Gymnasium, Prague 6, Arabská 14

Field of programming

Graduation thesis

DUCK

(Docker Ultimate Container Keeper)

Administration system for Docker containers

Martin Voplakal, 4.E

April 2024

I declare that I am the sole author of this project, all citations are properly acknowledged, and all literature and other sources used are listed in the thesis. In accordance with the Act 121/2000 Coll. (the so-called Copyright Act) as amended, I hereby grant the school Gymnázium, Praha 6, Arabská14 the right to exercise the right to reproduce the work (§ 13) and the right to communicate the work to the public (§ 18) for an unlimited period of time and without limitation of territorial scope.

In Prague on: Martin Voplakal

ANNOTATION AND PROJECT ASSIGNMENT

This final thesis deals with the design and implementation of an administration system for a web application provider that allows its customers to run their own application instances in the form of Docker containers. The system provides a form-based web interface where the customer fills in the necessary information to run the requested service, which can be, for example, WordPress or another web application. After submitting the form, the customer's service instance is started automatically and without administrator intervention. Each application may be subject to a certain monthly service fee and each user has the option to pay this subscription fee using a payment gateway.

The processes are designed so that the administrator can customize them as much as possible using application-specific initialization scripts. The system allows you to limit the hardware resources of each container (CPU, RAM, SWAP, disk) to certain values to avoid overloading the server and limiting other customers.

TABLE OF CONTENTS

1. INTRODUCTION

This project was created in order to fulfill my practical need to use an administration system that will allow the general public to test and use the DEH system (Digital Education Hub, Digital Textbook System) with which we placed 5th in the national round of the Secondary School Vocational Activity Competition last year. Since the DEH system is developed as a web application and runs in Docker containers, for each instance of this application it is necessary to create and connect a database, configure a domain, SSL certificate, etc. Running multiple instances of the DEH system is therefore a routine but rather extensive job in the server terminal.

The administration system for Docker containers allows you to host any application in the Docker environment and is designed so that each user can run their own instance of the hosted application fully automatically and without the need for administrator intervention. The system is primarily designed for hosting web applications or similar systems such as WordPress, where each user needs their own instance of the application. In practice, however, this system can find completely different uses. See Chapter *3. Uses*.

2. DEFINITIONS OF NAMES AND TERMS

This project was called *DUCK - Docker Ultimate Container Kepper* (hereafter only this abbreviation is used in the text) and I created a logo for completeness and to complete the graphical design of the user interface.

Figure 1 DUCK logo

The DUCK system is designed to host other applications. I will refer to these applications hereafter as *hosted applications*.

An application instance refers to a specific running version of the application.

By the term *user* we also mean a user of the DUCK system who uses it to run a hosted application such as DEH, WordPress and others.

3. USES

The DUCK system is primarily intended for running web applications, but it can be used wherever there is a need to manage a large number of instances of any application. Such an application must be designed to run on a server and each client may require a different version of the application, for example for compatibility with the client's system. It is also great for presenting an application in order to give people the opportunity to try it for free and almost immediately.

3.1 SPECIFIC EXAMPLES

DEH (Digital Textbook System) has already been mentioned above and uses all the features of the DUCK system. It is a model example of use.

WordPress is another possible hosted application. Apart from its professional use for web development and content management, it is also widely used in school classes, where students learn the basics of web development in administration systems. [1] [2] The system can be tested for free, e.g., at the hosting provider WordPress.com, but the features available for free are insufficient for detailed instruction. For example, it is not possible to install plugins, without which it is impossible to create almost any website that is usable in practice. In order for the school not to be dependent on external hosting and to be able to teach students how to create their own e-shop, for example, the use of the DUCK system is directly offered for this purpose. WordPress is already officially released in the form of Docker Image. [3] So all you have to do is run the SQL database and create the appropriate scripts[1] to the DUCK system to connect WordPress to the database.

It is also possible to use this system when teaching the subject of operating systems, when students need access to the root account of the UNIX operating system to learn how to work with files, permissions or filesystems. Since Docker containers can be viewed as standalone operating systems with their own filesystem and permissions, if configured correctly, it is possible to give a user root ssh access to a Docker container without being able to control another container or the entire server from the container. The idea here is that each student would fill in the form with their required root password and their ssh-server container would be started and managed by DUCK. It is possible to use an already prepared Docker image[2] containing the *ssh-server*, or prepare such a Docker image according to your own requirements. At the end of the class, the teacher would then easily remove these containers using DUCK. Containers can also be restricted from accessing the Internet by a proxy server for security reasons, but I won't go into that here as it is not the focus of my paper.

[1] see chapter *4.1.1 Bash scripts*
[2] https://hub.docker.com/r/corbinu/ssh-server

4. ADMINISTRATION INTERFACE FUNCTIONS

In the following chapters, I discuss the individual functions and settings of the DUCK system. If an administrator decides to use the DUCK system to manage his or her hosted application, it is necessary to register the application with the system and set up its configuration scripts.

4.1 ADDING AN APPLICATION TO THE SYSTEM

In order to add a hosted application to the system, it must meet the requirement that it is already on the server in the form of a tested and functional Docker image. DUCK communicates with the Docker system using a terminal and when creating a hosted application, it will give the administrator a choice of all Docker images that exist on the server.

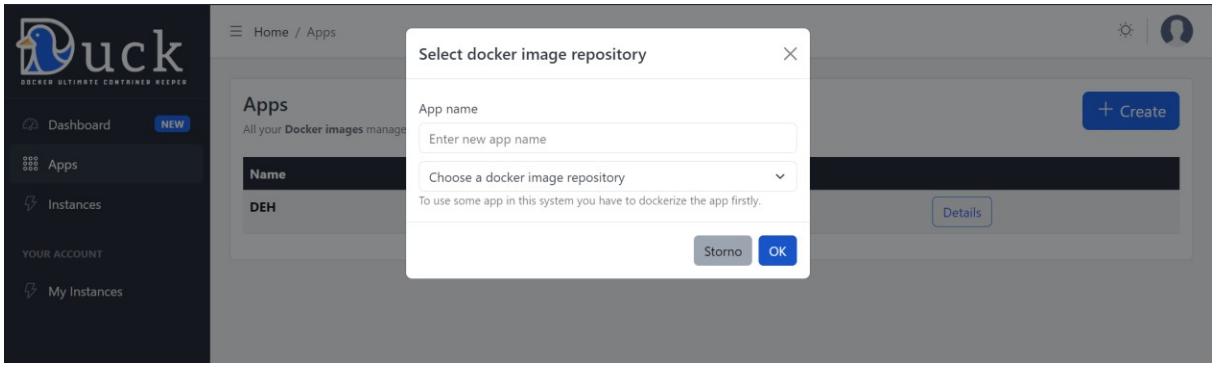

Figure 2 Adding an application to the system

4.1.1 BASH SCRIPTS

For each application in the DUCK system it is necessary to create several Bash[3] scripts. These scripts are specific to each web application, as each application may have different parameters. The scripts thus represent the interconnection component between the hosted application and the DUCK system.

In order for the scripts to preset the web application environment according to the user's requirements, the variables are populated with data from the user-filled initialization form described in the following chapter. However, this data must be pre-protected against execution to prevent Cross Site Scripting (XSS) attac, where an attacker fills out the form and the data in the form is then interpreted and executed as malicious code. This would have devastating consequences for the entire infrastructure in our case where it is a Bash script.

This section describes all the scripts that are used in the DUCK system and the administrator sets them up for each hosted application.

[3] Bash - Bourne again shell

- **Init script** - An initialization script that runs only once when a new instance of the hosted application is created. Its purpose is to configure the new application instance according to the user's requirements, create a database, prepare a configuration file or populate the database with placeholder data
- **Run script** - The run script ensures that the Docker container is started. The last command in this script must be *docker run, which* starts the container and returns its ID for subsequent saving to the database.
- **Delete script** - This script runs only once when the administrator or user decides to delete the application including all stored data. Its main purpose is to ensure that the database is deleted, as the DUCK system will take care of the application's file folder.

Scripts are always checked for syntax errors before saving using *bash -n,* where *-n* is a switch that allows reading commands from the input but prevents their execution. This syntax check will detect most common syntax errors, except for non-valid external command names.

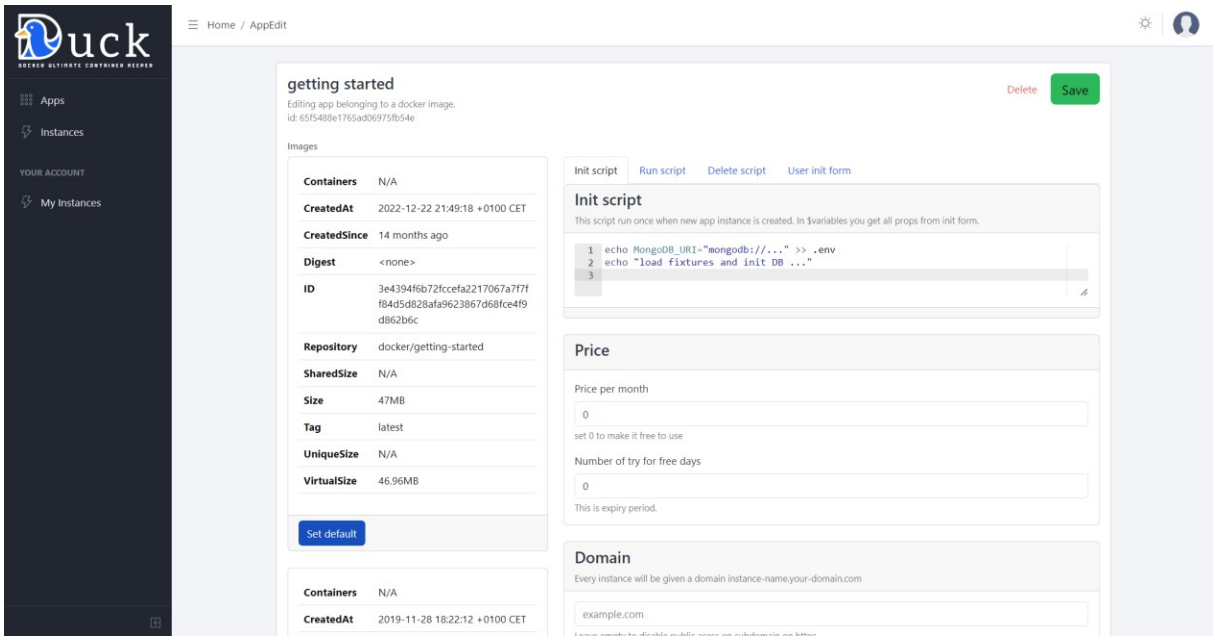

Figure 3 Hosted application settings

4.1.2 INITIATION FORM

In the right column of the hosted application settings, there are script editors and the editor *User init form*, an initialization form editor that the user fills in to start your own instance of the hosted application. This editor is displayed in the last tab next to the script editor in the form of a preview. The editor can then be opened in a maximized window, see Figure 4.

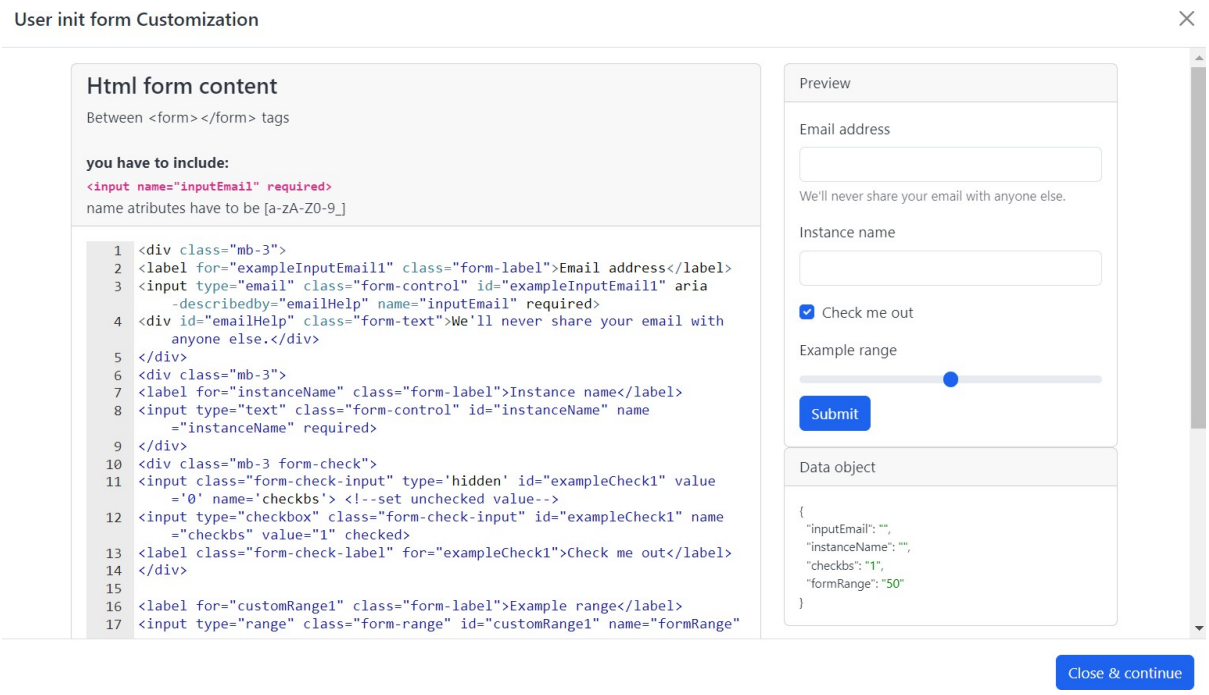

Figure 4 Initialization Form Editor

This editor was created to contain a live preview in the right column (a preview that is rendered in real time according to the edited code), then below it there is a *Data object, where* the administrator can check if the input elements of the form are configured correctly, so that the data from the form can be loaded properly. This user-filled data is inserted as variables into all scripts when the hosted application instance is initialized, where the application can be preconfigured or otherwise modified based on it. Finally, the filled in *Instance name* is also used as the name of the subdomain where the application instance is available.

It was also necessary to treat this form against DDOS attack, because launching application instances is a relatively computationally intensive process and automatic generation of requests from the form is very easy. So I have protected the form against robotic completion using reCAPTC HA.

4.1.3 SETTING THE MONTHLY FEE

As you can see in Figure 3, below the script editor and the form, there is a setting for the monthly fee and a setting for the number of days the user has to try the hosted application for free. After the set number of days has elapsed since the instance was created, the container is stopped and the user is prompted by email to pay the fee. Lastly, there is the setting of the domain on whose subdomains the instances will automatically run. Your containers and their subscriptions

can be managed by the user after login/registration to the DUCK system using the same email used when creating the instance.

4.1.4 DEFAULT DOCKER IMAGE

The left column of settings contains all Docker images relevant to the hosted application, i.e. belonging to the same Docker repository. Of these images, only one is always set as the default, which is used for newly created instances of the hosted application.

4.2 MANAGING APPLICATION INSTANCES

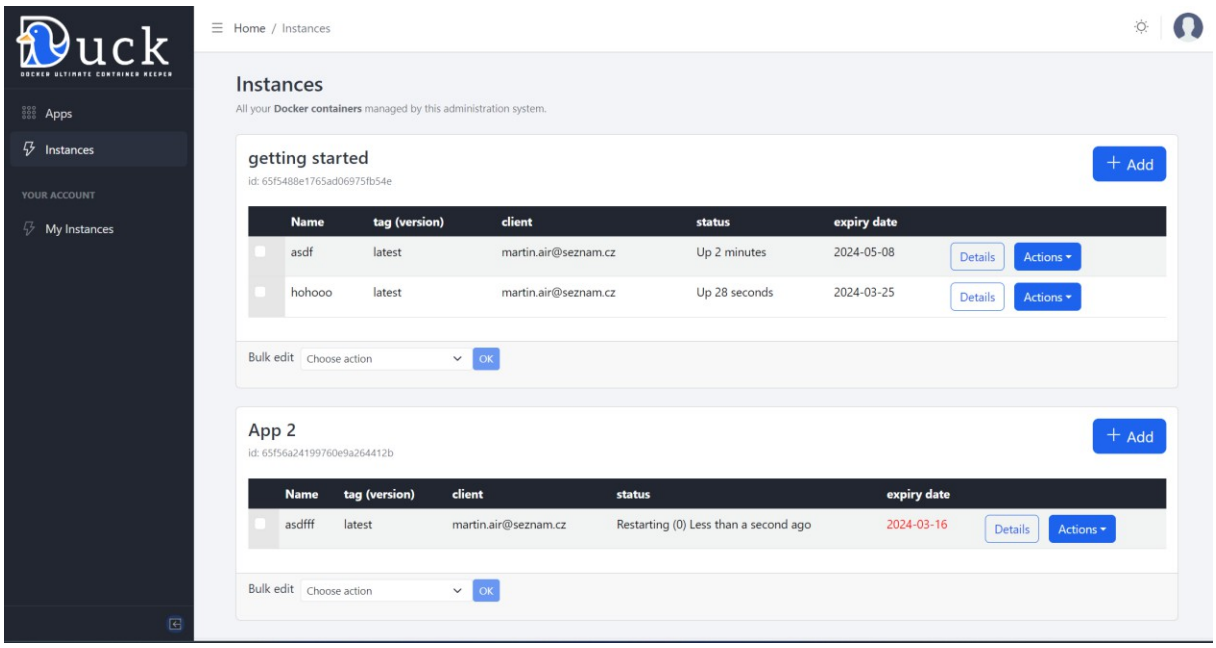

Figure 5 Managing Application Instances

This screen allows you to manage all the instances that are in the DUCK system. Instances are grouped by application and in the *status* column we can see what state the instance is currently in. Instances can be created, suspended, deleted, or upgraded to new versions of the hosted application using the *bulk edit* feature. In the *tag(version)* column, we can see what version of our application the instance is running on.

4.2.1 CREATING AN INSTANCE

To create an instance of the application, use the aforementioned initialization form that the administrator has prepared in the editor. This form is prepared so that it can be placed in an IFrame[4] into the presentation site of the respective hosted application. The link to the form is obtained by the administrator in the form editor. The administrator can create an instance of the hosted application, but also a non-registered user can create a shadow account, under

[4] An IFrame is an HTML element that allows a web page to define an area for inserting another web page

by which its instances are registered. Then, after filling in the form, he is informed by e-mail about the launch of his application and receives instructions for logging in and controlling the hosted application.

4.2.2 UPGRADE INSTANCE

Let's have a model example where we host an application for several tens or hundreds of users and we develop our hosted application over time. So we need to upgrade each user's application without losing their data, and as automated as possible. Without using DUCK, this would mean manually, using the server command line, removing each container with the old version of the application and starting a container with the new version. This is made much easier by DUCK, as we just need to select the instances to be upgraded and select the new application version in the *bulk edit* function. It is also possible to downgrade if, for example, a critical bug occurs in the new version. Another way to use the feature can be, for example, to deploy the application gradually, i.e. not to all users at once. This minimizes the number of dissatisfied clients when a bug occurs in a new version.

4.3 DETAIL AND SETTINGS OF A SEPARATE INSTANCE

Click *Details* in the instance overview to display the page for the selected instance. It contains the current information about the application instance loaded from the Docker engine, but also displays details about the application version, i.e. the version of the Docker image. Finally, it also allows you to set *resource limits,* i.e. limits on the use of allocated hardware resources.

4.3.1 RESOURCE LIMITS

These limits are crucial for stable and reliable use of the system. If every instance (container) had unlimited access to the server's hardware resources, one overloaded application could cause slowdown, unavailability or, in the worst case, even termination of other users' application instances when RAM is overloaded. Their default values are read from the configuration file and are the same for all instances. However, the administrator has the option to manually set them for each instance separately. The limits are set using the Docker engine, which also enforces them. Docker allows you to set different kinds of limits, which can be of two types. *Soft limits* are limits that allow the container to use any amount of resources until the server runs out of hardware resources. At that point, the limits begin to apply. *Hard limits*, on the other hand, are limits that do not allow a container to exceed a specified quota under any circumstances.

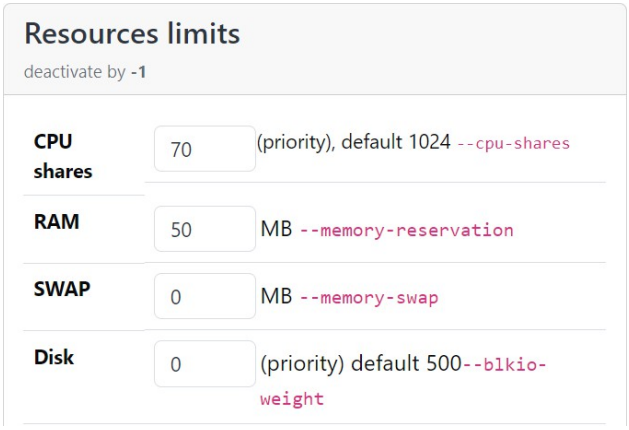

Figure 6 Resource limits

- CPU shares - If a situation occurs where the CPU is maxed out in the north, this priority in CPU usage is applied. The default and neural value is 1024. If we set the value higher, the container gets allocated more CPU time relative to other containers, and if the value is lower, it gets allocated less *(soft limit)*. [4]
- RAM - I chose soft limit *memory reservation* from several possible limit options, which limits the memory usage when the system is overloaded *(soft limit)*.
- Total memory (RAM + SWAP) - Represents a custom combination of settings that will limit the maximum memory usage *(hard limit)*. If the server's hardware resources allow, it is possible to exceed the RAM limit mentioned above, but the container is not allowed to exceed this limit even under conditions where free hardware resources are available.
- Disk - To limit disk work, I chose the *--blkio-weight* setting, which allows you to set the ratio in which disk write capacity will be redistributed between running containers. The default value is 500 and can be set between 10 and 1000.

4.3.2 SUBSCRIPTION EXPIRATION

Each application has a set expiration date, this date can be manually edited by the administrator for each instance. Primarily, however, the user will subscribe for several months using the payment gateway from their user profile in the DUCK system.

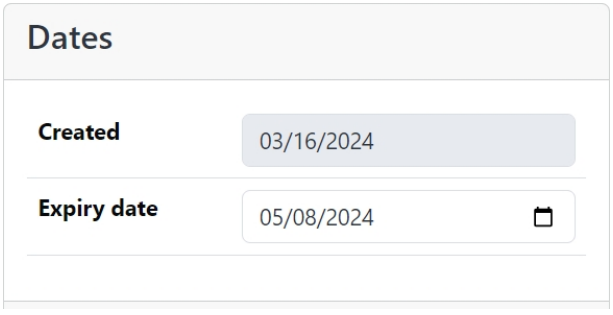

Figure 7 Setting the expiry date in the administration

4.4 USER LOGIN

If a user registers/logs into the DUCK system using the same email they used when creating their instance when filling out the form, they will be given access to manage all their instances. They can see their expiration date and can extend it by subscribing for a few more months using the Stripe payment gateway. He can also delete instances at will. See also Section *5.8.Stripe*, where a test credit card number is available to try out the subscription purchase.

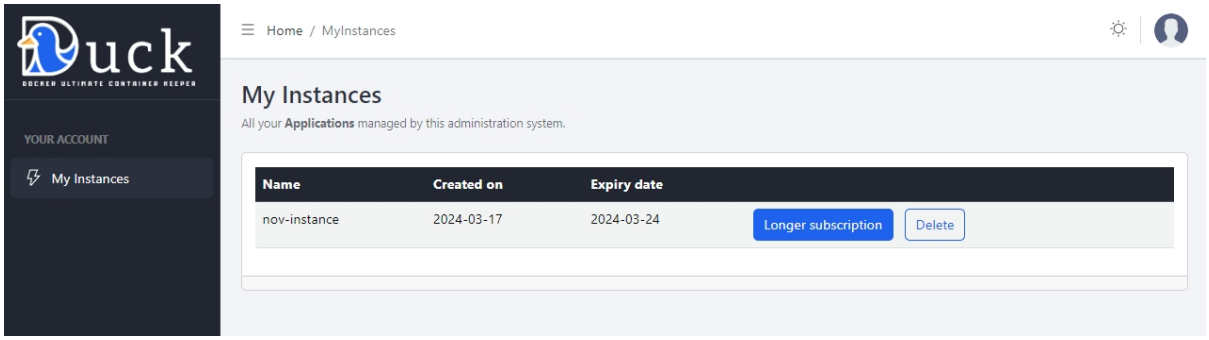

Figure 8 User Instance Overview

5. TECHNOLOGIES USED

Since the most suitable format for my work is a web application, the choice of technologies was based on this assumption. The following chapters will describe in detail the technologies used and my personal experience with them.

5.1 VUE.JS

When choosing a frontend framework that would make UI development easier, I decided between Vue.js and React. Both frameworks provide reactivity features, which means they automatically track JS state changes[5] and reactively update the DOM as changes occur.[6]. Vue.js is designed to be flexible, fast and customizable. It is built on standard HTML[7], CSS[8] and JS and provides a declarative component-based programming model to help develop user interfaces efficiently. Compared to the React framework, Vue.js is smaller in terms of the size of the resulting application and has higher performance, which is why I ultimately chose it.

[5] JS - JavaScript
[6] DOM - Document Object Model
[7] HTML - HyperText Markup Language
[8] CSS - Cascading Style Sheets

Components marked with the .vue extension can be created in two different API styles[9]: Options API and Composition API. In my project I use the newer Composition API (which addresses some shortcomings of the slightly older Options API), where we define the component logic using imported API functions. IN SFC[10] Composition API is usually used with <script setup>, or the equivalent <script> with the setup() method.

Another major advantage of Vue.js is the aforementioned reactivity, which is the ability of the framework to respond to changes in input data or state without the need for explicit response programming. This allows for a more dynamic and interactive user interface and makes application development easier. Vue.js works with a so-called declarative approach. This means that the developer defines the desired UI state and the system automatically takes care of updating it in response to changes. This is achieved by using data bindings that connect the UI components to the underlying data.

I use Vue Router to switch between the application screens (views). It monitors all changes in the URL and ensures transitions between screens without reloading the whole page, including passing URL parameters. If the user tries to enter a URL that is not valid in the application, he is automatically redirected to a page with a 404 - Not found error.

The data about the logged-in user is stored shared for all components in the Vuex Store. This data can only be changed via special methods called mutations. There are also shared functions, called actions, in which I have placed all requests to the backend API for clarity.

5.2 COREUI

CoreUI is an open-source framework for creating modern and responsive web interfaces. It offers a wide range of components, tools and UI elements that make web application development easier and faster. CoreUI is built on Bootstrap 5 and Sass technologies. It is compatible with major frameworks such as Angular, React and Vue.js.

It contains an extensive collection of pre-made components such as buttons, forms, tabs, modal windows and charts. It is designed to be intuitive and easy to use.

I used the CoreUI template to create the project and generate all the necessary components. Personally, I found this framework very easy to work with as I was forced to write only a small amount of CSS.

[9] API - Application Programming Interface
[10] SFC - Single-File Component

5.3 NODE.JS

I chose Node.js as the technology to create the backend API for my application. It's a runtime that allows you to run JavaScript code outside of a web browser. It is built on the Chrome V8 JavaScript engine, so the basis of this JS environment is the same as the Google Chrome web browser. It is important to mention that although you can write almost anything in this environment, its design and primary purpose is to create the server side of web applications (backend). This has commonalities with, for example, PHP or Python, which have the same focus. Unlike the aforementioned PHP, Node.js emphasizes high scalability and the ability to serve many connected clients at once. Because of this feature and high performance, Node.js is very popular today.

The core of Node.js is the event loop. All user requests enter it as events, which are then assigned to individual independent threads. Other operations, such as manipulating operating system resources, such as reading a file or working with a database, are then also handled by events (called non-blocking I/O[11]). Their call is again included in the event loop. In effect, this means that everything is managed by this loop using different events. This is a simple but very efficient solution, especially for web applications with a large number of user requests. JavaScript then fits into all of this very nicely, as these events are natively represented using JS callbacks.

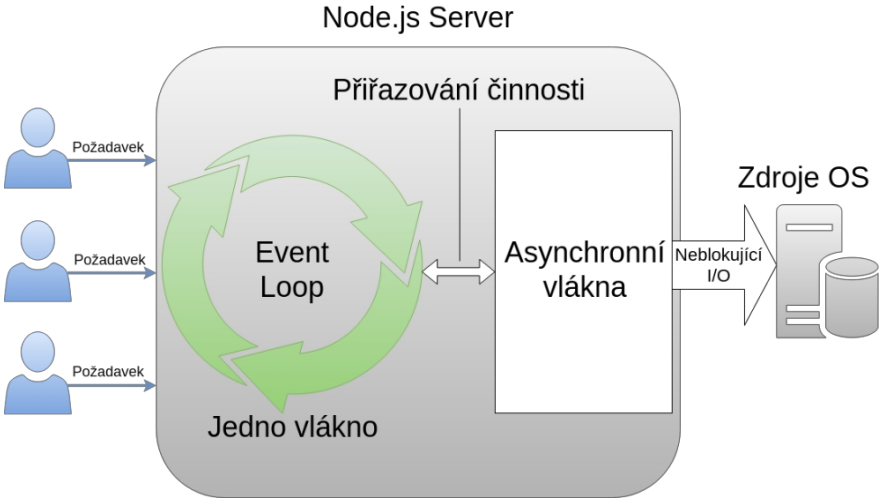

Figure 9 Event Loop

In my project, I use Express, which is a popular package for Node.js that simplifies working with the http protocol and is tailored to create REST APIs. It provides a simple environment with middleware options (e.g. JSON parsing) and error handling.

I'm using Node.js to create the API backend of my year-long project for the third time now and I've had a very good experience with it.

[11] I/O - Input/Output

5.4 MONGODB

For data storage I chose the MongoDB document database, which belongs to NoSQL databases. It is object-oriented, simple, dynamic and scalable. It uses collections instead of tables (as in a relational database), documents and arrays instead of rows and columns. Data is stored in BSON (Binary JSON) format, which is a binary-encoded form of JavaScript Object Notation (JSON) [5].

In our project we use Mongoose, which is a node.js package for MongoDB and Node.js. It allows us to manage relationships between data, provides data validation to prevent storing non-valid data and subsequent errors. For this purpose, schemas are used to define data types (Number, String, Boolean, Object, etc.). Along with this, the package also provides communication with the database (database server) and data transfer between the web application server and the database.

A schema is a sample data structure of a document that is validated by the Mongoose package when the data is saved. MongoDB as a database server itself does not work with *schemas*, so it does not have the ability to validate data. In my work, I use four data schemas to store documents in four collections:

- Users - users
- Roles - user roles (administrator, user)
- Apps - hosted applications
- Instances - instances of hosted applications

5.5 JWT

JWT (JSON Web Token) is a standardized format for user authentication and authorization in the form of a token. These tokens are compact, cryptographically signed and JSON-compatible, making them an ideal choice for login and authorization in web applications and APIs [6].

5.5.1 THE PRINCIPLE OF JWT-BASED LOGIN:

1. The client sends a request to the server with usernames and passwords.
2. The server verifies the validity of the entered data.
3. In the case of successful authentication, the server generates a JWT token that contains information about the user (ID, role, permissions) and signs it with a secret key.
4. The JWT token is sent back to the client and stored in its storage (in my c a s e localStorage, but a cookie is also possible).
5. On each subsequent request to the server, the client sends the JWT token in the HTTP request header.

6. The server validates the JWT token using the secret key and grants or denies access to the requested data/functionality based on the information in the token. [6]

5.6 CADDY

Caddy is an open source web server that is characterized by simplicity, speed and flexibility. Unlike conventional servers such as Apache or Nginx, Caddy focuses on clarity of configuration files and ease of use. Its unbeatable advantage over other web servers in use is the ability to configure in real-time using the REST API. This is the reason why I chose this webserver. Another big advantage is the built-in TLS retrieval and recovery feature[12] certificates to provide a secure HTTPS connection[13]. These Let's Encrypt[14] certificates are obtained fully automatically without any setup required. [7]

5.6.1 USE OF THE CADDY REST API

The caddy server is used as a reverse TSL proxy on the server, and the DUCK system uses its REST API to configure when it creates or deletes an instance of the hosted application. [8] A hosted application in the form of a container is assigned a single port on the server. The port interval assigned to containers is set in the DUCK system configuration file and is only used to redirect requests from the Caddy server locally. Thus, these ports are blocked in the firewall and all instances are automatically made available with the TSL on the configured domain set in the hosted application edit.

5.7 DOCKER

Docker is an open-source virtualization platform that allows you to encapsulate applications and their dependencies in containers. These containers are portable between Linux operating systems on which the *Docker engine* is installed, eliminating compatibility issues and simplifying the development process.

Docker Image is a static template for creating containers. It consists of a base image layer (e.g. Ubuntu, Debian, Alpine or other Linux systems) and additional layers that contain the installed software and configuration. The images are stored in registries such as Docker Hub and can be easily shared and reused.

Containers are created from a *Docker Image*. Using the Docker Engine, the image is run as an isolated process. Containers share the underlying operating system, saving system resources compared to

[12] TSL - successor to SSL
[13] HTTPS - Hypertext Transfer Protocol Secure
[14] Let's Encrypt - Certification Authority providing free TLS certificates

other forms of virtualization, such as virtual machines. Containers contain everything needed to run the application, including software, libraries and configuration.

5.8 STRIPE

I decided to implement the Stripe payment gateway to accept payments for hosted app subscriptions, mainly due to its friendly fees and ease of use. Compared to other payment gateways, there is one for almost every platform[15] SDK[16] with detailed documentation and tutorials, which made it very easy for me to develop. For use in this thesis, I have chosen only the API version for testing, as I do not plan to accept real payments in the near future and therefore do not want to perform personal data verification. However, it is possible to use the payment gateway and thus pay in the DUCK system using a test credit card number:

4242 4242 4242 4242 *12/33 (any future date) 123 (any three digits)*

6. PRESENTATION OF WORK

The DUCK system can be tested at *duck.ekdyson.site*. I would also like to thank the Gymnasium, Prague 6, Arabská 14 for providing the server to run this project. Among other things, this has made it possible to make the DEH system running as a hosted application on the DUCK system available to the public for testing.

[15] The SDK is currently available for Ruby, Python, Go, Java, Node.js, PHP and .NET
[16] SDK - software development kit

7. CONCLUSION

At the beginning I set a goal to program an application that will be able to automatically launch an instance of any web application that is pre-configured in the system. One of the goals was also to expose the DEH system to the public for testing prior to this year's regional round of the High School Vocational Competition. In doing so, I took advantage of a number of technologies and services that were new to me, such as the Caddy web server configured with a REST API, and the Stripe payment gateway. I met all the goals I set for the project and the result of my work is a real application.

8. RESOURCES USED

[1] Gymnasium, Prague 6, Arabská 14, "ŠVP - Application Software," [Online]. Available: https://www.gyarab.cz/media/svp/ICT_Aplika%C4%8Dn%C3%AD_software_2hu_p%C5%99.pdf.

[2] Gymnázium, Praha 6, Arabská 14, "WORDPRESS WORKS," 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.

[4] Docker Inc, "docs.docker.com," 2024 [Online]. Available: 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 Jun 2023 [Online]. Available: https://www.bezkoder.com/node-js-express-login-mongodb/. [Accessed 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. LIST OF FIGURES