

Gymnázium, Praha 6, Arabská 14

Obor programování



MATURITNÍ PROJEKT

Jan Ševčík, 4.E

Webová aplikace pro výuku Linuxu

Duben 2025

Prohlášení

Prohlašuji, že jsem jediným autorem tohoto projektu, všechny citace jsou řádně označené a všechna použitá literatura a další zdroje jsou v práci uvedené. Tímto dle zákona 121/2000 Sb. (tzv. Autorský zákon) ve znění pozdějších předpisů uděluji bezúplatně škole Gymnázium, Praha 6, Arabská 14 oprávnění k výkonu práva na rozmnožování díla (§ 13) a práva na sdělování díla veřejnosti (§ 18) na dobu časově neomezenou a bez omezení územního rozsahu.

V dne

.....
Jan Ševčík

Anotace

Ve své Maturitní práci jsem se zabýval vývojem webové aplikace, která může jak komerčně tak školám pomáhat s výukou toho jak používat operační systém Linux. Cílem této práce bylo vyvinout webovou aplikaci, která bude alespoň trochu schopna pomoci uživatelům s výukou systému Linux. Myslím si, že tohoto bylo v rámci možností docíleno a aplikace splňuje zadání.

Annotation

In my Graduation project I pursue the development of a web application, which can either commercially or in school help people understand the operating system of Linux. The aim of the development of this web application was to create an application that will help users understand Linux. This has been in my opinion done well and the application works as it is stated in the assignment

Zadání

Tento maturitní projekt bude webová aplikace, která se bude zabývat výukou operačního systému Linux. Uživatel bude mít možnost vytvořit si účet, přihlásit se, změnit si heslo atd. Uživatel bude mít možnost připojit se do „týmu“, nebo tým vytvořit a pozvat do tohoto týmu další uživatele.

Pro tyto týmy bude mít uživatel možnost mít vytvořit úkol, nastavit mu datum začátku/konce, jméno atd. Tyto úkoly následovně můžou uživatelé v týmu splnit na základě zadání úkolu ve virtuálním počítači - kontejneru, který jim bude poskytnut prostřednictvím webového rozhraní. Tento počítač bude monitorován a pokud jeho zátěž překročí určenou hodnotu bude zastaven. Výsledky těchto úkolů budou následovně zaznamenány pro uživatele, který je vytvořil pro opravu.

Obsah

1 Úvod	1
2 Použitý software	2
3 Struktura projektu	2
4 Frontend	3
4.1 index.html	3
4.2 Konfigurace Vue Routeru	4
4.3 Konfigurace „supabase”	4
4.4 „Css” soubory	5
4.5 Stránky	5
4.6 Komponenty	7
4.6.1 Styly komponentů s příkazovými řádkami	7
4.6.2 Skripty komponentů s příkazovými řádkami	8
4.6.3 Komponent HomeComponentLine	9
4.6.4 Komponent AuthCommandLine	10
4.6.5 Komponent UserCommandLine	12
4.6.6 Komponent AdminCommandLine	13
4.6.7 Styly jedno sloupových „selector” komponentů	16
4.6.8 Skript jedno sloupových „selector” komponentů	17
4.6.9 Komponent LessonCreateSelector	17
4.6.10 Komponent LessonCheckSelector	18
4.6.11 Styly dvou sloupových „selector” komponentů	19
4.6.12 Skript dvou sloupových „selector” komponentů	20
4.6.13 Komponent LessonPageSelector	20
4.6.14 Komponent LessonSolutionsSelector	22
4.6.15 „navbar” komponenty	23
5 Backend	24
5.1 Flask api	24
5.1.1 Spojení s frontendem	25
5.1.2 Services	26
5.1.3 Endpointy pro lekce	26
5.1.4 Endpointy pro týmy	31
5.1.5 Endpointy pro platby	34
5.2 Python script	35
5.2.1 Skript pro kontrolu kontejnerů	35
5.2.2 Logovací funkce pro kontrolu kontejnerů	37
5.2.3 Skript pro kontrolu lekcí	38
5.3 Supabase	39
5.3.1 Konfigurační soubor	39
5.3.2 Struktura databáze	39
5.3.3 Tabulka „user”	40

5.3.4 Tabulka „limitations”	40
5.3.5 Tabulka „team”	41
5.3.6 Tabulka „lesson”	42
5.3.7 Tabulka „container”	43
5.3.8 Tabulka „container_script”	44
5.3.9 „Triggers” v databázi	45
5.3.10 „Cron jobs”	45
5.3.11 Funkce v databázi	46
6 Závěr	47
Použitá literatura	48
Seznam obrázků	50

1 Úvod

Tento dokument je dokumentací mého maturitního projektu, jehož zadání bylo vymyšleno a schváleno na začátku roku. Dokument obsahuje dokumentaci obou dvou částí projektu tedy frontendu jeho stylů a skriptů, kterými je ovládána uživateli aplikace, a backendu včetně skriptů pro kontrolu zátěže, api endpointů, které odpovídají na dotazy z frontendu a operují na úrovni systému a služby „supabase” obsahující celou databázi a její speciální funkce. Celý program je napsán ve třech jazycích Pythonu, Javascriptu a plpgsql za pomoci dvou frameworků „VueJs” a „Flask”, služby „supabase” a aplikace „docker”.

2 POUŽITÝ SOFTWARE

Pro frontendový vývoj aplikace byl použit framework „Vue Js”, který využívá jazyků „Javascript”, „Html” a „Css”. Byl použit pro jeho mnohá doporučení a ve vývojářských komunitách dobrou reputaci pro přehlednost. Pro lepší vzhled aplikace byl navíc jako frontendový „css” framework použit framework „Tailwind”, který byl zvolen pro pouhé zpříjemnění práce s „Css” a ne úplné nahrazení práce na designu.

Na backendové stránce byla využita služba „Supabase”, která se stará o databázi a další funkce jako například „webhooky”, „sql funkce” a další. K této službě navíc musel být použit framework „Flask”, který používá jazyk „Python”. Frameworku bylo potřeba, protože aplikace používá „docker” pro poskytnutí uživatelům „linuxové” prostředí. „Docker” nelze jinak ovládat než pomocí backendového frameworku. Navíc je potřeba k některým částem, které je lepší provádět v backendové části programu. „Python” je dále používán pro napsání „skriptu”, který je používán pro kontrolu kontejnerů.

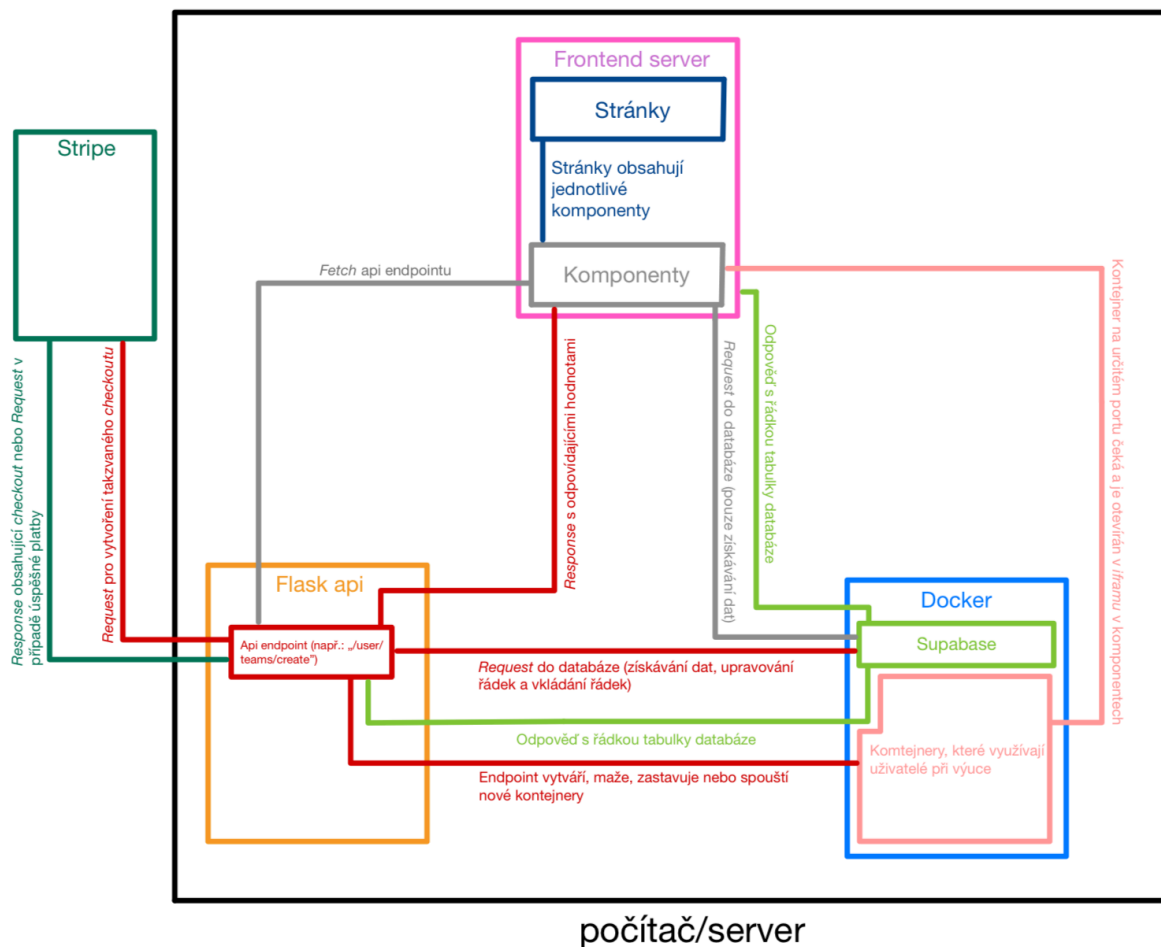
3 STRUKTURA PROJEKTU

Projekt je poměrně komplexní webovou aplikací, jak již plyne z kapitoly použitý software (viz. kapitola 2). Tudíž je i struktura, a to jak celý projekt zapadá do sebe poměrně komplexní a v této kapitole je vše rozvedeno a vysvětleno před tím než jsou v dalších kapitolách vysvětleny jednotlivé části celé aplikace do větších detailů.

Celý projekt lze rozdělit na dvě části a to sice backend (viz. kapitola 5), frontend (viz. kapitola 4) a jedno externí api. Takzvaný backend aplikace lze rozdělit na dvě další části a to “Flask” api a “docker”.

První z částí aplikace tedy frontend je prostředkem, kterým uživatel komunikuje s aplikací, tedy tvoří část, kterou si uživatel otevře v prohlížeči. Obsahuje jednotlivé stránky a je dělen na takzvané stránky (viz. kapitola 4.5) a jejich obsahy tedy komponenty (viz. kapitola 4.6), které tvoří vzhled stránky. Frontend souvisí nejprve s „Flask” api tím, že je hlavním zdrojem volání endpointů tohoto api a je jedinou částí, která od api dostává odpovědi. Další část, s kterou frontend souvisí, je služba „supabase” (viz. kapitola 5.3). Služba obsahuje databázi ze, které může frontend z bezpečnostních důvodů pouze získávat data, ale nemůže je upravovat. Poslední, s čím souvisí frontend jsou jednotlivé kontejnery uživatelů, ve kterých se vyučují a frontend obsahuje zobrazuje jejich příkazové řádky.

Druhou částí aplikace je backend, která jak již bylo zmíněno je rozdělena na dvě části. První částí je „flask” api, která souvisí s docker kontejnery skrze, které se uživatelé učí, tím že je může vytvářet, mazat, spouštět a zastavovat. Druhou souvislostí je manipulování s databází a získávání dat z databáze a poslední souvislostí je získávání takzvaného „checkoutu” z api služby „stripe” pro platby a zpracování úspěšných plateb. Druhou částí je “docker” a veškeré její souvislosti jsou již popsány výše.



Obr. 1: Graf struktury celého projektu

4 FRONTEND

V této kapitole je rozebrán veškerý grafický design aplikace. Tedy „html“ a „css“ stránky napsané za pomoci frameworku „tailwind“, a proč jsou tímto způsobem napsané. Je tu rozebrána i veškerá logika frontendu napsaná v jazyku „Javascript“ za pomoci frameworku „VueJs“, jako je funkčnost některých částí stránek a jednotlivých volání částí api (viz. kapitola 5.1) a zpracovávání odpovědí.

4.1 index.html

Framework „VueJs“ obsahuje tento soubor jako „html“ soubor, který je vždy zobrazen a pouze mění jeho obsah podle cesty, kterou si právě zobrazuje uživatel. Do tohoto souboru jsem proto přidal krátký kus kódu, který pokaždé když uživatel odejde z webové stránky na jinou změní titul stránky na „Come back :(and learn!“ a pokud se uživatel na stránku vrátí nastaví titulek stránky zpátky na původní.

4.2 Konfigurace Vue Routeru

Veškerá konfigurace takzvaných cest je definovaná v souboru „index.js” ve složce „router”. Tyto cesty jsou definované třemi nebo čtyřmi parametry a to jsou cesta v proměnné „path”, jméno „name”, komponenta (soubor s příponou „.vue”, obsahující logiku a design stránky, ze složky „views” (viz. kapitola 4.5)) „component” a některé cesty obsahují parametr „meta”, takzvané metadata, které umožňují definovat za jakých podmínek je možné se na stránku dostat. Je definováno pět druhů metadat a to jsou „requiresUnAuth”, „requiresAuth”, „requiresPremium”, „requiresTeam”, „requiresLessonOwner” a „requiresDesktop”. Některé cesty navíc obsahují takzvaný parametr, který je poté možné přechít v logice stránky a využít jej (definuje například sloupec „id” některé z řádek v databázi).

Pokud cestu v parametru „meta” tedy některá metadata obsahuje je volána metoda pro kontrolu příslušných metadat. Každá z těchto funkcí volá metodu na konci „next”, která určuje kam bude uživatel přesměrován podle výsledků kontroly. Metadata „requiresUnAuth” a „requiresAuth” kontrolují, zda není a je uživatel přihlášen respektive. „requiresPremium” kontroluje, zda má uživatel prémiový účet (ve sloupci „premium” v tabulce „user” má hodnotu true (viz. kapitola 5.3.3)), který je pro tuto stránku vyžadován. Metadata „requiresTeam” zjistí podle parametru v cestě, zda je uživatel součástí týmu, kterému je přiřazená lekce. „requiresLessonOwner” mají cesty, které vyžadují, aby jejich navštěvující uživatel byl majitelem řádky tabulky „lesson” (viz. kapitola 5.3.6) jejichž hodnota v sloupci „id” je v parametru této cesty. Poslední metadata „requiresDesktop” pouze kontrolují, zda přistupuje uživatel z mobilního zařízení a v případě, že ano zakáže mu přístup (GeeksforGeeks, 2023). (LearnVue, 2022)

4.3 Konfigurace „supabase”

Veškerá konfigurace propojení frontendu s backendovou službou „supabase” (viz. kapitola 5.3) je obsažena ve složce „supabase”, která obsahuje dva soubory a to sice „init” a „getFunctions”. První ze souborů (tedy „init”) obsahuje kód, potřebný pro inicializaci služby jako objekt, který je následně exportován pro další použití v aplikaci jako je čtení a zapisování dat do databáze, přihlašování uživatelů, používání takzvané realtime funkce služby. (Fireship, 2022)

Druhý soubor „getFunctions”, které se všechny zabývají získáváním dat z databáze. Většina frontend části aplikace používají tyto funkce pro čtení databáze pro jednodušší přístup. Navíc tento přístup umožňuje jednodušší kontrolu nad aplikací a zrychluje psaní kódu, neboť není potřeba vždy znovu psát stejný kus kódu. Typová metoda v souboru funguje tak, že nejprve zkontroluje, zda je zadán jeden z možných parametrů. Podle toho nastaví proměnné „queryField” (určuje podle, kterého sloupce v databázi se bude hledat) a „queryValue” (určuje hodnotu podle které se bude v databázi hledat). V případě že uživatel nezadal žádný parametr použije metoda většinou id aktuálního uživatele. Následuje už jen poslání příkazu do databáze a vrácení nalezených dat z databáze.

4.4 „Css” soubory

Veškeré „Css” třídy jsou rozděleny do několika souborů, které jsou dále rozděleny do složek. Názvy těchto složek odpovídají názvům složek, ve kterých jsou rozděleny jak takzvané „Views” (viz. kapitola 4.5), tak komponenty (viz. kapitola 4.6). Takto je projekt strukturalizován pro lepší čitelnost. V jednotlivých složkách jsou styly v některých případech ještě rozděleny do jiných souborů.

Názvy jednotlivých složek jako je například „commandline”, která obsahuje styly „Views” (viz. kapitola 4.5) a komponentů (viz. kapitola 4.6), které mají v názvu „commandLine” a tudíž obsahují stránku představující příkazovou řádku. Další složkou je „navbar”, která obsahuje styly dvou navigačních elementů jednoho na horní straně stránky a jednoho na dolní. Třetí složkou je „selector”, která obsahuje styly stránek představujících stránky složené z sektorů, které může uživatel vybrat a pomocí klávesnice „enter” použít tlačítko, které obsahují. Poslední složkou je „views” a ta obsahuje styly všech „Views” (stránek) (viz. kapitola 4.5).

Poslední důležitou částí, o které je třeba se zmínit je tmavý a světlý režim, který tato aplikace nabízí. Před některými třídami v celé aplikaci je přepínač „dark:”. Třídy pod tímto přepínačem jsou používány pouze tehdy když některý z kontejnerů (například element „div”), ve kterém jsou zabalené obsahuje třídu „dark”. V celé aplikaci je to vyřešeno způsobem, že tato třída je dynamicky přidávána a odebírána ze základního kontejneru (viz. kapitola 4.5). Odebírání a přidávání je prováděno ve skriptech komponentů se stylami příkazových řádek (viz. kapitola 4.6.2). (Szögyényi, 2021)

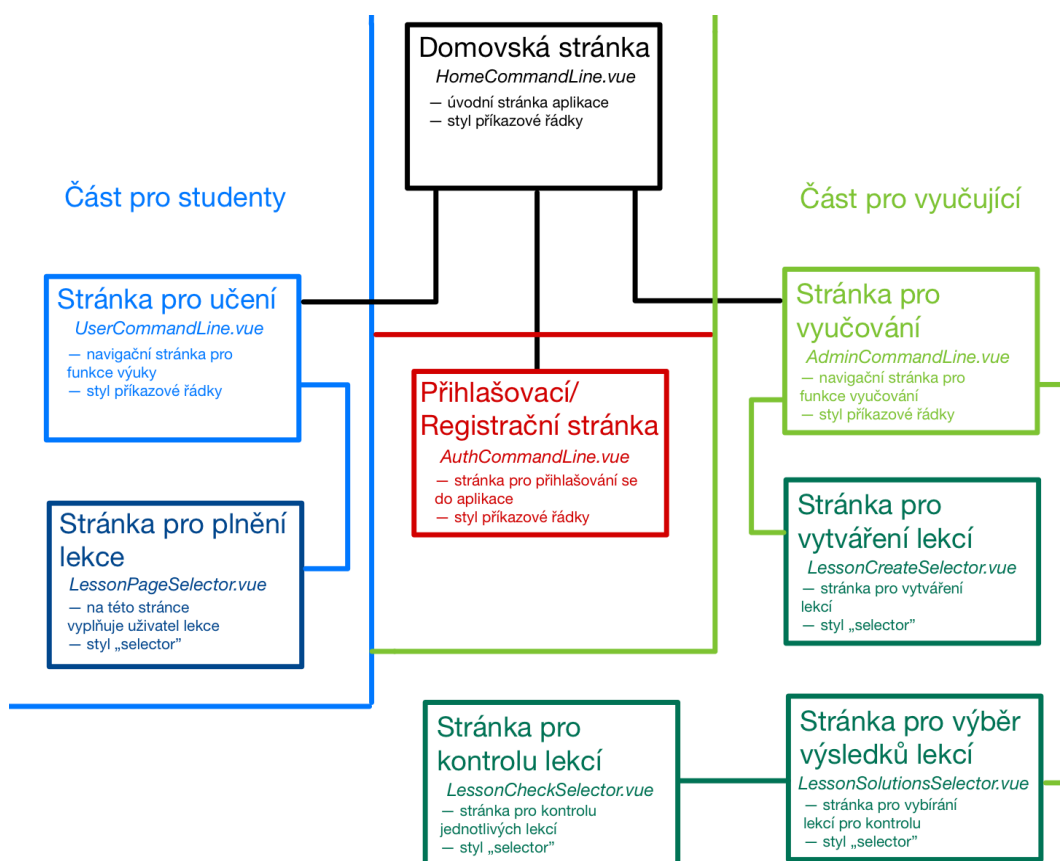
4.5 Stránky

Veškeré stránky jsou jako „css” soubory a Komponenty rozděleny do souborů jejichž jména odpovídají tomu, co soubory obsahují. To znamená že stránky s autentizací jsou ve složce s názvem „auth”, stránky obsahující platební portál a vše kolem něj a apod. Jediná stránka, která není ve své vlastní složce je „HomePage.vue”, která obsahuje domovskou stránku aplikace.

Všechny stránky (vyjma „LearningCreateLessonPage”, „LearningCheckContainerPage”, „LearningAdminPage”, „LessonPage” a „LessonSolutionsPage”) aplikace mají velice podobný obsah. Stránky, které nemají stejný obsah s výjimkou „LearningAdminPage” neobsahují styl příkazové řádky, ale nýbrž styl tabulky, takzvaného „selectoru”. Jejich „template” část obsahuje dva hlavní kontejnery první s třídou „title-container” a druhý s „main-container”. Kontejner „title-container” obsahuje element „pre” s třídou „ascii-container” a s id „ascii-art” (na některých stránkách je element „pre” obsáhnut dvakrát, a tudíž má dvě id „ascii-container1” a „ascii-container2”). Tento zmíněný element obsahuje ascii text vygenerovaný webovou stránkou (patorjk, n.d.) korespondující k obsahu stránky a jejich komponent. Dále jsou pod titulem obsáhnuty dva texty první se třídou „title-warning-text”, který obsahuje varování o obsahu stránky a druhý se třídou „title-ps-text”

obsahující dodatečné informace stránky. Následuje už jen oddělovač, který odděluje oba kontejnery, jediným obsahem druhého kontejneru potom je, odpovídající komponenta (viz. kapitola 4.6) importovaná ve skriptu.

„script” element každé stránky obsahuje, jak již bylo zmíněno „import” korespondující komponenty a dále je naimportován „hook” „onMounted”, který je spuštěn pokaždé co je stránka otevřena. V „hooku” je dále už jen jednoduchý „skript”, který vygeneruje pro titulek jednu, nebo dvě barvy, které nastaví jednotlivým titulům.



Obr. 2: Struktura stránek aplikace

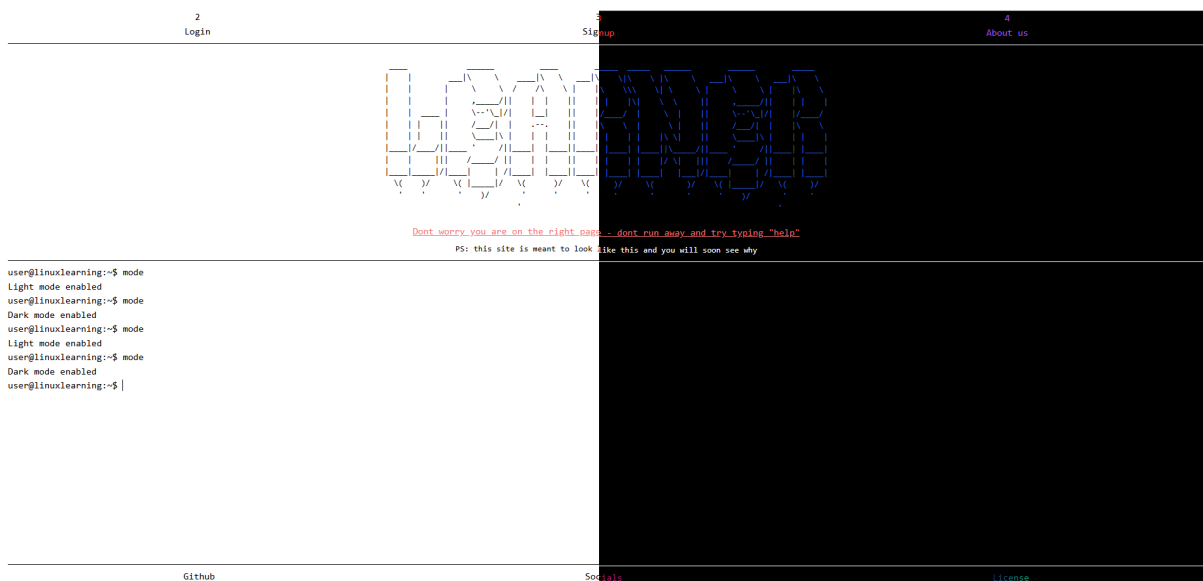
Struktura stránek (viz. obrázek 2) je rozdělena na tři části zahrnující tři různé funkce aplikace. První z nich je část pro přihlašování (na obrázku 2 označená červenou barvou), používá komponent „AuthCommandLine” (viz. kapitola 4.6.4) a slouží pro přihlašování a registraci do aplikace. Druhou částí je část pro samotné učení se a obsahuje dvě stránky využívající dvě komponenty. „UserCommandLine” (viz. kapitola 4.6.5) pro obecné informace o lekcích, přihlašování se do týmů a podobných a „LessonPageSelector” (viz. kapitola 4.6.13) pro vyplňování samotných lekcí. Poslední částí je ta pro vyučující, která obsahuje čtyři stránky s komponenty „AdminCommandLine” (viz. kapitola 4.6.4) pro používání základních funkcí a přehled informací o lekcích a týmech, „LessonCreateSelector” (viz. kapitola 4.6.9) pro vytváření nových lekcí a „LessonSolutionsSelector” (viz. kapitola 4.6.14) a „LessonCheckSelector” (viz. kapitola 4.6.10) oboje sloužící pro kontrolu lekcí. Toto je celková struktura a rozložení jednotlivých stránek v aplikaci, které tvoří celek aplikace.

4.6 Komponenty

Veškeré komponenty definovaný v souborech „vue” jsou jako Stránky a „css” soubory rozděleny do složek. Tyto složky jsou „commandline”, „navbar” a „selector” a obsahují soubory s obsahy odpovídajícími názvům složek. To znamená, že složka „commandline” obsahuje soubory obsahující „html” stránku vypadající jako příkazová řádka, složka „selector” obsahující soubory stylu „selector” a poslední složka „navbar” obsahující stylu navigační řádky. Styl „selector” je stránka složená z obdélníků a čtverců seřazených v listech a je navigovatelná klávesnicí.

4.6.1 Styly komponentů s příkazovými řádkami

Styly pro komponenty obsahující styl příkazové stránky jsou výrazně jednodušší než styly ostatních komponent. Proti tomu ale obsahují výrazně složitější skript. Prvním základní kontejnerem (elementem „div”) je kontejner se třídou „cmd-line-container”, který pouze nastavuje pozici, rozměr a rozložení jeho obsahu. Další v tomto kontejneru obsáhnutý kontejner je se třídou „cmd-output-container”, do kterého je přidáván text k příkazům zadaným do příkazové řádky (většinou chybový nebo informační). Na stejné úrovni následuje kontejner se třídou „cmd-line-input-container” a tento kontejner pouze srovná svůj obsah vedle sebe nastaví jeho šířku a barvu textu. Obsahuje jeden nadpis představující takzvaný „prompt” příkazové řádky a element „input” do kterého uživatel píše příkazy. Element „input” navíc poslouchá, zda uživatel při tom, kdy ho vyplňuje (píše příkaz) nezmáčkne „enter”, a v případě že toto nastane zavolá metodu „executeCommand”, která zpracovává příkazy (viz. kapitola 4.6.2). Poslední kontejner je „cmd-input-mobile-btn”, který je nastaven tak aby byl zobrazen jen když uživatel přichází z mobilního zařízení, protože tento kontejner obsahuje tlačítko na spuštění příkazu tak aby mohl uživatel spouštět příkazy bez klávesy „enter”. Toto tlačítko volá při zmáčknutí již výše zmíněnou metodu „executeCommand” (viz. kapitola 4.6.2).



Obr. 3: Vzhled komponentu se stylem příkazové řádky (levá část světlý režim a pravá tmavý režim)

4.6.2 Skripty komponentů s příkazovými řádkami

Všechny komponenty v základu fungují na stejném principu nahoře je definovaná konstanta „router” pomocí metody „useRouter” importované z knihovny „vue-router” pomocí které lze mimo jiné uživatele přesouvat aplikací. Dále jsou definovány čtyři proměnné důležité pro funkčnost příkazové řádky. První z nich je „cmdinput”, která obsahuje příkazy, které uživatel zadává do elementu „input”, jinými slovy do příkazové řádky. Druhou proměnnou je „executedCommand”, do které jsou postupně přidávány odpovědi spuštěných příkazů a je zobrazována v kontejneru „cmd-output-container” (viz. kapitola 4.6.1). Následují poslední dvě proměnné „executedList” a „executedIndex”, které slouží pro funkčnost používání již spuštěných příkazů (více níže).

Následuje metoda „onMounted” importována z frameworku „vue”, která vždy při zobrazení nastaví dva takzvané „EventListenery”. První z nich je volán potom co uživatel zmáčkne libovolnou klávesu a volá metodu „handleKeyDown”, druhý potom volá metodu „handleClick” a volá ji vždy když uživatel zmáčkne cokoli za pomoci myši. Jednodušší ze zmíněných metod tedy metoda „handleClick” a ta pouze nastaví uživatelský kurzor do elementu „input” (viz. kapitola 4.6.1) představující vstup příkazové řádky. Druhá metoda tedy „handleKeyDown” dostává parametr „event”, který obsahuje klávesu, kvůli které byla metoda zavolána. Dále už jen kontroluje metoda, jaké klávesy byly zmáčkнутy. Je několik případů, které kontroluje, a první z nich je kombinace kláves „ctrl” a „l”. V tomto případě je nejprve přeskočena funkce prohlížeče používána touto kombinací a poté vyčištěna proměnné „executedCommand” obsahující výsledky příkazů (toto provádí metoda „clearLines”). Další jsou klávesy „ArrowUp” a „ArrowDown”, které proměnnou „cmdinput” nastaví na jeden z předchozích příkazů. To funguje tak, že šipka nahoru prochází pole „executedList” od poslední přidané hodnoty do první a šipka dolů naopak (tyto hodnoty postupně mění v proměnné „cmdinput”).

Ve skriptu následuje nejdůležitější řídící metoda „executeCommand”, která je používána pro zpracování příkazů od uživatele. Nejprve se metoda podívá zda není proměnná „cmdinput” prázdná a pokud není tak přidá do listu v proměnné „executedList” tento příkaz a nastaví proměnnou „executedIndex” na délku listu příkazů a tímto je uložen použitý příkaz, který může uživatel nyní znovu použít. Následuje takzvaný „switch” příkaz, který dostává jako parametr proměnnou „cmdinput” a podle toho jaká je její hodnota provádí různé příkazy. První případy jsou, když je hodnota proměnné string „go ” a libovolné číslo. V takovýchto případech je uživatel za pomoci proměnné přemístěn na stránku jejíž jméno je napsáno v navigační menu a číslo, které udává za string „go ” je v navigačním menu pod jménem (viz. kapitola 4.6.15). Navíc v případě, pokud je string „go 6” v jakémkoliv komponentu tohoto typu je volána metoda „logout”, která jednoduše pomocí služby „supabase” uživatele odhlásí. Další dva případy jsou stringy „help” a „clear”. String „clear” pouze zavolá již výše zmíněnou metodu „clearLines” a string „help” zavolá metodu „commandOutput” s parametrem obsahujícím rady uživateli jaké příkazy může použít a jak stránku používat. Metoda „commandOutput” je vytvořena pro zjednodušení vypisování odpovědí příkazové řádky (přidává do proměnné „executedCommands”). Důležitým dalším případem je string „mode”, který odebere nebo přidá třídu „dark” k základnímu kontejneru stylů aplikace, čímž změní světlý režim na tmavý nebo naopak (Szógyényi, 2021). Ve „switchi” následují případy, kdy uživatel zadal příkaz „Github”, „Socials”, „License”, nebo „Get premium”. Pokud nastane případ jednoho z prvních tří je uživatel přesunut na stejnou jmennou stránku s „reklamním” profilem stránky. Pokud však nastane poslední případ je uživatel přesunut na stránku, kde si může pořídit prémiový účet (viz. kapitola 4.6.3). Následuje za vším už jen takzvaný „default” a ten jednoduše zavolá metodu „commandError” a ta nejprve zkontroluje, zda příkaz není prázdný a v takovém případě je odpověď příkazu pouze takzvaný „prompt”. Některé komponenty s příkazovou řádkou mají ještě jednu metodu, která je volaná v „defaultu” a obsahuje další případy pro příkaz.

4.6.3 Komponent *HomeCommandLine*

Tento komponent slouží jen jako uvítací stránka pro uživatele. Má proto jen jednoduché změny, z kterých první je změna proměnné „executedCommand” do které je přidán text pro informace nepřihlášených uživatelů. Druhá změna je v metodě „executeCommand”, kde je přidán jeden případ pro příkaz, a to sice pokud je roven stringu „buy premium”. V tomto případě je zavolána metoda „createCheckout” ta nejprve zkontroluje, zda aktuální uživatel nemá prémiový účet (v jeho řádce v tabulce „user” ve sloupci „premium” je boolean „true”). V případě že není je to pomocí metody „commandOutput” zodpovězeno uživateli. V opačném případě je zavolán api endpoint „/payments/create-stripe-session” (viz. kapitola 5.1.5). Ta se pokusí vytvořit takzvaný „checkout” pro uživatele a pokud úspěšně je uživatel přesměrován na tento „checkout” a může si zakoupit prémiový účet. (Coding Shiksha, 2022)

4.6.4 Komponent *AuthCommandLine*

Tento komponent je používán na stránkách pro přihlášení a registraci uživatele. V tomto komponentu jsou změněny jak styly tak skript. Ve stylech je kontejner se třídou „cmd-line-input-container” zabalen ještě do kontejneru „cmd-arrow-input” tak aby nad něj mohl být přidán text se speciálním textem ukazujícím uživateli v jakém kroku přihlášení je za pomoci proměnné „inputName”. V kontejneru „cmd-line-input-container” je navíc pozměněn takzvaný „prompt” na text obsahující pouze „>”. Pod kontejner „cmd-line-input-container” je ještě navíc přidán kontejner se třídou „cmd-password-strength-container”, který obsahuje vedle sebe srovnané čtyři elementy „p” (tyto elementy obsahují pouze text „#”), které mění barvu svého textu podle proměnné „strengthLevel”, tak aby bylo signalizováno, jak silné je uživatelem zadané heslo, když ho vytváří. Celý tento kontejner je navíc zobrazena jen když uživatel zadává heslo.

První změnou skriptu tohoto komponentu je v proměnných inicializovaných na začátku souboru první z nich je „inputName”, která je dále používána na indikování uživateli, jaká část přihlášení je očekávána a ze začátku je nastavena na string „Email:”. Následují proměnné „credentials” do které jsou ukládány uživateli přihlašovací nebo registrační údaje a proměnná „currentStep” ve které se ukládá kolikátý krok přihlašování nebo registrace uživatel provádí. Poslední proměnná je „showPassword”, která elementu „input” do kterého uživatel zadává příkazy ukazuje, zda má ukazovat text, nebo ho skrývat jako tečky v případě, že uživatel vyplňuje heslo.

Další změna je v metodě „executeCommand” a to sice na konci „switchu” v přepínači „default” a to v tom že teď volá vždy metodu „processCredential” a nepočítá s chybou příkazu. Tato metoda obsahuje také takzvaný „switch” tentokrát s parametrem proměnné „currentStep”.

V případě, že je hodnota proměnné „currentStep” 0 je volána metoda „saveEmail”. Metoda „saveEmail” nejprve použije jiné metody a to sice „testEmail” a podívá se, zda hodnota toho co uživatel zadal jako email odpovídá formátu emailu pomocí komplexního regexu (KingKongFrog, 2013). Pokud není vypíše uživateli do proměnné „executedCommands” předešlý „prompt” a chybovou hlášku. V případě že je email platný pouze do proměnné „executedCommands” opíše předchozí „prompt” a email, který uživatel zadal, následovně na první index pole „credentials” zadaný platný email, zvětší proměnnou „currentStep” o jedna a nakonec změni nadpis „promptu” v proměnné „inputName” na string „Password”, čímž posune uživatele na další část přihlašování/registrace.

Další případ, který může v metodě „processCredential”, nastat je, že je proměnná „currentStep” nastavena na 1 nebo 2. V obou případech je volána metoda „savePassword” v této metodě jsou dva postupy, pokud je proměnná „currentStep” 1 tak postupuje program tak že se nejprve podívá zda je cesta v prohlížeči „/login” (v takovémto případě není potřeba kontrolovat sílu hesla, neboť se uživatel pouze přihlašuje) a nebo pokud je hodnota proměnné „strengthLevel” 5 (podívá se zda je heslo pro nový účet dostatečně silné).

Proměnná „strengthLevel” je nastavována pomocí „hooku” „computed” importovaného z frameworku „vue”, který ji udržuje neustále aktuální, a určuje sílu zadávaného hesla. Hodnoty této proměnné mohou být od 1 do 5 a to sice podle další proměnné „passwordStrength” nastavované také pomocí „hooku” „computed”. Proměnná „passwordStrength” obsahuje jinou hodnotu než -1 jen v případě, že je proměnná „currentStep” nastavena na 1. Pokud je použit algoritmus pro evaluaci hesel získaný z (Webnoob, 2019) (viz. Obr. 4).

```
const passwordStrength = computed(() => {
  let score = -1
  if (currentStep.value == 1){
    score = 0

    let letters = {}
    for (let i = 0; i < cmdinput.value.length; i++) {
      letters[cmdinput.value[i]] = (letters[cmdinput.value[i]] || 0) + 1
      score += 5.0 / letters[cmdinput.value[i]]
    }

    let variations = {
      digits: /\d/.test(cmdinput.value),
      lower: /[a-z]/.test(cmdinput.value),
      upper: /[A-Z]/.test(cmdinput.value),
      nonWords: /\W/.test(cmdinput.value),
    };

    let variationCount = 0;
    for (let check in variations) {
      variationCount += (variations[check] === true) ? 1 : 0;
    }

    score += ((variationCount - 1) * 10);
  }

  return parseInt(score)
})
```

Obr. 4: Skript pro kontrolu síly hesla

Nyní pokud projde jedna z podmínek v metodě „savePassword”, když je proměnná „currentStep” 1. Je do proměnné „executedCommands” přidán „prompt” a příkaz (v tomto případě heslo) změněný pouze na string složený ze znaků „•”, tak aby nebylo zobrazeno heslo. Dále je nově zadané heslo uloženo do pole „credentials” proměnná „currentStep” zvětšena o jedna (stejně jako v metodě „saveEmail”) a proměnná „inputName” indikující informace, které má uživatel zadat dále změněna na „Repeat Password:”. Pokud výše zmíněná podmínka není splněna uživatel dostane chybovou hlášku indikující, že heslo musí změnit.

Jak již změna proměnné „inputName” napovídá druhý případ v metodě „savePassword” (tedy pokud je proměnná „currentStep” 2) je kontrolováno zda je zopakované heslo (hodnota proměnné „cmdinput”) stejné jako hodnota pole „credentials” na indexu 1 (tedy jestli zopakované heslo je stejné jako to původní) a pokud je následuje stejný proces jako, v případě kdy byla proměnná „currentStep” 1 s jediným rozdílem a to sice že nový nadpis „promptu” je string „Are you sure you want to log in [Y/n]:” (uložený v proměnné „inputName”). I v tomto

případě pokud hesla nejsou stejná dostane pomocí proměnné „executedCommands” uživatel odpověď, že hesla nejsou stejná.

Poslední případ v metodě „processCredentials” je pokud je proměnná „currentStep” 3 a v tomto případě je pouze volána metoda „auth”. Metoda „auth” nejprve zjistí, zda uživatel zadal do proměnné „cmdinput” string „Y” (tedy souhlasí s přihlášení/registrací). Pokud ne pouze znovu načte celou stránku a uživatel se musí přihlásit nebo registrovat znovu, ale pokud ano tak metoda ještě zjistí, zda je komponent využíván pro přihlášení nebo registraci pomocí cesty zadané v prohlížeči a zavolá metodu „login” (pro přihlášení), nebo „signup” (pro registraci). V metodě „login” se program nejprve pokusí uživatel přihlásit (pomocí služby „supabase”) s údaji zadanými v poli „credentials” a pokud program vrátí chybu zavolá metodu „errorMessage”, která vrátí celý komponent do „původního” stavu a pouze uživateli za pomocí proměnné „executedCommands” vypíše chybu, že uživatel neexistuje. Pokud je však uživatel úspěšně přihlášen je přesunut na domovskou stránku webové aplikace. V metodě „signup” aplikace zaregistruje uživatele pomocí služby „supabase” a přesune ho na domovskou stránku webové aplikace. Služba uživateli dále pošle email s odkazem na verifikaci účtu, který pokud následuje verifikuje účet a přihlásí uživatele.

4.6.5 Komponent *UserCommandLine*

Tento komponent se stylem příkazové řádky je používán uživateli pro vybírání lekcí k splnění, přihlašování se do týmů, odpojování se od týmů, a navíc obsahuje přehled nevyplněných lekcí a týmů, kterých je uživatel součástí. Tato stránka slouží pro normální uživatele, kteří nevytváří lekce ani týmy, a tedy nemají koupený prémiový účet (v tabulce „user” jejich řádek ve sloupci „premium” obsahuje hodnotu false (viz. kapitola 5.3.3)).

Styly tohoto komponentu nejsou nijak upravovány, ale za to je na konci metody „executeCommand” další metoda obsahující více případů pro proměnnou „cmdinput” (příkazy). Tato metoda je pojmenována „adminCommands” a jediné čím se liší od metody „executeCommand” změněné je že „switch” přepínač má jako parametr „true” a příkaz v proměnné „cmdinput” je porovnáván s chtěným výsledkem a pokud je to pravda provede příkaz. První možnost je pokud příkaz začíná stringem „team join” a v tomto případě zavolá metodu „joinTeam”. Tato metoda nejprve část příkazu „team join” prázdným stringem a pokud tento nový string není prázdný zavolá api endpoint „/teams/user/join” (viz. kapitola 5.1.4), který aktuálního uživatele přihlásí do týmu, jehož kód je ve zbylém. Nakonec metoda zavolá metodu „commandOutput” s parametrem odpovědi v bodě „msg”.

Další případ v metodě „adminCommands” je, pokud zadaný příkaz začíná stringem „team leave” v takovém případě je volána metoda „leaveTeam”, která stejně jako metoda „joinTeam” nahradí část stringu „team leave” prázdným stringem a pokud tento string není prázdný zavolá metoda api endpoint „/teams/user/leave” (viz. kapitola 5.1.4) opět se zbylým stringem jako parametrem a ten uživatel odstraní z týmu jehož jméno je ve zbylém stringu. Stejně jako první metoda zavolá poté metodu „commandOutput” s parametrem odpovědi api v bodě „msg”, aby uživatel viděl výsledek příkazu.

Třetí možnost v metodě „adminCommands” je, pokud je příkaz roven stringu „team ps” a v takovém případě je volána metoda „printTeams”. V takovém případě využije metoda metody „getTeam” importované ze souboru „getFunctions” (viz. kapitola 4.3) a tím že zavolá metodu bez parametru metoda vrátí z databáze veškeré týmy, kterých je aktuální uživatel členem. Následně všechny týmy program vypíše a pokud uživatel není členem žádného týmu tak logicky vypíše, že uživatel není členem žádného týmu. Veškeré výpisy jsou provedeny pomocí metody „commandOutput”. Tímto uživatel zjišťuje, jakých týmů je členem a jaká jsou jejich jména, aby je mohl opustit.

Předposlední z možností, které mohou nastat je pokud je příkaz roven stringu „lesson ps” a v takovém případě je volána metoda „printLessons”. Tato metoda nejprve jako předchozí za pomoci importované metody ze souboru „getFunctions” (viz. kapitola 4.3) získá veškeré týmy, kterých je uživatel součástí (opět vypíše chybu, pokud nenajde žádné pomoci „commandOutput”). Pokud některé týmy najde pro každý z nich najde v databázi za pomoci funkce „getLesson” importované ze souboru „getFunctions” zadané lekce a vypíše je za pomoci metody „commandOutput”. Uživatel takto zjistí jaké má, pro jaký tým zadané lekce a podrobné informace k nim.

Poslední případ, který může v metodě „adminCommands” nastat, je, pokud příkaz začíná stringem „lesson do” a pokud ano je volána metoda „doLesson” a příkaz je užíván pro vyplňování lekce. Metoda nejprve opět nahradí část příkazu „lesson do ” prázdným textem a pokud nezůstane prázdný string pokračuje dále. Druhá kontrola je pokud nepoužívá uživatel mobilní zařízení v takovém případě aplikace vypíše za pomoci metody „commandOutput” zákaz používání této funkce pomocí mobilního zařízení. Zda uživatel používá mobilní zařízení je zjišťována pomocí kódu získaného z (GeeksforGeeks, 2023). Následovně je využito dvou metod importovaných ze souboru „getFunctions” (viz. kapitola 4.3) a to sice „getLesson” a „getTeam” pomocí „getLesson” je získávána pomocí zbylého stringu ze začátku metody jako jménem lekce a jsou získány týmy, kterých je uživatel členem. Dalším krokem je zkontrolování, zda je lekce, jak je očekáváno pouze jedna a v opačném případě vypsání pomocí metody „commandOutput” chybové hlášky. Pokud kontrola projde následuje kontrola, zda aktuální uživatel není tvůrcem této lekce, protože v takovém případě ji nemůže vyplňovat a pokud i to projde je za pomoci procházení všech týmů uživatele zkontrolováno zda je uživatel členem týmu pro, který je lekce vytvořena. Nakonec pokud vše prošlo je zkontrolováno, zda časový úsek pro vyplnění lekce již nevypršel, nebo pokud lekce ještě nezačala a v opačném případě je uživatel přesunut na stránku pro vyplňování lekce (viz. kapitola 4.6.13). Veškeré výše zmíněné podmínky mají i opačný případ ve kterém je volána metoda „commandOutput” a vypsáno uživateli co nastalo za chybu.

4.6.6 Komponent *AdminCommandLine*

Tato komponenta slouží pro vytváření týmů, mazání týmů, mazání lekcí a podobný takzvaným „admin” akcím, které se týkají učení ostatních uživatelů. Styly tohoto komponentu jsou opět stejné jako v případě předchozího komponentu „UserCommandLine” (viz. kapitola 3.6.5) a stejně jako u tohoto komponentu obsahuje tato metoda navíc ještě na konci metody

„executeCommand” navíc metodu „adminCommands” fungující na stejném principu jako metoda „adminCommands” v komponentu „UserCommandLine” (viz. kapitola 4.6.5).

První možnost ve „switchi” metody „adminCommands” je, pokud začíná string příkazu „team create” je volána metoda „createTeam”. Tato metoda pouze vymění část stringu příkazu „team create ” za prázdný string a pokud string není prázdný zavolá api endpoint v cestě „/teams/admin/create” (viz. kapitola 5.1.4), který vytvoří nový tým se jménem, kterým je změněný string. Nakonec správu v bodě „msg” odpovědi api zobrazí pomocí metody „commandOutput”. Pokud změněný string byl prázdný pouze uživateli pomocí metody „commandOutput” zobrazí chybovou hlášku o tom, že jméno nebylo zadáno.

Další možnost v metodě „adminCommands” je pokud je příkaz roven stringu „team ps” a v tomto případě zavolá metodu „printTeams”. Metoda nejprve získá týmy, kterých je aktuální uživatel tvůrce. Pokud žádné týmy nejsou nalezeny jednoduše to program uživateli nahlásí opět pomocí metody „commandOutput”. V opačném případě vypíše metoda veškeré nalezené týmy a jejich kódy. Tento příkaz používá uživatel pro vypsání všech ním vytvořených a spravovaných týmů.

Následovný případ v metodě „adminCommands” je, pokud příkaz začíná stringem „team delete”. Při takovém případě je volána metoda „deleteTeam”. Tato metoda funguje na stejném principu jako většina metod volající api endpoint. Nejprve nahradí část příkazu „team delete” prázdným stringem poté pokud nový string existuje zavolá api endpoint s parametrem stringu v bodě „/teams/admin/delete” (viz. kapitola 5.1.4), která smaže tým, jehož jméno je napsáno v zmíněném parametru, a pomocí metody „commandOutput” vypíše návratovou hodnotu v bodě „msg”. Pokud string neexistuje pouze vypíše chybovou hlášku pro uživatele, aby věděl že nezadal platné jméno.

Čtvrtá možnost pro příkaz v metodě „adminCommands” je pokud začíná stringem „team users”. Tento uživatel využívá uživatel, pokud chce vypsát všechny uživatele ve svém týmu. Pokud nastane taková možnost je jako vždy zavolána metoda a to specificky „printTeamUsers”. Zmíněná metoda nejprve nahradí string příkazu v části „team users ” prázdným stringem. Pokud je tento string prázdný vypíše chybu, že nebylo zadáno jméno pomocí metody „commandOutput”. Pokud bylo jméno však zadáno, podle tohoto jména a id aktuálního uživatele následovně metoda získá tým z databáze (viz. kapitola 4.3), který uživatel vytvořil a zadal jeho jméno. Jestliže takovýto tým existuje najde v databázi všechny uživatele, kteří jsou členy tohoto týmu (viz. kapitola 4.3) (thorwebdev, 2023). Nakonec už jen pomocí metody „commandOutput” vypíše veškeré uživatele nalezeného týmu. Veškeré chyby jako nenalezení žádných uživatelů a podobné jsou samozřejmě ošetřeny a nahlášeny uživateli pomocí metody „commandOutput”.

Poslední příkaz týkající se týmu definovaný v metodě „adminCommands” je pokud příkaz začíná stringem „team kick”. Příkaz v tomto formátu je využíván pro odstranění jiného uživatele z týmu. V tomto případě je volána metoda „kickUser” a ta nejprve jako většina ostatních nejprve vymění část příkazu „team kick ” za prázdný string. Jelikož tento příkaz vyžaduje dva parametry je potřeba ošetřit to aby nový string nebyl prázdný, nebo aby nebyl

roven stringu „team kick”. Pokud toto projde je zbytek stringu, který je očekáván, aby byl dvě slova, rozdělen na dvě slova (Trott, 2012). Program už jen zkontroluje, zda nové dvě části existují a jestliže ano zavolá api endpoint s parametry těchto dvou částí v cestě „/teams/admin/kick” (viz. kapitola 5.1.4), který vyhodí uživatele definovaného v parametrech. Nakonec jako vždy je uživateli pomocí metody „commandOutput” ukázána správa v bodě „msg” odpovědi databáze. Opět musí být ošetřeny veškeré chyby s dělením příkazu, aby uživatel věděl, jakou část zadal špatně (docíleno pomocí metody „commandOutput”). První z příkazů v metodě „adminCommands” týkajících se lekcí je „lesson create”, který je používán, když chce uživatel vytvořit novou lekci. V případě tohoto příkazu volá komponent metodu „createLesson”. Metoda nejprve získá z tabulky „limitations” (viz. kapitola 5.3.4) objekt aktuálního uživatele a pokud je podle předpokladů tento objekt nalezen pouze jeden program pokračuje. Program zkontroluje, zda uživatel vytvořil některé týmy a pokud ještě může vytvořit některé lekce (uživatel může vytvořit jen omezený počet lekcí (viz. kapitola 5.3.4)). V případě, že všechny tyto podmínky projdou je uživatel přesměrován na stránku (pomocí objektu v proměnné „router”), kde může lekci vytvořit (viz. kapitola 5.6.13). Opět jsou všechny výjimky a chyby ošetřeny a pomocí metody „commandOutput” nahlášeny uživateli.

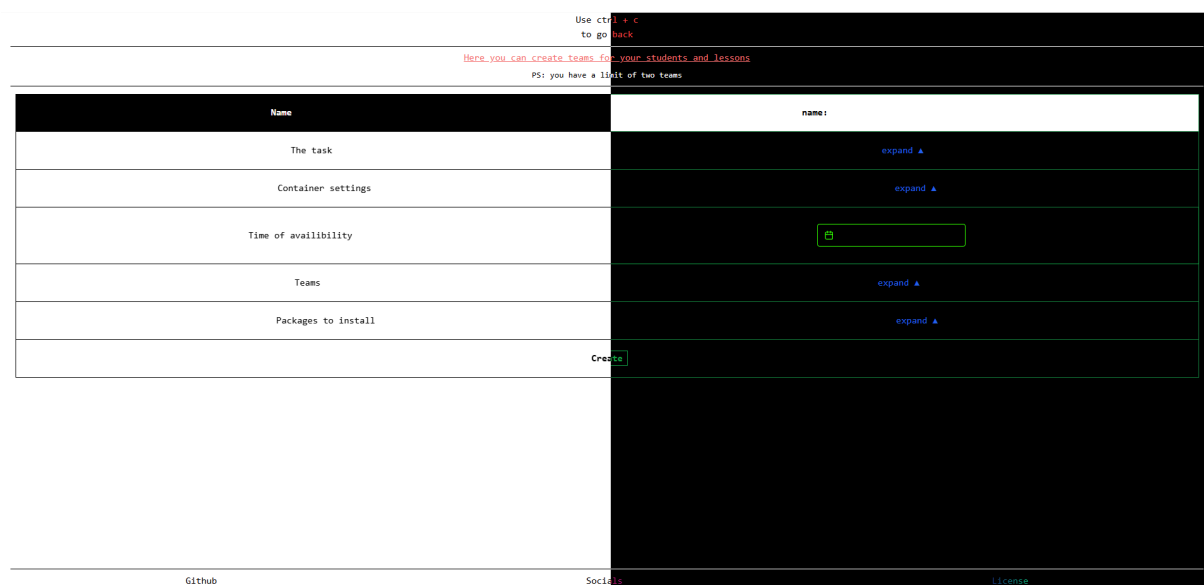
Další příkaz v metodě „adminCommands” je pokud string příkazu začíná „lesson cancel ” a pokud ano je volána metoda „cancelLesson” a tento příkaz využívá uživatel, když chce zrušit lekci. „cancelLesson” nejprve začne tím, že nahradí část stringu příkazu „lesson cancel ” prázdným stringem a pokud tento nový string není prázdný zavolá api endpoint „/lessons/admin/cancel” (viz. kapitola 5.1.3) (opět s parametrem upraveného stringu), který zruší lekci specifikovanou v parametrech. Pomocí metody „commandOutput” je pak vypsána návratová hodnota v bodě „msg”. Pokud nový string je prázdný je uživateli pomocí metody „commandOutput” vypsána pouze chyba.

Předposlední možnost pro příkaz v metodě „adminCommands” je pokud je příkaz roven stringu „lesson ps” a ten uživatel používá pro vypsání lekcí, které vytvořil. Pro tento případ je volána metoda „printLesson”, která nejprve pomocí metody „getLesson”, která je volána bez parametrů a je importována ze souboru „getFunctions”, získá všechny objekty lekce, které vytvořil uživatel z databáze (viz. kapitola 4.3). Následovně veškeré tyto objekty vypíše ve pro uživatele přehledném formát s veškerými důležitými parametry.

Posledním případem příkazu v této metodě je pokud příkaz začíná stringem „lesson solutions” (tento příkaz používá uživatel pro zobrazení vyplněných lekcí) v takovém případě je volána metoda „seeSolutions”. Metoda „seeSolutions” opět upraví příkaz v části „lesson solution ” za prázdný string. Pokud nový upravený string existuje je s jeho pomocí z databáze získán model lekce (viz. kapitola 5.3.4), který aktuální uživatel vlastní a jeho jméno je upravený string. V případě, pokud tato lekce existuje je uživatel přesunut na stránku, kde uživatel může zkontrolovat vyplněné lekce (viz. kapitola 4.6.10). Jestliže nový string neexistuje je uživateli nahlášena pomocí metody „commandOutput” chyba.

4.6.7 Styly jedno sloupových „selector” komponentů

Jednodušší ze dvou druhů „selector” stránek jsou stránky s jedním sloupcem. Jsou strukturalizovány tak, že jejich ovládací a informační prvky jsou pod sebou poskládány v jednotlivých „div” elementech se třídami „selection-item”. Třída nastavuje výšku jednotlivých obdélníků srovnání jejich obsahu tak aby byl pod sebou a okraje tak aby byl vytvořen výsledný styl. Všechny tyto kontejnery jsou obsáhnuty v kontejneru „div” se třídou „selections-container”. Tato třída nastavuje mezery od okrajů, srovnání jednotlivých již zmíněných kontejnerů, barvu textu, font a podobné detaily. Již zmíněné kontejnery se třídou „selection-item” následovně už obsahují různé elementy jako jsou nadpisy v elementech „p” ovládací prvky v (elementy „button”). Některé kontejnery „selection-item” obsahují navíc vedle sebe (v „html” souboru na stejné úrovni) další kontejner se třídou „selection-item-details” (kontejnery jsou používány pouze v souboru „LessonCreateSelector.vue”). Tyto kontejnery obsahují, jak název napovídá detailnější informace k informacím v kontejneru „selection-item”. Třída „selection-item-details” v kontejneru pouze upravuje, jak budou jednotlivé prvky srovnány (v tomto případě pod sebe) a okraje a jejich barvu. V některých kontejnerech „selection-item-details” jsou obsáhnuty další seznamy podobné celé stránce. Jsou celé zabalené v kontejneru „item-detail-checkbox-container”, který definuje, že mají být seřazeny pod sebou, centruje je a podobně dosahuje vzhledu stránky. V kontejneru jsou dále obsáhnuty další, které mají třídu „item-details-checkbox”, která už jen nastaví, jak mají tyto prvky seznamu vypadat. Posledním kontejnerem je kontejner se třídou „selections-error” a obsahuje chyby, kterých se uživatel může dopustit při používání stránky (vytváření lekcí).



Obr. 5: Vzhled komponentu se stylem jedno sloupový „selector” (levá část světlý režim a pravá tmavý režim)

4.6.8 Skript jedno sloupových „selector” komponentů

Ve vrchní části každého komponentu je definována proměnná „router” pomocí zavolání metody „useRouter” importované z knihovny „vue-router” a další proměnné potřebné pro funkčnost komponentů. Hlavní částí, na kterém stojí celý komponent je metoda „handleKeyDown”. Tato metoda je pomocí metody „onMounted” (metoda importována z frameworku „VueJs” spuštěna vždy při zobrazení Komponentu) nastavená jako takzvaný „EventListener” a je volána vždy, když uživatel zmáčkne klávesu nebo provede jinou akci. Metoda „handleKeyDown” bere jako parametr objekt „event”, který obsahuje akci, kterou uživatel provedl, na základě těchto akcí je potom ovládána stránka.

Základní klávesy pro používání této komponentu jsou klávesa nahoru („ArrowUp”) a dolů („ArrowDown”). Při stisknutí této klávesy je nastavena proměnná „currentIndex” a proměnná „displayDetails” na false. Proměnná „currentIndex” je nastavena na o jedna menší hodnotu, pokud je zmáčknuta klávesa „ArrowUp” a o jedna větší pokud „ArrowDown”. Navíc pokud je currentIndex na poslední hodnotě povolené délky, nebo na začátku a podle toho nastaví hodnotu „currentIndex” na první nebo poslední hodnotu. Tímto způsobem je ovládána uživatelem stránka a je posunuta zvýrazněná část stránky (viz. obrázek 5). Klávesa „enter” je používána uživatelem pro použití funkce, která je popsána v kontejneru „selection-item” (zobrazení kontejneru „selection-item-details”, vytvoření lekce atd.). Poslední je klávesová kombinace „ctrl” a „c”, která pouze pomocí v horní části skriptu inicializované proměnné „router” přesune uživatele na stránku s cestou „/learning/admin”.

4.6.9 Komponent *LessonCreateSelector*

Tento komponent obsahuje styl „selector” a je využívána pro tvorbu nových lekcí. Proměnné skriptu na začátku jdou rozdělit do tří kategorií. První kategorie proměnných obsahuje proměnné, které vyplnil uživatel v jednotlivých polích pro vytváření lekcí. Druhá kategorie se věnuje pouze chybám, kterých se uživatel může dopustit při vytváření lekcí a poslední kategorie jsou proměnné používané pro veškeré ovládání listů a navigaci webovou aplikací.

Ve skriptu následuje metoda „handleKeyDown”, která je upravena pro účely tohoto komponentu. První úpravou je, že klávesa „enter” ve všech případech až na jeden pouze mění hodnotu proměnné „displayDetails” na její opačnou hodnotu a podle této proměnné je určováno, zda má být zobrazen kontejner „selection-item-details” na zvoleném indexu. Na druhou stranu je klávesa „escape” používána pro měnění hodnoty proměnné „displayDetails” na hodnotu „false”. Již zmíněnou výjimkou při zmáčknutí klávesy „enter” je pokud je proměnná „currentIndex” nastavena na hodnotu 6. V tomto případě je volána metoda „createLesson”.

Metoda „createLesson” nejprve vytvoří pythonový objekt „dictionary” do proměnné „error” ve které jsou uloženy veškeré druhy chyb, kterých se uživatel může dopustit při vytváření lekce. Následovně jsou postupně zkontrolovány jednotlivé části formuláře, který celá stránka tvoří a pokud je některá část správně nastavena nastaví jednu z hodnot proměnné „error” na „false” (veškeré hodnoty jsou zprvu nastaveny na „true”). Na kontrolu jsou využívány

„regexi“ tak aby mělo jméno (Hadzhiev, 2024) (McCrossan, 2012) a zadání správný tvar a pomocí jednoduchých podmínek je kontrolováno, zda je vše potřebné vyplněno a podle toho je nastavena proměnná „error“. Aby uživatel dostal zpětnou vazbu je do globální proměnné „errors“ nastaven aktuální stav proměnné „error“. Podle této proměnné je následovně zobrazován obsah kontejneru „selections-error“ a uživatel tímto způsobem dostane zpětnou vazbu. Nakonec je kontrolováno, zda obsahuje proměnná „error“ některou hodnotu nastavenou na „true“ a pokud ne zavolá metodu „create“ a ta už pouze zavolá endpoint api „/lessons/admin/create“ (viz. kapitola 5.1.3). Na závěr pokud je hodnota vrácená endpointem api v parametru „status“ nastavena na „false“ nastaví proměnnou, která určuje chybu api na „true“ a proměnnou „apiErrorMsg“, která obsahuje zprávu pro uživatele s chybou api, na hodnotu vrácenou api v parametru „msg“.

Další z přidanych kombinací kláves do metody „handleKeyDown“ je „ArrowUp“, nebo „ArrowDown“ a klávesa „ctrl“. Tato kombinace je používána pro ovládání podlistů obsáhnutých v kontejnerech „item-detail-checkbox-container“. Na stejném principu poté funguje kombinace kláves „ctrl“ a „enter“, která je používána pro vybírání „checkboxů“ v již zmíněném kontejneru. Tato kombinace provádí různé akce na základě toho, zda je hodnota proměnné „currentIndex“ 2,4, nebo 5.

V první případě, který nastane, když je hodnota proměnné „currentIndex“ na 4, je volána metoda „selectTeamOptions“, která jednoduše nastaví hodnotu proměnné „teamSelected“ (tato proměnná obsahuje id týmu (viz. kapitola 5.3.5) a je využívána při vytváření lekce) na hodnotu „id“ seznamu „adminTeams“ na indexu hodnoty proměnné „subIndex2“.

4.6.10 Komponent *LessonCheckSelector*

Tento jedno sloupcový komponent je výrazně jednodušší než předešlí („LessonCreateSelector“ (viz. kapitola 4.6.9)), protože je využíván pro výrazně jednodušší potřeby. Konkrétně je využíván ke kontrole již uživateli vyplněných „docker“ kontejnerů (vyplněných lekcí).

Proměnné definované na začátku tohoto souboru jsou pouze proměnné pro funkčnost zmíněného listu. Důležité proměnné přidané jsou „url“, která obsahuje url na kterém běží grafický interface „docker“ kontejneru. Toto url je použito v html elementu „iframe“, přes který následně používá uživatel „docker“ kontejner. Další důležitá proměnná „container“ obsahuje objekt „docker“ kontejneru z databáze (viz. kapitola 5.3.7). Tato proměnná společně s dalšími je nastavena v metodě „setVariables“, která je volána vždy při zobrazení komponentu. Další proměnné nastavené v této metodě jsou „siteState“, která určuje aktuální stav stránky podle, které je zobrazován obsah posledního kontejneru „selection-item“ a je nastavena na string „running“ pokud je hodnota proměnné „container“ v bodě „running“ nastavena na „true“ a znamená to, že „docker“ kontejner běží nebo je zastaven. Dále už jsou nastaveny jen další méně důležité proměnné.

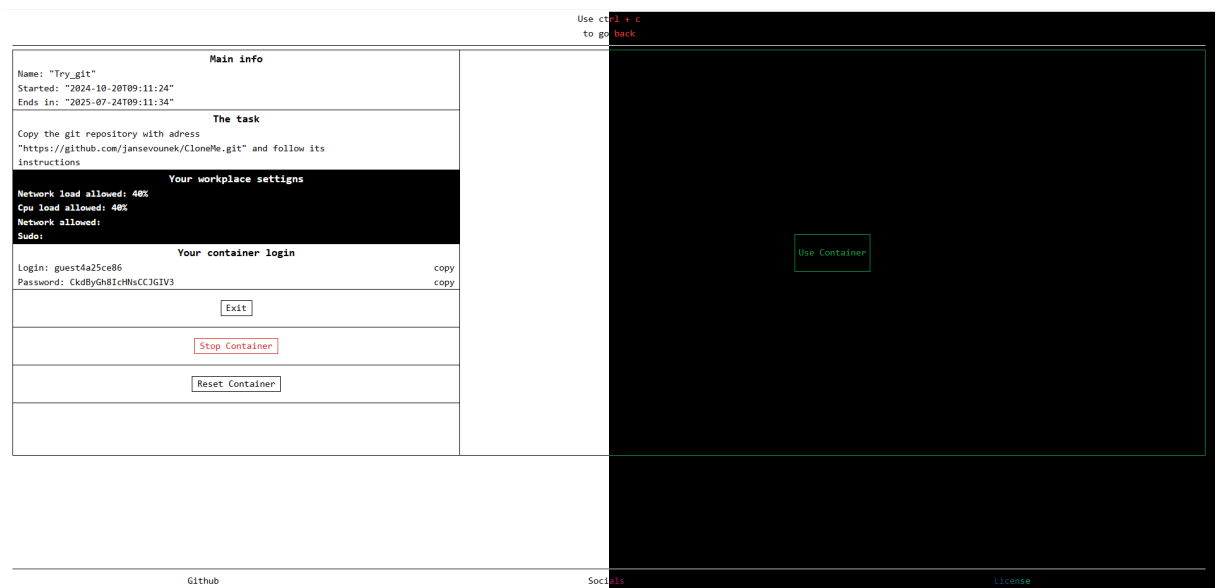
Další úvodní částí skriptu je takzvaný „subscribe“, který je funkcí služby „supabase“ (viz. kapitola 5.3). Tato funkce volá metodu v kódu komponentu „handleRunningChange“, který

kontroluje, zda objekt docker kontejneru v databázi (viz. kapitola 5.3.7), který uživatel využívá nebyl změněn (kontejner byl zastaven) a pokud ano změni proměnnou „siteState” na string „container_exists”, aby uživatel neviděl chybu, že kontejner není dostupný.

Metoda „handleKeyDown” nemá v tomto případě žádné klávesy navíc pouze upravuje co se stane při zmáčknutí klávesy „enter”. Pokud uživatel zmáčkne klávesu „enter” a proměnná „currentIndex” je nastavena na 1 je uživatel pomocí metody „back” proměnné „router” přesunut na předchozí stránku. Když je hodnota proměnné „currentIndex” 4 je volána metoda „startContainer”, která volá endpoint api „/lessons/admin/start-container” (viz. kapitola 5.1.3), který spustí kontejner, a pokud je návratová hodnota v proměnné „status” nastavena na „true” provede následující kroky. Nejprve nastaví proměnnou „siteStatus” na string „starting” poté počká 9 sekund (toto je provedeno, aby uživateli nebyla zobrazena chyba iframu) (Dascalescu & blackgreen, 2009) a nakonec na string „running”. V opačném případě pouze nastaví proměnnou „error_msg” na chybu, kterou vrátilo api a tato chyba je zobrazena v kontejneru „selections-error”. Poslední případ je když je proměnná „currentIndex” nastavena na 3 a v tomto případě je opět voláno api na endpointu „/lessons/user/stop-container” (viz. kapitola 5.1.3). Nakonec už je jen zkontrolováno, zda návratová hodnota v proměnné „status” a pokud je je proměnná „siteState” nastavena na string „start”.

4.6.11 Styly dvou sloupových „selector” komponentů

Celé styly tohoto komponentu se třídou „split-container” a dvou kontejnerech o úroveň níže se třídami „control-container” a „linux-container”. „split-container” zařizuje, aby stránka měla dva sloupce. Tyto dva sloupce, avšak stránka nemůže mít v případě, že ji uživatel navštěvuje z mobilního telefonu. Tento kontejner proto svůj obsah v případě menšího zařízení rovná pod sebe a v případě většího vedle sebe. První ze dvou částí obsahu tohoto kontejneru je kontejner „control-container”, který opět pro lepší zobrazení na mobilním pro malá zařízení nastavuje svou šířku na „100 %” a při širším zařízení na „75 %”. Dále už jen pro styl stránky kontejner nastaví šířku svých okrajů a obarví je na zeleno. Druhá část je kontejner „linux-container”, který nastavenou statickou výšku na „70vh” následuje komplikovaná část týkající se okrajů kontejnerů, které je potřeba pro zobrazení na menších zařízeních a na větších upravovat. Okraje jsou pro menší zařízení zobrazovány pouze na levé, pravé a horní straně, pro větší zařízení následovně nahoře, dole a na levé straně. Tento kontejner nakonec už jen rovná svůj obsah do svého středu, neboť má ve svém obsahu jen jeden element. Je důležité se zmínit, že kontejner „control-container” obsahuje list na podobném principu jako jedno sloupcové komponenty s rozdílem, že list používá kontejnery se třídou „selection-container-addon” (tento kontejner nastavuje pouze estetiku v kontejneru, a to sice barvu textu na zelenou, velikost spodního okraje a jeho barvu).



Obr. 6: Vzhled komponentu se stylem dvou sloupcový „selector” (levá část světlý režim a pravá tmavý režim)

4.6.12 Skript dvou sloupcových „selector” komponentů

Stejně jako u jedno sloupcových komponentů (viz. kapitola 4.6.8) stojí tyto komponenty na metodě „handleKeyDown”, která je nastavena jako takzvaný „EventListener”, který je volán vždy když uživatel stiskne některý „input”. Opět stejně jako u jedno sloupcových komponentů (viz. kapitola 4.6.8) jsou hlavní ovládací klávesy „ArrowUp” a „ArrowDown” navíc jsou tu však přidány „ArrowLeft” a „ArrowRight”. Pro navigování polí dvou sloupců jsou používány dvě proměnné „horIndex” (určuje horizontální index v komponentu) a „currentIndex” (určující aktuální index). Hlavní nápad je, že pokud uživatel mačká „ArrowDown” (stejný akorát otočený princip funguje pro „ArrowUp”) jen projde nejprve první sloupec odshora dolu a následně druhý sloupec (dolu z posledního pole znamená na první pole). V takovém pořadí mají pole přidělený index. Klávesy „ArrowLeft” a „ArrowRight” už jen mění hodnotu „horIndex” čímž přesouvají uživateli mezi sloupci. Pro to aby tyto čtyři klávesy správně fungovaly je potřeba ještě kódu, který kontroluje a pamatuje si indexy, aby uživatelská zkušenost byla co nejlepší.

Další klávesy ve kontrolovány ve výše zmíněné metodě jsou „Enter”, které pro každou stránku dělají jinou akci na specifikované hodnotě proměnné „currentIndex”. Jediná kombinace kláves, která je navíc přidána je „ctrl” a „c”, která pouze uživatele přesune na stránku, ze které se dostal na tuto stránku.

4.6.13 Komponent *LessonPageSelector*

Tento komponent slouží pro vyplňování zadaných lekcí, a tedy samotnému přístupu ke kontejnerům pro výuku. První částí je inicializace různých proměnných potřebných pro správnou funkčnost komponenty. Jako všechny stránky pro používání kontejnerů obsahuje tato stránka proměnnou „siteState”, proměnnou „container” obsahující objekt získaný z databáze (viz. kapitola 5.3.7) definující kontejner používaný na této stránce. Další proměnné

jsou „currentIndex”, „horIndex”, „lastControllIndex” a podobné které jsou používány v navigaci a zvýrazňování polí (kontejnerů „selection-container-addon”).

Následuje metoda „onMounted” importovaná z „vue” frameworku, která nastavuje metodu „handleKeyDown”. Předtím však volá metodu „setVariables” tato metoda nastaví proměnné „lesson” podle parametru v cestě na databázový objekt (viz. kapitola 5.3.6) potřebný pro fungování stránky, „extra” podle databázového objektu „user” (viz. kapitola 5.3.3) a objekt „container” podle „extra_id” z proměnné „extra” a „lesson_id” nastavené lekce v proměnné „lesson” z databáze (viz. kapitola 5.3.6). Tato metoda už jen pokud proměnná „container” není prázdná nastaví proměnnou „url” na url potřebné k nastavení objektu „iframe” ve kterém se zobrazuje rozhraní příkazové řádky pro používání docker kontejneru. Dále už jen pokud je proměnná „container” v parametru „running” nastavena na „true” nastaví proměnnou „siteState” na string „container_running” a pokud ne nastaví ji na „container_exists”. Nakonec zavolá metodu „formatText”, která formátuje text zadání lekce získaný z databáze, tak aby byl správně odřádkovaný a čitelný.

Další úvodní částí skriptu je takzvaný „subscribe”, který je funkcí služby „supabase” (viz. kapitola 5.3). Tato funkce volá metodu „handleRunningChange” v případě změny tabulky „container” v databázi a ta pouze změní proměnnou „siteState” na string „container_exists”, aby uživatel neviděl chybu, že kontejner není dostupný.

Následuje metoda „handleContainer”, ve které je upravena pouze to co se stane, když zmáčkne uživatel klávesu „enter”. První z případů je pokud je hodnota proměnné „currentIndex” 4 v takovém případě je pouze volána metoda „push” objektu „router”, která přesune uživatele na předchozí stránku. Další případ je pokud je hodnota proměnné „currentIndex” 5 v tomto případě je volána metoda „stopContainer”.

Tato metoda nejprve zkontroluje, zda kontejner běží (proměnná „siteState” je nastaven na string „container_running”). Pokud tato kontrola projde nastaví proměnnou „siteState” na string „container_stopping”, aby uživatel měl zpětnou vazbu, že se kontejner zastavuje. Dále je zavolán endpoint api „/lessons/user/stop-container” (viz. kapitola 5.1.3), který zastavuje kontejnery. Pokud je odpověď api v bodě „status” nastavena na „true” nastaví program proměnnou „siteState” na „container_exists” v opačném případě na „container_running” a nastaví proměnnou „error_msg” na hodnotu proměnné „msg” odpovědi api, skrze kterou uživatel dostane odpověď na to proč nebyl kontejner zastaven.

V dalším případě zmáčknutí klávesy „enter” v metodě „handleKeyDown” je když je proměnná „currentIndex” nastavena na 6. V tomto případě je volána metoda „resetContainer”. Tato metoda je podobná metodě „stopContainer” (viz. výše) s jediným rozdílem, že nejprve nekontroluje, zda je hodnota proměnné „siteState” „container_exists” a nakonec podle odpovědi api endpointu „/lessons/user/reset-container/” (viz. kapitola 5.1.3) pokud je „true” v bodě „status” nastaví proměnnou „siteState” na hodnotu „container_exists” a v opačném případě provede pouze nastavení chybové hlášky do proměnné „error_msg” (viz. metoda „stopContainer”).

Posledním případem v metodě „handleKeyDown” je pokud je klávesa „enter” zmáčknuta v kombinaci s proměnnou „currentIndex” nastavenou na 7. V tomto případě je nejprve zkontrolováno, zda je proměnná „siteState” string „container_exists” nebo „start” a pokud projde je volána metoda „useContainer”. Tahle metoda nejprve z databáze získá objekt „container” podle „id” aktuálního uživatele a podle „id” lekce, kterou uživatel na této stránce provádí. Podle toho, jak dlouhý je seznam získaný z databáze jsou volány metody. V případě, že je 1 je volána metoda „startContainer” a pokud 0 je volána metoda „createContainer”.

První z metod „createContainer” nejprve zavolá endpoint api „/lesson/user/create-container” (viz. kapitola 5.1.3), který je využíván pro vytváření výukových kontejnerů. Na základě hodnoty odpovědi api v bodě „status” je pokud je nastavena na „true” v tomto případě je proměnná „siteState” nastavena na „container_exists” a v opačném případě je opět jako v metodě „stopContainer” nastavena proměnná „error_msg”. Před tím, než metoda však nastaví proměnnou „siteState” ještě získá z databáze z tabulky „container” řádek používaného kontejneru a nastaví ho do proměnné „container” pro další využití v aplikaci.

Druhá metoda „startContainer” stejně jako metoda „createContainer” zavolá nejprve api endpoint „/lesson/user/start-container” (viz. kapitola 5.1.3) používaný pro spouštění kontejnerů. Pokud je odpověď v bodě „status” nastavena na „true” metoda nejprve nastaví proměnnou „siteState” na string „container_loading” (uživatel místo docker kontejneru nejprve vidí text „loading”). Z důvodu, že se musí docker kontejner nejprve spustit, je spuštěn 9sekundový „timeout” (Dascalescu & blackgreen, 2009) a poté co skončí je nastavena proměnná „siteState” na string „container_running” (uživateli je zobrazen docker kontejner). Nakonec už jen pokud není hodnota odpovědi v bodě „status” „true” je jako u předchozích metod nastavena proměnná „error_msg”.

4.6.14 Komponent *LessonSolutionsSelector*

Tento „selector” je používán pro výběr lekcí pro kontrolu. Jako u všech komponentů druhů „selector” jsou v první části skriptu inicializované různé proměnné. Jsou zde inicializované veškeré proměnné, které jsou potřebné pro funkčnost ovládání stránky, ale navíc jsou zde inicializované další proměnné. První z nich je „lesson” obsahující objekt z databáze lekce (viz. kapitola 5.3.6) u které jsou na této stránce kontrolovány kontejnery. Druhá je „containers”, která je list a obsahuje veškeré objekty kontejnerů z databáze (viz. kapitola 5.3.7), které se týkají lekce v proměnné „lesson”. Poslední úprava je pouze jen modifikovaná konstanta „listLength”, která je změněna na proměnnou.

Následuje metoda „onMounted”, která má stejný obsah jako v komponentu „LessonPageSelector” (viz. kapitola 4.6.13) a stejně jako v tomto komponentu volá metodu „setVariables” a ta v tomto komponentu nejprve získá podle „id” obsáhnutého v cestě k stránce objekt lekce z databáze (viz. kapitola 5.3.6) a nastaví ho do zmíněné proměnné „lesson”. Pokud tento objekt není prázdný jsou dále do již zmíněné proměnné „containers” nastaveny objekty kontejneru z databáze podle „id” již získané lekce (jsou získány veškeré kontejnery s řešeními lekce). Podle délky listu je dále v proměnné „containers” je dále nastavena proměnná „listLength”. Nakonec už je jen pro každý kontejner získán email jeho

tvůrce (toho kdo kontejner vyplnil) a tento email je přidán k příslušnému objektu kontejneru (aby byl schopen kontrolující identifikovat uživatele, který vyplnil lekci).

Ve skriptu následuje metoda „handleKeyDown”, která v tomto skriptu obsahuje pouze dva případy pro zmáčknutí klávesy „enter” jsou, pokud je proměnná „currentIndex” nastavena na 3 a více nebo 2. V případě, že je proměnná nastavena na 2 je uživatel vrácen na předchozí stránku. V druhém případě je volána metoda „selectContainer”. Tato metoda nejprve spočítá podle proměnné „currentIndex”, který kontejner si uživatel vybral na zkontrolování (uživatel si na hodnotách proměnné „currentIndex” větších než 3 může vybrat, který kontejner chce zkontrolovat). Dále zkontroluje, zda není uživatel na webové stránce z mobilního zařízení (GeeksforGeeks, 2023) a pokud ne přesměruje ho na stránku, kde může kontejner zkontrolovat (viz. kapitola 4.6.10).

4.6.15 „navbar” komponenty

Složka „navbar” obsahuje dva soubory „NavBarBottom” a „NavBarTop”, které obsahují elementy, které jsou obsaženy na každé stránce, neboť jsou importovány na stránce „App” (viz. kapitola 4.5). Jejich úkolem je umožnit uživateli jednodušší navigaci po aplikaci.

První z těchto souborů tedy soubor „NavBarTop” je vždy na horní straně stránky a zabírá celou její šířku. Tohoto je docíleno pomocí obsahu „template” elementu souboru. První je kontejner s třídou „navbar-top”, který docílí toho že šířka elementu přes celou stránku a že je element vždy umístěn na horní straně displeje. Další věci, které tato třída nastavuje je font na takzvaný mono font pozadí na černé a posouvá element do popředí. Dále obsahuje kontejner další kontejner se třídou „navbar-top-item-container” a „hr” element se třídou „navbar-top-splitter”. „hr” element pouze graficky odděluje navigační řádku od zbytku webové stránky. Zmíněný kontejner se třídou „navbar-top-item-container” dále jen zařizuje, aby její obsah byl srovnán v jedné řádce. Dále už jen zmíněný kontejner obsahuje další kontejner se třídou „navbar-top-item” a jednu ze tříd z frameworku „tailwind”, která nastavuje text na některou barvu. Všechny tyto kontejnery jsou dále skryty nebo zobrazeny na základě toho na, které stránce se uživatel nachází a obsahují text napovídající uživateli, jak webovou aplikaci navigovat.

Následuje „script” element tohoto souboru, který je převážně zaměřen na získání parametrů (jsou uloženy v konstantách) podle, kterých jsou následovně zobrazovány, nebo skryty různé kontejnery se třídou „navbar-top-item”. Parametry jdou rozdělit na dvě skupiny, a to sice ty, které jsou definované pomocí „hooku”, který je naimportován výše v souboru. První je konstanta „currentRoute”, kterou „hook” nastavuje na hodnotu cesty na, které se uživatel aktuálně nachází. Cesta je získána z parametru „path” výše načteného objektu uloženého v konstantě „route” pomocí metody „useRoute” importované z knihovny „vue-router”. Druhá konstanta je „isWithoutCmd” je definována pomocí „hooku”, který pomocí pěti „regexů” zkontroluje, zda pasují na cestu uloženou v konstantě „currentRoute” (aarcoraci, 2018), nebo zda není rovna „/learning/create-lesson” a pokud ano nastaví ji na „true” a naopak (viz. Obr. 4). Poslední částí „skriptu” jsou metody „setUser” a „getPremiumStatus”. Prvně je pomocí importovaného „hooku” „onMounted” (tento „hook” provede obsažený kód vždy při

zobrazení elementu) zavolá metodu „setUser” ta pouze volá metodu importovanou ze souboru „getFunctions” (viz. kapitola 4.3) a pokud tato metoda vrátí model uživatele zavolá druhou metodu („getPremiumStatus”) a nastaví konstantu „localUser” na „true”. Následuje už jen zavolaná metoda, která získá „extra” informace o uživateli a nastaví proměnnou „premiumUser” na hodnotu sloupce „premium” (viz. kapitola 5.3.3).

Druhý soubor „NavBarBottom” tento soubor je velice podobný předchozímu souboru. Jedním z rozdílů je obsah elementu „template”, který obsahuje první kontejner s třídou „navbar-bottom”, která nastavuje jeho pozici na konec stránky. Tento kontejner má následovně otočené pořadí „hr” elementu a kontejneru, který u předchozího souboru měl třídu „navbar-top-item-container” a u aktuálního souboru „navbar-bottom-item-container”. U tohoto souboru následovně srovná jeho obsah do jedné řádky. Tento obsah je následovně uzavřen v kontejnerech se třídami „navbar-bottom-item”, které následovně nastaví mezery mezi kontejnery, tak aby byly rovnoměrně seřazeny, aby měli správný text a aby jejich obsah byl v jejich prostředku. Kontejnery následovně obsahují jednu ze tříd „instagram-gradient-text”, „github-gradient-text”, „license-gradient-text” a „premium-gradient-text” nastavující obsáhnutí text na gradient různých barev.

Element „script” je v souboru úplně stejný s rozdílem, že v tomto již není potřeba „hooku” nastavující konstanty „isWithoutCmd” a „currentRoute” (aarcoraci, 2018). Jako v předchozím souboru jsou nastavené proměnné, aby bylo možné zobrazit kontejnery se třídami „navbar-bottom-item”.

5 BACKEND

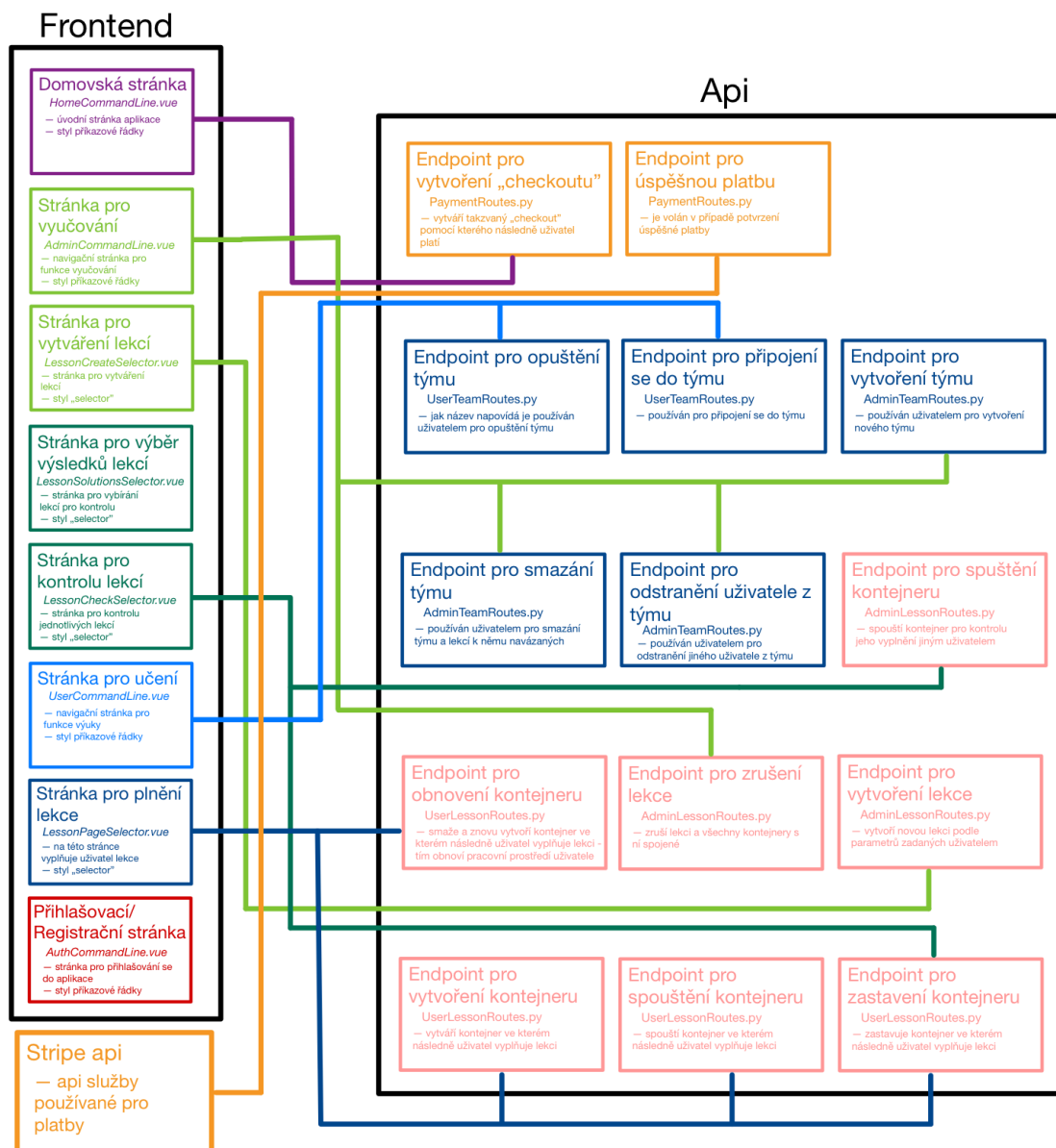
Celý takzvaný backend této webové aplikace lze rozdělit na dvě části. Api napsané v pythonu za pomoci frameworku „flask” a službu „supabase”. Api je zde pro složitější upravování databáze tak aby nebyl frontend příliš komplikovaný, ale hlavně pro přístup k docker kontejnerům, které je potřeba pro fungování hlavní funkce aplikace. Dalším důležitým faktorem pro existenci api je přístup k procesům, protože api musí spouštět skripty, které kontrolují zátěž a další informace ohledně kontejneru. Služba „supabase” dále zařizuje databázi ve které jsou uloženy veškeré důležité údaje zařizuje dále své vlastní api skrze které je následně služba využívána ve zmíněném api a i ve frontend části aplikace.

5.1 Flask api

Tato část aplikace byla vytvořena převážně pro manipulaci s takzvanými docker kontejnery, s kterými není možné manipulovat z frontendu ani ze služby „supabase”. Zároveň však je využívána pro generování kódu pro platební bránu a různé komplexnější operace, které by nebylo vhodné pro provádění na frontend části aplikace i přesto, že by to služba „supabase” skrze své api umožňovala. Veškeré api endpointy jsou rozděleny do čtyř souborů a tyto soubory rozděleny do dvou složek a to sice „lessons” (cesty zabývající se operacemi s lekcemi (viz. kapitola 5.1.3)) a „teams” (cesty zabývající se operacemi s týmy (viz. kapitola 5.1.4)). Jediným souborem s cestami, který není uložen ve své specifické složce je

„PaymentRoutes” (viz. kapitola 5.1.5), protože nepatří ani do jedné kategorie. Poslední částí aplikace je složka „Services” (viz. kapitola 5.1.2), která obsahuje nastavení některých služeb používané dále v aplikaci.

5.1.1 Spojení s frontendem



Obr. 7: Struktura propojení api s frontendem

Graf uvedený výše (obr. 7) popisuje ze kterých stránek/komponent (viz. kapitola 4) jsou volány jednotlivé endpointy. První ze stránek, tedy ta domovská (viz. kapitola 4.6.3) volá pouze jeden endpoint (viz. kapitola 5.1.5) a to sice ten pro vytváření takzvaného „checkoutu” skrze, který následovně uživatel platí. Druhá ze stránek, která volá endpointy je stránka pro vyučování (viz. kapitola 4.6.6). Tato stránka volá endpointy pro vytvoření týmu (viz. kapitola 5.1.4), když uživatel chce vytvořit tým, pro smazání týmu (viz. kapitola 5.1.4), když uživatel chce smazat tým, pro odstranění uživatele z týmu (viz. kapitola 5.1.4), když chce vlastník

týmu vyhodit jiného uživatele z týmu a pro zrušení lekce (viz. kapitola 5.1.3), když uživatel ruší vytvořenou lekci. Další stránka volající endpoint je stránka pro vytváření lekcí (viz. kapitola 4.6.9), která volá endpoint s podobným názvem a se stejným účelem, a to sice endpoint pro vytvoření lekce (viz. kapitola 5.1.3). Stránka pro kontrolu lekcí následně volá dva endpointy a to sice ten pro spuštění kontrolovaného kontejneru (viz. kapitola 5.1.3), pokud chce uživatel kontrolovat vyplněný kontejner uživatelem, který vyplňoval jeho lekci, a ten pro zastavení kontejneru (viz. kapitola 5.1.3), pokud chce přestat kontrolovat tento kontejner. Předposlední ze stránek, které volají endpointy, je stránka pro učení (viz. kapitola 4.6.5) volající endpoint pro připojení se do týmu (viz. kapitola 5.1.4), pokud se uživatel chce připojit do některého týmu a endpoint pro opuštění týmu (viz. kapitola 5.1.4), pokud chce uživatel naopak tým opustit. Posledním elementem, který volá endpoint tohoto backendu je služba „stripe” využívána pro platby a volá endpoint pro úspěšnou platbu (viz. kapitola 5.1.5) v případě, že úspěšně proběhla některá platba.

5.1.2 Services

Celá tato část aplikace se nachází ve složce „services” a skládá se ze tří souborů. První z nich je „docker_service”, který je velice jednoduchý a stará se pouze o inicializaci exportované proměnné „docker” přes kterou následně kód používá software docker pro používání kontejnerů. Druhým souborem je „stripe_service” inicializující proměnnou „stripe” za pomoci api klíče a následnému exportování této proměnné, tak aby mohla být využita v souboru „PaymentRoutes” pro komunikaci s platební bránou. Posledním souborem je soubor „supabase_service”, který je z těchto tří souborů zdaleka nejkomplicovanější a je popsán níže.

Nejprve se v tomto souboru inicializuje klient služby „supabase” do proměnné nesoucí stejný název (viz. kapitola 5.3) a je následovně používána v několika metodách níže v kódu. Tyto metody jsou podobné metodám definovaným v kapitole (viz. kapitola 4.3) ve frontendu. Metody jsou stejně jako ve zmíněné kapitole používány pro získávání dat z databáze a jsou zde definovány pro zjednodušení kódu v ostatních částech api. Všechny tyto metody jsou napsány v podobném formátu a to tak, že začínají nastavením proměnných, podle kterých metody získávají data z databázi. Proměnné jsou nastaveny za pomoci takzvaných „kwargs” s kterými metoda může ale nemusí být volána. Následně jsou nastaveny dvě proměnné „query_field” obsahující sloupec podle kterého je v databázi vyhledáváno a „query_value” hodnota která bude srovnávána se sloupcem. Proměnné jsou nastaveny podle toho, jaké hodnoty byly nalezeny v „kwargs”. Nakonec už jen získá metoda z databáze za pomoci vytvořeného klienta v proměnné „supabase” data podle dvou zmíněných proměnných a vrátí je.

5.1.3 Endpointy pro lekce

Tyto endpointy jsou rozděleny do dvou souborů a to sice „AdminLessonRoutes”, kde cesty slouží pro vytváření a mazání lekcí a kontrolování kontejnerů a „UserLessonRoutes”, které slouží pro akce spojené s používáním, zastavováním a obnovováním kontejnerů. Oba tyto soubory jsou obsáhnuty ve složce „lessons”.

První ze dvou z výše zmíněných souborů tedy soubor „AdminLessonRoutes” obsahuje celkem 3 endpointy. Těmito endpointy jsou „/lesson/admin/create”, která se stará o vytváření nových lekcí, „/lesson/admin/cancel” zabývající se rušením lekcí a „/lesson/admin/start-container”, která stará o spouštění docker kontejneru pro kontrolu zadavatelem lekce.

První z endpointů tedy „/lesson/admin/create” volá metodu „create_lesson”. Metoda nejprve podle parametru získaného z json objektu, s kterým je endpoint volán získá řádku z tabulky „limitations” (viz. kapitola 5.3.4) a „user” (viz. kapitola 5.3.3) (parametr je id uživatele, který použil funkci, která zavolala tento endpoint, ve frontendu). Pokud tyto dvě řádky existují metoda pokračuje, ale v opačném případě api pošle odpověď, že se proces nepovedl a důvod. Tímto způsobem jsou následovně provedeny veškeré další podmínky. V případě tedy že zmíněná podmínka projde je z databáze získána řádka z tabulky „lesson” (viz. kapitola 5.3.6), která má jméno stejné jako lekce, kterou se uživatel snaží vytvořit (jméno vytvářené lekce je uloženo v jsonu se kterým je endpoint zavolán). Pokud žádná takováto řádka (uživatel nemůže vytvořit stejnojmennou funkci) neexistuje pokračuje metoda dalšími dvěma podmínkami. První z nich kontroluje, zda uživatelem zadaný čas (opět obsáhnutý v jsonu tentokrát pod názvem „date” na indexu 1) v kterém by měla lekce skončit není dříve než dnešek a druhá podmínka kontroluje, zda řádka z tabulky „limitations” (viz. kapitola 5.3.4) získaná na začátku metody ve sloupci „lesson_limit” je větší než hodnota ve sloupci „lessons” (viz. kapitola 5.3.6) (tímto je kontrolováno zda uživatel nedosáhl limitu vytvořených lekcí). Jestliže všechny podmínky prošli je vytvořen pythonový objekt takzvaný „dictionary” a obsahuje celou definici řádky, která bude vložena do tabulky „lesson” (viz. kapitola 5.3.6), tak aby byla vytvořena nová lekce. Tento objekt je vytvořen pomocí informací definovaných v jsonu připojenému při volání tohoto endpointu. Do tabulky „lesson” (viz. kapitola 5.3.6) je následně tento objekt vložen a tím je vytvořena nová lekce. Poslední akcí, kterou metoda provede už je jen upravení řádky v tabulce „limitations” příslušící aktuálnímu uživateli ve sloupci „lessons” na hodnotu o jedna větší, tak aby zde byl aktuální počet lekcí vytvořených uživatelem.

Druhý endpoint „/lesson/admin/cancel” volá metodu „cancel_lesson”, která nejprve za pomoci hodnot jsonu zasláného při volání api najde řádku v tabulce „user” (viz. kapitola 5.3.3) příslušící uživateli používajícímu aktuálně aplikaci, řádku v tabulce „limitations” (viz. kapitola 5.3.4), která opět přísluší tomuto uživateli, a nakonec z tabulky „lesson” získá lekci se jménem zadaném do jsonu v bodě „lesson_name”. Následně se metoda podívá, zda existují všechny tyto tři řádky. Stejně jako u předešlého endpointu jsou veškeré opačné případy podmínek ošetřeny tak, že je odeslána odpověď api s chybou. Pokud tedy všechny podmínky projdou metoda dále pokračuje tím, že vymaže řádku z tabulky „lesson”, která má ve sloupci „name” stejnou hodnotu jako json v bodě „lesson_name” a ve sloupci „creator_id” stejnou hodnotu jako již nalezený řádek z tabulky „user” (viz. kapitola 5.3.3) ve sloupci „id”, čímž je smazána lekce z databáze a tedy zrušena. Je však možné, že zůstali vytvořeny ještě některé kontejnery pro tuto lekci, a tak metoda smaže v další části kódu veškeré docker kontejnery jejichž začátek jména odpovídá jsonu v bodě „lesson_name” a tímto způsobem jsou smazány veškeré docker kontejnery, které by mohly po lekci zůstat (viz. Obr. 8). Nakonec je už jen

upravená tabulka „limitations” (viz. kapitola 5.3.4), která přísluší uživateli, který zavolal tuto funkci skrze frontend, tak že je hodnota v jejím sloupci „lessons” snížena o jedna (počet jeho lekcí je snížen).

```
containers = docker.containers.list(all=True)

for container in containers:
    if container.name.startswith(json["lesson_name"] + "-"):
        if container.status == "running":
            container.stop()
            container.remove()
```

Obr. 8: Způsob mazání kontejnerů

Poslední endpoint, který tento soubor obsahuje, je endpoint „/lesson/admin/start-container” a volá metodu „start_container”. Metoda nejprve kromě načtení jsonu získá čtyři řádky z databáze a to sice z tabulky „user” (viz. kapitola 5.3.3) podle id uživatele, který tuto funkci skrze frontend zavolal, z tabulky „limitations” (viz. kapitola 5.3.4) opět podle id uživatele, který tuto funkci skrze frontend zavolal, z tabulky „container” (viz. kapitola 5.3.7) podle „id”, které musí být rovno hodnotě jsonu v bodě „container_id” a nakonec z tabulky „container_script” (viz. kapitola 5.3.8) podle jejího sloupce „container_id”, který musí být roven jsonu v bodě „container_id”. Jako v předešlých metodách i pro tuto metodu platí, že veškeré podmínky jsou ošetřeny a v případě jejich selhání je zaslána odpověď s chybou. První dvě podmínky jen kontrolují, zda byly úspěšně nalezeny všechny čtyři výše zmíněné řádky. Následně je zkontrolováno, zda řádek z tabulky „limitations” (viz. kapitola 5.3.4) ve sloupcích „running_container_id” a „checking_container_id” neobsahuje jinou hodnotu než nula, protože to by znamenalo, že uživatel má již spuštěný jiný kontejner. V případě že předešlé podmínky projdou získá metoda objekt docker kontejneru za pomoci docker „cli” (bylo inicializováno v souboru „docker_service” (viz. kapitola 5.1.2)) a pokud tento objekt existuje pokračuje metoda dále. Následující kód připravuje spuštění skriptu pro kontrolu docker kontejneru (viz. kapitola 5.2.1) a to znamená, že nejprve nastaví pole „args” na přídatné argumenty pro skript. Toto pole obsahuje nejprve hodnotu z jsonu v bodě „cpu_load” a tento json je získán z řádky z tabulky „container_script” (viz. kapitola 5.3.8) ze sloupce „settings” (řádka byla získána na začátku metody), druhá hodnota v poli je ze stejného řádku a sloupce s rozdílem toho že z jsonu v bodě „network_load”, poslední hodnotou je ze řádku získaného na začátku metody z tabulky „container” (viz. kapitola 5.3.7) ze sloupce „name”. Následně je připraveno, jak má být příkaz spuštěn do pole „command”, první hodnota pole je string „python3” a druhá je string „container_script.py” a poslední hodnota je pole „args”. Poslední přípravná proměnná pro spuštění skriptu je „target_cwd” obsahující cestu k skriptu pro kontrolu kontejneru (viz. kapitola 5.2.1). Se všemi těmito proměnnými se následně pokusí metoda spustit skript jako proces a uložit si jeho objekt do proměnné „process” a pokud se to metodě podaří pokračuje dalšími akcemi. Další akce jsou spuštění výše získaného docker kontejneru, upravení sloupců tabulky „container_script” (viz. kapitola 5.3.8) „pid” na hodnotu „pid” (process id) výše spuštěného procesu a „running” na boolean „true”, upravení sloupce tabulky „container” (viz. kapitola 5.3.7) ve sloupci

„running” na boolean „true” a nakonec upravení tabulky „limitations” (viz. kapitola 5.3.4) ve sloupci „checking_container_id” na hodnotu řádky z tabulky „container” získanou na začátku metody ve sloupci „id”, tak aby bylo jasné, že má uživatel spuštěný kontejner.

Druhý ze zmíněných souborů tedy soubor „UserLessonRoutes” obsahuje celkem 4 endpointy, kterými jsou „/lesson/user/create-container” používaná pro vytváření docker kontejneru, který následně používají uživatelé, „/lesson/user/start-container”, která slouží pro spuštění zmíněného docker kontejneru a kontrolujícího skriptu (viz. kapitola 5.2.1), „/lesson/user/stop-container” sloužící pro zastavování zmíněných docker kontejnerů a jejich skriptů (viz. kapitola 5.2.1) a poslední endpoint „/lesson/user/reset-container” používán pro obnovení kontejneru do původního stavu.

První z endpointů je „/lesson/user/create-container” volající metodu „create_container”, která nejprve získá json s kterým je endpoint volán. Podle tohoto jsonu v bodě „user_id” nejprve získá řádku z tabulky „user” (viz. kapitola 5.3.3) podle sloupce „user_id”. Následně získá podle jsonu v bodě „lesson_id” řádku z tabulky „lesson” (viz. kapitola 5.3.5) podle sloupce „id” a následně za pomoci navíce hodnoty získaného řádku z tabulky „user” ve sloupci „id” získá řádek z tabulky „container”. Následně pokud program našel první dvě řádky, ale ne poslední postupuje následně. Je důležité zmínit, že pokud neprojde některá z podmínek je api zaslána odpověď s chybou. Pokud tedy podmínky projdou je volána metoda „generateContainerInformation”, která vygeneruje pro kontejner unikátní jméno, heslo (gerrit, 2010), přihlašovací údaje uživatele a unikátní port pro docker kontejner na kterém nebude v konfliktu s žádným jiným kontejnerem. Tyto data metoda následně vrátí pro další využití v původní metodě. Původní metoda nejprve zkontroluje, zda metoda „generateContainerInformation” a pokud ano zavolá s nimi v parametru metodu „createContainer”. Tato metoda nejprve zformátuje data z řádku „lesson” a data, které vytvořila metoda „generateContainerInformation”, do pythonového objektu „dictionary”, který je možné předat docker kontejneru. Následně už jen metoda z těmito zformátovanými daty vytvoří kontejner používající image „garo/shellinabox:latest” (garo, 2019) a vrátí objekt pythonový objekt tohoto kontejneru. Původní metoda už pouze zkontroluje, zda zmíněný vrácený objekt kontejneru existuje a pokud ano vloží data, do kterých byly přidány ještě dva body a to sice „lesson_id” obsahující sloupec hodnotu jsonu pod jménem „lesson_id” a „user_id” obsahující sloupec „id” z řádky tabulky „user” nalezené na začátku metody, do tabulky „container” a skončí.

Další endpoint je „/lesson/user/start-container”, který volá metodu „start_container”, ta opět nejprve získá json s kterým je volána a následně získá 4 řádky z různých tabulek. Tyto zmíněné řádky jsou z tabulek „user” (viz. kapitola 5.3.3), která je získána podle sloupce „user_id” a musí být rovna jsonu v bodě „user_id”, „limitations” (viz. kapitola 5.3.4), která musí být ve sloupci „extra_id” stejná jako řádek získaný z tabulky „user” (viz. kapitola 5.3.3) ve sloupci „id”, „lesson” (viz. kapitola 5.3.6) se stejnou hodnotou ve sloupci „id” jako json pod jménem „lesson_id” a nakonec „container” (viz. kapitola 5.3.7) získaný podle sloupce „id”, který musí být roven jsonu v bodě „container_id”. Metoda jako u všech ostatních nejprve zkontroluje, zda všechny tyto řádky a v případě, že některá z nich neexistuje vrátí

endpoint chybu (takto fungují všechny ostatní podmínky) a navíc provede kontrolu, zda řádek z tabulky „limitations” ve sloupcích „running_container_id” a „checking_container_id” neobsahuje jinou hodnotu než nula, protože to by znamenalo, že má uživatel spuštěný ještě jiný docker kontejner (uživatel může mít spuštěný pouze jeden). Následně metoda získá z tabulky „container_script” (viz. kapitola 5.3.8) podle sloupce „container_id” rovnému hodnotě jsonu v bodě „container_id”. Poté přichází další podmínka, která rozhoduje, jak bude metoda pokračovat, a to sice zda hledaná řádka z tabulky „container_script” (viz. kapitola 5.3.8) existuje. Oba postupy jsou stejné s jedním menším rozdílem, který bude níže zmíněn. Metoda v další části připravuje ke spuštění docker kontejner a skript pro jeho kontrolu (viz. kapitola 5.2.1). To je provedeno nejprve nastavením pole „args”, jehož první dvě hodnoty jsou nastaveny podle sloupce „settings” řádky tabulky „lesson” získané na začátku metody, který obsahuje json a první dvě hodnoty pole jsou tedy tento json v bodech „cpu_load” a „network_load”. Poslední hodnota pole „args” je pak ještě řádek z tabulky „container” získaný na začátku metody ve sloupci „name”. Další pole, které je připraveno je „command”, které se skládá ze tří hodnot, a to sice stringu „python3”, stringu „container_script.py” a pole „args”. Následuje poslední proměnná potřebná pro spuštění skriptu a to sice „target_cwd”, která obsahuje cestu ke skriptu pro kontrolu docker kontejnerů (viz. kapitola 5.2.1). Za pomoci výše zmíněných proměnných se následně metoda pokusí spustit takzvaný „subprocess”, kterým je skript pro kontrolu kontejnerů (viz. kapitola 5.2.1) a uložit si jeho objekt do proměnné „process”. Pokud tento objekt existuje pokračuje metoda jedinou rozdílnou částí (dvě části jsou zmíněny výše u jedné z podmínek). V prvním případě pokud řádka v tabulce „container_script” (viz. kapitola 5.3.8) neexistovala je za pomoci objektu procesu a řádek získaných na začátku metody vytvořena a do tabulky vložena nová řádka a v druhé možnosti jsou pouze její sloupce „pid” a „running” změněny. Následuje spuštění samotného docker kontejneru, který je získán podle sloupce „name” řádky získané z tabulky „container” (viz. kapitola 5.3.7) na začátku metody. Nakonec, než metoda končí následují dvě změny databáze a to sice upravení objektu spuštěného docker kontejneru v databázi ve sloupci „running” na boolean „true” a nakonec upravení řádky tabulky „limitations” uživatele, který vyvolal tuto akci, ve sloupci „running_container_id” na sloupec „id” řádky získané z tabulky „container” na začátku metody.

Předposlední endpoint je „/lesson/user/stop-container” volá metodu „stop_container”, která nejprve jako všechny ostatní metody volané endpointy získá json se kterým je endpoint volán. Za pomoci tohoto endpointu metoda následně získá z databáze tři řádky, a to sice z tabulek „container_script” (viz. kapitola 5.3.8) a to tak, že musí být ve sloupci „container_name” roven jsonu v bodě „name”, „container” (viz. kapitola 5.3.7) tak že sloupec „id” tohoto řádku musí mít stejnou hodnotu jako sloupec „container_id” řádku získaného v předešlém kroku a „limitations” (viz. kapitola 5.3.4) podle sloupce „extra_id”, který musí být roven sloupci „user_id” řádky získané v předešlém kroku. Kromě těchto řádek navíc metoda na začátku ještě podle hodnoty zadané v jsonu pod jménem „name” získá objekt docker kontejneru s touto hodnotou ve jménu. Dále je zkontrolováno, zda všechny tyto hodnoty byly nalezeny (u všech podmínek jsou opět v případě že neprojdou odesílány chyby jako odpověď api). V případě, že všechny hodnoty byly nalezeny zastaví metoda kontejner a zastaví i process (Bakuriu, 2013), jehož „pid” je uloženo v řádce tabulky „container_script” (viz. kapitola

5.3.8) nalezené na začátku metody ve sloupci „pid”. V případě, že vše toto projde jsou provedeny tři úpravy databáze, a to sice řádky tabulky „container” (viz. kapitola 5.3.7), která byla nalezena na začátku metody, ve sloupci „running” na boolean „false”, řádky tabulky „container_script” opět stejné jako té, která byla nalezena na začátku metody, ve sloupci „running” na boolean „false” a nakonec řádky tabulky „limitations”, která byla nalezena na začátku metody, ve sloupcích „running_container_id” na nula a „checking_container_id” na nula.

Poslední z endpointů, které druhý soubor obsahuje, je endpoint „/lesson/user/reset-container” volající metodu „reset_container”. Metoda nejprve kromě získání jsonu s kterým je volána získá tři řádky z databáze a to sice z tabulky „container” (viz. kapitola 5.3.7), kde se hodnota ve sloupci „name” rovná hodnota jsonu pod jménem „name”, z tabulky „lesson” (viz. kapitola 5.3.6), u které se hodnota ve sloupci „id” rovná hodnotě jsonu v bodě „lesson_id” a nakonec z tabulky „limitations” (viz. kapitola 5.3.4), které hodnota ve sloupci „user_id” je rovna hodnotě jsonu pod jménem „user_id”. Následně ještě metoda nalezne objekt docker kontejneru za pomoci docker „cli” (viz. kapitola 5.1.2) se stejným jménem jako je hodnota jsonu v bodě „name”. Následně je zkontrolováno, zda veškeré řádky a objekt existují a pokud ne je api vrácena chybová hláška (takto fungují veškeré podmínky v této funkci). Dále je kontrolováno, zda je hodnota ve sloupci „resets” menší než hodnota ve sloupci „reset_limit” v řádce získané na začátku metody z tabulky „limitations” (viz. kapitola 5.3.4). Následuje poslední podmínka, která se podívá, zda je hodnota řádky získané z tabulky „container” (viz. kapitola 5.3.7) na začátku metody ve sloupci „running” nastavena na boolean „false” a pomocí objektu docker kontejneru je zjištěno zda docker kontejner, který představuje běží. Pokud všechny tyto podmínky jsou dodrženy je docker kontejner představovaný objektem získaným na začátku metody smazán a za pomoci metody „createContainer” dokumentované výše je opět vytvořen. Metoda už jen nakonec upraví řádek v databázi tabulky „limitations” nalezený na začátku metody ve sloupci „resets” na hodnotu o jedna vyšší.

5.1.4 Endpointy pro týmy

Všechny tyto endpointy jsou rozděleny do dvou souborů s různými cíli. První z těchto souborů je „AdminTeamRoutes” používaný pro endpointy zabývající se vytvářením týmů, mazání týmů a vylučování uživatelů z týmů. Druhým souborem je soubor „UserTeamRoutes” jehož endpointy se zabývají přihlašování uživatelů do týmů a odhlašování se uživatelů z týmů. Oba tyto soubory jsou uloženy ve složce „teams”.

První z dvou souborů „AdminTeamRoutes” obsahuje tři endpointy a to sice „/teams/admin/create” pro vytváření týmů, „/teams/admin/delete” pro mazání týmů a docker kontejnerů, které případně vzniklé v lekcích asociovaných s tímto týmem (viz. kapitola 5.3.2) a nakonec „/teams/admin/kick” pro odstraňování uživatelů z týmů a mazání docker kontejnerů, které byly případně vytvořeny v lekcích, které uživatel vyplňoval.

První z endpointů v souboru „AdminTeamRoutes” tedy endpoint „/teams/admin/create”, jak již bylo zmíněno slouží pro vytváření týmů a volá metodu „create_team”. Metoda nejprve za pomoci jsonu s kterým je endpoint volán získá z tabulek „user” (viz. kapitola 5.3.3) řádku

příslušící uživateli, který tuto funkci z frontendu aktivoval, „limitations” (viz. kapitola 5.3.4) řádku příslušící stejnému uživateli jako předchozí řádka a „team” (viz. kapitola 5.3.5), kde sloupec „name” je roven jsonu v bodě „name”. Nejprve metoda zkontroluje, zda existují dva první řádky a pokud ano podívá se zda řádek z tabulky „team” (viz. kapitola 5.3.5) a jestliže existuje metoda nepokračuje, protože nemůžou existovat dva týmy se stejným jménem. Pokud některé z těchto podmínek, a i z nadcházejících neprojde odešle api zpět odpověď z příslušnou chybou. Dále nastává ještě poslední kontrola, a to zda má získaná řádka z tabulky „limitations” (viz. kapitola 5.3.4) za začátku metody větší hodnotu ve sloupci „team_limit” než ve sloupci „teams”, protože uživatel má limit na počet vytvořených lekcí nastavený ve sloupci „team_limit”. V případě, že tato podmínka vyjde vloží metoda do tabulky „team” (viz. kapitola 5.3.5) v databázi novou řádku, kde ve sloupci „name” je hodnota jsonu v bodě „name” a ve sloupci „creator_id” hodnota sloupce „id” z řádku získaného na začátku metody z tabulky „user”. Tento řádek si následně ještě metoda uloží do proměnné „team” pro další využití. Následně ještě metoda přidá na konec pole ve sloupci „teams” v řádce tabulky „user” (viz. kapitola 5.3.3) získané na začátku metody hodnotu sloupce „id” nově vytvořeného v předešlém kroku a uloženém v proměnné „team”. Nakonec metoda už jen upraví řádek v tabulce „limitations” získaný na začátku metody z databáze, tak že ve sloupci „teams” zvětší hodnotu o jedna, čímž je uložen počet týmu uživatele.

Další z endpointů tedy druhý v souboru „AdminTeamRoutes” je „/teams/admin/delete”, který je používán pro mazání týmů a volá metodu „delete_team”. Kromě získání jsonu s kterým je tým volán metoda nejprve získá dva řádky nejprve jeden z tabulky „user” (viz. kapitola 5.3.3), který přísluší uživateli, který funkci spustil, a druhý z tabulky „team” (viz. kapitola 5.3.5) jehož sloupec „name” má stejnou hodnotu jako json v bodě „team_name”. Následně zkontroluje metoda, zda tyto dva řádky existují a pokud ne pošle odpověď api z chybou (takto to funguje pro všechny podmínky). Následně z tabulky „lesson” (viz. kapitola 5.3.6) získá metoda všechny řádky, které mají ve sloupci „team_id” stejnou hodnotu jako již nalezený řádek z tabulky „team” (viz. kapitola 5.3.5) ve sloupci „id”. Pro každou nalezenou řádku následně metoda opakuje proces s tím, že smaže veškeré docker kontejnery se jménem, které začíná stringem obsaženém ve sloupci „name” této řádky (viz. Obr. 8). Nakonec následují už jen dvě úpravy databáze. První z nich je úprava řádky tabulky „limitations” (viz. kapitola 5.3.4), která byla nalezena na začátku metody a to sice ve sloupcích „lesson”, kde je odečten počet nalezených lekcí výše v metodě a ve sloupci „teams”, kde je odečteno jedna od původní hodnoty, protože byl smazán jeden tým. Druhá z úprav je smazání řádku z tabulky „team” (viz. kapitola 5.3.5), který byl nalezen na začátku metody, čímž je odstraněn tým.

Poslední z endpointů je „/teams/admin/kick”, která volá metodu „kick_user”. Metoda nejprve, kromě získání jsonu s kterým je endpoint zavolán, získá řádek z tabulky „user” (viz. kapitola 5.3.3), který přísluší uživateli, který tuto funkci ve frontendu zavolal (viz. kapitola 4.6). Pokud tato řádka existuje pokračuje metoda tím (v opačném případě odešle api odpověď s chybovou hláškou a takto je to se všemi podmínkami v této metodě), že získá z tabulky „team” (viz. kapitola 5.3.5) řádku, která má ve sloupci „name” stejnou hodnotu jako json v bodě „team_name” a přísluší uživateli, který tuto funkci ve frontendu spustil (viz. kapitola 4.6). V případě, že řádka existuje, získá metoda z tabulky databáze „user” (viz. kapitola 5.3.3)

řádku, která má ve sloupci „code” stejnou hodnotu jako json pod jménem „user_code”. Pokud tento řádek existuje pokračuje metoda tak, že zkontroluje, zda ve sloupci „id” mají dva řádky z tabulky „user” nalezeny v předešlé části metody stejné hodnoty a pokud ano vrátí api chybovou hlášku, protože uživatel, který tým vlastní nemůže z týmu vyhodit sám sebe. Potom metoda získá z tabulky „lesson” (viz. kapitola 5.3.6) řádky ve kterých je hodnota sloupce „team_id” stejná jako hodnota sloupce „id” nalezené řádky z tabulky „team” (viz. kapitola 5.3.5) dříve v metodě. Následně pro každou nalezenou lekci metoda najde z tabulky „container” (viz. kapitola 5.3.7) řádek, který přísluší uživateli vyhazovanému z týmu a k lekci. Dále pokud tento řádek existuje smaže docker kontejner, který nalezený řádek představuje, a nakonec smaže tento řádek z databáze. Poslední dva kroky metody jsou úpravy databáze, z kterých je první upravení tabulky „user” řádku uživatele, který je z týmu vylučován, tak že je hodnota sloupce „id” týmu, z kterého je uživatel vyhazován a jeho řádka byla nalezena na začátku metody, odstraněna z sloupce „teams” tohoto řádku. Poslední úprava databáze je upravení řádku získaného na začátku metody tabulky „team” ve sloupci „member_count” na hodnotu o jedna menší, protože byl odebrán jeden uživatel.

Druhý ze dvou souborů je soubor „UserTeamRoutes”, který obsahuje na rozdíl od předchozího souboru jen dva endpointy. Tyto dva zmíněné endpointy jsou „/teams/user/leave”, používaný, pokud uživatel chce opustit tým a „/teams/user/join” užívaný na přidání uživatele do týmu.

První z endpointů v souboru „UserTeamRoutes” je „/teams/user/leave” a jak již bylo zmíněno je používán pro opuštění týmu a volá metodu „leave_team”. Zmíněná metoda nejprve získá json, s kterým je volána a za jeho pomoci následně získá tři řádky z databáze. Řádky jsou z tabulek „team” (viz. kapitola 5.3.5) pro který se hodnota sloupce „name” rovná hodnotě jsonu v bodě „team_name”, „user”, kde platí že řádka patří k uživateli, který funkci skrze frontend (viz. kapitola 4.6) spustil a „lesson”, který má ve sloupci „team_id” nastavenou hodnotu stejnou jako zmíněný řádek z tabulky „team” ve sloupci „id”. Následně se metoda podívá, zda tyto tři řádky existují (pokud neexistují stejně jako u všech podmínek pošle api odpověď s chybou) a v případě že ano podívá se zda řádka z tabulky „user” (viz. kapitola 5.3.3) ve sloupci „id” má stejnou hodnotu jako řádka z tabulky „team” (viz. kapitola 5.3.5) ve sloupci „creator_id”. Tímto metoda zjistí, zda uživatel není vlastníkem týmu, který chce uživatel opustit. Jestliže i tato metoda projde pro každou nalezenou řádku z tabulky „lesson” (viz. kapitola 5.3.6) nalezenou na začátku metody. Pro každou z těchto řádek nalezne metoda řádku z databáze z tabulky „container” (viz. kapitola 5.3.7), která patří k dané lekci a uživateli, který funkci spustil. Následně v případě, že tato řádka z tabulky „container” (viz. kapitola 5.3.7) existuje, smaže docker kontejner se jménem stejné hodnoty jako získaná řádka z tabulky „container” ve sloupci „name”. Poslední operací, která je pro každou lekci provedena smazání řádku nalezeného z tabulky „container” (viz. kapitola 5.3.7) z databáze. Nakonec už metoda provede úpravy databáze, a to sice nejprve řádky nalezené na začátku metody tabulky „user”, kde je z pole sloupce „teams” odebrána hodnota sloupce „id”, který uživatel opouští. Druhou úpravou je upravení řádku opuštěného týmu získaného na začátku metody ve sloupci „member_count” v němž sníží hodnotu o jedna.

Poslední z endpointů je endpoint „/teams/user/join”, která volá metodu „join_team”. Metoda „join_team” v první řadě nejprve získá json s kterým je endpoint zavolán. V dalším kroku metoda získá dvě řádky z databáze, a to sice nejprve z tabulky „user” (viz. kapitola 5.3.3) a to sice řádku příslušící uživateli, který funkci na frontendu zavolal a další řádku z tabulky „team” (viz. kapitola 5.3.5) pro kterou platí, že ve sloupci „team_code” má stejnou hodnotu jako json v bodě „team_code”. Dále metoda zkontroluje, zda tyto dva řádky existují a pokud ano pokračuje dále. V případě však, že řádky neexistují je api vrácena odpověď obsahující korespondující chybu (toto platí pro většinu podmínek v metodě). Pokud tedy oba řádky existují je do proměnné „arr” nastaveno pole obsažené v řádce z tabulky „user” (viz. kapitola 5.3.3) nalezené na začátku metody ze sloupce „teams”. Následně je za pomoci této proměnné zkontrolováno, zda uživatel není součástí týmu, do kterého se chce připojit (metoda se podívá zda není hodnota řádku z tabulky „team” (viz. kapitola 5.3.5) ve sloupci „id” nalezeného na začátku metody obsažená v listu uloženém v proměnné „arr”) a jestli není tým již plný (metoda se podívá zda je hodnota sloupce „member_count” menší než hodnota sloupce „member_limit” v řádku z tabulky „team” (viz. kapitola 5.3.5) nalezeném na začátku metody). Nakonec následují už poslední dvě úpravy databáze, a to nejprve řádky tabulky „user” (viz. kapitola 5.3.3) získané na začátku metody ve sloupci „teams” ve kterém je uloženo pole a je do něj vložen upravené pole, které je kratší o hodnotu sloupce „id” ze řádky tabulky „team” (viz. kapitola 5.3.5) nalezeného na začátku metody, čímž je uživatel odebrán z týmu, a nakonec je upraven zmíněný řádek z tabulky „team” (viz. kapitola 5.3.5) v němž je hodnota ve sloupci „member_count” snížena o jedna.

5.1.5 Endpointy pro platby

Pro platby existují pouze dva endpointy a oba dva jsou definovány v souboru „PaymentRoutes”. Dva endpointy jsou „/payments/create-stripe-session” používající metodu POST a volající metodu „create_session” a „/payments/payment-successfull” také používající metodu POST a volající metodu „payment_suuccessfull”.

První z metod tedy metoda „create_session” nejprve získá z jsonu se kterým je endpoint zavolán. Za pomoci toho co získá metoda z jsonu z dat vytvoří pomocí služby „stripe” do proměnné „session” takzvaný „checkout” s parametry „mode” nastaveným na „payment” pro platbu, „success_url” obsahující kam má být uživatel přesunut v případě úspěšné platby a „cancel_url” v případě neúspěšné platby, „line_items” obsahující další dva parametry „price” obsahující id prodávaného produktu a „quantity” definující kolikrát uživatel produkt bude kupovat a poslední parametr je „metadata” obsahující sloupec „id” tabulky uživatele, který produkt zakupuje. Nakonec tato metoda vrátí pod parametrem „sessionId” id objektu v proměnné „session”, tak aby k tomuto „checkoutu” mohl být uživatel přesunut a mohl zaplatit.

Druhá metoda „payment_successfull” v první části nejprve pomocí kódu získaného z (OpenAI, 2024) získá takzvaná metadata „checkoutu” (viz. Obr. 9) z kterého je tento api endpoint zavolán. Z těchto metadat je následovně vytaženo id uživatele, který platbu prováděl a pro kterého byl tento „checkout” vytvářen (viz. výše). Nakonec už je jen upravena tabulka

„user” příslušící tomuto uživateli ve sloupci „premium” na boolean hodnoty „true” a vrácena v odpovědi api zpráva o úspěšném provedení.

```
payload = request.data
sig_header = request.headers.get('Stripe-Signature')

event = stripe.Webhook.construct_event(
    payload, sig_header, endpoint_secret
)

if event['type'] == 'checkout.session.completed':
    session = event['data']['object']

    user_id = session['metadata']['userId']
```

Obr. 9: Získání metadat po platbě

5.2 Python script

5.2.1 Skript pro kontrolu kontejnerů

První část celého kódu je nastavení některých základních proměnných, které jsou následně používány napříč kódem. První je „docker”, které je takzvané „cli” skrze které lze ovládat docker kontejnery, a hlavně se jich ptát na jejich stav a parametry. Následují tři proměnné nastaveny pomocí tří parametrů, se kterými je skript spouštěn (falsetru, 2013). Tyto tři proměnné jsou „load_limit” (obsahuje informace ohledně maximální zátěže kontejneru v oblasti procesoru), „network_limit” (definuje zátěž v oblasti sítě kontejneru) a nakonec proměnná „container_name” (obsahuje jméno kontejneru). Z proměnné „network_limit” je následně přepočítána nová hodnota do proměnné „network_load_limit” se kterou je následovně ve skriptu počítáno. Další proměnná je „previous_network_stats”, která je používána pro počítání zátěže sítě kontejneru. Poslední částí už je jen jedna konstanta „LOG_FILE” definující cestu k souboru se záznamy ohledně docker kontejneru kontrolovaného instancí tohoto skriptu.

Nejdůležitější částí skriptu, na které celý skript stojí je metoda „main”, která je volána jako první. Tato metoda hned ze začátku zavolá jinou metodu, a to sice metodu „setup_logger” (Margaret, 2024). Metoda „setup_logger” jen vytváří takzvaný „logger” a „handler” pomocí kterých následně skript zapisuje do souborů s koncovkou „.log”. V metodě už dále začíná jen smička, která se opakuje do nekonečna (nemá podmínku) a je přerušena tedy jen příkazem „break”. Celá smička začíná zavoláním vestavěné metody „sleep” díky které je program zastaven na 2 sekundy. Následně pomocí docker „cli” získá metoda docker kontejner se jménem zadaným v proměnné „container_name”. Podle tohoto nalezeného kontejneru je zjištěno, zda je spuštěný. Pokud není kontejner spuštěný je to jednoduše zaznamenáno pomocí zaznamenávacího systému inicializovaného výše pomocí metody „setup_logger”. V opačném případě, pokud kontejner běží jsou nejprve získány statistiky kontejneru do proměnné „stats” a do proměnné „uptime” je pomocí metody „get_container_uptime” nastaveno jak dlouho

kontejner již běží. Další částí metody „main” je kontrola, zda kontejner běží již déle než minutu. V případě že neběží je proces kontroly přeskočen a je to vypsáno pomocí zaznamenávacího systému. Pokud však kontejner již běží déle než minutu jsou použity další tři podmínky na stejné úrovni a všechny tři fungují v podobném formátu. U všech tří podmínek je před nimi nejprve pomocí metody nastavená proměnná podle, které se poté vyhodnocuje, zda podmínka projde. Tato podmínka projde jen pokud proměnná nastavena pomocí metody je nastavena na boolean „false”. Pokud podmínka projde je vypsáno pomocí zaznamenávacího systému, proč byl kontejner zastaven a pomocí metody „stop_container” (Jain, 2024) je následovně zavolán api endpoint „/lessons/user/stop-container” (viz. kapitola 5.1.3) (Jain, 2024) a pokud je jeho odpověď v bodě „status” „true” skript se zastaví. Je však důležité zmínit se o třech metodách před podmínkami a to jsou „check_container_memory”, „check_container_timeout” a „check_container_network”.

První z metod tedy metoda „check_container_memory” kontroluje zde nebyl uživatelem kontejneru přesáhnout limit zabránění paměti ram kontejnerem. Pomocí cyklu je získána pouze první část proměnné „stats” obsáhnuté v parametru metody (tato proměnná obsahuje detailní statistiky kontrolovaného docker kontejneru). Následně je do proměnné „memory_percentage” spočítáno kolik procent paměti ram, kterou může kontejner používat již používá, zapomocí metody „get_memory_percentage”. Dále pokud je proměnná „memory_percentage” větší než proměnná „load_limit” inicializovaná na začátku skriptu je metodou vrácen boolean „false”, čímž bude kontejner zastaven, a v opačném případě je vrácen string obsahující zátěž spočítanou do proměnné „memory_percentage”.

```
def get_memory_percentage(stat):
    memory_stats = stat.get('memory_stats', {})
    memory_usage = memory_stats.get('usage', 0)
    memory_limit = memory_stats.get('limit', 1)

    memory_limit = max(memory_limit - memory_stats.get('stats', {}).get('cache', 0), 1)

    ram_percentage = (memory_usage / memory_limit) * 100

    return ram_percentage
```

Obr. 10: Metoda „get_memory_percentage” pro získání zátěže kontejneru

Druhá z metod je metoda „check_container_timeout”, která kontroluje, zda kontejner neběží zbytečně bez uživatelského přístupu. Tohoto docílí tím, že nejprve získá záznamy kontejneru (to co je vypsáno po spuštění uživatelského příkazu v kontejneru) z těchto záznamů poté získá metoda poslední řádku a tu rozdělí na dva stringy z kterých první obsahuje čas kdy byl záznam proveden a druhý, který obsahuje text záznamu. Dále metoda změní formát, ve kterém je string času zapsán na formát, který lze porovnávat s objektem „datetime” vestavěným do pythonu. Po této změně už pouze počítá metoda, jak dlouho uběhlo od posledního záznamu docker kontejneru a pokud uběhlo více než pět minut vrátí metoda boolean „false”, kterým zastaví docker kontejner. V opačném případě pouze vrátí string obsahující záznam o tom, jak dlouho uběhlo od poslední aktivity (viz. Obr. 11).

```

def check_container_timeout(container):
    logs = container.logs(timestamps=True).decode('utf-8').strip()

    last_log_line = logs.split('\n')[-1]
    timestamp_str, message = last_log_line.split(' ', 1)

    last_interaction_time = datetime.fromisoformat(timestamp_str.replace('Z', '+00:00'))

    elapsed = datetime.now(last_interaction_time.tzinfo) - last_interaction_time

    if elapsed > timedelta(minutes=5):
        return False

    return " - last activity: " + str(elapsed) + "\n"

```

Obr. 11: Metoda „check_container_timeout” pro získání času bez příkazu

Poslední z metod je metoda „check_container_network”, která kontroluje zátěž na síti v ohledu vysílaných bytů a přijímaných bytů. Nejprve metoda získá nejaktuálnější statistiky docker kontejneru a podle nich následovně kontroluje kontejner. K tomuto je používána proměnná „previous_network_stats”. Pokud neobsahuje tato proměnná ještě informace ohledně kontejneru jsou sem přidány do bodů „last_rx” (obsahuje kolik bytů kontejner přijal přes síť) a „last_tx” (obsahuje kolik bytů kontejner vyslal přes síť) pod bod obsahující jméno kontrolovaného docker kontejneru. V opačném případě jsou nejprve získána data v bodě názvu docker kontejneru. Data jsou následně odečtena od aktuálních hodnot a jsou nastavena do proměnných „rx_delta” a „tx_delta” respektive. Dále jsou aktualizovány hodnoty v proměnné „previous_network_stats” „last_rx” a „last_tx” v bodě názvu docker kontejneru na aktuální hodnoty. Následně jsou proměnné „rx_delta” a „tx_delta” převedeny na procenta a pokud jedna z nich obsahuje větší hodnotu než proměnná „network_load_limit” vrátí metoda hodnotu „false” a tím zastaví docker kontejner. Pokud tato podmínka však neprojde je vrácen text obsahující statistiky ohledně sítě kontejneru.

5.2.2 Logovací funkce pro kontrolu kontejnerů

Toto takzvané „logování”, které je v předešlé kapitole (viz. kapitola 5.2.1) nazýváno zaznamenáváním funguje na bázi jedné třídy nazvané „CircularBufferHandler”, která rozšiřuje třídu „Handler”. Třída má takzvaný konstruktor, který má tři parametry podle, kterých nastavuje tři parametry třídy a to sice „file_name” obsahující cestu k souboru do kterého bude zapisovat, „max_logs” obsahující maximum řádek zapsaných v souboru (po kolika řádcích začne soubor přepisovat starší řádky) a poslední parametr „buffer” vytvořený pomocí metody „deque”, která vytváří lépe upravovatelný list (parametr obsahuje veškeré záznamy). Následují dvě metody této třídy a to sice „emit” a „flush_to_file”. Metoda „emit” je používána pro přidávání záznamů do parametru „buffer” a metoda „flush_to_file”, která otevře soubor jako objekt a do tohoto souboru zapíše veškerý obsah parametru „buffer” (viz. Obr. 12).

```

class CircularBufferHandler(logging.Handler):
    def __init__(self, file_name, max_logs=100):
        super().__init__()
        self.file_name = file_name
        self.max_logs = max_logs
        self.buffer = deque(maxlen=max_logs)

    def emit(self, record):
        log_entry = self.format(record)
        self.buffer.append(log_entry)

    def flush_to_file(self):
        with open(self.file_name, 'w') as log_file:
            log_file.write("\n".join(self.buffer))
            log_file.write("\n")

```

Obr. 12: Třída „CircularBufferHandler” pro záznamování

5.2.3 Skript pro kontrolu lekcí

Tento skript slouží pro mazání lekcí (viz. kapitola 5.3.6) z databáze (mazání řádek z tabulky „lessons”), které už skončili před dříve než jedním měsícem. Toto je prováděno skriptem a ne funkcí v databázi (viz. kapitola 5.3.11), protože musí v tomto případě být smazány docker kontejnery a ne jen řádka v databázi.

Tento skript nejprve vytvoří klienta přes, kterého je možné ovládat program docker do proměnné „client”. Následně je do proměnné „supabase” nastaven klient této služby přes který je možné získávat data z databáze, přidávat je tam a podobné. Následně je už jen volána metoda „main” obsahující hlavní cyklus tohoto skriptu.

Výše zmíněná metoda „main” nejprve ve také zmíněném cyklu získá veškeré řádky z tabulky „lesson” (viz. kapitola 5.3.6), a pro každou zjistí, zda je ve sloupci „end_time” datum starší než 1 měsíc a v případě, že ano je zavolána metoda „deleteTable” s parametrem řádky pro kterou byla kontrola prováděna. V opačném případě se cyklus pouze opakuje.

Zmíněná metoda „deleteTable” nejprve z databáze vymaže řádku, kterou dostala v parametru a následně uživateli, který tuto lekci vlastnil v tabulce „limitations” (viz. kapitola 5.3.4), ve sloupci „lessons” sníží počet lekcí tak aby mohl vytvořit novou lekci. Po tomto metoda zavolá metodu „deleteContainers”, která nejprve získá veškeré kontejnery a pak pro každý z nich zkontroluje, zda jejich jméno začíná jménem smazané lekce a pokud ano tento kontejner metoda případně zastaví a smaže (veškeré kontejnery mají jméno ve formátu „název lekce-náhodné písmena”) (viz. Obr. 8).

5.3 Supabase

Jak již bylo zmíněno „supabase” je služba starající se o databázi, přihlašování uživatelů, ale je důležité zmínit, že se stará ještě o funkce v databázi psané v jazyku plpgsql, které upravují databázi, když jsou volány. Služba dovoluje vytvořit takzvané „triggers”, které volají některé funkce v případě, že je podmínka pro jejich spuštění splněna. Poslední částí, kterou je že tato služba je v případě tohoto projektu „hostována” lokálně a je potřeba upravit její konfigurační soubor „.env”.

5.3.1 Konfigurační soubor

Tento soubor je ve formátu, který je často používán pro konfiguraci jiných projektů a i dokonce tohoto projektu a to sice „.env”. To znamená, že jsou zde konstanty definované velkými písmeny a jejich hodnoty jsou definovány pomocí znaku „=” a bez jakýchkoliv uvozovek. V tomto souboru je většina konstant již definovaných, ale je potřeba některé změnit, nebo přidat pro správnou funkčnost projektu.

Nejprve je podle návodu dostupného na oficiálních stránkách této služby potřeba vygenerovat pomocí stránek projektu vygenerovat takzvaný „anon_key” a přidat ho do odpovídající konstanty („ANON_KEY”) a „service_key” a přidat také ho přidat do konstanty („SERVICE_ROLE_KEY”).

Následovně bylo potřeba změnit některé obecné konstanty jako je přihlašovací jméno uživatele v konstantě „DASHBOARD_USERNAME” a heslo pro uživatele v proměnné „DASHBOARD_USERNAME”. Toto heslo je používáno pro přístup do prostředí pro nastavování služby nikoliv do aplikace. Poslední takováto obecná změna byla adresa webové aplikace pro, kterou je tato služba spuštěna v konstantě „SITE_URL”.

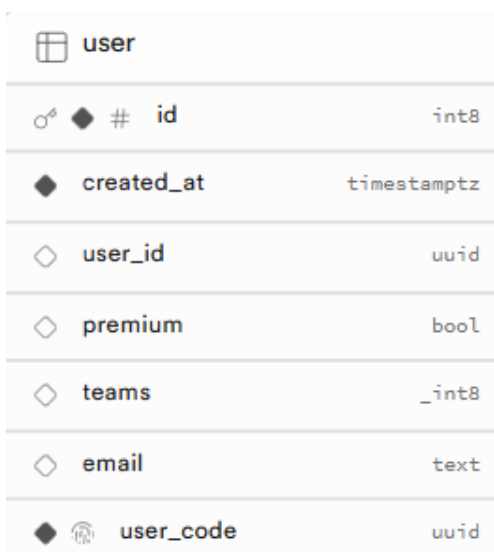
Další změněné proměnné se všechny týkají zaslání emailů. Pro tuto funkčnost bylo potřeba změnit řadu proměnných. Navíc bylo potřeba vytvořit email, který byl pro jednoduchost vytvořen u poskytovatele „seznam.cz”, který z předešlých zkušeností funguje nejlépe.

5.3.2 Struktura databáze

Celá struktura databáze tohoto projektu je rozdělena do 6 tabulek a zbytek, které má služba vestavěna a nebyly nijak upravovány. Všechny tyto tabulky jsou součástí schématu „public”. Tyto tabulky jsou „user” obsahující detailnější informace ohledně uživatele, které služba neukládá, „limitations”, která obsahuje limity a další informace k uživateli (rozšiřuje tabulku „user”), „team” obsahující veškeré týmy a informace k těmto týmům, „lesson” definující veškeré lekce a jejich detaily, „container”, která obsahuje důležité informace ohledně docker kontejneru a nakonec „container_script” ve kterém jsou obsáhnuty informace ohledně skriptu kontrolujícím zátěž kontejneru. Je důležité navíc zmínit, že všechny tabulky mají sloupec „id”, který je jejich identita a primární klíč. Další sloupec, který je ve všech tabulkách je „created_at” obsahující čas, kdy byla řádka vytvořena.

5.3.3 Tabulka „user”

Tato tabulka slouží jako základ aplikace, protože se k ní stahuje většina dalších tabulek. První sloupec tabulky je „user_id”, který je primární klíč odkazující na interní tabulku služby „supabase” obsahující uživatele. Další jsou čtyři proměnné, které jsou specifické pouze pro tuto tabulku. První je „premium”, která je pouze boolean a určuje, zda má uživatel zakoupen prémiový účet. Další sloupec je „teams” obsahující pole vyplněno „id” tabulek týmů (viz. kapitola 5.3.5) a určující, kterých týmů je uživatel členem. Třetí sloupec je „email” obsahující text definující email uživatele. Poslední sloupec je „user_code”, který obsahuje unikátní kód aktuálního uživatele a musí být unikátní. (viz. Obr. 13)



user	
id	int8
created_at	timestampz
user_id	uuid
premium	bool
teams	_int8
email	text
user_code	uuid

Obr. 13: Tabulka „user” obsahující profil uživatele

5.3.4 Tabulka „limitations”










„limitations” je další tabulka, která obsahuje limitace uživatele v aplikaci. První sloupec této tabulky je „extra_id” referující na řádku v tabulce „user” (určuje pro kterého uživatele jsou tyto limitace). Následují tři dvojice „team_limit” a „teams” určující kolik týmů může uživatel vytvořit a kolik už jich uživatel vytvořil, „lesson_limit” a „lessons” definující kolik uživatel může vytvořit lekcí a kolik už jich vytvořil a poslední „resets” a „reset_limit” obsahující kolikrát již uživatel resetoval dnes kontejner a kolikrát tuto akci provést může. Poslední dva sloupce jsou „running_container_id” určující, který kontejner má uživatel aktuálně spuštěný, a „checking_container_id” definující, pokud pro to uživatel má práva, id kontejneru, který uživatel zrovna kontroluje a má ho spuštěný. (viz. Obr. 14)

limitations	
♂ # id	int8
created_at	timestampz
extra_id	int8
team_limit	int8
teams	int8
lesson_limit	int8
lessons	int8
running_container_id	int8
checking_container_id	int8
resets	int8
reset_limit	int8
team	

Obr. 14: Tabulka „limitations” rozšiřující profil uživatele

5.3.5 Tabulka „team”













Tabulka „team” obsahuje informace ohledně vytvořených týmů. První unikátní sloupec pro tuto tabulku je „team_code” obsahující vždy unikátní uuid podle kterého se uživatelé následně připojují do týmu. Další sloupec je „creator_id”, který je primary key a odkazuje na řádku v tabulce „user” a tedy ukazuje na tvůrce týmu. Následující sloupec je „name”, který obsahuje pouze text se jménem týmu. Nakonec následuje další dvojice sloupců „member_count” a „member_limit” definující aktuální počet uživatelů a maximální počet uživatelů, který tým může mít. (viz. Obr. 15)

team	
  # id	int8
 created_at	timestampz
  team_code	uuid
 creator_id	int8
 name	text
 member_count	int8
 member_limit	int8

Obr. 15: Tabulka „team” obsahující týmy

5.3.6 Tabulka „lesson”











Tato tabulka obsahuje jednotlivé lekce, které byly vytvořeny, a jejich detaily. První sloupec je „creator_id” obsahující stejně jako u tabulky „team” (viz. kapitola 5.3.5) „id” uživatele, který ho vytvořil. Následují dva sloupce „start_time” a „end_time” obsahující informace ohledně toho, kdy čas na vyplnění lekce začíná/začal a kdy končí/skončil. Další sloupec je „team_id”, který je primární klíč, ukazující, pro který tým je lekce určena. Poslední čtyři sloupce této funkce jsou „task” obsahující úkol pro uživatele vyplňujícího lekci, „setting”, který je json a obsahuje nastavení docker kontejneru, který bude pro uživatele případně vytvořen, „name” obsahující jméno lekce a „packages” ve kterém je pole s programy, které budou do docker kontejneru po jeho vytvoření pro uživatele přidány. (viz. Obr. 16)

lesson	
  #	id int8
	created_at timestamptz
	creator_id int8
	start_time timestamp
	end_time timestamp
	task text
	team_id int8
	settings json
 	name text
	packages _text

Obr. 16: Tabulka „lesson” obsahující lekce

5.3.7 Tabulka „container”









Další z tabulek čili tabulka „container” obsahuje informace ohledně docker kontejneru. Prvním sloupcem tabulky je „lesson_id”, který je primary key a odkazuje na řádku v tabulce „lesson” (viz. kapitola 5.3.6) pro kterou byl kontejner vytvořen. Další primary key této tabulky je „user_id”, který je primary key odkazující na řádku tabulky „user” a tedy na uživateli který nechal tento kontejner vytvořit. Další sloupce jsou „login” a „password” obsahující přihlašovací údaje do kontejneru. Následuje sloupec „running” určující, zda kontejner běží a je to pouze boolean. Poslední sloupec je „port” a obsahuje číslo portu na kterém kontejner běží a je tu pro to aby ho uživatel mohl otevřít. (viz. Obr. 17)

container	
  # id	int8
 created_at	timestampz
 lesson_id	int8
 name	text
 login	text
 password	text
 running	bool
 user_id	int8
 port	int8

Obr. 17: Tabulka „container” obsahující kontejnery

5.3.8 Tabulka „container_script”

Poslední tabulka tedy tabulka „container_script” obsahuje detaily ke skriptu, který kontroluje kontejner. Prvním sloupcem je „container_id”, který je primary key a odkazuje na řádku v tabulce „container” (viz. kapitola 5.3.7) pro kterou je vytvořen. Další sloupce jsou „container_name” obsahující jméno uživatele, „pid” ve kterém je uložen id běžícího procesu, „settings”, který je json a obsahuje omezení kontejneru a poslední sloupec je „running”, který stejně jako v tabulce „container” (viz. kapitola 5.3.7) je boolean a určuje, zda je skript spuštěn. (viz. Obr. 18)

container_script	
  # id	int8
 created_at	timestampz
 container_id	int8
 container_name	text
 pid	int8
 settings	jsonb
 running	bool

Obr. 18: Tabulka „container_script” obsahující skript pro kontrolu kontejneru

5.3.9 „Triggers” v databázi

Tyto takzvané „triggery” neustále kontrolují databázi a pokud je jejich podmínka splněna jsou schopni upravit databázi, zavolat endpoint api, nebo jak je to v případě tohoto projektu volat funkci. (viz. kapitola 5.3.11)

V celé webové aplikaci jsou nastaveny dva „triggery” první z nich je „on_team_delete”, který volá funkci „remove_team_id_from_users”, v případě že je z tabulky „team” smazána řádka (viz. kapitola 5.3.5) a je napsán, aby bylo id týmu odebráno ze sloupce „teams” tabulky „user” (viz. kapitola 5.3.3) a nezůstávaly v tomto sloupci zbytečně id smazaných týmů. Druhý „trigger” je pojmenován „on_auth_user_created” a ten je volán v případě, pokud je vytvořen nový uživatel, volá metodu „handle_new_user” (viz. kapitola 5.3.11) a je zde, aby při vytvoření nového uživatele v aplikaci byly přidány i řádky do tabulek, které rozšiřují informace o uživateli.

5.3.10 „Cron jobs”

Takzvaný „cron job” je funkce, která v daný čas provede některou akci s databází. V případě této webové aplikace je nastaven pouze jeden „cron job” a to sice jeden pojmenovaný „daily_reset”. Tento „cron job” každý den o půlnoci volá funkci „daily_reset” (viz. kapitola 5.3.11), která následně nastaví některé sloupce databáze.

5.3.11 Funkce v databázi

Veškeré funkce v této databázi jsou napsány v jazyce „plpgsql” a mají účel měnění jejich částí. První z nich je funkce „daily_reset” a ta jedinou řádkou kódu změní všechny řádky v tabulce „limitations” v sloupci „resets” na 0 (viz. kapitola 5.3.4). Další z funkcí je funkce „handle_new_user” a ta vytváří dvě řádky. První z těchto řádek je vytvořena v tabulce „user” (viz. kapitola 5.3.1), v ní nastaví sloupce „email” a „user_id” (tato metoda je volána pomocí „triggerů”, když je vytvořen nový uživatel) a zbytek nechá nastaven na takzvané „default” hodnoty (viz. návod na toto). Druhá řádka je vložena do tabulky „limitations” (viz. kapitola 5.3.4) a je nastaven její sloupec „extra_id” na sloupec „id” předchozí řádky v tabulce „user” (viz. kapitola 5.3.3). Poslední funkce je funkce „remove_team_id_from_users”, která v případě zavolání upraví tabulku „user” tak, že ji v sloupci „teams” odebere z pole, který sloupec obsahuje, „id” smazaného řádku z tabulky „team” (viz. kapitola 5.3.5) (tato funkce je volaná takzvaným „triggrem”, který tuto funkci volá jen v případě, že je smazán řádek z tabulky „team” (viz. kapitola 5.3.5)).

6 ZÁVĚR

Po dokončení této práce stojím za názorem, že jsem vytvořil responzivní a funkční webovou aplikaci odpovídající zadání. Práce na tomto projektu mi umožnila rozšířit si obzory ohledně programování a pracování s novými frameworky a službami. Rozšířil jsem si obzory ohledně používání dockeru, který je důležitou částí dnešního vývoje aplikací. Následně jsem si osvojil používání služeb jako je právě „supabase”. Byl jsem schopen překonat, nebo vymyslet řešení některých komplexnějších problémů. Ve finále jsem dosáhl výsledku, se kterým jsem spokojen a splňuje zadání chtěl bych pokračovat ve vývoji a zlepšování aplikace.

POUŽITÁ LITERATURA

- aarcoraci. (2018, December 25). Dynamically show components depending on the current route, vuejs. Stack Overflow. Retrieved February 21, 2025, from <https://stackoverflow.com/questions/53926267/dynamically-show-components-depending-on-the-current-route-vuejs>
- Bakuriu. (2013, July 25). How to terminate process from Python using pid? Stack Overflow. Retrieved February 21, 2025, from <https://stackoverflow.com/questions/17856928/how-to-terminate-process-from-python-using-pid>
- Coding Shiksha. (2022, 3 25). Vue.js Stripe Payment Gateway Example to Integrate One Time Product Checkout Using vue-stripe in JS. youtube. https://www.youtube.com/watch?v=g2Dtt4_dQQ4
- Dascalescu, D., & blackgreen. (2009, June 4). What is the JavaScript version of sleep()? Stack Overflow. Retrieved February 21, 2025, from <https://stackoverflow.com/questions/951021/what-is-the-javascript-version-of-sleep>
- falsetru. (2013, July 9). How do I run Python script using arguments in windows command line. Stack Overflow. Retrieved February 21, 2025, from <https://stackoverflow.com/questions/17544307/how-do-i-run-python-script-using-arguments-in-windows-command-line>
- Fireship. (2022, 11 25). Supabase in 100 Seconds. youtube. <https://www.youtube.com/watch?v=zBZgdTb-dns>
- garo. (2019, 7 8). garo/shellinabox. dockerhub. <https://hub.docker.com/r/garo/shellinabox>
- GeeksforGeeks. (2023, 2 17). How to detect whether the website is being opened in a mobile device or a desktop in JavaScript ? GeeksforGeeks. <https://www.geeksforgeeks.org/how-to-detect-whether-the-website-is-being-opened-in-a-mobile-device-or-a-desktop-in-javascript/>
- gerrit. (2010, October 4). Generate password in Python. Stack Overflow. Retrieved February 21, 2025, from <https://stackoverflow.com/questions/3854692/generate-password-in-python>
- Hadzhiev, B. (2024, 3 3). Check if String contains only Letters and Numbers in JS. bobbyhadz.com. <https://bobbyhadz.com/blog/javascript-check-if-string-contains-only-letters-and-numbers#check-if-string-contains-only-letters-and-numbers-in-javascript>

Jain, S. (2024, February 23). How To Create And Use .env Files In Python. GeeksforGeeks. Retrieved February 21, 2025, from <https://www.geeksforgeeks.org/how-to-create-and-use-env-files-in-python/>

Jain, S. (2024, August 12). GET and POST Requests Using Python. GeeksforGeeks. Retrieved February 21, 2025, from <https://www.geeksforgeeks.org/get-post-requests-using-python/>

KingKongFrog. (2013, May 7). Check if a string contains an email address? Stack Overflow. Retrieved February 21, 2025, from <https://stackoverflow.com/questions/16424659/check-if-a-string-contains-an-email-address>

LearnVue. (2022, 1 4). Vue 3 + Firebase Authentication in 10 Minutes. youtube. <https://www.youtube.com/watch?v=xceR7mrrXsA>

Margaret, D. (2024, 9 4). Efficient Circular Buffer Implementation in Python 3. DNM. <https://dnmtechs.com/efficient-circular-buffer-implementation-in-python-3/>

McCrossan, R. (2012, April 21). How to detect string which contains only spaces? Stack Overflow. Retrieved February 21, 2025, from <https://stackoverflow.com/questions/10261986/how-to-detect-string-which-contains-only-spaces>

patorjk. (n.d.). Text to ASCII Art Generator (TAAG). patorjk.com. Retrieved February 21, 2025, from <https://patorjk.com/software/taag/#p=display&f=Graffiti&t=Type%20Something%20>

Szögyényi, Z. (2021, 12 9). How to Build a Dark Mode Switcher With Tailwind CSS and Flowbite. freeCodeCamp. <https://www.freecodecamp.org/news/how-to-build-a-dark-mode-switcher-with-tailwind-css-and-flowbite/>

thorwebdev. (2023, April 19). Supabase- check if a Postgres JSON array contains a number. Stack Overflow. Retrieved February 21, 2025, from <https://stackoverflow.com/questions/76052148/supabase-check-if-a-postgres-json-array-contains-a-number>

Trott. (2012, April 22). Split string on the first white space occurrence. Stack Overflow. Retrieved February 21, 2025, from <https://stackoverflow.com/questions/10272773/split-string-on-the-first-white-space-occurrence>

Webnoob. (2019, 3 29). Password Strength Checker with Vue JS. youtube. <https://www.youtube.com/watch?v=qQP3WBSn2yU>

SEZNAM OBRÁZKŮ

Obr. 1: Graf struktury celého projektu	3
Obr. 2: Struktura stránek aplikace	6
Obr. 3: Vzhled komponentu se stylem příkazové řádky (levá část světlý režim a pravá tmavý režim)	8
Obr. 4: Skript pro kontrolu síly hesla	11
Obr. 5: Vzhled komponentu se stylem jedno sloupcový „selector” (levá část světlý režim a pravá tmavý režim)	16
Obr. 6: Vzhled komponentu se stylem dvou sloupcový „selector” (levá část světlý režim a pravá tmavý režim)	20
Obr. 7: Struktura propojení api s frontendem	25
Obr. 8: Způsob mazání kontejnerů	28
Obr. 9: Získání metadat po platbě	35
Obr. 10: Metoda „get_memory_percantage” pro získání zátěže kontejneru	36
Obr. 11: Metoda „check_container_timeout” pro získání času bez příkazu	37
Obr. 12: Třída „CircularBufferHandler” pro záznamování	38
Obr. 13: Tabulka „user” obsahující profil uživatele	40
Obr. 14: Tabulka „limitations” rozšiřující profil uživatele	41
Obr. 15: Tabulka „team” obsahující týmy	42
Obr. 16: Tabulka „lesson” obsahující lekce	43
Obr. 17: Tabulka „container” obsahující kontejnery	44
Obr. 18: Tabulka „container_script” obsahující skript pro kontrolu kontejneru	45