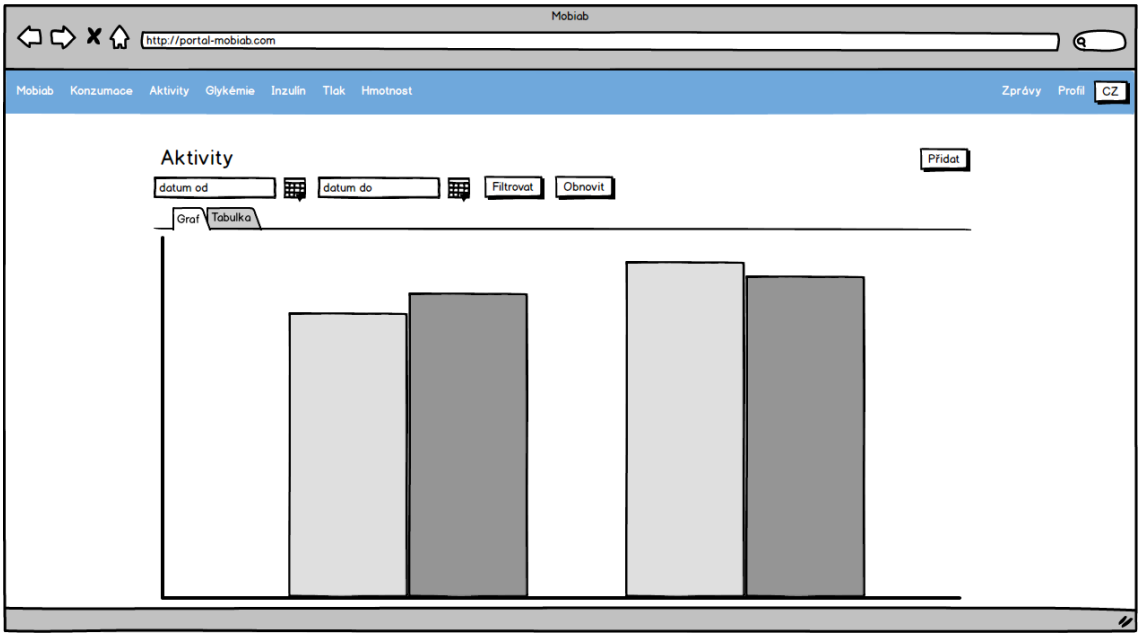
2.5 Návrh vzhledu aplikace

Při návrhu grafického rozhraní jsem se zaměřil na to, aby byl vzhled aplikace co nejjednodušší s maximálním důrazem na dobrou čitelnost dat.

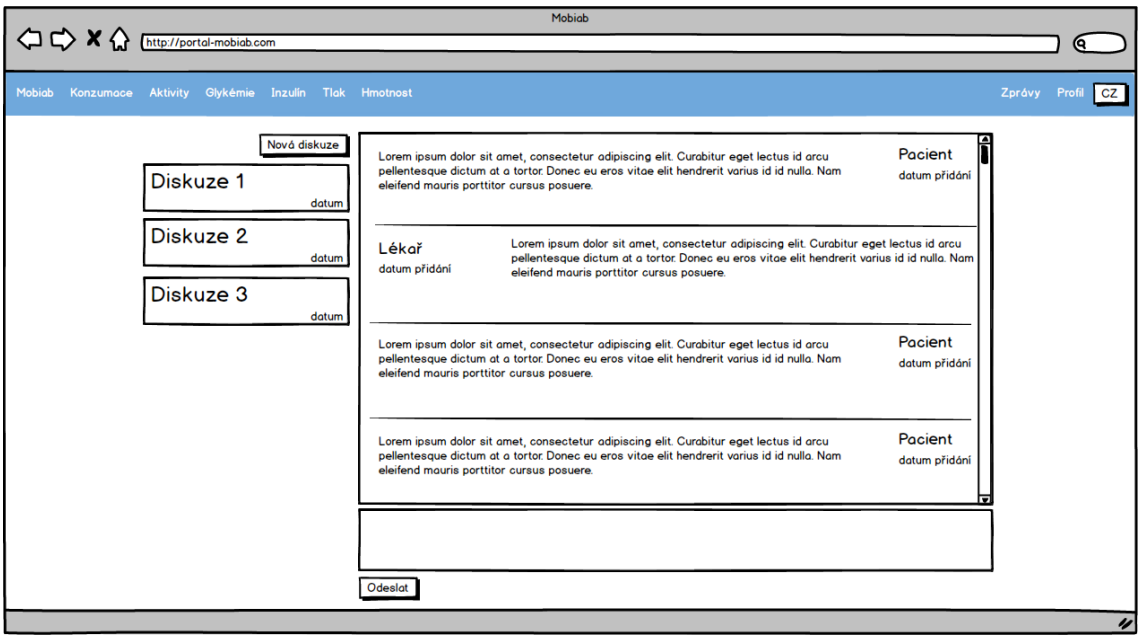
Důležitým prvkem celé aplikace je tzv. Dashboard. Jde o kompletní přehled záznamů přes všechny moduly. Níže je ukázka vybraných grafických návrhů.



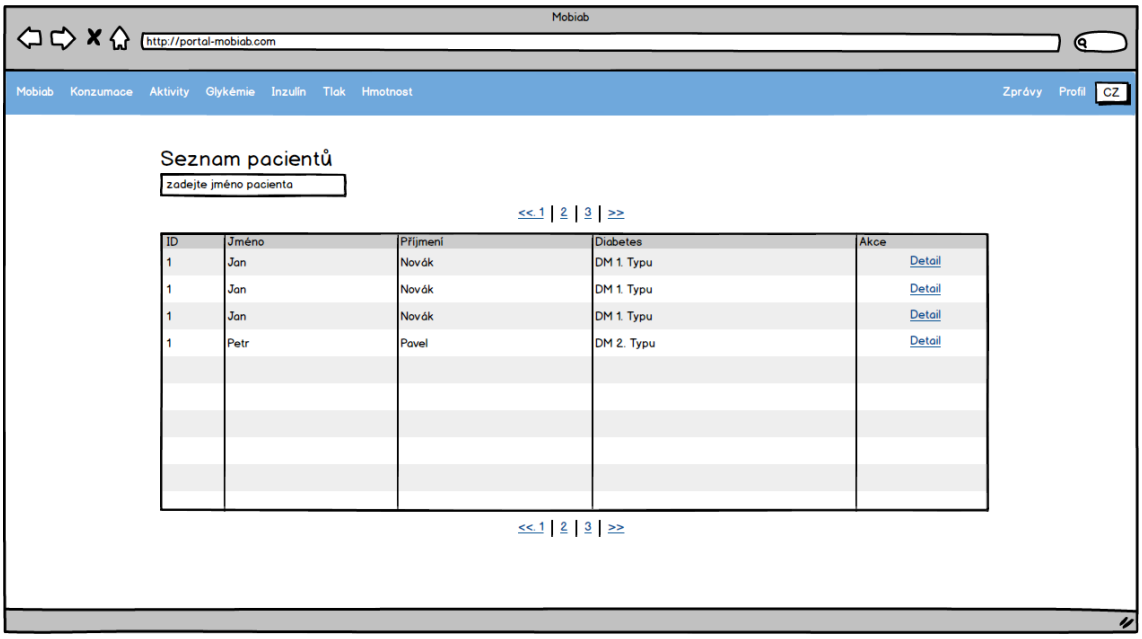
Obrázek 2.10: Grafické rozhraní - Dashboard



Obrázek 2.11: Grafické rozhraní - Modul aktivit



Obrázek 2.12: Grafické rozhraní - Zprávy



Obrázek 2.13: Grafické rozhraní - Seznam pacientů

Kapitola 3

Realizace

Na začátku kapitoly bude popis použitých technologií při implementaci. Jako další bude popsána architektura celé aplikace. Poté zde bude rozebrána databáze a zajímavé implementační detaily. Nakonec se budu věnovat nasazení aplikace.

3.1 Použité technologie

Popis technologií je rozdělen do dvou celků. Technologie na straně serveru a technologie na straně klienta. Všechny použité technologie jsou poskytovány jako Open Source. Za jejich použití programátor nemusí platit. Aplikace byla vyvíjená v Eclipse, což je asi nejrozšířenější nástroj pro tvorbu webových aplikací napsaných v Javě.

3.1.1 Technologie na straně serveru

3.1.1.1 Java

Java je silně objektově orientovaný programovací jazyk. V poslední době se hojně používá k tvorbě komplexních webových aplikací jako jsou například bankovní systémy. Pro tvorbu webových aplikací vznikla Java EE, neboli Enterprise Edition. Jde o rozšíření Javy SE (Standard Edition) o technologie a knihovny podporující tvorbu vysoce škálovatelných, vícevrstvých, dostupných a vysoce zabezpečených webových aplikací. Aktuální verze Javy EE je verze 7. [11]

Klíčovým prvkem Javy EE jsou Servlety. Jedná se o rozhraní pro zpracování HTTP požadavků a následné vytvoření HTTP odpovědi. Pro člověka, který s Javou EE začíná je poměrně těžké se v dané problematice zorientovat. Před začátkem implementace je nutné nakonfigurovat plno věcí jako je například spojení s databází, mapování servletů na konkrétní url adresu, nastavení správného kódování, a mnoho dalších věcí, mezi které patří i konfigurace aplikačního serveru, abychom mohli webovou aplikaci otestovat.

3.1.1.2 Spring

Spring je momentálně asi nejrozšířenější framework (aktuálně ve verzi 4.2.6) postavený nad Javou EE, který usnadňuje a urychluje implementaci webových aplikací. Framework je roz-

dělen do několika samostatných modulů. Máme zde například modul pro přístup k datům kam patří spojení s databází, nebo webový modul. Ten zahrnuje knihovny pro tvorbu webových aplikací pomocí MVC architektury.

Spring obrovsky usnadňuje práci i díky podpoře tzv. Dependency Injection (zkráceně DI). Při tvorbě aplikace nám po čase vznikají závislosti mezi komponentami. DI je technika, jak tyto závislosti co nejvíce eliminovat. Například tím, že vytváření závislostí přesuneme mimo komponenty. K tomu účelu ve Springu slouží IoC container. IoC, neboli Inversion of Control je obecná technika Dependency Injection.

Další silná stránka Springu spočívá v zabezpečení. Každá větší webová aplikace obsahuje funkce, které jsou přístupné pouze uživatelům s určitými právy. Spring Security se zaměřuje na autentizaci a autorizaci uživatelů. Vývojáři poskytuje rychlý způsob, jak zabezpečit aplikaci. [17] [24]

Spring obsahuje knihovny pro tvorbu webových služeb jako je REST, i to byl jeden z důvodů, proč je zde popisovaná aplikace postavená na Springu.

3.1.1.3 MySQL

MySQL je jeden nejrozšířenějších Open Source databázových systémů. Data z původní aplikace byla uložena právě v MySQL, proto byla volba poměrně jasná.

3.1.1.4 Hibernate

Hibernate je framework, který zajišťuje konverzi dat mezi relační databází jako je MySQL a Javou. Obecně se tato technika nazývá ORM, neboli Object Relational Mapping. Nejaktuálnější verze je 5.1.0. Hibernate je vlastně implementací Java Persistence API, což je standard pro správu relačních dat v jazyce Java. Kromě relačního mapování Hibernate nabízí vysokou škálovatelnost, což je jeden z požadavků kladených na vyvíjenou aplikaci. [19]

Dvojice Spring a Hibernate je v Java světě velmi častá. Integrace těchto dvou frameworků je tedy prakticky bez problémů.

3.1.1.5 Maven

Maven je nástroj na správu softwarových projektů. Usnadňuje a sjednocuje buildovací proces. Jeho nejsilnější stránkou je správa externích knihoven. Projekt spravovaný Mavenem je popsán pomocí tzv. Project Object Model. Jde o soubor, ve kterém lze definovat externí knihovny (jako je třeba Spring nebo Hibernate). Tyto knihovny Maven automaticky stáhne z úložiště (repozitáře) a to včetně jejich závislostí. [20]

Repozitář je databáze knihoven, kde každá knihovna je definovaná podle groupId a artifactId. GroupId je unikátní identifikátor projektu. ArtifactId je jméno jar souboru. Kromě oficiálního centrálního repozitáře existuje plno menších, například firemních repozitářů. [21] [15]

3.1.2 Technologie na straně klienta

3.1.2.1 HTML 5

HTML, neboli HyperText Markup Language je naprostý základ každé webové aplikace. Jde o značkovací jazyk jehož základem jsou tzv. tagy. Pomocí těchto tagů definujeme strukturu HTML dokumentu. Každý tag má nějaký význam. Třeba tag <p> je určený pro víceřádkový text, <nav> se používá pro označení menu. Význam tagů je důležitý hlavně pro vyhledávače, které tak mohou určit jaké informace na stránce jsou důležité a které nikoliv. U tagů lze definovat atributy, ty pak mohou ovlivňovat chování tagu. [26]

Poslední verzí HTML je verze 5. Ta přinesla mnoho nového, byly definovány nové tagy, některé tagy byly zrušeny. HTML 5 přišlo s podporou WebSockets, což je technologie určená pro obousměrnou komunikaci mezi klientem a serverem. Při komunikaci přes HTTP klient posílá požadavky na server a ten mu odpovídá. Klient tedy vždy začíná komunikaci, která je navíc bezestavová. Naproti tomu při použití WebSockets se mezi klientem a serverem vytvoří spojení, které trvá dokud jej klient neukončí. Díky tomu může server notifikovat konkrétního klienta. Tato vlastnost se hodí při implementaci chatovacích aplikací. [8]

3.1.2.2 Bootstrap

Bootstrap je frond-end framework, který usnadňuje práci při tvorbě vzhledu aplikace. Obsahuje množství hotových komponent jako jsou tabulky, tlačítka, navigace a formuláře. Dokonce pomáhá při tvorbě layoutu. Tyto komponenty jsou kompatibilní přes všechny moderní prohlížeče a navíc jsou plně responzivní. To znamená, že se automaticky přizpůsobují velikosti obrazovky. [25]

Framework je postavený na CSS, neboli kaskádových stylech. Jde o jednoduchý jazyk pro popis vzhledu aplikace. Pomáhá oddělit strukturu HTML dokumentu od jeho vzhledu a tím zvyšuje přehlednost a udržitelnost celé aplikace.

3.1.2.3 Angular

Angular je framework založený na architektuře MVC. Je postavený nad skriptovacím jazykem JavaScript. Narozdíl od Javy se JavaScript a tedy i Angular vykonává na straně klienta. Díky tomu lze vytvářet vysoce dynamické webové aplikace. Silnou stránkou Angularu je jeho automatická synchronizace dat mezi modelem a view.

Dalším zajímavým konceptem jsou direktivy. Pomocí direktiv lze rozšířit jinak statické HTML o dynamické prvky. Kromě předem nadefinovaných direktiv lze vytvářet i vlastní direktivy. Často používanou direktivou je například ng-click. Ta se používá k volání určité funkce kontroleru po kliknutí na HTML element spjatý s direktivou. I Angular poskytuje řešení Dependency Injection.

Rozhodnutí použít Angular pro implementaci padlo hlavně z důvodu, že serverová část poskytuje kompletní funkcionalitu skrze REST API. Angular poskytuje rozhraní pro volání HTTP požadavků a automaticky deserializuje přijatá data do JavaScriptového objetu. To obrovsky usnadňuje práci programátora. Navíc není nutné duplikovat logiku kontrolerů na straně severu. [18]

3.1.2.4 Google charts

Vizualizace dat pomocí grafů je základem celé aplikace, proto měl výběr vhodné knihovny vysokou prioritu. Nakonec jsem se rozhodl pro Google Charts a to hlavně díky velkému výběru grafů a dobře zpracované dokumentaci.

3.2 Architektura

V následujícím textu bude popis použité architektury. Nejprve bude popsán princip architektury klient server. Nakonec bude popsána architektura z pohledu serveru a poté z pohledu klienta.

3.2.1 Klient-Server

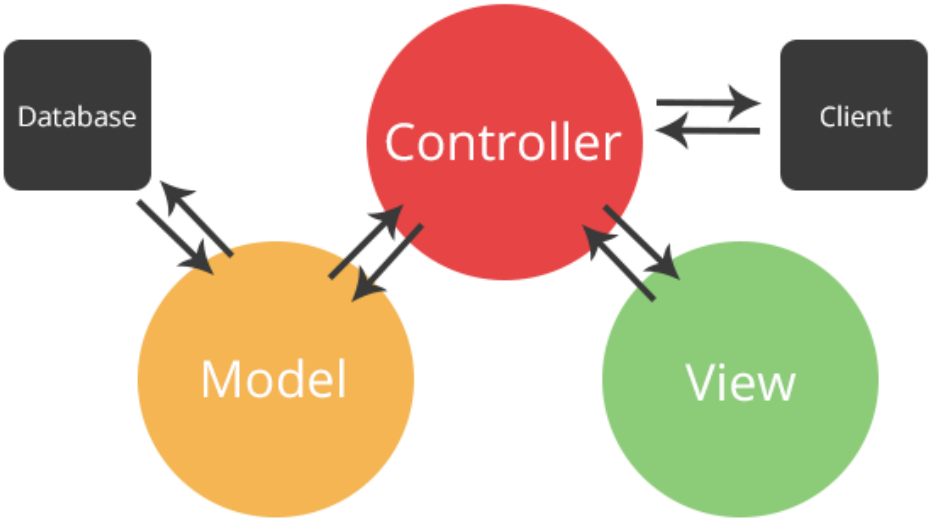
Webové aplikace jsou typickým příkladem klient-server architektury. Někdy se tato architektura označuje jako producer-consumer architektura. Server poskytuje služby směrem ke klientům. Tyto služby jsou poskytovány na požádání, což znamená, že komunikaci vždy začíná klient. Službou může být prakticky cokoliv. Například uložení dat, jejich úprava nebo výpis a dokonce i poskytnutí výpočetního výkonu lze označit jako služba. [1]

3.2.2 Model-View-Controller

MVC architektura rozděluje aplikaci na tři hlavní celky. Tyto celky jsou na sobě nezávislé, což programátorům usnadňuje vývoj a následné testování aplikace a dokonce i údržbu aplikace. Díky nezávislosti je možné každý z celků upravit, aniž by bylo potřeba kvůli tomu nějak zásadně upravovat zbývající celky. Následuje popis jednotlivých celků.

- Model - Představuje datovou vrstvu aplikace. Kromě přístupu k datům se stará o jejich zabezpečení a de facto tvořit bussiness model celé aplikace.
- View - Jinak také pohledová vrstva, představuje prezentaci dat směrem k uživateli. Je to tedy ta část aplikace, kterou uživatel vidí na svém monitoru.
- Controller - Reaguje na události (které obvykle pochází od uživatele) a případně volá akce na modelu a následně posílá data do view. Obecně lze říct, že jde o zprostředkovatele mezi modelem a view.

Existuje velké množství variant, ale obecně lze popsat fungování MVC takto: Uživatel zavolá příslušnou akci kontroleru, například odešle vyplněný formulář. Kontroler nejprve validuje přijatá data a pokud jsou validní, předá je do modelu, kde se naváže spojení s databází a poté se data uloží. Nakonec kontroler vybere view, který se má zobrazit. Schéma MVC je vidět na obrázku 3.1 [22]



Obrázek 3.1: Schéma Model View Controller

3.2.2.1 Spring MVC

MVC ve Springu je založeno na tzv. DispatcherServletu. Jde o normální Java Servlet na který jsou delegovány všechny HTTP požadavky. Tento servlet pak podle předem stanovených kritérií vybere konkrétní řídící třídu a metodu, kterou nakonec vykoná. Povinným kritériem je URL adresa, ale metody lze rozlišovat i podle HTTP požadavku nebo podle nastavení určitých HTTP hlaviček.

Spring hojně používá Java anotací. Jde o jakási metadata, která popisují jednotlivé fragmenty kódu od tříd až po jednotlivé proměnné. Řídící třídu ve Spring pak definujeme pomocí anotace @Controller, případně @RestController, pokud má být HTTP požadavek místo HTML ve formě XML či JSON. Další důležitou anotací je @RequestMapping. Tato anotace slouží k definování kritérií pro přístup ke třídě a jejím metodám.

I model má svou vlastní anotaci jménem @Service. @Controller i @Service jsou vlastně tzv. beany. Jde o objekty, které mohou být vytvořeny pomocí IoC kontejneru. Níže je ukázka Controlleru pro správu konzumací s metodou pro výběr konzumace.

```
1  @RestController
2  @RequestMapping(Context.REST_ROOT)
3  public class EatingController extends AbstractRestController {
4
5  ...
6
7    @Autowired
8    private EatingService eatingService;
9
10 ...
11
```

```
12   @RequestMapping(value = "user/{userId}/eating/{eatingId}", method =
       RequestMethod.GET)
13   public ResponseEntity<Response> eatingDetail(
14       Authentication authentication,
15       @PathVariable("userId") Long userId,
16       @PathVariable("eatingId") Long eatingId
17       ) {
18     Guard.preAuthorize(authentication, userId);
19     Eating eating = this.eatingService.getEatingWithImages(eatingId);
20     Guard.notNull(eating, new EntityNotFoundException("Konzumace nebyla
       nalezena"));
21     Guard.mustEqual(userId, eating.getUser().getId(), new AuthException("
       Uzivatel neni vlastnikem konzumace"));
22     return new ResponseEntity<Response>(createEatingDetail(eating), HttpStatus
       .CREATED);
23   }
24
25 ...
26
27 }
```

Listing 3.1: Controller - Detail konzumace

Třída je označena jako RestController. Poté vidíme vytvoření modelu eatingService pomocí IoC kontejneru. Na prvním řádku metody je pak vidět, že metoda je přístupná na adrese 'user/userId/eating/eatingId' při použití HTTP metody GET. Následuje výčet parametrů. Authentication je objekt reprezentující přihlášeného uživatele. Poté vidíme anotaci @PathVariable. Ta mapuje definované části url na proměnné. Následují instrukce pro výběr dané konzumace. Nakonec se vytvoří objekt ResponseEntity, což je obálka pro HTTP odpověď.

```
1  @Service("eatingService")
2  public class EatingService {
3
4  ...
5
6    @PreAuthorize("hasRole('ROLE_USER')")
7    @Transactional(readOnly = true)
8    public Eating getEatingWithImages(Long eatingId) {
9      Eating eating = (Eating) this.dao.get(Eating.class, eatingId);
10     Criteria criteria = this.dao.criteria(EatingImage.class).add(Restrictions.
       eq("eating.id", eatingId));
11     eating.setImages(criteria.list());
12     return eating;
13   }
14
15 ...
16
17 }
```

Listing 3.2: Model - Detail konzumace

Další ukázka kódu ukazuje třídu z vrstvy modelu. Ukázaná metoda z databáze vybírá konkrétní konzumaci. Na začátku vidíme anotaci, která povolí výběr konzumace pouze přihlášenému uživateli. Následuje anotace, která zaručí komunikaci s databází v transakci.

3.2.2.2 Angular MVC

Jak již bylo popsáno, Angular používá automatickou synchronizaci dat mezi modelem a view zvanou Two Way Data Binding. K této synchronizaci se používá objekt Scope, který je přístupný z kontroleru a obsahuje data modelu. Následuje ukázka přidání hmotnosti v Angularu. [23]

```
route.controller("WeightController", function ($scope, $http, $attrs,
    $translate, $compile, $location,
    validation, errorRender, filter, model, httpErrorHandler, GlobalMessages,
    parameterParser, addressStorage, dateProvider) {

...

  $scope.addWeight = function() {
    errorRender.clear(angular.element("#weight-form"));
    $scope.weight.applyValidation(function(){
      addressStorage.get("user-weight", function(address) {
        $scope.weight.form.date = $scope.weight.form.date.toISOString();
        model.add(address, $scope.weight.form, angular.element("#weight-form")
    ,
            function(response) {
          $scope.weightList();
          angular.element($scope.weight.modalSelector).modal("hide");
          GlobalMessages.push("success", "weight.add.success");
        }, function(response) {
          if (response.status != 400)
            angular.element($scope.weight.modalSelector).modal("hide");
        });
      });
    });
  }

...

});
```

Listing 3.3: Angular - Přidání hmotnosti

Metoda addWeight se vykoná po kliknutí tlačítka s nastavenou direktivou ng-click = "addWeight()". Pokud formulář odešleme s nevalidními daty, server vrátí status 400 a seznam validačních hlášek. Tyto hlášky se zobrazí ve formuláři u konkrétních formulářových polí, do kterých byla nevalidní data vložena. Z toho důvodu se jako první věc v metodě addWeight volá metoda na vymazání těchto hlášek. Poté se aplikuje validace. Po validaci je potřeba získat adresu (entry point) pro přidání hmotnosti na straně serveru. Poté následuje samotné vykonání požadavku. Při úspěšném přidání se pak aktualizuje seznam hmotností a skryje se okno s formulářem pro přidání.

3.3 REST

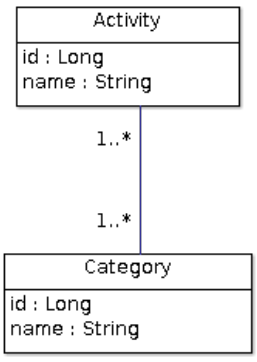
Representational State Transfer (REST) je jedním z typů webové služby orientující se na snadný přístup ke zdrojům (datům). Je silně inspirován HTTP protokolem. Pro rozlišení jednotlivých operací používá HTTP metody. Tyto operace jsou:

- GET - Základní požadavek pro přístup k datům. Jde o nejčastější metodu.
- POST - Požadavek na vytvoření nových dat. U typických webových aplikací se používá pro odesílání formulářů. Na rozdíl od GET požadavku se data v POST požadavku předávají v těle požadavku.
- PUT - Podobný požadavek jako POST jen s tím rozdílem, že jej voláme nad konkrétním zdrojem za účelem jeho upravení.
- DELETE - Složí ke smazání dat.

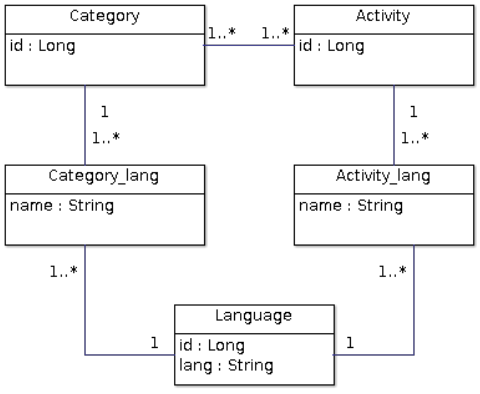
REST je založen na několika základních pilířích. Prvním je jednotné rozhraní. Data mezi serverem a klientem jsou přenášena v nezávislém formátu na platformě. Obvykle ve formátu XML nebo JSON. REST je bezestavový. Neukládá si stav mezi jednotlivými požadavky a zároveň jednotlivé zdroje na sobě nejsou závislé. Dalším pilířem je kešování. Každá HTTP odpověď by měla nést informaci o tom, zda lze odpověď kešovat či nikoliv. [9]

3.4 Datová vrstva

Jedním z požadavků kladených na aplikaci je snadné rozšíření aplikace o další jazyky. To se týká i dat v databázi. Původní databáze Mobiabu s dalšími jazyky nepočítala, a proto bylo potřeba ji upravit. Úprava se týkala hlavně tabulek s názvy aktivit a jejich kategorií a s názvy jídel. Původní tabulky bylo nutné rozdělit na dvě. V jedné tabulce zůstaly informace, které nepotřebují překlad. Druhá tabulka obsahuje odkaz na první tabulku, přeložená data a reference na jazyk. Kvůli velkému počtu záznamů bylo potřeba napsat skript, který data automaticky převede. Úprava je znázorněna na obrázku 3.2 a 3.3.



Obrázek 3.2: Databáze před úpravou



Obrázek 3.3: Databáze po úpravě

Toto řešení je o něco pomalejší, jelikož je pro výběr dat je potřeba spojit dvě tabulky. Toto lze částečně vyřešit vytvoření pohledu nad databází. Pohled je vlastně virtuální tabulka vytvořená z ostatních tabulek. Použití pohledů může pozitivně ovlivnit výkon databáze.

3.5 Implementace

Na následujících řádcích jsou popsány zajímavé části implementace.

3.5.1 Kontrola unikátních emailů

Aby mohl uživatel využívat funkcí aplikace, musí se nejprve zaregistrovat. Při registraci zadává svou emailovou adresu, ta musí být unikátní pro každého uživatele. Bylo tedy nutné implementovat vlastní validační pravidlo. Spring podporuje Bean Validation, což je je standard Javy EE pro validování dat pomocí anotací. V rámci Springu existují validátory téměř pro všechny základní validace jako je povinné pole a email.

```
1
2  @Documented
3  @Target({ElementType.METHOD, ElementType.FIELD, ElementType.ANNOTATION_TYPE})
4  @Constraint(validatedBy = UniqueUserEmailValidator.class)
5  @Retention(RetentionPolicy.RUNTIME)
6  public @interface UniqueUserEmail {
7
8      String message() default "cvut.mobiab.constraint.annotation.
       uniqueUserEmail";
9
10     Class<?>[] groups() default {};
11
12     Class<? extends Payload>[] payload() default {};
13 }
```

Listing 3.4: Validace unikátního emailu - anotace

Zdrojový kód nad tímto textem ukazuje definici vlastní anotace. Jde o velmi jednoduchou definici s parametrem message. Jde o zprávu, která se zobrazí pokud validace není úspěšná. Následuje ukázka samotného validátoru.

```
1
2  @Component
3  public class UniqueUserEmailValidator implements ConstraintValidator<
       UniqueUserEmail, String> {
4
5      @Autowired
6      private UserService userService;
7
8      @Override
9      public void initialize(UniqueUserEmail annotation) {
10     }
11
12     @Override
13     public boolean isValid(String value, ConstraintValidatorContext context) {
14         if (value == null || value.isEmpty()) {
15             return true;
```

```
16          }
17          return this.userService.isEmailUnique(value);
18      }
19
20  }
```

Listing 3.5: Validace unikátního emailu - anotace

Validátor musí implementovat rozhraní ConstraintValidator. Rozhraní obsahuje dvě metody. První je metoda initialize, ta se volá jako první a slouží k nadefinování vlastností validátoru. Druhou metodou je isValid, která obsahuje vlastní validační logiku. V tomto případě volá metodu, která zjistí, zda se email nachází v databázi či nikoliv. Následuje ukázka validátoru v praxi.

```
1
2       @UniqueUserEmail(message = "validation.error.email.unique")
3       private String email;
```

Listing 3.6: Validace unikátního emailu - anotace

3.5.2 Výpočet Bazálního metabolismu (BMR)

Kalorický výdej by měl být v ideálním případě stejný jako kalorický příjem. Pro stanovení doporučeného denního limitu příjmu energie jsem tedy použil Harris-Benedictův vzorec pro výpočet bazálního metabolismu. Výsledek zahrnuje množství vydané energie v klidovém stavu. Jde o stav, který je dán fungováním životně důležitých orgánů. Vzorec je různý pro muže a ženy. Vstupními parametry jsou: pohlaví, výška v centimetrech, váha v kilogramech a věk v letech.

Vzorec pro muže

BMR = 66.5 + (13.75 x váha v kg) + (5.003 x výška v cm) – (6.755 x věk v letech) [12]

Vzorec pro ženu

BMR = 655.1 + (9.563 x váha v kg) + (1.850 x výška v cm) – (4.676 x věk v letech) [12]

Jelikož výsledek BMR vrátí výdej v klidovém stavu, který je z praktického hlediska nepoužitelný, bylo potřeba bazální metabolismus přenásobit koeficientem podle typu aktivity uživatele. Koeficienty jsou vypsané v tabulace 3.1. Nutno dodat, že tato metoda určení denní příjmu je pouze orientační. Je pravděpodobné, že v další fázi vývoje budou implementovány funkce pro stanovení denního příjmu přímo lékařem.

Tabulka 3.1: Výpis koeficientů aktivit pro přepočet BMR

Aktivita	**Koeficient**	**Popis**
sedavá	1,2	Málo aktivního pohybu. Například práce v kanceláři
lehká	1,375	Lehké sportování nebo pohybu, 1-3 dny v týdnu.
střední	1,55	Střední sportování nebo pohybu, 3-5 dní v týdnu
těžká	1,725	Hodně sportování nebo pohybu, 6-7 dní v týdnu
extrémní	1,9	Každodenní náročná fyzická aktivita

Výsledek metody je v kilokaloriích. Pro přepočet na kilojouly musíme výsledek přenásobit konstantou 4,184. [27]

3.5.3 Proces přihlášení

Původní databáze Mobiabu obsahovala informace o téměř osmi stech uživatelích. Hesla těchto uživatelů jsou zahashována pomocí funkce MD5. Tato hashovací funkce je už dávno překonána, a proto jsou hesla nových uživatelů zahashována funkcí BCRYPT. Tato funkce (na rozdíl od MD5) přidá k heslu náhodně vygenerovaný řetězec (tzv. sůl). To znemožňuje uhádnutí původního hesla z hashe. Hashovací funkce je jednosměrná transformace dat na řetězec fixní délky. Uživatelé bohužel mají tendenci volit lehká hesla. Velmi časté heslo je "12345". Tomu odpovídá hash "827ccb0eea8a706c4c34a16891f84e7b"a pokud tuho hash zadáme do vyhledávače Google hned na první stránce je několik odkazů s původní hodnotou "12345". Právě tuto slabinu vyřeší osolení původního řetězce.

Nabízí se otázka, jak spojit stávající uživatele s heslem v MD5 a ty nové s BCRYPT. Rozhodl jsem se, že původní hesla nechám zahashovaná v MD5 a ke každému uživateli přidám informaci o typu hashovací funkce. Při přihlášení je tedy nutné zjistit typ funkce. Proces přihlášení je vidět pod tímto textem.

```
1
2       @Override
3       public Authentication authenticate(Authentication a) throws
        AuthenticationException {
4
5           String email = a.getName();
6           String password = a.getCredentials().toString();
7
8           Person person = this.user.getUserAuthData(email);
9           AppRoleEnum role = AppRoleEnum.ROLE_USER;
10
11          if (person == null) {
12            person = this.user.getDoctorAuthData(email);
13              role = AppRoleEnum.ROLE_DOCTOR;
14          }
15
16          HashProvider hash = null;
17          if (person != null) {
18              hash = resolveHashProvider(person.getEncoder());
19              log.info("Hash algorithm for: "+email+" is: "+hash.getType());
20          }
21
22          String isHash = request.getHeader("X-Hash");
23
24          if (hash != null && person.getEmail().equals(email) && matchesHash(
        hash, person.getHash(), password, isHash)) {
25              List<GrantedAuthority> roles = new ArrayList<>();
26              roles.add(new SimpleGrantedAuthority(role.name()));
27              log.info(email+" logged with role "+role.name());
28              return new UsernamePasswordAuthenticationToken(person, password,
        roles);
29          } else {
30              log.info("Authentification process failed for: "+email);
31              throw new AuthException("Authentification process failed");
```

```
32          }
33      }
```

Listing 3.7: Proces přihlášení do systému

Metoda ze třídy implementující rozhraní AuthenticationProvider. Jde o rozhraní Springu právě pro podporu přihlašovacího procesu. Uživatel může své heslo odeslat na server již v zahashované podobě (týká hesel v MD5). Aby server poznal, zda je heslo v zahashované podobě, musí uživatel poslat HTTP hlavičku X-Hash s nastavenou hodnotou na true.

3.5.4 Výpis jídel

U výpisu jídel byl kladen velký důraz na snadné nalezení požadovaného jídla. Jídla lze filtrovat podle kategorie, lze je stránkovat nebo jídlo vyhledávat podle názvu. K sestavení dotazu nad databází jsem použil Criteria API od Hibernate. Díky kritériím lze sestavovat dotazy na úrovni objektů. Dotaz je sestavován tak, aby bylo možné jednotlivé filtry libovolně aplikovat.

```
1
2  @Transactional(readOnly = true)
3  public List<Food> allFoods(String query, Long categoryId, Long ownerId,
       Language language, String order, Integer limit, Integer offset, boolean
       onlyOwner) {
4    Criteria criteria = this.dao.criteria(Food.class);
5
6    applyAllFoodCriteria(criteria, query, categoryId, ownerId, language,
       onlyOwner);
7    if (limit != null)
8      criteria.setMaxResults(limit);
9    if (offset != null)
10     criteria.setFirstResult(offset);
11   if (order != null)
12     criteria.addOrder(parser.orderBy(order));
13   return criteria.list();
14 }
15
16 ...
17
18 private void applyAllFoodCriteria(Criteria criteria, String query, Long
       categoryId, Long ownerId, Language language, boolean onlyOwner) {
19   criteria.add(Restrictions.eq("isVisible", true));
20   if (query != null) {
21     List<String> words = Arrays.asList(string.index(query).split(" "));
22     for (String w : words) {
23       criteria.add(Restrictions.like("index", "%"+w+"%"));
24     }
25   }
26   if (categoryId != null) {
27     criteria.createAlias("categories", "category");
28     criteria.add(Restrictions.eq("category.id", categoryId));
29   }
30   if (onlyOwner) {
31     criteria.add(Restrictions.eq("user.id", ownerId));
32   } else {
```

```
33      if (ownerId != null) {
34        criteria.add(Restrictions.or(Restrictions.eq("user.id", ownerId),
        Restrictions.isNull("user.id")));
35      } else {
36        criteria.add(Restrictions.isNull("user.id"));
37      }
38    }
39    if (language != null) {
40      Lang lang = getLanuageByCode(language);
41      criteria.add(Restrictions.or(Restrictions.eq("language.id", lang.getId()),
         Restrictions.isNotNull("user.id")));
42    }
43  }
```

Listing 3.8: Výpis jídel

3.5.5 Upload fotografií ke konzumacím

Fotografie jsou ukládány na file systém. Jejich název se navíc ukládá do databáze, společně s referencí na konkrétní jídlo. Spring uploadovaný soubor na server obalí do objektu MultipartFile, který je pak dostupný z kontroleru. Aplikace podporuje upload více fotografií najednou, v kontroleru, tak je dostupný seznam patřičných objektů. Před samotným uložením každá fotografie projede validací. Povolené formáty jsou jpg, png a gif. Maximální velikost obrázku je omezena na 300 kb. Obojí omezení se mohou do budoucna změnit.

Pokud všechny fotografie prošly validací, jsou uloženy na file systém a poté do databáze. V rámci aplikace je implementována funkce na zmenšování fotografií, neboli náhledů. Funkce je vidět na následující ukázce.

```
1
2  public BufferedImage scaleByWidth(BufferedImage image, int newWidth) {
3    double ratio = (double) newWidth / image.getWidth();
4    double newHeight = image.getHeight() * ratio;
5    return scale(image, newWidth, (int) newHeight);
6  }
7
8  public BufferedImage scale(BufferedImage image, int newWidth, int newHeight) {
9
10   int width = image.getWidth();
11   int height = image.getHeight();
12
13   BufferedImage newImage = new BufferedImage(newWidth, newHeight, image.
       getType());
14   AffineTransform at = new AffineTransform();
15   double wRatio = (double) newWidth / width;
16   double hRatio = (double) newHeight / height;
17   at.scale(wRatio, hRatio);
18
19   AffineTransformOp scaleOp = new AffineTransformOp(at, AffineTransformOp.
       TYPE_BILINEAR);
20
21   newImage = scaleOp.filter(image, newImage);
22   return newImage;
23 }
```

Listing 3.9: Zmenšování fotografií

3.5.6 Single Entry point

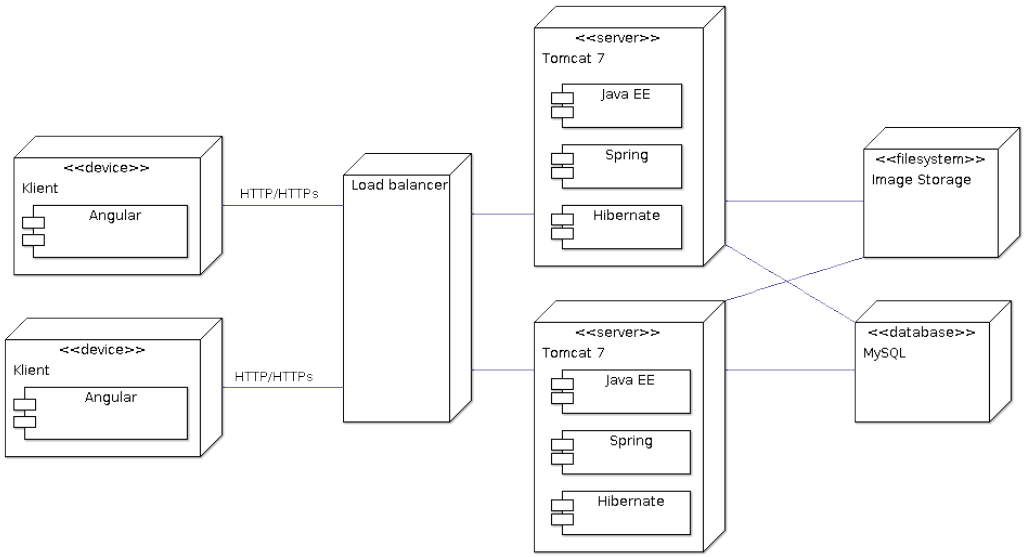
Webová služba jako REST by měla být samo-vysvětlující. Z toho důvodu je na adrese http://<host>/rest přístupný seznam všech entry pointů, které může klient využít. Seznam závisí na roli uživatele. Seznam pro nepřihlášeného uživatele tak obsahuje entry point pro registraci a přihlášení, zatímco seznam uživatele obsahuje adresy na záznamy jednotlivých modulů. Stejně tak data odeslaná ze serveru na klienta obsahují adresy na detail konkrétních záznamů. To má smysl například při úpravě nebo smazání konkrétního záznamu.

```
{
    "count": 18,
    "items": [
        {
            "code": "registration",
            "name": "Registrace",
            "url": "http://localhost:8080/Mobiab/rest/user/registration"
        },
        {
            "code": "login",
            "name": "Prihlaseni",
            "url": "http://localhost:8080/Mobiab/rest/authenticate"
        },
        {
            "code": "logout",
            "name": "Odhlaseni",
            "url": "http://localhost:8080/Mobiab/rest/logout"
        },
        {
            "code": "activity-category-list",
            "name": "Seznam kategorii aktivit",
            "url": "http://localhost:8080/Mobiab/rest/activity/category"
        },
        {
            "code": "activity-list",
            "name": "Seznam aktivit",
            "url": "http://localhost:8080/Mobiab/rest/activity"
      },

   ...

}
```

Listing 3.10: Seznam zdrojů

3.6 Nasazení

Aplikace je nasazena na cloudové platformě OpenShift. Jde o PaaS cloudovou službu vytvořenou a provozovanou společností Red Hat. [16]



Obrázek 3.4: Diagram nasazení

Kapitola 4

Testování

Na začátku této kapitoly popíšu průběh testování, poté představím jednotlivé participanty a na závěr se budu věnovat nalezeným chybám.

4.1 Průběh testování

V první fázi bylo potřeba vybrat vhodné participanty pro testování. Výběr probíhal formou dotazníku, neboli screeneru. Jde o několik jednoduchých otázek a odpovědí, které ukáží, zda je daná osoba vhodná jako participant pro testování. Otázek je celkem 6 a u každé odpovědi je preferovaný počet osob, kteří vybrali tuto odpověď.

Screener

1. Jaký je váš věk?

1. 10 - 15
2. 16 - 30 *(2x)*
3. 31 - 50 *(2x)*
4. 51 - 70 *(1x)*
5. víc než 70

2. Jaké je vaše pohlaví?

1. Muž *(3x)*
2. Žena *(2x)*

3. Vyberte tvrzení, které na vás nejvíc sedí.

1. Na internet téměř vůbec nechodím.
2. Na internetu si prohlížím pouze fotografie. *(2x)*
3. Občas nakupuji na internetu *(2x)*
4. Používám internet každý den téměř na cokoliv *(1x)*

4. Znáte nějakou aplikaci na záznam aktivit nebo příjmu potravy? Pokud ano, jakou?

1. Ne *(3x)*
2. Ano *(2x)*

5. Trpíte cukrovkou?

1. Ne *(1x)*
2. Ano, trpím 1. typem *(2x)*
3. Ano, trpím 2. typem *(2x)*

6. Máte zájem podílet se na testování aplikace?

1. Ano *5x*
2. Ne

Po výběru participantů následovalo sestavení několika úkolů. Participant postupně plnil úkoly a já jej při tom pozoroval. Po splnění úkolů následoval krátký rozhovor, ve kterém jsem se doptával na různé věci ohledně úkolů a samotné aplikace.

Seznam úkolů

- Zaregistrujte se do systému a následně se zkuste přihlásit.
- Přidejte potravinu "Zvolenský jogurt"do zkonzumovaných potravin jako snídani. Poté ji upravte a nakonec smažte.
- Vytvořte vlastní jídlo a poté jej přidejte do zkonzumovaných potravin.
- Vytvořte novou aktivitu a pokuste se ji uložit do oblíbených.
- Zobrazte v tabulce naměřené glykémie za poslední tři dny.
- Vyberte si lékaře a napište mu zprávu.

Nakonec je nutné dodat, že aplikace ještě není zdaleka hotová zvláště z pohledu uživatelského rozhraní. I tak má testování smysl jelikož pomůže odhalit chyby ve stávajícím řešení a dá nám náhled do uvažování a orientace uživatelů v systému.

4.2 Popis participantů

Testování se účastnilo celkem 5 participantů. Dvě ženy a tři muži. Testování probíhalo pouze pro roli uživatele systému. Níže je krátký popis jednotlivých participantů.

Participant 1

První participantkou je žena, je jí 25 let. Ačkoliv není diabetičkou, má zkušenosti z diabetickými dětmi. Sama se živí jako programátorka. Jde tedy o velmi pokročilou uživatelku v oblasti IT.

Participant 2

Druhým participantem je muž, 60 let. Jde o středně zkušeného uživatele internetu. Shodou okolností jde o lékaře ortopeda. Z toho důvodu jsem mu ukázal aplikaci i z pohledu lékaře.

Participant 3

Třetím participantem je muž, kterému je 28 let. Jde o velmi zkušeného uživatele internetu.

Participant 4

Čtvrtou participantkou byla žena, 49 let. Má pouze základní znalosti internetu.

Participant 5

Posledním participantem byl opět muž, 25 let. Jeho znalost internetu je na vyšší úrovni.

4.3 Přehled nálezů

Každý nález má přiřazenou jednu ze tří priorit. Priorita označuje nutnost opravy jednotlivých chyb.

Vyhledání vlastního jídla (vysoká priorita)

Při přidání konzumace nikde není vidět seznam jídel, která si uživatel sám vytvořil. Uživatelem vytvořená jídla lze najít pouze přes vyhledávací pole. Tato chyba se dá opravit přidáním položky "Má jídla"do seznamu kategorií jídel.

Výběr data narození (střední priorita)

Téměř každý participant měl problém se zadáním data narození. Jednak u políčka chybí informace o formátu data, mnohem větší problém byl pro participanty výběr data přes kalendář. Po otevření kalendáře se zobrazí aktuální měsíc. Pokud chceme vybrat jiný měsíc nebo dokonce rok, musíme kliknout na horní panel, ve kterém se nachází aktuální měsíc a rok. Participantům nedocházelo, že lze takto vybrat jiný měsíc či rok. Kalendář je hotová komponenta, která je součástí knihovny Angular UI. Tento nález nález lze opravit buď nahrazení komponenty za jinou, nebo se mohu pokusit upravit zdrojové kódy knihovny. Knihovna je pod OpenSource licencí.

Výběr lékaře (střední priorita)

Participantům se nedařilo najít formulář pro výběr lékaře. Výběr se nachází pod odkazem se jménem přihlášeného uživatele. Tento odkaz se nachází v pravém horním rohu. Pod odkazem jsou schované funkce pro úpravu uživatelova profilu a mezi nimi i výběr lékaře. Možná by bylo vhodnější odkaz pojmenovat jako "nastavení". Dále by dobré uživatele po čase informovat o možnosti výběru lékaře. Například formou vyskakovacího okna.

Přihlášení po registraci (nízká priorita)

Po úspěšné registraci se uživateli zobrazí hláška, která ho vyzývá k přihlášení. Odkaz na přihlášení se nachází v pravém horním rohu a je poměrně dobře viditelný. Zkušenější uživatelé neměli s tímto umístěním problém, ale participantka číslo 4 měla nejprve problém odkaz najít. Tato chyba se dá lehce odstranit přidáním odkazu ve formě tlačítka přímo pod text, který nabádá k přihlášení.

Přidání konzumace (nízká priorita)

Přidání konzumace bylo téměř bez problémů. Uživatelé rychle našli formulář pro přidání, ovšem ve chvíli, kdy se snažili najít jídlo se objevilo pole pro vyhledání jídla a pod ním výpis kategorií jídel. Dva participanti byli mírně zmatení tímto výpisem, jelikož zde chybí popis tohoto výpisu.

Po splnění úkolů jsem se ptal, zda si všimli, že jídlo lze uložit do oblíbených. Participanti obvykle odpovídali něco v tom smyslu, že si všimli tlačítka oblíbené a poté samotné hvězdičky, ale nevěděli co znamená. K hvězdičce je potřeba přidat text a nebo ji umístit k poli pro výběr jídla.

4.4 Další nalezené problémy

Participantka 1, která má zkušenosti s diabetem navrhla pár kosmetických změn, jako je přejmenování inzulínu k jídlu na bolusový inzulín a přejmenování chlebových jednotek na výměnné jednotky. Tyto věci již byly opraveny.

Participant 2 zase navrhl, aby uživatelé vyplňovali své rodné číslo podle kterého by bylo možné pacienta vyhledat. Hledání pacientů podle rodného čísla je u lékařů běžný postup.

Z další diskuze s participanty a členy týmu Mobiab vzešly další nálezy. U výpisu dat pacienta v roli lékaře se zobrazují všechny data pod sebou jak ve formě tabulky, tak i ve formě grafu. Data lze filtrovat podle času. Filtrovaná data mají nejednotnou časovou osu (osu x). Osy přes všechny grafy je potřeba sjednotit, aby lékař ihned viděl všechny hodnoty v grafech v konkrétním čase.

Při zadávání konzumací by bylo dobré, kdyby měl uživatel možnost zobrazit si detail vybraného jídla a mohl si tak prohlédnout jeho výživové hodnoty.

Participantům se rovněž nelíbilo, že společně s hmotností musí zadat i obvod pasu.

4.5 Závěr testování

Testování objevilo několik nedostatků, ale zároveň ukázalo, že aplikace je celkem přehledná. Vzhled aplikace byl hodnocen kladně, participanti oceňovali jednoduchý a přehledný design.

Kapitola 5

Závěr

Cílem práce bylo kompletně předělat původní aplikaci pro sběr dat z mobilního telefonu a zároveň umožnit správu dat skrze webový prohlížeč. Podařilo se implementovat všechny klíčové funkce a poskytovat je skrze REST. Rozhraní RESTu se od původního v mnohém liší. Například bylo nutné přidat informace o umístění jednotlivých zdrojů. Původní rozhraní není nijak zdokumentováno, a proto se liší i struktura dat. Jelikož se počítá s kompletním přepsáním mobilní aplikace, nebude změna API velký problém.

Aplikace je chápána jako centrální datové úložiště. Díky použitým technologiím ji lze celkem snadno rozšířit a nadále spravovat. Aplikace je připravena k nasazení na více serverů a měla by tedy zvládnout větší zátěž než doposud.

Toto byl můj první velký projekt webové aplikace, napsané v jazyce Java. Předtím jsem vytvářel webové aplikace v jazyce PHP a velice mě překvapil fakt, že svět Javy EE je od světa PHP velmi vzdálený. Java EE, narozdíl od PHP, obsahuje pokročilé nástroje, knihovny a postupy. Po této kladné zkušenosti hodlám psát webové aplikace pouze v Javě.

5.1 Budoucí rozšíření systému

Aplikace není ani zdaleka ve finálním stavu. Následuje seznam věcí, které je potřeba dodělat nebo doladit.

Implementace okrajových funkcí

V aplikaci chybí funkce pro odeslání zapomenutého hesla. Uživateli samozřejmě nebude odesláno jeho původní heslo, bude mu vygenerováno náhodné heslo a to se mu odešle na email vyplněný při registraci. S tím souvisí další nedodělaná věc a tou je potvrzení registrace. Pro aktivování svého profilu bude uživatel nucen kliknout na aktivační odkaz, který mu bude odeslán emailem.

Zprávy

Zprávy mezi lékařem a pacientem lze odesílat už nyní. Co ale chybí je notifikace o přijetí nové zprávy. Zde by se mi líbilo použití technologie WebSockets pro obousměrnou komunikaci mezi serverem a klientem.

Modul zpráv je připraven na komunikaci více uživatelů v jedné diskuzi, ale zatím není tato funkcionalita implementována.

Vylepšení grafu aktivit

Aktivity jsou vizualizovány pomocí sloupcového grafu, kde jeden sloupec představuje hodinový kalorický výdej. Chybí zde informace o typu aktivit. V budoucnu by sloupce mohli být barevně odděleny podle aktivity.

Efektivní zpracování fotografií

Při uploadu fotografií dochází k vytvoření náhledu fotografie. Tato úprava je časově náročná. V budoucnu bude úprava vykonávána tzv. workery, což je neustále běžící proces na pozadí.

Párování profilů

U menších dětí nelze očekávat, že by sami vyplňovali jednotlivé záznamy. Předpokládá se, že by to za ně dělali rodiče. Aby se rodič nemusel přihlašovat na více účtů (svůj a dítěte), bude mít možnost svůj účet spárovat s účtem dítěte. Párování bude mít několik úrovní. Na té základní úrovni bude mít pouze přístup k datům. Na nejvyšší úrovni pak bude moci data libovolně upravovat.

Synchronizace dat mezi serverem a klientem

Velké množství uživatelů mobilních telefonů má přístup na internet pouze přes Wi-Fi a často bývají bez internetu. Mobilní aplikace se těmto uživatelům snaží vyjít vstříc tak, že data ukládá do paměti telefonu a v případě připojení k internetu je odešle na server. K tomu je potřeba na serveru implementovat mechanizmus, který data správně synchronizuje.

Literatura

[1] autoři Techopedia. [online]. [cit. 18. 5. 2016]. Dostupné z: <https://www.techopedia.com/definition/438/clientserver-architecture>.

[2] Diabetes.co.uk. [online]. [cit. 16. 5. 2016]. Dostupné z: <http://www.diabetes.co.uk/diabetes_care/blood-sugar-level-ranges.html>.

[3] Diabetická asociace ČR. [online]. 2014. [cit. 20. 5. 2016]. Dostupné z: <http://www.diabetickaasociace.cz/co-je-diabetes/>.

[4] DiaMonitor. [online]. [cit. 16. 5. 2016]. Dostupné z: <https://diamonitor.com>.

[5] Dětská diabatologie. [online]. [cit. 16. 5. 2016]. Dostupné z: <http://www.detskydiabetes.cz/co-ovlivnuje-glykemii>.

[6] Kalorické tabulky. [online]. [cit. 16. 5. 2016]. Dostupné z: <http://www.kaloricketabulky.cz/>.

[7] Léčba cukrovky. [online]. 2015. [cit. 20. 5. 2016]. Dostupné z: <http://www.lecbacukrovky.cz/lecba>.

[8] Malte Ubl a Eiji Kitamura. [online]. 2010. [cit. 18. 5. 2016]. Dostupné z: <http://www.html5rocks.com/en/tutorials/websockets/basics/>.

[9] Martin Malý. [online]. 2009. [cit. 18. 5. 2016].

[10] MUDr. Tomáš Edelsberger. [online]. [cit. 16. 5. 2016]. Dostupné z: <http://www.lecbacukrovky.cz/slovnicek/bolusova-davka>.

[11] Oracle. [online]. [cit. 17. 5. 2016]. Dostupné z: <http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html>.

[12] přispěvatelé Wikipedie. [online]. 2016. [cit. 18. 5. 2016].

[13] přispěvatelé Wikipedie. [online]. 2016. [cit. 18. 5. 2016].

[14] přispěvatelé Wikipedie. [online]. 2016. [cit. 20. 5. 2016]. Dostupné z: <https://cs.wikipedia.org/wiki/Diabetes_mellitus>.

[15] přispěvatelé Wikipedie. [online]. [cit. 18. 5. 2016]. Dostupné z: <https://cs.wikipedia.org/wiki/Apache_Maven>.

[16] přispěvatelé Wikipedie. [online]. 2016. [cit. 23. 5. 2016].

[17] přispěvatelé Wikipedie. [online]. 2016. [cit. 17. 5. 2016]. Dostupné z: <https://en.wikipedia.org/wiki/Spring_Framework>.

[18] tvůrci Angularu. [online]. [cit. 18. 5. 2016]. Dostupné z: <https://docs.angularjs.org/guide/introduction>.

[19] tvůrci Hibernate. [online]. [cit. 17. 5. 2016]. Dostupné z: <http://hibernate.org/orm/>.

[20] tvůrci Mavenu. [online]. [cit. 18. 5. 2016]. Dostupné z: <https://maven.apache.org/>.

[21] tvůrci Mavenu. [online]. [cit. 18. 5. 2016]. Dostupné z: <http://maven.apache.org/guides/mini/guide-naming-conventions.html>.

[22] tým Tutorialized. [online]. [cit. 18. 5. 2016]. Dostupné z: <http://www.tutorialized.com/tutorial/Fundamentals-of-an-MVC-Framework/81946>.

[23] tým Tutorialspoint. [online]. [cit. 18. 5. 2016]. Dostupné z: <http://www.tutorialspoint.com/angularjs/angularjs_scopes.htm>.

[24] tým Tutorialspoint. [online]. [cit. 17. 5. 2016]. Dostupné z: <http://www.tutorialspoint.com/spring/index.htm>.

[25] tým w3schools. [online]. [cit. 18. 5. 2016]. Dostupné z: <http://www.w3schools.com/bootstrap/bootstrap_get_started.asp>.

[26] tým w3schools. [online]. [cit. 18. 5. 2016]. Dostupné z: <http://www.w3schools.com/html/html_intro.asp>.

[27] Václav Burda. [online]. 2012. [cit. 18. 5. 2016].

Příloha A

Seznam použitých zkratek

HTTPS	Hypertext Transfer Protocol Secure
REST	Representational State Transfer
API	Application Programming Interface
XML	Extendible Markup Language
JSON	JavaScrip Object Notation
MVC	Model-VIew-Controller
PHP	Hypertext Preprocessor
BMR	Basal metabolic rate
PaaS	Platform-as-a-Servise

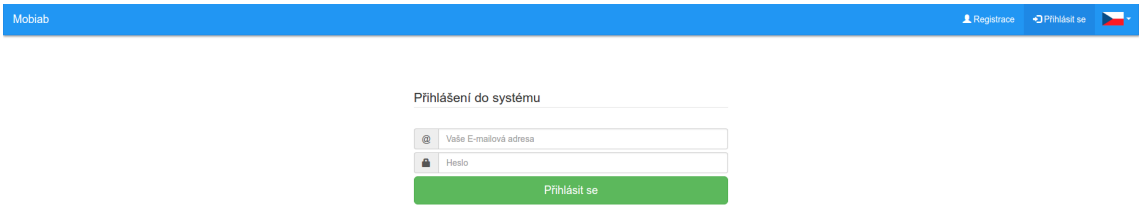
Příloha B

Slovníček pojmů

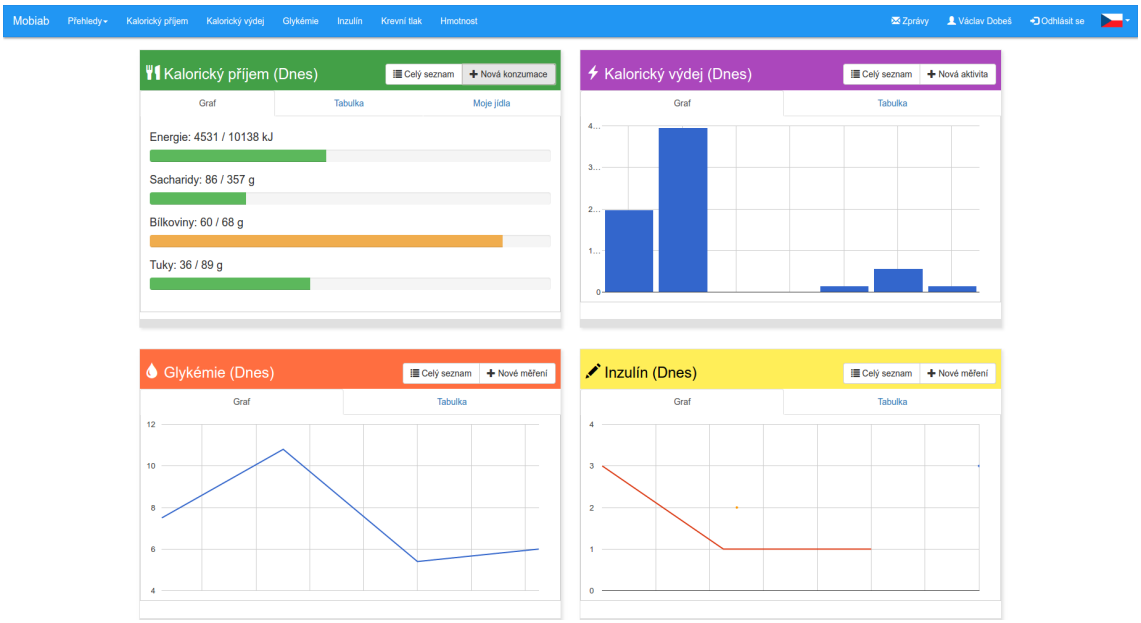
vertikální škálovatelnost	Zvýšení výkonu serveru za účelem snížení času odezvy aplikace při určitém počtu požadavků na sever.
horizontální škálovatelnost	Přidání dalšího serveru za účelem snížení času odezvy aplikace při určitém počtu požadavků na sever.
framework	Sada knihoven a postupů, které usnadňují vývoj softwaru.
hyperglykémie	Stav, kdy je množství cukru v krvi nad limitem 7 mmol/l.
systolický tlak	Nejvyšší tlak krve, kterého je dosaženo během srdeční systoly.
diastolický tlak	Nejnižší tlak krve, kterého je dosaženo během srdeční diastoly.
autentizace	Proces ověření proklamované identity subjektu.
autorizace	Proces získávání souhlasu s provedením nějaké operace.
jar	Souborový formát, používaný platformou Java
hashovácí funkce	funkce, která jako vstup přijímá řetězec znaků o libovolné délce a výsledkem je řetězec znaků s pevnou délkou, tzv.otisk.

Příloha C

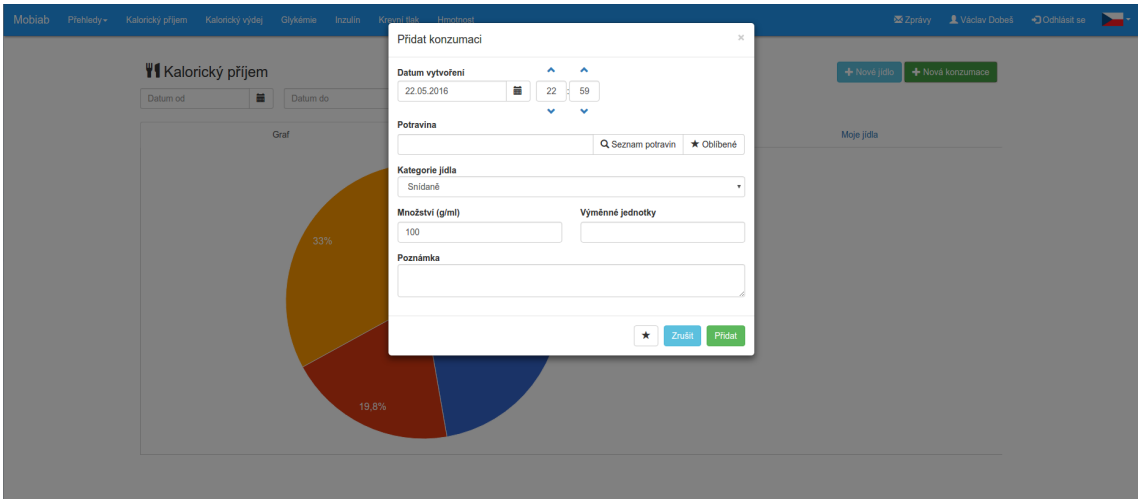
Obrazová příloha



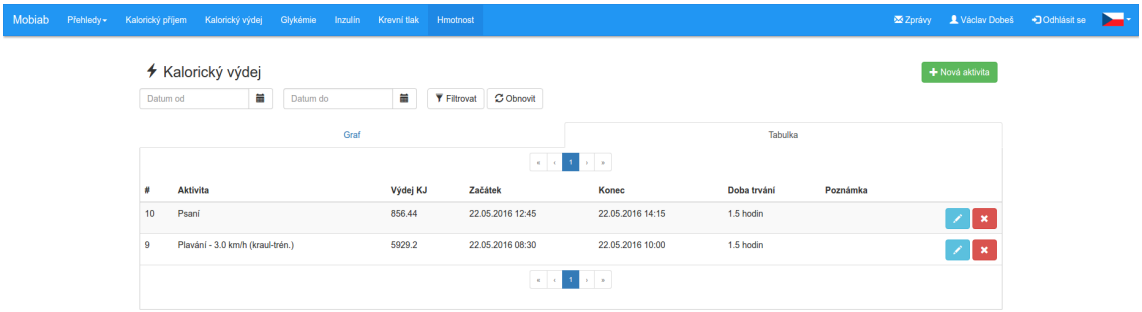
Obrázek C.1: Přihlášení do aplikace



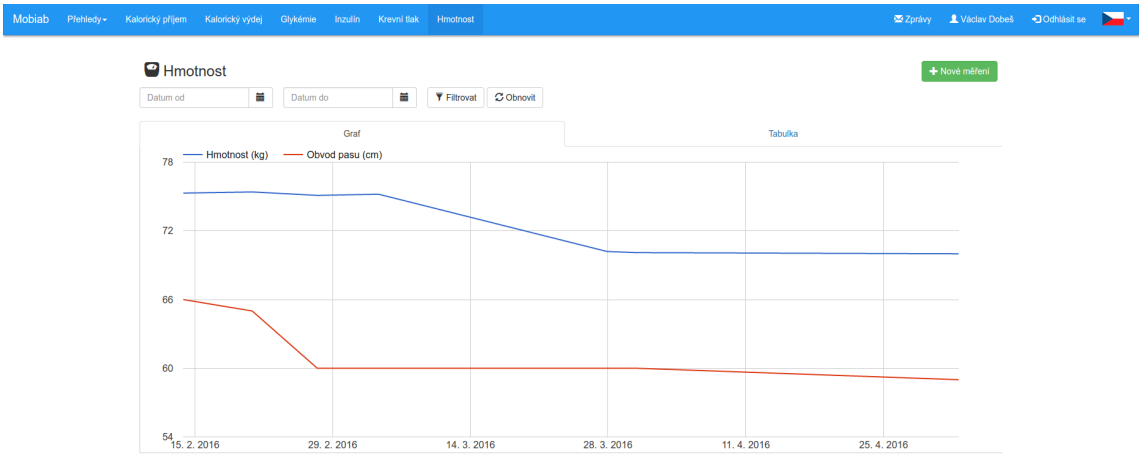
Obrázek C.2: Celkový přehled - dashboard



Obrázek C.3: Přidání nové konzumace



Obrázek C.4: Přehled aktivit v tabulce



Obrázek C.5: Přehled měření hmotností v grafu

Příloha D

Obsah přiloženého CD

```
CD/
├── database
│   └── mobiab.sql
├── src
│   ├── maven
│   └── portal-mobiab.war
├── text
│   ├── latex
│   └── dp_dobesvac_2016.pdf
└── README.txt
```