

**STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ**

**SOCIÁLNA SIEŤ**

**Adam Hladík**

2020

**SOCIÁLNA SIEŤ**  
**OBHAJOBA VLASTNÉHO PROJEKTU**  
**Adam Hladík**

Stredná priemyselná škola elektrotechnická, Hálova 16, Bratislava

Študijný odbor: 2675 M Elektrotechnika

Konzultant: Dávid Urban

Dátum odovzdania: 9. apríla 2020

Bratislava 2020

# Obsah

<b>Obsah .....</b>	<b>3</b>
<b>Úvod .....</b>	<b>5</b>
<b>1 Použité nástroje pre tvorbu tejto webovej aplikácie.....</b>	<b>6</b>
1.1 Programovacie jazyky & Frameworky.....	6
1.1.1 Značkovací jazyk HTML .....	6
1.1.2 Štýlovací jazyk CSS.....	8
1.1.3 Skriptovací jazyk JS.....	9
1.1.4 JavaScript framework Vue.....	10
1.1.5 Skriptovací jazyk PHP .....	12
1.1.6 Jazyk SQL.....	13
1.2 Softvér .....	15
1.2.1 Microsoft Visual Studio Code .....	15
1.2.2 XAMPP.....	16
1.2.3 Microsoft Edge.....	18
1.2.4 Webová aplikácia phpMyAdmin .....	19
<b>2 Všeobecný opis postupu tvorby sociálnej siete .....</b>	<b>21</b>
2.1 Dizajn .....	21
2.2 Architektúra.....	23
2.2.1 Frontend .....	23
2.2.2 Backend.....	27
<b>3 Detailný opis postupu tvorby sociálnej siete .....</b>	<b>29</b>
3.1 Kostra webu.....	29
3.1.1 Frontend - komponenty.....	30
3.1.2 Frontend – dizajn .....	48
3.1.3 Backend.....	53
<b>4 Zadania.....</b>	<b>59</b>
4.1 Demonštrácia pojmov Web a Internet .....	59
4.2 Popis vývoja webu.....	59
4.3 Všeobecné nariadenie o ochrane údajov (GDPR) .....	60
<b>5 Záver.....</b>	<b>62</b>
<b>6 Zoznam použitej literatúry.....</b>	<b>63</b>

# Úvod

Zadanie Sociálna Sieť som si vybral pretože som si chcel rozšíriť svoje vedomosti v programovacích jazykoch JavaScript, PHP, MySQL a v JavaScript-ových frameworkoch ako je Vue. Celá webová aplikácia je rozdelená na 2 časti.

Prvou časťou je tzv. *frontend* v ktorom je naprogramované celé užívateľské rozhranie a takisto aj logika webu ktorá prebieha na strane užívateľa/klienta. Na *frontende* sú použité jazyky HTML, CSS a vo veľmi veľkej miere JavaScript a to preto, pretože celý *frontend* je spravený vo frameworku Vue. Vue som si vybral kvôli tomu že mi umožňuje si vytvárať z viacerých HTML elementov vytvárať komponenty s vlastnou logikou, a následne ich vkladať do aplikácie tak aby dokázali medzi sebou fungovať a komunikovať.

Druhou časťou je tzv. *backend* v ktorom je naprogramovaná celá logika toho ako má server spracovávať požiadavky od klienta a ako má komunikovať s databázou. Na *backende* sú použité jazyky PHP a z časti MySQL. Jazyk PHP je mostom pre údaje v databáze a užívateľom. Stará sa o to aby keď mu z *frontendu* príde požiadavka aby mu dal údaje z databázy tak PHP túto požiadavku spracuje a pošle požiadavku databáze a výsledky z databázy pošle naspäť užívateľovi.

Na testovanie tejto webovej stránky som používal serverový program XAMPP ktorý mi umožňuje spúšťať PHP súbory, na úpravu zdrojového kódu som používal editor Visual Studio Code od firmy Microsoft a na manuálnu prácu s databázou som použil webový klient phpmyadmin ktorý je poskytovaný hostinovou službou od websupportu ktorú mám zakúpenú aj s doménou na ktorej bude fungovať táto webová aplikácia.

Stránka čo sa týka funkcií ponúka registráciu, prihlásenie, možnosť pridávania príspevkov či už textových alebo aj s fotografiami, označovať príspevky ako obľúbené, mazať príspevky a sekciu s nastaveniami ako zmena hesla, zmena profilovej fotografie a zmena popisu profilu.

Vyhlasujem, že som túto komplexnú odbornú prácu napísal samostatne pod odborným vedením konzultanta práce a použil iba uvedenú literatúru.

# 1 Použité nástroje pre tvorbu tejto webovej aplikácie

## 1.1 Programovacie jazyky & Frameworky

### 1.1.1 Značkovací jazyk HTML

#### 1.1.1.1 Úvod

HTML (Hypertext Markup Language) je značkovací jazyk ktorý je považovaný za štandard keď ide o vytváranie webových stránok a webových aplikácií. V dnešnej dobe sa nepoužíva ako jediný jazyk pre tvorbu webstránok a webových aplikácií, kombinuje sa aj s jazykmi ako CSS a JavaScript. Dôvod prečo je to značkovací jazyk a nie programovací je ten, že programovací jazyk posielá inštrukcie stroju na ktorom beží aby vykonal nejakú úlohu zatiaľ čo HTML len označuje časti textu do tzv. *tagov* (elementov) aby prehliadač ktorý zobrazuje daný súbor vedel ako má zobrazovať rôzne časti dokumentu.

```
<b>Tučný text</b>  
<i>Šikmý text</i>  
<u>Podčiarknutý text</u>
```

Kód č.1

Výstup tohto kódu by mal v prehliadači vyzerat' takto:

**Tučný text**

*Šikmý text*

Podčiarknutý text

Na to aby sme mohli tvoriť stránky pomocou jazyka HTML potrebujeme vytvoriť súbor s príponou .html alebo .htm. Pre zobrazenie tejto stránky nepotrebujeme dokonca ani webový server, stačí jedine prehliadač ktorý dokáže tento súbor zobrazit' lokálne. To že tento jazyk nieje programovací ale značkovací však neznamená že nemá pravidlá ohľadom toho ako sa má písať. Takto vyzerá základná kostra webstránky.

```
<!DOCTYPE html>1  
<html>2  
  <head>3  
    <!-- Metadáta o stránke -->4  
    <title>5Webstránka 1.0</title>8  
  </head>7  
  <body>6  
    <!-- Obsah ktorý vidí užívateľ -->
```

```
<p>7Ja som paragraf</p>8  
</body>8  
</html>8
```

#### 1.1.1.2 Vysvetlivky ku kódu č.2:

1. Riadok hovorí prehliadaču že so súborom má pracovať ako s HTML dokumentom.
2. HTML tag obsahuje všetko čo obsahuje celá stránka či už ide o viditeľný obsah alebo nie.
3. HEAD tag, alebo inak nazývaná aj hlavička webstránky obsahuje metadáta a rôzne konfiguračné tagy ktoré nesúvisia s obsahom webu ako takým ale skôr veci ako nadpis ktorý sa zobrazuje na karte v prehliadači, typ kódovania na stránke alebo to či je responzívna alebo nie.
4. Komentár ktorý sa môže písať do kódu HTML a bude ignorovaný pri vykresľovaní elementov.
5. TITLE tag obsahuje vyššie spomínaný nadpis web stránky ktorý sa zobrazuje v karte prehliadača.
6. BODY tag (telo) na druhej strane obsahuje všetko čo je viditeľné na stránke ako napr. rôzne texty, obrázky alebo formuláre.
7. P tag slúži ako paragraf ktorý prehliadač odsadzuje o niekoľko pixelov z každej strany.
8. Toto nieje referencia na konkrétny tag ale na tag ako taký že každý tag ako začína tak aj končí. Tento typ tagu sa nazýva párový pretože sa vždy dáva do páru aj s končiacim tagom. Sú aj výnimky kde sa používajú iba párové tagy bez nutnosti vkladania ukončovacieho tagu.

#### 1.1.1.3 História

Prvú definíciu jazyka HTML vytvoil v roku 1991 Tim Berners-Lee ako súčasť projektu WWW, ktorý mal umožniť vedcom zaoberajúcich sa fyzikou zdieľať svoje výsledky ohľadom ich výskumu po celom svete. Nieje teda náhodou že sa na tomto projekte pracovalo v CERNe (Európske centrum jadrového výskumu), ktoré sa nachádza neďaleko Ženevy. Jednalo sa o výsledok dvojročného projektu, ktorý mal vyriešiť problémy so zdieľaním informácií vo veľkej inštitúcii akoou je napríklad aj samotný CERN. Táto verzia HTML bola popísaná v dokumente HTML Tags.

Umožňovala text rozčleniť na niekoľko logických úrovní, použiť niekoľko druhov zvýraznenia textu a zaradiť do textu odkazy a obrázky.

Berners-Lee pri návrhu HTML nepredpokladal, že by autori webových stránok museli tento jazyk poznať. Prvá verzia WWW softvéru bola napísaná pre operačný systém NextStep a obsahovala prehliadač a integrovaný editor webových stránok. Softvér bol takisto vyvinutý v CERNe. V roku 1991 bol tento jazyk bol zverejnený širšej verejnosti.

### 1.1.2 Štýlovací jazyk CSS

#### 1.1.2.1 Úvod

Jazyk CSS (Cascading style sheets) je jazyk ktorý určuje to ako sa má vizuálne reprezentovať dokument napísaný v značkovacom jazyku akým je napríklad jazyk HTML. Skladá sa z tzv. *selectorov* a vlastností pre tieto selectory.

```
body1 {2
    background-color3:4 red5;6;
}7

#mojeid8 {
    color: #F1F1F1;
    font-size: 30px;
    padding: 10px 20px;
}

.mojaclassa9 {
    width: 300px;
    height: 200px;
    margin: 2px 4px 6px 8px;
}

button:hover10 {
    text-decoration: underline;
}
```

Kód č.3

#### 1.1.2.2 Vysvetlivky ku kódu č.3:

1. Názov selectoru podľa ktorého ma CSS nájsť daný element a naštýlovať ho podľa pravidiel ktoré si my určíme.
2. 7. Množinové zátvorky medzi ktoré už idú konkrétne vlastnosti a ich konkrétne hodnoty pre tieto vlastnosti.
3. Názov vlastnosti ktorú chceme modifikovať.
4. Symbol ktorý oddeľuje názov vlastnosti a hodnotu ktorú priradzujeme.

5. Hodnota adekvátne pre vlastnosť ktorú chceme upraviť, v tomto prípade je hodnota `red` čo je červená farba a vlastnosť je `background-color` čo je farba pozadia pre body element. Týmto riadkom sme teda nastavili červené pozadie pre element ktorý zodpovedá selectoru ktorý sme si zadefinovali o riadok vyššie.
6. Na koniec každej takejto modifikácie treba napísať bodkočiarku.
8. Tento selector sa zameriava iba na elementy ktoré majú atribút `id` ktorého hodnota je `mojeid`
9. Tento selector sa zameriava iba na elementy ktoré majú atribút `class` ktorého hodnota je `mojaclassa`.
10. Toto je tzv. pseudoselector ktorý sa zameria na selector pred dvojbodkou a použije sa iba vtedy keď pre prvý selector nastane nejaká udalosť, v tomto prípade je to udalosť `hover` takže čo sa stane je že element `button` (tlačidlo) bude mať podčiarknutý text pokiaľ nad ním prejdeme kurzorom.

Jazyk CSS je jednoduchý na pochopenie ale veľmi zložitý na zvládnutie. Pri väčších projektoch, sa programátor stretáva s kľudne aj 100 selectormi ktoré musí vedieť dobre naštýlovať nie len z hľadiska dizajnu a estetiky ale aj funkčnosti. Je takisto náročné na zvládnutie vedieť kombinovať vlastnosti aj ich hodnoty tak aby dobre fungovali. V jazyku CSS ich približne 200 takýchto vlastností.

#### 1.1.2.3 História

Prvá verzia CSS (CSS level 1) vznikla už v roku 1996 a umožňovala prácu s písmami, okrajmi a farbami. Bola navrhnutá Håkon Wium Lie-vom ktorý v tej dobe tiež spolupracoval s Tim Berners-Lee -om. V roku 1998 bola doplnená o nové možnosti a vznikol CSS level 2. V súčasnosti je podporovaná vo všetkých novších verziách prehliadačov (Internet Explorer, Opera, Mozilla, Netscape, Safari). Aktuálna verzia je CSS level 3.

### 1.1.3 Skriptovací jazyk JS

#### 1.1.3.1 Úvod

JavaScript je skriptovací jazyk pôvodne vytvorený Brendanom Eichom. Zámer tohto jazyka bolo vytvoriť jazyk ktorý sa bude používať pri tvorbe webových stránok, no dnes sa používa tento jazyk vo veľmi rôznorodých oblastiach ako IoT, hernom priemysle a pri tvorbe serverových aplikácií. Dôvod prečo sa názov tohto jazyka podobá



názvu jazyka Java je ten že zdieľajú podobnú syntax. Syntax jazyk JavaScript vyzerá takto:

```
var1 cislo2 =3 594;5  
console6.log7(cislo8);
```

#### 1.1.3.2 Vysvetlivky ku kódu č.4:

1. Kľúčové slovo pre inicializáciu premennej. Narozdiel od jazykov ako je C alebo Java v JavaScripte netreba určovať typ danej premennej, určuje sa totižto za behu a určuje sa podľa toho aký typ hodnoty sa v premennej nachádza.
2. Názov premennej ktorú môžeme použiť v celom skripte.
3. Operátor priradenia ktorý hodnotu z pravej strany priradí premennej na ľavej strane.
4. Hodnota ktorá sa priradí k premennej, v tomto prípade je to číslo.
5. Symbol pre ukončenie príkazu avšak redundantný, skript by fungoval aj kebyže tam bodkočiarku nenapíšeme pretože JavaScript si ju vie podľa kontextu doplniť sám.
6. Objekt ktorý referuje na konzolu v prostredí v ktorom beží skript. Objekt v JavaScripte je dátová štruktúra ktorá môže obsahovať viac premenných a aj metód.
7. Názov metódy ku ktorej chceme pristúpiť a ktorú chceme použiť. Funkcia **log** nám vypíše do konzoly parameter ktorý jej zadáme.
8. Argument funkcie.

#### 1.1.4 JavaScript framework Vue

##### 1.1.4.1 Úvod

Vue (vyslovuje sa /vju:/) je knižnica napísaná v jazyku JavaScript vďaka ktorej sa dajú vytvárať užívateľské rozhrania. Nemožno ju však prirovnať k oveľa známejšej knižnici ako je **jQuery**, pretože má celkom iný účel. Narozdiel od jQuery knižnice ktorá uľahčuje manipuláciu s **DOM** objektom, Vue uľahčuje skôr tvorbu architektúry pre dané užívateľské rozhranie/aplikáciu ako takú. Výhodou Vue je hlavne tzv. **reaktivita** a možnosť rozčlenenia si HTML blokov do samostatných komponentov s vlastnou logikou.

Tu sú znázornené fragmenty zdrojového kódu pre jednoduchú Vue aplikáciu:

HTML

```
<div id="app">
  {{ pozdrav }}
</div>
```

JavaScript

```
var app = new Vue({
  el: '#app',
  data: {
    pozdrav: 'Vitaj!'
  }
})
```

#### 1.1.4.2 Vysvetlivky ku kódom

V kóde č.5 máme časť HTML kódu v ktorom máme div element ktorý má **id app** a vnútri je zapísaná premenná v tzv. **šablónovej syntaxi**. V našom HTML kóde sme si teda nastavili ako sa majú reprezentovať údaje z našej Vue aplikácie ktorú sme si zdefinovali v kóde č.6. Na to aby naša aplikácia fungovala si do premennej **app** priradíme Vue objekt s objektom ako parametrom. Tento objekt obsahuje premennú **el** ktorá hovorí Vue kde v HTML kóde má fungovať. Ďalej máme objekt **data** ktorý už obsahuje naše premenné s ktorými chceme pracovať.

V tomto príklade máme premennú **pozdrav** ktorá obsahuje text „Vitaj!“. Keď si skúsime otvoriť našu stránku tak namiesto toho aby nám prehliadač vypísal **{{ pozdrav }}** nám vypíše aktuálnu hodnotu tejto premennej. Čo je však najlepšie, tak keby sme chceli zmeniť hodnotu tejto premennej niekde inde v kóde, napr. príkazom **app.pozdrav = „Vitaj znova!“**; tak bez toho aby sme museli manuálne meniť v **div** elemente napr. takýmto príkazom **document.getElementById(„app“).innerHTML = „<naša správa>“** to spraví Vue za nás automaticky.

Ide totiž o to že aj stránka v HTML a aj aplikácia vo Vue sú teraz previazané a fungujú ako jeden celok spolu.

#### 1.1.4.3 História

Knižnicu Vue vytvoril Evan You v roku 2013 a dodnes na nej pracuje aj s komunitou ľudí z celého sveta. Jeho idea bola použiť niektoré koncepty z už

existujúceho frameworku **Angular** a vytvoriť niečo jednoduché na používanie ale mocné na fungovanie. Takisto chcel aj rozšíriť funkcionality samotného tvorenia stránok pomocou HTML a JavaScriptu.

### 1.1.5 Skriptovací jazyk PHP

#### 1.1.5.1 Úvod

PHP je skriptovací jazyk, ktorý je špeciálne navrhnutý na tvorbu web stránok bežiacich na webovom serveri. Všetok PHP kód je vykonávaný pomocou PHP runtime, aby dynamicky vytvoril obsah na webovej stránke. Taktiež môže byť využitý na skriptovanie z príkazového riadku alebo klientovo orientované aplikácie s grafickým rozhraním. PHP môže byť nasadené na väčšine Webových serverov, operačných systémov a platformách a môže sa používať v spojení s mnohými relačnými databázami. PHP bolo inšpirované jazykmi podporujúcimi procedurálne programovanie. Najviac vlastností prebralo od jazyka C a jazyka Perl. V neskorších verziách bolo rozšírené o možnosť používať objekty.

Na to aby sa dal jazyk PHP používať ako jazyk na tvorbu web stránok, potrebujeme webový server ktorý nám umožní spracovať PHP súbory a naservírovať ich ako webové stránky. Syntax jazyka PHP vyzerá takto:

```
<?php1
    $meno2 = „Martin“;
    $vek = 30;

    echo3 „<h1>Ahoj $meno tvoj vek je $vek rokov.</h1>“;4
?>5
```

#### 1.1.5.2 Vysvetlivky ku kódu

1. Takto sa definuje začiatok PHP skriptu.
2. Premenné v PHP sa definujú tak že pred názov premennej treba napísať znak doláru (\$). Takisto ako aj v jazyku JavaScript, v PHP netreba písať typy premenných keďže PHP vie počas behu skriptu zistiť typ podľa hodnoty ktorá sa v premennej nachádza.
3. Príkaz echo slúži ako štandardný výstup. Ako je vidieť tak v tomto prípade nebolo treba použiť žiadny konkatenančný operátor – premenné sme mohli rovno napísať do úvodzoviek a PHP si už dosadí hodnotu za týmito premennými.
4. Bodkočiarku treba písať na koniec každého príkazu v PHP.

## 5. Takto sa definuje koniec PHP skriptu

Keď si tento skript otvoríme už na bežiacom webovom servery tak výstup v prehliadači bude `<h1>Ahoj Martin tvoj vek je 30 rokov.</h1>` čoho výsledkom bude veľký text na obrazovke keďže tag **h1** symbolizuje veľký nadpis.

### 1.1.5.3 Použitie

Ako je spomenuté vyššie tak na to aby sme mohli jazyk PHP použiť ako jazyk s ktorým sa dajú tvoriť dynamické stránky, potrebujeme webový server s podporou PHP. Pod pojmom „webový server“ sa myslí program ktorý vie sprítupniť náš PHP súbor cez prehliadač.

Hoci sa dá tento program stiahnuť samostatne tak vo veľkej väčšine prípadov sa nachádza už v existujúcich balíčkoch ktoré obsahujú aj programy pre podporu databáz a samotnú podporu jazyka PHP alebo aj jazyka Perl. Jedným z takýchto balíčkov je aj balíček XAMPP (Cross-platform Apache + MariaDB + PHP + Perl).

### 1.1.5.4 História

PHP bolo pôvodne navrhnuté ako niekoľko skriptov v jazyku Perl, neskôr prepísaných do jazyka C. Autorom bol Rasmus Lerdorf v roku 1994. O rok neskôr svoje skripty zverejnil pod názvom "Personal Home Page Tools".

## 1.1.6 Jazyk SQL

### 1.1.6.1 Úvod

SQL je základný nástroj pre manipuláciu, správu a organizovanie dát uložených v databázových systémoch. Je určený predovšetkým pre užívateľov, ale v mnohých smeroch ho využívajú aj tvorcovia aplikácií. Je adaptabilný pre akékoľvek prostredie. SQL nie je len dotazovací jazyk. S jeho pomocou môžeme definovať, vyplňať stĺpce tabuľky dátami a definovať vzťahy a organizáciu medzi položkami dát. Taktiež umožňuje spoločné využívanie dát a zabezpečuje hladký priebeh činností v prípade, že k dátam pristupujú viacerí užívatelia.

Na to aby sa jazyk SQL mohol dať použiť sú potrebné nejaké dáta v nejakej dátovej schéme. Tieto dáta sa dajú predstaviť ako bunky v tabuľke ktorá má stanovené stĺpce.

Pre vytvorenie takejto dátovej schémy v podobe tabuľky si musíme najprv určiť názov každého stĺpca a typ hodnôt ktorý sa bude nachádzať v stĺpcoch.

Povedzme že chceme spraviť databázu v ktorej sa bude nachádzať tabuľka s našimi užívateľmi. Tabuľka užívateľov by mala obsahovať stĺpce pre unikátny identifikátor (ID), užívateľské meno, heslo, popis a cestu k profilovej fotke užívateľa.

```
CREATE TABLE users (  
    ID INT PRIMARY KEY AUTO_INCREMENT,  
    firstname VARCHAR(255) NOT NULL,  
    lastname VARCHAR(255) NOT NULL,  
    password VARCHAR(255) NOT NULL,  
    picture VARCHAR(255)  
);
```

Keďže je syntax tohto jazyka celkom intuitívna a veľmi sa približuje ľudskej reči tak len načrtnem zhruba čo ktoré slovo znamená.

\*Slová ktoré su podfarbené modrou farbou sú rezervované slová jazyka SQL. Slová podfarebné čiernou sú už slová ktoré si podľa potreby dopĺňa užívateľ.

**CREATE TABLE** <názov>() – Vytvára tabuľku s názvom ktorý jej priradíme a s parametrami do ktorých špecifikujeme stĺpce ktoré tabuľka bude obsahovať.

**INT** – Určuje že hodnoty v stĺpci ID budú dátového typu Integer čo je celé číslo.

**NOT NULL** – Hovorí že daná bunka nesmie byť nikdy prázdna pri práci s tabuľkou. Ak bude, tak zápis/prepísanie s touto prázdnu hodnotou v tabuľke bude neúspešne.

**PRIMARY KEY** – Určuje pre daný stĺpec že sa jedná o unikátny identifikátor pre každý jeden stĺpec. Tieto identifikátory sú užitočné – priam potrebné pri manipulácii s dátami v tabuľke. Príklad, povedzme že máme tabuľku užívateľov kde sa nachádza 100 000 užívateľov a nastane prípad kde dvaja užívatelia majú to isté meno a priezvisko. Ak by nemali údaj ktorý by ich odlišoval, nerátajúc heslo a fotku, tak by sa

nedalo rozlíšiť ktorý užívateľ je ktorý. Primary key je údaj ktorý nemusíme my zadávať do tabuľky, tabuľka si ho automaticky dopĺňa pre každý jeden nový zápis

**AUTO\_INCREMENT** – Hovorí tabuľke že daný stĺpec, v našom prípade ID, sa bude automaticky zvyšovať o hodnotu 1. Takže každý ďalší užívateľ bude mať ID číslo o jedna väčšia.

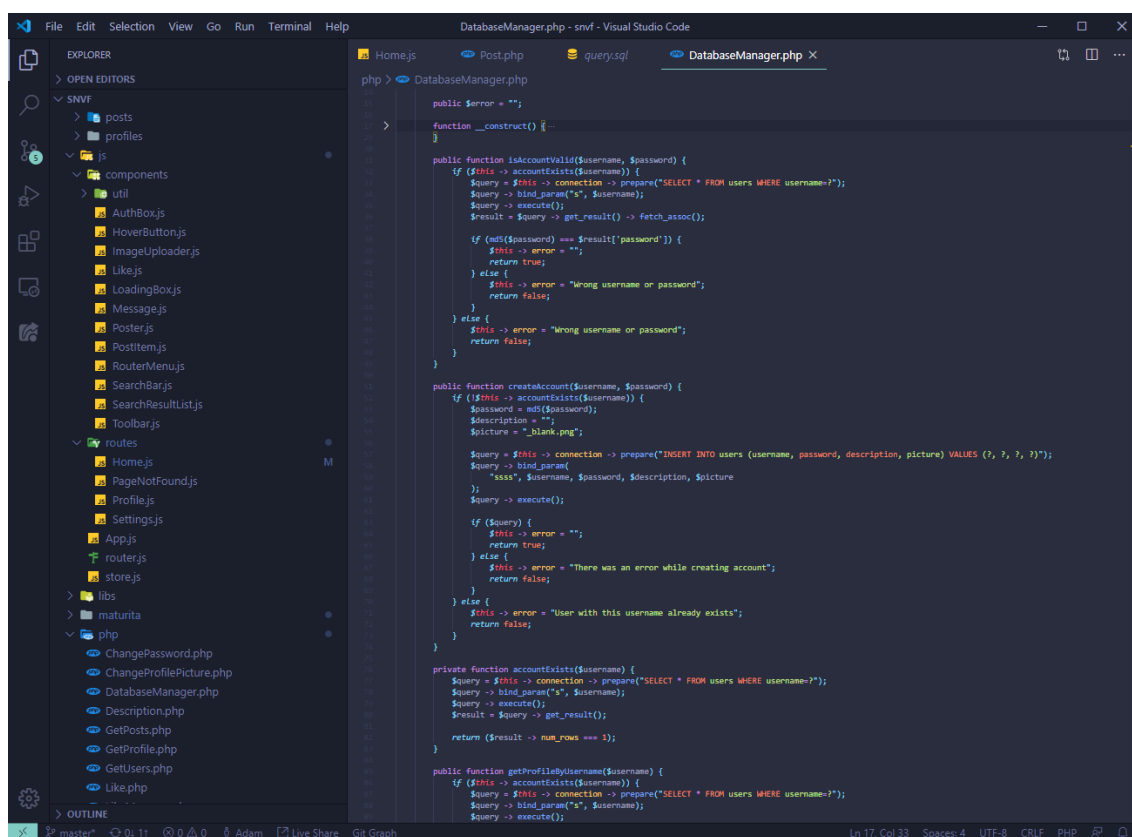
**VARCHAR(n)** – Varchar takisto ako aj INT hovorí tabuľke že hodnoty v danom stĺpci budú konkrétneho dátového typu, v tomto prípade to bude textový reťazec kde maximálna dĺžka reťazcu je n znakov. Takže ak by sme chceli do tohto stĺpca vložiť reťazec ktorého dĺžka je 256 tak tabuľka odmietne túto požiadavku a nevykoná ju.

## 1.2 Softvér

### 1.2.1 Microsoft Visual Studio Code

Visual studio code je open-source softvér pre písanie zdrojových kódov v rôznych programovacích jazykoch akými sú napríklad **HTML**, **JavaScript**, **Java** a **C++** - v podstate pre akýkoľvek programovací jazyk no nie pre každý môže existovať podpora rôznych funkcií ktoré tento editor používa. Medzi najviac používané funkcie patrí podpora pre ladenie programov, vnorená podpora pre verzovací systém **GIT**, podfarbovanie syntaxi pre veľkú škálu programovacích jazykov, inteligentné dopĺňanie kódu, možnosť vizuálnej úpravy programovacieho štúdia pomocou rôznych integrovaných tém, rozšírené klávesové skratky no hlavne možnosť dopĺňania rôznych rošírení od developerov tretích strán, vďaka ktorým sa dá produktivita posunúť na ďalší level.

Softvér bol vytvorený korporáciou **Microsoft** a je založený na frameworku **Electron** čo je framework ktorý umožňuje vytvárať desktopové programy pomocou webových technológií.

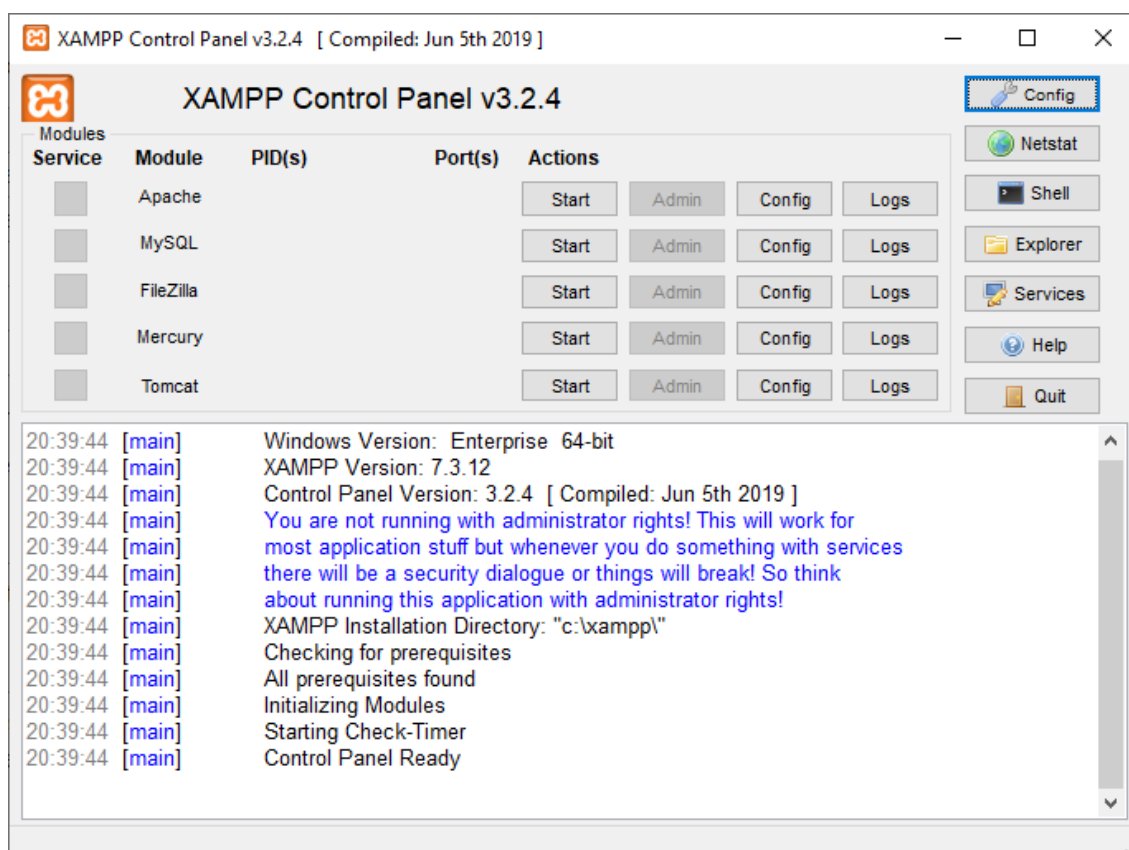


Obrázok 1 – Uživatelské rozhranie programu Visual Studio Code

Na **obrázku č. 1** je rozloženie a výzor programu Visual Studio Code. Na ľavej strane sa nachádza panel v ktorom sa listujú súbory ktoré sa nachádzajú v priečinku v ktorom momentálne pracujete a v ktorom sa nachádzajú vaše zdrojové kódy. Na pravej strane je už samotný editor kódu.

### 1.2.2 XAMPP

XAMPP Je voľne dostupný a open-source multiplatformový webový serverový balík vytvorený firmou Apache, pozostávajúci hlavne zo softvéru ako Apache HTTP Server, MariaDB databázou, FileZilla FTP klientom a interpretermi skriptovacích jazykov akými sú PHP a Perl. Výhodou tohto balíčku je to že je veľmi jednoduchý na používanie s už spomenutou výhodou že je multiplatformový takže program sa rovnako používa na všetkých známych OS.



Obrázok 2 – Uživatelské rozhranie programu XAMPP

Na **obrázku č. 2** sú znázornené funkcie programu XAMPP. Na vrchu sa nachádza zoznam služieb ktoré je možné spustiť na pozadí. Najčastejšie používanými službami sú Apache (Webový server) a MySQL (Databázový server). Po spustení webového servera sa nám na lokálnej sieti spustí na pozadí bežiaci server ktorý po načítaní prehliadača navracia súbory nachádzajúce sa v štandardne nastavenej ceste **C:/xampp/htdocs**. Na pravej strane sa nachádza panel ktorý obsahuje ďalšie funkcie ako nastavenie programu, prístup k terminálu, prístup k súborovému prieskumníku. A na záver, na spodku sa nachádza textové pole ktoré slúži ako výstup informácií o priebehu služieb ktoré program spúšťa.

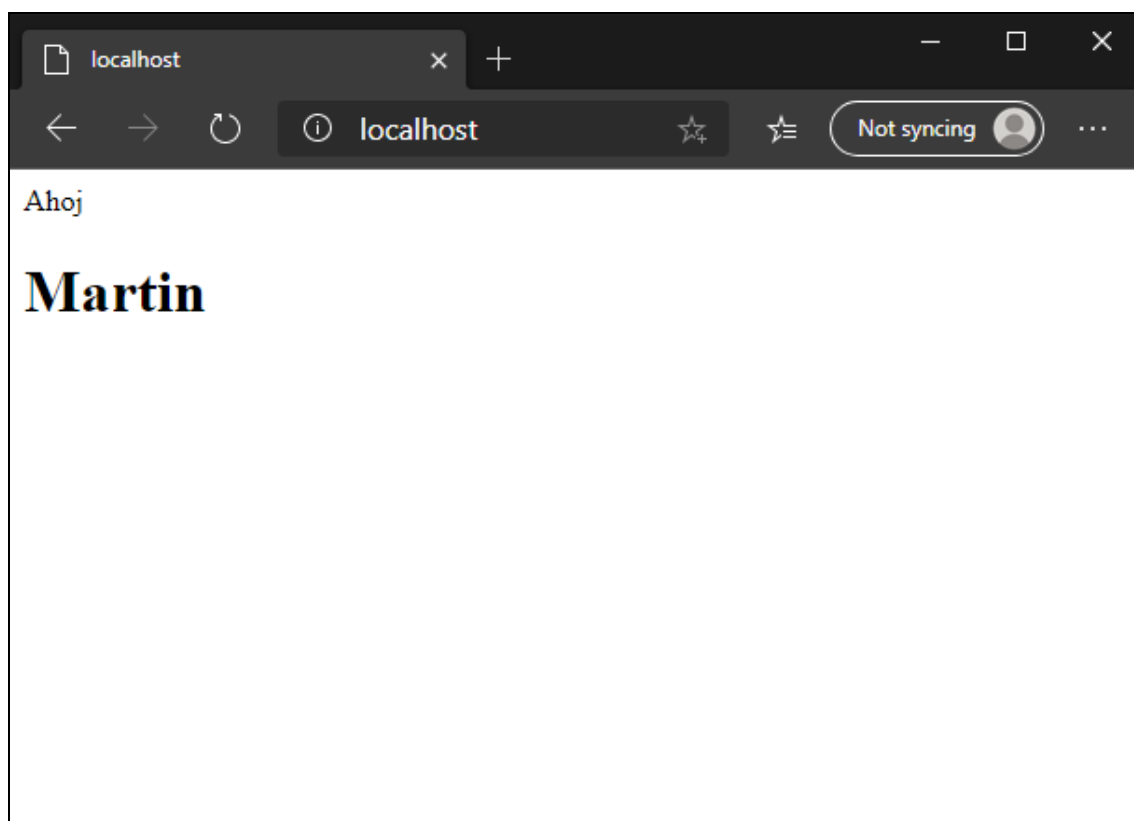
Po zapnutí servera sme následne schopný si pozrieť súbory (v našom prípade stránky) ktoré sa nachádzajú vo vyššej spomenutej ceste. Vďaka tomu že XAMPP podporuje jazyk PHP tak sme takisto schopný stránky písať v tomto jazyku a bez problémov ich môžeme prehliadať.

Povedzme že máme v tejto ceste súbor index.php s takýmto kódom:



```
<?php
    $meno = „Martin“;
    echo „<h1>Ahoj $martin</h1>“;
?>
```

Po uložení a za predpokladu že máme spustený webový server si tento súbor môžeme zobrazit', a to zadaním následovnej adresy **http://localhost/index.php** do webového prehliadača.



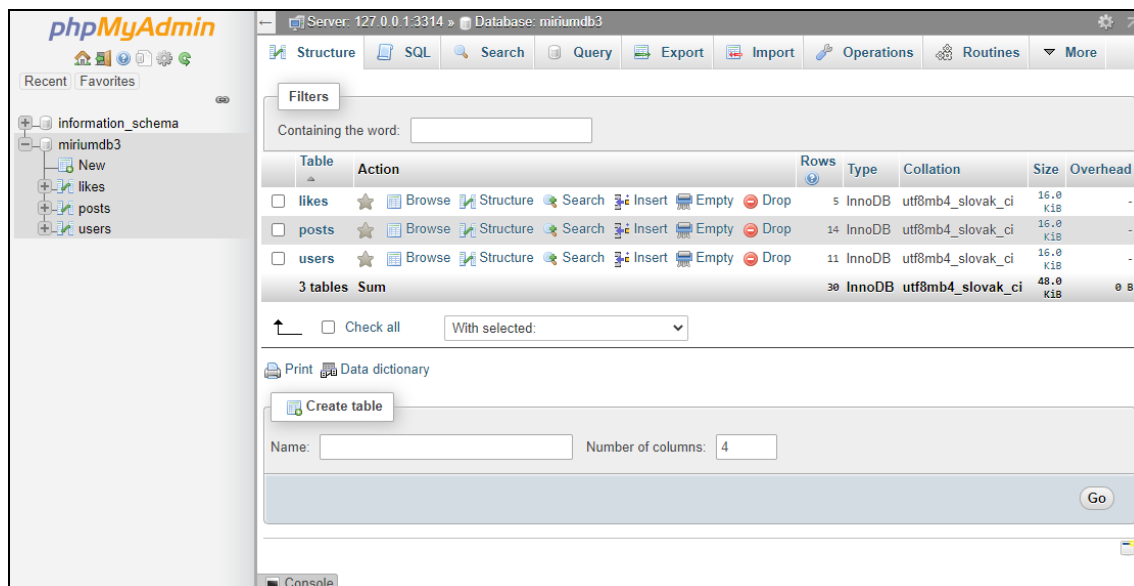
Obrázok 3 – Výstupný text zo súboru index.php zobrazený vo webovom prehliadači

### 1.2.3 Microsoft Edge

Microsoft Edge je webový prehliadač ktorý firma Microsoft založila na začiatku roka 2018. Prehliadač prvotne používal svoje vykreslovacie podprogramy (EdgeHTML) a rozhrania pre skriptovacie jazyky akým je napríklad JavaScript (ChakraEngine). Nanešťastie boli tieto nástroje pozadu oproti konkurencii (Google Chrome a Mozilla Firefox). Preto v roku 2019 vyšla nová verzia Microsoft Edge ktorá bola založená na tzv. *Chromium* jadre na ktorom je postavený prehliadač Google Chrome. Pre tento prípad sa Edge javí ako lepšia alternatíva Google Chrome pretože ponúka tú istú podporu pre webové stránky no popri tom je viac optimalizovaný a energeticky úspornejší.

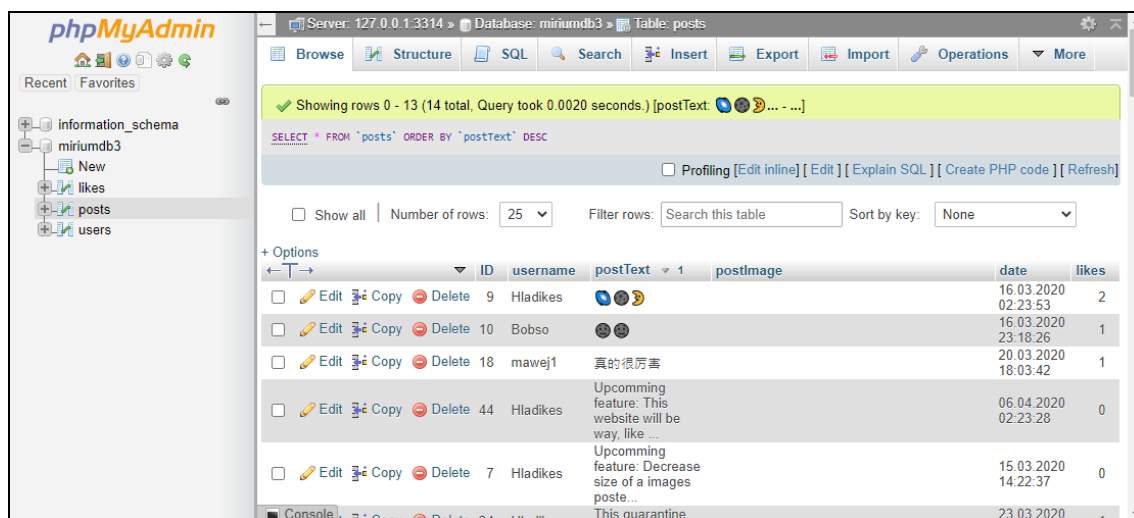
## 1.2.4 Webová aplikácia phpMyAdmin

PhpMyAdmin je voľne dostupný webový administračný nástroj pre správu MySQL databáz. S týmto nástrojom je možné priamo spúšťať SQL príkazy, pozeráť a upravovať tabuľky a samotné údaje v tabuľkách.



Obrázok 4 – Užívateľské rozhranie aplikácie phpMyAdmin

Na ľavej strane aplikácie sa nachádza panel ktorý obsahuje zoznam databáz a ich tabuliek ktoré tento databázový server obsahuje. V tomto prípade je miriumdb3 databáza ktorá obsahuje tabuľky likes, posts a users. V strede aplikácie sa nachádza detailný zoznam tabuliek ktoré databáza ponúka, a s ním aj rôzne možnosti akcií ktoré je možné vykonať.



Obrázok 5 – Sekcia s obsahom dát z konkrétnej tabuľky

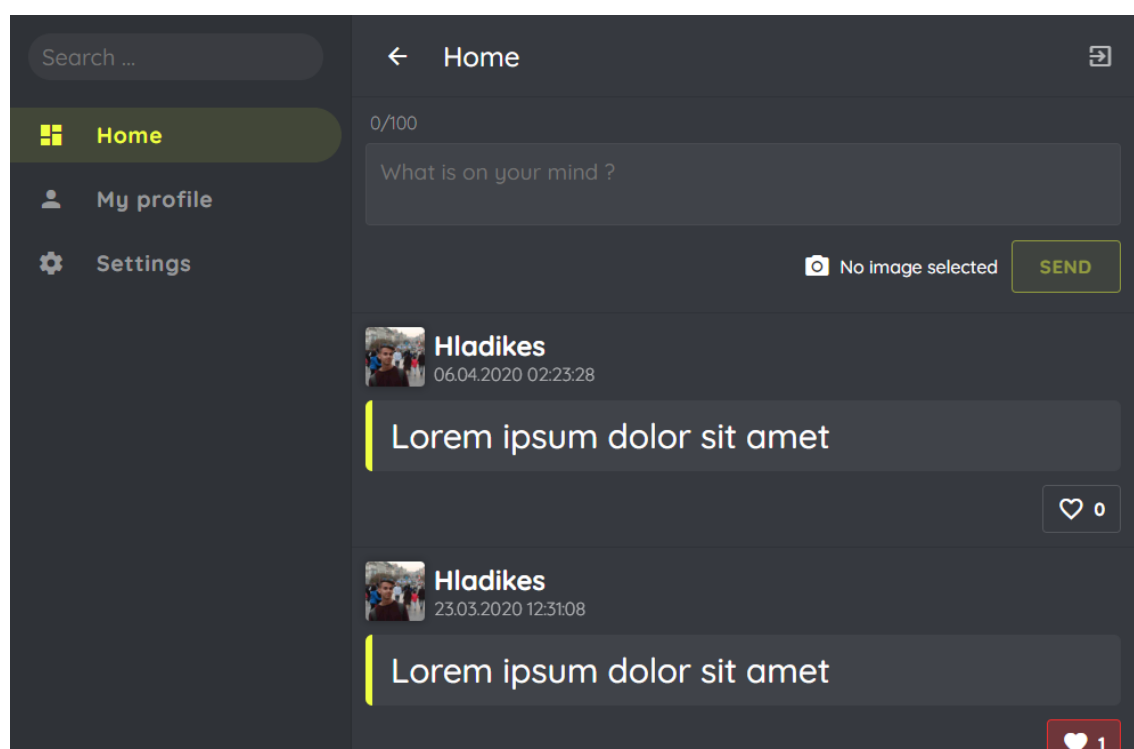
Na **obrázku č. 5** sa nachádza (okrem už vopred spomínaného panelu) zoznam údajov ktoré sa nachádzajú v tabuľke. V tomto prípade ide o tabuľku príspevkov kde v riadkoch sa nachádza pole údajov ktoré korešpondujú daným stĺpcom. Okrem toho sa v riadkoch zobrazujú akcie ktoré je možné vykonať s konkrétnou sadou údajov.

Pre túto maturitnú prácu som nemusel inštalovať túto aplikáciu pretože mi ju poskytuje k hostingu spoločnosť websupport.

## 2 Všeobecný opis postupu tvorby sociálnej siete

### 2.1 Dizajn

Ako prvé som musel mať predstavu o tom ako by táto sociálna sieť mala vyzerat'. Inšpiroval som sa webovým komunikačným portálom *Discord* ktorý obsahuje jednoduchý dizajn ktorý je intuitívny a podľa mňa dokonalý z hľadiska konzistencie – všetky vizuálne prvky totiž pôsobia na mňa správne či už ide o veľkosť, farbu alebo umiestnenie. Takisto som sa inšpiroval novým dizajnom ktorý priniesla sociálna sieť *Facebook*. Aj napriek tomu že som sa veľmi inšpiroval dizajnom týchto dvoch webov tak som sa snažil aby dizajn pôsobil originálne. Keď som mal predstavu tak som si spravil šablónu kde som kombinoval rôzne farebné kombinácie a pre túto stránku som vybral primárnu šedú farbu **#363c3f**, tmavý odtieň primárnej šedej farby **#212527** a tzv. *akcentovú farbu* čo je veľmi dôležitá farba ktorú zvýrazňuje prvky na stránke tak aby boli čo najlepšie odlišiteľné od ostatných – a to svetlo žltú **#FFFF42**.



Obrázok 6 – Dizajn webstránky

Ako je vidieť tak táto žltá farba upriamuje užívateľovu pozornosť na prvky ktoré sú najdôležitejšie a tými su navigačná sekcia, klikateľné položky alebo v tomto prípade príspevky od užívateľov, na ktoré stojí v podstate celá webstránka.

Čo sa týka konštrukcie samotného dizajnu tak som použil konštrukčné moduly v CSS ktorými sú *flexbox*. Tento modul umožňuje nastavovať veľkosť elementov na základe ich váhy a ich polohu na základe jednoduchých pravidiel bez toho aby bolo potrebné nastavovať a prepočítavať rozmery elementov.

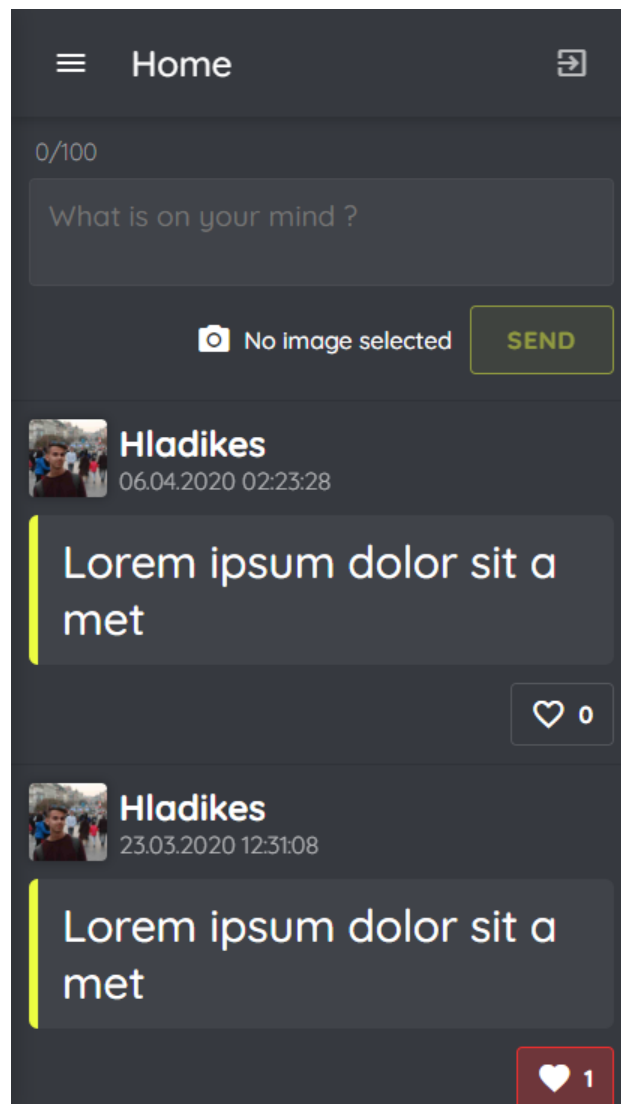
```
div {  
    display: flex;  
    flex-direction: row;  
    justify-content: center;  
    align-items: center;  
}
```

Vyššie je znázornený príklad toho ako jednoducho sa dá vytvoriť kontajner ktorého tzv. *child elements* sa budú zobrazovať vodorovne, a zarovnané na stred po vertikálnej a aj horizontálnej osi.

```
div:first-child {  
    flex: 1;  
}  
  
div:nth-child(2) {  
    flex: 3;  
}
```

Vyššie je znázornený príklad toho ako funguje nastavovanie veľkosti elementov nachádzajúcich sa už v nami definovanom div elemente ktorému sme nastavili flexbox vlastnosti. Kebyže si chceme spustiť tento kód tak vidíme div element v ktorom prvý element bude mať váhu 1 a druhý element váhu 3. Keď si spravíme súčet všetkých váh ktoré majú elementy tak si vieme zistiť akú veľkosť zaberá každý element. V tomto prípade je súčet 4 a prvý element bude tým pádom zaberáť  $\frac{1}{4}$  plochy tohto rodičovského elementu a druhý element bude zaberáť  $\frac{3}{4}$ . Môžeme si to predstaviť aj vo forme % kde prvý element zaberá 25% a druhý 75%.

Taktiež bol pri tvorbe do úvahy braný fakt, že stránku budú užívatelia používať aj na mobilných telefónoch a preto som pri tvorbe dizajnu programoval aj pravidlá pre zobrazovanie na menších obrazovkách



Obrázok 7 – Mobilné zobrazenie webstránky

## 2.2 Architektúra

### 2.2.1 Frontend

#### 2.2.1.1 Úvod

Pri tvorbe frontend-u tejto sociálnej siete som sa rozhodol že postavená na JavaScript frameworku Vue. Vue z toho dôvodu pretože je jednoduchý na implementáciu a, používanie no zároveň veľmi výkonný keď ide o rozsiahle projekty. Medzi hlavné funkcie ktoré poskytuje tento framework je reaktivita – automatické synchronizovanie premenných s užívateľským rozhraním, a možnosť rozdelenia stránky

na tzv. komponenty, čo umožňuje si naprogramovať samostatné moduly ktoré môžeme používať.

```
Vue.component('pocitadlo', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<button v-on:click="count++">Stlačil si ma {{ count }}x krát.</button>'
})
```

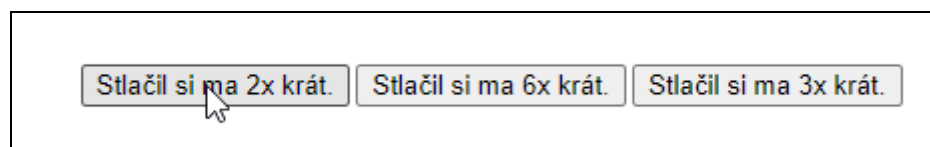
Vyššie je znázornený príklad toho ako pomocou Vue vieme vytvoriť jednoduchý komponent. Vue umožňuje tvorbu komponentu niekoľkými spôsobmi, jedným z nich je aj tento vyššie uvedený.

Vue poskytuje metódu **component** ktorá vyžaduje aby boli zadane dva argumenty. Prvým je názov komponentu a druhým je už samotný objekt ktorý obsahuje všetky premenné a metódy s ktorými daný komponent pracuje. V tomto prípade je to tlačidlo ktoré sa chová ako počítadlo. Obsahuje premennú **count** ktorá slúži pre uloženie stavu tohto tlačidla. Okrem nej sa v objekte nachádza **template** premenná ktorá slúži ako šablóna pre HTML kód. V nej sa nachádza tlačidlo ktoré obsahuje atribút **v-on:click** v ktorom je zadaná akcia ktorá sa vykoná keď užívateľ klikne na tlačidlo. V tomto prípade sa inkrementuje hodnota **count**. Text tohto tlačidla obsahuje text spolu s názvom premennej ktorý je zapísaný tzv. **šablónovou syntaxou**.

Potom ako funkciu zavoláme tak sme schopný používať tento komponent ako obyčajný HTML element.

```
<div id="demo">
  <pocitadlo></pocitadlo>
  <pocitadlo></pocitadlo>
  <pocitadlo></pocitadlo>
</div>
```

Potom ako si vytvoríme inštanciu Vue objektu s referenciou na element s id **demo** nám na stránke vznikne tlačidlo ktoré funguje ako počítadlo.



Obrázok 8 – Výstup z prehliadača

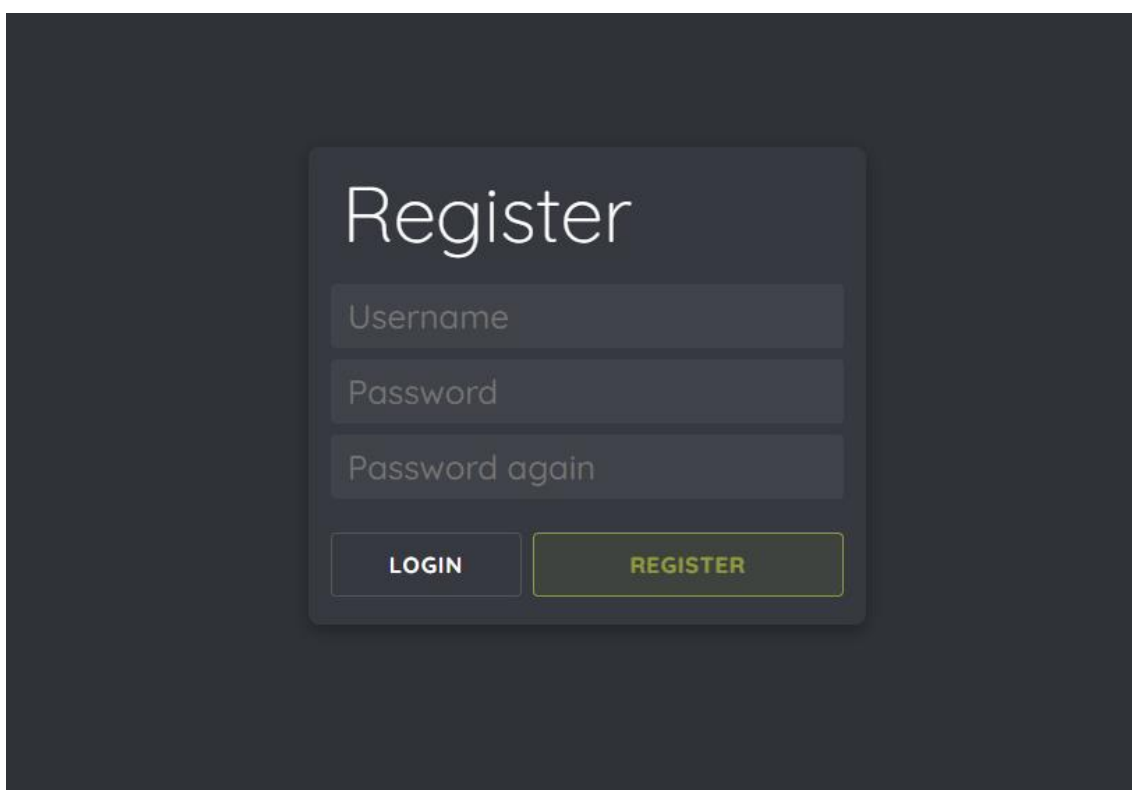
Ako je možné vidieť, tak na stránke vznikli tri samostatné tlačidlá ktoré po kliknutí zvýšia svoj stav o 1. Čo je veľmi veľkou výhodou je, že každý komponent má samostatne enkapsulované data a neovplyvňuje stav iných kópií tohto komponentu.

**Poznámka:** *Pre pochopenie celej architektúry bolo nevyhnutné vysvetlenie toho ako tieto komponenty fungujú a prečo sú take dôležité vzhľadom na fakt že celá stránka bola postavená z niekoľkých samostatne fungujúcich komponentov.*

#### 2.2.1.2 Postup

Na to aby som mohol začať s tvorbou užívateľského rozhrania som okrem dizajnu taktiež musel vedieť aké stránky a podstránky bude táto sociálna sieť obsahovať. Keďže som chcel aby bola stránka jednoduchá na používanie tak obsahuje iba 4 sekcie.

Prvou z nich je prihlasovací a registračný formulár ktorý slúži k autorizácii užívateľov.

A dark-themed user interface for a registration form. The form is centered and has a title 'Register' in a large, white, sans-serif font. Below the title are three input fields: 'Username', 'Password', and 'Password again', each with a light gray placeholder text. At the bottom of the form are two buttons: a 'LOGIN' button with white text on a dark background, and a 'REGISTER' button with green text on a dark background. The entire form is set against a dark gray background.

Obrázok 8 – Registračný formulár webstránky

Pre zachovanie jednoduchého používania sú vyžadované iba údaje ako prezývka a heslo pod ktorým sa užívateľ autorizuje pri vstupe na web.



Keďže webstránka potrebuje počas behu poznať meno prihláseného užívateľa a jeho autorizačný token, tak tieto údaje musia byť dočasne uložené a k dispozícii. Preto táto stránka obsahuje knižnicu **Vuex** ktorá je od tvorcov Vue. Je to knižnica ktorá je schopná ukladať a pracovať s dátami počas behu aplikácie a takisto byť schopná reaktivity. Tieto dáta sú zmazané pri zatvorení karty v prehliadači.

Čo sa týka rozhrania pre prihláseného užívateľa tak som sa opäť sústredil na to aby bolo intuitívne a jednoduché na používanie. Toto rozhranie ponúka lištu umiestnenú na vrchu ktorá obsahuje tlačidlo ktoré presmeruje užívateľa o krok späť, nadpis sekcie v ktorej sa aktuálne nachádza a tlačidlo pre odhlásenie. Ďalej sa naravo nachádza panel v ktorom je textové pole na vyhľadávanie iných užívateľov a hneď pod ním sa nachádza navigácia s ponukou rôznych ďalších sekcií, **viď obrázok 6.**

Jednou zaujímavosťou je fakt že celá webstránka obsahuje iba jeden index.html súbor a aj napriek tomu, že to robí dojem že existuje viacero separátnych podstránok, dokáže stránka meniť svoj obsah bez toho aby sa musela obnovovať a znova načítavať. Túto funkcionality do stránky prináša knižnica **Vue Router** ktorá takisto ako **Vuex** bola vytvorená tými istými vývojármi frameworku Vue. Táto knižnica mi umožnila si zadať URL podadresy a pridať im komponent ktorý sa zobrazí keď užívateľ túto adresu navštívi.

```
const routes = [
  {
    path: '/home',
    name: 'home',
    component: () => import('Home.js')
  }
]

const router = new VueRouter({
  routes,
})
```

Kód vyššie znázorňuje ako jednoducho sa tento proces dá nastaviť. V prvom rade je potrebné si vytvoriť pole objektov kde každý objekt reprezentuje už cieľovú cestu. Vlastnosť **path** definuje fyzickú trasu k stránke, **name** pomenúva túto trasu a **component** určuje aký komponent sa má zobrazíť keď sa navštívi daná adresa. Toto pole sa následne vloží do konštruktora objektu **VueRouter**.

## 2.2.2 Backend

### 2.2.2.1 Úvod

Ako každá stránka ktorá potrebuje dlhodobo ukladať dáta, tak aj pre tento prípad bolo nutné spraviť funkčný backend systém ktorý tieto dáta bude spracovávať, a následne ukladať. Pre tento postup som si vybral jazyk PHP a databázový systém MySQL keďže s jazykom PHP funguje veľmi dobre.

Na strane servera sa nachádzajú 3 typy zdrojových kódov napísaných v jazyku PHP. Prvým typom sú pomocné triedy ktoré zaobalujú rozsiahlu logiku do funkcií ktoré sú často využívané. Druhým typom sú súbory ktoré slúžia ako prostredník medzi klientom a serverom. Tieto súbory spracúvajú požiadavky klienta a následne ich vykonávajú pomocou posledného typu triedy, ktorá je priamym komunikantom s databázou.

### 2.2.2.2 Postup

Na to aby frontend mohol komunikovať so serverom je potrebné implementovať kód ktorý dokáže posielat' vyššie spomínané požiadavky. Frontend používa pre tento typ komunikácie knižnicu **Axios** ktorá je schopná jednoduchým zápisom posielat' tieto požiadavky.

Na odoslanie takej to požiadavky sú potrebné 3 parametre, a to typ požiadavky (POST, GET, ...), adresu kam bude požiadavka odoslaná a dáta ktoré budú s touto požiadavkou odoslané.

```
axios({
  url: „/php/Post.php“,
  method: „POST“,
  data: {
    username: „user1“,
    message: „Toto je moja správa!“
  }
}).then(...).catch(...)
```

Vyššie je znázornený kód ktorý odosiela takúto požiadavku na server a to konkrétne súboru Post.php, metódou POST a s uvedenými dátami.

Akonáhle príde požiadavka na server, a teda konkrétny súbor, je potrebné ju odchytiť a spracovať. Pre túto prácu som si spravil triedu RequestGuard ktorá kontroluje každú prijatú požiadavku na server, skontroluje či obsahuje všetky potrebné údaje pre

vykonanie konkrétnej akcie, a ak je všetko v poriadku tak dovoľuje ďalšiu prácu s jej údajmi.

```
RequestGuard::handle(  
    "POST", ["username", "token"], function($params) {  
        // TODO  
    }  
);
```

Vyššie je uvedený konkrétny príklad zápisu tohto procesu. O všetko sa stará funkcia **handle** ktorá vyžaduje 3 parametre. Prvým je metóda akou bola požiadavka odoslaná, pole v ktorom sa nachádzajú názvy vyžadovaných údajov a anonymná funkcia ktorá sa zavolá ak požiadavka spĺňa všetky parametre.

Na záver sa treba postarať o to aby server dokázal naservírovať dáta podľa požiadavky. Tieto dáta získava z databázy a o tento proces sa stará trieda `DatabaseManager` ktorý ma na starosti akúkoľvek manipuláciu s databázou.

```
$connection = new mysqli(server, meno, heslo, db, port);  
$query = prepare("SELECT * FROM users");  
$query -> execute();  
  
if ($query) {  
    // Požiadavka na databázu prebehla úspešne  
} else {  
    // Niekde nastala chyba  
}
```

Vyššie je znázornený jednoduchý kód toho ako prebieha manipulácia s databázou pomocou `mysqli` triedy. V prvom rade je potrebné si vytvoriť inštanciu triedy `mysqli` a to zadáním potrebných parametrov do konštruktora akými sú url adresa alebo IP adresa servera kde sa databáza nachádza, prihlasovacie meno, heslo, názov konkrétnej databázy a port na ktorom je spustený databázový server.

**Poznámka:** *Pre pochopenie celej architektúry backendu bolo nevyhnutné vysvetlenie toho ako tieto tieto kľúčové triedy pracujú pretože je na nich postavená celá logika toho ako web funguje na pozadí.*

## 3 Detailný opis postupu tvorby sociálnej siete

### 3.1 Kostra webu

Na začiatku toho ako som začal robiť webstránku som si musel premyslieť spôsob akým budú súbory poukladané v priečinkoch, keďže som chcel docieľiť aby navigácia medzi súbormi bola jednoduchá a intuitívna.

```

+---assets
+---css
|   +---components
|   \---routes
+---img
|   +---posts
|   \---profiles
+---js
|   +---components
|   |   \---util
|   \---routes
+---libs
+---php
\---sql

```

Tabuľka 1 – Vysvetlivky pre vyššie zobrazenú priečinkovú štruktúru

Cesty	Typy súborov nachádzajúce sa v priečinkoch
/assets	Súbory ktoré sa nemenia ako pozadie a favicony
/css	Pravidlá štyľovania pre celý web
/css/components	Pravidlá štyľovania pre jednotlivé komponenty
/css/routes	Pravidlá štyľovania pre jednotlivé sekcie
/img	Fotografie pridané užívateľmi
/img/posts	Fotografie k príspevkom
/img/profiles	Profilové fotografie užívateľov
/js	Skripty ktorých logika sa týka hlavne frontendu
/js/components	Skripty v ktorých je zadefinovaná logika komponentov
/js/components/util	Pomocné skripty uľahčujúce zapis niektorých procesov
/js/routes	Skripty v ktorých je zadefinovaná logika sekcií
/libs	Externé knižnice ktoré web využíva
/php	Skripty ktorých logika sa týka backendu
/sql	Súbory ktoré obsahujú presnú štruktúru SQL tabuliek

### 3.1.1 Frontend - komponenty

#### 3.1.1.1 Súbor `index.php` & `/php/Util.php`

Súbor `index.php` sa skladá z troch častí. Prvou je hlavička súboru kde sa nachádzajú skripty ktoré stránka využíva a taktiež sa sem automaticky vkladajú všetky existujúce `.css` súbory ktoré sa nachádzajú v priečinku `css`.

```
<?php
    $util = new Util();
    $util -> importStylesFrom("css");
?>
```

Dôvod prečo som chcel zautomatizovať pridávanie týchto súborov do hlavičky bol ten že som pridával `.css` súbor pre každý komponent zvlášť a chcel som mať istotu že sa mi všetky tieto štýly bez problémov načítajú. O tento zautomatizovaný proces sa stará PHP trieda `Util`, ktorá predtým ako načíta web prejde rekurzívne priečinok `css`, získa všetky súbory ktoré sa v ňom nachádzajú a vloží ich v danom formáte:

```
<link rel='stylesheet' href='css\components\auth-box.css'
type='text/css'>
```

Druhou časťou súboru je telo (**body**), ktoré je hlavnou časťou ktorá je viditeľná pre užívateľa. V tomto tele sa nachádza jediný element `div` ktorý obsahuje id podľa ktorého knižnica `Vue` vie kde má vykresľovať celú stránku.

Tretou časťou je sekcia **script** v ktorej sa importujú inicializačné skripty pre knižnice **Vuex** a **Vue Router** a predvolený a zároveň hlavný komponent v ktorom funguje celá stránka.

```
<script type="module">
    import App from './js/App.js'
    import router from './js/router.js'
    import store from './js/store.js'

    new Vue({
        router,
        store,
        render: h => h(App)
    }).$mount('#app')
</script>
```

Na inicializáciu knižnicu `Vue` sa použije konštruktor ktorého parametrom je objekt obsahujúci vyššie spomínané inicializačné skripty a metódu ktorá vykresľuje hlavný komponent. Na záver sa na koniec tohto konštruktora napíše funkcia `$mount` ktorej parameter je id elementu v ktorom sa má vykresľovať tento predvolený komponent.

### 3.1.1.2 Súbor /js/App.js

V súbore App.js sa definujú kľúčové komponenty ktoré sú viditeľné pre celú stránku ako napríklad okno s načítavaciou animáciou a komponent ktorý informuje užívateľa o možnej chybe ktorá môže za behu nastať. Taktiež obsahuje nastavenie usporiadania hlavných komponentov akými je vedľajší panel s navigáciou, horná lišta, a komponent zobrazujúci sekciu ktorá sa ma zobrazovať.

```
export default {  
  components: {},  
  data() {},  
  provide(),  
  computed: {},  
  methods: {},  
  template: ``  
}
```

Vyššie je znázornený kód ktorý reprezentuje štruktúru komponentu pre knižnicu Vue.

Tabuľka 2 – Vysvetlivky pre vyššie zobrazenú štruktúru komponentu

Vlastnosti objektu	Obsah konkrétnych vlastností objektu
components	Objekt v ktorom sú zapísané komponenty ktoré využíva daný komponent
data	Funkcia ktorá navracia objekt obsahujúci premenné s ktorými komponent pracuje
provide	Funkcia ktorá navracia objekt obsahujúci metódy a premenné ktoré môžu používať dcérske komponenty
computed	Objekt ktorý obsahuje dynamické premenné ktoré môžu za behu meniť svoju hodnotu v závislosti od iných premenných alebo v závislosti od nami zadaných logiky
methods	Objekt ktorý obsahuje metódy/funkcie s ktorými komponent pracuje
template	Obsahuje reťazec ktorý obsahuje HTML kód s ktorým komponent pracuje a ktorý sa vykreslí pri použití tohto komponentu

V tomto komponente sa nachádzajú funkcie ktoré sa primárne starajú o viditeľnosť vedľajšieho navigačného panelu, spracovanie vstupu do poľa vyhľadávania a funkcie ktoré sa starajú o viditeľnosť a dizajn informačného komponentu **Message** a načítavacieho okna.

Tento súbor hrá veľkú rolu čo sa týka zobrazovania obsahu podľa toho či je užívateľ prihlásený alebo nie. Ak užívateľ nieje prihlásený tak sa mu zobrazí komponent **AuthBox** a v prípade že je tak sa začnú načítavať príslušné sekcie.

#### 3.1.1.3 Súbor /js/store.js

Tento súbor obsahuje **Vuex.Store** objekt (úložisko) v ktorom sa nachádzajú dáta s ktorými aplikácia môže manipulovať a mať prístup počas behu celej aplikácie.

```
export default Vuex.Store({  
  state: {},  
  mutations: {},  
  getters: {}  
})
```

Vyššie je znázornený kód ktorý ukazuje základnú štruktúru tohto **Vuex** objektu.

Tabuľka 3 – Vysvetlivky pre vyššie zobrazenú štruktúru Vuex objektu

Vlastnosti objektu	Obsah konkrétnych vlastností objektu
state	Obsahuje objekt obsahujúci konkrétne premenné ktoré toto úložisko bude obsahovať
mutations	Obsahuje objekt ktorý obsahuje funkcie ktoré manipulujú s premennými nachádzajúce sa v objektovej vlastnosti <b>state</b>
getters	Obsahuje objekt obsahujúci metódy/funkcie ktoré navracajú premenné z objektovej vlastnosti <b>state</b>

V súbore store.js sa nachádzajú konkrétne iba jeden objekt ktorý zaobaluje užívateľské meno a autorizačný token ktorý slúži pre vykonávanie akejkoľvek akcie s backendom. Dôvod prečo sú na manipuláciu potrebné externé metódy je ten že Vuex sa takto stará o reaktivitu týchto premenných a vie ľahko zistiť kedy sa hodnota premennej zmenila.

Čo sa týka praktického využitia, tak toto úložisko poskytuje okrem vyššie spomenutých užívateľských dát aj špeciálne funkcie ktoré navracajú stav týchto premenných a to konkrétne či sú prázne alebo vyplnené. Takisto

#### 3.1.1.4 Súbor /js/router.js

Tento súbor obsahuje pole objektov ktoré definujú každú jednu trasu a k nej príslušný komponent ktorý sa má zobraziť, a taktiež inštanciu objektu **VueRouter**

ktorého parameter je objekt obsahujúci toto pole. Táto inštancia sa používa v súbore `index.php` kde sa vkladá ako parameter do konštruktora **Vue**.

```
const routes = [
  {
    path: '/',
    redirect: '/home'
  },
  {
    path: '/home',
    name: 'home',
    component: () => import("./routes/Home.js")
  },
  {
    path: '/profile/:username',
    name: 'profile',
    component: () => import("./routes/Profile.js")
  },
  {
    path: '/settings',
    name: 'settings',
    component: () => import("./routes/Settings.js")
  },
  {
    path: '/404',
    name: "error",
    component: () => import("./routes/PageNotFound.js")
  },
  {
    path: '/*',
    redirect: "/404"
  },
  {
    path: '*',
    redirect: "/404"
  },
]

const router = new VueRouter({
  routes,
})

export default router
```

V tomto poli sú nastavené všetky trasy k existujúcim sekciám no takisto tu je aj nastavená logika ktorá sa stará o presmerovanie na stránku s chybovou hláškou, v prípade že užívateľ chce navštíviť neexistujúcu stránku. Ako je možné vidieť tak pri nastavení komponentu pre konkrétnu trasu sa používa anonymná funkcia ktorá importuje konkrétny komponent. Je to neštandardný zápis ktorý sa používa vtedy keď chceme docieľiť to aby sa komponent nenačítal na začiatku behu stránky ale počas behu. Takýmto štýlom sa dá ušetriť výkon zariadenia z hľadiska sťahovania súboru a jeho



spracovania. V skratke to znamená že komponent/sekcia sa nenačíta až kým ju užívateľ nechce zobrazit'.

#### 3.1.1.5 Súbor /js/util/Requester.js

Funkcia Requester slúži ako nadstavba knižnice **Axios**. Táto funkcia pridáva malú časť logiky ktorá sa stará a ošetruje prípady kedy požiadavka na server neprebehla úspešne. Ako argument si vyžaduje objekt obsahujúci dáta ktoré bude požiadavka obsahovať, adresu kam bude požiadavka smerovať, metódu požiadavky a tzv. *callback* funkcie ktoré špecificky volajú podľa situácie. Funkcia Requester poskytuje možnosť zavolania týchto metód v prípade že požiadavka prebehne úspešne, že nastane chyba počas odosielania požiadavky a situáciu kedy celá situácia s odosielaním požiadavky skončí, bez ohľadu na to či úspešne alebo neúspešne.

#### 3.1.1.6 Súbor /js/components/AuthBox.js

Jeden z kľúčových komponentov ktorý kontroluje to či sa užívateľ prihlásil pod správnymi údajmi a podľa toho presmeruje na na sekcie. Tento komponent je z hľadiska výzoru jednoduchý no na pozadí prebieha veľa procesov ktoré sa snažia zabezpečiť bezpečný vstup do aplikácie. Ako je vidieť na **obrázku 8** tak tento komponent obsahuje v prípade registrácie 3 textové polia. V prípade prihlásenia iba 2 a to užívateľské meno a heslo. Taktiež obsahuje 2 tlačidlá kde prvé mení formulár z prihlasovacieho na registračný a opačne, a druhé ktoré už vykonáva konkrétnu akciu ktorá je závislá od typu formuláru. Takže ak je viditeľný formulár na prihlásenie tak odošle požiadavku na súbor **php/Login.php** so zadanými údajmi a ak je viditeľný registračný formulár tak odošle požiadavku na súbor **php/Register.php**. Následne ako sa vráti odpoveď zo servera, tak ju tento komponent musí vyhodnotiť, a to konkrétne či ide o chybovú hlášku alebo o informačnú hlášku o úspešnej registrácii. Hláška o úspešnom prihlásení nieje potrebná keďže užívateľ si hneď všimne že prihlásenie prebehlo úspešne na základe následného presmerovania na domovskú stránku.

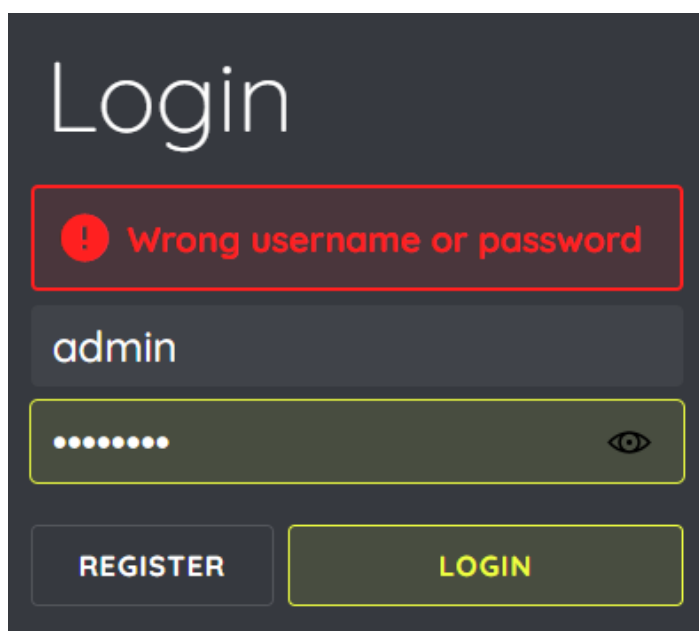
Tento komponent okrem autorizácie obsahuje dynamickú premennú **canPerformPrimaryAction** ktorá navracia booleovskú hodnotu ktorá vyhodnocuje či sú vyplnené všetky potrebné polia a na základe toho nastaviť tlačidlo na klikateľné.

Vzhľadom na to že toto je prvý komponent ktorý je pre užívateľa viditeľný tak obsahuje funkciu **mounted()** ktorá sa spúšťa pri inicializácii komponentu (niečo podobné

konštruktoru v OOP). Obsahom tejto funkcie je logika ktorá kontroluje či sa v pamäti **localStorage** nenachádzajú údaje o užívateľovi z predošlých prihlásení. Ak áno tak ich vyberie a automaticky podľa nich prihlási užívateľa. Tieto údaje sú pri každom prihlásení na novo uložené do tohto úložiska.

#### 3.1.1.7 Súbor /js/components/**Message.js**

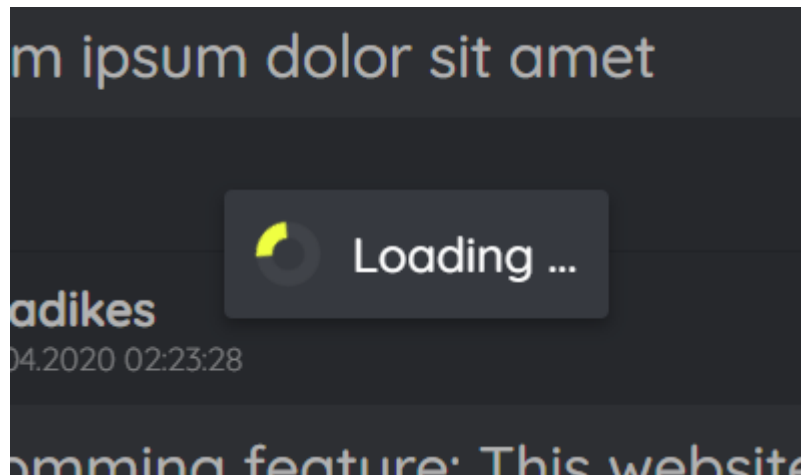
Komponent ktorý slúži iba hlásenie pre užívateľa pokiaľ sa za behu stránky vyskytne chyba. Bol navrhnutý tak aby ho mohol použiť akýkoľvek dcérsky komponent komponentu App. Týmto dcérskym komponentom poskytuje možnosť nastavenia jeho textu a farby.



Obrázok 9 – Ukážka Message komponentu

#### 3.1.1.8 Súbor /js/components/**LoadingBox.js**

Komponent ktorého jedinou úlohou je sa zobrazit' zakaždým keď na pozadí prebieha komunikácia so serverom.



Obrázok 10 – Ukážka LoadingBox komponentu

To aby sa koliečko vedľa textu mohlo točiť som docielil pomocou **animácií** ktoré mi umožňuje vytvoriť jazyk **CSS**.

```
.loading-box-loader {
  border: 8px solid var(--color-controls-normal);
  border-radius: 50%;
  border-top: 8px solid var(--color-accent);
  width: 32px;
  height: 32px;
  -webkit-animation: spin 2s linear infinite;
  animation: spin 2s linear infinite;
}

@-webkit-keyframes spin {
  0% { -webkit-transform: rotate(0deg); }
  100% { -webkit-transform: rotate(360deg); }
}

@keyframes spin {
  0% { transform: rotate(0deg); }
  100% { transform: rotate(360deg); }
}
```

Vyššie je znázornený kód ktorý okrem iného ukazuje mechanizmus ktorým sa dá docieľiť rotácia elementu. Selector **.loading-box-loader** odkazuje na krútiace sa koliečko. Nižšie sú zadefinované animácie pre tradičné prehliadače a taktiež pre prehliadače používajúce vykresľujúci engine s názvom **WebKit**. V týchto animáciach stačí zadať počiatočný stav elementu a záverečný. Potom pomocou atribútu **animation** môžeme nami vytvorenú animáciu použiť, a k nej môžeme priložiť zopár parametrov ktoré určujú dĺžku animácie jej priebeh a to či prebehne raz alebo bude prebiehať nepretržite.

### 3.1.1.9 Súbor /js/components/Toolbar.js

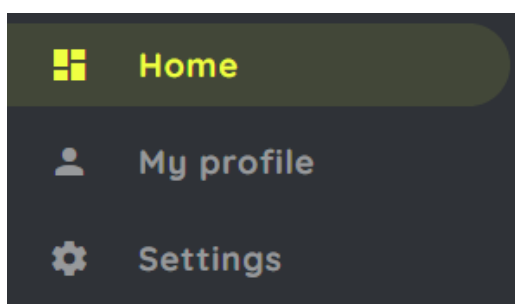
Jednoduchý komponent ktorý je vložený pre každú užívateľskú sekciu. Obsahuje 2 tlačidlá kde jedno slúži ako krok späť vrámci aplikácie a druhé pre odhlásenie. Takisto obsahuje aj nadpis ktorý sa dynamicky mení v závislosti od užívateľovej lokácie na webe. Taktiež obsahuje 2 tzv. *tooltip* objekty ktoré slúžia ako popis funkcie tlačidiel keď na ne užívateľ prejde kurzorom na počítačovej myši.



Obrázok 11 – Výzor hornej lišty

### 3.1.1.10 Súbor /js/components/RouterMenu.js

Jednoduchý komponent ktorý slúži ako navigačný panel v ktorom sú dostupné odkazy na rôzne sekcie tejto webstránky. Komponent obsahuje premennú **routes** ktorá je pole objektov kde každý objekt reprezentuje cestu ku každej zo sekcií, no narozdiel od pola routes ktoré sa nachádza v súbore **router.js** je toto len pomôcka ktorá navyše obsahuje ikonu a nadpis sekcie, a je to pole podľa ktorého sa automaticky pridávajú tieto linky cez direktívu **v-for** ktorá je poskytovaná knižnicou **Vue**. Ako je možné vidieť tak v menu sa nenachádzajú klasické hypertextové odkazy (**<a> elementy**) ale tzv. *router-link* komponenty ktoré sú automaticky dostupné knižnicou **Vue Router**. Hoci majú tieto komponenty tú istú funkciu ako hypertextové odkazy, jediný rozdiel je ten že pri presmerovaní neobnovujú celú stránku odznova.

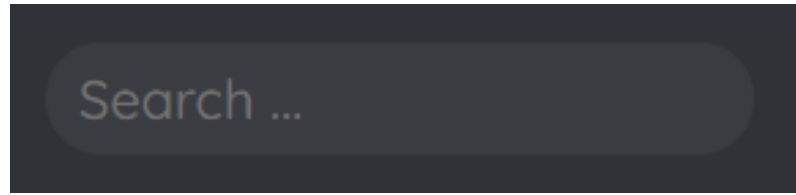


Obrázok 12 – Výzor navigačného panelu

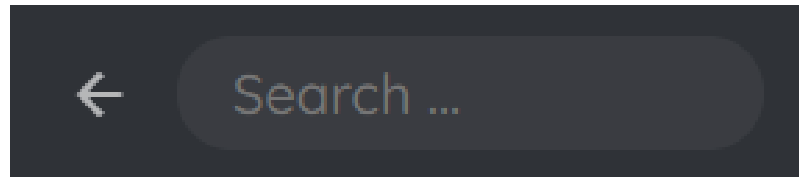
### 3.1.1.11 Súbor /js/components/SearchBar.js & SearchResultList.js

Relatívne komplexný komponent ktorý aj napriek tomu že obsahuje iba jedno textové pole a tlačidlo, počúva a následne reaguje na užívateľom vyvolané udalosti

akými sú napríklad kliknutie na komponent a na odkliknutie od komponentu, a podľa toho mení štýl zobrazenia.

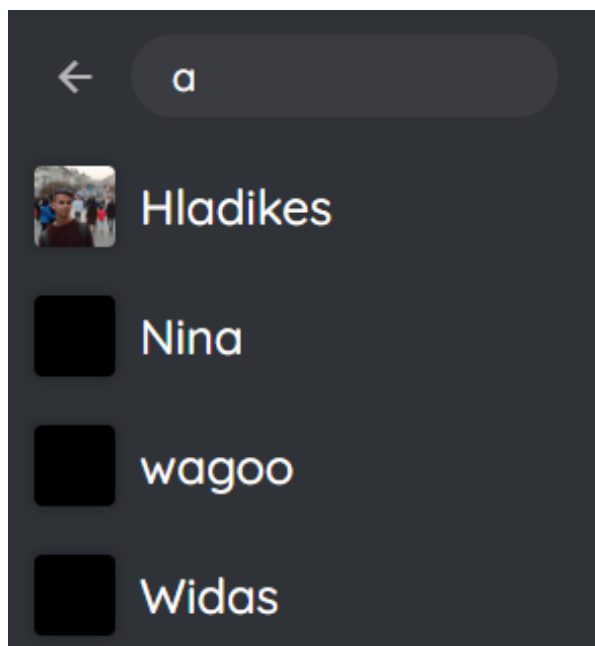


Obrázok 13 – Výzor textového poľa pre vyhľadávanie, v neaktívnom stave



Obrázok 14 – Výzor textového poľa pre vyhľadávanie, v neaktívnom stave

Ako je možné vidieť na **obrázkoch 13 a 14** tak komponent mení viditeľnosť tlačidla späť v závislosti od toho či užívateľ zaklikol textové pole alebo nie. Okrem toho že samotný komponent spracúva tieto zmeny tak poskytuje možnosť rodičovskému elementu zareagovať na tieto zmeny a to pomocou metódy `$emit` ktorej parametrami sú názov našej vlastnej udalosti a premennú ktorú chceme spolu s touto udalosťou poslať. Rodič je následne schopný si odchytiť tieto zmeny a následne vykonať nami doprogramovanú logiku. V tomto prípade je rodič **App.js** ktorý obsahuje okrem tohto komponentu aj komponent **SearchResultList.js**. Údaje ktoré súbor **App.js** odchyti následne odošle dcérskeho komponentu **SearchResultList** pomocou tzv. *props* ktorý si vie tieto dáta taktiež odchytiť. Tento komponent následne potom ako obdrží názov profilu ktorý užívateľ chce vyhľadať, pošle požiadavku s týmto názvom a server navráti korešpondujúce profily vo forme poľa.



Obrázok 15 – Ukážka zoznamu korešpondujúcich užívateľov

Na **obrázku 15** je ukázaný príklad toho ako tento komponent funguje. Povedzme že užívateľ zadá písmeno „a“. Tento text sa pošle v požiadavke serveru a server navráti zoznam profilov ktoré v názve obsahujú tento text.

Komponent `SearchResultList` dokáže sám zistiť kedy sa zmenil užívateľov vstup a na základe toho reagovať. Túto funkcionality mu umožňuje tzv. *watcher* čo je mechanizmus v knižnici `Vue` ktorý umožňuje pozorovať premennú, resp. jej hodnotu a na základe toho zareagovať.

```
watch: {
  query(newValue) {
    if (newValue) {
      this.getUsers(this.query)
    } else {
      this.usersList = []
    }
  }
}
```

Vo vyššie uvedenom kóde je ukážka toho ako tento *watcher* funguje. Je to objekt ktorý obsahuje funkcie ktoré majú ten istý názov ako premenné na ktoré majú reagovať. Akonáhle nastane zmena tak sa funkcia zavolá a zároveň ako argument pošle novú hodnotu premennej po jej zmene.

Ako som vyššie spomínal tak komponent `App` a `SearchResultList` komunikuje pomocou tzv. *props*. Toto je ďalší mechanizmus poskytnutý knižnicou `Vue` pomocou

ktorého si môžeme zadať vlastné atribúty ktoré komponent potrebuje pre svoje fungovanie. Hoci sa v komponente `SearchResultList` nenachádza premenná `query`, napriek tomu dokáže počúvať na jej zmenu. A to preto že táto premenná je definovaná mimo dát.

```
props: {  
  query: String  
},  
  
data() {  
  return {  
    userList: []  
  }  
},
```

Ako môžeme vyššie vidieť tak táto premenná je definovaná v objekte **props**. V tomto objekte si vieme zadať akýkoľvek atribút ktorý náš komponent potrebuje pre svoju prácu. Akonáhle je zadefinovaný tak rodič v ktorom sa komponent nachádza mu môže nasledujúcim štýlom poskytnúť/odoslať dáta:

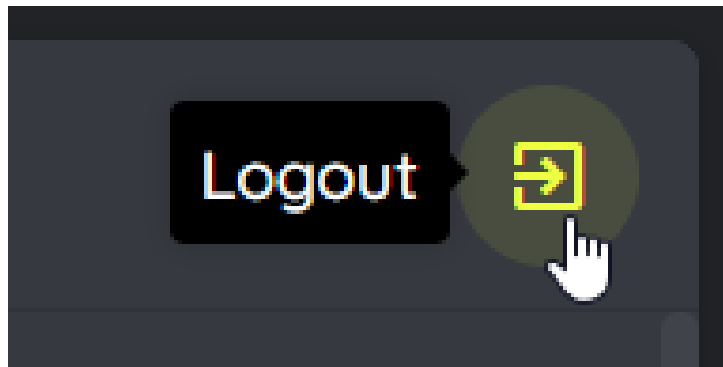
```
<SearchResultList  
  :query="searchQuery"  
  v-show="isSearchResultListVisible" >
```

Hoci sa v štandardnej verzii HTML nenachádza atribút **query**, Vue ho dokáže rozpoznať a manipulovať s ním.

#### 3.1.1.12 Súbor `/js/HoverButton.js`

Ako je možné vidieť tak v žiadnom z komponentov keď sa používa tlačidlo tak sa nepoužíva základný **<button>** element ale namiesto neho sa používa **HoverButton** komponent. Hoci tento komponent slúži ako tlačidlo tak ponúka extra funkcionality.

Prvou je tá že ak rodič poskytne dáta do **tooltip** atribútu tak je možné zobraziť pomocnú hlášku ktorá sa zobrazí pri prejdení tlačidlom kurzorom počítačovej myši alebo pri priamom dotyku tlačidla na dotykovej obrazovke.



Obrázok 16 – Ukážka toho ako vyzerá pomocná hláška

Druhou funkcionalitou je skôr estetická vec. Ak chceme na element v jazyku **CSS** zmeniť štýl iba pokiaľ sa užívateľ nad daným elementom vznáša kurzorom, tak mu pridáme tzv. *pseudo selector* a v ňom nastavíme štýl na ktorý sa má element zmeniť. Problémom však je že pri dotykových zariadeniach nieje žiadny kurzor a preto pri dotyku nenastane tá istá situácia. Z môjho hľadiska to je škoda pretože užívateľ by mal zakaždým získať instantnú odozvu ohľadom toho či stláča tlačidlo alebo nie.

Preto sa v `HoverButton` komponente nachádza *props* s názvom `nClass` ktorá ako parameter berie **CSS class selector** a aplikuje ho v prípade že sa užívateľ buď vznáša kurzorom nad tlačidlom alebo sa ho prstom dotýka na obrazovke dotykového zariadenia. Toto počúvanie na dané udalosti funguje cez integrované poslucháče udalostí (*event listeners*). Tieto poslucháče sa nastavujú pri inicializácii `HoverButton` komponentu a počúvajú až pokiaľ komponent nezanikne.

```
button.addEventListener("touchend", () => {
    this.isActive = false

    if (this.isDeviceTypeSet &&
this.hasDeviceTouchableScreen) {
        button.classList.remove(this.nClass)
    }
}, { passive: true })

button.addEventListener("mouseover", () => {
    this.isActive = true

    if (!this.isDeviceTypeSet) {
        this.isDeviceTypeSet = true
        this.hasDeviceTouchableScreen = false
    }

    if (this.isDeviceTypeSet &&
!this.hasDeviceTouchableScreen) {
        button.classList.add(this.nClass)
    }
})
```



```

    }, { passive: true })

    button.addEventListener("mouseout", () => {
        this.isActive = false

        if (this.isDeviceTypeSet &&
!this.hasDeviceTouchableScreen) {
            button.classList.remove(this.nClass)
        }
    }, { passive: true })

```

Vyššie je ukážka toho ako komponent počúva na udalosť prejdienia kurzorom nad elementom.

### 3.1.1.13 Súbor /js/routes/Home.js

Hoci tento súbor slúži ako komponent, napriek tomu je používaný ako sekcia tejto webstránky. Preto sa nachádza v priečinku **routes** a bol definovaný v súbore /js/router.js.

Tento komponent má jednoduchú úlohu. Dať užívateľovi možnosť pridať príspevok a prezerať príspevky ostatných užívateľov. Celá sekcia obsahuje dokopy 2 komponenty. Prvým je komponent **Poster** a druhým je **PostItem**. Komponent **Poster** je samostatnou jednotkou ktorej úloha je získať vstup od užívateľa a po stlačení tlačidla odoslať tento text ako príspevok na server, kde sa uloží do databázy a následne ho je možné vidieť medzi ostatnými príspevkami.

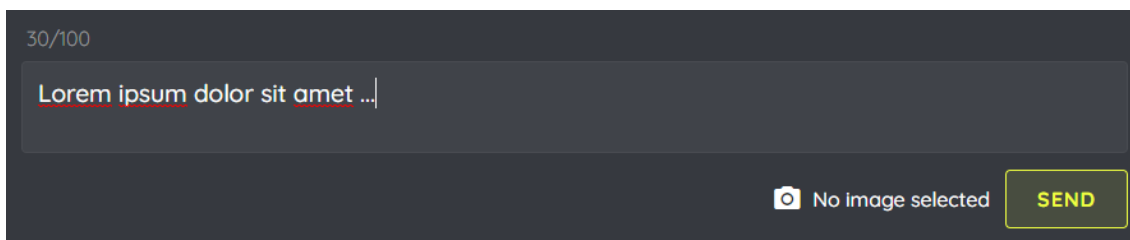
```

<div class="post-list">
  <Poster @onPostAdded="load"></Poster>

  <PostItem
    v-for="(item, index) in posts"
    :key="item.ID"
    :username="item.username"
    :isLiked="item.isLikedByUser"
    :profilePicture="item.profilePicture"
    :image="item.postImage"
    :postID="item.ID"
    :time="item.date"
    :likes="item.likes"
    :content="item.postText"></PostItem>
</div>

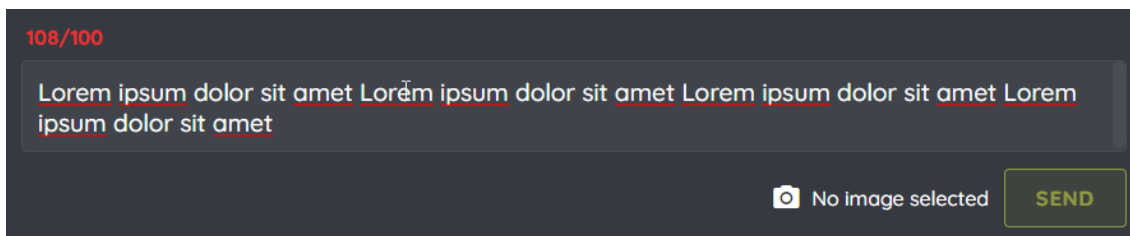
```

Vyššie uvedený kód ukazuje ako malý počet riadkov jazyka HTML tento komponent skutočne využíva.



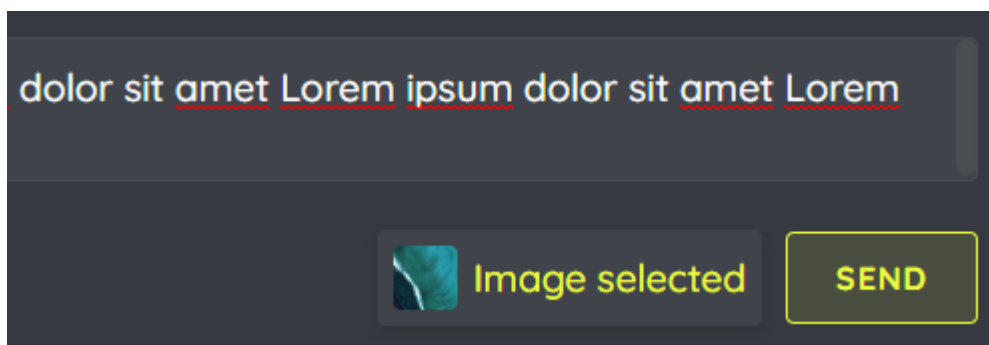
Obrázok 17 – Výzor Poster komponentu

Aj napriek tomu že komponent má jednoduchú úlohu obsahuje hneď viacero špeciálít. Automaticky počas písania kontroluje počet znakov a v prípade že počet znakov presahuje hodnotu 100 tak informuje užívateľa podfarbením horného limitu červenou farbou, vid' obrázok 18, a takisto mu v tomto prípade neumožní stlačiť tlačidlo odoslať (send).

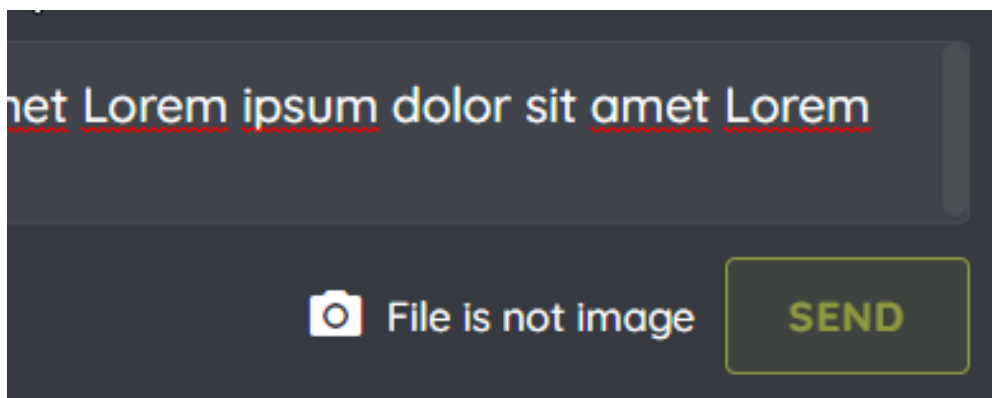


Obrázok 18 – Výzor Poster komponentu po prekročení limitu

Ďalšou funkciou ktorú poskytuje tento komponent je využitie ďalšieho komponentu a tým je **ImageUploader**. Tento komponent slúži ako vstup pre súbory, v tomto prípade súbory obrázkového formátu. Po kliknutí sa zobrazí možnosť vybrať súbor z počítača. Následne potom ako užívateľ vyberie súbor sa prehodnotí či je formát podporovaný a ako áno tak zobrazí malý náhľad tohto obrázku s hlásením že obrázok bol zvolený. Ak nie tak vypíše hlásenie že súbor nieje obrázok.



Obrázok 19 – Prípád kedy vložený súbor je obrázok

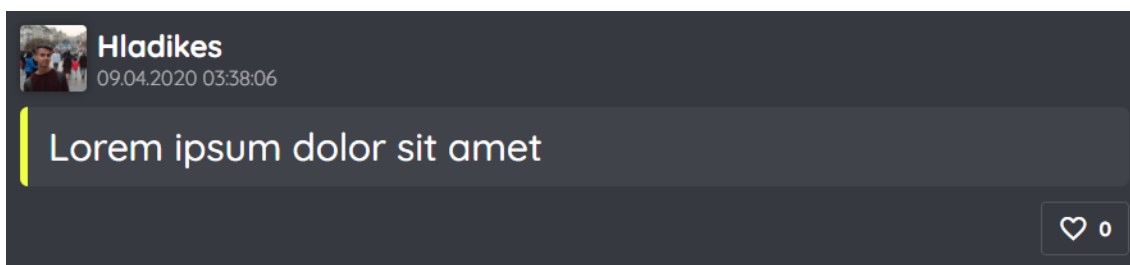


Obrázok 20 – Prípád kedy vložený súbor nieje podporovaný

Aj napriek tomu že tento komponent zohráva jednoduchú úlohu, tak na pozadí je veľa faktorov ktoré musí vziať do úvahy čo sa týka spracovania obrázku. Jedným je kontrola samotného súboru, ďalej samotné získavanie súboru z dočasnej pamäte a jeho zobrazenie a správne manipulovanie.

#### 3.1.1.15 Súbor /js/components/PostItem.js & Like.js

Ďalším komponentom ktorý domovská sekcia využíva je PostItem. Tento komponent slúži ako náhľad príspevku s možnosťou zmazania ktorá sa zobrazí iba v sekcii profilu a v prípade že autorom príspevku je prihlásený užívateľ.



Obrázok 21 – Výzor komponentu PostItem

Komponent sa skladá z viacerých elementov ktoré sú nevyhnutné pre zobrazenie všetkých údajov o príspevku. Obsahuje fotografiu autora, názov autora, dátum a čas pridania príspevku, text samotného príspevku a samostatný **Like** komponent ktorý slúži ako tlačidlo ktorým vie užívateľ označiť že sa mu daný príspevok páči.

```
<div class="post-container" v-if="!isDeleted">
  <div class="post-header">
    

    <div class="post-header-data">
      <router-link
        :to="'/profile/' + username">
```

```

class="post-header-username">{{ username
}}</router-link>
    <span class="post-header-time">{{ time
}}</span>
</div>

<HoverButton
    :tooltip="deletePostTooltip"
    @onClick="deletePost"
    v-if="isDeletable || false"
    class="post-action-delete"
    n-class="post-action-delete---hover"><i
class="material-icons">delete_outline</i></HoverButton>
</div>

    <div class="post-body">
        <span class="post-body-text">{{ content
}}</span>
        
    </div>

    <div class="post-footer">
        <Like :likes="likes" :liked="isLiked"
:id="postID"></Like>
    </div>
</div>

```

Kód vyššie ukazuje komplexitu **PostItem** komponentu. Hoci tento komponent nieje komplexný z hľadiska logiky, tak je komplexný a to z hľadiska štruktúry elementov z ktorých sa skladá

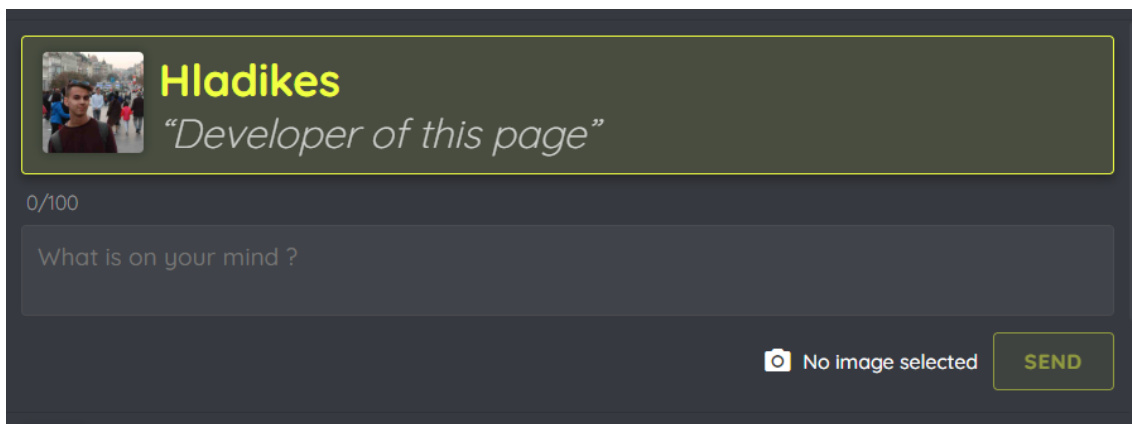
Obsahom každého **PostItem** komponentu je **Like** komponent ktorý je zodpovedný za korektné označovanie príspevku ako obľúbeného, a za korektné zobrazovanie počtu užívateľov ktorým sa príspevok páčil. Jedinou úlohou tohto komponentu na frontende je aby posielal po zmene akéhokoľvek stavu odoslal túto zmenu na server kde sa tento údaj hneď uloží pod konkrétny príspevok.

Čo sa týka výzoru tak toto tlačidlo (tak ako aj iné) využíva ikony z voľne dostupnej sady tzv. *Material Icons*. Je to sada ikon ktoré boli navrhnuté spoločnosťou Google, ktoré taktiež spĺňajú normy materiálového dizajnového systému, ktorý bol takisto vyvinutý touto spoločnosťou.

#### 3.1.1.16 Súbor /js/routes/Profile.js

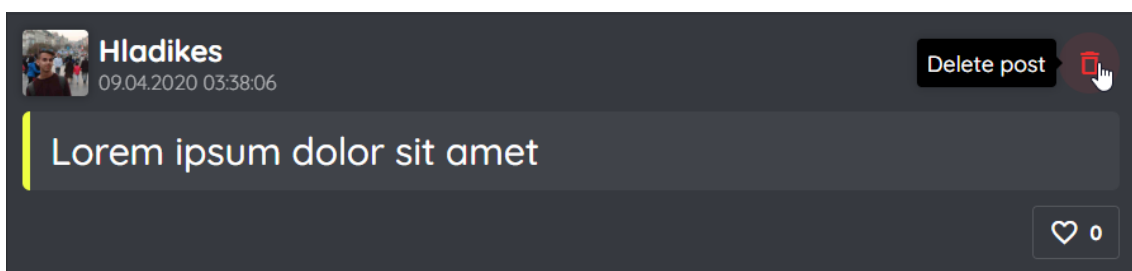
Sekcia/komponent **Profile** sa moc nelíši v porovnaní so sekciou **Home** no má zopár rozdielov. Tými rozdielmi sú že na stránke sú zobrazené príspevky iba daného užívateľa

a že sekcia obsahuje baner na vrchu celej sekcie v ktorom je fotografia prihláseného užívateľa, jeho meno a popis.



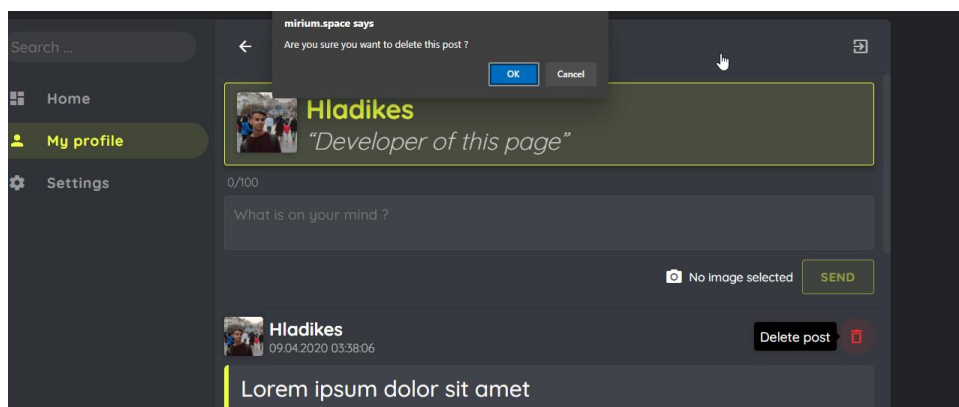
Obzárók 22 – Výzor baneru v sekcii Profil

Ďalším rozdielom je možnosť zmazania príspevku v prípade že autor tohto príspevku je prihláseným užívateľom. Ak áno tak sa mu na každom príspevku zobrazí tlačidlo ktoré mu umožňuje zmazať daný príspevok.



Obrázok 23 – Príspevok s možnosťou zmazania

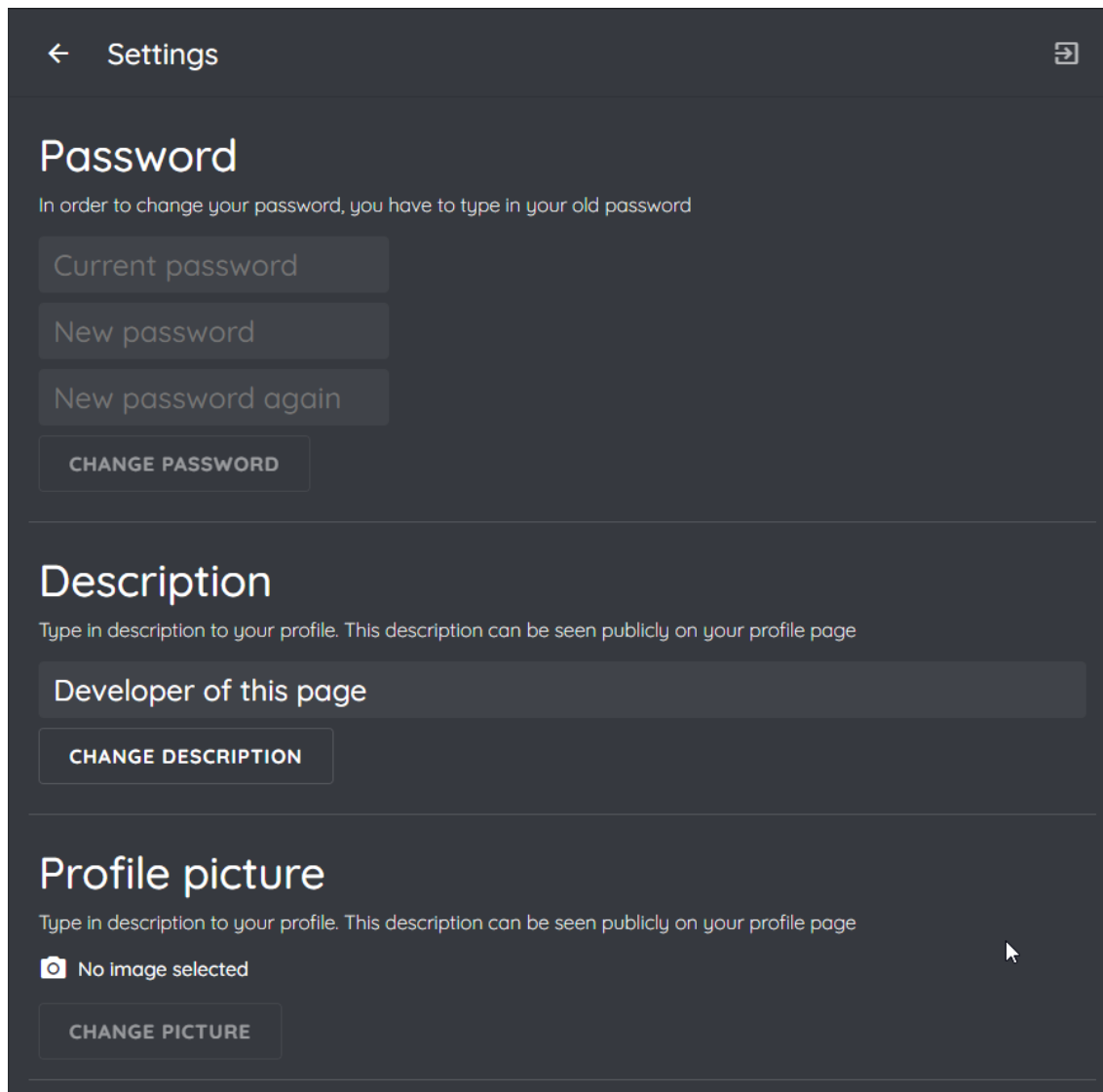
Po jeho kliknutí bude užívateľ vyzvaný klasickým dialógovým oknom s otázkou či si je naozaj istý že chce príspevok zmazať. Až po odkliknutí súhlasu sa daný príspevok zmaže.



Obrázok 24 – Ukážka dialógového okna pri mazaní príspevku

### 3.1.1.17 Súbor /js/routes/**Settings.js**

Komponent/sekcia Settings obsahuje zopár nastavení profilu uživateľa. Medzi ne patrí možnosť zmeny hesla, popisu profilu a profilovej fotografie. Sekcia využíva už všetky vyššie spomenuté komponenty pre svoj chod. Z hľadiska dĺžky kódu ide o najrozsiahlejší komponent z celej webstránky. Dôvodom je fakt že sa sekcia skladá z oveľa viac komponentov ako iné sekcie, a taktiež fakt že odosiela až 4 odlišné požiadavky na server.



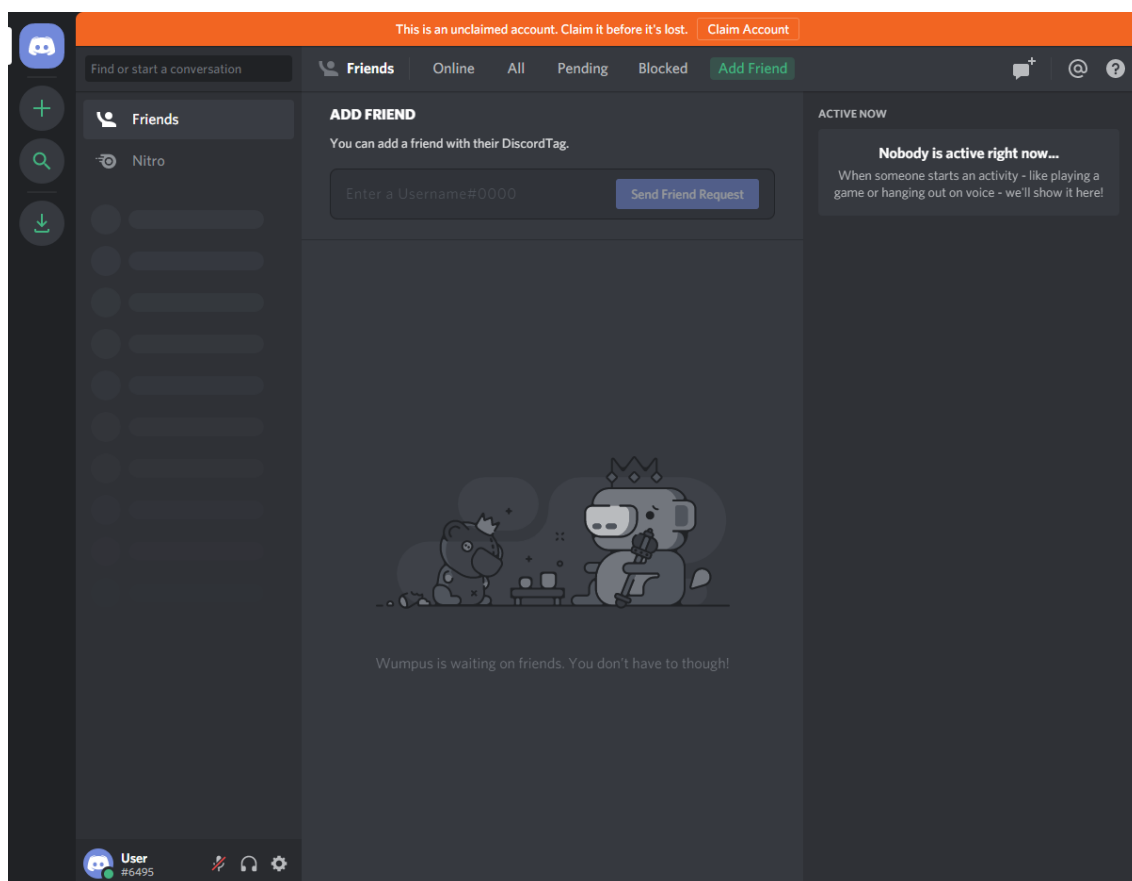
The screenshot shows a mobile application interface for a 'Settings' page. At the top, there is a back arrow and the title 'Settings', and a share icon on the right. The page is divided into three main sections: 'Password', 'Description', and 'Profile picture'. The 'Password' section has a subtitle 'In order to change your password, you have to type in your old password' and three input fields labeled 'Current password', 'New password', and 'New password again', followed by a 'CHANGE PASSWORD' button. The 'Description' section has a subtitle 'Type in description to your profile. This description can be seen publicly on your profile page' and a single-line text input field with the placeholder 'Developer of this page', followed by a 'CHANGE DESCRIPTION' button. The 'Profile picture' section has a subtitle 'Type in description to your profile. This description can be seen publicly on your profile page' (repeated), a camera icon with the text 'No image selected', and a 'CHANGE PICTURE' button. The entire interface is dark-themed with white text.

Obrázok 25 – Ukážka sekcie Settings

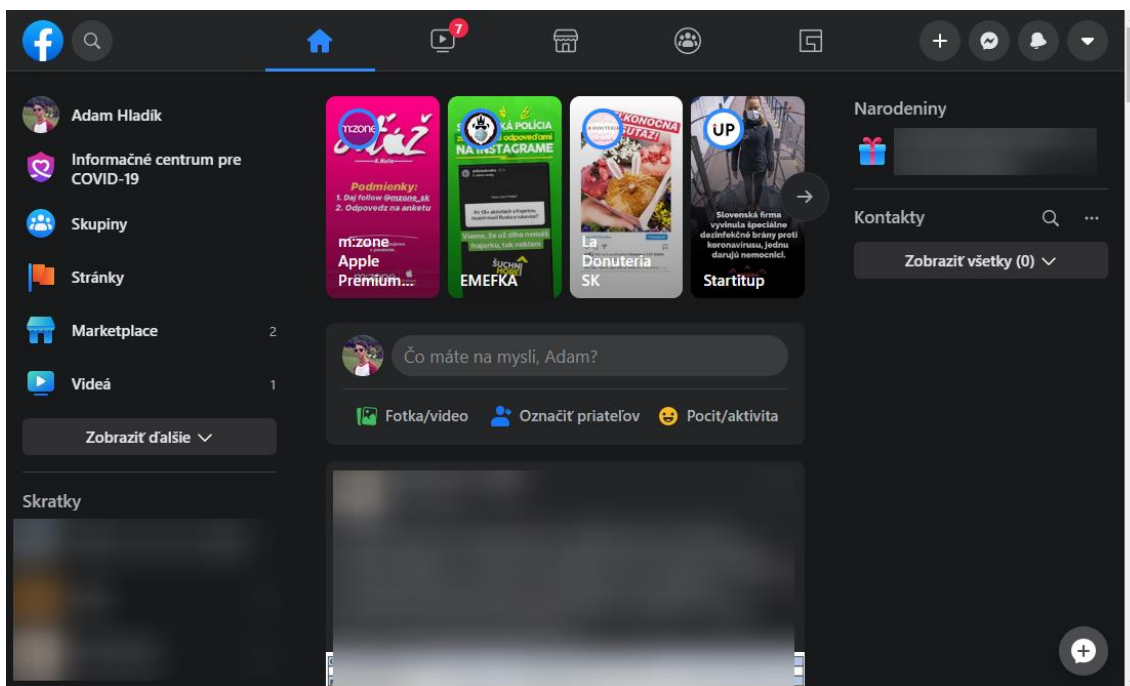
Táto sekcia sa skladá zo samostatných podsekcí, kde každá plní inú úlohu. Takisto má každá podsekcia svoj nadpis, špecifický popis nastavenia, adekvátne textové polia a tlačidlo na uloženie zmeny.

### 3.1.2 Frontend – dizajn

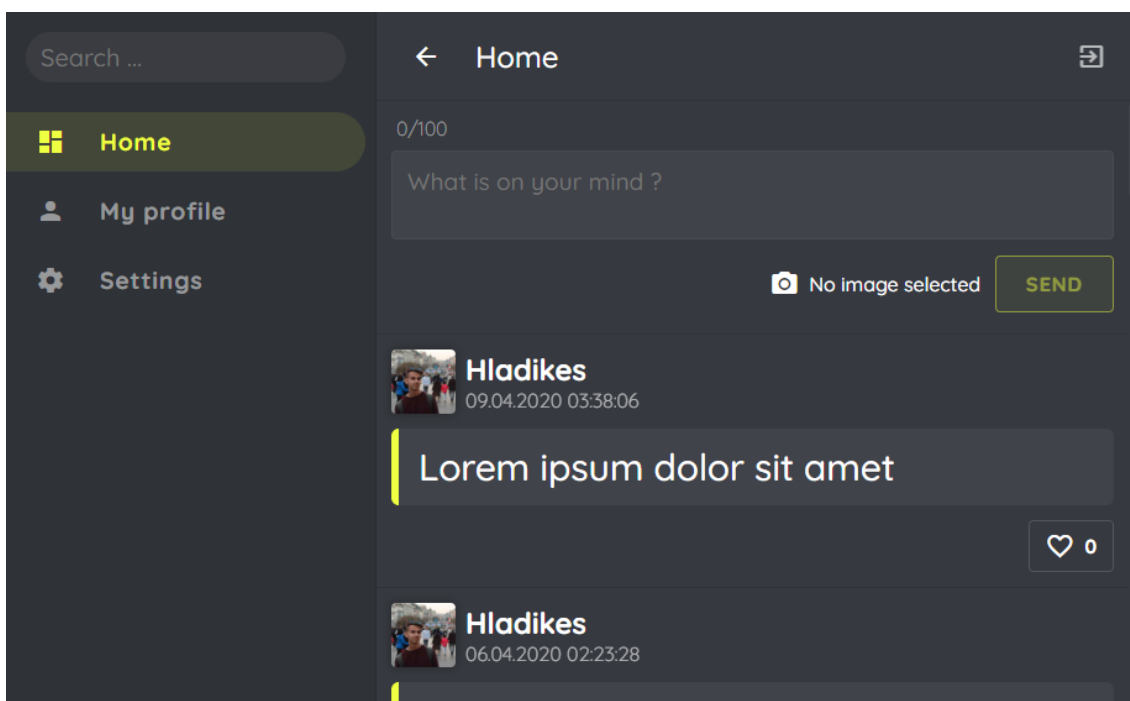
Ako som už spomínal tak pri tvorbe tejto stránky som sa inšpiroval dizajnom webovej stránky **Discord** a novej verzie stránky **Facebook**. Zo stránky Facebook som sa čiastočne inšpiroval tým ako boli umiestnené navigačné prvky na stránke, ako vedľajší navigačný panel, vrchná lišta a vyhľadávacie pole v rohu. Zo stránky Discord som sa inšpiroval hlavne farbami ktoré web obsahuje.



Obrázok 26 – Dizajn webu Discord



Obrázok 27 – Nový dizajn webu Facebook



Obrázok 28 – Dizajn mojej sociálnej siete

Ako je možné vidieť na **obrázku 26, 27, 28** tak skutočne stránka zdieľa niektoré dizajnové princípy, a taktiež princípy toho ako sú umiestnené jednotlivé elementy



**Poznámka:** *Vzhľadom na to že štylovanie webstránky je relatívne jednoduché, nepovažujem za nutné do dokumentácie znázorňovať to ako som písal jednotlivé .css súbory. Štýly všetkých elementov a komponentov som tvoril sám podľa vlastného uváženia tak, aby po estetickej stránke vyzerali dobre, konzistentne a zároveň aby boli dobre použiteľné pre koncového užívateľa.*

Aj napriek tomu že tvorenie týchto štýlov je relatívne jednoduché, tak bude nižšie popísaná funkcionálna trocha kľúčových .css súborov.

### 3.1.2.1 Súbor /css/colors.css

V tomto súbore sa nenachádzajú konkrétne pravidlá toho ako majú vyzeráť niektoré elementy, no predsa je to najľúčovejší súbor. V tomto súbore sú totiž zadefinované premenné ktoré obsahujú farby podľa ktorých sa konkrétne zobrazujú ostatné elementy.

Syntax zápisu týchto premenných vyzerá asi takto:

```
:root {
  --color-accent: hsl(65, 100%, 63%);
  --color-accent-01: hsla(65, 100%, 63%, 0.1);
  --color-primary: hsl(220, 8%, 23%);
  --color-primary-darker: hsl(218, 8%, 20%);
  --color-primary-dark: hsl(220, 7%, 14%);
}
```

Do *selectoru* :root sa zadefinujú premenné ktoré chceme aby boli k dispozícii pre všetky ostatné elementy. Syntax definovania týchto premenných, ako je možné vidieť, obsahuje dve pomlčky pred začiatkom názvu premennej, dvojbodku a následne jej hodnotu.

Dôvod prečo tu boli použité premenné je ten, že v prípade že by sme chceli zmeniť napríklad žltú farbu viacerých elementov tak by sme manuálne museli nájsť každý jeden *selector* ktorý túto farbu využíva a zmeniť ho. Takto sme schopný zmeniť farbu na jednom mieste a výsledok sa odrazí na všetkých elementoch ktorý ju používali.

Okrem týchto premenných sa v tomto súbore nachádza aj tzv. *media query* v ktorom sa nachádzajú ďalšie pravidlá zobrazovania ale iba v konkrétnom kontexte.

```
@media (prefers-color-scheme: light) {

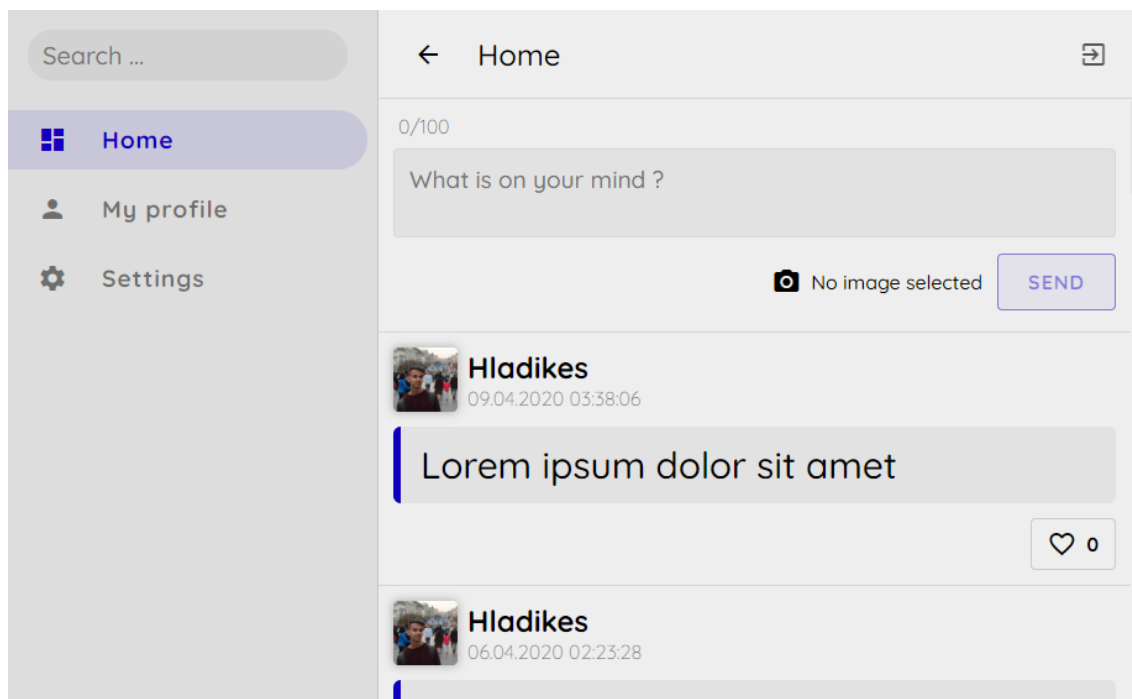
  :root {
    --color-accent: hsl(245, 100%, 37%);
    --color-accent-01: hsla(245, 100%, 37%, 0.1);
    --color-primary: #EEEEEE;
    --color-primary-darker: #DDDDDD;
  }
}
```

```

    --color-primary-dark: #BBBBBB;
  }
}

```

V tomto prípade sa pravidlá aplikujú len vtedy ak má užívateľ na zariadení, na ktorom web otvára, nastavený svetlý režim. Ak tento prípad nastane tak všetky farby sa zmenia a budú nahradené za svetlejšie.



Obrázok 29 – Dizajn webu za použitia svetlej témy na zariadení

Tieto zmeny sa aplikujú hneď bez nutnosti obnovovania webstránky.

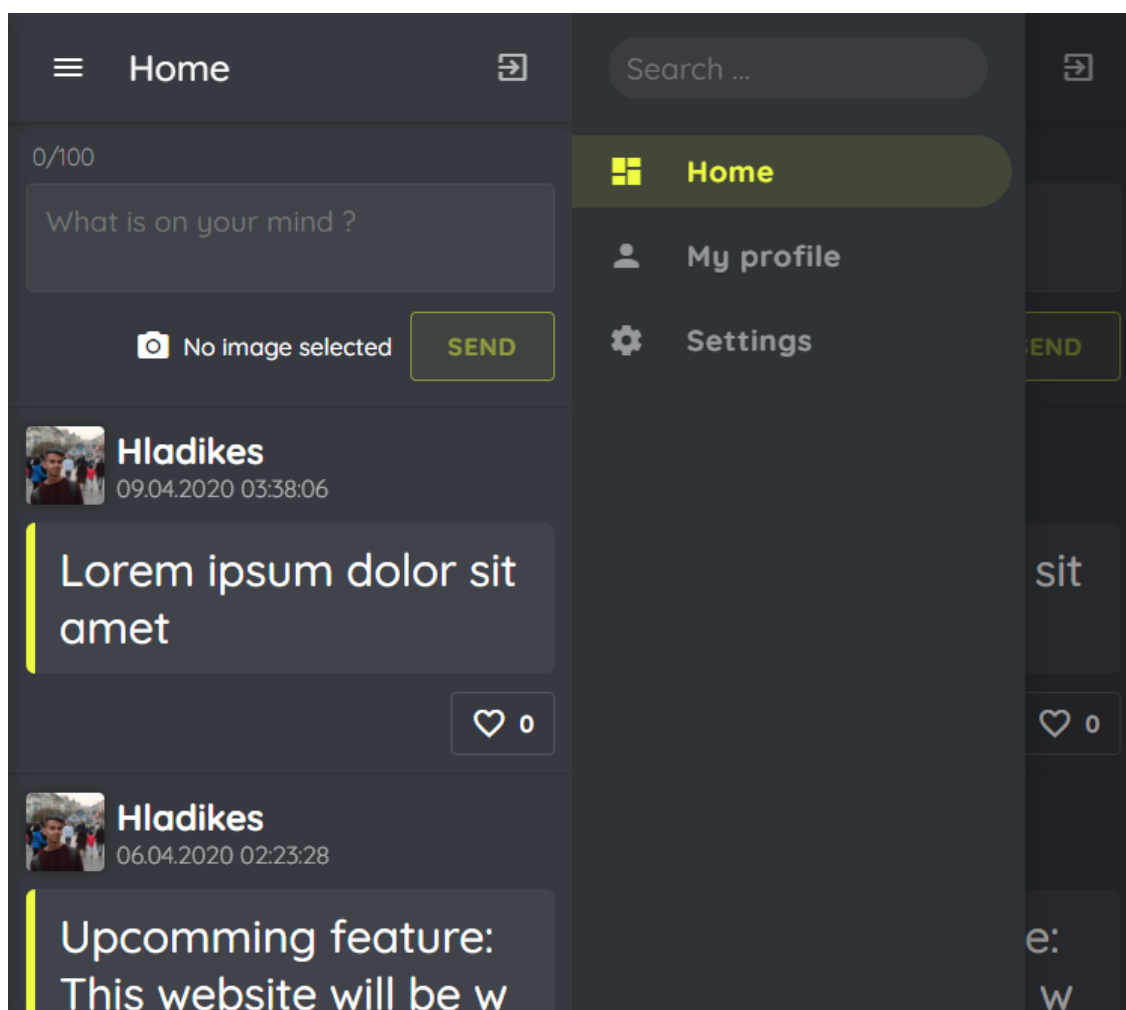
Čo sa týka vyššie spomenutých *media queries* tak existuje ešte jedna veľmi využívaná podmienka podľa ktorej sa dokážu za behu meniť pravidlá zobrazovania.

```

@media only screen and (max-width: 800px) { ... }

```

V tomto prípade sa zmenia pravidlá iba vtedy ak sa zmení šírka zariadenie v ktorom je stránka zobrazená. Nielen že môžeme zareagovať na zmenu, ale môžeme nastaviť kedy chceme aby zmena vypukla, konkrétnejšie pri akej veľkosti obrazovky sa majú aplikovať rôzne pravidlá. Na **obrázku 28** môžeme okrem dizajnu vidieť aj stránku v prípade že je otvorená na zariadení ktorého veľkosť obrazovky je >800px. Na **obrázku 30 a 31** je však znázornená ukážka toho ako stránka vyzerá a ako sa chová pokiaľ šírka zariadenia na ktorom sa zobrazí je <800px.



Obrázok 30 (vľavo), Obrázok 31 (vpravo) – Ukážka výzoru webstránky na menších obrazovkách

### 3.1.2.2 Súbor /css/reset.css

Súbor reset.css má veľmi jednoduchú no zároveň dôležitú úlohu. Jeho úlohou je resetovať niektoré konkrétne vlastnosti pre všetky elementy na stránke. Táto funkcionality je dôležitá pri umiestňovaní a nastavovaní veľkosti elementov, keďže niektoré majú prirodzene zadané tieto nežiadúce vlastnosti čo môže niekedy viesť k problémom.

```
* {
  outline: 0px solid transparent;
  border: 0;
  padding: 0;
  margin: 0;
  appearance: none;
  -webkit-appearance: none;
  -webkit-tap-highlight-color: transparent;
  box-sizing: border-box;

  font-family: 'Quicksand', sans-serif;
}
```

```
}  
  
body, html {  
    background-color: var(--color-primary-dark);  
}
```

Okrem svojej reset funkcie obsahuje súbor pozadie elementu **body** a **html**.

### 3.1.2.3 Súbor /css/mirium-ui.css

Súbor obsahuje štýly pre často používané elementy akými sú hlavne tlačidlá a textové polia. Dôvod prečo som vytvoril samostatný súbor je ten že pokiaľ nastane prípad kedy budem chcieť zmeniť ich výzor tak mi bude stačiť zmeniť tieto štýly iba v jednom súbore. Tieto často používané tlačidlá a textové polia je možné vidieť z predlohlých obrázkov ktoré sa nachádzajú v tomto dokumente.

## 3.1.3 Backend

### 3.1.3.1 Súbor /php/DatabaseManager.php

Ide o najorozsiahlejší súbor ktorý sa nachádza na celej webovej stránke. Obsahuje 360 riadkov. Tento súbor alebo teda objekt/trieda, je zodpovedný za všetku komunikáciu s databázou. Okrem toho že to je najrozsiahlejší súbor tak je aj najdôležitejší. Na tomto súbore totiž funguje celá architektúra backendu na tejto stránke. Chybovosť tohto súboru by zapríčinila nefunkčnosť celej sociálnej siete keďže je primárne závislá od dát ktoré sa nachádzajú v databáze.

Na začiatok toho aby DatabaseManager mohol správne fungovať bolo potrebné v paneli **phpMyAdmin** vytvoriť databázu s potrebnými tabuľkami. Tie boli vytvorené pomocou nižšie uvedených troch príkazov:

```
CREATE TABLE users (  
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  
    username VARCHAR(255) NOT NULL,  
    password VARCHAR(255) NOT NULL,  
    description VARCHAR(255),  
    picture VARCHAR(255) NOT NULL  
);  
  
CREATE TABLE posts (  
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  
    username VARCHAR(255) NOT NULL,  
    postText VARCHAR(255) NOT NULL,  
    postImage VARCHAR(255),  
    date VARCHAR(255) NOT NULL,  
    likes INT
```

```
);

CREATE TABLE likes (
    username VARCHAR(255) NOT NULL,
    postID INT NOT NULL
)
```

Akonáhle existuje databáza s ktorou môže DatabaseManager pracovať, tak bolo potrebné naprogramovať logiku ktorá už priamo komunikuje s databázou.

**Tabuľka 4 – Vysvetlivky pre jednotlivé metódy triedy DatabaseManager**

Metódy	Vysvetlenie metód
__construct	Konštruktor triedy. Je volaný pri vytváraní inštancii tejto triedy. Slúži pre pripájanie sa na databázu
registerToken	Metóda ktorá vytvorí náhodný autorizačný token, ktorý následne uloží do databázy
isAccountValid	Metóda vracajúca logickú hodnotu ktorá určuje či je zadané užívateľské meno a heslo správne
isAccountTokenValid	Metóda vracajúca logickú hodnotu ktorá určuje či je zadané užívateľské meno a autorizačný token
createAccount	Metóda ktorá vytvára účet na webe na základe poskytnutých parametrov
accountExists	Metóda vracajúca logickú hodnotu ktorá určuje či existuje užívateľ
getProfileByUsername	Metóda vracajúca pole údajov o užívateľovi
sendPost	Metóda ktorá ukladá príspevok
getPosts	Metóda ktorá navracia pole príspevkov požadovaných užívateľov
getUsers	Metóda ktorá navracia pole požadovaných užívateľov
sendLike	Metóda ktorá do v príspevku v databáze modifikuje číslo/aktuálny počet ľudí ktorí označili že sa im páči príspevok
deletePost	Metóda ktorá maže príspevok z databázy
updateDescription	Metóda ktorá modifikuje status užívateľa
changePassword	Metóda ktorá mení heslo užívateľa
changeProfilePicture	Metóda ktorá mení profilovú fotografiu užívateľa

### 3.1.3.2 Súbor /php/RequestGuard.php

Funkcia triedy RequestGuard bola už vyššie vysvetlená. Nižšie je znázornený kód na základe ktorého táto trieda funguje:

```
<?php
    require_once "Reporter.php";
    class RequestGuard {
        private const output = false;
        public static function handle($method, $params,
$callback) {
            if ($_SERVER["REQUEST_METHOD"] === $method) {
                $exists = true;
                $actualParams =
array_keys($_GLOBALS["_$_method"]);
                sort($params);
                sort($actualParams);
                if ($params === $actualParams) {
                    $callback($_GLOBALS["_$_method"]);
                } else {
                    if (self::output) {
                        echo Reporter::errorMessage("Invalid
request #2");
                    }
                }
            } else {
                if (self::output) {
                    echo Reporter::errorMessage("Invalid request
#1");
                }
            }
        }
    }
}
```

### 3.1.3.3 Súbor /php/PictureParser.php

Ide o jednoduchú triedu ktorá obsahuje metódy ktoré navracajú absolútnu cestu ku konkrétnym obrázkom. Táto trieda akurát zbavuje nutnosť písať do reťazca celú trasu obrázku. Namiesto toho mi stačí vložiť ako argument názov obrázku a funkcia mi navráti jeho absolútnu cestu.

```
<?php
    class PictureParser {
        public static function profilePictureDir($picture) {
            return "../snvf/img/profiles/$picture";
        }
        public static function postPictureDir($picture) {
            return "../snvf/img/posts/$picture";
        }
    }
}
```

#### 3.1.3.4 Súbor /php/PictureUploader.php

Ide o triedu ktorá sa stará o manipuláciu servera s fotografiami. Trieda obsahuje jedno verejnú a jedno privátnu metódu. Verejná metóda **upload** po vložení obrázku tento obrázok následne komprimuje pomocou privátnej metódy **compressImage** a následne ho uloží, a navráti jeho názov a cestu kde bol uložený.

#### 3.1.3.5 Súbor /php/Reporter.php

Táto trieda je iba pomocnou triedou ktorá obsahuje 3 metódy. Tieto metódy slúžia pre kódovanie výstupu z backendu pre frontend. Metódy kódujú vložené parametre do formátu JSON. Tento formát je jedným z najpoužívanejších formátov keď ide o komunikáciu medzi serverom a klientom (backendom a frontendom). Táto trieda má metódy ktoré aj napriek tomu že majú jeden účel, tak ich oddeľuje obsah správ ktoré sa odošlu klientovi. V prípade že chceme klientovi poslať správu o tom že nastala chyba použijeme metódu **errorMessage**. Ak akcia prebehla úspešne tak **successMessage** a pokiaľ chceme navrátiť dáta tak **returnData**.

```
public static function successMessage($message) {  
    return json_encode(  
        array(  
            "type" => "success",  
            "message" => $message  
        )  
    );  
}
```

Vyššie je znázornený kód ktorý ukazuje ako jedna z týchto metód funguje. Metóda vyžaduje jeden argument ktorý sa vloží do asociatívneho poľa, ktoré je parametrom funkcie **json\_encode** ktorá z tohto poľa spraví reťazec.

#### 3.1.3.6 Súbor /php/LikeManager.php

Táto trieda je taktiež len pomocnou triedou ktorá uľahčuje postup označovania príspevku ako obľúbeným. O svojej jednoduchosti hovorí aj fakt že obsahuje iba dve metódy like a dislike ktoré označujú príspevok ako obľúbeným alebo ho odznačujú.

```
public function like($loggedUsername, $postId) {  
    if ($this -> db -> sendLike($loggedUsername, $postId, 1)) {  
        $this -> error = "";  
        return true;  
    } else {  
        $this -> error = $this -> db -> error;  
        return false;  
    }  
}
```

```
}
}
```

### 3.1.3.7 Zvyšné súbory nachádzajúce sa v adresári /php

Detailne popisovať zvyšné súbory je zbytočné keďže všetky plnia tú istú úlohu. Jedine čo sa líši sú dáta s ktorými pracuje a zámer s akým dáta používa.

Tabuľka 5 – Vysvetlivky pre jednotlivé súbory a ich zámer

Súbory/skripty	Funkcia skriptu
ChangePassword.php	Zabezpečuje zmenu hesla pre užívateľa
ChangeProfilePicture.php	Zabezpečuje zmenu profilovej fotografie užívateľa
Description.php	Zabezpečuje úpravu popisu užívateľa
GetPosts.php	Získava príspevky z databázy
GetProfile.php	Získava informácie užívateľa
GetUsers.php	Získava užívateľov
Like.php	Zabezpečuje označenie príspevku ako obľúbeným a taktiež jeho odznačenie
Login.php	Zabezpečuje proces prihlásenia pre užívateľa
Post.php	Zabezpečuje proces pridávania príspevku
Register.php	Zabezpečuje proces registrácie pre užívateľa

Všetky tieto uvedené súbory/skripty sú komunikačným mostom pre frontend a backend. Keď z frontendu príde požiadavka tak jedine na tieto súbory, odkiaľ následne putuje priama manipulácia s triedou **DatabaseManager**.

```
<?php
    require_once "Reporter.php";
    require_once "DatabaseManager.php";
    require_once "RequestGuard.php";
    RequestGuard::handle("POST", ["username", "description"],
function($post) {
    $dbm = new DatabaseManager();
    if ($dbm -> updateDescription($post["username"],
$post["description"])) {
        echo(Reporter::successMessage("ok"));
    } else {
        echo(Reporter::errorMessage($dbm -> error));
    }
    });
?>
```



Vyššie je znázornený kód ktorý prijme požiadavku ohľadom zmeny popisu užívateľa. Ak dostane všetky potrebné dáta na vykonanie tejto akcie tak následne zavolá príslušnú metódu triedy **DatabaseManager** ktorá túto požiadavku vykoná a následne navráti logickú hodnotu podľa toho či všetko prebehlo úspešne. V každom prípade sa na frontend vráti správa v JSON formáte. O formát tejto správy sa stará trieda **Reporter**.

## 4 Zadania

### 4.1 Demonštrácia pojmov Web a Internet

**Internet** (z angl. interconnected networks – prepojené siete; hovorovo: net, sieť) je verejne dostupný celosvetový systém vzájomne prepojených počítačových sietí, ktoré prenášajú dáta pomocou prepínania paketových dát s použitím štandardizovaného Internetového Protokolu (IP) a mnohých ďalších protokolov. Pozostáva z tisícok menších komerčných, nekomerčných, akademických, súkromných, vládnych a vojenských počítačových sietí. Slúži ako prenosové médium pre rôzne informácie a služby ako napr. elektronická pošta, chat a najmä systém vzájomne prepojených hypertextových dokumentov (webstránok) a webových aplikácií nazývaných World Wide Web (WWW). Internet nemá jednotnú centralizovanú správu implementácií technológií ani zásad prístupu a používania. Každá sieť, ktorá je jeho súčasťou, si stanovuje svoje vlastné zásady. Spoločné definície dvoch hlavných pilierov internetu: priestoru internetového protokolu adres (IP adresa) a systému názvov domén (DNS) sú riadené Internetovou spoločnosťou pre pridelovanie názvov a čísiel (Internet Corporation for Assigned Names and Numbers - ICANN).

Web je distribuovaný hypertextový internetový informačný systém, ktorý umožňuje univerzálny prístup k širokému univerzu dokumentov, ktoré obsahujú odkazy na iné miestne alebo vzdialené dokumenty.

Je to oficiálne označenie tej časti internetu, kde sa informácie nachádzajú vo forme webových stránok. Každý dokument má svoju špecifickú adresu – URL, vďaka ktorej môže byť nájdený a zobrazený v programoch nazývaných webový prehliadač. Webové stránky môžu obsahovať hypertextové odkazy. Vďaka týmto odkazom, sú dokumenty navzájom poprepájané a vytvárajú sieť.

### 4.2 Popis vývoja webu

World Wide Web vyvinul v rokoch 1989 – 1990 Timothy Berners-Lee a Robert Cailliau v Európskom laboratóriu pre časticovú fyziku CERN v Ženeve. Spolupráca vedcov vyžadovala spoločné využívanie dát. Tim Berners-Lee, inšpirovaný víziou Memexu a neskôr systémom Xanadu, sa rozhodol vytvoriť hypertextový systém, ktorý by bol integrálnou súčasťou internetu. 12. marca 1989 predložil návrh na vytvorenie takéhoto informačného systému. V októbri 1990 začal prácu na prvom klientovi, ktorý

by umožnil vytváranie, editovanie a prezeranie web-stránok. Aby bol klientský program čo najjednoduchší a nezávislý od platformy, vyvinul jazyk HTML ako podmnožinu komplikovaného jazyka SGML. Ďalej vytvoril HyperText Transfer Protocol (HTTP) ako súbor pravidiel, ktoré mali využívať počítače komunikujúce v sieti internet využívajúce hypertextové prepojenia nezávisle od ich umiestnenia. Pre tento účel navrhol Universal Document Identifier (UDI), ktorý sa neskôr nazýval Universal Resource Locator (URL) ako štandard, ktorý umožnil prideliť dokumentom v celom internete jedinečnú adresu. Už v polovici novembra 1989 uskutočnil prvú úspešnú komunikáciu medzi klientom a serverom HTTP. V decembri 1990 boli vytvorené základy World Wide Web (názov bol názvom pre html editor, ktorý vytvoril Tim na svojom počítači Next).

Okolo roku 2000 World Wide Web spojila viac ako miliardu webových stránok na svete. Fenomén internetu zaznamenal veľký rozmach najmä po zavedení tejto služby, ktorú vytvoril anglický počítačový expert Timothy Berners-Lee. Je atraktívna pre používateľov najmä poskytovaním možnosti „preskakovať“ z jedného dokumentu alebo obrazového, prípadne zvukového súboru na druhý, a to veľmi jednoducho, klikaním na jednotlivé linky.

### **4.3 Všeobecné nariadenie o ochrane údajov (GDPR)**

Všeobecné nariadenie o ochrane údajov (anglicky General Data Protection Regulation, skrátene GDPR), plným názvom Nariadenie Európskeho parlamentu a Rady (EÚ) 2016/679 z 27. apríla 2016 o ochrane fyzických osôb pri spracúvaní osobných údajov a o voľnom pohybe takýchto údajov, ktorým sa zrušuje smernica 95/46/ES (všeobecné nariadenie o ochrane údajov), je nariadenie Európskej únie, ktorého cieľom je výrazné zvýšenie ochrany osobných údajov občanov. V Úradnom vestníku Európskej únie bolo vyhlásené 27. apríla 2016. Nariadenia, v schválenom znení, sú priamo účinné pre každý členský štát EÚ. V slovenskom právnom poriadku je nariadenie premietnuté aj do zákona č. 18/2018 Z. z. o ochrane osobných údajov a o zmene a doplnení niektorých zákonov z 29. novembra 2017. Zákon 18/2018 Z. z. je účinný na Slovensku rovnako ako GDPR od 25. mája 2018.

Časť smernice sa zaoberá ochranou osobných údajov (doslovne "ochranou fyzických osôb pri spracúvaní osobných údajov") ale aj o voľnom pohybe údajov, preto sa názov prekladá všeobecnejšie ako Všeobecné nariadenie o ochrane údajov, ktoré mal

pôvodná smernica 95/46/ES. GDPR všeobecne určuje zásady, povinnosti, kódexy správania a ďalšie postupy najmä pre „prevádzkovateľov“ „sprostredkovateľov“ ale aj „tretie strany“, ktorými nie sú len firmy ale všeobecne fyzická alebo právnická osoba, orgán verejnej moci, agentúra alebo iný subjekt.

GDPR nie je jediná relevantná smernica EÚ. Problematiku osobných údajov v elektronickej komunikácii upravuje aj smernica staršia číslo 2002/58/ES. GDPR upravilo svoj vzťah voči smernici tak, že GDPR neukladá "dodatočné povinnosti, pokiaľ ide o spracúvanie v súvislosti s poskytovaním verejne dostupných elektronických komunikačných služieb vo verejných komunikačných sieťach v Únii, v prípadoch, keď podliehajú konkrétnym povinnostiam s rovnakým cieľom stanoveným v smernici.“ (Článok 95 GDPR)

Ak zahrnieme úplne všetky spôsoby získavania a používania osobných údajov na Slovensku, tak GDPR z veľkej časti túto oblasť pokrylo, nie však úplne. Kým väčšina spôsobov získavania a používania osobných údajov je zaraditeľná pod konkrétnu právnu úpravu, pri niektorých operáciách s osobnými údajmi nie je úplne jasné, ktorú časť pokrýva GDPR a ktorú nepokrýva, a ktorá je teda pokrytá slovenským zákonom o ochrane osobných údajov (18/2018 Z. z.) a inými právnymi predpismi, napríklad zákonom č. 351/2011 Z. z. o elektronických komunikáciách.

## 5 Záver

Tvorba tejto sociálnej siete bola pre mňa neuveriteľne veľkým prínosom. Naučil som sa pokročilo pracovať s jazykmi CSS a JavaScript, hlavne s knižnicou Vue.js, naučil som sa programovať v jazyku PHP ale čo je hlavné tak som sa naučil princípom ako písať efektívny a čistý kód čo je veľmi dôležitým aspektom pri tvorbe takéhoto projektu. Taktiež som pochopil ako sa tvoria projekty o takejto škále a to zahŕňa pomenovanie súborov, vytvorenie priečinkovej štruktúry ktorá je intuitívna a ako vytvárať použiteľné komponenty.

**Poznámka:** *Môže nastať prípad kedy znázornené zdrojové kódy sa nemusia presne rovnať tým ktoré sú skutočne v projekte. Dôvodom je že aj napriek tomu že je projekt hotový tak na ňom neustále pracujem a vylepšujem.*

## 6 Zoznam použitej literatúry

<https://css-tricks.com/how-many-css-properties-are-there/>  
[https://en.wikipedia.org/wiki/Cascading\\_Style\\_Sheets](https://en.wikipedia.org/wiki/Cascading_Style_Sheets)  
<https://sk.wikipedia.org/wiki/JavaScript>  
<http://htmlguru.cz/uvod-historie.html>  
[https://sk.wikipedia.org/wiki/Hypertextov%C3%BD\\_zna%C4%8Dkov%C3%BD\\_jazyk](https://sk.wikipedia.org/wiki/Hypertextov%C3%BD_zna%C4%8Dkov%C3%BD_jazyk)  
<https://en.wikipedia.org/wiki/Vue.js>  
<https://vuejs.org/v2/guide/>  
[https://sk.wikipedia.org/wiki/PHP\\_\(skriptovac%C3%AD\\_jazyk\)](https://sk.wikipedia.org/wiki/PHP_(skriptovac%C3%AD_jazyk))  
[https://sk.wikipedia.org/wiki/Structured\\_Query\\_Language](https://sk.wikipedia.org/wiki/Structured_Query_Language)  
[https://en.wikipedia.org/wiki/Visual\\_Studio\\_Code](https://en.wikipedia.org/wiki/Visual_Studio_Code)  
<https://en.wikipedia.org/wiki/XAMPP>  
[https://en.wikipedia.org/wiki/Microsoft\\_Edge](https://en.wikipedia.org/wiki/Microsoft_Edge)  
<https://en.wikipedia.org/wiki/PhpMyAdmin>  
<https://vuejs.org/v2/guide/components.html>  
<https://sk.wikipedia.org/wiki/Internet>  
[https://sk.wikipedia.org/wiki/World\\_Wide\\_Web](https://sk.wikipedia.org/wiki/World_Wide_Web)  
[https://sk.wikipedia.org/wiki/V%C5%A1eobecn%C3%A9\\_nariadenie\\_o\\_ochrane\\_%C3%BAdajov](https://sk.wikipedia.org/wiki/V%C5%A1eobecn%C3%A9_nariadenie_o_ochrane_%C3%BAdajov)