

STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ
KAROLA ADLERA 5, 841 02 BRATISLAVA



Inteligentné ovládanie miestnosti - časť: webová aplikácia

Tomáš Bordák

Bratislava 2022

STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ
KAROLA ADLERA 5, 841 02 BRATISLAVA



Praktická časť odbornej zložky maturitnej skúšky

Študijný odbor: 2675 M elektrotechnika

Oblasť prípravy: Počítačové systémy

Konzultant: Ing. Michal Mančík

Bratislava 2022

ZAMEŇTE STRANU ZA ZADANIE PRÁCE

ABSTRAKT

V dokumentácii sa dozvieme ako, prečo a čo bolo potrebné na vytvorenie webovej aplikácie na ovládanie inteligentnej miestnosti. Ukážeme si, ako môžeme dynamicky meniť obsah na stránke bez toho, aby sa musela obnovovať. Predstavíme si koncepty, ktoré sa dodržiavajú pri tvorení single-page aplikácii a tiež sa naučíme, akými spôsobmi sa dá obmedziť prístup užívateľa k obsahu na stránke.

Vďaka podrobne rozpísaným funkciám, dostaneme vedomosti a základné princípy na výrobu podobnej webovej aplikácie. Dozvieme sa o tom, ako aplikácia komunikuje s databázou alebo ako vytvoriť administrátorské rozhranie s vyššími privilégiami.

Neodmysliteľnou súčasťou nášho projektu sú frameworky, ktorých základom je JavaScript alebo CSS. Povieme si o rozdieloch medzi knižnicou a frameworkom. A taktiež, že aj normálny človek – nie programátor, dokáže takúto aplikáciu vymyslieť a vytvoriť s pomocou kurzov a dostatku času.

Kľúčové slová: odborná práca, dokument, maturita, JavaScript, CSS, webová aplikácia, framework, single-page

ABSTRACT

In the documentation, we will learn how, why and what was needed to create a web application to control a smart room. We'll show you how we can dynamically change content on a page without having to refresh it. We will introduce the concepts that are followed when creating a single-page application and we will also learn how to restrict user access to content on the site.

Thanks to detailed functions, we get the knowledge and basic principles for producing a similar web application. We will learn how the application communicates with the database or how to create an administrator interface with higher privileges.

Frameworks based on JavaScript or CSS are an integral part of our project. We will talk about the differences between the library and the framework. And also that even a normal person - not a programmer, can invent and create such an application with the help of courses and enough time.

Keywords: professional work, document, graduation, JavaScript, CSS, web application, framework, single-page

OBSAH

ZOZNAM SKRATIEK, ZNAČIEK	7
ÚVOD	8
1 NAVRHNITE A VYTVORTE WEBOVÚ APLIKÁCIU PRE OVLÁDANIE IOT PROJEKTU	9
.....	9
1.1 UDEMY	9
1.2 THE WEB DEVELOPER BOOTCAMP 2022.....	10
1.3 FRONT-END VS. BACK-END	11
1.4 FRONT-END – HTML, CSS A JAVASCRIPT	12
1.5 BULMA	13
1.6 VUE.JS.....	14
1.7 VISUAL STUDIO CODE.....	15
1.8 HTTP POŽIADAVKY (REQUESTY)	16
1.9 VUE CLI A SÚBOROVÁ ŠTRUKTÚRA PROJEKTU	17
2 VYTVORTE ROZHRAŇIE PRE NEPRIHLÁSENÉHO POUŽÍVATEĽA	20
2.1 PREZERANIE INFORMÁCIÍ O STAVE MIESTNOSTI BEZ MOŽNOSTI OVLÁDANIA	21
3 VYTVORTE ROZHRAŇIE PRE PRIHLÁSENÉHO POUŽÍVATEĽA, KTORÝ BUDE MÔCŤ:	28
3.1 OVLÁDAŤ SVETLÁ	28
3.2 REGULOVAŤ VENTILÁTOR A SVETLO.....	31
3.3 ZAPISOVAŤ INFORMÁCIE NA DISPLEJ	35
3.4 VIDIEŤ INFORMÁCIE O TEPLOTE, VLHKOSTI A TLAKU V MIESTNOSTI.....	37
3.5 VIDIEŤ GRAF TEPLoty ZA POSLEDNÉ DNI	42

3.6	NAHLIADNUŤ DO MIESTNOSTI POMOCOOU KAMERY	43
4	VYTVORTE ADMINISTRÁTORSKÉ ROZHRANIE, KTORÉ BUDE OBSAHOVAŤ:.....	45
4.1	MOŽNOSŤ PRIDÁVANIA, MAZANIA A ÚPRAVY UŽÍVATEĽOV ...	45
4.2	SŤAHOVAŤ NAMERANÉ DÁTA Z DATABÁZY CEZ WEBOVÉ ROZHRANIE	52
4.3	PRIHLÁSENIE ADMINISTRÁTORA	53
	ZÁVER	55
	ZOZNAM POUŽITEJ LITERATÚRY	56

ZOZNAM SKRATIEK, ZNAČIEK

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

IoT – Internet of Things

IT – Information Technology

CLI – Command-Line Interface

TCP/IP – Transmission Control Protocol/Internet Protocol

npm – node package manager

URI – Uniform Resource Identifier

URL – Uniform Resource Locator

UI – User Interface

API – Application Programming Interface

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

ID – Identification

JSON – JavaScript Object Notation

ÚVOD

Cieľom nášho projektu bolo vytvoriť inteligentné ovládanie miestnosti. Skladá sa z hardwaru, back-endu a front-endu. Aby všetko do seba zapadalo a fungovalo, komunikácia v tíme bola viac než kľúčová.

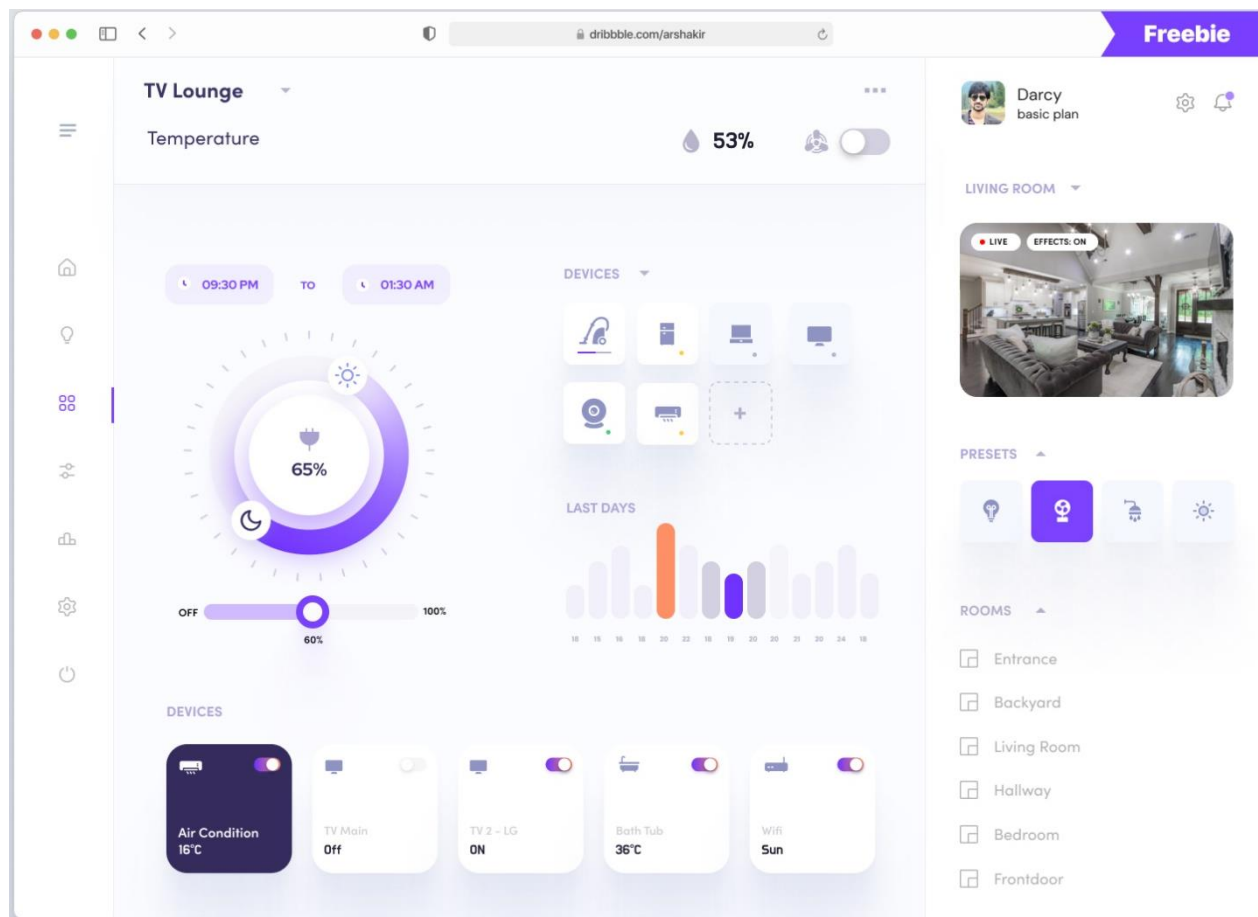
Pri vytváraní mojej časti – front-end, som využíval hlavne JavaScript framework vue.js, s ktorým bolo radosť programovať užívateľskú stranu projektu. O dizajn aplikácie sa hravo postaral ďalší framework, no teraz odvodený z jazyku CSS – Bulma. Dokumentácia k tomuto frameworku bola veľmi potrebná a využíval som ju stále.

Vytvoril som užívateľské prostredie, ktoré ponúka informácie aj pre neprihláseného aj pre prihláseného používateľa. Administrátorské prostredie, ktoré sťahuje namerané dáta z databázy alebo upravuje používateľov som implementoval ako posledné.

V tejto časti uvidíme obojstrannú komunikáciu s databázou založenou na HTTP požiadavkách. Ukážeme si, aké metódy týchto požiadaviek existujú alebo ako funguje API v našom projekte.

Uvidíme, ako vyrobiť užívateľské prostredie, ktoré sa nestratí ani na mobilnom zariadení ani na notebooku. Pri mobile boli odobrané určité funkcie aplikácie, aby mal používateľ pohodlnejšie ovládanie a nenastavil niečo inak než by chcel. Zmeny v užívateľskom rozhraní pri zariadeniach nastávajú z princípu používania týchto zariadení.

1 NAVRHNITE A VYTVOŘTE WEBOVÚ APLIKÁCIU PRE OVLÁDANIE IOT PROJEKTU



Obrázok 1 Inšpirácia na grafický dizajn aplikácie

Nápady na grafický dizajn aplikácie som hľadal na internete. Našiel som veľa podobných, no tento sa mi páčil najviac (viď Obrázok 1). Páčili sa mi hlavne jednoduché farby a minimalistický vzhľad. Pohľadom dole na Obrázku 1 vidíme dizajnový prvok kartičiek, ktoré svojim tieňom po obode núkajú oku pocit, ako keby sa vznášali. Tento prvok vo mne utkvel a použil som ho aj pri konečnom dizajne aplikácie.

1.1 UDEMY

Pred týmto projektom som vedel maximálne základy v programovacom jazyku C. Od nápadu, ktorý nám s mojimi dvomi spolužiakmi napadol, som nechcel upustiť

ani pre moje slabšie programovacie znalosti. Vyriešil som jednoducho, ba aj dokonca ešte lepšie než som predpokladal. Navštívil som stránku www.udemy.com, na ktorej v tom čase prebiehali zľavy až do -90%.

Na spomínanej stránke sa nachádzajú kurzy a vyučovacie dokumenty pre naozaj hocikoho. Vďaka širokému spektru kategórii nemá problém nájsť to čo potrebuje ani hudobník a ani programátor alebo kulturista.

Ja som však hľadal v kategórii vývoj aplikácií. Samozrejme že som našiel presne taký, ktorý mi vyhovoval a bolo tam všetko podstatné. Dokonca som nenašiel jeden, ale po splnení asi 50% z prvého som začal s druhým kurzom.

Pri vytváraní používateľských rozhraní, ktoré zastrešuje webová aplikácia, je určite potrebné vedieť: HTML, CSS a JavaScript. To je základ, ktorý som nadobudol prvým kurzom s názvom The Web Developer Bootcamp 2022. S dostatočnými znalosťami som sa potom ocitol v neprepádanom svete programovania spolu s ďalším kurzom Vue – The Complete Guide (incl. Router & Composition API), kde som sa naučil JavaScript framework vue.js, s ktorým bola táto aplikácia naprogramovaná.

1.2 THE WEB DEVELOPER BOOTCAMP 2022

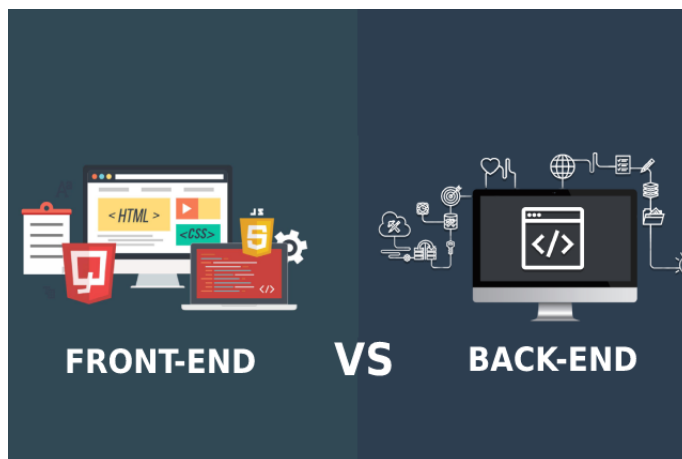
Ako som už spomínal, v tomto kurze som sa naučil povinný základ pre frontend. Celý kurz má 63.5 hodiny a je zložený z viac ako 40 dokumentov na stiahnutie, 60 praktických cvičení a za jeho plné absolvovanie dostanete certifikát.

Po zoznámení sa s plain JavaScriptom som vedel, že to nebude práve to v čom chcem programovať. Písať veľa funkcií na jednoduché veci, ktoré sa dookola opakujú a callback s asynchrónnymi funkciami, je jednoducho zdĺhavé a nelogické. Už dlhšie som registroval pojem framework, ktorý sa spája s programovacími jazykmi. A tak som jeden skúsil a teraz si neviem predstaviť aké by to bolo vytvoriť túto aplikáciu bez neho.

So záujmom o JavaScript framework sa mi naskytala otázka aký je rozdiel medzi týmto výrazom a knižnicou, ktorá je jednoduchšia na pochopenie a pozná ju aj začínajúci programátor v jazyku C. Knižnica je teda len súbor funkcií napísaných iným človekom, ktoré môžeme využívať kde je potreba. Nemení sa nič, len si takúto knižnicu stiahnem do súboru, v ktorom ju chcem využívať a funkciu z knižnice môžem zavolať. Čo sa ale mení, je syntax a celé programovanie v určitom jazyku vďaka frameworku. Ten so sebou prináša veľa výhod, ale taktiež sa musíme riadiť novými pravidlami frameworku, ktoré určuje. Ak sú logické, s frameworkom nám pôjde práca oveľa rýchlejšie. A tak tomu bolo aj pri vue.js (JavaScript framework) alebo Bulma (CSS framework). Je dôležité, aby spoločnosti, ktoré framework vyvíjajú, mali dokumentáciu. Veď predsa bez nej sa nepohneme.

Taktiež som sa v tomto kurze dozvedel rozdiel medzi front-endom a back-endom, ktoré sú 2 základné časti webovej aplikácie.

1.3 FRONT-END VS. BACK-END



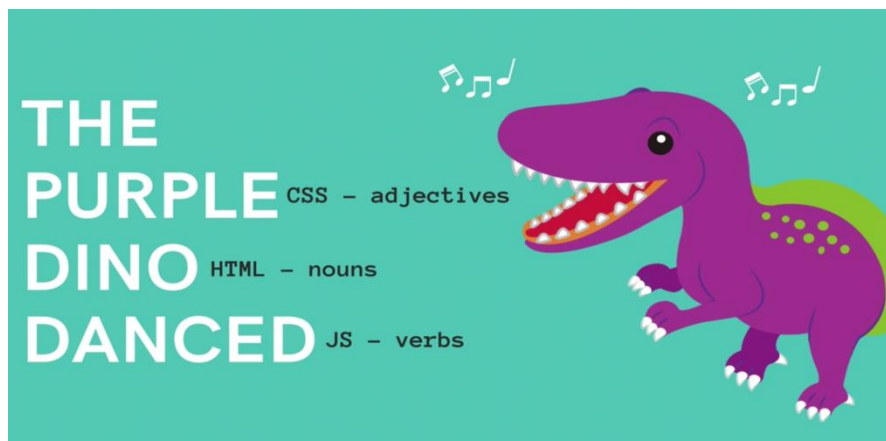
Obrázok 2 Front-end vs. back-end

Front-end je to čo používateľ vidí na webovej stránke a skladá sa z HTML, CSS a k nim sa pridáva aj JavaScript. Sú to rôzne obrázky, tlačidlá, videá, tabuľky alebo dotazníky.

Back-end je to čo používateľ na stránke nevidí. Je to to, čo pracuje niekde za front-endom. Front-end je vlastne taká spojka medzi používateľom a back-endom. Back-end má väčšinou podobu serveru, na ktorom je napríklad databáza alebo úložisko pre veci, ktoré používateľovi chceme cez front-end ukázať.

Ako príkladom z tohto projektu by mohlo byť toto. Používateľ vyplní polia na zaregistrovanie a kliknutím na tlačidlo zaregistruj sa, sa pošlú tieto dáta na back-end, ktorý ich spracuje, uloží a naspäť front-endu pošle prihlasovací token, ktorý používateľa identifikuje. Vďaka tomuto tokenu potom môžeme na front-ende zistiť, čo používateľovi na stránke ukázať. Ak je nejaký token prítomný, používateľ je prihlásený, naopak ak používateľ prihlásený nie je, token bude prázdny. Tento princíp si ešte popíšeme v ďalších kapitolách.

1.4 FRONT-END – HTML, CSS A JAVASCRIPT



Obrázok 3 HTML/CSS/JS

Ako spolu HTML, CSS a JavaScript pracujú aby vytvorili používateľské rozhranie? Ide o veľmi závislú činnosť všetkých troch jazykov. HTML je to čo na stránke vidím – tlačidlo, obrázok, text. CSS upravuje ako HTML vyzerá, napríklad upraví farbu tlačidla, veľkosť obrázku alebo písmo textu. JavaScript robí to, ako sa tieto určité časti stránky správajú, čo sa stane po kliknutí na tlačidlo, čo spraví obrázok, na ktorý kliknem alebo spraví text dynamickým na základe pohybu myši.

Ako si môžeme všimnúť na obrázku 3, ak by sme prirovnali HTML, CSS a JavaScript k slovným druhom. Tak HTML by boli podstatné mená, CSS prídavné mená a JavaScript slovesá.

V jednoduchosti:

- HTML – čo na stránke je – tlačidlo
- CSS – ako to vyzerá – tlačidlo je modré s bielym okrajom
- JavaScript – čo to robí – po kliknutí na tlačidlo sa na stránke zobrazí video

1.5 BULMA

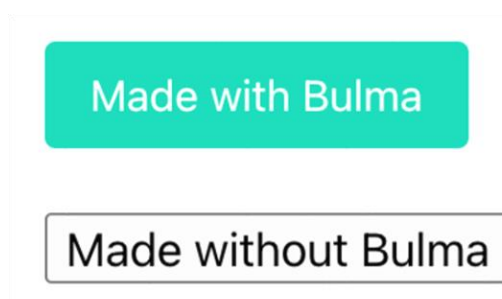
Je moderný CSS framework, s ktorým som si uľahčil alebo doslova obišiel zdĺhavé kódovanie v CSS. Umožňuje nám nastavovať farbu alebo veľkosť rovno na HTML tagu. Pracuje s flexboxom a na základe neho nám je umožnené vytvárať dynamické rozloženie objektov na stránke, ktoré sa mení pri zmenení zariadenia. Na dotykovom zariadení, ktoré zväčša bývajú menšie, napríklad mobil – tam bude iné rozloženie ako na notebooku, pretože rozhranie na ktoré sa používateľ pozerá je menšie.

Na obrázku 4 vidím, ako taký HTML tag vyzerá. Je to jednoduché tlačidlo s textom Send email, s farbou nastavenou rovno na tagu. Takéto takzvané „inline“ nastavovanie vlastností alebo tried HTML tagu je pri CSS nepraktické a využíva sa na to celý nový súbor, v ktorom určujem štýly. Toto ale mení framework Bulma, s ktorým viem do triedy (class) HTML tagu pridávať napríklad farbu alebo veľkosť. Vďaka rozumnému syntaxu tak nemusím hľadať akú bude mať tlačidlo farbu, ale vidím to rovno na HTML tagu button, ktoré toto tlačidlo vytvorí. Napríklad is-primary (viď Obrázok 4) určujem tlačidlu určitú zelenú farbu definovanú Bulmou.

```
<button class="button is-primary">Send email</button>
```

Obrázok 4 HTML tag

Na obrázku 5 môžeme pre porovnanie vidieť tlačidlo s použitím Bulmy a bez jej použitia. Prvé zelené tlačidlo je samozrejme ten prvý prípad.



Obrázok 5 Porovnanie

1.6 VUE.JS

Na výrobu front-endu nášho projektu som si vybral spomínaný JavaScript framework vue.js. Očaril ma svojou jednoduchosťou a celou myšlienkou single-page aplikácií. To sú, v jednoduchosti povedané, webové aplikácie, ktoré sa pri zmene obsahu stránky neobnovia a nenačítajú znova, ale menia len svoj obsah na jednej stránke. Vue.js dynamicky mení takzvané komponenty, ktoré sa na stránke zobrazujú. Buď pridáva alebo odoberá HTML kód viditeľný na stránke. Samozrejme toto správanie je veľmi využiteľné ak majú používatelia rôzne privilégia. Pretože im viem na základe určitých vecí odobrať alebo ukázať funkcionality aplikácie.

Je open-source. Je neplatený a dostane sa k nemu každý. S pomocou kurzu, ktorý som si zaplatil som nemal problém pochopiť princípy a využitie vue.js v praxi.

Obsahuje viacej špeciálnych prvkov, ktoré nám dovoľujú robiť určité veci. Jedným z nich je store. Ak potrebujeme prenášať uložené premenné v jednom súbore do druhého, napríklad na ďalšie funkcie, store je práve to čo by sme mali hľadať. Obrazne si môžeme predstaviť, že to je taký pomyselný balík premenných a funkcií, ktoré spolu súvisia a má predpísané princípy, ktoré by sa mali dodržiavať.

Store je jedným prvkom, ktorý rieši určitý problém. Druhým takýmto prvkom je router, ktorý ponúka svoju funkcionality na to, aby sme vedeli kde sa na stránke

nachádzame. Mení URL, na ktorej sa nachádzame. A taktiež obsahuje strážcu navigácie na stránke. S ním môžeme vďaka podmienok určovať kto uvidí určitý obsah a kto nie.

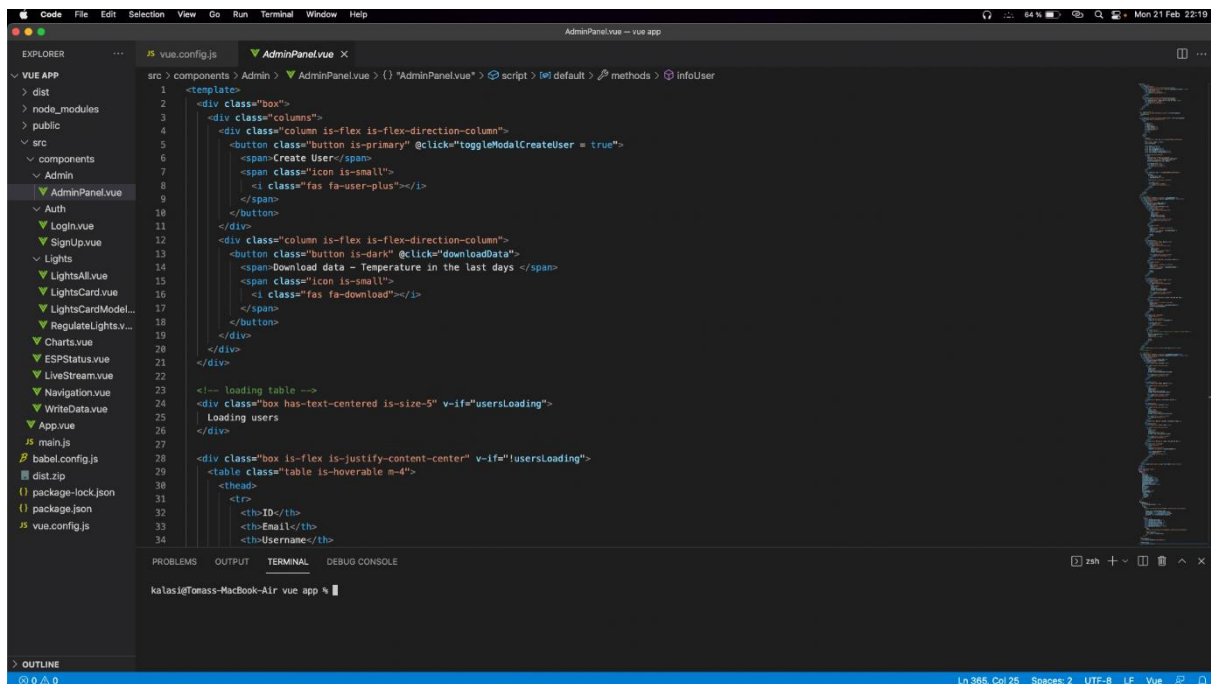
1.7 VISUAL STUDIO CODE

Mojím prostredím na programovanie sa stala Visual Studio Code aplikácia dostupná na <https://code.visualstudio.com/>. Je to kódový editor dostupný pre operačné systémy Windows, macOS aj Linux. Hneď po stiahnutí podporuje JavaScript, TypeScript a Node.js. Avšak táto aplikácia ponúka rozšírenia, ktoré sú zadarmo, a to aj napríklad pre vue.js, ktoré sa volá Vetur. Toto rozšírenie ovláda syntax vue.js a základnú funkcionality tohto frameworku. Okrem farebne oddelenej syntaxe, Vetur ponúka aj nápovede pri programovaní, ktoré sú užitočné nie len pre začínajúcich programátorov. Krátky text nám povie čo daná funkcia robí alebo aké argumenty má.

Vo Visual Studio Code sa stane programovanie hračkou. Rozhranie aplikácie si vieme rozdeliť na viaceré súbory, pri ktorých vieme súčasne programovať viacej vecí naraz. Taktiež veľkou výhodou je že si vieme otvoriť terminál, ktorý sa uloží do dolnej časti aplikácie. Terminál bol nevyhnutnou súčasťou programovania vo vue.js a tak si neviem predstaviť stále prepínať medzi súborom a terminálom.

Na obrázku 6 je zobrazené ako vyzerá moje prostredie v tejto aplikácii. Tmavý režim si neviem vynachváliť, pretože nie je nič horšie ako v tme programovať na svetlom podklade, ktorý rozožiera oči.

V ľavej časti aplikácie sa nachádza Explorer (viď Obrázok 6) – tam mám otvorený priečinok so všetkými súbormi. Najväčšiu časť tvorí okno s otvorenými súbormi a v dolnej časti sa nachádza terminálové okno. Kód je farebne triedený vďaka spomínanému rozšíreniu Vetur.



Obrázok 6 Visual Studio Code

1.8 HTTP POŽIADAVKY (REQUESTY)

Protokol HTTP definuje určitý počet metód na prácu s nejakým zdrojom dát, ktorým môže byť napríklad databáza. Tieto metódy sa využívajú pri požiadavkách, kde musím nastaviť čo si prajeme aby sa s dátami stalo. HTTP pracuje na aplikačnej vrstve a jeho požiadavka je rámec, ktorý sa skladá z riadku žiadosti (špecifikujem zdroj dát), hlavičky a tela.

Metódy, ktoré som používal:

- **GET** – od zdroja dát si dáta pýtam, telo je prázdne
- **POST** – zdroju dát posielam dáta, v tele obsahuje dáta, ktoré posielam
- **PATCH** – to isté ako POST s tým rozdielom, že PATCH mení len to čo je treba (ak sa už niečo z dát, ktoré posielam nachádza v zdroji dát, tak potom PATCH len vymení/dopíše to čo tam ešte nie je) – menej zaťažuje databázu ako POST

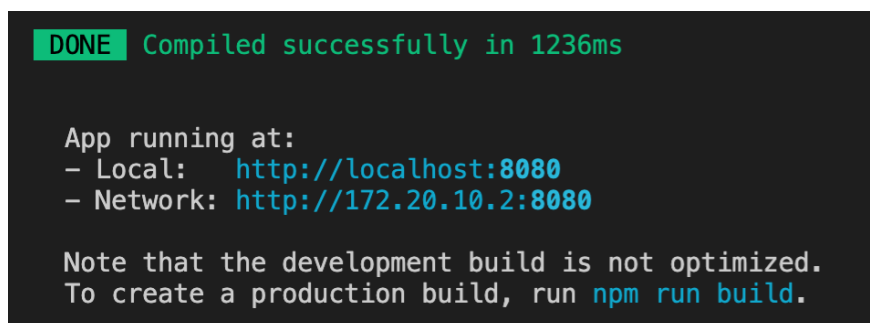
1.9 VUE CLI A SÚBOROVÁ ŠTRUKTÚRA PROJEKTU

Je vytvorený systém pre efektívne a rýchle narábanie s frameworkom vue.js. Nový projekt a celú jeho štruktúru so všetkými potrebnými súbormi vytvorím len jedným príkazom v termináli. Vue CLI (@vue/cli) je globálne inštalovateľný npm balík. Nainštalujeme ho pomocou príkazu, ktorý vložíme do terminálu na počítači: **npm install -g @vue/cli**.

Tento balík obsahuje príkazy:

- **vue create** - pre vytvorenie projektu (so všetkými potrebnými súbormi pre fungovanie vue.js aplikácie)
- **npm run serve** – pre spustenie development serveru (lokálny server spustený na našom počítači dostupný buď na localhost alebo našej lokálnej IP adrese na porte 8080) (viď Obrázok 7)
- **npm run build** – pre vytvorenie optimalizovaného súboru, ktorý je určený na publikáciu na internet

Prečo by sme ale pri programovaní tohto projektu potrebovali vytvoriť development server? Po otvorení HTML súboru dvojitým kliknutím naň, ho budeme vidieť v prehliadači zobrazený cez FILE protokol (file:// pred názvom súboru), ktorý je používaný na prácu s lokálnymi súbormi v počítači a využíva URI. Ale HTTP alebo HTTPS (https:// pred názvom domény) protokol využívajúci TCP/IP a pracuje na strane užívateľa - nie serveru, využíva URL. Takže na prácu s projektom, ktorý bude niekedy hostnutý na reálnom serveri dostupnom na internete potrebujem vytvoriť lokálny server, aby som videl presne to, čo uvidí používateľ aplikácie na internete.



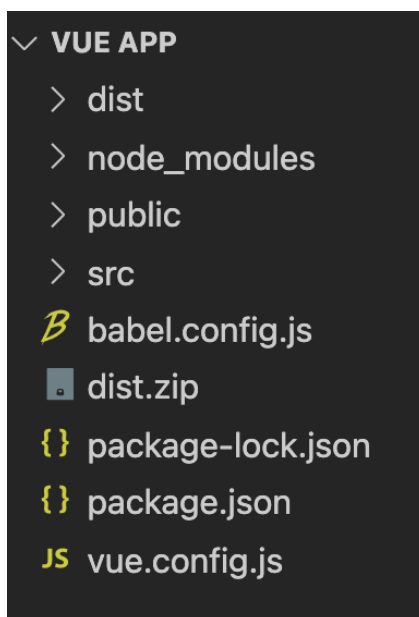
```
DONE Compiled successfully in 1236ms

App running at:
- Local:   http://localhost:8080
- Network: http://172.20.10.2:8080

Note that the development build is not optimized.
To create a production build, run npm run build.
```

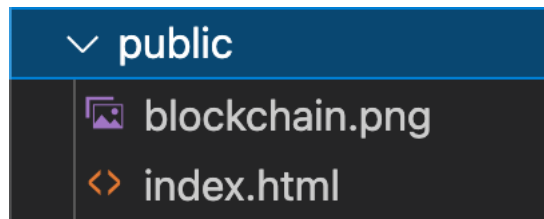
Obrázok 7 Terminálové okno – vytvorenie lokálneho serveru

Na obrázku 7 vidíme výsledok príkazu `npm run serve`, ktorý sme zadali v terminálovom okne, keď sme sa nachádzali v našom projekte. Vytvoril nám 2 linky, cez ktoré je dostupná naša aplikácia na lokálnom serveri. Na tieto linky sa dá kliknúť a po ich otvorení sa nám v prehliadači zobrazí naša webová aplikácia. Taktiež na tomto obrázku dole vidíme správu, ktorá hovorí o tom, že v prípade ak budeme chcieť vytvoriť verziu projektu určenú do produkcie, pomôže nám s tým príkaz `npm run build`. Vytvorí nám súbor `dist` (viď Obrázok 10), ktorý potom posielame back-endu pre publikáciu na internet.

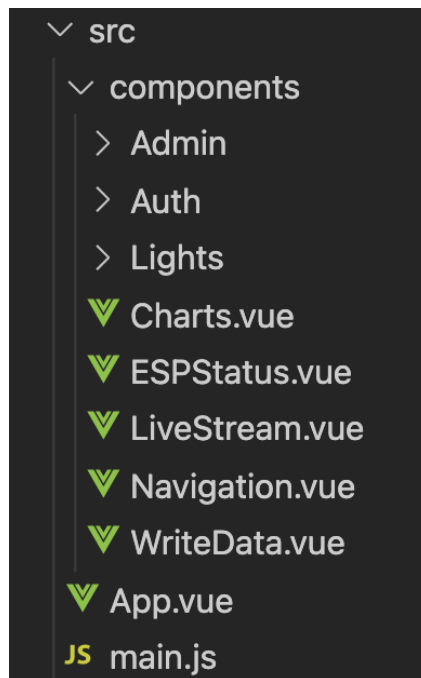


*Obrázok 8 Štruktúra súborov
webovej aplikácie*

V `node_modules` (viď Obrázok 8) nájdeme všetky balíky používané v projekte. Súbor `public`, ako už názov napovedá, je určený pre „verejnosť“ čiže jeho obsahom nie je logika projektu, tam sa nachádza HTML súbor (`index.html`), ktorý je vidieť na internete po otvorení našej webovej aplikácie (viď Obrázok 9). Obsah tohto HTML súboru je dynamicky menený pomocou komponentov, ktoré sa nachádzajú v súbore `src` (source) – zdrojový kód (viď Obrázok 10). Babel je JavaScript kompilátor a `babel.config.js` je jeho konfigurácia. V súboroch `package.json` a `package-lock.json` sa tam nachádzajú preto, aby nám viedli záznam o použitých npm balíkoch a ich verziách.

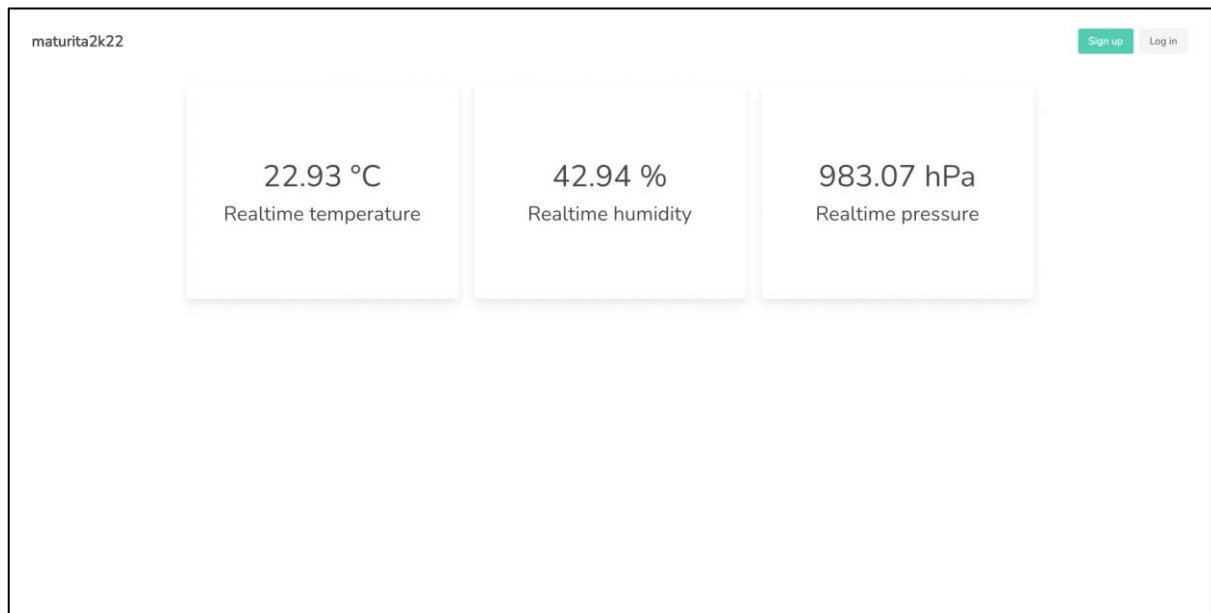


Obrázok 9 Súbor public



Obrázok 10 Súbor src

2 VYTVOŘTE ROZHŘANIE PRE NEPRIHLÁSENÉHO POUŽÍVATEĽA



Obrázok 11 UI neprihláseného používateľa

Na obrázku 11 je vidieť to čo uvidí používateľ, v momente otvorenia našej webovej aplikácie na internete. Používateľ nie je logicky ešte prihlásený a tak vidí len obmedzené informácie. Môžeme tu tiež vidieť použitý dizajnový prvok, o ktorom som rozprával v bode 1 obsahu tejto dokumentácie.

Toto používateľské rozhranie obsahuje kartičky, ktoré svojim tieňom dostávajú vzhľad ako keby sa vznášali. Taktiež tam môžeme vidieť názov stránky v ľavom hornom rohu a v tom pravom dve tlačidlá, jedno s ktorým sa zaregistrujem a druhé na prihlásenie. Tieto tlačidlá majú rôznu farbu. Spravil som to tak z toho dôvodu, aby si používateľ pri prezeraní stránky všimol tlačidlo pre zaregistrovanie.

Pre celý projekt som si vybral používať práve konkrétnu farbu zelenej `hsl(171, 100%, 41%)`, ktorá sa označuje v Bulme ako `is-primary`. Všetky ostatné farby použité v projekte sú zadefinované v Bulma dokumentácii. V nej nájdeme napríklad takto rozpísané farby pre tlačidlá (viď Obrázok 12).



Obrázok 12 Bulma dokumentácia – farby tlačidiel

2.1 PREZERANIE INFORMÁCIÍ O STAVE MIESTNOSTI BEZ MOŽNOSTI OVLÁDANIA

Neprihlásený užívateľ môže vidieť len teplotu, vlhkosť a tlak vzduchu v miestnosti. Tieto hodnoty sa dynamicky menia vďaka spôsobu, ktorý vue.js umožňuje. Stránka sa pri zmene čísel neobnovuje, len sa menia čísla. JavaScript framework vue.js dynamicky vymieňa tieto hodnoty.

Na Obrázku 13, je zdrojový kód k tomu čo vidí neprihlásený užívateľ. \$store.getters.token nám vracia token prihláseného používateľa, ak používateľ ešte prihlásený nie je, tak \$store.getters.token bude prázdny. Podľa tohto zistujeme čo používateľovi na stránke ukážeme. Takže, ak používateľ nie je prihlásený a tým pádom nám \$store.getters.token vráti takzvaný prázdny string, tak potom používateľovi ukážeme tento kód (viď Obrázok 13). \$store.getters.token je funkcia so špeciálnou syntaxou, ktorú vue.js definuje.

Na to, aby sme spravili také kartičky a v tomto rozložení (Obrázok 12) sme použili Bulmu. Pomocou triedy columns, na prvom <div> tagu určujeme že ideme robiť riadok. V tomto <div> tagu sa nachádzajú ďalšie <div> tagy, ktoré majú triedy

column čo spraví stĺpec. Takže celkovo budeme mať jeden riadok a tri stĺpce (viď Obrázok 13). Kartyčky vytvorené máme a teraz do nich musíme vložiť hlavne číslo a doplnujúci text. Vo vue.js existuje špeciálna syntax, ktorá nám umožní dávať JavaScript kód do HTML kódu resp. tam, kde by mal byť text vložíme JavaScriptovú funkciu. V prvom prípade kedy potrebujeme dostať do kartyčky teplotu, to urobíme nasledovne. `$store.getters.temperatureNumber` mi vráti číslo – aktuálnu teplotu z databázy. Toto číslo dám do `<p>` tagu a zaň doplním veličinu – °C, pridám vlastnosti, aby text mal určitú veľkosť (`is-size-1`) a aby bol text vycentrovaný (`has-text-centered`). Následne pod číslo s veličinou pomocou ďalšieho `<p>` tagu napíšem, že sa jedná o aktuálnu nameranú hodnotu, ktorú v našom prípade nazývam teplotu nameranú v reálnom čase.

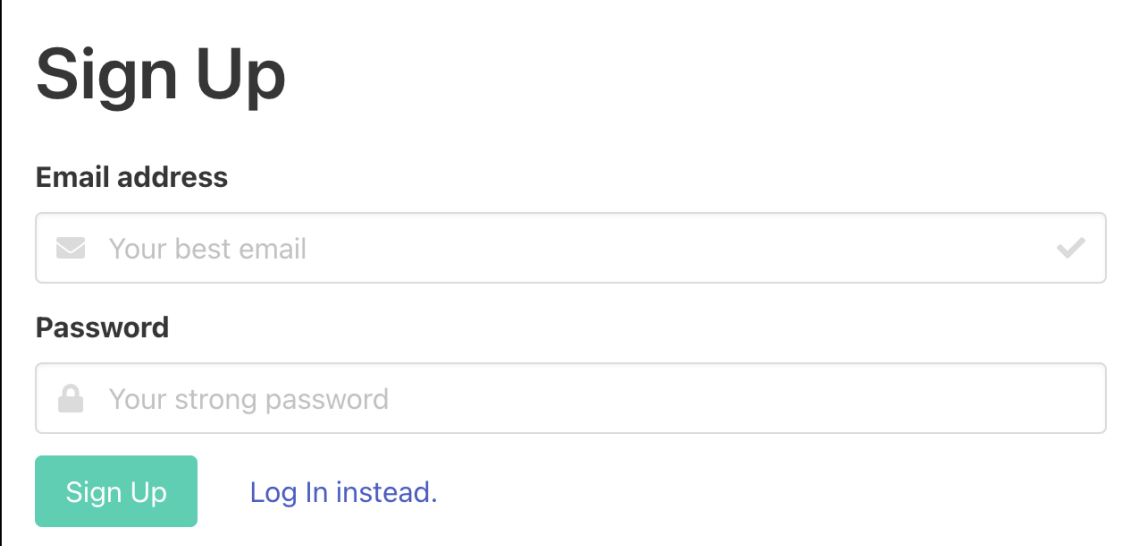
Prvú kartyčku máme, ostatné spravíme tak isto. `$store.getters.humidityNumber` ako už názov funkcie (názov funkcie je len `humidityNumber`) napovedá, mi vracia vlhkosť ako číslo. Vlhkosť udávame v percentách - %. A podobne `$store.getters.pressureNumber` mi vracia číslo označujúce tlak vzduchu v miestnosti. Tlak vzduchu udávame v hektopascaloch - hPa.

```
<!-- auth === "" -->
<div v-if="$store.getters.token === ''" class="columns">
  <div class="column">
    <div class="box padMe">
      <p class="is-size-1 has-text-centered">
        {{ $store.getters.temperatureNumber }} °C
      </p>
      <p class="is-size-3 has-text-centered">Realtime temperature</p>
    </div>
  </div>
  <div class="column">
    <div class="box padMe">
      <p class="is-size-1 has-text-centered">
        {{ $store.getters.humidityNumber }} %
      </p>
      <p class="is-size-3 has-text-centered">Realtime humidity</p>
    </div>
  </div>
  <div class="column">
    <div class="box padMe">
      <p class="is-size-1 has-text-centered">
        {{ $store.getters.pressureNumber }} hPa
      </p>
      <p class="is-size-3 has-text-centered">Realtime pressure</p>
    </div>
  </div>
</div>
```

Obrázok 13 /home – neprihlásený používateľ

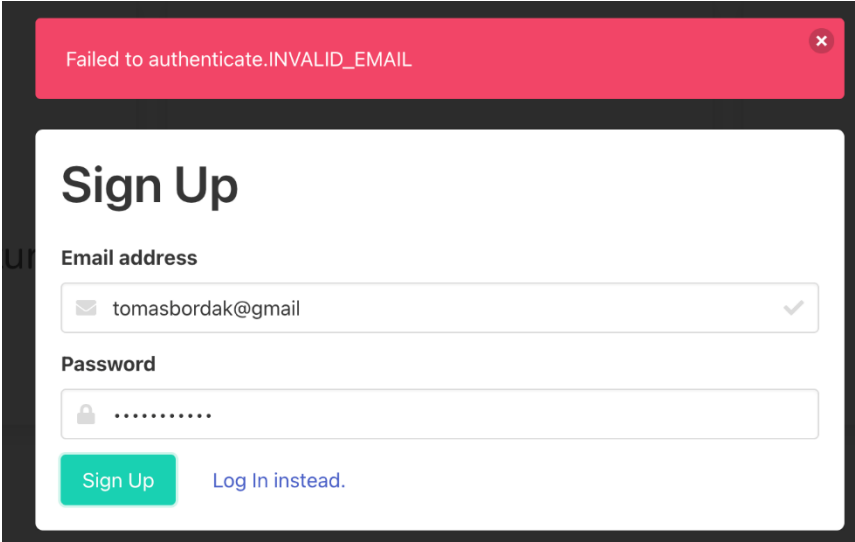
Po stlačení zeleného tlačidla Sign up v hornom pravom rohu webovej aplikácie na nás „vybehne“ okno s veľkým nápisom Sign Up (Obrázok 14) . Tu sa používateľ zaregistruje pomocou emailovej adresy a hesla. Stlačením zeleného tlačidla Sign Up front-end funkcia pošle back-endu zadaný email a heslo.

Po zadaní zlej emailovej adresy, z pohľadu správnosti emailovej adresy, nám aplikácia nad oknom Sign Up ukáže varovnú správu s dôvodom prečo sa nepodarilo registrovať. Tento dôvod je odoslaný z databázy (viď Obrázok 15).



The image shows a 'Sign Up' modal window. It has a title 'Sign Up' in large bold font. Below it are two input fields: 'Email address' with a placeholder 'Your best email' and a checkmark icon, and 'Password' with a placeholder 'Your strong password' and a lock icon. At the bottom, there is a green 'Sign Up' button and a blue link 'Log In instead.'

Obrázok 14 Sign up modal

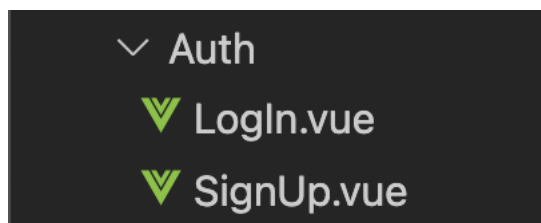


The image shows the same 'Sign Up' modal window as in the previous image, but with an error message displayed at the top. The error message is 'Failed to authenticate.INVALID_EMAIL' in white text on a red background. The input fields now contain the email 'tomasbordak@gmail.com' and a masked password '.....'. The 'Sign Up' button is still green, and the 'Log In instead.' link is still blue.

Obrázok 15 Zlý email

Do poľa s emailovou adresou sme zadali email tomasbordak@gmail, ktorý je samozrejme zlý, pretože mu chýba .com na konci. Takže sme od databázy dostali správu INVALID_EMAIL.

Logika pre registráciu a prihlasovanie sa nachádza v súbore Auth (viď Obrázok 10). V súbore Auth sa nachádzajú dva .vue súbory. To sú také, ktoré nám dovoľujú používať template (HTML), script (vue.js) a style (CSS) spolu v jednom súbore. V normálnom súbore napríklad index.html používame len HTML alebo v app.css používame len CSS. Template, script aj style sú HTML tagy, ktoré nám vyznačujú oblasti, v ktorých využívame k nim viazané jazyky.



Obrázok 16 Súbor Auth

Na obrázku 17 vidíme ako takýto súbor .vue vyzerá a že naozaj obsahuje aj HTML aj CSS aj JavaScript. V prvom tagu <script>, ktorý sa ukončuje ako každý iný tag </script>, sa nachádza špeciálna syntax vue.js (JavaScript). V druhom tagu, <template> sa nachádza HTML kód, v ktorom používam údaje definované v <script></script> pomocou vue.js. A posledný tag - <style>, v ktorom sa nachádza CSS a určuje mi farbu a typ písma pre spomínané dáta.

```
<script>
export default {
  data() {
    return {
      greeting: 'Hello World!'
    }
  }
}
</script>

<template>
  <p class="greeting">{{ greeting }}</p>
</template>

<style>
.greeting {
  color: red;
  font-weight: bold;
}
</style>
```

Obrázok 17 Súbor .vue

A teraz na tú logiku a ako to celé funguje. V súbore SignUp.vue (viď Obrázok 16) sa nachádza logika pre registráciu a v súbore LogIn.vue je logika pre prihlásenie. Spôsob akým sa registrujem alebo prihlasujem je na základe komunikácii medzi aplikáciou a back-endom podobný, takže si ukážeme len ako prebieha registrácia.

V SignUp.vue sa nachádza <template> tag, v ktorom je to, ako Sign Up okno vyzerá a čo v ňom je (viď Obrázok 18). Je to celé v podstate Bulma syntax pre dotazníky doplnená o vue.js. Na Obrázku 18 sa nachádza HTML kód použitý pre pole emailovej adresy.

Tagy:

- <div class="field"> - združuje kód poľa
- <label> - názov poľa
- <input> - samotné pole na vyplnenie údajov
- <i> - ikona

Pomocou HTML vlastnosti v-model (pridanou vue.js) vieme uložiť vstup používateľa do premennej v JavaScripte (viď Obrázok 18). Vďaka .trim metóde na v-model dosiahneme odstránenie všetkých nepotrebných medzier v poli (oseká string od medzier pred a po texte).

```
<div class="field">
  <label class="label">Email address</label>
  <div class="control has-icons-left has-icons-right">
    <input
      class="input"
      :class="{ 'is-danger': !emailIsValid }"
      type="email"
      placeholder="Your best email"
      v-model.trim="email"
    />
    <span class="icon is-small is-left">
      <i class="fas fa-envelope"></i>
    </span>
    <span class="icon is-small is-right">
      <i class="fas fa-check"></i>
    </span>
  </div>
  <p v-if="!emailIsValid" class="help is-danger">
    Check the email please.
  </p>
</div>
```

Obrázok 18 Pole na vyplnenie emailovej adresy

Na Obrázku 19 vidíme data property, ktorá sa nachádza v <script> tagu v súbory SignUp.vue. Obsahuje premenné, ktoré potrebujem na to, aby som do nich vedel uložiť vstup používateľa a potom ho poslal do databázy.

```
data() {  
  return {  
    email: '',  
    password: '',  
    emailIsValid: true,  
    passwordIsValid: true,  
    error: 'Failed to authenticate.',  
    successSignUp: 'Sign up succesful.',  
  };  
},
```

Obrázok 19 data property

Pomocou volania funkcie, ktorému dávam spomínané premenné, ich pošlem do tejto funkcie. Táto funkcia sa zavolá keď používateľ klikne na tlačidlo Sign Up (viď Obrázok 14). Funkcia signUp prijíma email a heslo zadané používateľom. Zavolám ju takto (viď Obrázok 20).

```
this.$store.dispatch('signUp', {  
  email: this.email,  
  password: this.password,  
});
```

Obrázok 20 Volanie funkcie signUp

Funkcia signUp (viď Obrázok 21) sa nachádza v súbore main.js vo vue.js prvku store. A funguje takto:

- jej zavolaním prijme email a heslo zadané používateľom
- odošle tieto dáta na databázu a žiada, aby databáza naspat' poslala prihlasovací token
- po úspešnej komunikácii medzi front-endom a databázou nastavíme ďalšie premenné potrebné na chod aplikácie na dáta, ktoré sme dostali od databázy

- po neúspešnej komunikácii sa nám na stránke zobrazí chybová hláška spolu s odpoveďou od databázy prečo sa nepodarilo zaregistrovanie používateľa

```

signUp(context, payload) {
  axios
    .post(
      'https://identitytoolkit.googleapis.com/v1/accounts:signIn?key=AIzaSyA4H37PEw-gpUki2jU2r5WP8IadIh-r7ws',
      {
        email: payload.email,
        password: payload.password,
        returnSecureToken: true,
      }
    )
    .then((response) => {
      console.log(response.data);
      context.commit("showSuccess");
      context.commit("setUser", {
        token: response.data.idToken,
        userId: response.data.localId,
        tokenExpiration: response.data.expiresIn
      });
      context.commit("toggleModalSignUp")
      context.commit("showSuccess")
    })
    .catch((error) => {
      context.commit("errorMessage", {
        message: error.response.data.error.message
      });
      context.commit("showError");
      console.log(error.response.data.error.message);
    });
});

```

Obrázok 21 Funkcia signUp

Vďaka npm balíku axios viem vytvoriť HTTP požiadavku jednoducho. Napríklad pre metódu POST použitú aj vo funkcii signUp to vo všeobecnosti vyzerá takto.

- axios.post([zdroj dát], [obsah požiadavky])

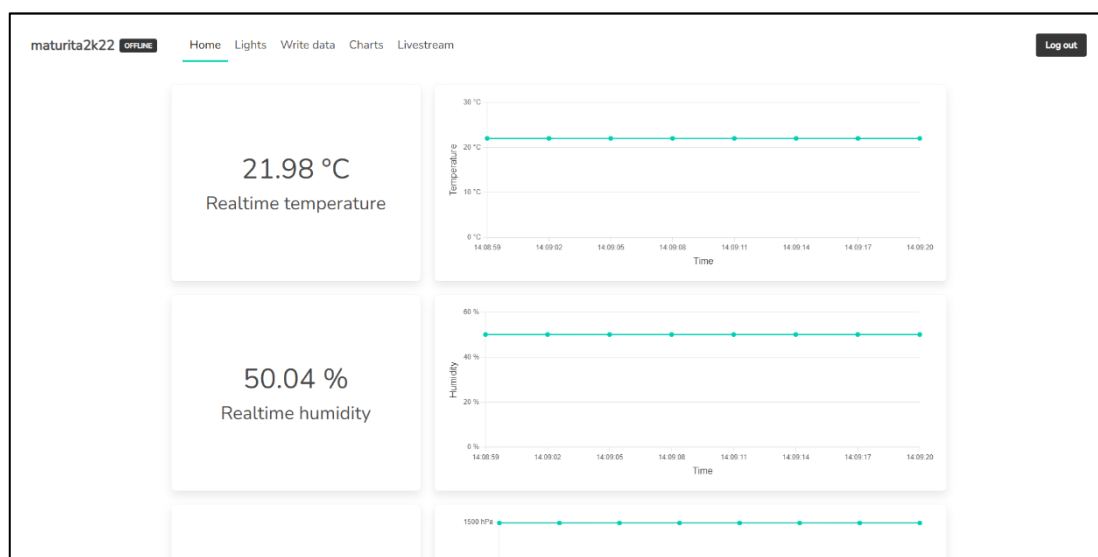
Zdrojom dát bude v tomto prípade databáza a obsahom požiadavky bude zadaný email a heslo používateľom. Metódou .then určujem čo sa má stať po vykonaní požiadavky a metódou .catch určujem čo sa má stať po vykonaní požiadavky, pri ktorej nastala chyba.

Od databázy dostanem prihlasovací token, identifikačné číslo a čas dokedy prihlasovací token platí. Po uplynutí tohto času používateľa automaticky odhlási.

3 VYTVORTE ROZHRANIE PRE PRIHLÁSENÉHO POUŽÍVATEĽA, KTORÝ BUDE MÔČŤ:

3.1 OVLÁDAŤ SVETLÁ

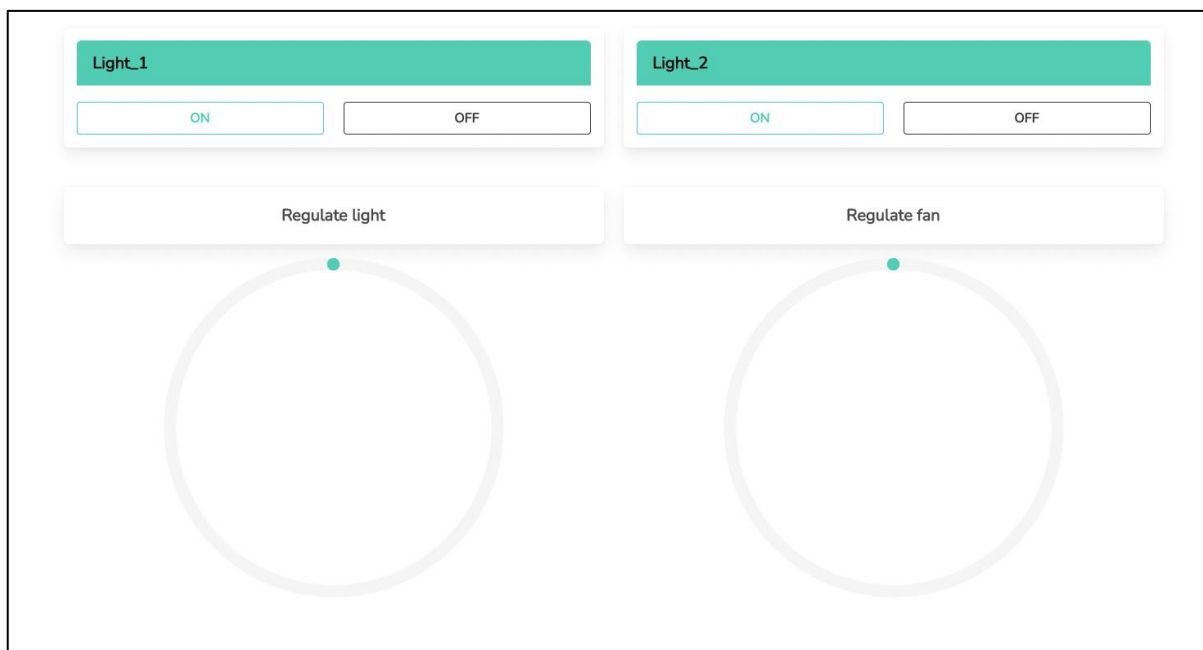
Sme zaregistrovaní a pred očami sa nám zmenila stránka. Podotýkam bez toho, aby sa musela obnoviť. Oproti stavu, kde používateľ nebol zaregistrovaný alebo prihlásený sa nám pridalo, ale aj ubudlo pár vecí.



Obrázok 22 UI prihláseného používateľa

V navigácii stránky (viď Obrázok 22) nám pribudli tlačidlá na vymieňanie komponentov. Ak kliknem na Home, tak mi to ukáže iný komponent než keď kliknem na Lights. Taktiež tam je esp status – stav nášho hardwaru, ktorý používame. Ak je zapnutý – stav bude online a bude to zelené tlačidlo, ak bude vypnuté čiže stav je offline, tak potom tlačidlo bude čierne presne ako je na Obrázku 22 pri texte maturita2k22.

Ovládať svetlá prihlásený používateľ môže ak sa preklikne cez tlačidlo Lights v navigácii.



Obrázok 23 Lights

Po kliknutí na tlačidlo Lights používateľ na obrazovke uvidí spôsoby akým môže miestnosť ovládať. Môže vypínať a zapínať svetlo 1 a 2. A regulovať môže ďalšie svetlo a ventilátor.

Na stránke vidí vlastne komponent LightsAll.vue, ktorý združuje niekoľko na ďalších komponentov. Tieto všetky komponenty sa nachádzajú v priechniku Lights (viď Obrázok 10).

```
<template>
  <article class="panel is-primary p-4">
    <p class="panel-heading has-text-weight-bold has-text-black">
      {{ bulb }}
    </p>
    <div class="columns is-mobile mt-2 is-6">
      <div class="column">
        <button
          class="
            button
            is-primary is-outlined is-fullwidth
            has-text-weight-bold
          "
          @click="patchData('ON')"
        >
          ON
        </button>
      </div>
      <div class="column">
        <button
          class="button is-outlined is-fullwidth is-dark has-text-weight-bold"
          @click="patchData('OFF')"
        >
          OFF
        </button>
      </div>
    </div>
  </article>
</template>
```

Obrázok 24 Template pre zapínanie a vypínanie svetiel

Na obrázku 24 vidíme template komponentu pre vypínanie a zapínanie svetiel. Tento komponent je napísaný pre jedno svetlo a tak v komponente LightsAll.vue je použitý dvakrát lebo ovládame dve svetlá.

Pomocou @click vlastnosti na prvom tlačidle dosiahnem to, že po kliknutí na tlačidlo ON sa zavolá funkcia patchData(„ON“), ktorá posieľa na databázu informáciu že chce používateľ zapnúť svetlo. Funkcia patchData vyzerá takto (viď Obrázok 25).

```
<script>
import axios from 'axios';

export default {
  props: ['bulb'],

  methods: {
    patchData(param) {
      axios.patch(
        `https://vue-http-908f5-default-rtdb.europe-west1.firebaseio.com/Lights/${this.bulb}.json?auth=` +
        this.$store.getters.token,
        {
          isLight: param,
        }
      );
    },
  },
};
</script>

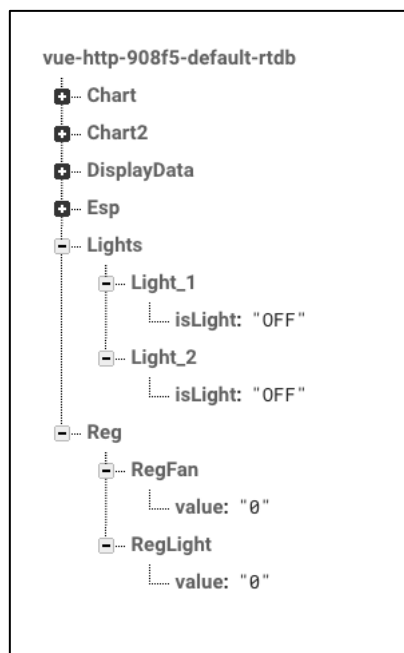
<style scoped>
@import url('https://fonts.googleapis.com/css2?family=Nunito&display=swap');
p,
button {
  font-family: 'Nunito', sans-serif;
}
</style>
```

Obrázok 25 Funkcia patchData

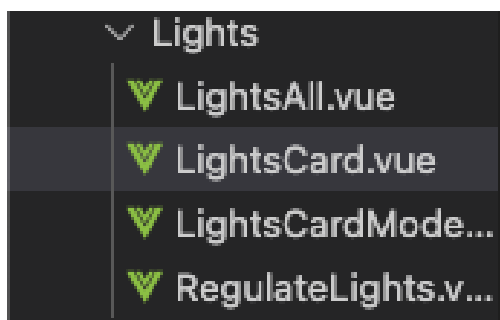
Funkcia patchData využíva npm balík axios pre vytvorenie HTTP požiadavky. Zdrojom dát v tejto požiadavke je presné miesto na databáze, ku ktorému pridávam prihlasovací token používateľa. Bez neho by databáza odmietla komunikovať na základe pravidiel nastavených databázou. Obsahom požiadavky je argument funkcie, ktorý funkcia prijíma pri jej volaní a to buď ON alebo OFF na základe toho, na ktoré tlačidlo používateľ klikol.

Pomocou prvého riadku v <script> tagu (viď Obrázok 25) si do komponentu prenosieme balík axios. Bez tohto by nám balík axios nefungoval v komponente.

Na Obrázku 26 vidíme štruktúru databázy použité v našom projekte.



Obrázok 26 Databáza – Lights a Reg



Obrázok 27 Súbor Lights

Na Obrázku 27 vidíme komponenty použité pre časť stránky dostupnej po kliknutí na tlačidlo Lights.

3.2 REGULOVAŤ VENTILÁTOR A SVETLO

Túto funkciu stránky nájdeme v komponente RegulateLights.vue (viď Obrázok 27). Zase ako každý iný súbor .vue obsahuje <template>, <script> a <style>.


```

<template>
  <div class="columns">
    <div class="column is-flex is-flex-direction-column">
      <div class="box has-text-centered has-text-weight-bold is-size-5">
        <p>Regulate light</p>
      </div>
      <slider
        class="is-flex is-align-self-center"
        v-model="valueLight"
        orientation="circular"
        color="hsl(171, 100%, 41%)"
        trackColor="hsl(0, 0%, 96%)"
        width="25rem"
        :height="15"
        :tooltip="true"
        max="255"
      ></slider>
    </div>
    <div class="column is-flex is-flex-direction-column">
      <div class="box has-text-centered has-text-weight-bold is-size-5">
        <p>Regulate fan</p>
      </div>
      <slider
        class="is-flex is-align-self-center"
        v-model="valueFan"
        orientation="circular"
        color="hsl(171, 100%, 41%)"
        trackColor="hsl(0, 0%, 96%)"
        width="25rem"
        :height="15"
        :tooltip="true"
        max="255"
      ></slider>
    </div>
  </div>
</template>

```

Obrázok 28 Template RegulateLights.vue

Na Obrázku 28 je <template> nášho komponentu, ktorý zabezpečuje reguláciu svetla a ventilátoru. Na túto reguláciu sú využité takzvané „slidery“. Tie sú z npm balíka vue3-slider. Pomocou nastavení tohto slidera meníme jeho tvar, farbu, šírku alebo dĺžku.

Zaujímavé ale je ako toto regulovanie funguje. Nefunguje totiž na klik, ale funguje na zmenu premennej. Táto metóda urobí niečo ak sa zmení premenná je urobená pomocou takzvanej watch property dostupnej vo vue.js. Pri zmene hodnoty regulácie automaticky zavolá funkciu patchData (iná než v predošlom komponente). Musí byť však dodržané meno premennej, ktorú má vue.js „pozerať“. Vo watch (viď Obrázok 29) sú dve funkcie presne pomenované ako premenné, ktoré sledujeme a pri zmene tejto premennej sa zavolá funkcia pomenovaná po tej premennej, ktorej obsahom je volanie funkcie patchData s argumentom podľa toho, ktorá premenná sa

zmení. Tento spôsob volania patchData je presne ten istý ako sme si už ukázali v predošlom komponente.

Premenné vo vue.js zapisujeme do data property – funkcia, ktorá nám vracia premenné. Ďalej pomocou mounted property, čo je ďalšia funkcia, sa dostávame ku takzvanému životnému cyklu vue.js aplikácie. Mounted nám umožňuje vykonať veci, ktoré potrebujeme mať vybavené pri načítaní komponentu na stránku. V tomto prípade popritom ako sa komponent dáva na stránku, my si aktualizujeme hodnotu regulácie z databázy, aby nám sedeli hodnoty v databáze a aj na stránke.

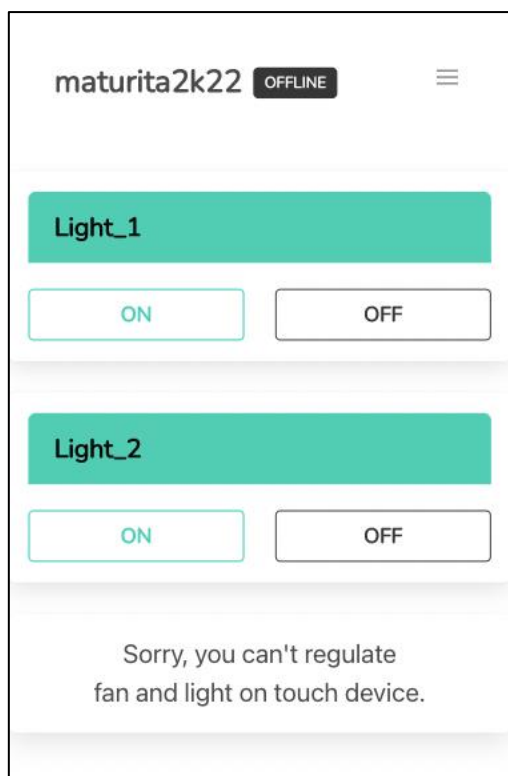
```
<script>
import slider from 'vue3-slider';
import axios from 'axios';
export default {
  components: {
    slider,
  },
  data() {
    return {
      valueLight: 0,
      valueFan: 0,
    };
  },
  watch: {
    valueLight() {
      this.patchData('light');
    },
    valueFan() {
      this.patchData('fan');
    },
  },
  mounted() {
    axios
      .get(
        'https://vue-http-908f5-default-rtdb.europe-west1.firebaseio.com/Reg.json'
      )
      .then((response) => {
        this.valueLight = response.data.RegLight.value;
        this.valueFan = response.data.RegFan.value;
      });
  },
};
```

Obrázok 29 Script – RegulateLights.vue

```
methods: {
  patchData(param) {
    let isValue = 0;
    let whereData = '';
    if (param === 'light')
      (isValue = this.valueLight), (whereData = 'RegLight');
    else if (param === 'fan')
      (isValue = this.valueFan), (whereData = 'RegFan');
    axios.patch(
      'https://vue-http-908f5-default-rtdb.europe-west1.firebaseio.com/Reg/${whereData}.json?auth=' +
      this.$store.getters.token,
      {
        value: isValue.toString(),
      }
    );
  },
};
};
</script>
```

Obrázok 30 Funkcia patchData

Na Obrázku 30 môžeme vidieť ako spomínaná funkcia `patchData` v komponente `RegulateLights.vue` vyzerá. Podmienkami si vyselektujeme, či ideme regulovať svetlo alebo ventilátor. Potom už len cez HTTP request vytvorený s axios knižnicou pošlem dáta do databázy. Premenná v tomto prípade je číslo, ktoré potrebujem dostať do stringu kvôli hardwaru. Toto urobíme cez metódu `toString()` (viď Obrázok 30).



Obrázok 31 Lights – zobrazenie na mobilnom zariadení

V prípade otvorenia stránky na mobile, po prekliku na Lights uvidíme v dolnej časti správu, ktorá nám hovorí že na dotykovom zariadení nemôžeme regulovať svetlo a ventilátor. Je to z bezpečnostných dôvodov, pretože tento slider nefunguje tak ako by sme si mohli predstavovať. Po potiahnutí stránky nižšie, aby sme sa dostali k regulovaniu, čo by sme museli spraviť potiahnutím cez slider, by sa hodnota regulácie zmenila – čo by používateľ možno ani nechcel spraviť. Preto je táto funkcia na dotykovom zariadení nedostupná. Vyberanie obsahu na stránke, ktorý zobrazujeme na základe zariadenia, na ktorom používateľ je, sme spravili cez Bulmu (viď Obrázok

32). Je to trieda HTML tagu, ktorú Bulma ponúka a tou je: is-hidden-tablet. Čiže pokiaľ sme na dotykovom zariadení, ukáže sa nám text, ktorý je v <p> tagu.

```
<template>
  <lights-card-model></lights-card-model>
  <regulate-lights class="mt-5 is-hidden-mobile"></regulate-lights>
  <p class="box has-text-centered is-hidden-tablet is-size-5 p-4">
    Sorry, you can't regulate <br />
    fan and light on touch device.
  </p>
</template>

<script>
import LightsCardModel from './LightsCardModel.vue';
import RegulateLights from './RegulateLights.vue';

export default {
  components: {
    LightsCardModel,
    RegulateLights,
  },
};
</script>
```

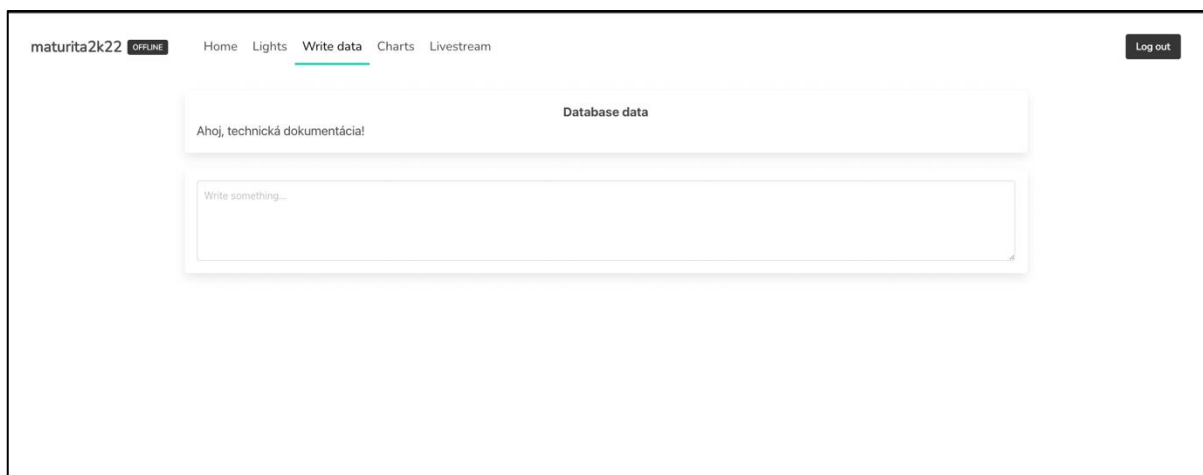
Obrázok 32 LightsAll.vue

3.3 ZAPISOVAŤ INFORMÁCIE NA DISPLEJ

V tomto projekte primárna funkcia stránky je dostávať a následne zobrazovať informácie z databázy alebo určité informácie na databázu posielat'. Teraz ideme robiť práve tú prvú možnosť. Už z nadpisu kapitoly vyplýva, že informácie budeme hlavne odosielať. Funguje to jednoducho. Používateľ zadá do poľa určeného na písanie čo sa má poslať a stlačením klávesy enter dáta pošle. Informáciami, ktoré môžeme v tomto prípade na displej zapisovať môžu byť buď písmená, špeciálne znaky alebo čísla. Tento celý vstup sa na databázu pošle ako string. Ako vyzerá template tohto komponentu vidíme na Obrázku 33. Komponent sa volá WriteData.vue a nachádza sa v priečinku components (viď Obrázok 10).

```
<template>
  <div class="box">
    <p class="has-text-centered is-size-5 has-text-weight-bold">
      Database data
    </p>
    <p class="is-size-5">{{ databaseMessage }}</p>
  </div>
  <div class="box">
    <textarea
      class="textarea"
      v-model="message"
      placeholder="Write something..."
      @keyup.enter="patchData"
    ></textarea>
  </div>
</template>
```

Obrázok 33 Template – WriteData.vue



Obrázok 34 Zapisovanie informácií na displej

Na Obrázku 34 vidíme ako sa template z Obrázku 33 zobrazuje na stránke. Ako som už spomínal, do poľa používateľ napíše čo chce zobraziť na displeji a stlačením enteru sa začne práca funkcie `patchData` (viď Obrázok 35). Naše funkcie sme pomenúvali stále tak isto pretože v tom nie je žiadny problém, tieto funkcie existujú len v týchto komponentoch resp. súboroch.

```
<script>
import axios from 'axios';

export default {
  data() {
    return {
      message: '',
      databaseMessage: '',
    };
  },
  mounted() {
    setInterval(() => {
      axios
        .get(
          'https://vue-http-908f5-default-rtdb.europe-west1.firebaseio.com/DisplayData/value.json'
        )
        .then((response) => {
          this.databaseMessage = response.data;
        });
    }, 1000);
  },
  methods: {
    patchData() {
      axios.patch(
        'https://vue-http-908f5-default-rtdb.europe-west1.firebaseio.com/DisplayData.json?auth=' +
        this.$store.getters.token,
        {
          /* nakonci stringu zostavalo \n */
          value: this.message.slice(0, this.message.length - 1),
        }
      );
      this.message = ''; /* clear vstup */
    },
  },
};
</script>
```

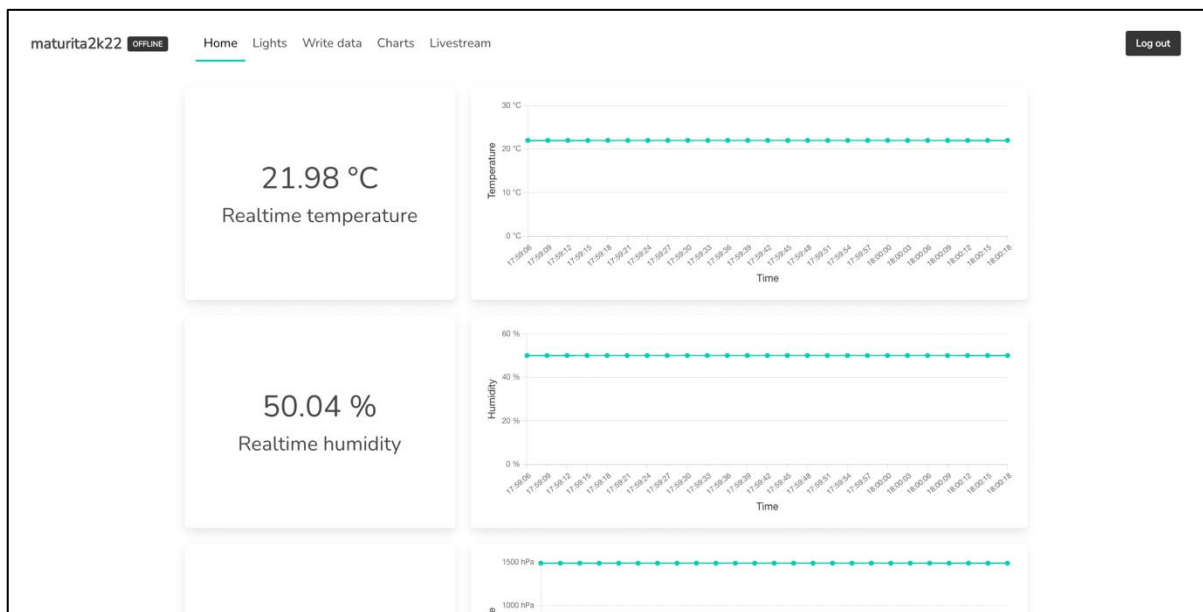
Obrázok 35 Script – WriteData.vue

Obsahom funkcie `patchData` je zase len HTTP požiadavka, no teraz s metódou `patch`. Metóda `patch` sa používa ak chceme v databáze vymeniť len iné časti stringu, ktoré tam ešte nie sú. Takže pri poslaní používateľom vety „Ahoj, technická dokumentácia!“ sa táto veta zapíše na databázu. No pri ďalšej zmene vety napríklad na „Dobrý deň, technická dokumentácia!“, sa vymení len prvá časť vety po čiarku. Táto metóda sa teda používa vtedy, keď vieme, že sa môže meniť obsah len čiastočne, a tým pádom nebudeme zaťažovať databázu viac než je nutné.

Mounted property sa v komponente `WriteData.vue` (viď Obrázok 35) stará o načítanie zapísaných dát v databáze. A robí to každých tisíc milisekúnd resp. každú sekundu. Môže za to funkcia `setInterval()`, ktorá je zadefinovaná v základnej knižnici JavaScriptu. Do tej sme ako prvý argument vložili takzvanú arrow funkciu (nová syntax funkcií používaná štandardom ES6 alebo ECMAScript 6 pre JavaScript), ktorá je trochu iná od normálnych funkcií. Dovolí nám pristupovať k všetkému JavaScriptu ako keby sme neboli vo funkcii. Pretože normálne funkcie nám toto nedovolia. Normálne funkcie sú `scoped`, čiže vidia len to čo majú v argumentoch a svojom tele. My sme potrebovali pristúpiť z tejto funkcie k premennej `databaseMessage`, ku ktorej sa dostaneme v `<script>` tagu len pomocou špeciálneho slovíčka `this`. A toto slovíčko `this` vždy ukazuje na niečo iné záleží na tom, kde sa práve v kóde nachádzame. No a preto že sme potrebovali práve tú premennú, ktorá bola vo funkcii `data()`, nijak inak by sme sa k nej takto šikovne nedostali, len cez arrow funkciu. Jej obsahom je HTTP požiadavka s metódou `GET`, pretože určitú informáciu z databázy chcem vybrať. Následne po odozve z databázy, prijatú informáciu uložíme do spomínanej premennej `databaseMessage`, ktorú ukážeme používateľovi použitím `vue.js` syntaxe `{{ databaseMessage }}` pod nápisom Database data (viď Obrázok 34).

3.4 VIDIEŤ INFORMÁCIE O TEPLOTE, VLHKOSTI A TLAKU V MIESTNOSTI

Informácie ohľadom teploty, vlhkosti a tlaku vzduchu v miestnosti mohol už aj neprihlásený užívateľ (viď Obrázok 11) . Prihlásenie dáva však používateľovi privilégia vidieť aj dynamické grafy (viď Obrázok 36).



Obrázok 36 /home – prihlásený používateľ

Grafy, ktoré sa aktualizujú každé tri sekundy. Počet dát na grafe nepresiahne číslo 25. Funkcia, ktorá sa stará o dostávajúce informácie o nových hodnotách z databázy sa volá `getData` (viď Obrázok 37) a nachádza sa v store v súbore `main.js`.

```

getData(context, payload) {
  let type = "";
  const chartLength = 25;
  const date = new Date();

  //context.commit("name", {payload}) - mutations
  context.commit("setTime", {
    time: `${(date.getHours() < 10 ? '0' : '') + date.getHours()}:${(date.getMinutes() < 10 ? '0' : '') + date.getMinutes()}:${(date.getSeconds() < 10 ? '0' : '') + date.getSeconds()}`
  })

  // context.state.time;
  if (payload.type === "temperature") {
    type = "temperature"
  } else if (payload.type === "humidity") {
    type = "humidity"
  } else if (payload.type === "pressure") {
    type = "pressure";
  }
}

```

Obrázok 37 Funkcia `getData`

Funkcia `getData` si najprv získa dátum cez JavaScriptovú funkciu `Date()`. Túto funkciu sme si uložili do premennej `date`, na ktorú sme následne naviazali metódy, ktoré nám funkcia `Date()` ponúka. Ako napríklad `getHours()`, čo nám vráti počet hodín. S ostatnými metódami to je podobne, názov metódy napovedá čo dostaneme. Chceli sme, aby mal čas určitý formát a tak sme museli upraviť to čo sme dostali z funkcií

getHours(), getMinutes() a getSeconds(). Formátom malo byť toto: hh:mm:ss, čiže aby mali 2 miesta hodiny, 2 miesta minúty a 2 miesta sekundy. Kebyže máme číslo 8, tak pred neho musíme dopísať ešte 0, aby to spĺňalo náš formát. A to sme aj spravili pomocou JavaScriptovej syntaxe na porovnanie hodnôt (viď Obrázok 37). Tento náš výstup sme uložili cez setTime funkciu do premennej time, ktorá sa nachádza v store.

```
axios
.get(
  'https://vue-http-908f5-default-rtdb.europe-west1.firebaseio.com/Chart.json'
)
.then((response) => {
  if (type === "temperature") {
    context.commit("setTemperature", {
      data: response.data.temperature,
    })

    const chartArray1 = context.state.temperature.chart;
    if (chartArray1.length > chartLength) {
      context.commit("editChartTemperature");
    }
  } else if (type === "humidity") {
    context.commit("setHumidity", {
      data: response.data.humidity,
    })
    const chartArray2 = context.state.humidity.chart;
    if (chartArray2.length > chartLength) {
      context.commit("editChartHumidity");
    }
  } else if (type === "pressure") {
    context.commit("setPressure", {
      data: response.data.pressure,
    })
    const chartArray3 = context.state.pressure.chart;
    if (chartArray3.length > chartLength) {
      context.commit("editChartPressure");
    }
  }
});
```

Obrázok 38 HTTP požiadavka – funkcia getData

Na Obrázku 38 robíme API call (HTTP požiadavku), ktorým dostávame namerané dáta z databázy. Následne ich ukladáme cez funkciu napríklad pri teplote setTemperature do premennej temperature v store. Aby sme na grafe nemali viac ako 25 hodnôt, počet uložených hodnôt do poľa premennej temperature porovnávame s premennou chartLength (viď Obrázok 37). Ak nastane situácia, že v poli je viac hodnôt ako je číslo uložené v chartLength, tak potom z poľa vyberieme prvú a teda najstaršiu hodnotu.

Narábanie s prvkom store, je síce trochu ťažšie, ale vyplatí sa. Preto na uloženie teploty do poľa voláme funkciu setTemperature, pretože je predpísaný spôsob akým sa majú aktualizovať premenné v store. Store ponúka actions, mutations a getters properties. Actions sú normálne funkcie ako vidíme na obrázkoch 37 a 38. Mutations

sú funkcie ako napríklad `setTemperature`, ktoré robia výlučne len s premennými v store (viď Obrázok 39). A getters sú tiež funkcie, ktoré nám len premenné z store vracajú (viď Obrázok 40).

```
setTemperature(state, payload) {  
  //payload.data = actual value of temperature  
  state.temperature.data = payload.data;  
  
  //chart takes values in format [osX, osY] - [time, value]  
  state.temperature.chart = state.temperature.chart.concat([  
    [state.time, payload.data]  
  ])  
},
```

Obrázok 39 Mutations

```
temperatureChart(state) {  
  return state.temperature.chart;  
},
```

Obrázok 40 Getters

To aby sa nám hodnoty menili nielen pri načítaní komponentu s týmito grafmi, ale aby sa načítavali aktuálne hodnoty stále keď sme na stránke, je spravené v `App.vue` komponente – srdce `vue.js` aplikácie (viď Obrázok 41). Tu sa odohráva to čo používateľ vidí na stránke, tu sa menia tie komponenty pomocou ďalšieho špeciálneho prvku `vue.js` - `router`. Takže akonáhle si používateľ otvorí našu webovú aplikáciu, tak sa spustí `created` property (viď Obrázok 41). Tu zavoláme funkciu `getData`, vždy s iným typom hodnoty, ktorú chceme z databázy vytiahnuť. Ďalej sme funkciu `getData` dali aj do funkcie `setInterval`, aby sa opakovala každé tri sekundy (druhý argument funkcie `setInterval`).

`Router` (viď Obrázok 42) je prvkom, ktorý sa stará o to, aby sme vedeli povedať kde sa na stránke nachádzame. Ide o tvorbu linku. Ak klikneme na `Home` tlačidlo v navigácii – link stránky sa nám zmení na `[doména]/home` a načíta sa komponent, ktorý je s týmto linkom spätý. Toto sa nastavuje vo funkcii `createRouter` kde musím špecifikovať pole `routes`, ktoré obsahuje hlavne cesty a komponenty. A komponent sa načíta práve tam, kde je určené na to miesto a to je na Obrázku 41 HTML tag `<router-view>`.

```

<template>
  <navigation></navigation>
  <div class="container mt-4">
    <router-view></router-view>
  </div>
</template>

<script>
import Navigation from './components/Navigation.vue';

export default {
  components: {
    Navigation,
  },
  created() {
    this.$store.dispatch('getData', {
      type: 'temperature',
    });
    this.$store.dispatch('getData', {
      type: 'humidity',
    });
    this.$store.dispatch('getData', {
      type: 'pressure',
    });
    this.$store.dispatch('getChart2');

    setInterval(() => {
      this.$store.dispatch('getData', {
        type: 'temperature',
      });
      this.$store.dispatch('getData', {
        type: 'humidity',
      });
      this.$store.dispatch('getData', {
        type: 'pressure',
      });
    }, 3000);
  },
};
</script>

```

Obrázok 41 App.vue

```

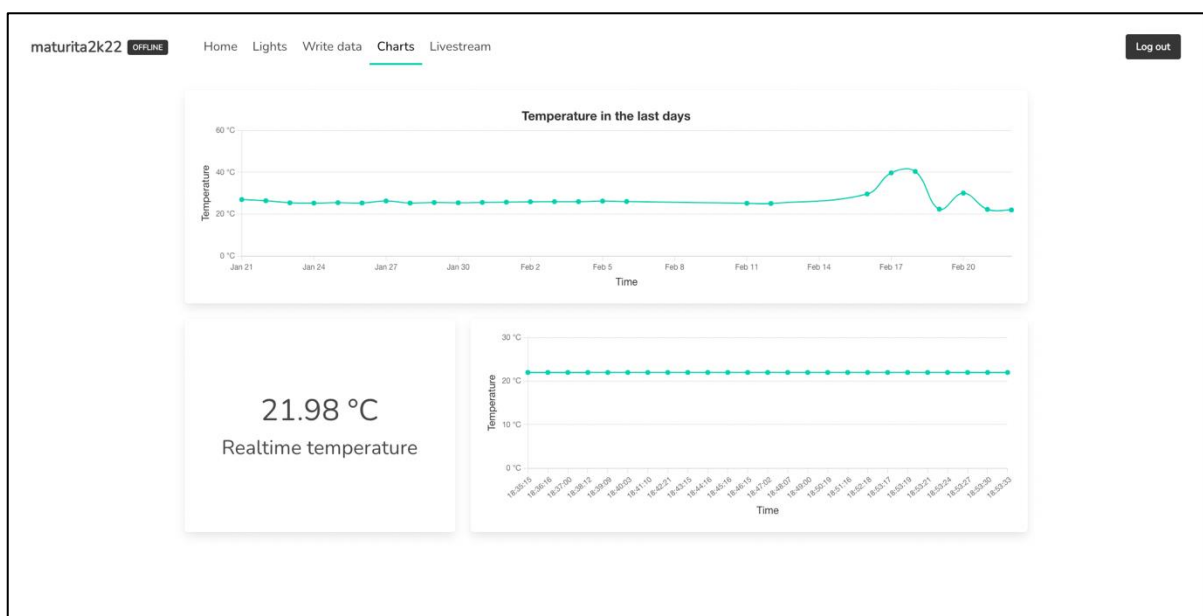
const router = createRouter({
  history: createWebHistory(),
  routes: [
    { path: "/", redirect: "/home" },
    { path: "/home", component: Charts },
    { path: "/lights", component: LightsAll, meta: { auth: true } },
    { path: "/write-data", component: WriteData, meta: { auth: true } },
    { path: "/charts", component: Charts, meta: { auth: true } },
    { path: "/live-stream", component: LiveStream, meta: { auth: true } },
    { path: "/admin", component: AdminPanel, meta: { admin: true } },
    { path: "/*", redirect: "/home" }
  ],
  linkActiveClass: "is-active is-tab"
});

```

Obrázok 42 Router

3.5 VIDIEŤ GRAF TEPLOTY ZA POSLEDNÉ DNI

Do tejto časti stránky sa dostaneme pomocou tlačidla Charts v navigácii. Tento zobrazený komponent obsahuje graf teploty za posledné dni, ktorý sa nachádza v hornej časti. Pod ním, sa pre porovnanie, nachádza kartička s aktuálnou teplotou a graf, vyjadrujúci 25 hodnôt teploty nie starších ako minútu a 12 sekúnd (viď Obrázok 43).



Obrázok 43 Charts

Dáta pre každý graf v tejto aplikácii, musia byť uložené v poli. Vyzerat' by to malo nasledovne. Máme `pole[]`. Každé miesto v poli je ďalšie pole – `pole[[čas, hodnota], [čas, hodnota], [čas, hodnota]]`. To pole ako sme už ukázali je názorne vyplnené na mieste s indexom 0 časom a na mieste indexu 1 je hodnota. Z takéhoto poľa vieme spraviť graf. V našej aplikácii máme jedno základné pole pre teplotu, ďalšie základné pole pre vlhkosť a ďalšie základné pole pre tlak. A pri grafe hodnôt teploty za posledné dni tomu nebude inak.

```
setChart2(state, payload) {  
  state.chart2.chart = state.chart2.chart.concat([[payload.day, payload.value]])  
},
```

Obrázok 44 Tvorba poľa pre graf

Týmto polom pre graf za posledné dni pre hodnoty teploty je pole chart (viď Obrázok 45), ktoré sa nachádza v state property v store. State je to isté ako data property v normálnom komponente. Používa sa len na premenné.

```
chart2: {  
  chart: []  
}
```

Obrázok 45 pole
chart

Pomocou funkcie setChart2() (viď Obrázok 44) ukladáme do poľa chart na ďalšie indexované miesta (ktoré začínajú od 0) ďalšie polia, ktoré majú obsadený index 0 a 1, kde sa nachádzajú dátum a hodnota teploty v tom dátume. Funkcia setChart2() necháva v poli to čo tam už bolo, len vďaka metóde concat pripája ďalšie polia do poľa chart – tým pádom sa vytvárajú nové hodnoty v grafe s konkrétnym dátumom a hodnotou teploty.

3.6 NAHLIADNUŤ DO MIESTNOSTI POMOCOU KAMERY

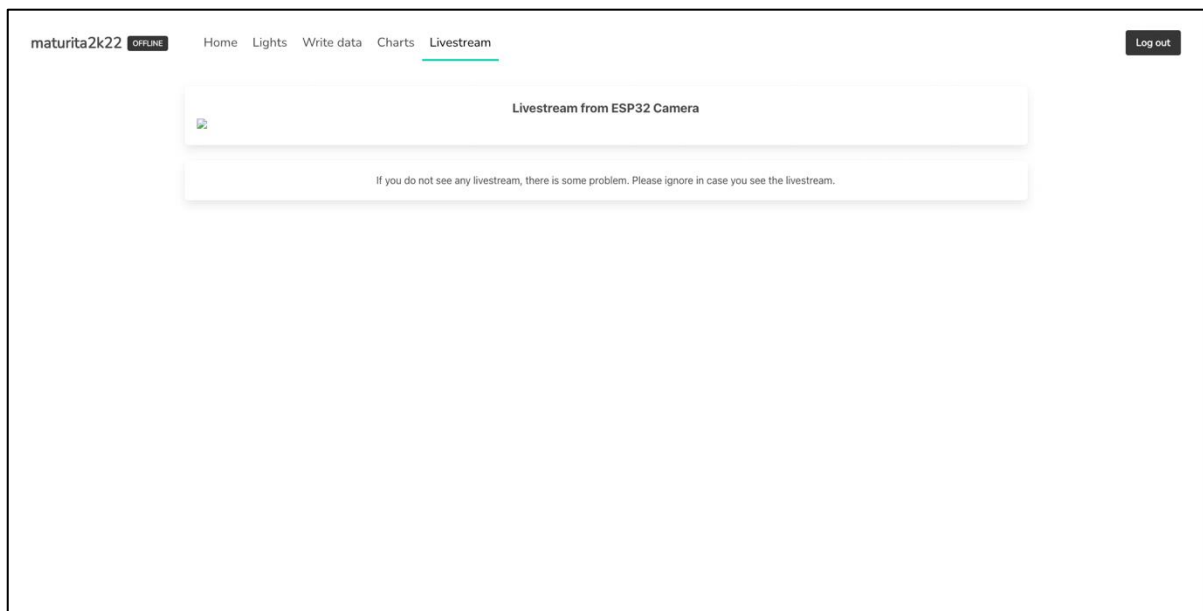
Nahliadnuť do miestnosti môžeme pomocou kamery, ktorá vysiela živý prenos na lokálnej sieti. Tento prenos pomocou tagu a lokálnej IP adresy kamery zobrazujeme na stránke.

Template vyzerá takto (viď Obrázok 46). Na zobrazenie prenosu potrebujeme len tag, ktorému ako zdroj dát určíme server resp. jeho port, na ktorom sa tento prenos vysiela.

```
<template>  
  <div class="box">  
    <p class="is-size-5 has-text-centered has-text-weight-bold">  
      Livestream from ESP32 Camera  
    </p>  
    <div>  
        
    </div>  
  </div>  
  <div class="box has-text-centered">  
    If you do not see any livestream, there is some problem. Please ignore in  
    case you see the livestream.  
  </div>  
</template>
```

Obrázok 46 LiveStream.vue

Pokiaľ kamera, ktorá je založená na platforme esp32 (mikrokontrolér), nie je zapnutá. Používateľ sa bohužiaľ musí uspokojiť textom, ktorý mu vysvetľuje, že niekde nastal problém.



Obrázok 47 Livestream

4 VYTVORTE ADMINISTRÁTORSKÉ ROZHRANIE, KTORÉ BUDE OBSAHOVAŤ:

4.1 MOŽNOSŤ PRIDÁVANIA, MAZANIA A ÚPRAVY UŽÍVATEĽOV

V administrátorskom rozhraní (viď Obrázok 48) nájdeme, ak by sme išli od hora, dve tlačidlá jedno zelené a jedno čierne, o tom čiernom sa budeme rozprávať v neskorších bodoch dokumentácie. Zadanie tohto bodu bolo vytvoriť administrátorovi možnosť pridávať, mazať alebo upraviť užívateľa. A najrozumnejším spôsobom, ako toto spraviť nám prišlo vytvorenie tabuľky s údajmi o užívateľovi, za ktorými by potom boli tlačidlá na odstránenie alebo upravenie. Tlačidlo na pridanie užívateľa nemá s už existujúcimi užívateľmi nič spoločné a preto sme ho dali nad tabuľku. Zelený riadok v tabuľke, v ktorom sa nenachádzajú tlačidlá na odstránenie alebo upravenie používateľa, označujeme administrátora.

ID	Email	Username	Phone number	Creation time	Last Sign-In	
1	dada@dada.sk			Fri, 04 Mar 2022 19:10:09 GMT	Fri, 04 Mar 2022 19:10:09 GMT	Delete user ✖ Edit 🛠
2	tomas@tomas.sk			Fri, 04 Mar 2022 22:17:44 GMT	Sat, 05 Mar 2022 18:43:07 GMT	Delete user ✖ Edit 🛠
3	tomas@gmail.com			Sun, 20 Feb 2022 18:01:46 GMT	Sun, 20 Feb 2022 18:14:21 GMT	Delete user ✖ Edit 🛠
4	tomko@tomko.sk			Tue, 25 Jan 2022 23:08:26 GMT	Sat, 05 Mar 2022 18:49:44 GMT	
5	tomasbordak@das.sk			Thu, 03 Mar 2022 21:15:07 GMT	Thu, 03 Mar 2022 21:15:07 GMT	Delete user ✖ Edit 🛠
6	riperino@peperino.sk	JesJES	+421910019822	Sun, 20 Feb 2022 18:48:20 GMT		Delete user ✖ Edit 🛠
7	jozef.mihok@gmail.com	Jozef Mihok	+421908727454	Wed, 02 Feb 2022 21:25:16 GMT		Delete user ✖ Edit 🛠
8	eltonsetan2@gmail.com	eltonos setanos		Tue, 15 Feb 2022 21:34:32 GMT	Wed, 16 Feb 2022 21:22:37 GMT	Delete user ✖ Edit 🛠

Obrázok 48 Admin options

Na Obrázku 49 môžeme vidieť začiatok tabuľky napísaný v HTML kóde. Tag `<table>` označuje tabuľku, `<thead>` označuje hlavičku tabuľky a `<tbody>` označuje telo tabuľky. Tagy `<thead>` a `<tbody>` pochádzajú z Bulmy. Tag `<tr>` označuje riadok a `<th>` označuje stĺpec. V hlavičke tabuľky sa nám nachádza ako prvé ID, druhé Email a tretie Username – ďalej to pokračuje presne tak, ako je definované HTML kódom (viď Obrázok 49).

```
<table class="table is-hoverable m-4">
  <thead>
    <tr>
      <th>ID</th>
      <th>Email</th>
      <th>Username</th>
      <th>Phone number</th>
      <th>Creation time</th>
      <th>Last Sign-In</th>
      <th></th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>
        <div class="{
          'is-selected': user.uid === 'XCvGaNAKsPR6ADhcvCC9hzFC5po2',
        }"
        v-for="(user, index) in users"
        :key="user[index]"
      >
        <th>{{ index + 1 }}</th>
        <td>{{ user.email }}</td>
        <td>{{ user.displayName }}</td>
        <td>{{ user.phoneNumber }}</td>
        <td>{{ user.metadata.creationTime }}</td>
        <td>{{ user.metadata.lastSignInTime }}</td>
      </td>
    </tr>
  </tbody>
</table>
```

Obrázok 49 Tabuľka – 1. časť

V dolnej časti Obrázku 49 vidíme spôsob, ktorým vypisujeme užívateľov do tabuľky. Mounted property nám dáva používateľov z back-endu pomocou API callu (viď Obrázok 50). Týchto používateľov sme si uložili do premennej `users`, ktorou sa po naplnení užívateľmi stáva objekt. Ako v každom inom objekte sa vieme pohybovať pomocou prvkov. V tomto prípade to vyzerá napríklad takto: `user.email` nám dá email užívateľa alebo `user.phoneNumber` nám dá jeho telefónne číslo. Vďaka špeciálnej `vue.js` syntaxe sme mohli prechádzať for cyklom cez objekt, ktorý sa opakoval až do posledného užívateľa.

To čo sme dostali z API callu sme si najprv vypísali do konzole, kde sme zistili ako nám boli poslané dáta a potom sme ich vedeli uložiť. Tak sme prišli na to, že čas založenia účtu a čas posledného prihlásenia sa nenachádza spolu s údajmi

o užívateľovi hneď v premennej user, ale ešte sme museli ísť o úroveň nižšie do user.metadata.

```
mounted() {
  this.usersLoading = true;
  axios
    .get('https://us-central1-vue-http-908f5.cloudfunctions.net/getUsers')
    .then((response) => {
      this.users = response.data;
      this.usersLoading = false;
    });
  setInterval(() => {
    axios
      .get('https://us-central1-vue-http-908f5.cloudfunctions.net/getUsers')
      .then((response) => {
        this.users = response.data;
      });
  }, 5000);
},
```

Obrázok 50 Mounted – AdminPanel.vue

Na odstránenie užívateľa potrebujeme jeho identifikačné číslo, ktoré zašleme na určený API call, za ktorým sa skrýva back-end funkcia, ktorá tohto užívateľa z back-endu vymaže.

```
deleteUser(uid, index) {
  this.loadingDel[index] = true;
  axios
    .post('https://us-central1-vue-http-908f5.cloudfunctions.net/delUser', {
      uid,
    })
    .then(() => {
      axios
        .get(
          'https://us-central1-vue-http-908f5.cloudfunctions.net/getUsers'
        )
        .then((response) => {
          this.users = response.data;
          this.loadingDel[index] = false;
        });
    });
},
```

Obrázok 51 deleteUser funkcia


```

<td class="has-text-centered">
  <button
    class="button is-danger is-outlined"
    v-if="user.uid != 'XCvGaNAKsPR6ADhcvCC9hzFC5po2'"
    :class="{ 'is-loading': loadingDel[index] }"
    @click="deleteUser(user.uid, index)"
  >
    <span>Delete user</span>
    <span class="icon is-small">
      <i class="fas fa-times"></i>
    </span>
  </button>
</td>
<td>
  <div v-if="user.uid != 'XCvGaNAKsPR6ADhcvCC9hzFC5po2'">
    <button
      @click="
        infoUser(user.uid);
        toggleModalEdit = true;
      "
      class="button is-outlined is-primary"
    >
      <span>Edit</span>
      <span class="icon is-small">
        <i class="fas fa-user-edit"></i>
      </span>
    </button>
  </div>
</td>
</tr>
</tbody>
</table>

```

Obrázok 52 Tabuľka – 2. časť

Funkcia deleteUser (viď Obrázok 51) na vymazanie užívateľa berie hlavne už spomínaný identifikátor užívateľa. Tlačidlo na vymazanie užívateľa je vytvorené cez for cyklus a každý jeden cyklus je nový užívateľ, čiže po stlačení tlačidla Delete user sa zavolá funkcia deleteUser (viď Obrázok 52), za čo môže vlastnosť na HTML tagu @click. Do tejto funkcie sa pošle ako prvý argument identifikátor presne toho užívateľa, pri ktorom kliknuté tlačidlo stojí. Druhým argumentom pre funkciu deleteUser je index používateľa, ktorý používame len na to, aby začala animácia načítavania tlačidla Delete user pri tom správnom užívateľovi.

Pomocou v-if vlastnosti na HTML tagu (viď Obrázok 52) robíme to, aby sa pri administrátorovi nezobrazili tlačidlá vymazať užívateľa alebo upraviť užívateľa. Ako administrátora sme si určili email: tomko@tomko.sk , našli sme si jeho identifikátor a tento porovnávali vo vlastnosti v-if. Ak sa identifikátor užívateľa zhodoval s identifikátorom administrátora, tlačidlá neboli pre tohto užívateľa vyrobené.

Upraviť užívateľa môžeme tlačidlom, ktoré sa nachádza v riadku užívateľa. Po stlačení zeleného tlačidla Edit sa nám ukáže okno Edit user, v ktorom môžeme

užívateľovi zmeniť: meno, emailovú adresu a telefónne číslo. Ak chceme zmeniť len jednu z týchto informácií, použijeme na to tlačidlo Update, ktoré patrí poľu, ktoré meníme (viď Obrázok 53).

Edit user

Username

[Update](#)

Email address

[Update](#)

Must be a valid email address

Phone number

[Update](#)

Must be in format: +421 901 919 919

[Edit All ✓](#)[Close](#)

Obrázok 53 Edit User okno

Ak by sme chceli zmeniť všetky dostupné informácie o užívateľovi, vyplníme polia a tlačidlom Edit All potvrdíme zmenu. Po zmene sa okno automaticky zavrie a tabuľka sa obnoví, aby informácie boli vždy aktuálne.

Funkcia iba na zmenu emailovej adresy, čiže pomocou vyplnenia poľa emailovej adresy a stlačenia tlačidla Update pri tomto poli sa zavolá funkcia submitEmail.

```

submitEmail() {
  this.loadingEmailUpdate = true;
  const uid = this.userId;
  const email = this.email;
  axios
    .post('https://us-central1-vue-http-908f5.cloudfunctions.net/upUser', {
      uid,
      email,
    })
    .then(() => {
      axios
        .get(
          'https://us-central1-vue-http-908f5.cloudfunctions.net/getUsers'
        )
        .then((response) => {
          this.users = response.data;
          this.loadingEmailUpdate = false;
          //setting empty string on input values
          this.email = '';
        });
    });
},

```


Obrázok 54 funkcia submitEmail

Funkcia submitEmail (viď Obrázok 54) posiela na back-end identifikátor užívateľa, ktorému chceme zmeniť email a samozrejme nový email, na ktorý chceme starý email zmeniť. Ako som už spomínal, po úspešnej odozve od back-endu, funkcia načíta všetkých užívateľov znova, aby informácie o užívateľoch boli aktuálne. Typ tejto funkcie sa opakuje aj pri ďalších poliach. Pri zmene všetkých informácií o užívateľovi naraz, funkcia pošle spolu s identifikátorom aj všetky nové informácie.


Pridanie užívateľa realizujeme stlačením zeleného tlačidla Create user (viď Obrázok 48). Po stlačení sa nám zobrazí okno Create User, ktoré obsahuje 4 polia a to: meno, emailová adresa, heslo a telefónne číslo. Po vyplnení týchto polí a stlačení tlačidla Create New User (viď Obrázok 55), sa tieto údaje pošlú na back-end. Pošlú sa práve vďaka funkcii createUser() (viď Obrázok 56).

Create User

Username


 Username

Email address

 Email address


Must be a valid email address

Password

 Password

Must be a atleast 6 characters long!

Phone number

 Phone number

Must be in format: +421 901 919 919

Create New User ✓

Obrázok 55 Create User okno

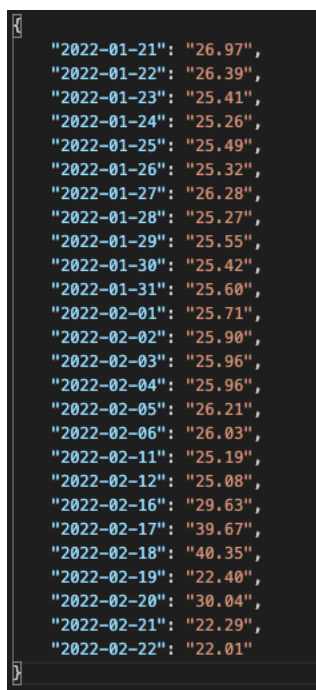
```
createUser() {  
  this.loadingCreateUser = true;  
  axios  
    .post(  
      'https://us-central1-vue-http-908f5.cloudfunctions.net/createUser',  
      {  
        email: this.createUserData.email,  
        phoneNumber: this.createUserData.number,  
        displayName: this.createUserData.username,  
        password: this.createUserData.password,  
      }  
    )  
    .then(() => {  
      this.createUserData.email = '';  
      this.createUserData.number = '';  
      this.createUserData.username = '';  
      this.createUserData.password = '';  
      this.loadingCreateUser = false;  
      this.toggleModalCreateUser = false;  
      axios  
        .get(  
          'https://us-central1-vue-http-908f5.cloudfunctions.net/getUsers'  
        )  
        .then((response) => {  
          this.users = response.data;  
        });  
    });  
},
```

Obrázok 56 createUser funkcia

4.2 SŤAHOVAŤ NAMERANÉ DÁTA Z DATABÁZY CEZ WEBOVÉ ROZHRANIE

Pre dáta, ktoré sa dajú stiahnuť, sme si vybrali namerané hodnoty za posledné dni, pretože tie majú asi najväčší zmysel v tom ich nevidieť len na stránke, ale mať stiahnuté v počítači. Túto funkciu webovej aplikácie vykonáva čierne tlačidlo Download data (viď Obrázok 48).

Po stlačení čierneho tlačidla Download data – Temperature in the last days sa nám stiahne .json súbor s nameranými hodnotami teploty za posledné dni (viď Obrázok 57). Funguje to tak, že najprv musíme stiahnuť všetky namerané hodnoty teploty za posledné dni z databázy a potom ich vo funkcii prepísať na JSON objekt s rozumnou štruktúrou pre prehľadné čítanie alebo hoci aj pre vloženie do aplikácie Excel. Funkcia, ktorá toto zastrešuje je vlastne funkciou v mutations v store (viď Obrázok 58).



```
"2022-01-21": "26.97",
"2022-01-22": "26.39",
"2022-01-23": "25.41",
"2022-01-24": "25.26",
"2022-01-25": "25.49",
"2022-01-26": "25.32",
"2022-01-27": "26.28",
"2022-01-28": "25.27",
"2022-01-29": "25.55",
"2022-01-30": "25.42",
"2022-01-31": "25.60",
"2022-02-01": "25.71",
"2022-02-02": "25.90",
"2022-02-03": "25.96",
"2022-02-04": "25.96",
"2022-02-05": "26.21",
"2022-02-06": "26.03",
"2022-02-11": "25.19",
"2022-02-12": "25.08",
"2022-02-16": "29.63",
"2022-02-17": "39.67",
"2022-02-18": "40.35",
"2022-02-19": "22.40",
"2022-02-20": "30.04",
"2022-02-21": "22.29",
"2022-02-22": "22.01"
```

Obrázok 57
teplotaZaPosledneDni.json

```

showData(state) {
  const firstArray = JSON.parse(JSON.stringify(state.chart2.chart))
  const theObject = {}
  for (let i = 0; i < firstArray.length; i++) {
    if (firstArray[i] !== undefined) {
      theObject[firstArray[i][0]] = firstArray[i][1]
    }
  }
  console.log(JSON.stringify(theObject))
  /* dokumentacia k tomu na npm download(data, filename, mimetype) */
  download(JSON.stringify(theObject), "teplotaZaPosledneDni.json", "text/plain")
}

```

Obrázok 58 Stiahnutie nameraných hodnôt

4.3 PRIHLÁSENIE ADMINISTRÁTORA

Prihlásenie užívateľa a teda to, akým spôsobom vyberáme administrátora, ktorý vidí administrátorské rozhranie (viď Obrázok 48) je jednoduchý. Robíme to pomocou v-if, ktoré už v tejto dokumentácii vysvetlené je. Vo v-if porovnávame niečo s niečím a ak to bude pravdivé čiže boolean true, daný obsah užívateľovi ukážeme.

```

<router-link
  v-if="$store.getters.userId === 'XCvGaNAKsPR6ADhcvCC9hzFC5po2'"
  to="/admin"
  class="navbar-item is-primary is-size-5"
>Admin options</router-link
>

```

Obrázok 59 Admin options tlačidlo

Takto vyzerá v HTML (vue.js) kóde napísané Admin options tlačidlo v navigácii. Pomocou v-if ho buď zobrazíme alebo na stránku jednoducho nedáme. Aj keby sa ale podarilo niekomu možno cez developer tools v Chrome napríklad, pridať toto tlačidlo, existuje ešte spôsob, ktorým toto možné zneužitie developer tools prakticky zakážeme resp. nebude to vôbec k ničomu. A tým sú navigation guards teda takzvaný strážcovia navigácie (viď Obrázok 60). Na Obrázku 42 si môžeme všimnúť dáta pomenované ako meta. Určím meno a poviem, že to musí byť pravdivé. Strážcovia navigácie sa postarajú o to, aby žiadny iný email nemal prístup k administrátorskému rozhraniu a to pomocou toho, že si najprv overia či ten užívateľ má takéto meta dáta (viď Obrázok 60). Pred každým presmerovaním užívateľa na iný

link a teda na iný komponent si to tento strážca navigácie preverí a to podmienkami. Router obsahuje metódu `beforeEach`, ktorá nám dovoľuje skontrolovať práva užívateľa pred tým než ho presmeruje na nový komponent. V prípade administrátorského rozhrania to je podmienka, ktorá kontroluje či používateľ sa snaží dostať na komponent alebo link, ktorý obsahuje meta dáta admin. Dostane sa tam v prípade ak platí celá podmienka, ktorá znie: meta dáta admin sú true a identifikátor používateľa sa zhoduje s tým, ktorý bol nami vybraný (email: tomko@tomko.sk).

```
/* navigation guards */
router.beforeEach((to, _, next) => {
  if (to.meta.auth && store.state.token === "") {
    next("/home");
    store.commit("toggleModalSignUp")
  } else if (to.meta.admin && store.state.userId !== "XCvGaNAKsPR6ADhcvCC9hzFC5po2") {
    next("/home");
    store.commit("toggleModalSignUp")
  } else {
    next();
  }
})
```

Obrázok 61 Navigation guards

ZÁVER

V tomto projekte sme si mohli vyskúšať princípy a spôsoby, ktorými sa pristupuje k výrobe podobnej aplikácii. Naučili sme sa vytvoriť komunikáciu medzi front-endom a back-endom. Vyskúšali sme si ako funguje práca v tíme, kde má každý inú úlohu.

Vytvorili sme vue.js aplikáciu a získané vedomosti z kurzov sme použili v praxi. Naučili sme sa:

- ako funguje vue.js app life cycle – created, mounted ..
- čo je to single-page aplikácia a ako funguje
- ako obísť zdĺhavé kódovanie v CSS vďaka Bulme
- čo je to store a router
- ako sa vytvárajú HTTP požiadavky, aké metódy majú
- čo je to API a ako by mala fungovať v praxi
- ako pracovať s terminálom pri vytváraní vue.js aplikácie
- čo sú to npm balíky
- ako spolu súvisí HTML, CSS a JavaScript
- aký je rozdiel medzi frameworkom a knižnicou
- čo musí obsahovať vue.js aplikácia
- ako obmedziť prístup k funkciám webovej aplikácie jej používateľom (prihlásený/neprihlásený používateľ/administrátor)

Na vytvorenie tejto aplikácie sme potrebovali 2 kurzy v aktuálnej hodnote 170€ (6.3.2022). Programovaním aplikácie sme strávili 120 hodín, teda ak by sme rátali s hodnotou 5€/hodina, tak by cena tejto aplikácie vyšla na 600€. Náklady na udržiavanie stránky sa na internete pohybujú od 10€/mesiac, avšak pri tejto aplikácii použitej v tomto projekte sa platí len za používanie databázy. Čiže náklady na aplikáciu samotnú, by boli nulové.

ZOZNAM POUŽITEJ LITERATÚRY

YOU, E. *Vue CLI* [online], 2018. [navštívené dňa 2022-03-02], dostupné na: <https://cli.vuejs.org/> .

AR, S. *Smart Home Dashboard UI Concept - Freebie* [online]. [navštívené dňa 2022-02-17], dostupné na: <https://dribbble.com/shots/15418426-Smart-Home-Dashboard-UI-Concept-Freebie/attachments/7185647?mode=media> .

GUPTA, A. 2022. *Top 12 Front & Back-End JavaScript Frameworks for Web Development in 2022* [online], 2022. [navštívené dňa 2022-02-21], dostupné na: <https://blog.sagipl.com/javascript-frameworks/>.

MORRIS, J. 2020. *Front-end vs. back-end roles in software development* [online], 2020. [navštívené dňa 2022-02-21], dostupné na: <https://www.mcgregorboyall.digital/blog/2020/11/front-end-vs-back-end-roles-in-software-development> .

STEELE, C. 2022. *The Web Developer Bootcamp 2022* [online], 2022. [navštívené dňa 2022-02-22], dostupné na: <https://www.udemy.com/course/the-web-developer-bootcamp/>.

THOMAS, J. 2022. *Bulma* [online], 2022. [navštívené dňa 2022-02-23], dostupné na: <https://bulma.io/> .

vidabytes.com . *Čo je NPM? (Správca balíkov uzlov v programovaní)* [online]. [navštívené dňa 2022-02-23], dostupné na: <https://vidabytes.com/sk/%C4%8Do-je-npm/>.

Talking about the File protocol and Http protocol and the difference [online]. [navštívené dňa 2022-02-23], dostupné na: <https://www.programmerall.com/article/4339551145/>.

javatpoint.com. *What is ES6?* [online]. [navštívené dňa 2022-03-04], dostupné na:
<https://www.javatpoint.com/es6>.

SCHWARZMÜLLER, M. 2022. *Vue – The Complete Guide (incl. Router & Composition API)* [online], 2022. [navštívené dňa 2022-03-04], dostupné na:
<https://www.udemy.com/course/vuejs-2-the-complete-guide/>.