Gymnázium, Praha 6, Arabská 14

Programování

Ročníkový projekt

2023 Michal Bogdanov, Kryštof Maxa, Šimon Hájek

Gymnázium, Praha 6, Arabská 14

Arabská 14, Praha 6, 160 00

Ročníkový projekt

Předmět: Programování
Téma: Aktivitář

Školní rok: 2022/2023
Autoři: Michal Bogdanov, Kryštof Maxa, Šimon Hájek
Třída: 3.E
Vedoucí práce: Mgr. Jan Lána
Třídní učitel: Mgr. Blanka Hniličková

Rádi bychom na tomto místě poděkovali panu Ing. Chalupníčkovi, který nám pomohl s výběrem jazyka a frameworku, dále bychom chtěli poděkovat panu profesoru a Mgr. Janu Lánovi za radu ohledně SQL problému.

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V Praze dne ..... 2023 podpis: .....

Michal Bogdanov, Kryštof Maxa, Šimon Hájek

Anotace:

Aktivitář je webová aplikace sloužící jako nástroj pro zápis aktivit, které na denní bázi vykonáváte, ale také pro sledování množství kalorií, které jste za daný den přijali. Tyto záznamy jsou poté vyobrazeny na různých kartičkách a grafech, které dělají orientaci v datech jednodušší. Dále je zde možnost uložit si délku spánku za poslední noc, což umožňuje mít lepší kontrolu nad vaší pravidelností. Aplikace tedy slouží jako místo, kde si můžete ukládat veškeré důležité události, které souvisejí se zdravým životním stylem a je dobrá pro všechny, kteří hledají způsob, jak mít kontrolu nad svým způsobem života a také pro ty, kteří hledají motivaci nepřestávat. Pro lepší motivaci lze přidat kamarády, s kterými poté můžete své výkony porovnávat.

Abstract:

Aktivitář is a web-based application that serves as a tool for recording the activities you do on a daily basis, but also for tracking the amount of calories you have consumed in a given day. These records are then displayed on various cards and graphs that make navigating the data easier. There is also the option to save your sleep duration for the last night, allowing you to have better control over your regularity. The app serves as a place where you can save all the important events that are related to a healthy lifestyle and is good for those who are looking for a way to be in control of their way of living and also for those who are looking for motivation not to stop. For better motivation, you can add friends with whom you can then compare your performance.

Zadání ročníkového projektu:

Aktivitář

Cílem ročníkové práce je vytvořit webovou stránku s názvem “Aktivitář”, která umožní uživatelům sledovat a zaznamenávat svou fyzickou aktivitu během dne. Aplikace poskytne uživatelům možnost jednoduchého zaznamenávání aktivit a počtu spálených kalorií. Klíčovým prvkem projektu je poskytnout uživatelům přehledné vizualizace, jako jsou grafy, tabulky a statistiky, které zobrazí jejich výkony v různých časových obdobích – od denních až po týdenní a měsíční. Je zde tedy možnost srovnání nynějšího výkonu s výkony z uplynulých období.

Obsah

1 Úvod

Účel naší aplikace spočívá v poskytnutí jednoduchého řešení pro zaznamenávání fyzické aktivity během dne. Tento nápad jsme si vybrali na základě našich společných zkušeností se špatnou organizací záznamů o našich fyzických výkonech a chtěli jsme tak vytvořit program, který by tento problém vyřešil.

Technologie, které jsme na vývoj aplikace využili byly *HTML, JavaScript a CSS* kterými jsme psali ve vývojovém prostředí *Visual Studio Code*. Jako framework jsme si vybrali Express.js a pro backend jsme si zvolili relační databázový model MySQL. Důvod, proč jsme si jako framework vybrali Node.js byl ten, že nám nevyhovuje framework Django a dostali jsme doporučení od pana Ing. Chalupníčka.

2 MySQL databáze

V naší ročníkové práci jsme používali relační databázi. Relační z toho důvodu, že se ideálně hodí pro práci s uživatelskými daty a záznamy, které jsou našimi uživateli ukládány. Vybrali jsme si MySQL, což je relační databáze typu DBMS (database managment system) vytvořený švédskou firmou MySQL AB, nyní je vlastníkem společnost Oracle Corporation. MySQL jsme si vybrali proto, protože část našeho týmu již měla menší zkušenost v ročníkové práci z druhého ročníku. Operace s databází jsou ošetřeny proti útokům typu *SQLinjection*[1]

2.1 Struktura databáze

2.1.1 Tabulka users

V tabulce users jsou uchovávány uživatelská data, které uživatel musí zadat při registraci do naší webové aplikace. Je zde i heslo, které uchováváme zahashované pomocí knihovny **node.bcrypt.js.**[2]

2.1.2 Tabulka activity_log

Tabulka *activity_log* je určena k ukládání všech aktivit, které si uživatel zaznamená. Aktivity jsou rozděleny do tří různých typů, podle kterého se dále pracuje s těmito záznamy v samotném kódu.

První možnost provedení struktury tabulek pro ukládání záznamů aktivit byla taková, že by existovala hlavní tabulka activity_log a k ní rozšiřující tabulky (sport, calories , sleep), které by měly určitý počet kolonek k danému typu a nedocházelo by k tomu, že některé ze záznamů by měly v kolonkách hodnotu null (např. pro záznam spánku by kolonka activ_distance měla hodnotu null).

Druhé provedení, které jsme po zvážení obou možností použili, je takové, že existuje pouze jedna tabulka *activity_log* pro všechny typy záznamů a jelikož by se problematika, kterou zmiňujeme v první možnosti provedení, týkala pouze a maximálně tří kolonek a takovéto provedení zjednodušuje operaci s databází, tak jsme zvolili právě takovéto řešení. Tabulka activity_log je nejrozsáhlejší tabulkou

[1] https://portswigger.net/web-security/sql-injection

[2] https://www.npmjs.com/package/bcrypt

a bude v ní uchováváno největší množství dat.

2.1.3 Tabulka friendpair

Naše webová aplikace umožňuje uživatelům spojit se s jejich kamarády. Dvojice přátel udržujeme právě v tabulce friendpair.

2.1.4 Tabulka friendrequest

V tabulce friendrequset jsou uloženy záznamy o tom, kdo koho požádal o přátelství.

2.1.5 Tabulka user_profile

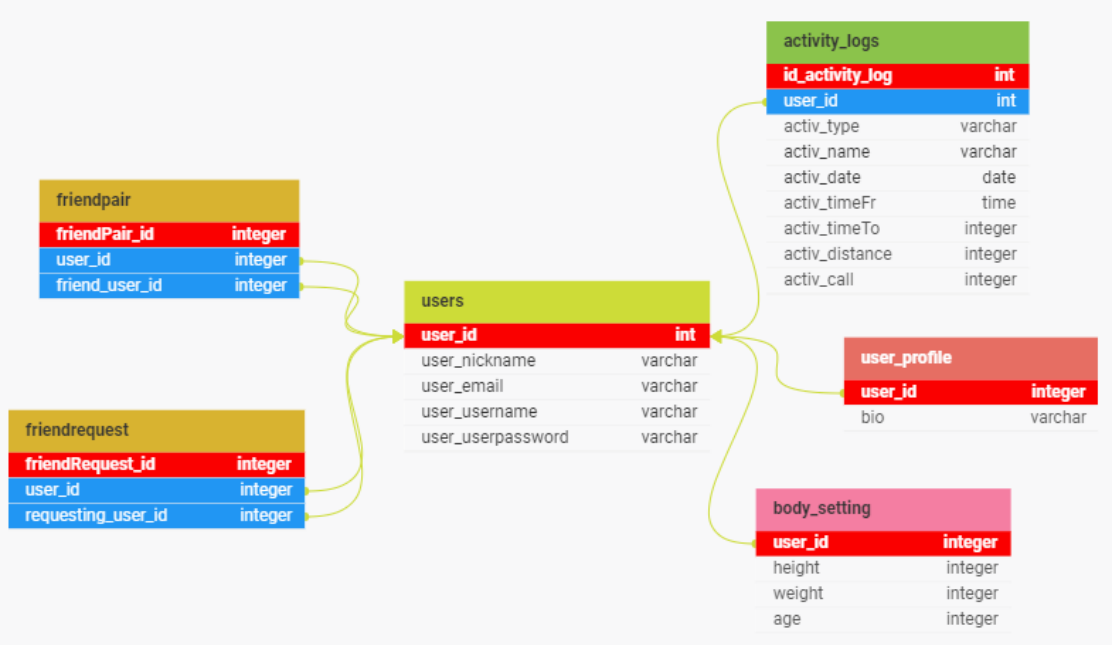
Tabulka user_profile slouží ke skladování "bia"pro konkrétního uživatele.

2.1.6 Tabulka body_settings

V tabulce body_settings jsou uchovávány informace, ze kterých se následně vypočítá **BMI** [3]uživatele.

[3]https://en.wikipedia.org/wiki/Body$_m ass_i ndex$

2.2 Schéma databáze



Obrázek 1: Schéma databáze

2.3 Database connection

Databázi hostujeme na webové stránce Aiven.io[4].

```
var mysql = require('mysql2');
var conn = mysql.createConnection({
  // host name
  host: 'aktivitar-krystof-aktivitar.b.aivencloud.com',
  port: '23949',
  user: 'avnadmin',      //  database username
  password: '',      // database password
  //( heslo nebudeme publikovat )
  database: 'aktivitar'      //  database Name
});
conn.connect(function(err) {
  if (err) throw err;
  console.log('Database is connected successfully !');
});
module.exports = conn;
```

[4]https://aiven.io/

3 Registrace, přihlašování a odhlašování

Naše aplikace využívá přihlašování formou registrací a přihlášení. Pro porozumění programu registrace a přihlášení je od toho tato kapitola.

3.1 Registrace

Registrace se skládá ze tří hlavních souborů a to *„register.ejs“*, *„confirmHandler.js“* a *„register.js“* a dále obsahuje soubory pro vzhled, které se tady popisovat nebudou.

3.1.1 register.ejs

Soubor *„register.ejs“* je taky spíš vzhledový, ale obsahuje prvky, které jsou důležité pro získání dat, mezi prvky patří např. textová pole se jmény *„registerMail“*, *„firstName“*, *„registerPass“* a *„repeatedPass“*. Do textových polí uživatel napíše svůj e-mail, který slouží k přihlašování a ověření, že uživatel už neexistuje, přezdívku sloužící k oslovování v aplikaci, případné k přidání přátel, poté heslo.

3.1.2 confirmHandler.js

Ovšem před tím než se uživateli zobrazí stránka pro registraci, tak na uživatele vyskočí okno, které se ptá, zda souhlasí s našimi smluvními podmínkami, jelikož některé informace, které dává do aplikace můžou být osobní. Vyskakovací okno je naprogramováno v souboru *„confirmHandler.js“*. Využíváme k tomu metodu z javascriptu a to metodu *confirm()*, která okno vytvoří za nás my pouze napíšeme text, který lze vidět na stránce, také okno vrátí *true* nebo *false*, tím se dozvíme, jestli potvrdil smluvní podmínky nebo ne. Pokud nesouhlasí, vrátí uživatele na předchozí navštívenou stránku, jestliže souhlasí, pokračuje dál na registraci.

3.1.3 register.js

Dejme tomu, že uživatel zadal informace do textových polí a klikl na tlačítko „Registrovat se“. Zavolá se tedy metoda POST a posbírá data z textových polí do serveru. Následně se před připraví SQL příkaz, který slouží k tomu, abychom ověřili několik možností, které by mohli způsobit programu nebo uživateli problémy.

Na takové problémy jsme využili *'express-validator'*[5], kterému jsme řekli, aby zkontroloval, zda e-mail je validní a zkontroloval, že heslo obsahuje alespoň 6 znaků. Dále jsme ověřovali, zda již uživatel není náhodou registrován nebo jestli nenapsal opakované heslo špatně[6]. Jelikož uživateli přiřazujeme #, aby mohlo být více uživatelů se stejným jménem a mohli se přidávat na základě jména, tak kvůli dalšímu zpracování dat v program, je potřeba ověřit, že jméno neboli přezdívka neobsahuje právě zmíněný znak. Nejsou-li podmínky splněny přesměruje to uživatele zpátky na registraci společně s upozorněním.

Pokud jsou podmínky splněny, tak se pomocí knihovny *'bcrypt'* zašifruje heslo pomoci hash funkce. Následně se vytvoří v metodě *generateUniqueString()* unikátní řetězec[7], který se přiřadí ke jménu uživatele, což tvoří uživatelské jméno.

```
1 function generateUniqueString() {
2   const uuid = uuidv4();
3   const uniqueString = uuid.replace(/-/g, '').substring(0, 5);
4   return uniqueString;
5 }
```

Listing 1: unikátní řetězec

Jakmile je vše připravené, tak se upravené informace a informace, které nebylo potřeba upravit, vloží do databáze. Když vložení do databáze proběhne v pořádku, tak uživatele přesměruje na přihlášení s upozorněním, že byl úspěšně registrován.

[5]https://www.npmjs.com/package/express-validator
[6]https://codingstatus.com/create-registration-and-login-form-in-node-js-mysql/
[7]https://www.npmjs.com/package/uuid

3.2 Přihlášení

Přihlášení se skládá také z mnoha souborů, ale pro nás nejdůležitější budou *„login.js“*, *„passport-config.js“*. Zbylé soubory se týkají vzhledu, případně co bylo předdchozí podkapitole.

3.2.1 login.js

Soubor obsahuje několik metod, které slouží především jako před příprava pro soubor *„passport-config.js“* a mnohé metody se používají až v jiném souboru[8]. Jde většinou o metody, které porovnávají data z textového pole s daty z databáze jako je např. funkce *findUserByEmail*, která nalezne uživatele za pomoci e-mailu. Také jsou i metody, které se využívají rovnou v souboru například *handleLogin*, která se ujistí, že existuje a následně porovná zašifrovaná hesla. Metoda POST nastane poté, co uživatel klikne na tlačítko „Přihlásit se“, a přesměruje uživatele na stránku přehledu po úspěšném přihlášení, po neúspěšném přihlášení přesměruje zpět na login stránku, kdde bude upozornění, co uživatel udělal při přihlašování špatně.

3.2.2 passport-config.js

V tomto souboru využíváme knihovnu *'passport'* a její strategii *'passport-local'*[9]. Lokální strategii jsme zvolili, kvůli tomu, že pracuje s vlastní databází pro uživatele. Jinak soubor slouží podobně jako *„login.js“* před přípravě, akorát se to tu kombinuje. Metoda je tu pouze jedna slouží k zpracování dat a následně co se pošle prohlížeči.

Metoda se volá v *„server.js“* a zpracovává se nějak takto:

```
1 const methodOverride = require('method-override')
2 const flash = require('express-flash')
3 const session = require('express-session')
4 const passport = require('passport')
5
6 app.use(flash())
7 app.use(session({
8   secret: 'secret',
```

[8] https://github.com/haryphamdev/loginSystemNodeJSFinal/blob/master/src/

[9] http://www.passportjs.org/packages/passport-local/

```
9    resave: false,
10   saveUninitialized: false,
11 }))
12 app.use(passport.initialize());
13 app.use(passport.session());
14 app.use(methodOverride('_method')
```

Listing 2: inicializace passport

Z ukázky je možné vidět, že používáme další knihovny a to *'method-override'*, *'express-flash'* a *'express-session'*. Flash slouží na upozornění uživatele na chyby při registraci nebo přihlašování. Session používáme, abychom věděli jestli je uživatel přihlášen nebo ne, je to také kvůli odhlašování.

3.3 Odhlášení

Pro odhlášení je v *„server.js“* jedna metoda, ale pro kontrolu přihlášení jsou tu další dvě.

```
1 app.delete('/logout', (req, res) => {
2    req.logout(function(err) {
3       if (err) { return next(err); }
4       res.redirect('/');
5    });
6 });
```

Listing 3: odhlášení

Na to aby se uživatel odhlásil je tlačítko, které „nenápadně“[10] přesměruje uživatel na logout stránku, která smaže data potřebné k přihlášení ve prohlížeči a přesměruje uživatele zpět na úvodní stránku.

[10]pozn. Je to řečeno hodně zjednodušeně. Ve skutečnosti je to trochu složitější, ale pro naše chápání to postačí. Dobře popsáno to je na této stránce: https://www.geeksforgeeks.org/express-js-app-delete-function/

```
function checkAuthenticated(req, res, next) {
    const userName = require('./routes/userID')
    if (!req.isAuthenticated()) {
      return res.redirect('/login');
    }
    userName.setUser(req.user.user_id)
    next();
 }
```

Listing 4: kontrola-přihlášení

Tato metoda kontroluje zda je uživatel přihlášen, kdyby nebyl a chtěl se dostat například na stránku přehledu, tak ho to nepustí a přesměruje ho to na login stránku. Pokud je však přihlášen pokračuje dále na stránce, zároveň uložíme uživatelovo ID do proměnné, abychom s tím mohli dál pracovat. Metodu přikládáme ke každé GET metodě pro stránku, na kterou nechceme aby se dostal nepřihlášený uživatel.

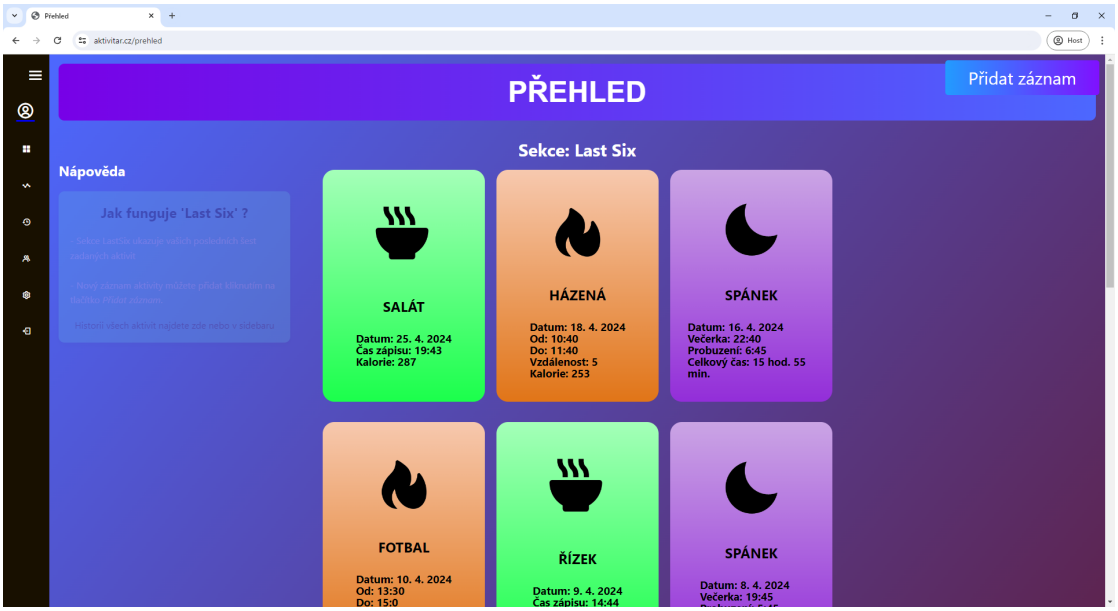
```
function checkNotAuthenticated(req, res, next) {
    if (req.isAuthenticated()) {
      return res.redirect('/prehled');
    }
    next();
 }
```

Listing 5: kontrola-odhlášení

Tato funkce řeší pravý opak předchozí funkce. Tedy pokud je někdo přihlášen, tak nechceme, aby se znovu přihlašoval či dokonce registroval, proto uživatele přesměrujeme rovnou na stránku přehledu. Znovu funkci přidáváme k metodě GET pro stránku, kde bude plnit svoji funkci

4 Přehled

Stránka přehled je hlavní scénou naší aplikace. Přehled se zobrazí hned po přihlášení a jsou zde nejdůležitější záležitosti, které by uživatele mohly zajímat. Jako první je zde takzvaná sekce ,,Last Six", která už svým jménem vypovídá o jejím obsahu. Vpravo nahoře se nachází tlačítko pro přidání záznamu. Tlačítko přidat záznam zobrazí pop-up menu, kam uživatel zadá potřebné informace k přidáni aktivity. Dále je zde BMI uživatele a jako poslední sekcí je graf poslední aktivity. Všechny tyto sekce rozebereme.



Obrázek 2: Scéna Last Six

4.1 Last Six

Sekce Last Six zobrazuje posledních šest aktivit. K načtení posledních šesti aktivit používáme funkci *getLastSix()*. V databázi těchto šest záznamů hledáme v tabulce *activity_log* pomocí user_id (id uživatele), řadí se podle sloupečku *activ_date* (datum aktivity) a vezme se maximálně šest záznamů.

Funkce *getLastSix()* vrací pole výsledků, které pomocí metody fetch() v sobouru lastSixHandler.js zpracováváme a následně zobrazujeme v kartičkách do prohlížeče.

```
1    function getLastSix(){
2     var userID = user_id.getLogUserId();
```

```
3
4       return new Promise((resolve, reject) => {
5           var sql = "SELECT activ_type, activ_name,
6           activ_date, activ_timeFr, activ_timeTo,
7           activ_distance, activ_cal from activity_log
8           WHERE user_id = ? ORDER BY activ_date DESC LIMIT 6;";
9           db.query(sql, [userID], function (err, result, fields) {
10              if (err) {
11                console.log(err);
12                reject(err);
13              }
14              resolve(result);
15            });
16          });
17      }
```

Listing 6: Funkce getLastSix()

4.1.1 LastSixHandler.js

V souboru *lastSixHandler.js* pracujeme s načteným polem posledních šesti aktivit ze souboru *server.js*. Pomocí metody *forEach()* postupně načítáme jednotlivé řádky pole s informacemi ke konkrétní aktivitě. Podle typu aktivity generujeme kartičku konkrétního typu aktivity. Datum a čas před vložením do kartičky aktivity je nutné upravit, aby jejich tvar odpovídal správnému formátu. K tomu slouží funkce *displayTimeWS(timeString)* (čas se zobrazí bez sekund) a *calculTime(time1, time2)* (pro zobrazení celkového času aktivity).

Funkce *displayTimeWS(timeString)* a *calculTime(time1, time2)* byly obě vytvořené umělou inteligencí, konkrétně ChatGPT[11] („How to calculate time between these two strings in nodejs var string1 = '11:20'; var string2 = '12:50'? ; „How to split time string in minutes and hours")

```
lastSix.forEach(result => {
    var newFrame;
    const listItemSport = document.createElement('div');
    if(result.activ_type=='sport'){
        newFrame = aktivitaFrameTemplate.cloneNode(false);

        listItemSport.innerHTML = '<div id="sport" class="card">
        <div class="card-image-sport"> </div>
        <div class="card-title">'+result.activ_name+'</div>
        <div class="card-text">
        <h3 id="1label111">Datum:  '+new
        Date(result.activ_date).toLocaleDateString('cs-cz')+'</h3>
        <h3 id="1label21">Od:'+displayTimeWS(result.activ_timeFr)+'</h3>
        <h3 id="1label12">Do:'+displayTimeWS(result.activ_timeTo)+'</h3>
        <h3 id="1label13"> Vzd lenost:'+result.activ_distance+'</h3>
        <h3 id="1label14"> Kalorie:'+result.activ_cal+'</h3>
        </div></div>'
        newFrame.appendChild(listItemSport);
        pocetpolozekvDB += 1;
    }else if (result.activ_type == 'call'){...
```

Listing 7: lastSixHandler.js generování kartiček

[11] https://chat.openai.com/

4.2 BMI

Jako další sekcí v přehledu je vaše BMI. Je zde pouze informativně a pracovat se s ním dá dále v nastavení. BMI načítáme pomocí funkce *getBodySetting()*, která z databáze vrátí BMI uživatele.

4.3 Graf poslední aktivity

Poslední sekcí, kterou v přehledu najdete je graf poslední aktivity. Graf funguje na stejném principu jako grafy ve statistikách (viz. kapitola 6 Statistiky), ale s tím rozdílem, že nezobrazujeme grafy přátel. Jelikož bereme pouze jednu aktivitu a to poslední přidanou, tak jediné, co je potřeba zkontrolovat, je typ aktivity.

V případě sportu jsme zvolili pouze graf spálených kalorii, protože je to nejdůležitější informace pro uživatele z těch tří možností. Graf nabraných kalorii spánku je stejný jako ve statistikách. Jestliže nemá zatím přidanou aktivitu, pole, které posílá data ze serveru, má hodnotu null. V takovém případě se místo grafu zobrazí text *„Přidejte aktivitu, aby jste viděli graf“*.

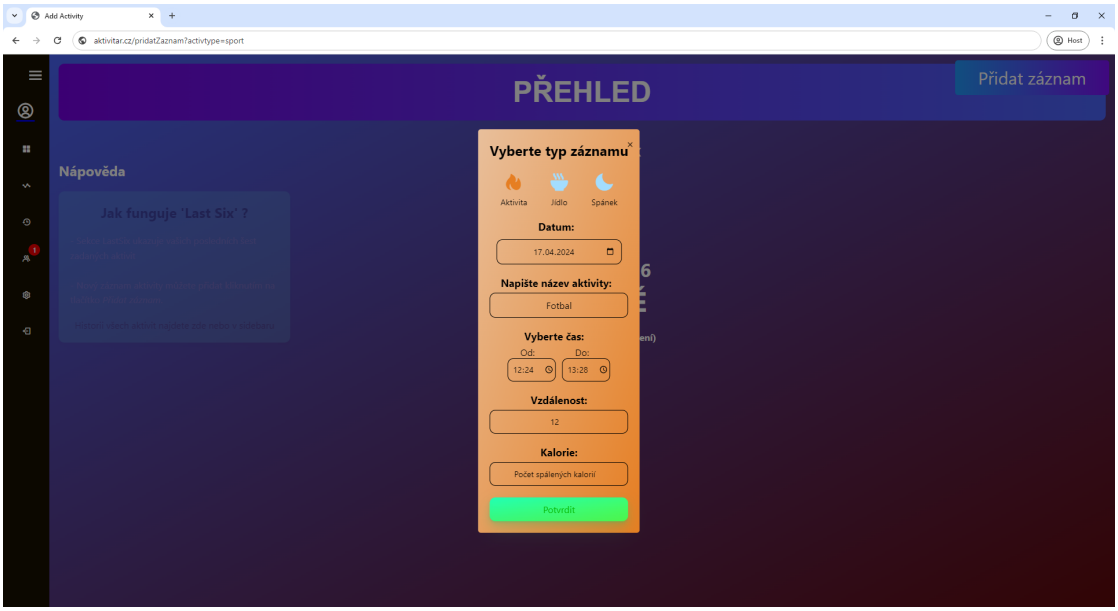
5 Přidání aktivity

Jedna z našich nejhlavnějších dovedností naší webové aplikace je přidání aktivity. Uživatel má na výběr ze tří typů aktivit.

Prvním typem je sport. V aktivitě typu sport uživatel uloží jméno nebo druh aktivity (např. Fotbal), datum kdy aktivita probíhala, čas kdy s aktivitou začal a kdy ji ukončil, počet kilometrů, kterých během aktivity dosáhl a počet spálených kalorií.

Druhým typem je aktivita typu kalorie. Zde si může uživatel vést záznamy o konzumaci jednotlivých jídel. Konkrétně počet kalorií, jméno jídla, datum a čas konzumace.

Posledním typem je aktivita spánek. U spánku si uživatel nastaví datum a časy, kdy šel spát a kdy se vzbudil.



Obrázek 3: Scéna přidat záznam

5.1 Funkce přidání aktivity

Po kliknutí na tlačítko přidat záznam se zobrazí pop-up menu, ve kterém si uživatel vybere, jaký typ aktivity si přidá. Má na výběr z aktivity typu sport, kalorie nebo spánek. Po vyplnění všech kolonek a kliknutí na tlačítko potvrdit se

daná aktivita uloží do databáze. Toto tlačítko uživatele doprovází na některých ostatních scénách. Funkčnost přidání aktivity se řeší v souboru *pridatZaznam.js*.

5.2 pridatZaznam.js

Přidání aktivity řešíme metodou post. Z *query-string* dostaneme typ aktivity *req.query.activtype* a funkce *getLogUserId()* nám vrátí user_id uživatele. Z vyplněných kolonek načteme data do konstanty *inputData*.

```
1         //nacteni dat pro sport
2         const inputData = {
3             activ_name: req.body.activName,
4             activ_date:  req.body.activDate,
5             activ_timeFr: req.body.activFrTime,
6             activ_timeTo: req.body.activToTime,
7             activ_distance:  req.body.activDistance,
8             activ_cal:  req.body.activCal      }
```

Listing 8: Načtení dat - přidání záznamu

Dalším krokem je uložení dat do databáze.

```
1         //ulozeni dat do DB pro sport
2         var sql = "INSERT INTO activity_log (user_id,
3          activ_type,activ_name, activ_date, activ_timeFr,
4          activ_timeTo, activ_distance, activ_cal)
5          VALUES  (?,?,?,?,?,?,?,?)";
6
7         db.query(sql,[user_id,typ_activ,inputData.activ_name,
8         inputData.activ_date,inputData.activ_timeFr,
9         inputData.activ_timeTo, inputData.activ_distance,
10        inputData.activ_cal], function(err,result){
11            if(err) throw err
12            console.log("Activity add correctly?");
13        });
```

Listing 9: Uložení dat do databáze - přidání záznamu

6 Statistiky

V aplikaci si uživatel zapisuje různá data nějaké aktivity. Pro vytvoření vzhledové statistiky jsme zvolili grafy a knihovnu *ChartJS*[12]. Také jsme zkoušeli i jiné knihovny pro vytvoření grafu na serverové straně, a poté poslat fotku uživateli, ale většinou nefungovali správně nebo jsou zastaralé a pro dnešní prohlížeče nefunkční.

6.1 Informace z databáze

Pro to abychom získali informace z databáze jsme rozdělili program do několika metod rozdělených do několika souborů a hlavními soubory jsou *„getActivityNames.js“*, *„dataGraph.js“*. Data se pak pošlou do prohlížeče a následně zpracují pro graf.

6.1.1 getActivityNames.js

Soubor je určen k získání aktivit našeho uživatele. Aktivity se zapíší do pole *„data“* a jsou rozděleny na základě typu aktivity do tří skupin „sport“, „cal“ a „sleep“. U kategorie sport se vytvoří *Set()*, kvůli tomu že uživatel může mít vícekrát přidanou aktivitu se stejnou aktivitou a my potřebujeme název aktivity pouze jednou. Pro kategorii sleep a cal se uloží pouze jedna aktivita, protože pro zpracovaní dat v těchto případech je pro nás důležité pouze to, jestli tam je nebo není.

6.1.2 dataGraph.js

Soubor jsme rozdělili do dvou metod, které mají funkci získat taková data z databáze, abychom vytvořili graf.

Ve funkci *getDBInfo()* vytváříme takzvaný *Promise()*[13], který využíváme skoro ve všech metodách, které získávají data z databáze, protože někdy je dat mnoho a získání dat trvá, proto potřebujeme nějak asynchronně počkat až přijdou data a následně s nimi pracovat. Dále vytváříme „slovník“ pro pět polí „labels“, „valuesBurnCal“, „valuesDis“, „time“, „friendsData“. Označení „labels“ je pro souřadnice x-ové osy[14], následující grafy jsou data pro y-ovou osu. Pro tři různé grafy jsou „labels“

[12] https://www.chartjs.org/docs/latest/

[13] https://javascript.info/async-await

[14] pozn. v ChartJS se označují takhle pole souřadnic pro osu x, pro y jsou to pak data

stejné, ale „data“ jiné, proto používáme tři pole, které nesou data, podle jejich názvů např. „time“ je pole doba trvání aktivity v milisekundách, která se vzpočítá ve funkci *calculateTime(časOd, časDo)*.

```
1  function calculateTime(timeFrom, timeTo){
2      let start = new Date("1970-01-01T" +timeFrom).getTime()
3      timeFrom = timeFrom.split(':')
4      if(timeTo != null){
5          let end = new Date("1970-01-01T" +timeTo).getTime()
6          return end - start
7      }else{
8           return start
9      }
10 };
```

Listing 10: odhlášení

Funkce *calculateTime()* vytvoří z času, co zadal uživatel k aktivitě, datum, od kterého se počítá čas, a převede se na milisekundy, poté se časy odečtou a získáme čas trvání aktivity.

Na základě parametru(jména aktivity) rozdělíme SQL příkaz na dva, následně se výsledky pro danou aktivitu uloží do pole a pokud aktivita se týká sportu, tak se zavolá druhá metoda, ta dělá to samé, ale pro všechny přátele uživatele, jestliže mají stejnou aktivitu také zaznamenanou. Jakmile je vše hotové v pořádku funkce vrátí slovník „info“ a vytvoří nový slovník pro další aktivitu.

V *„server.js“* se volají všechny potřebné funkce, vyčká se na data a pošle se prohlížeči.

6.2 Problémy

S volbou *ChartJS* přichází mnoho problému, které probereme společně s řešením těchto problémů.

6.2.1 Poslat pole prohlížeči

Jedním problémem bylo, že *ChartJS* je pro javascript v prohlížeči, ale data získáme na serveru. Tedy data poslat ze serveru do prohlížeče.

Na problém jsem se zeptal AI[15]: „Jakým způsobem a za pomocí jaké knihovny dostanu data ze serveru do prohlížeče?“ odpověděl, ukázkou globální funkcí *fetch()*[16].

```
1 fetch('/statistiky',  {headers: {
2     'Accept': 'application/json'
3   }})
4     .then(response => response.json())
5     .then(info => {makeHtml(info)})
6     .catch(error => console.error('Error:', error));;
```

Listing 11: metoda fetch

Jako první parametr je ze které stránky má data očekávat, dále co má očekávat. Data uloží do proměnné a v našem případě zavoláme funkci, kde ty data zpracujeme.

6.2.2 Uživatel

Uživatel může způsobit mnoho nehezkých potíží, které jsme museli ošetřit tyto chyby jsme především ošetřili v souboru *„activityGraph.js“*.

Uživatel nebo uživatelův kamarád vynechá datum nebo přidá předchozí starší data. Abychom tomu zamezili, vytvořili funkci *makeDateArray()*, která najde minimální hodnoty z dat uživatele a jeho kamarádů. Z minimální a maximální hodnoty se vygenerují data[17], mezi nimi a vrátí pole. Pole následně vrátí *makeDateArray()*. Ovšem mezi roky může být rozdíl několik let a tak velké pole už *ChartJS* nezpracuje, proto minimální hodnota může mít rozdíl s maximální okolo jednoho roku.

Přiřazení správně souřadnic bylo problémové, protože *ChartJS* bere souřadnice z pole na stejném indexu. Tedy k nově vygenerovaným datem přiřazujeme hodnotu null. Nastává další problém a to, že hodnoty stále nejsou na správném indexu. K tomu využíváme takzvaných *Map()*, kde nalezneme pomocí funkce *findIndex()*[18], která nám pomůže přiřadit hodnoty na stejný index jako má datum, ke kterému patří. Toho samozřejmě musíme vytvořit i přátel, aby starší datum nebylo více vpravo oproti dnešnímu dni.

[15] https://chat.openai.com/

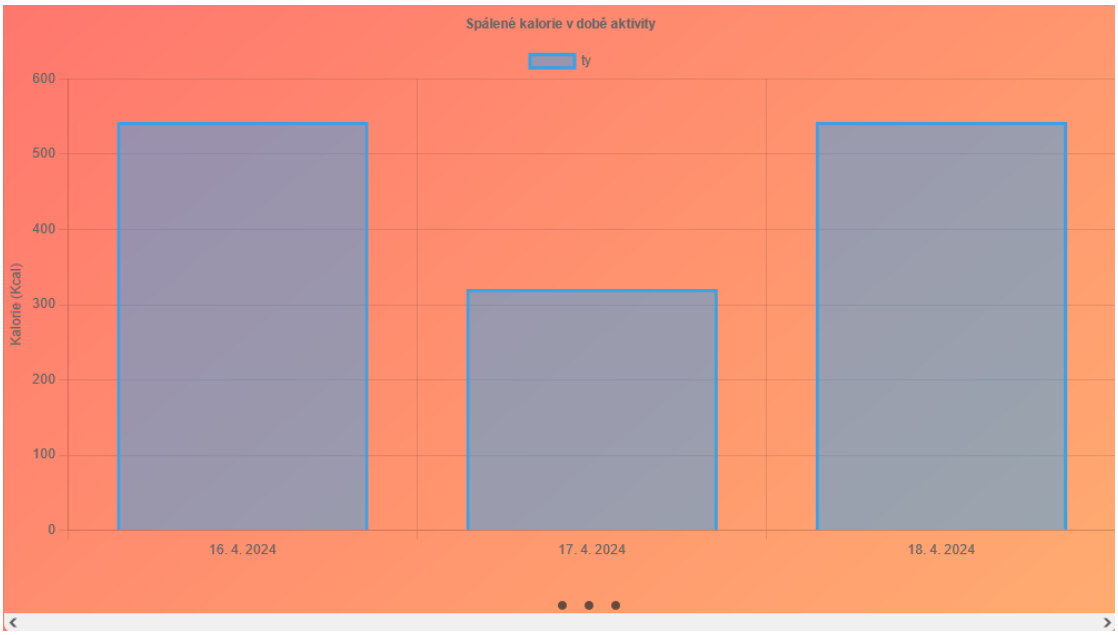
[16] https://developer.mozilla.org/en-US/docs/Web/API/Fetch

[17] https://geekybeginners.com/generate-dates-between-two-dates-in-javascript/

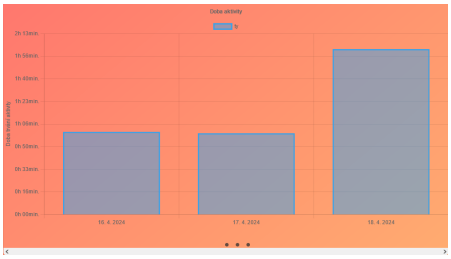
[18] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/findIndex

6.3 Finální vzhled

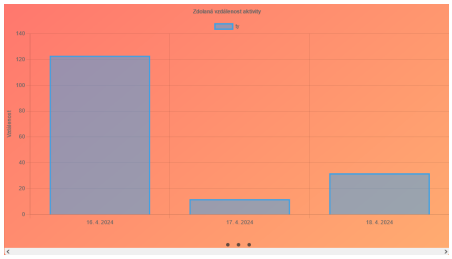
Vzhled grafů je tvořen v CSS, ale jednotlivé přípravy probíhají právě v *„activityGraph.js“* v metodách *makeHtml()* a *createGraph()*. V *makeHtml()*[19][20] je vytvoření prostoru pro graf, který má tři typy. V *createGraph()* se vytváří grafy na základě typu aktivity.[21]



Obrázek 4: graf sportovní aktivity(spálené kalorie)



(a) graf doby trvání



(b) graf zdolané vzdálenosti

Obrázek 5: graf sportovní aktivity(vzdálenost, trvání)

V obr. 2 můžeme vidět tři tečky ty slouží k přepínání mezi grafy a jsou pouze pro graf typu sport. Dále je trochu hůře vidět lišta pod grafem, která slouží k

[19]https://www.youtube.com/watch?v=McPdzhLRzCg

[20]https://www.youtube.com/watch?v=oa0$_E$sFzxBg

[21]pozn. graf pro jídlo je možné vidět na stránce https://aktivitar.cz/ a graf pro spánek je stejný pouze fialovou barvou.

tomu když uživatel přidá mnoho aktivit, tak aby se graf nezmenšoval a byl čitelný a uživatel mohl projít všechny data.

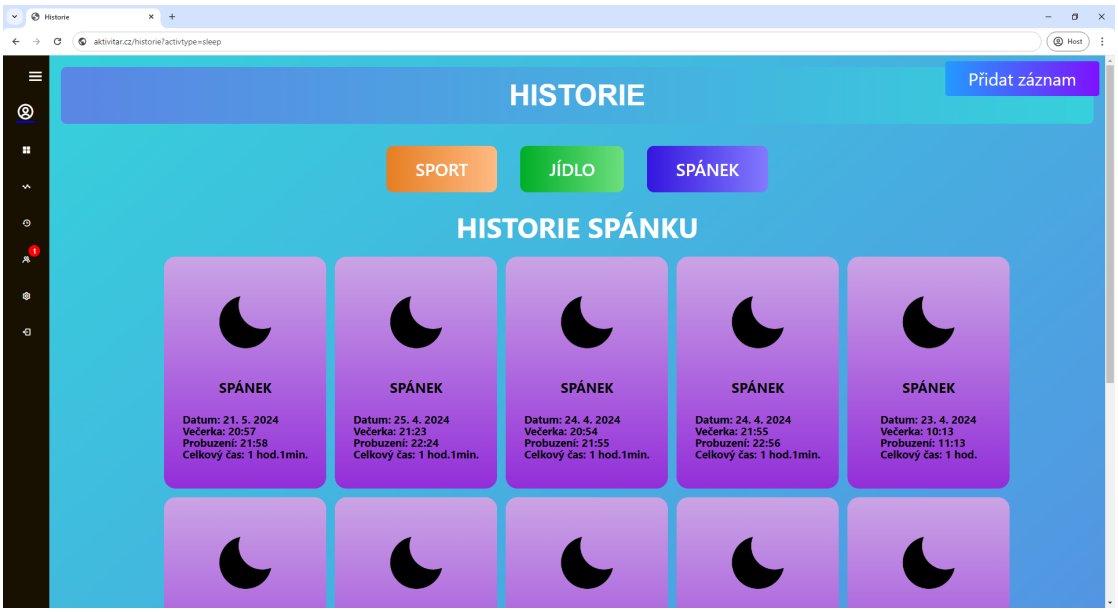
Pro vyhledávání mezi grafy je v souboru *„statistikyHandler.js“* funkce *searchFunction()*, která vyhledává podle názvu aktivity. Název aktivity je v *h2* tagu, takže hledá podle *h2* a tuho schovává grafy aktivit, které nemají takové písmeno v názvu.

```
1  function searchFunction() {
2    var input, filter, divs, h2, txtValue;
3    input = document.getElementById('searchInput');
4    filter = input.value.toUpperCase();
5    divs = document.getElementsByClassName('slider-wrapper');
6    for (var i = 0; i < divs.length; i++) {
7      h2 = divs[i].getElementsByTagName('h2')[0];
8      txtValue = h2.textContent || h2.innerText;
9      if (txtValue.toUpperCase().indexOf(filter) > -1) {
10       divs[i].style.display = "";
11     } else {
12       divs[i].style.display = "none";
13     }
14   }
15 }
```

Listing 12: funkce pro vyhledávání ve statistikách

7 Historie

Historie je rozdělena na jednotlivé typy aktivit a to na sport, kalorie a spánek. Uživatel zde vidí všechny záznamy, které k vybranému typu aktivity přidal. Po kliknutí na kartičku aktivity lze konkrétní záznam upravit, či smazat.



Obrázek 6: Scéna historie

7.1 Zobrazení historie

K zobrazení historie používáme funkci *getActivities()* s parametrem *typ_activ*, který určuje o jaký typ aktivity se jedná. Tato funkce následně vrátí všechny aktivity daného typu, které si uživatel kdy uložil. Jednotlivé záznamy (řádky v databázi) se pošlou ve formátu *json* pomocí metody *fetch()* do souboru *historieHandler.js*, kde se následně generují jednotlivé kartičky, které se zobrazí uživateli.

```
1     function getActivities(typ_activ){
2       var userID = user_id.getLogUserId();
3
4       return new Promise((resolve, reject) => {
5         var sql = "SELECT activity_log_id,activ_type, activ_name,
6         activ_date, activ_timeFr, activ_timeTo, activ_distance,
7         activ_cal FROM activity_log
8         WHERE user_id = ? AND activ_type = ?
9         ORDER BY activ_date;";
10
11        db.query(sql, [userID, typ_activ],
12        function (err, result, fields) {
13            if (err) {
14              console.log(err);
15              reject(err);
16            }
17            resolve(result);
18          });
19        });
20  }
```

Listing 13: Funkce getActivities(typ_activ)

Základně je nastaveno, aby se zobrazila historie sportu, jelikož je to pro uživatele příjemnější, než kdyby se mu nezobrazil ani jeden záznam.

7.2 Změna aktivity

Pro jakoukoliv úpravu aktivity stačí kliknout na konkrétní kartičku a poté se zobrazí "pop-up", kde uživatel může upravit jednotlivé informace z dané aktivity a poté ji znovu uložit. Změnu záznamu řešíme pomocí metody post, která se odbavuje v souboru *zmenaZaznam.js* . Sloupec activity_log_id je primárním klíčem v tabulce activity_log, který nám umožňuje velice jednoduché pracování s jednotlivými řádky neboli záznamy. Právě pomocí tohoto primárního klíče řešíme úpravu a smazání konkrétní aktivity. Po kliknutí na kartičku se hodnota id uloží do *query stringu*[22] do parametru id a v souboru *zamenaZaznam.js* se lehce použije při ukládání změn dat do databáze.

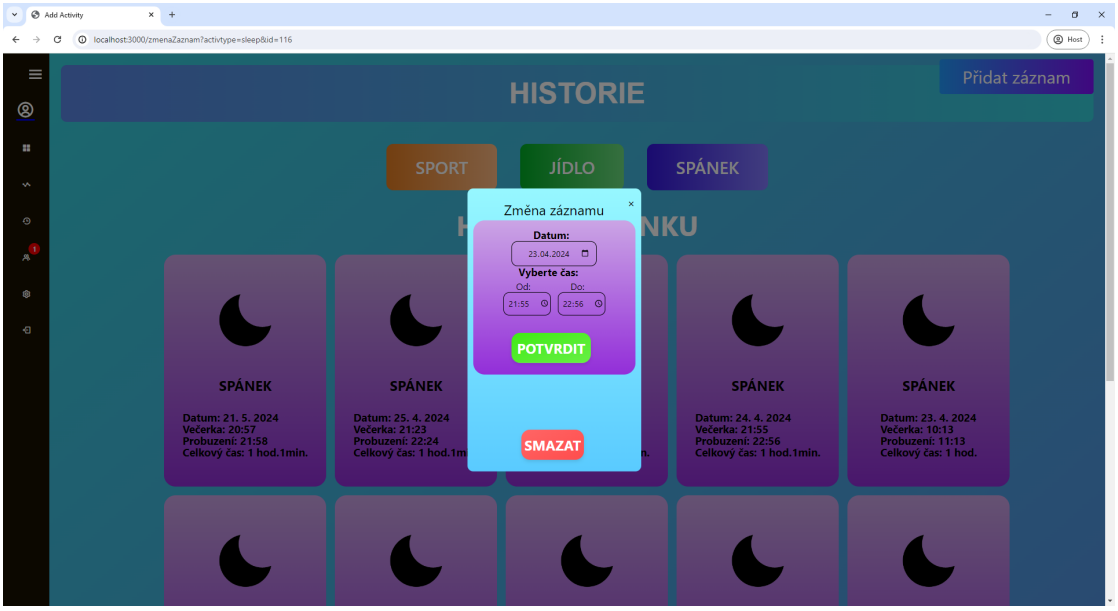
[22]https://stackoverflow.com/questions/724526/how-to-pass-multiple-parameters-in-a-querystring

```
if(activ_type == "sport"){
        //zmena zaznamu typu sport
        let sql = "UPDATE activity_log SET activ_name = ?,
        activ_date = ?,activ_timeFr = ?, activ_timeTo = ?,
        activ_distance =?, activ_cal = ?  WHERE activity_log_id = ?;"
        db.query(sql,[inputData.activ_name, inputData.activ_date,
        inputData.activ_timeFr, inputData.activ_timeTo,
        inputData.activ_distance, inputData.activ_cal, activ_log_id ],
        function(err, result, fields){
            if (err) throw err
            console.log("Change was succesfull");
        })
    }
```

Listing 14: Změna záznamu v databázi

7.3 Smazání aktivity

Smazání aktivity funguje naprosto ekvivalentně, jako změna aktivity. Zde však nepotřebujeme znát typ aktivity, tedy nám stačí pouze activity_log_id pro vymazání konkrétní aktivity.



Obrázek 7: Scéna změny aktivity

8 Přátelé

Chtěli jsme uživateli umožnit porovnání výkonu svého s výkonem svých přátel, proto jsme do aktivitáře přidali sekci přátel.

8.1 Vyhledávání přátel

Vyhledávání funguje tak, že při vyhledávání se do URL připisuje takzvaný *„query string“*, který potom voláme na serveru pomocí *req.query.term*. Query string následně zpracujeme ve funkci, takovým způsobem, že vyhledáváme jména přátel, kteří obsahují znaky, které jsou ve query stringu, následně seznam přátel pošleme v JSON souboru jako pole. Jména zobrazíme pod vyhledávacím panelem a po kliknutí na jedno z jmen se vybrané jméno automaticky přesune do vyhledávacího panelu.

8.2 Přidání přítele

Pokud uživatel vyhledal uživatele, kterého si chce přidat mezi přátelé, klikne na tlačítko *„Přidat“*.

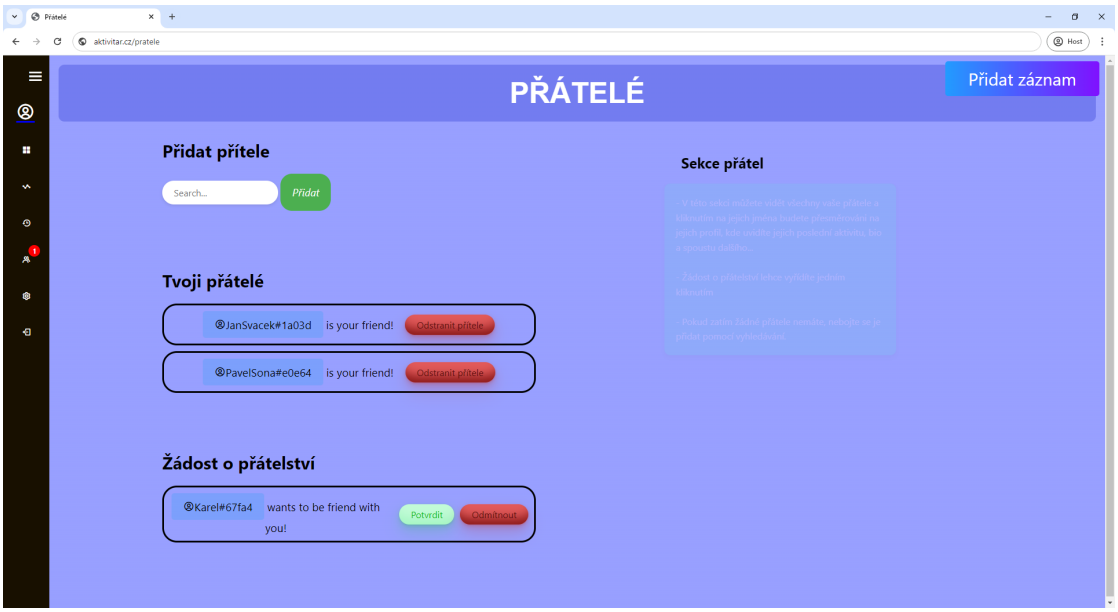
Předtím než v databázi vytvoříme žádost, musíme zkontrolovat, jestli uživatel:

- a) chtěl přidat neexistujícího uživatele.
- b)přidal sám sebe.
- c)přidal uživatele, který je jeho kamarád.
- d)jestli už žádost neodeslal.

Když je vše v pořádku vytvoří se v tabulce *friendrequest* záznam, který drží informaci který uživatel požádal které uživatele o přátelství. V tento moment se čeká na vyřízení žádosti druhým uživatelem. Kdyby vše v pořádku nebylo, tak v prvních dvou případech vyskočí na uživatele upozornění a ve zbylých dvou případech se záznam nevytvoří.

8.3 Seznam přátel

Jakmile uživatel potvrdí žádost, tak se vytvoří do databáze kamarádský pár a smaže se žádost v databázi. Ze seznamu se kamarádských párů se v metodě *showMyFriends()* vezmou jména přátel a vloží do pole. Pole se pošle do prohlížeče a zobrazí se jména všech přátel společně s tlačítkem *„odstranit přítele“*. To slouží k tomu, kdyby uživatel se již nechtěl kamarádit s jiným uživatelem. Po kliknutí se smaže pár s daným přítelem z databáze.



Obrázek 8: Scéna přátel

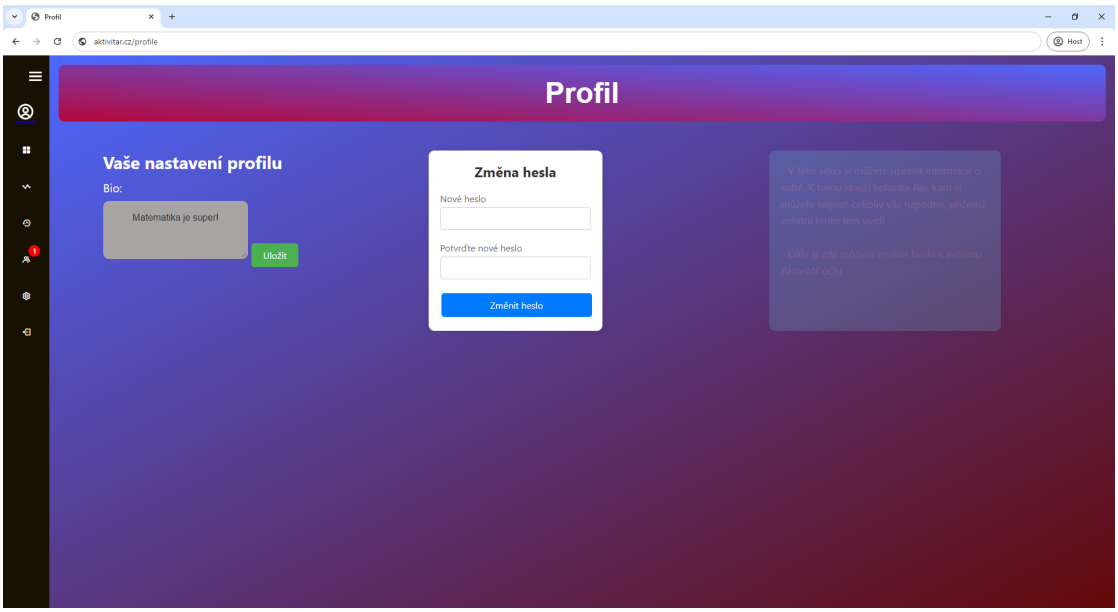
Mít přidané kamarády přináší možnost vidět graf společných aktivit nebo dokonce jeho profil (viz. kapitola 9.2 Profil přítele). V grafech pak uživatel vidí všechny data všech přátel a může tedy porovnávat svoje výkony s přáteli.

9 Profil

Abychom více zhodnotili možnost přátel, inspirovali jsme se od různých sociálních sítí a do naší webové aplikace jsme přidali sekci ”profile”a ”friend profile”.

9.1 Uživatelův profil

V sekci profile si uživatel může vytvořit své bio, které uvidí jeho ostatní přátelé. Do bia si může napsat cokoliv, co bude chtít např. něco o sobě, co má rád atd. Dále je zde uživateli umožněno změnit své heslo na nové. Přidání bia a změna hesla se řeší v souboru profile.js.



Obrázek 9: Scéna profil

Bylo by na místě uvážit zlepšení změny hesla. Uživatel by mohl heslo změnit teprve po správném zadání aktuálního hesla.

```
router.post('/saveBio', function(req,res) {
    var userID = user_id.getLogUserId()
    var bio =  req.body.BioTextarea;
    var sql = "INSERT INTO user_profile VALUES (?,?)"
    db.query(sql, [userID, bio], function(err, result){
        if (err){
            var sqlUpdate = 'UPDATE user_profile SET bio = ?
            WHERE user_id = ?;'
            db.query(sqlUpdate,[ bio, userID], function(err,result){
                if(err) throw err
                console.log("Body updated correctly");
            });
        }
    })
    res.redirect('/profile')
})
```

Listing 15: Uložení bia

9.2 Profil přítele

V kapitole přátelé se dočtete, že po kliknutí na jméno vašeho přítele vám aplikace ukáže profil tohoto uživatele. Profil přítele jsme navrhli tak, aby se zde ukázalo jeho bio, šest posledních aktivit a graf poslední aktivity.
Bio přítele načteme pomocí funkce *getFriendBio(frienduserName)*. Pokud uživatelův přítel nemá bio vytvořené, do bia v prohlížeči se zobrazí nápis:,, Tento uživatel nemá vytvořené bio ". Načtené bio poté pošleme při renderování stránky *friendProfile* do proměné **bio**.

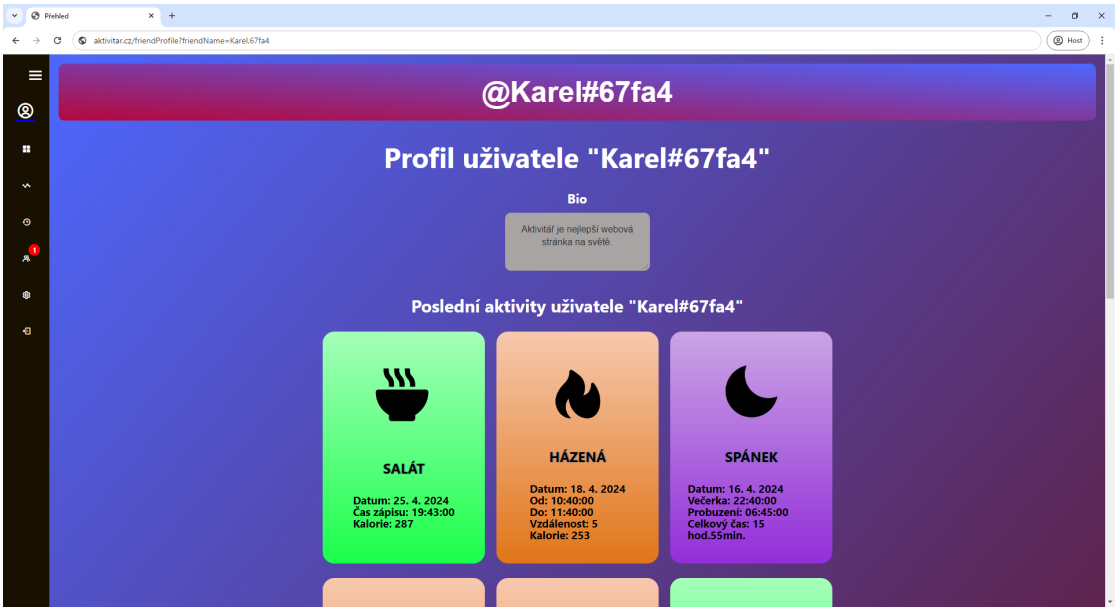
```
 const bio = await user_profile.getFriendBio(friendName);

   var bioRespond;

   if(bio==null){
     bioRespond = 'Tento u ivatel nem  vytvo en  bio';
   }else{
     bioRespond = bio[0].bio;
   }
```

```
10 }
```

Listing 16: Nahrání bia (server.js)

```
1 res.format({
2     html: async () => {
3     res.render('friendProfile', {name: nameUser[0], username: nameUser[1],
4     bio: bioRespond, friendName: friendName,
5     notificationsCount: await lengthNoti()}); // Render HTML
6        },
```

Listing 17: Renderování stránky Friend pofile (server.js)



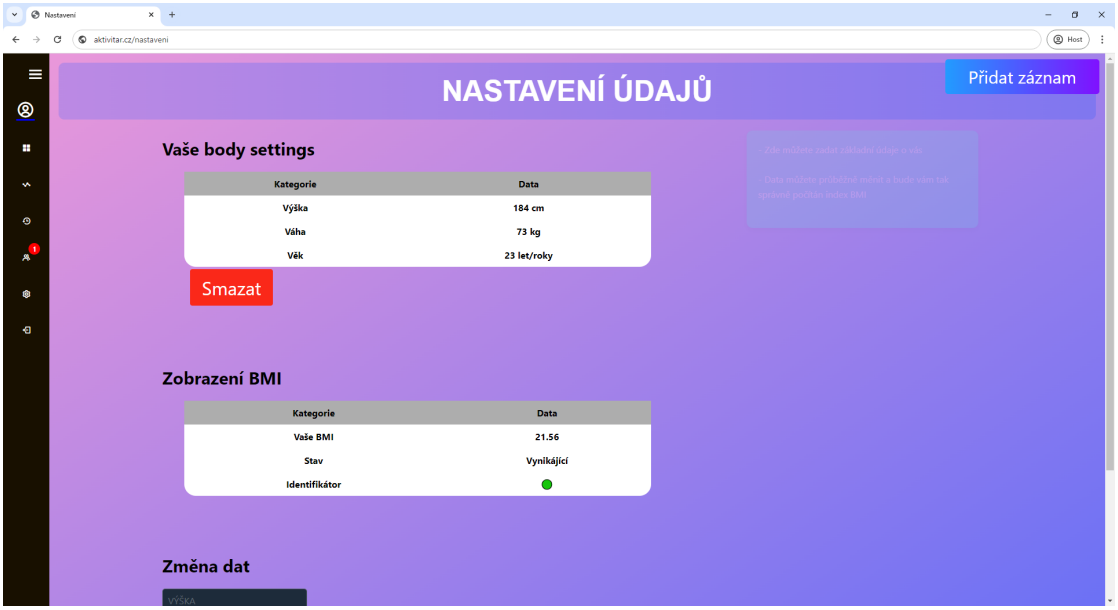
Obrázek 10: Scéna friend profile

Šest posledních aktivit řešíme podobným způsobem, jako v přehledu, ale v tomto případě používáme funkci *„getFriendsLastSix(friendUserName)"* s parametrem *„friendUserName"*, díky kterému najdeme v databázi posledních šest aktivit konkrétního uživatele (přítele).

Poslední, co v profilu přítele můžete najít je graf jeho poslední přidané aktivity.

10 Nastavení

V sekci nastavení si uživatel může nastavit výšku, váhu a věk a poté se mu vypočítá jeho BMI a zobrazí se do přehledné tabulky.



Obrázek 11: Scéna nastavení

„Body Mass Index neboli index tělesné hmotnosti (BMI) vyjadřuje vztah mezi tělesnou hmotností a tělesnou výškou. Normální hodnota BMI se pohybuje v rozmezí 18,5–25. Hodnoty pod 18,5 signalizují podváhu či podvýživu, hodnoty nad 25 signalizují nadváhu, nad 30 obezitu.“.[23] BMI se dále uživateli zobrazuje v přehledu. [24]

$$\mathbf{BMI} = \frac{\mathbf{hmotnost[kg]}}{(\mathbf{výška[m]})^2}$$

[23] https://www.wikiskripta.eu/w/BMI

[24] Vzorec pro výpočet BMI: https://www.wikiskripta.eu/w/BMI

11 Závěr

Na závěr bychom rádi shrnuli naši práci, na které jsme tento školní rok pracovali. Projekt jako takový definitivně splnil zadání být stránkou, kde uživatelé mohou sledovat a zaznamenávat svou fyzickou aktivitu během dne. Při vývoji se vyskytlo mnoho problémů ale také mnoho úspěchů, které nás rozhodně udělaly o něco chytřejšími a vzdělanějšími v oboru vytváření *frontend* i *backend* částí webových stránek. Jako členové týmu se shodujeme v tom, že nás tento projekt obohatil o velké množství důležitých informací, které se zaručeně budou v pozdější programovací kariéře hodit.

Myslíme si, že jsme velmi dobře zvládli komunikaci a spolupráci v týmu. Všichni účastníci se na práci podíleli a tento ročníkový projekt nás obohatil o spoustu zkušeností. Naučili jsme se vzájemné spolupráce, i přes momenty, kdy se naše názory lišily.

Pokud bychom měli projekt nějak zhodnotit, jednalo by se určitě o hodnocení velmi kladné. Díky nově nabitým zkušenostem s NodeJs a Express.js a osvěžením práce s HTML a CSS. Nakonec bychom tedy rádi zmínili, že jsme s prací ve výsledku velmi spokojeni.

12 Speciální citace

Web Dev Simplified

https://www.youtube.com/c/webdevsimplified

Z tohoto kanálu jsme čerpali základní informace pro NodeJS. Velmi nám pomohl porozumět a pochopit základy NodeJS.

Seznam obrázků

Seznam ukázek kódu