

**Gymnázium, Praha 6, Arabská 14**

Obor programování



Maturitní práce

**DUCK**  
**(Docker Ultimate Container Keeper)**

Administrační systém pro Docker kontejnery

Martin Voplakal, 2.E

duben 2024

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 Praze dne:

Martin Voplakal

## **ANOTACE A ZADÁNÍ PROJEKTU**

Tato maturitní práce se zabývá návrhem a implementací administračního systému pro provozovatele webových aplikací, který umožňuje jeho zákazníkům spouštět vlastní instance aplikací v podobě Docker kontejnerů. Systém poskytuje formulářové webové rozhraní, kde zákazník vyplní potřebné údaje pro spuštění požadované služby, kterou může být například WordPress nebo jiná webová aplikace. Po odeslání formuláře je zákazníkovi automaticky a bez zásahu administrátora spuštěna jeho instance služby. Každá aplikace může podléhat určitému měsíčnímu poplatku za provoz a každý uživatel má možnost zaplatit toto předplatné pomocí platební brány.

Procesy jsou navrženy tak, aby si je administrátor mohl co nejvíce přizpůsobit za pomoci inicializačních skriptů pro jednotlivé aplikace. Systém umožňuje omezit hardwarové zdroje každého kontejneru (CPU, RAM, SWAP, disk) na určité hodnoty, aby se zabránilo přetížení serveru a omezení ostatních zákazníků.

# OBSAH

1.	Úvod.....	5
2.	Definice názvů a pojmů .....	5
3.	Využití .....	6
3.1	Konkrétní příklady .....	6
4.	Funkce administračního rozhraní.....	7
4.1	Přidání aplikace do systému .....	7
4.1.1	Bash skripty.....	7
4.1.2	Inicializační formulář.....	8
4.1.3	Nastavení měsíčního poplatku .....	9
4.1.4	Defaultní Docker image.....	10
4.2	Správa instancí aplikací .....	10
4.2.1	Vytvoření instance .....	10
4.2.2	Upgrade instance.....	11
4.3	Detail a nastavení samostatné instance .....	11
4.3.1	Resource limits .....	11
4.3.2	Expirace předplatného .....	12
4.4	Přihlášení uživatele .....	13
5.	Použité technologie.....	13
5.1	Vue.js .....	13
5.2	CoreUi.....	14
5.3	Node.js .....	15
5.4	MongoDB .....	16
5.5	JWT.....	16
5.5.1	Princip fungování přihlašování na bázi JWT:.....	16
5.6	Caddy .....	17
5.6.1	Využití Caddy REST API.....	17
5.7	Docker.....	17
5.8	Stripe.....	18
6.	Prezentace práce .....	18
7.	Závěr .....	19
8.	Použité zdroje .....	20
9.	Seznam obrázků.....	20

# 1. Úvod

Tento projekt vznikl s cílem splnit mojí praktickou potřebu použít administrační systém, který umožní široké veřejnosti vyzkoušet a využívat systém DEH (Digital Education Hub, Digitální učebnicový systém) se kterým jsme se v loňském roce umístili na 5. místě v republikovém kole soutěže Středoškolská odborná činnost. Jelikož je systém DEH vyvinut jako webová aplikace a provozuje se v Docker kontejnerech, pro každou instanci této aplikace je nutné vytvořit a napojit databázi, konfigurovat doménu, SSL certifikát apod. Spuštění několika instancí systému DEH je proto rutinní, ale poměrně obsáhlá práce v terminálu serveru.

Administrační systém pro Docker kontejnery umožňuje hostování libovolné aplikace v prostředí Docker a je navržen tak, aby každý uživatel měl možnost si plně automaticky a bez nutnosti zásahu administrátora spustit svoji vlastní instanci hostované aplikace. Systém je primárně určen pro hostování webových aplikací nebo podobných systémů jako je např. WordPress, kdy každý uživatel potřebuje svoji instanci aplikace. Prakticky ale tento systém může nalézt i zcela odlišné způsoby využití. Viz kapitola 3. *Využití*.

## 2. DEFINICE NÁZVŮ A POJMŮ

Tento projekt byl nazván *DUCK – Docker Ultimate Container Keeper* (dále v textu se používá již pouze tato zkratka) a pro úplnost a ucelení grafického designu uživatelského rozhraní jsem vytvořil i logo.



Obrázek 1 Logo systému DUCK

Systém DUCK je navržen pro hostování jiných aplikací. Tyto aplikace budu dále označovat jako *hostované aplikace*.

*Instance aplikace* označuje konkrétní spuštěnou verzi dané aplikace.

Pojmem *uživatel* dále myslíme uživatele systému DUCK, který si s jeho pomocí zprovozní hostovanou aplikaci např. DEH, WordPress a další.

## 3. VYUŽITÍ

Systém DUCK je primárně určen k provozování webových aplikací, své využití však může nalézt všude tam, kde je potřeba spravovat větší množství instancí jakékoliv aplikace. Taková aplikace musí být určena k provozu na serveru a každý klient může požadovat jinou verzi této aplikace například z důvodu kompatibility se systémem klienta. Skvěle se také hodí pro prezentování aplikace s cílem dát lidem možnost si ji zdarma a téměř okamžitě vyzkoušet.

### 3.1 KONKRÉTNÍ PŘÍKLADY

DEH (Digitální učebnicový systém) byl již výše zmíněn a využívá všechny funkce systému DUCK. Je to modelový příklad využití.

Systém WordPress je další možnou hostovanou aplikací. Mimo své profesionální využití k tvorbě webů a správě obsahu, je hojně využíván také ve výuce na školách, kdy se studenti seznámí se základy tvorby webových stránek v administračních systémech. [1] [2] Systém je možné zdarma vyzkoušet např. u poskytovatele hostingu WordPress.com, ale funkce dostupné zdarma jsou pro detailnější výuku nedostatečné. Není možné například instalovat pluginy, bez kterých není možné vytvořit takřka žádný, v praxi použitelný, web. Aby škola nebyla závislá na externím hostingu a mohla studenty naučit jak si např. vytvořit vlastní e-shop, využití systému DUCK se k tomu přímo nabízí. WordPress je v podobě Docker Image již oficiálně vydáván. [3] Stačí tedy jen spustit SQL databázi a vytvořit příslušné scripty<sup>1</sup> do systému DUCK, které WordPress s databází propojí.

Možnost využít tento systém je také při výuce předmětu operační systémy, kdy žáci potřebují k výuce práce se soubory, oprávněními či filesystémy přístup k root účtu operačního systému UNIX. Jelikož se na Docker kontejnery dá nahlížet jako na samostatné operační systémy s vlastním souborovým systémem i oprávněními, je při správné konfiguraci možné dát uživateli root ssh přístup do Docker kontejneru, aniž by mohl z kontejneru ovládnout jiný kontejner nebo celý server. Myšlenka je zde taková, že by každý student vyplnil do formuláře své požadované root heslo a jeho kontejner s ssh-server by byl spuštěn a spravován systémem DUCK. Možné je využít už připravený Docker image<sup>2</sup> obsahující *ssh-server*, nebo si takový Docker image připravit podle vlastních požadavků. Na konci vyučování této látky by pak učitel za pomoci systému DUCK tyto kontejnery snadno odstranil. Kontejnerům je také možné z bezpečnostních důvodů omezit přístup k internetu pomocí proxy serveru, nicméně to zde nebudu konkrétněji rozebírat, protože to není předmětem mé práce.

---

<sup>1</sup> viz. kapitola 4.1.1 *Bash scripty*

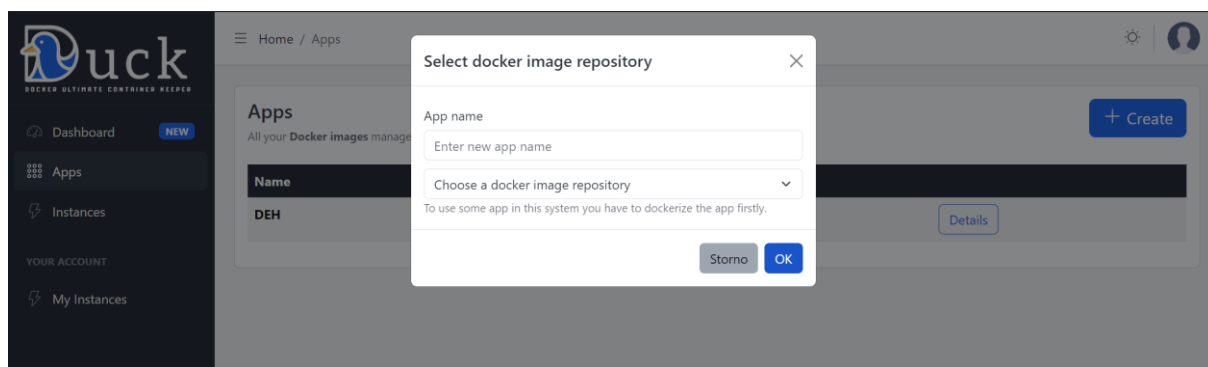
<sup>2</sup> <https://hub.docker.com/r/corbinu/ssh-server>

## 4. FUNKCE ADMINISTRÁČNÍHO ROZHRAŇÍ

V následujících kapitolách se věnuji jednotlivým funkcím a nastavením systému DUCK. Pokud se administrátor rozhodne systém DUCK použít ke správě své hostované aplikace, je nutné aplikaci do systému registrovat a nastavit její konfigurační skripty.

### 4.1 PŘIDÁNÍ APLIKACE DO SYSTÉMU

Abychom mohli hostovanou aplikaci do systému přidat, musí splňovat předpoklad, že se již na serveru nachází v podobě otestovaného a funkčního Docker image. DUCK komunikuje se systémem Docker pomocí terminálu a při vytváření hostované aplikace dá administrátorovi na výběr ze všech Docker image které se na serveru nacházejí.



Obrázek 2 Přidání aplikace do systému

#### 4.1.1 BASH SCRIPTY

Pro každou aplikaci v systému DUCK je nutné vytvořit několik Bash<sup>3</sup> scriptů. Tyto skripty jsou specifické pro každou webovou aplikaci, jelikož každá aplikace může mít jiné parametry. Skripty tak představují propojovací komponent mezi hostovanou aplikací a systémem DUCK.

Aby skripty mohly přednastavit prostředí webové aplikace dle požadavků uživatele, do proměnných mají před svým spuštěním vložena data z uživatelem vyplněného inicializačního formuláře, který je popsán v následující kapitole. Tato data však musejí být předem ošetřena proti spuštění, aby nebylo možné provést Cross Site Scripting (XSS) attac, kdy útočník vyplní formulář a data ve formuláři jsou následně interpretována a spuštěna jako škodlivý kód. To by v našem případě, kdy se jedná o Bash script mělo pro celou infrastrukturu likvidační důsledky.

V tomto odstavci jsou popsány všechny skripty, které se v systému DUCK používají a administrátor je nastaví pro každou hostovanou aplikaci.

---

<sup>3</sup> Bash – Bourne again shell

- **Init script** – Inicializační script, který se spustí pouze jednou při vytvoření nové instance hostované aplikace. Jeho cílem je konfigurovat novou instanci aplikace podle požadavků uživatele, vytvořit databázi, připravit konfigurační soubor nebo naplnit databázi placeholder daty
- **Run script** – Spouštěcí script zajišťuje spuštění Docker kontejneru. Jako poslední příkaz tohoto scriptu musí být *docker run*, který spustí kontejner a vrátí jeho ID pro následné uložení do databáze.
- **Delete script** – Tento script se spustí pouze jednou ve chvíli, kdy se administrátor nebo uživatel rozhodne aplikaci smazat včetně všech uložených dat. Jeho cílem je zajistit hlavně smazání databáze, protože o souborovou složku aplikace se postará systém DUCK.

Scripty, jsou před uložením vždy kontrolovány na syntaktické chyby pomocí *bash -n*, kdy *-n* je přepínač, který umožní čtení příkazů ze vstupu, ale zabrání jejich provedení. Tato kontrola syntaxe odhalí většinu běžných syntaktických chyb, kromě nevalidních názvů vnějších příkazů.

The screenshot shows the 'getting started' configuration page in the Duck hosting management interface. The page is divided into several sections:

- Images:** A table showing details for the 'docker/getting-started' image, including Containers (N/A), CreatedAt (2022-12-22 21:49:18 +0100 CET), CreatedSince (14 months ago), Digest (<none>), ID (3e4394f6b72fcef2217067a717f184d5d828afa9623867d68fce4f9d862b6c), Repository (docker/getting-started), SharedSize (N/A), Size (47MB), Tag (latest), UniqueSize (N/A), and VirtualSize (46.96MB). A 'Set default' button is at the bottom.
- Containers:** A table showing details for the 'docker/getting-started' container, including Containers (N/A) and CreatedAt (2019-11-28 18:22:12 +0100 CET).
- Init script:** A section with tabs for 'Init script', 'Run script', 'Delete script', and 'User init form'. The 'Init script' tab is active, showing a script editor with the following content:
 

```
1 echo MongoDB_URI="mongodb://..." >> .env
2 echo "load fixtures and init DB ..."
3
```
- Price:** A section with input fields for 'Price per month' (set to 0), 'Number of try for free days' (set to 0), and 'This is expiry period'.
- Domain:** A section with a text input field for 'Domain' (set to example.com) and a note: 'Every instance will be given a domain instance-name-your-domain.com'.

Obrázek 3 Nastavení hostované aplikace

## 4.1.2 INICIALIZAČNÍ FORMULÁŘ

V pravém sloupci nastavení hostované aplikace se nachází editory scriptů a editor *User init form*, tedy editor inicializačního formuláře, který uživatel vyplní, pokud chce spustit svoji vlastní instanci hostované aplikace. Tento editor se zobrazuje v poslední záložce vedle editoru scriptů v podobě náhledu. Následně je možné editor otevřít do maximalizovaného okna viz obr. 4.



### Html form content

Between <form> </form> tags

**you have to include:**

`<input name="inputEmail" required>`

name attributes have to be [a-zA-Z0-9\_]

```

1 <div class="mb-3">
2 <label for="exampleInputEmail1" class="form-label">Email address</label>
3 <input type="email" class="form-control" id="exampleInputEmail1" aria
  -describedby="emailHelp" name="inputEmail" required>
4 <div id="emailHelp" class="form-text">We'll never share your email with
  anyone else.</div>
5 </div>
6 <div class="mb-3">
7 <label for="instanceName" class="form-label">Instance name</label>
8 <input type="text" class="form-control" id="instanceName" name
  ="instanceName" required>
9 </div>
10 <div class="mb-3 form-check">
11 <input class="form-check-input" type="hidden" id="exampleCheck1" value
  ='0' name='checkboxs'> <!--set unchecked value-->
12 <input type="checkbox" class="form-check-input" id="exampleCheck1" name
  ="checkboxs" value="1" checked>
13 <label class="form-check-label" for="exampleCheck1">Check me out</label>
14 </div>
15
16 <label for="customRange1" class="form-label">Example range</label>
17 <input type="range" class="form-range" id="customRange1" name="formRange"

```

### Preview

Email address

We'll never share your email with anyone else.

Instance name

☒ Check me out

Example range

### Data object

```

{
  "inputEmail": "",
  "instanceName": "",
  "checkboxs": "1",
  "formRange": "50"
}

```

Obrázek 4 Editor inicializačního formuláře

Tento editor byl vytvořen tak, aby v pravém sloupci obsahoval live preview (náhled, který se v reálném čase vykresluje podle editovaného kódu), pod ním se pak dále nachází *Data object*, kde si administrátor při editování může ověřit, zda jsou input elementy formuláře správně konfigurované, aby se data z formuláře dala v pořádku načíst. Tyto vyplněné údaje, které uživatel vyplní, jsou při inicializaci instance hostované aplikace vloženy jako proměnné do všech scriptů, kde podle nich může být aplikace přednastavena či jinak upravena. V neposlední řadě se také vyplněný název *Instance name* použije jako název subdomény, na které je instance aplikace k dispozici.

Tento formulář bylo také nutné ošetřit proti DDOS útoku, protože spouštění instancí aplikací je poměrně výpočetně náročný proces a automatické generování requestů z formuláře je velmi snadné. Ošetřil jsem tedy formulář proti robotickému vyplnění pomocí reCAPTCHA.

### 4.1.3 NASTAVENÍ MĚSÍČNÍHO POPLATKU

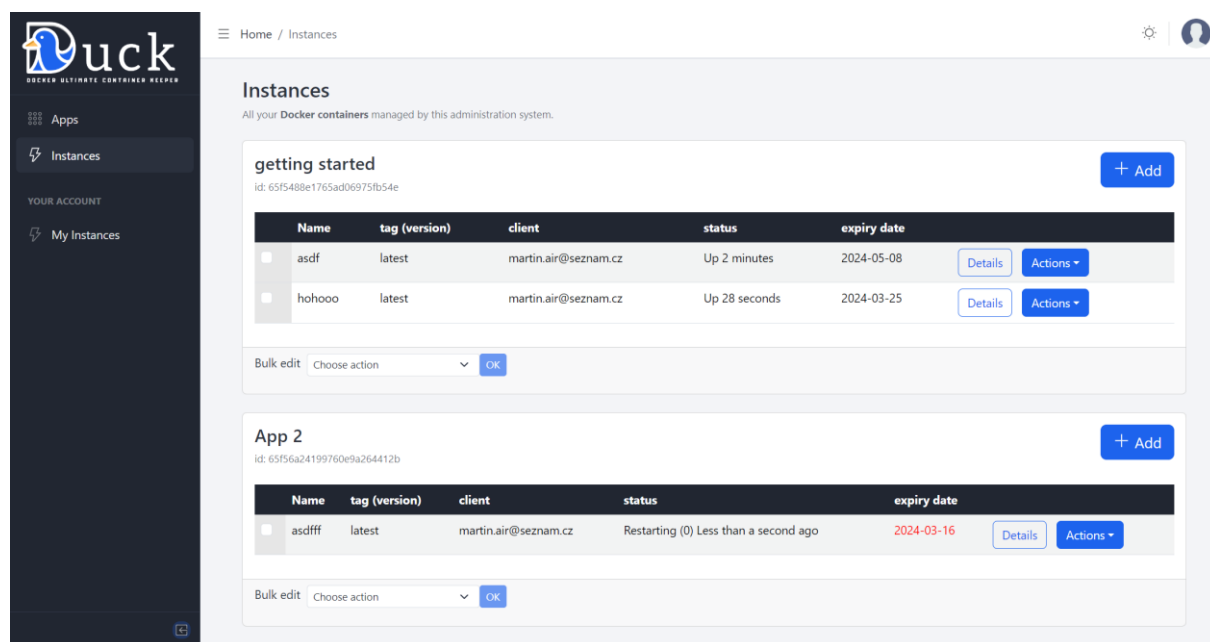
Jak je vidět na obr. 3, pod editorem scriptů a formulářem se nachází nastavení měsíčního poplatku a nastavení počtu dní, které má uživatel zdarma na vyzkoušení hostované aplikace. Po uplynutí nastaveného počtu dní od vytvoření instance se kontejner zastaví a uživatel je vyzván e-mailem k zaplacení poplatku. Jako poslední se zde nachází nastavení domény, na jejíž subdoménách budou instance automaticky spouštěny. Svoje kontejnery a jejich předplatné

může uživatel spravovat po přihlášení/registraci do systému DUCK stejným e-mailem jako byl použit při vytváření instance.

#### 4.1.4 DEFAULTNÍ DOCKER IMAGE

Levý sloupec nastavení obsahuje všechny Docker images příslušné k hostované aplikaci, tj. patřící ke stejné Docker repository. Z těchto images je vždy pouze jeden nastaven jako defaultní, který se používá pro nově vytvořené instance hostované aplikace.

### 4.2 SPRÁVA INSTANCÍ APLIKACÍ



Obrázek 5 Správa instancí aplikací

Tato obrazovka umožňuje správu všech instancí, které se v systému DUCK nachází. Instance jsou sdruženy podle aplikace a ve sloupci *status* vidíme, v jakém stavu se instance právě nachází. Instance můžeme vytvářet, pomocí funkce *bulk edit* pozastavovat, mazat, nebo upgradovat na nové verze hostované aplikace. Ve sloupci *tag(version)* vidíme na jaké verzi naší aplikace instance běží.

#### 4.2.1 VYTVOŘENÍ INSTANCE

K vytvoření instance aplikace slouží již výše zmíněný inicializační formulář, který administrátor v editoru připravil. Tento formulář je připraven tak, aby mohl být umístěn v podobě `IFrame`<sup>4</sup> do prezentačního webu příslušné hostované aplikace. Odkaz na formulář získá administrátor v editoru formuláře. Vytvořit instanci hostované aplikace může administrátor, ale hlavně také neregistrovaný uživatel, který si tím vytvoří stínový účet, pod

<sup>4</sup> `IFrame` je HTML element umožňující ve webové stránce vymezit plochu pro vložení jiné webové stránky

kterým jsou jeho instance registrovány. Následně po vyplnění formuláře je informován e-mailem o spuštění jeho aplikace a obdrží pokyny k přihlášení a ovládání hostované aplikace.

## 4.2.2 UPGRADE INSTANCE

Mějme modelový příklad, kdy hostujeme aplikaci pro několik desítek či stovek uživatelů a v průběhu času naši hostovanou aplikaci vyvíjíme. Je tedy zapotřebí každému uživateli jeho aplikaci upgradovat, aniž by přišel o svá data, a to možná co nejvíce automatizovaně. Bez použití systému DUCK by to znamenalo ručně, za pomoci příkazové řádky serveru, každý kontejner se starou verzí aplikace odstranit a spustit kontejner s verzí novou. Toto nám systém DUCK značně ulehčuje, jelikož stačí vybrat instance, které se mají upgradovat a ve funkci *bulk edit* vybrat novou verzí aplikace. Možné je provést i downgrade v případě, kdy se například v nové verzi vyskytne kritická chyba. Další možnost, jak funkci využívat může být například postupné nasazování aplikace, tzn. ne všem uživatelům najednou. Minimalizujeme tak množství nespokojených klientů ve chvíli, kdy se v nové verzi vyskytne chyba.

## 4.3 DETAIL A NASTAVENÍ SAMOSTATNÉ INSTANCE

Po kliknutí na tlačítko *Details* v přehledu instancí se zobrazí stránka příslušná k vybrané instanci. Obsahuje aktuální informace o instanci aplikace načtené z Docker engine, ale také zobrazí podrobnosti o verzi aplikace, tj. verzi Docker image. V neposlední řadě také umožňuje nastavit *resource limits*, tedy limity využívání přidělených hardwarových prostředků.

### 4.3.1 RESOURCE LIMITS

Právě tyto limity jsou klíčové pro stabilní a spolehlivé využívání systému. Kdyby totiž měla každá instance (kontejner) nastaven neomezený přístup k hardwarovým zdrojům serveru, mohla by jedna přetížená aplikace zapříčinit zpomalení, nedostupnost či v nejhorším případě, kdy dojde k přecerpání paměti RAM i k ukončení instancí aplikací jiných uživatelů. Jejich defaultní hodnoty se načítají z konfiguračního souboru a jsou stejné pro všechny instance. Administrátor je ale má možnost ručně nastavit pro každou instanci zvlášť. Limity jsou nastaveny za pomoci Docker engine, který také vynucuje jejich dodržování. Docker umožňuje natavit různé druhy limitů, které mohou být dvojího typu. *Soft limits* jsou omezení, které umožňují kontejneru využívat libovolné množství prostředků až do chvíle, kdy dojde na serveru k nedostatku hardwarových prostředků. V tu chvíli se začnou limity uplatňovat. *Hard limits* jsou naopak limity, které nedovolují kontejneru za žádných okolností překročit stanovenou kvótu.

Resources limits

deactivate by -1

CPU shares	70	(priority), default 1024 --cpu-shares
RAM	50	MB --memory-reservation
SWAP	0	MB --memory-swap
Disk	0	(priority) default 500--blkio-weight


Obrázek 6 Resource limits

- RAM – Z několika možných variant limitu jsem vybral soft limit *memory reservation*, který omezí možnosti využití paměti ve chvíli přetížení systému (*soft limit*).
- Total memory (RAM + SWAP) – Představuje vlastní kombinaci nastavení, která omezí maximální využití paměti (*hard limit*). Pokud to hardwarové prostředky serveru dovolí, je možné překročit RAM limit zmíněný výše, ale překročit tento limit není kontejneru umožněno ani za podmínek, kdy jsou dostupné volné hardwarové zdroje.
- Disk – Pro omezení práce s diskem jsem zvolil nastavení *--blkio-weight*, které umožňuje nastavit poměr ve kterém bude přerozdělována kapacita pro zápis na disk mez běžící kontejnery. Výchozí hodnota je 500 je ji možné nastavit v rozmezí 10 až 1000.

### 4.3.2 EXPIRACE PŘEDPLATNÉHO

Každá aplikace má nastavené datum expirace, toto datum může administrátor u každé instance editovat ručně. Primárně si však uživatel předplatí několik měsíců pomocí platební brány ze svého uživatelského profilu v systému DUCK.

Dates

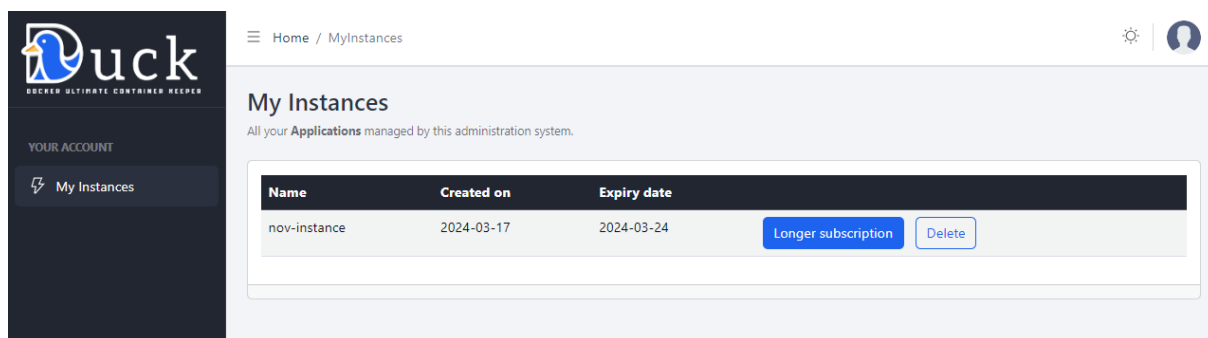
Created	03/16/2024
Expiry date	05/08/2024 

Obrázek 7 Nastavení data expirace v administraci

- CPU shares – Pokud nastane situace, kdy na severu je CPU maximálně využito, aplikuje se tato priorita ve využívání CPU. Výchozí a neurální hodnota je 1024. Pokud nastavíme hodnotu vyšší, kontejner dostane přiděleno v poměru k ostatním kontejnerům více procesorového času a pokud je naopak hodnota nižší, dostane přiděleno méně (*soft limit*). [4]

## 4.4 PŘIHLÁŠENÍ UŽIVATELE

Pokud se uživatel registruje/přihlásí do systému DUCK stejným e-mailem jako využil během vytváření své instance při vyplňování formuláře, dostane přístup ke správě všech svých instancí. Vidí jejich datum expirace a může ho prodloužit předplacením několika dalších měsíců za pomoci platební brány Stripe. Instance také může libovolně mazat. Viz. také kapitola 5.8.Stripe, kde je k dispozici testovací číslo kreditní karty pro vyzkoušení nákupu předplatného.



Obrázek 8 Uživatelský přehled instancí

## 5. POUŽITÉ TECHNOLOGIE

Vzhledem k tomu, že nejvhodnějším formátem pro mou práci je webová aplikace, volba technologií vycházela z tohoto předpokladu. Následující kapitoly se budou věnovat podrobnému popisu použitých technologií a mým osobním zkušenostem s nimi.

### 5.1 VUE.JS

Při volbě frontend frameworku, který by mi usnadnil vývoj uživatelského rozhraní, jsem se rozhodoval mezi Vue.js a React. Oba frameworky poskytují funkce reaktivity, což znamená, že automaticky sledují změny stavu JS<sup>5</sup> a při změnách reaktivně aktualizují DOM<sup>6</sup>. Vue.js je navržen tak, aby byl flexibilní, rychlý a přizpůsobitelný. Je postaven na standardním HTML<sup>7</sup>, CSS<sup>8</sup> a JS a poskytuje deklarativní programovací model založený na komponentách, které pomáhají efektivně vyvíjet uživatelská rozhraní. V porovnání s frameworkem React je Vue.js menší co se týče velikosti výsledné aplikace a disponuje vyšším výkonem, proto jsem si ho také nakonec vybral.

---

<sup>5</sup> JS – JavaScript

<sup>6</sup> DOM – Document Object Model

<sup>7</sup> HTML – HyperText Markup Language

<sup>8</sup> CSS – Cascading Style Sheets

Komponenty, označené příponou `.vue`, mohou být vytvořeny ve dvou různých stylech API<sup>9</sup>: Options API a Composition API. V mém projektu používám novější Composition API (které řeší určité nedostatky o něco staršího Options API), kde definujeme logiku komponentu pomocí importovaných funkcí API. V SFC<sup>10</sup> se Composition API obvykle používá s `<script setup>`, nebo ekvivalentním `<script>` s metodou `setup()`.

Další hlavní výhoda Vue.js je již zmíněná reaktivita, což je schopnost frameworku reagovat na změny vstupních dat, nebo stavu bez nutnosti explicitního programování reakce. To umožňuje dynamičtější a interaktivnější uživatelské rozhraní a usnadňuje vývoj aplikace. Vue.js pracuje s tzv. deklarativním přístupem. To znamená, že vývojář definuje požadovaný stav uživatelského rozhraní a systém se automaticky stará o jeho aktualizaci v reakci na změny. Toho je dosaženo pomocí datových vazeb, které propojují komponenty uživatelského rozhraní s podkladovými daty.

K přechodům mezi jednotlivými obrazovkami aplikace (tzv. views) používám Vue Router. Ten hlídá veškeré změny v URL a zajišťuje přechody mezi obrazovkami bez opětovného načítání celé stránky včetně předávání parametrů URL. Pokud se uživatel pokusí zadat URL, která není v aplikaci platná, je automaticky přesměrován na stránku s chybou 404 - Not found.

Data o přihlášeném uživateli jsou uložena sdíleně pro všechny komponenty ve Vuex Store. Tato data lze měnit pouze přes speciální metody zvané mutace. Také se zde nachází sdílené funkce, tzv. akce, do těch jsem pro větší přehlednost umístil všechny requesty na backendové API.

## 5.2 COREUI

CoreUI je v základní verzi open-source framework pro tvorbu moderního a responzivního webového rozhraní. Nabízí širokou škálu komponent, nástrojů a UI prvků, které usnadňují a urychlují vývoj webových aplikací. CoreUI je postaven na technologiích Bootstrap 5 a Sass. Je kompatibilní s majoritními frameworky jako Angular, React a Vue.js.

Obsahuje rozsáhlou kolekci předpřipravených komponent, jako jsou tlačítka, formuláře, karty, modální okna a grafy. Je navržen tak, aby byl intuitivní a snadno se používal.

Pro založení projektu a vygenerování všech nezbytných komponent jsem použil CoreUI template. Osobně se mi s tímto frameworkem pracovalo velmi dobře, jelikož jsem byl nucen napsal jen malé množství CSS.

---

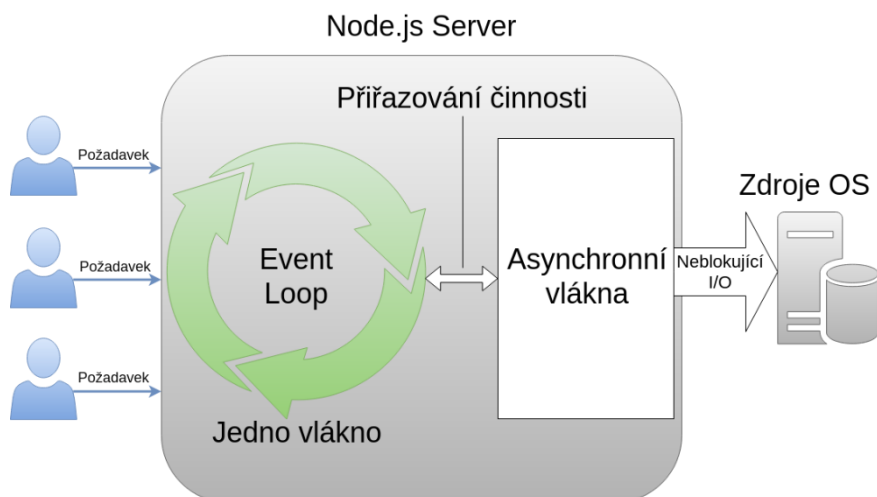
<sup>9</sup> API – Application Programming Interface

<sup>10</sup> SFC – Single-File Component

## 5.3 NODE.JS

Jako technologii pro vytvoření backend API pro mojí aplikaci jsem zvolil Node.js. Je to runtime umožňující spouštět JavaScript kód mimo webový prohlížeč. Je postaveno na Chrome V8 JavaScript engine, proto základ tohoto JS prostředí je stejný jako ve webovém prohlížeči Google Chrome. Důležité je zmínit, že byť v tomto prostředí je možné psát téměř cokoliv, jeho návrh a primární účel je tvorba serverové části webových aplikací (backend). Toto má společné např. s jazykem PHP nebo Python, který má stejné zaměření. Na rozdíl od zmíněného PHP je v Node.js kladen důraz na vysokou škálovatelnost a schopnost obsloužit mnoho připojených klientů naráz. Pro tuto vlastnost a vysokou výkonnost je dnes Node.js velmi oblíbený.

Jádro celého Node.js tvoří tzv. smyčka událostí (event loop). Do ní vstupují všechny uživatelské požadavky jako události, které jsou poté přiděleny jednotlivým nezávislým vláknům. Další operace, jako manipulace se zdroji operačního systému, což jsou např. čtení souboru nebo práce s databází, jsou poté řešeny také pomocí událostí (tzv. neblokující I/O<sup>11</sup>). Jejich volání je opět zařazeno do smyčky událostí. Ve výsledku to znamená, že vše řídí tato smyčka pomocí různých událostí. Je to jednoduché, avšak velice efektivní řešení, obzvláště pro webové aplikace s velkým množstvím uživatelských požadavků. JavaScript pak do toho všeho velice hezky zapadá, jelikož tyto události jsou nativně reprezentovány pomocí JS callbacků.



Obrázek 9 Smyčka událostí (Event Loop)

V mém projektu používám Express, což je oblíbený balíček pro Node.js, který zjednodušuje práci s http protokolem a je přizpůsobený k tvorbě REST API. Poskytuje jednoduché prostředí s možnostmi využití middlewaru (např. parsování JSON) a error handlingu.

Node.js používám pro tvorbu API backendu svého ročníkového projektu již potřetí a mám s tím velmi dobré zkušenosti.

---

<sup>11</sup> I/O – Input/Output

## 5.4 MONGODB

Pro ukládání dat jsem zvolil dokumentovou databázi MongoDB, která patří mezi NoSQL databáze. Je objektově orientovaná, jednoduchá, dynamická a dobře škálovatelná. Místo tabulek (jako v relační databázi) používá kolekce, místo řádků a sloupců dokumenty a pole. Data se ukládají ve formátu BSON (Binary JSON), což je binárně kódovaná podoba textové notace JSON (JavaScript Object Notation). [5]

V našem projektu používáme Mongoose, což je node.js balíček pro MongoDB a Node.js. Umožňuje nám spravovat vztahy mezi daty, poskytuje ověřování (validaci) dat, která zabrání uložení nevalidních dat a následným chybám. Pro tento účel se používají schémata, ve kterých se definují datové typy (Number, String, Boolean, Object apod.). Spolu s tím také balíček zabezpečuje komunikaci s databází (databázovým serverem) a transfer dat mezi serverem webové aplikace a databází.

*Schema* je vzor datové struktury dokumentu, která je ověřována prostřednictvím balíčku Mongoose při ukládání dat. MongoDB jako samotný databázový server nepracuje se *schématy*, tedy nemá schopnost validovat data. V mé práci používám čtyři datová schémata pro uložení dokumentů do čtyř kolekcí:

- Users – uživatelé
- Roles – uživatelské role (administrátor, uživatel)
- Apps – hostované aplikace
- Instances – instance hostovaných aplikací

## 5.5 JWT

JWT (JSON Web Token) je standardizovaný formát pro ověřování a autorizaci uživatele ve formě tokenu. Tyto tokeny jsou kompaktní, kryptograficky podepsané a kompatibilní s JSON, čímž se stávají ideální volbou pro přihlašování a autorizaci ve webových aplikacích a API. [6]

### 5.5.1 PRINCIP FUNGOVÁNÍ PŘIHLAŠOVÁNÍ NA BÁZI JWT:

1. Klient odešle serveru požadavek s uživatelskými jmény a hesly.
2. Server ověří platnost zadaných údajů.
3. V případě úspěšné autentizace server vygeneruje token JWT, který obsahuje informace o uživateli (ID, role, oprávnění) a podepíše ho tajným klíčem.
4. Token JWT je odeslán zpět klientovi a uložen v jeho úložišti (v mém případě localStorage, ale možno i cookie).
5. Při každém dalším požadavku na server klient odešle token JWT v hlavičce HTTP požadavku.



6. Server ověří platnost tokenu JWT pomocí tajného klíče a na základě informací v tokenu udělí nebo zamítne přístup k požadovaným datům/funkcionálnostem. [6]

## 5.6 CADDY

Caddy je webový server s otevřeným zdrojovým kódem, který se vyznačuje jednoduchostí, rychlostí a flexibilitou. Oproti běžným serverům, jako je Apache nebo Nginx, se Caddy zaměřuje na přehlednost konfiguračních souborů a snadné použití. Jeho bezkonkurenční výhodou oproti jiným používaným webserverům je možnost konfigurace v reálném čase za pomoci REST API. Právě díky tomu jsem tento webserver zvolil. Další velkou výhodou je vestavěná funkce získávání a obnovování TLS<sup>12</sup> certifikátů pro poskytování zabezpečeného spojení HTTPS<sup>13</sup>. Tyto Let's Encrypt<sup>14</sup> certifikáty jsou získávány plně automaticky bez nutnosti jakéhokoli nastavení. [7]

### 5.6.1 VYUŽITÍ CADDY REST API

Caddy server je na serveru používán jako reverse TSL proxy a systém DUCK využívá jeho REST API ke konfiguraci ve chvíli, kdy vytváří nebo maže instanci hostované aplikace. [8] Hostované aplikaci v podobě kontejneru je přiřazen jeden port na serveru. Interval portů přidělovaných kontejnerům je nastaven v konfiguračním souboru systému DUCK a slouží pouze k lokálnímu přesměrování requestů z Caddy serveru. Tyto porty jsou tedy zablokované ve firewallu a všechny instance jsou automaticky zpřístupněny s TSL na konfigurovatelné doméně nastavené v editaci hostované aplikace.

## 5.7 DOCKER

Docker je open-source platforma pro virtualizaci, která umožňuje uzavřít aplikace a jejich závislosti do tzv. kontejnerů. Tyto kontejnery jsou přenosné mezi operačními systémy Linux na kterých je nainstalován *Docker engine*, čímž se eliminují problémy s kompatibilitou a zjednodušuje se vývojový proces.

*Docker Image* je statická šablona pro vytváření kontejnerů. Skládá se z vrstvy základního obrazu (např. Ubuntu, Debian, Alpine či dalších Linux systémů) a dalších vrstev, které obsahují nainstalovaný software a konfiguraci. Obrazy se ukládají do registrů, jako je Docker Hub, a lze je snadno sdílet a znovu používat.

Z *Docker Image* se vytváří kontejnery. Pomocí Docker Engine se image spustí jako izolovaný proces. Kontejnery sdílí základní operační systém, čímž se šetří systémové prostředky oproti

---

<sup>12</sup> TSL – nástupce protokolu SSL

<sup>13</sup> HTTPS – Hypertext Transfer Protocol Secure

<sup>14</sup> Let's Encrypt – certifikační autorita poskytující zdarma TLS certifikáty

jiným způsobům virtualizace např. virtuálním počítačům. Kontejnery obsahují vše potřebné pro spuštění aplikace, včetně softwaru, knihoven a konfigurace.

## 5.8 STRIPE

Pro přijímání plateb za předplatné hostovaných aplikací jsem se rozhodl implementovat platební bránu Stripe a to hlavně díky jejím přívětivým poplatkům a snadnému použití. V porovnání s ostatními platebními branami existuje téměř pro každou platformu<sup>15</sup> SDK<sup>16</sup> s podrobnou dokumentací a návody, což mi velmi usnadnilo vývoj. Pro použití v této maturitní práci jsem zvolil pouze verzi API pro testování, jelikož neplánuji v nejbližší budoucnosti přijímat opravdové platby a nechci tedy proto provádět ověření osobních údajů. Platební bránu je ale možné použít a v systému DUCK tak zaplatit za pomoci testovacího čísla platební karty:

4242 4242 4242 4242  
12/33 (*jakékoli budoucí datum*)  
123 (*libovolné tři číslíčky*)

## 6. PREZENTACE PRÁCE

Systém DUCK je možné vyzkoušet na doméně *duck.ekdyson.site*. Tímto bych také rád poděkoval Gymnáziu, Praha 6, Arabská 14 za poskytnutí serveru pro provoz tohoto projektu. Bylo tím mimo jiné umožněno zpřístupnit veřejnosti k vyzkoušení systém DEH provozovaný jako hostovaná aplikace v systému DUCK.

---

<sup>15</sup> V současné době je SDK dostupné pro jazyky: Ruby, Python, Go, Java, Node.js, PHP a .NET

<sup>16</sup> SDK – software development kit

## 7. ZÁVĚR

Na začátku jsem si vytyčil cíl naprogramovat aplikaci, která bude schopna automaticky spustit instanci libovolné, do systému předem nakonfigurované webové aplikace. Jedním z cílů také byl zveřejnit tak systém DEH veřejnosti k vyzkoušení před letošním krajským kolem soutěže Středoškolská odborná činnost. Při práci jsem využil množství pro mě nových technologií a služeb, kterými byli např. webový server Caddy nakonfigurovaný pomocí REST API, nebo platební brána Stripe. Splnil jsem všechny cíle, které jsem si na projektu vytyčil a výsledkem mé práce je reálně používaná aplikace.

## 8. POUŽITÉ ZDROJE

- [1] Gymnázium, Praha 6, Arabská 14, „ŠVP - Aplikační software,“ [Online]. Available: [https://www.gyarab.cz/media/svp/ICT\\_Aplika%C4%8Dn%C3%AD\\_softwar\\_e\\_2hu\\_p%C5%99.pdf](https://www.gyarab.cz/media/svp/ICT_Aplika%C4%8Dn%C3%AD_softwar_e_2hu_p%C5%99.pdf).
- [2] Gymnázium, Praha 6, Arabská 14, „PRÁCE VE WORDPRESSU,“ 2022 (date from web.archive.org). [Online]. Available: <https://www.gyarab.cz/p/prace-ve-wordpressu>.
- [3] Docker Inc., „Docker Hub,“ WordPress, [Online]. Available: [https://hub.docker.com/\\_/wordpress](https://hub.docker.com/_/wordpress).
- [4] Docker Inc., „docs.docker.com,“ 2024. [Online]. Available: [https://docs.docker.com/config/containers/resource\\_constraints/#configure-the-default-cfs-scheduler](https://docs.docker.com/config/containers/resource_constraints/#configure-the-default-cfs-scheduler).
- [5] MongoDB, „mongodb.com,“ 2024. [Online]. Available: <https://www.mongodb.com/basics/bson>.
- [6] Bezkoder, „BezKoder,“ 28 6 2023. [Online]. Available: <https://www.bezkoder.com/node-js-express-login-mongodb/>. [Přístup získán 2024].
- [7] ZeroSSL, „Caddy server,“ 2024. [Online]. Available: <https://caddyserver.com/>.
- [8] Caddy Documentation, „Caddy server Docs,“ 2024. [Online]. Available: <https://caddyserver.com/docs/api>.

## 9. SEZNAM OBRÁZKŮ

Obrázek 1 Logo systému DUCK .....	5
Obrázek 2 Přidání aplikace do systému .....	7
Obrázek 3 Nastavení hostované aplikace .....	8
Obrázek 4 Editor inicializačního formuláře.....	9
Obrázek 5 Správa instancí aplikací.....	10
Obrázek 6 Resource limits .....	12
Obrázek 7 Nastavení data expirace v administraci .....	12
Obrázek 8 Uživatelský přehled instancí .....	13
Obrázek 9 Smyčka událostí (Event Loop).....	15