



Assignment of bachelor's thesis

Title:	DevOps concepts - CI/CD, implementation of authorization & authentication, presented on a BI-DBS portal frontend
Student:	Volha Chukava
Supervisor:	Ing. Oldřich Malec
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Web Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2023/2024

Instructions

The BI-DBS portal undergoes an evolutionary transfer to a microservice architecture with a Vue.js frontend. Therefore, the frontend needs new automated deployment, testing, and a clear permissions structure.

Fulfill the requirements:

- Describe the current and planned state of the BI-DBS portal.
- Analyze and describe DevOps principles, focus on CI/CD.
- Configure and describe an automated deployment for the BI-DBS portal.
- Analyze roles and permissions of the BI-DBS portal.
- Implement and describe the authorization and authentication services and provide suitable regression tests for a CI pipeline.



**FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE**

Bachelor's thesis

**DevOps concepts – CI/CD,
implementation of authorization &
authentication, presented on a BI-DBS
portal frontend**

Volha Chukava

Department of Software Engineering
Supervisor: Ing. Oldřich Malec

May 10, 2023

Acknowledgements

Firstly, I would like to thank my supervisor Ing. Oldřich Malec for giving me the flexibility in creating my thesis, constructive and valuable feedback and assistance. Secondly, I would like to thank a person who made it possible for me to write this thesis, the maintainer of the project Ing. Jiri Hunka. Thirdly, I would like to acknowledge the contribution of Ing. Adnrii Plyskach and Bc. Max Hejda, who have shared their knowledge, expertise, and resources, and have played an important role in helping me to achieve my thesis goals. Finally, I would also like to thank the testers, who contributed their time and effort. My family and friends who supported me throughout my study period.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No.121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 10, 2023

.....

Czech Technical University in Prague
Faculty of Information Technology

© 2023 Volha Chukava. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Chukava, Volha. *DevOps concepts – CI/CD, implementation of authorization & authentication, presented on a BI-DBS portal frontend*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Abstrakt

Pro předmět BI-DBS na FIT ČVUT v Praze je vyvíjena nová aplikace. Požadavky pro tuto aplikaci jsou odvozeny z již existující aplikace. Tato bakalářská práce se zaměřuje na zlepšení procesů vývoje a údržby nového frontendu webové aplikace na základě analýzy stavu stávající aplikace a plánovaného stavu. Hlavními oblastmi zlepšení jsou přijetí metodiky DevOps a navržení nového zjednodušeného a přehledného systému řízení přístupů. Nakonec hlavním cílem této práce je implementace autentizace a autorizace, včetně omezení oprávnění, a automatizace procesů testování a nasazování.

Klíčová slova webová aplikace, frontend, CI, CD, OAuth, autentizace, autorizace

Abstract

A new application is being developed for teaching the BI-DBS subject at FIT CTU in Prague. The requirements for the application are derived from the existing application. This bachelor's thesis is focused on improving the development and maintenance processes of the new frontend for the web application based on analyses of the state of the existing application and the planned state. The main improvement points are adopting DevOps methodology and designing a new simplified and clear access management system. Conclusively, the main goal of the thesis is the implementation of authentication and authorization including restricting permissions, and automation of testing and deployment.

Keywords web application, frontend, DevOps, CI, CD, OAuth, authentication, authorization

Contents

List of Code Examples	xix
Introduction	1
1 Analysis of the application state	3
1.1 The BI-DBS portal	3
1.2 Current state of the application	3
1.2.1 Architecture	4
1.2.2 Technologies	5
1.3 Planned state of the application	6
1.3.1 Architecture	7
1.3.2 Technologies	8
1.4 Summary and implications	9
1.4.1 Summary	9
1.4.2 Implications	10
2 Analysis of the DevOps model	13
2.1 What is DevOps?	13
2.2 DevOps concepts	13
2.2.1 Automation	14
2.2.2 Data-Based Decision Making	14
2.2.3 Responsibility Throughout the Lifecycle	14
2.2.4 Constant Improvement	15
2.2.5 Collaboration	15
2.3 DevOps cycle and practices	16
2.3.1 Continuous development	16
2.3.2 Continuous integration (CI)	16
2.3.3 Continuous delivery (CD)	16
2.3.4 Continuous deployment (CDE)	17
2.3.5 Continuous monitoring (CM)	17

2.3.6	Infrastructure as Code (IaC)	17
2.3.7	Containerization	17
2.4	DevOps adoption	17
3	Analysis and design of the access control system	19
3.1	Roles	19
3.1.1	KOS roles	19
3.1.2	Other roles	22
3.2	Permissions	23
4	Implementation	27
4.1	Used software	27
4.1.1	Pinia	27
4.1.2	Router	28
4.1.3	Axios	28
4.2	Authentication and authorization	28
4.2.1	OAuth protocol	28
4.2.2	Tokens	28
4.2.2.1	Access token	28
4.2.2.2	Refresh token	29
4.2.3	Authorization server	29
4.2.4	Apps manager	30
4.2.5	Authentication and authorization flow	31
4.2.6	Refresh token flow	32
4.3	Security	33
4.3.1	Tokens storage	33
4.3.1.1	Solution for the refresh token	34
4.3.1.2	Solution for the access token	35
4.3.2	Communication with backend	36
4.3.2.1	Communication	36
4.3.2.2	HTTP headers	36
4.4	Access control	37
4.4.1	Role-based access control	37
4.4.2	Authentication control for requests	40
5	Automation – CI/CD	43
5.1	Used tools	43
5.1.1	Docker	43
5.1.2	GitLab	43
5.1.3	CloudFIT	44
5.1.4	Buildah	44
5.1.5	Nginx	44
5.1.6	Yarn	44
5.2	Implementation	44

5.2.1	Containerization	45
5.2.2	CI/CD pipeline configuration	45
5.2.2.1	Test	46
5.2.2.2	Build	46
5.2.2.3	Deploy	48
5.3	Documentation	50
6	Testing	51
6.1	Tests for CI	51
6.1.1	Vitest	51
6.1.2	Tests	51
6.2	Manual testing	54
	Conclusion	59
	Sources	61
	A Acronyms	67
	B Contents of enclosed media	69

List of Figures

1.1	Monolithic architecture, MVP pattern	4
1.2	Microservice architecture	7
2.1	Devops cycle and practices	16
3.1	KOS roles	21
3.2	Other roles	22
4.1	Example of encoded and decoded JWT token from jwt.io. [41] . .	29
4.2	The example of the application registry in the App Manager [44] .	30
4.3	Sequential diagram of generating an access token	31
4.4	Sequential diagram of refresh an access token using refresh token .	32
4.5	Access denied component	39
5.1	CI/CD pipeline in GitLab	46
5.2	Documentation of server and CI/CD management	50
6.1	CI tests	54

List of Tables

1.1	Visualisation of changes.	10
6.1	Scenario for manual testing.	56

List of Code Examples

4.1	Cookies setting on server	34
4.2	Pinia store configuration for user states	35
4.3	Base Axios configuration	37
4.4	The example of component configuration in Router	38
4.5	Validation of access by role	38
4.6	Role validation	40
4.7	Filtering displayed components by role	40
4.8	Access verification for making requests	41
5.1	Dockerfile	45
5.2	Gitlab CI/CD stages	45
5.3	Test stage in the CI/CD pipeline	46
5.4	Build stage in the CI/CD pipeline	47
5.5	Deploy stage in the CI/CD pipeline	48
5.6	Deployment script	48
5.7	docker-compose.yml	49

Introduction

This thesis is a continuation of my development journey of the BI-DBS web application, which started in February 2022 as a part of the frontend development team. The project was very complex and challenging for maintenance and development. Not a long time after our team got into analyzing and developing the current application, a new solution to the maintenance problem was introduced to us by Ing. Andrii Plyskach. He came up with a new architectural design based on microservices architecture for the application, which he described in his master thesis[1]. The decision was made to follow that design, which meant creating a new project based on the requirements of the current application. The development process, as well as the project, was divided into client and server side parts.

For our team, it meant starting frontend development from the very beginning using the new technologies. Thus it gave us a lot of space for our ideas, but also a big responsibility. Therefore, since the creation of a new frontend project, my goal was to configure the project in a way to make it structured and secure, organize efficient development in a team as well as work on the implementation of the features. This goal remains the same for this thesis.

Firstly, I am going to analyze the current application state and the planned state, to see what we can expect from the changes. Secondly, for improving the efficiency of the software development process and application improvement and maintenance I will introduce the DevOps model and the instruction for its adoption. Besides, I will use that instruction for automating testing and development processes for the frontend. Finally, I am going to implement authorization and implementation based on OAuth 2.0 protocol[2] using the authorization microservice implemented in Andrii's thesis[1].

Analysis of the application state

In this chapter, I will introduce the educational web application helping to teach database systems subject at the university. Furthermore, I will describe the current and planned state of the application from the perspective of software architectural patterns and the used set of core technologies with a focus on the frontend. The goal is to identify the existing problems of the current application, outline how they will be solved in a new portal, as well as to indicate what difficulties we can face developing the new application using a new stack of technologies and new architecture.

1.1 The BI-DBS portal

The BI-DBS portal is a web application used for teaching database systems subject in a bachelor's study program at the Czech Technical University at the Faculty of Information Technology. The portal is complex and has many useful functionalities. It allows managing and tracking all the student's studying progress during the semester, including semester tests, complex semester work, and exams. Besides, teachers have an overview of all their student's work in one application.

The application is developed by students and teachers in subjects BI-SP1, BI-SP2 subjects, bachelor and master theses. That is a unique fact about this project. Every year, new students begin working on application development. They are open to sharing their ideas for improving the application. Thus, we are designing and implementing a better and better product each year.

1.2 Current state of the application

The current BI-DBS portal was deployed for the first time in 2016 [3]. Over time it gained new features and grew large. Currently, it has a total of around

120000-140000 lines of code [1]. Used technologies became less relevant and it became difficult to maintain it.

1.2.1 Architecture

The current application was built in a traditional way, using a monolithic architecture approach and following the Model-View-Presenter architectural pattern [4] [5]. Figure 1.1 shows the visualization of the architecture. The application is presented as one monolithic unit, and it is composed of three components.

- *The model:* Communicates with the database and handles domain and business logic.
- *The view:* Provides visualization and directs user commands to the presenter, does not contain logic.
- *The presenter:* Manages interactions between the database and the view. Receives data from the model and formats it to display in the view.

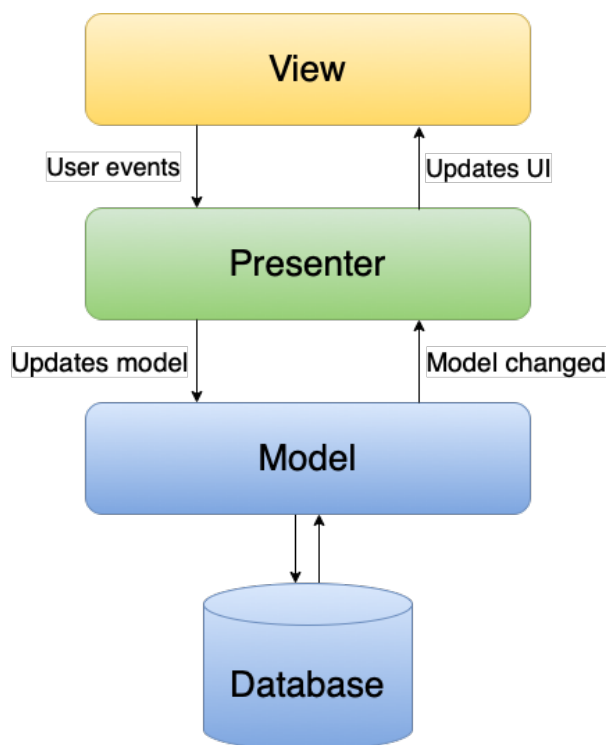


Figure 1.1: Monolithic architecture, MVP pattern

This architecture's main concept is having one code base that benefits in simplifying development, testing, debugging, and deployment. However, we can have those benefits only until the application grows large. Then all those processes get slower, more complex, and become problematic. In addition, with a lack of flexibility and scalability, it becomes challenging to maintain the application and keep it secure.

The BI-DBS portal is being developed by students. Students generally do not have much experience developing large applications and dealing with complex dependencies. Besides, they have limited time to progress in learning and then designing and developing the portal. Therefore it takes a lot of time for students to learn before contributing to the project. Thus it is more problematic for students to benefit from learning and for maintainers to keep it functioning correctly.

1.2.2 Technologies

PHP. PHP is a general-purpose, open-source scripting language that can be integrated into HTML. [6, 7] It differs from client-side scripting languages in that its HTML is generated on a server and then sent to a client. That feature allows rapidly building a web application with a thick server and thin client. This is one of the approaches to using PHP to build an application, and it is used in the current project.

Using this approach leads to creating dependencies between the user interface and the application logic, which makes any changes more effortful since a developer needs to adjust it on both sides.

Doctrine. "Doctrine ORM is an object-relational mapper for PHP 7.1+ that provides transparent persistence for PHP objects. It sits on top of a powerful database abstraction layer. One of its key features is the option to write database queries in a proprietary object-oriented SQL dialect called Doctrine Query Language." [8]

This framework did not cause problems during the development process and has no significant disadvantages for the correct operation of the BI-DBS.

Nette. Nette is an open-source framework for creating web applications in PHP. It helps with developing both the client and server sides of the application and also reduces security vulnerabilities. Moreover, it manages application states using sessions and routing. [9]

Frontend and backend dependencies are strengthened, indicating that they are a single unit. The fact that they are so strongly dependent is a drawback. Because of this, it is difficult to make changes to one side without having an impact on the other.

Latte. Nette framework uses a template system called Latte. It compiles templates down to the optimal PHP code. [10]

AdminLTE. AdminLTE is a fully responsive administration template. Based on Bootstrap 4.6 framework and also the JS/jQuery plugin. [11]

Vue 2. Vue.js is a javascript framework for building user interfaces and single-page applications.

Most of the frontend is implemented using Latte templates and AdminLTE bootstrap. However, in order to reduce dependencies between the frontend and the backend and also modernize it, a few components were refactored to the Vue.js version 2. The logic is defined using the Options API. It is a traditional object-oriented way, and up until Vue 2 it was the only way to create components in Vue. [12, 13]

Javascript. JavaScript is a high-level programming language used for defining the behavior of webpages. It is a dynamically-typed scripting language that lets you control multimedia, animate graphics, and generate dynamically changing content. [14]

In the current BI-DBS portal it is used for defining logic on the frontend. Dynamically-typed languages are easy for development, but this feature reduces the code's readability, requires more testing and are prone to run-time errors. Large applications like BI-DBS are likely to experience problems as a result of its drawbacks because it is better suited for smaller applications with simple logic.

Webpack. Primarily, Webpack is a static module bundler for modern JavaScript programs. When Webpack processes your application, it internally creates a dependency tree from one or more entry points and then merges every module your project requires into one or more bundles. [15]

Webpack is used in a current application for bundling a few modernized frontend components implemented in Vue.js and javascript.

1.3 Planned state of the application

The main reason for creating a new application instead of refactoring the current one is a change in the application's architecture. A new modernized architectural design of the BI-DBS portal was composed by Ing. Andrii Plyskach in his master thesis [1]. We are aiming to correct all the mistakes made in the current application. It is essential to ascertain that we have chosen the right stack of technologies according to the newly chosen architecture.

1.3.1 Architecture

Microservices architectural pattern [5] is based on the concept of a series of loosely-coupled services. They can be developed using different technologies and deployed independently. It is more complex architecture than a standard monolithic one. You can see the diagram illustrating microservices architecture in Figure 1.2.

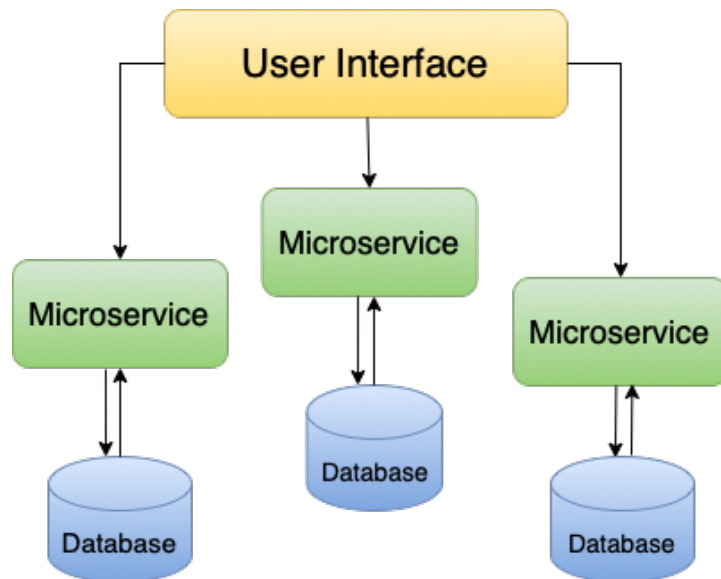


Figure 1.2: Microservice architecture

Advantages:

- *Code readability:* When services are not strongly dependent the code appears to be better structured and easier to understand, which is a significant benefit for the BI-DBS portal.
- *Independency in choosing a stack of technologies:* Microservices can be developed using different technologies which can be chosen according to each microservice functionality without affecting other microservices.
- *Faster deployments:* Since all microservices can be deployed independently, the deploying part is much smaller and the time for deploying one service is rather shorter.
- *Fault tolerance:* Because of loose-coupling, failing one of the microservices will not bring down the entire application.

Disadvantages:

- *Difficult debugging and testing:* Each service needs to be first tested separately and only then as one unit. Besides, it is more difficult to track down errors.
- *DevOps required:* To benefit from the fast deployment it should be configured and maintained. It requires knowledge of development operations.
- *Longer development time and limited reuse of code:* Microservices need to be managed separately, therefore it requires more time.

1.3.2 Technologies

PHP. Since version 5.0, PHP supports object-oriented functionality [16]. PHP is easy to learn, flexible, and supports all required functionalities for our application. It is used in a new project for a domain and business logic on the backend for API implementation.

Symfony. Symfony is a powerful backend framework for creating complex applications which consists of reusable components. [17] Thanks to Doctrine Symfony provides all of the tools required to use databases in the application. It is constantly growing and improving, besides it has a strong community. It is easy to learn and has well-written documentation.

Vue 3. When the decision to create a new BI-DBS portal had not yet been made, its frontend was getting modernized by rewriting components to Vue.js version 2. In the new project, it was chosen to carry on using the Vue.js framework but use a new version 3. This version comes with certain advantages for the application. [18]

New features:

- *Composition API:* Composition API is a set of APIs that allow us to create Vue components by importing functions rather than defining options. It mainly benefits our project in better code organization, thus making the project better structured and the code easier to read. Moreover, Composition API enables efficient logic reuse. [19]
- *Vite:* Vite is a frontend build tool from the creator of Vue.js - Evan You. It is a module bundler that bundles the entire project on startup, hot reloads, and compilation. The primary purpose for the change is for speed. The server starts instantly since it uses native browser support for JavaScript modules. [20]

- *Increased rendering performance.*
- *Smaller Vue core.*

Typescript. Typescript is based on JavaScript, which is dynamically typed. TypeScript has an additional syntax that makes it statically typed. That has advantages in catching errors during development. It also gives a code more structure and makes it predictable. Typescript is more suitable for big applications than JavaScript. For our project, writing code that will be easy to read is crucial. [21]

Quasar. Quasar is a web framework based on Vue.js. It provides us with ready-to-use components which are customizable and easy-extendable. Moreover, it makes the application less vulnerable to XSS attacks due to its escaping feature. When using Quasar, developers do not need deep knowledge of CSS and scripting languages to build a good-looking and responsive application. Besides, it is suitable for developing single-page applications(SPA). [22]

Pinia. Pinia is a Vue.js storage library and state management framework. It is mainly designed for the development of front-end web applications, and it uses declarative syntax as well as its own state management API. It allows sharing a state across components and pages securely. Moreover, it has server-side rendering support. With Pinia plugins we can also persist the state across page reloading using local or session storage. [23]

1.4 Summary and implications

The BI-DBS development team aims to dispose of problems and modernize the current project in every single aspect of development. Starting from choosing the right stack of technologies and designing a suitable architecture to implementing more complex and valuable features. However, even correctly chosen technologies and architecture for reducing the problems of the current project do not save us from the potential new challenges brought by the changes.

1.4.1 Summary

In order to summarize all changes and provide a better visualization of them, I arranged them all in Table 1.1.

Evidently, the Nette framework is the core of the current project, which is responsible for managing the application in many ways. Although it can function well, it creates dependencies between the functionalities and makes the project less flexible, which is a significant disadvantage for large applications like the BI-DBS portal.

1. ANALYSIS OF THE APPLICATION STATE

The planned state does not have dominating technologies that would cause this problem. Most of them are replaceable and flexible.

	Current state	Planned state
Architecture	Monolithic	Microservices
Backend language	PHP 7.2	PHP 8.0
Backend framework	Nette	Symfony
Frontend framework	Nette, Vue.js 2	Vue.js 3
Frontend templates and styling	Latte, AdminLTE	Quasar
Scripting language	Javascript	Typescript
Frontend templates and styling	Latte, AdminLTE	Quasar
Module bundler	Webpack	Vite
State management	Nette Sessions	Pinia
Routing	Nette Router	Vue.js Router

Table 1.1: Visualisation of changes.

1.4.2 Implications

From the analyses in sections 1.2 and 1.3, we can see that existing problems in the current project are eliminated by chosen architecture and technologies for the planned state. Let's examine the main possible negative effects of the changes and how to deal with them.

- Microservices architecture provides such advantages as agility and fast deployment. This architecture is more complex in comparison with a monolithic one. Therefore it comes along with establishing some of the DevOps principles for the project. Mainly configuring continuous integration and automated deployment. DevOps concepts and automation including CI and CD are described in the chapter 2.
- In the current application, Nette is a core full-stack framework that is also responsible for managing the application's security. Besides, the monolithic architecture allows you to store all the data on the server side. The communication between the client side and server side is secure. In microservices applications, there is constant communication between the frontend and the backend and exchanging sensitive data. Therefore it is crucial to control every step of that communication with control of permissions and inputs validation on both the client and server sides. Thus the application should have a clearly defined access management system which I will introduce in the chapter 3.
- Since all the services are developed, deployed and tested separately, there is a higher chance of failure in communication between them. Obviously,

1.4. Summary and implications

It is not enough to test only the functionalities of single services but to test their integration. Therefore it is essential to design a new integration testing system for the application. It is necessary to eliminate the possibility of the cascade failure of services.

Analysis of the DevOps model

The DevOps and microservices are two important trends in application development. Considering that both of them are mainly focused on providing better agility, flexibility, and operational efficiency, we can assume that they would work better together. [24]

In this chapter, I will describe the main DevOps concepts and practices, analyze their possible advantages for the BI-DBS application and decide whether adopting the DevOps model would be beneficial.

2.1 What is DevOps?

The term **DevOps** is derived from the combination of software **d**evelopment and IT **o**perations.

DevOps is a relatively new term. Around 2007 and 2008 concerns about the separate work of software creators and software operators were raised. The concept started to grow on online forums and meet-ups. The first conference named "DevOps" was held in 2009. [25]

DevOps is a software development methodology composed of a set of cultural philosophies, practices, and tools that improve an organization's ability to deliver applications, services, and improve products faster than traditional software development and infrastructure management processes. It represents a cultural shift that significantly affects a team that adopted that methodology and the software they make. [26]

2.2 DevOps concepts

DevOps concepts are a common set of rules which are the core of this methodology. It is not just a set of tools, but a cultural philosophy, a way of project lifecycle organization. Those rules are not strictly defined, they come from a DevOps culture and can be interpreted differently describing the same model.

In this section, I will analyze and combine the culture philosophy and most frequently mentioned rules [27, 28] in five concepts that represent the DevOps methodology.

2.2.1 Automation

DevOps approach is meant to benefit in fast development and improvement. Needless to mention that automation is one of the golden rules for increasing the speed of the application lifecycle. Everyone in a team should aim to automate as many phases of the process as it is possible. As a result, team members are satisfied with a decreased need for doing repetitive tasks. Thus they can focus on significant tasks and work on new features. Overall it helps minimize human errors and boosts team output.

Usage in the BI-DBS project This concept was the main reason for me to consider adopting DevOps model in the BI-DBS project. Due to microservices architecture the project needs to have new automated deployment and testing processes. These and other automation practices we might want to adopt are described in subsection 2.4.

2.2.2 Data-Based Decision Making

With the DevOps approach, decisions from choosing a technology stack for the application to adding features should be made based on collected data. The first part of making a decision should always be collecting as much relevant data as it is possible. Then, based on the collected data analysis of the team, a decision should be made. It helps to create software that solves real problems effectively. Decisions made without considering client feedback data, colleagues' opinions, and proper analysis would lead to creating badly-functional software full of useless features which does not fulfill the client's needs.

Usage in the BI-DBS project This concept is very suitable for our new growing project since in the current phase we create are creating a core which should be done properly based on analyses of collected data to avoid having useless features, too complex design and irrelevant technologies.

2.2.3 Responsibility Throughout the Lifecycle

DevOps methodology comes with a requirement for team members to fully understand the process of software development from the feature idea to implementation and deployment and take responsibility for it. End-to-end responsibility helps to reduce failures and resolve bugs quickly.

Usage in the BI-DBS project From my experience, students usually want to finish their part of the job as fast as it is possible. Therefore they sometimes tend to skip spending time to understand the idea of the task properly. Thus they get to implement it without thinking of the consequences their changes might cause. Moreover, they do not always get to test it properly. It is essential to integrate this concept more into development student teams to increase the quality of produced code.

2.2.4 Constant Improvement

Constant improvement is a special concept and practice of DevOps methodology. The main idea is a focus on improvements, updates, and experimenting. It tells each team member not to be afraid of failures but take them as an opportunity to learn. Whatever the outcome of an experiment, a person will have a deeper understanding of what works and what does not. Besides, this rule gives more responsibility to a person and allows them to consistently push code changes to minimize waste, do speed optimization and improve development efficiency.

Usage in the BI-DBS project This concept is friendly for students in a way that they can try new things without fear of failure if things do not work out. Adopting this concept will benefit the project in case it is used with the two previous concepts, otherwise, students might take the development less seriously and do experiments only for the purpose of faster finishing the task, but not improving.

2.2.5 Collaboration

This concept is a collaboration of different IT departments on the project. That means that the team's roles are not as strict and independent as in a traditional work and team organization. Developer and operation roles are getting closer to full-stack roles, leading to a better understanding of the software development lifecycle by the whole team.

Usage in the BI-DBS project This illustrating DevOps methodology concept is beneficial in two ways for the BI-DBS portal. Firstly, students will learn essential operation processes and understand the basics of automation. Secondly, the portal always needs at least one person to be available to manage application operations. Using this concept will increase the number of people who understand the processes and thus are able to manage operations in case of a need.

2.3 DevOps cycle and practices

DevOps concepts reflect in a set of practices during the DevOps delivery cycle. The cycle is visualized in Figure 2.1. These practices mainly represent the automation concept but also come together with other concepts. [29]

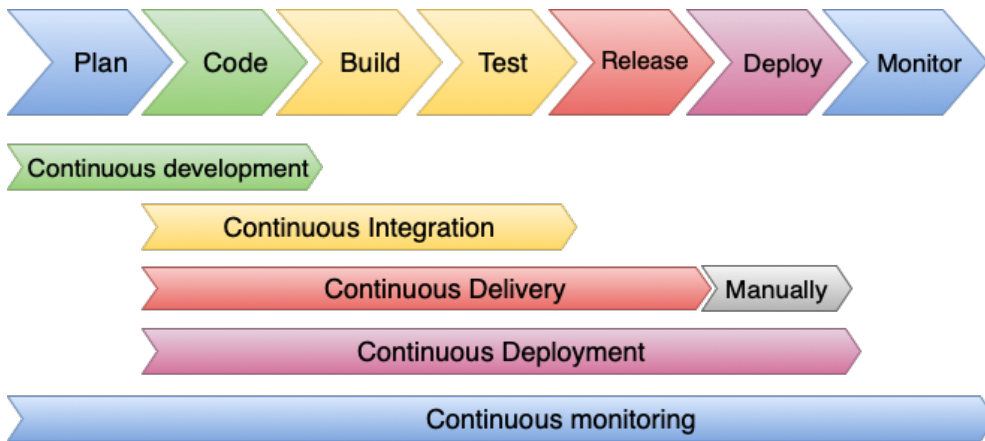


Figure 2.1: Devops cycle and practices

2.3.1 Continuous development

Continuous development is a practice composed of agile planning and coding. The goal of agile planning is to divide big problems into smaller logical problems, estimate the complexity of created tasks and plan the amount of tasks for some short time period(sprint), usually it is from one to four weeks period. This method allows getting some large significant tasks done in a shorter time, because after its division developers can work on the subtasks simultaneously and it is more effortless for testing.

2.3.2 Continuous integration (CI)

In order to avoid a problem with the integration of large parts of code, DevOps CI offers continuous pushing the code changes to the remote shared repository on the server. Every change pushed to the repository triggers a build and tests configured in a CI pipeline to make sure new changes do not affect already functioning features and also does not contain new errors.

2.3.3 Continuous delivery (CD)

Continuous delivery is an extension of continuous integration. After building and testing the code from the repository it automatically deploys releases to the testing environment and also prepares it to be deployed to the production.

It requires human intervention to deploy a release to production. This is a safer version of fast and frequent deployment, in a case when the pipeline does not contain strong testing tools and the application needs to be tested manually.

2.3.4 Continuous deployment (CDE)

Coupled with continuous delivery, the continuous deployment also deploys the release to the production. Using this practice no human intervention is required. Every change pushed to the main shared remote repository will be automatically deployed to production. The only obstacle to the deployment would be a failed build or test.

2.3.5 Continuous monitoring (CM)

Continuous monitoring is an automated method that allows to observe and discover compliance concerns and security vulnerabilities throughout the DevOps lifecycle. It also finalizes the cycle by providing feedback on monitoring and informing about existing or possible failures. It helps to resolve issues in real-time.

2.3.6 Infrastructure as Code (IaC)

Infrastructure as a code is a practice of managing infrastructure that enables automation in the DevOps lifecycle. It offers using scripts for configuring deployment environment and other infrastructure, including establishing a version control system.

2.3.7 Containerization

Containerization is the practice of packaging an application in one container. It provides better flexibility for deployment and needs fewer resources to run. Currently, Docker provides the most frequently used container toolset.

2.4 DevOps adoption

The idea of adopting the DevOps model came to me with a need to configure the new deployment of the new BI-DBS portal due to the transfer to microservices architecture. Before analyzing the DevOps concepts, I assumed that the DevOps model is just an automation idea. In fact, I was wrong and did not know it is a solid methodology bringing huge advantages to the project. From my own observations, it is a pretty common misunderstanding of the DevOps model, which leads to missing out important concepts.

”Even while automation helps speed up manual operations, cooperation and communication are the key objectives of DevOps. Automating your operations won’t bring about the desired business benefits unless everyone involved in the software development, delivery, testing, and operating processes adopts excellent communication and collaborative practices.” [30]

The analysis makes it clear that the DevOps model is suitable and valuable for the BI-DBS portal project management and development.

Adoption steps:

1. *Devops philosophy.* This thesis can be used to introduce the DevOps methodology to students. Before getting to development as well as learning the processes of development students should learn team organization management including DevOps concepts.
2. *DevOps Practices.* Analyze which practice we would like to adopt and how it will be beneficial and then complete the three next steps:
 - a) Choosing relevant tools for a practice we would like to adopt
 - b) Application of the practice using chosen tools
 - c) Document the configuration of the practice for a team

I will adopt the most important DevOps practices for the BI-DBS portal in the chapter 5 using these steps.

Analysis and design of the access control system

The BI-DBS portal uses role-based access control which is the way of granting access to the authorized user based on their role that is associated with their identity. [31]

In this chapter I will describe the access control system designed for the new BI-DBS portal frontend, based on roles and permissions analysis. I will introduce existing roles, the main application modules and their components from the perspective of different roles. My goal is to make an access control system as clear as possible and provide an overview of the permissions for the roles.

3.1 Roles

The role itself is a collection of permissions. To decide what roles the application needs and what permissions those roles would have, we have to start by analyzing the users of the application and their needs. My analysis will be based on the existing roles and access control system.

Since it is the best practice to assign as few roles to one user as it is possible [31] to avoid creating an excessively complex management system I will aim for simplification of the current access control system. In this section I will describe the roles of the current BI-DBS portal and offer a simplification for the new application, which I will use for my implementation of role-based access control described in section 4.4.1.

3.1.1 KOS roles

The BI-DBS portal receives information about an authorized user from the study information system(KOS) based on the course information. The course information contains the identifier of the semester and the type of study program. Generally, there are seven user roles that are defined by the KOS for

subjects and courses. [32]

KOS roles for subjects and their general description: [33]

- *Guarantor*. Guarantor is a course administrator. Thus a person with this role typically has all permissions across all course management.
- *Examiner*. Examiners are responsible for managing and estimating students' exams. Therefore, this role would usually provide access to exam materials and students' grades view and management.
- *Editor*. Editor is a person who can edit the information about the subject. This role would provide access to subject information management.
- *Lecturer*. Lecturer role indicates that an individual with this role would need to be provided with access to manage course content including lectures, assignments and other study materials.
- *Instructor*. Instructor is the role for teachers of exercises parallels. This role implies that a user needs permission to manage exercise materials and also estimate students' tests, semester works and other assignments.
- *Teacher*. The teacher is a general role for a person, who does teaching in the course.
- *Student*. It is a base role for students, that usually grants basic permissions to a user like access to their personal information, study program information and its resources, and also allows managing their projects and submitting assignments.

One of those roles is not used for the BI-DBS portal. It is the editor role, the BI-DBS portal does not provide functionalities for changing the information about the subject. Therefore, we have a total of six roles used for permissions control in the current application.

Based on the feedback from the teachers and developers of the current BI-DBS portal and also my own research, we came to the conclusion that the access management system can be simplified by grouping the roles into three roles in total.

Reasons for simplifying the access management:

1. Based on the permissions research of the current project, I can report that most of the roles for teachers have the same or almost the same permissions. And the differences are insignificant.
2. All the analyses and requirements of the BI-DBS portal designed by students including the theses are made for only three roles.

- Due to a lack of information about the roles developers often do not have a good understanding of all roles meaning and thus they tend to forget to permit access for some roles or confuse them with others.

I am offering the simplification, which is based on grouping teaching roles that do not have significant permissions differences such as lecturer, examiner, instructor and teacher into one role. As I have already mentioned above, all the analyses are usually built on three roles where these four roles are taken as one role - teacher role, then there is a student and guarantor roles left which makes it a total of three.

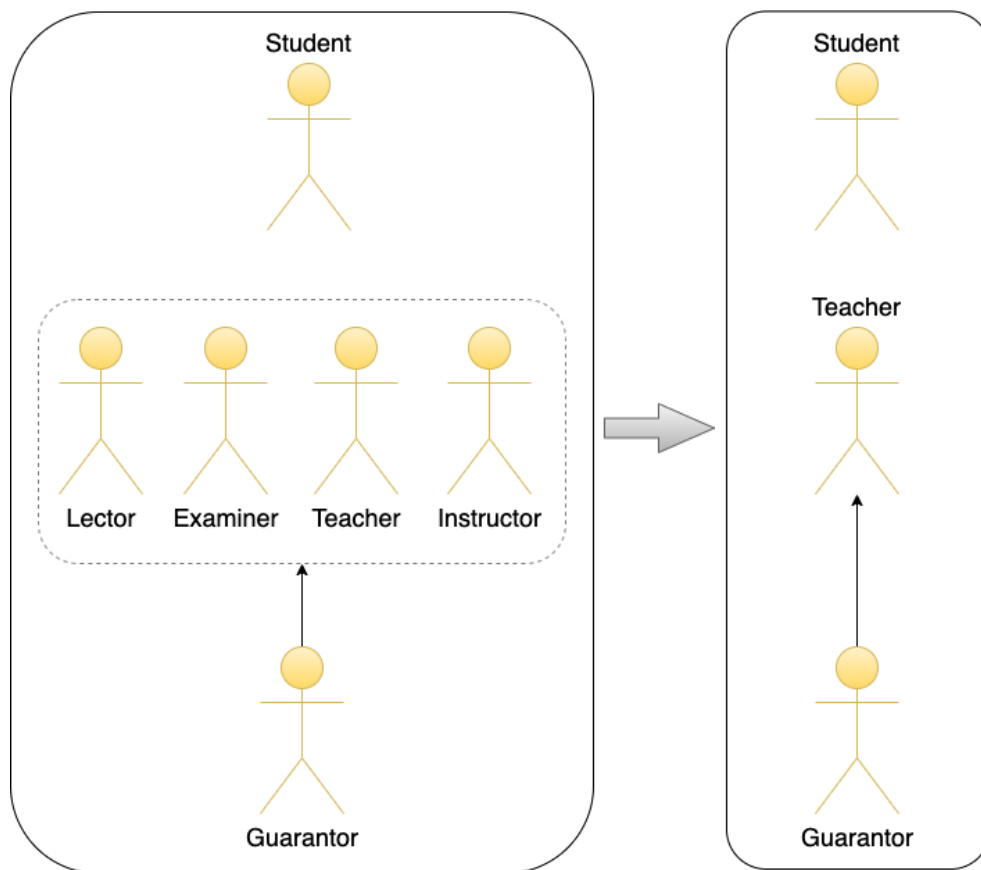


Figure 3.1: KOS roles

After researching the functionalities and permissions structure of the current project and discussing the possible change with developers and managers, I came to the conclusion that it is absolutely safe to generalize the permissions for those four teacher roles. As a result, I got a new clear KOS roles structure, which is visualized in Figure 3.1.

3.1.2 Other roles

However, the user roles defined by KOS are not fulfilling all the requirements for the application. There are some special cases that require additional roles such as:

- *Impersonation as a student.* For the purpose of demonstrating the process of creating a semester work and its management, teachers need to have a functionality that will allow them to authorize as a student to show the whole process from the students' side. This feature requires creating a test student, which is identical to a usual student but needs to differ from the usual KOS student role to exclude such student records from counting the statistics.
- *Development and testing.* For the development and testing processes there is a need to have a test admin user, which will have access to all functionalities.

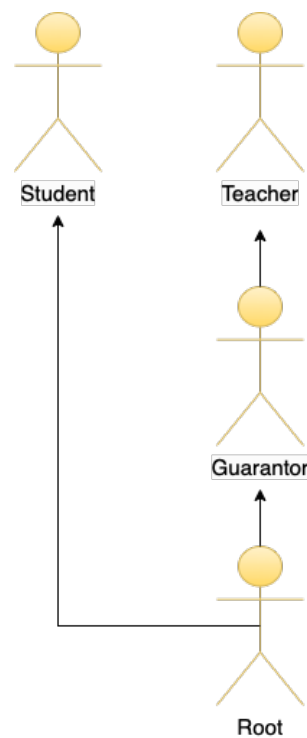


Figure 3.2: Other roles

For these requirements there were created two corresponding special roles:

- *Test student.* This role is identical to the KOS student role from the perspective of permissions. Therefore on the frontend we do not need

to differ those two roles and the student test role will be mapped to a usual student role.

- *Admin(Root)*. Role with access to all functionalities.

Conclusively, the solution for the access control system lowered complexity and added flexibility, due to the rule that one user can have only one role. Besides, now the roles system is very intuitive in usage. The result of constructing a roles system for the role-based access control is visualized in Figure 3.2.

3.2 Permissions

One user can have only one role at the same time, but one role has many permissions. The BI-DBS portal is a complex application composed of several modules. For giving an overview of the permissions for the defined roles I will describe the modules and their functionalities first and then classify them for groups and describe permissions to them from the perspective of users with different roles.

Modules and their functionalities:

- *Administration*. The administration module provides functionalities for semester configurations.
- *Semester work*. The semester work module contains all the features for managing the semester work from creation to evaluation.
- *Tests*. All components for the management of demo, semester and exam tests are placed in the tests module.
- *Connections*. The module which provides the configuration of the database connection is the connections module.
- *Students' score*. Users of the application can see the results of the student's performance during the semester in the student's score module.
- *Users*. Users module provides an overview of the users of the portal.
- *Data modeler*. Data modeler is a playground for drawing conceptual schemas.
- *Transformation modeler*. Transformation modeler is an extension of the data modeler, which also provides the generation of a create script based on the drawn schema.
- *Home*. The home page is composed of the overview of the semester.

3. ANALYSIS AND DESIGN OF THE ACCESS CONTROL SYSTEM

- *Authorization.* The authorization module provides login and logout features.

These modules can be divided into three groups:

1. Modules with the same permissions for all roles: transformation modeler, data modeler, connections, authorization.
2. Modules available only for a certain role: administration, users.
3. Modules available for all the roles but with different permissions for their components: semester work, tests, students' score, home.

The first group does not need to be provided with an access management structure as all the modules from the group are available for any authorized user with any of the roles described in the previous section. Therefore the access validation to these modules and their components is simple and clear. The modules from the second group are available only for two roles: guarantor and root.

Finally, the last group of modules has a more complex access management structure. These modules mostly have two types of components. The first type is components accessible for student, test student and root roles and the second type is accessible for teacher, guarantor and root roles. Therefore in the short description of the module's permissions by roles, I will use only two roles, student for the first type and teacher for the second type.

Semester work

- *Permissions for student.*
 - Semester work editor
 - Check and submission
 - Classification requirements
- *Permissions for teacher.*
 - Submitted semester works and submission status view
 - Semester works evaluation
 - Import and set deadlines
 - Classification requirements

Tests

- *Permissions for student.*
 - Taking demo tests
 - Taking assigned tests
- *Permissions for teacher.*
 - Create and edit assignments
 - Create and edit questions
 - Create test templates
 - Assign and start tests
 - Evaluate tests
 - Tests classification and statistics

Students' score

- *Permissions for student.*
 - View students' score with anonymized personal data
- *Permissions for teacher.*
 - View students' score

Home

- *Permissions for student.*
 - View of personal and course data
 - View of personal progress in a course
- *Permissions for teacher.*
 - View of the course statistics of students progress and activity

The detailed accesses to the components and functionalities of the BI-DBS portal are going to be presented by the use case diagrams by students, who will be implementing them. An example of such work is Bc. Radoslav Hašek's master thesis [34] which focuses on analyses, designing and implementation of the tests module.

Implementation

Authentication is a process of identification and verification of the user's access to the application. Authorization is a process of granting or denying access to the application based on the authorized user's identity and the permissions for that identity. [35] In this chapter, I will describe the implementation of those two processes for the client side along with the role-based access control described in the third chapter.

My implementation uses the authentication microservice that was designed and described along with the general authorization flow for this service by Ing. Andrii Plyskach in his master thesis [1]. For the purpose of reducing security vulnerabilities, I will also describe a few adjustments to the functionalities of that microservice.

4.1 Used software

The technologies used for the development of the BI-DBS portal are described in section 1.3. In this section I will mention which of them I have used for the implementation part of this thesis and also describe the used libraries.

The UI is implemented in Vue.js [36] together with the Quasar framework [22] for creating good-looking components and Typescript [21] as a programming language.

4.1.1 Pinia.

Pinia is a reactive state management library. It allows to easily update data and react to changes in the application's state. Besides, it provides type safety and thus integrates well with Typescript. In my implementation I have used it for storing the state of users' authorization along with the user's identity information such as their role. [37]

4.1.2 Router

Vue Router is a routing library that provides mapping URLs to the components, allowing SPA to load pages based on the given URL dynamically. I have used it for managing access to the components based on the user's authentication state and identity. More about role-based access control is in section 4.4.1. [38]

4.1.3 Axios

Axios is a Javascript library for making HTTP requests from the browser of various request methods. It automatically transforms the response data to its response type and allows it to intercept requests and responses. I have used it for communication with the backend. [39]

4.2 Authentication and authorization

Authentication and authorization processes are implemented using the secure OAuth 2.0 protocol [2]. The whole procedure is based on communication and data exchange between four roles and components: the resource owner(user) the client app, the authorization server and the authorization microservice.

4.2.1 OAuth protocol

OAuth protocol is a modern industry-standard protocol for authorization. In general, the OAuth protocol enables users to provide a client with secure access to server resources and without a need for users to share their credentials directly with an application they want to access. OAuth is widely used by major companies to allow users to grant access to their resources to trusted third-party applications. [2] This protocol is also obviously suitable for university applications and the BI-DBS portal in particular.

4.2.2 Tokens

Token in this context is a string that has some value for authentication and authorization processes. In the implementation, the client gets the authorization code from the authorization server and exchanges it for access and refresh tokens with the server.

4.2.2.1 Access token

The OAuth protocol uses an access token to represent the authorization status in the application. The access token is a short-time living token used to verify the user's access to the application. [40] An access token is sent in every HTTP request made to the server. Thus the server can verify if the token is

valid and then complete a request. It can be used for communication with the application server until that token expires. The expiration time is set to one hour.

The implementation uses a JWT token type for creating the access token.

JWT token. JSON web token(JWT) is a token type standard that allows securely transmitting JSON objects. It consists of three parts: header which specifies the algorithm used to sign the token, payload containing the data being sent and signature that validates the token's integrity and assures it has not been modified. [41]

The image shows a screenshot of the jwt.io website. On the left, under the heading "Encoded" with the subtext "PASTE A TOKEN HERE", there is a text area containing a JWT token: `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MzY1MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c`. On the right, under the heading "Decoded" with the subtext "EDIT THE PAYLOAD AND SECRET", there is a form showing the decoded components of the token. The "HEADER: ALGORITHM & TOKEN TYPE" section contains a JSON object: `{ "alg": "HS256", "typ": "JWT" }`. The "PAYLOAD: DATA" section contains a JSON object: `{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }`. The "VERIFY SIGNATURE" section shows the HMACSHA256 function: `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), your-256-bit-secret)`. There is an input field for the secret and a checkbox for "secret base64 encoded".

Figure 4.1: Example of encoded and decoded JWT token from jwt.io. [41]

4.2.2.2 Refresh token

Refresh token is a type of token used for requesting a new access token when the current one expires. It is a long-time living token because it is meant to live longer than an access token due to its existence purpose. With the help of a refresh token user does not need repeatedly log in every hour which helps to improve user experience. [42]

4.2.3 Authorization server

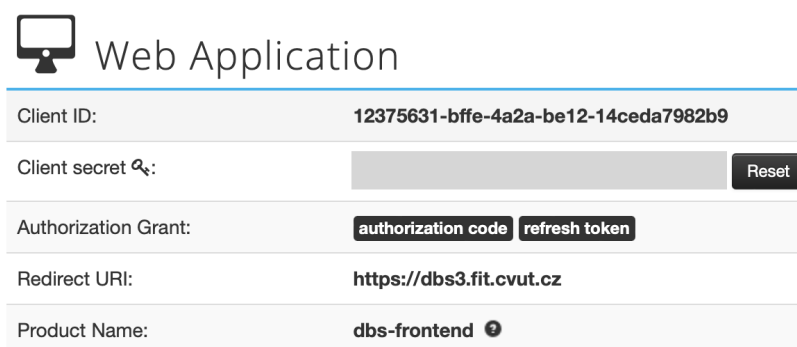
The authorization server is a key component of the authorization and authentication processes based on the OAuth 2.0 protocol. For the authentication and authorization in the BI-DBS portal, we use the faculty's authorization server called Zuul OAAS. It is open-source and available for anybody from CTU. [43] Zuul OAAS provides three endpoints:

- *Authorization endpoint: /oauth/authorize.* It is used for the authentication and authorization processes of a user. It displays the login form for a user and after the submission of that form, it validates credentials and in a case of success does a redirect back to the BI-DBS client app server with the authorization code.
- *Token Endpoint: /oauth/token.* This endpoint provides us with two important functionalities:
 - Exchanging authorization code from the authorization endpoint for access and refresh tokens.
 - Generating new access token by accepting the refresh token.
- *Check Token Endpoint: /oauth/check_token.* For controlling the validity of the token, the authorization server provides this endpoint which checks the token for being valid.

With the use of these endpoints we have constructed the authentication and authorization processes, which are more deeply described in 4.2.5 and shown in Figure 4.3.

4.2.4 Apps manager

To communicate with the authorization server, the application must be registered in the Apps Manager [44]. For further communication with the server we will need three parameters: client id, client secret and redirect URL. The first two parameters are generated by the app manager. These are simplified login and password for our application. But the redirect URL can be set to the URL we want, this URL will be used for the redirect back to our application after the successful authorization of a user. These parameters are used as a part of the HTTP requests, that frontend and backend send to the authorization server.



The screenshot shows a web application configuration page. At the top, there is a monitor icon and the title 'Web Application'. Below this, there are several rows of configuration data:



Client ID:	12375631-bffe-4a2a-be12-14ceda7982b9
Client secret  :	<input type="password"/> Reset
Authorization Grant:	authorization code refresh token
Redirect URI:	https://dbs3.fit.cvut.cz
Product Name:	dbs-frontend 

Figure 4.2: The example of the application registry in the App Manager [44]

4.2.5 Authentication and authorization flow

For better visualization I have provided the implicit flow of the implemented services and a sequential diagram in Figure 4.3.

Explicit flow:

1. Unauthorized user is trying to access the application and access validation redirects a user to the login page.
2. By clicking on the login button user gets transferred to the page provided by the authorization server with a form for submitting credentials.
3. After submitting credentials authorization server redirects the user back to the BI-DBS client with an authorization code which the client exchanges with the backend for the JWT access and refresh tokens.
4. Finally the user is redirected to the page they wanted to access.

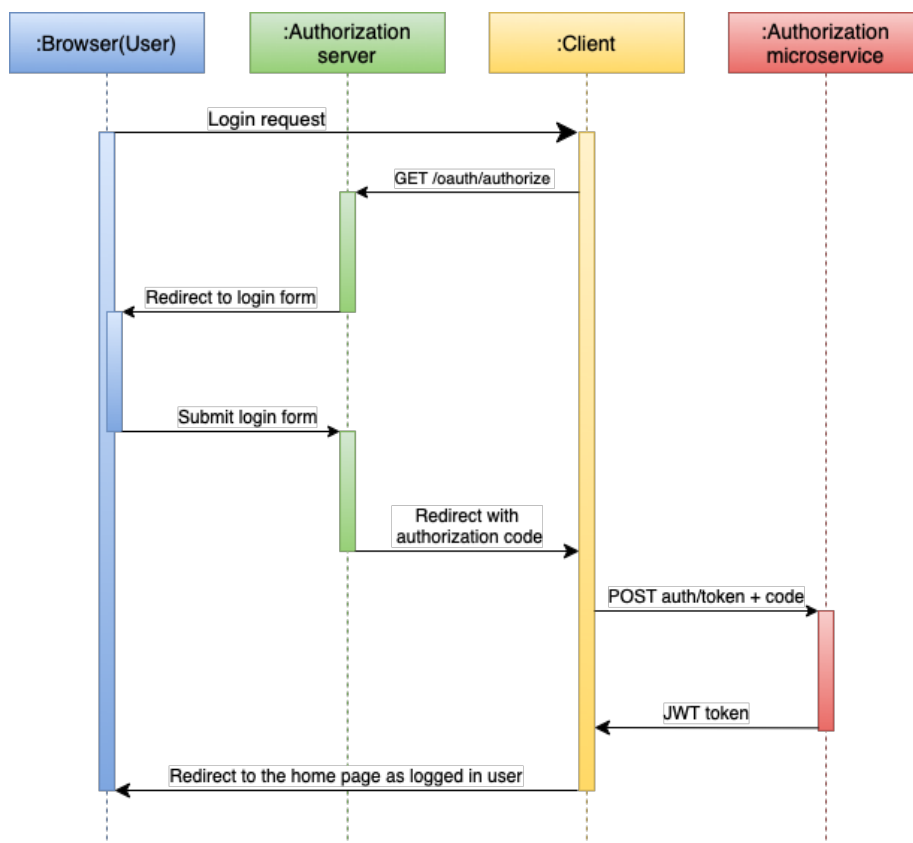


Figure 4.3: Sequential diagram of generating an access token

4.2.6 Refresh token flow

Another very important part of the implementation is refreshing the access token with the usage of the refresh token. A user does not know about that process and does not participate in it. This process replaces the constant logging in by a user, which improves security since a user does not need to enter credentials repeatedly and also does not annoy or interrupt a user. The explicit flow and sequential diagram of the token refreshing process in Figure 4.4 are provided for showing the implementation details.

Explicit flow:

1. A user who has already been authorized in the application is trying to make a request. However, the access token has expired and cannot be used anymore for the requests.
2. The client detects that the access token has expired and sends a request to the authorization microservice for a new access token in exchange for the current access token and the refresh token.
3. After receiving a new access token client completes the request for a user with the usage of that access token.

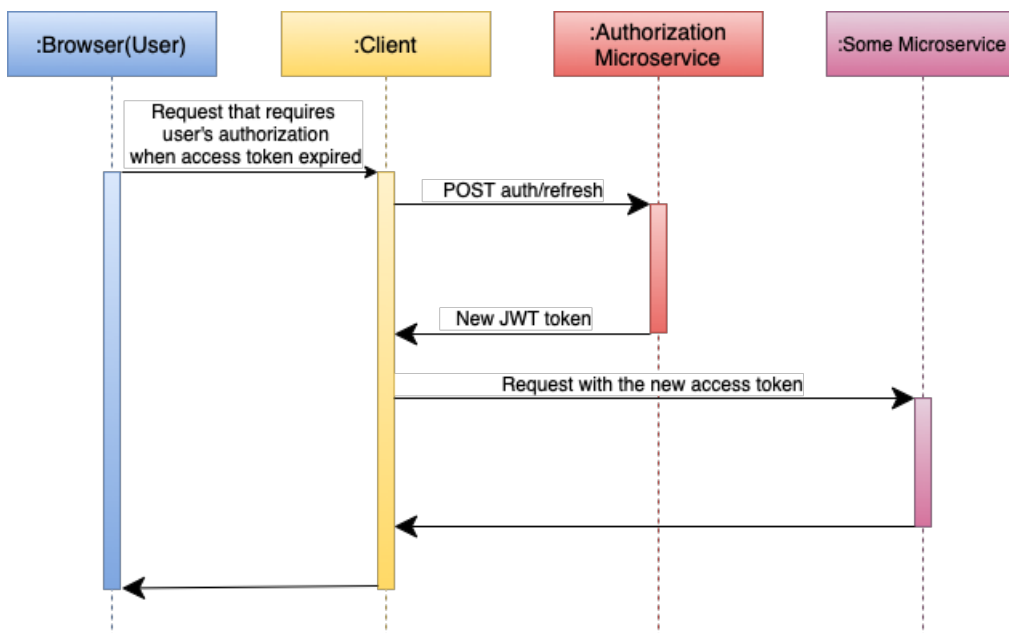


Figure 4.4: Sequential diagram of refresh an access token using refresh token

4.3 Security

Needles to say that the development process comes together with analyzing security vulnerabilities and implementing the solution to reduce them. My implementation contains two points where the setup plays a big role: the solution for storing sensitive data such as access and refresh tokens and the configuration of communication between the client and server sides.

4.3.1 Tokens storage

Access and refresh tokens are both very sensitive pieces of data. In the case of successfully stealing the refresh token, an attacker will get long-term access to the application, or a short term in the case with the access token. That is why it is important to store these tokens as securely as it is possible.

Most common attacks: [45]

- *Cross-site scripting (XSS) attacks.* This is a type of attack where an attacker injects malicious scripts into a web page viewed by other users. It basically happens when an application trusts a user and the content, which can be inserted by them.
- *Cross-site request forgery (CSRF) attacks.* In general, this attack can be described as tricking a user into unintentionally performing some action. In this way, attackers just use the user's account to perform some harmful requests.

Types of browser storages: [46]

- *Local storage.* Local storage is web storage that allows the saving of key-value pairs of data. It provides the stored data persistence across the page reloading and can fit up to 5MB of data. Content from the local storage cannot be automatically sent and it makes it prone to CSRF attacks. However, it is not considered secure storage for keeping sensitive data due to the vulnerability to XSS attacks.
- *Session storage.* Session storage is client-base storage available by the browser which is very similar to the local storage. It has the same memory limit and vulnerabilities. But in addition to that it does not provide the persistence of data, which is a disadvantage to the user experience due to the need for repeated authentication.
- *Cookies.* Cookies are another way of storing data using a browser. Cookies can store much less data - up to 4KB, which might not always be enough for storing big tokens, but it is absolutely enough for my implementation. Cookies are vulnerable to both CSRF and XSS attacks, but

with a proper configuration of secure attributes they are less vulnerable than local or session storage. Cookies can also be configured to be automatically sent with every HTTP request or some certain request.

4.3.1.1 Solution for the refresh token

The perfect solution for storing sensitive data in a browser does not exist and in order to keep the data secure there should be also implemented other vulnerability-reducing features like input data validation, escaping and others. However, with the proper use of security attributes, cookies are much more likely to mitigate those attacks. [47] Moreover, using cookies for storing tokens is recommended by the OWASP [48] community due to their secure configuration options. [49]

I have come to the solution of storing the refresh token, which is the most sensitive token requiring persistence across page reloads, in httpOnly cookies. HttpOnly cookies must be set on the server side as they are not accessible for JavaScript. Therefore, I had to adjust the implementation of sending and receiving the refresh token in the authentication microservice from the body of requests to cookies. Listing 4.1 shows the configuration of security attributes for the refresh token cookie.

```
1 public function setRefreshTokenCookie(View $view, JWTToken $token
2     ): View
3 {
4     $cookie = new Cookie(
5         name: 'refresh_token',
6         value: $token->getRefreshedToken(),
7         path: '/refresh-token',
8         secure: true,
9         httpOnly: true,
10        sameSite: 'Strict'
11    );
12    $view->getResponse()->headers->setCookie($cookie);
13    return $view;
14 }
```

Listing 4.1: Cookies setting on server

Security attributes: [50]

- *Path*. URL path to which the cookie will be automatically sent.
- *Secure*. This flag specifies that the cookie can be sent only through HTTPS encrypted connections.
- *HttpOnly*. The httpOnly flag is created to mitigate XSS attacks. With this flag, cookies can not be accessed by the Javascript.

- *SameSite*. The strict value of that parameter means that the cookie is only sent if you are on the site that the cookie is set for. SameSite attribute is introduced for protection against CSRF.

4.3.1.2 Solution for the access token

The access token is represented as a JWT token, it contains important information about the expiration time of the token and the user's identity. It cannot be stored in the same way as a refresh token as a client needs to have access to that token directly.

Another way of storing sensitive data which is considered more secure than local and session storage and cookies is storing it in memory (in a variable). This way is considered more secure because it is more prone to XSS attacks. The disadvantage of that method is that it doesn't provide persistence, but in a case with an access token it is a resolvable challenge.

First of all, I have decided to store the access token and parsed information about a user in the Pinia store, which is more secure than global variables. The visualization of a Pinia store is shown in Listing 4.2.

```

1 export const useUserStore = defineStore('auth', {
2   state: () => ({
3     isLoggedIn: false,
4     user: emptyUser,
5     token: '',
6     previousPage: '/'
7   }),
8   actions: {
9     login(user: User, token: string) {
10      this.isLoggedIn = true
11      this.user = user
12      this.token = token
13    },
14    logout() {
15      this.isLoggedIn = false
16      this.user = emptyUser
17      this.token = ''
18    },
19    setPreviousPage(page: string) {
20      this.previousPage = page
21    }
22  }
23 })

```

Listing 4.2: Pinia store configuration for user states

The fact that Pinia store is flexible and simple to use is important, as the stored information is frequently used for making requests, verifying accesses on the frontend and other functionalities.

Pinia storage does not provide data persistence on its own, but with the use of

plugins it can enable this functionality with the use of local or session storage. Surely it is not suitable for the implementation as I am trying to use as most secure ways to store data as it is possible. Therefore the users' data will be lost together with an access token after the page reloads.

The BI-DBS is a SPA application, meaning that page reloads are not a part of the usual functioning of the web page. However, for the case when a user does the page reload and store resets to the default values I have implemented the silent sign-in. Silent sign-in is shortened way of the authentication and authorization process described in 4.2.5. The user does not participate in this process, the client just quickly gets the access token as the authorization server already knows the user and does not require to provide the credentials before the access token expires.

4.3.2 Communication with backend

Backend plays a huge role in the application and without it the frontend features I have implemented would be non-functional. The backend and the frontend in the new BI-DBS portal are two separate units that communicate through HTTP requests. It is essential to correctly configure communication between them.

4.3.2.1 Communication

Frontend sends HTTP requests [51] to the backend using promised-based HTTP client Axios. Axios offers a wide range of configurations of the request. Starting from choosing a request type, setting an URL and body to interceptors for request and response. The example of configuring the Axios client is shown in Listing 4.3.

4.3.2.2 HTTP headers

HTTP headers are vital components of communication between client and server. They provide detailed control over how requests and responses are managed, which leads to improvements in such aspects as security, performance, and user experience. Headers provide additional information about the HTTP request or response including security-related information like Cross-Origin Resource Sharing(CORS) [52] and authorization [53]. While access allowance is set on the server side, the authorization is provided by the client.

Base configuration For the base configuration of headers on the client side I have sent an accepted content type to `application/json` which can automatically be serialized and parsed to objects by Axios. The next header I have configured is the authorization header to contain the access bearer token. It is necessary for getting the data from a backend when it comes

to different microservices from the authorization one. Other microservices require authentication and must be provided with the access token in every request to validate the user and their access for making that request, which is implemented for security reasons. Header configuration is visualized in Listing 4.3.

```
1 const BASE_URL = import.meta.env.VITE_BASE_URL
2 export const axiosClient = axios.create({ baseURL: BASE_URL })
3
4 axiosClient.interceptors.request.use((config) => {
5     return verifyAccess().then((token) => {
6         config.headers.Authorization = 'Bearer ' + token
7         config.headers.Accept = 'application/json'
8         return config
9     })
10 }, (error) => {
11     return Promise.reject(error)
12 })
```

Listing 4.3: Base Axios configuration

4.4 Access control

The BI-DBS portal uses role-based access control [31]. In the chapter 3 I have designed the roles system and described their permissions. With the use of that system I have implemented the regulation of displaying the components and controlling access by roles and also verification of users' authorization for accessing components and making requests.

4.4.1 Role-based access control

Vue Router is first of all responsible for displaying the components by the given path. However, it has multiple other features which I have used for my implementation.

Firstly it allows dynamically setting of different parameters in meta object, you can see the example of a component configuration and meta object in Listing 4.4.

Meta parameters for the component:

- *requireAuth*. This flag simply tells if authorization is required for accessing the component.
- *allowAccess*. This flag specifies the role that has permission to that component.

4. IMPLEMENTATION

- *sidebar*. Sidebar parameter tells the name of the configuration for navigation sidebar items.
- *showSideBar*. This attribute is defining whether the navigation sidebar should be displayed.
- *showTopBar*. This flag is used for the configuration of the top navigation bar and it makes clear if that bar should be displayed for that component.

```
1 {
2   path: '/administration',
3   component: ImportSemester,
4   meta: {
5     requireAuth: true,
6     allowAccess: Role.GUARANTOR,
7     sidebar: 'administration',
8     showSideBar: true,
9     showTopBar: true
10  }
11 }
```

Listing 4.4: The example of component configuration in Router

First two meta parameters are used for the access control, which is implemented as Router configuration. The Router allows to set a set of functionalities before each routing which I have used for the verification of a user's access to the component. This validation is presented in Listing 4.5.

```
1 export function configureGuard() {
2   Router.beforeEach((to, from, next) => {
3     if (to.meta.requireAuth === false) {
4       next()
5     } else if (verifyAccess(to, from)) {
6       next()
7     }
8   })
9 }
10
11 function verifyAccess(to: RouteLocationNormalized, from:
12   RouteLocationNormalized) {
13   const userStore = useUserStore()
14   userStore.setPreviousPage(from.path)
15
16   if (!userStore.isLoggedIn) {
17     login()
18   } else {
19     if (userStore.user.exp_time < new Date()) {
20       refreshToken()
21     } else {
22       return verifyRole(to)
23     }
24   }
25 }
```

```
23     }
24     return false
25 }
26
27 function verifyRole(to: RouteLocationNormalized) {
28     if (to.meta.allowAccess == undefined || hasRole(to.meta.
29         allowAccess as Role)) {
30         return true
31     } else {
32         Router.push('/error403')
33         return false
34     }
35 }
```

Listing 4.5: Validation of access by role

Firstly this configuration checks the users' authorization status if the authorization is required for the component. Secondly, it verifies that a user does have permission to access this component. In a case user is trying to access some component directly without having access they will be redirected to the error page with an option to get to the previous page, the error component is shown in figure 4.5.

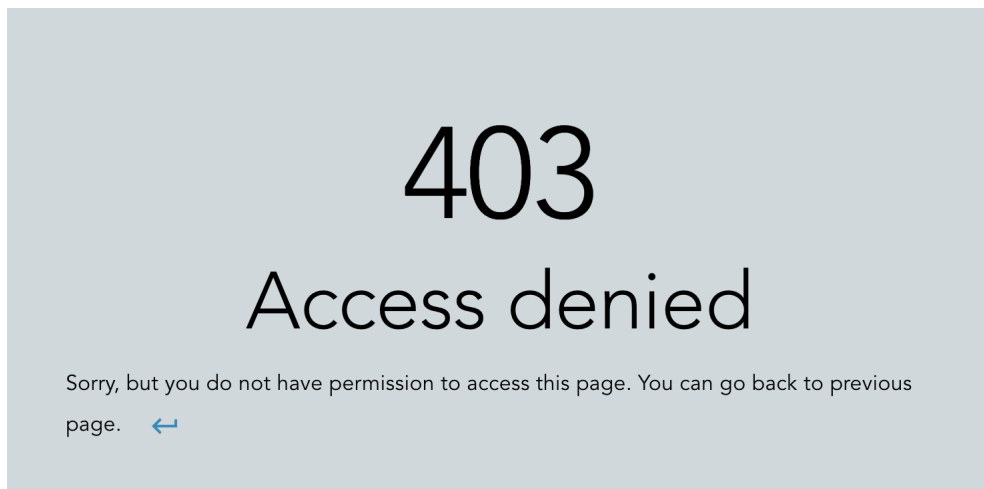


Figure 4.5: Access denied component

Display control of components. Another example of role-based access control benefits and usage is using it for the display control of UI components. A developer can easily control displaying of any component by setting the required role for it and using the `hasRole()` function shown in Listing 4.6. The code Listing 4.7. shows exactly the case like that. In the top navigation bar configuration I have set the required role for displaying the administration

4. IMPLEMENTATION

component and used a filter for displaying only the components user has access to.

```
1 export function hasRole(role: RoleTOBE) : boolean {
2   switch (role) {
3     case RoleTOBE.GUARANTOR:
4       return isGuarantor()
5     case RoleTOBE.TEACHER:
6       return isStudent()
7     case RoleTOBE.STUDENT:
8       return isTeacher()
9     default:
10      return false
11   }
12 }
```

Listing 4.6: Role validation

```
1 function useTopBarConfig() {
2   return computed(() => {
3     const { t } = useI18n()
4     return {
5       tabs: [
6         { name: t('base.home'), path: '/' },
7         { name: t('admin.administration'), path: '/
administration' , allowAccess: Role.GUARANTOR },
8       ].filter(item => item.allowAccess == undefined ||
hasRole(item.allowAccess))
9     }
10  })
11 }
```

Listing 4.7: Filtering displayed components by role

The implementation of regulating accesses and displaying components is clear, flexible and easy to use. It is possible due to the role-based access control, which allows simplified permissions management. Moreover, such system decreases the risk of data leaking and provides good visibility over the application.

4.4.2 Authentication control for requests

Another important validation is verification of access for making requests to avoid the pointless calling of backend services when the user is not authorized or the access token has expired. Especially in the case when the access token has expired we need to detect it and execute the refresh process.

In 4.3.2.2 I have provided a code Listing 4.3 of the configuration of the Axios interceptors for the request. That configuration contains a `verifyAccess()` function which makes sure the user is authorized before making the request.

That function is illustrated in Listing 4.8.

```
1 function verifyAccess() : Promise<string>{
2   return new Promise((resolve, reject) => {
3     const userStore = useUserStore()
4
5     if(!userStore.isLoggedIn){
6       reject('Unauthorized')
7     } else if (userStore.user.exp_time < new Date()) {
8       Auth.refreshToken({
9         access_token: userStore.token
10      }).then((response) => {
11        processResponseToken(response.data.access_token)
12        resolve(userStore.token)
13      }).catch(() => {
14        reject('Unauthorized')
15      })
16    } else {
17      resolve(userStore.token)
18    }
19  })
20 }
```

Listing 4.8: Access verification for making requests

Automation – CI/CD

Automation is the base of DevOps methodology. My implementation of automation includes containerization, automated testing and deployment. With the use of DevOps adoption instruction from 2.4 I will describe all the used tools, the implementation of the automation itself and also provide documentation.

5.1 Used tools

First step of adopting the automation concept is choosing suitable tools for the planned automation of processes.

5.1.1 Docker

Docker is an open-source platform that allows to build, run and deploy applications quickly using virtualization of server hardware in containers. Containers are lightweight units created by the containerization of the application. The containerization provided by Docker is a technology of software packaging including code, libraries and other necessary dependencies and configurations for building the application.

Docker was already chosen as a tool for running the application and I have decided to also use it for the automation of building and development processes due to its flexibility, lightweight containers and speed. [54, 55]

5.1.2 GitLab

GitLab is a web service based on the Git version control system that is also a DevSecOps [56] platform with multiple functionalities. It allows to plan, track and manage issues, as well as manage automation processes using CI/CD pipelines. Besides it lets to schedule jobs, create merge requests, do code reviews and provides many other features. Its main benefit is that a software

development team can use GitLab instead of using many other tools as it combines many features. [57, 58]

GitLab is generally used for many school applications including the BI-DBS portal. Therefore it was the obvious choice to use it for the automation processes.

5.1.3 CloudFIT

CloudFIT is a platform for everyone from FIT CTU, which provides server management from usual web applications hosting to complex calculations and simulations. [59]

I have chosen CloudFIT for hosting the BI-DBS frontend, because the servers are managed by the faculty and faculty workers are open to consulting the server parameters. Besides, it is free of charge, has well-written documentation and is recommended for hosting school applications.

5.1.4 Buildah

Buildah is a tool for containerization which is compatible with for example Docker. I have decided to use Buildah because the faculty provided an image with Buildah for building docker containers, which is ready to use. Besides, from my own small research comparing it with for instance Docker in Docker(dind) it turned out to be the most stable and efficient variant. [60]

5.1.5 Nginx

Nginx is an open-source web server that can run in a docker container and allows numerous configuration options such as reverse proxy, load balancing, caching and others. The decision to use Nginx was made due to its simplicity in configuration. [61, 62]

5.1.6 Yarn

Yet Another Resource Negotiator(Yarn) is a JavaScript package manager which helps to manage project dependencies. It assists the application with managing packages, managing scripts and caching. Yarn was already in use in the BI-DBS frontend project, it has valuable benefits over other package managers like for instance parallel installation of packages and installation of packages without an internet connection. [63, 64]

5.2 Implementation

Second step of adopting the automation practices is setting up the processes using chosen tools.

5.2.1 Containerization

I have adopted containerization using Docker for use in CI/CD pipeline. The Dockerfile simply contains the instruction for building a Docker image. It is composed of specific commands which tell how to build the image. A Docker image is a read-only file containing a set of instructions. When these instructions are executed, a Docker container is created.

```
1 FROM node:latest as build
2 WORKDIR /app
3 COPY package*.json ./
4 RUN yarn
5 COPY ./ .
6 RUN yarn run build
7
8 FROM nginx
9 RUN mkdir /app
10 COPY --from=build /app/dist /app
11 COPY nginx.conf /etc/nginx/nginx.conf
```

Listing 5.1: Dockerfile

Dockerfile showed on Listing 5.1 contains installing dependencies on the fifth row and application bundling on the six's row using Yarn and also adding Nginx configuration from the `nginx.conf` file on the last row, which defines parameters like hostname, listen port and others.

5.2.2 CI/CD pipeline configuration

CI/CD pipeline is a set of steps that are executed when the pipeline was triggered.

Listing 5.2 shows all the stages of the CI/CD pipeline. The first two stages were configured by Ing. Oldřich Malec with the first setup of the frontend projects. The download stage installs all the dependencies required for the application to run. The codestyle stage provides linting of the source code. Other three stages I have implemented in this thesis.

```
1 stages:
2   - download
3   - codestyle
4   - test
5   - build
6   - deploy
```

Listing 5.2: Gitlab CI/CD stages

GitLab provides the visualization of the pipeline steps and allows to execute any steps manually as well as monitor the log of any job from the pipeline. Figure 5.1 shows how the passed pipeline looks in the BI-DBS portal frontend.

5. AUTOMATION – CI/CD

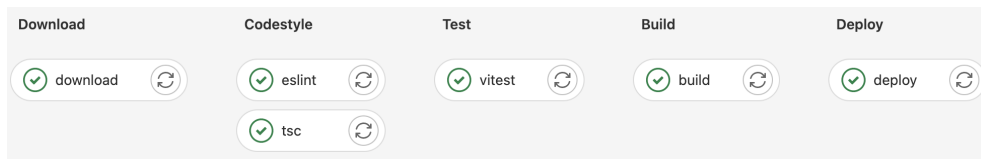


Figure 5.1: CI/CD pipeline in GitLab

The pipeline stages are executed from left to right and if any of the stages fails all the other ones will be skipped. It helps to avoid the pointless running of jobs because the stages are put in a way that they depend on the result of the previous ones.

5.2.2.1 Test

Test stages contain one job which I have named Vitest, because it executes unit tests implemented using Vitest framework which will be introduced in the chapter 6. For running the tests I have used Yarn, which provides the execution of the tests by running just one command which is added to the script section on line fifteen of the Listing 5.3.

```
1 .image_template: &image
2   image: $CI_REGISTRY/ict/images/alpine/ci:3.16
3   before_script:
4     - apk add -U nodejs yarn
5
6 .cache_pull_template: &cache
7   key: $CI_COMMIT_REF_SLUG
8   paths:
9     - node_modules/
10
11 vitest:
12   <<: *image
13   stage: test
14   script:
15     - yarn run test
16   cache:
17     <<: *cache
```

Listing 5.3: Test stage in the CI/CD pipeline

For the test execution setup in the pipeline I have used image and cache pull templates, which were already prepared and used for the download and codestyle stages.

5.2.2.2 Build

Build stage is focused on building an image and adding it to the GitLab container registry of the project. I have used the faculties image containing

buildah and buildah itself for the script which is shown in Listing 5.4. Command from row number seven will create the image and the next command from row number eight will upload it to the GitLab registry.

```
1 build:
2   image: $CI_REGISTRY/ict/images/buildah:v1
3   stage: build
4   variables:
5     IMAGE_TAG: $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_NAME
6   script:
7     - buildah build --squash --tag $IMAGE_TAG -f Dockerfile
8     - buildah push $IMAGE_TAG
9   only:
10    - master
```

Listing 5.4: Build stage in the CI/CD pipeline

The last two rows of the listing define the only branch for which this job will be executed. The BI-DBS frontend does not have a production version yet. Therefore I have decided to create the environment for deploying the application for testing during the development process. The changes are configured to be built and deployed only from a master branch.

Change in the deployment flow. In the current application the master branch is configured to be a production state branch, while the branch named devtest is used for deployment to the test environment. From my implementation there is a change that is going to be established to the new project, when a master will be to representing a state of a test environment and the deployment to the deployment to the production environment will be implemented by using tags for the master branch with the application version. This approach has several advantages for the BI-DBS frontend project.

- *Easier maintenance and flexibility.* Using just one branch reduces the complexity of merging branches and maintenance of multiple branches.
- *Simulation of a production.* The master branch state always represents a state which is very similar to the production state. It means that developers can see how their changes would behave in production. This implies a few next benefits.
 - It allows to thoroughly test the application before deploying to the production.
 - Provides a possibility to give faster and higher quality feedback on code and changes.

5.2.2.3 Deploy

When it comes to deploy stage, it means that the application went through all the previous pipeline stages and is ready to be deployed. The deployment process of the CI/CD pipeline is composed of the execution of the deployment script by the server. For the connection to the server GitLab uses CI/CD variables that allow securely storing sensitive data, which need to be used in the pipeline like for example deployment user showed on line number six of Listing 5.5.

```
1 deploy:
2   stage: deploy
3   dependencies:
4     - build
5   script:
6     - ssh -A "${DEPLOYER_IP_ADDRESS}" -l ${DEPLOYER_USER} 'cd ~/
7       dbs-frontend && git pull && cd ../.docker/server && bash
8       deploy-dbs-frontend.sh'
```

Listing 5.5: Deploy stage in the CI/CD pipeline

Deployment script. In a difference with a current application the deployment script is now being versioned and stored in the git remote repository instead of a server only. Therefore, before the script execution the server pulls all the changes from the remote repository.

The deployment script, which is shown in Listing 5.6 contains a few steps such as pulling the Docker image from the GitLab created in the build stage on row number seven, then running a container defined and configured in the `docker-compose.yaml` file showed in Listing 5.7. Then the script provides the "healthcheck" of the environment, which is a check for controlling the availability of the resource. And finally on the last two rows script starts showing the logging of the container and removes unused images from the local docker host for removing the disk space.

```
1 #!/bin/bash
2
3 set -e
4
5 HEALTHCHECK_URL="https://dbs3.fit.cvut.cz"
6
7 docker compose pull
8 docker compose up -d
9
10 attempt_counter=0
11 max_attempts=30
12
```

```
13 until $(curl --output /dev/null --silent --head --fail "${HEALTHCHECK_URL}")
14 do
15     if [ "${attempt_counter}" -eq "${max_attempts}" ]
16     then
17         echo "Max attempts reached"
18         docker logs dbs-microservices-frontend
19         exit 1
20     fi
21
22     attempt_counter=$((attempt_counter+1))
23     echo "Waiting for URL ${HEALTHCHECK_URL}... (${attempt_counter}
24        }/${max_attempts})"
25     sleep 2
26 done
27 docker logs dbs-microservices-frontend
28 docker image prune --force
```

Listing 5.6: Deployment script

Listing 5.7 shows the file which is used for the configuration and run of the container. Firstly, it defines the image which will be downloaded from GitLab. Secondly, it defines a network, that allows communication with the services from the same network. Thirdly it tells the service to restart automatically in case it stops for any reason. Then it specifies the container and host names as well as port mapping. Finally it gives a label, which can be used for the service identification as a microservice of the application.

```
1 version: '3.8'
2 services:
3     dbs-frontend:
4         image: 'gitlab.fit.cvut.cz:5000/dbs/dbs-frontend:master'
5         networks:
6             - internal
7         restart: always
8         container_name: 'dbs-microservices-frontend'
9         hostname: 'dbs3.fit.cvut.cz.internal'
10        ports:
11            - '8080:80'
12        labels:
13            cz.cvut.fit.dbs.microservice: 'Frontend'
14 networks:
15     internal:
16         external: true
```

Listing 5.7: docker-compose.yml

5.3 Documentation

The last step from the DevOps practices adoption instruction is the documentation for the team. I have written the documentation shown in Figure 5.2 in the new documentation project created by Dana Suchomelová, which is used for frontend project documentation. It contains all the sources with needed information, instructions on how to get access to the server and of course the file containing the 5.1 and 5.2 sections of this chapter which have all the necessary information about the pipeline setup and used technologies.

DevOps

Test environment: dbs3.fit.cvut.cz

CI/CD

Branch: [master](#)

CI/CD file: [gitlab-ci.yml](#)

GitLab CI/CD (automation): [FIT docs](#)

Building images: [FIT docs](#)

Deployment script and other docker configuration files for server: in `./.docker/server` of a master branch

Documentation of the implementation is in [thesis by Volha Chukava](#) Or [Redmine](#)

CloudFIT server

maintainer - *Jiri Hunka*, **administrator** - *Volha Chukava*

Server - [CloudFIT docs](#)

Access: For getting access to the server or deployment scripts contact the maintainer or the administrator

Figure 5.2: Documentation of server and CI/CD management

Testing

I have provided two types of tests. First of them is unit tests, created specially for the CI pipeline, which tests the main functionalities like managing the data and processing of authentication and authorization processes. Second type of tests is manual functional tests. I have created test scenarios for manual testing of all the implemented features, including the cases for different roles.

6.1 Tests for CI

Providing a set of regression tests for the CI pipeline is clearly good practice since they will be executed with every change provided to the remote repository and make sure that if any of those changes will affect the functionality of the tested code the pipeline will fail and the code with defects will not be deployed. Moreover, tests will tell the developer exactly what unit was affected by their changes and require a fix.

6.1.1 Vitest

Vite is a frontend build tool used for the BI-DBS frontend. Vitest is a unit test framework built on top of Vite. It is a suitable tool for implementing tests in the project like the BI-DBS frontend which uses Vite. Vitest cares a lot about performance and speed, which makes it a good choice for tests that will be a part of CI pipeline. [65]

6.1.2 Tests

The authorization module is composed of four main services which I have implemented for authentication and authorization processes including additional features like a refreshing of token and logging out. Therefore I have created unit tests for the functionalities of those services.

Generating the access token service. The process of getting the access token enables the application to log in the user on the frontend after an authorization server has successfully authorized the user and provided the client with the authorization code. This service is focused on exchanging that code for an access token with the backend and handling the response from the server.

Test cases for the token generating service

- *Positive scenario.*
 - *Conditions:* A valid token is successfully returned in the valid response.
 - *Expected result:* User was successfully authorized.
- *Negative scenario.*
 - *Conditions:* An invalid token is successfully returned in the valid response.
 - *Expected result.* The errors from parsing the token are successfully handled and the user was not authorized.
- *Negative scenario.*
 - *Conditions:* The request for getting the access token fails.
 - *Expected result:* The errors from the request are successfully handled and the user was not authorized.

Refreshing the access token service. Refreshing process prolongs the authorized status for a user without the need of submitting credentials, which is implemented by exchanging the refresh token and current access token for the new access token with the backend.

Test cases for the token refreshing service

- *Positive scenario.*
 - *Conditions:* User is authorized and the request for refreshing the access token succeeded and the server returned a new valid access token.
 - *Expected result:* The authorization time for a user was prolonged to the expiration time of the newly received access token.
- *Negative scenario.*

- *Conditions:* User is authorized and the request for refreshing the access token succeeded but returned an invalid token.
- *Expected result:* The errors from parsing the token are successfully handled and the user was logged out.
- *Negative scenario.*
 - *Conditions:* The request for refreshing the access token fails.
 - *Expected result:* The errors from the request are successfully handled and the user was not logged out.

Rejecting the access token. The process of logging out is implemented as a rejection of a token which includes removing all of the user’s data from the user’s store along with permissions to access the components which require authorization.

Test cases for the token rejecting service

- *Positive scenario.*
 - *Conditions:* User is authorized and the request for rejecting the access token succeeded.
 - *Expected result:* User was successfully logged out.

Negative scenario test for the rejection of the token does not have to be provided as the service behavior as for the positive scenario.

Authorization service. The services described above also need some additional functionalities like parsing the data from the request and preparing the URL for communication with the authorization server which are implemented in the authorization service with both positive and negative scenarios.

Test cases for the authorization service

- *Positive scenario.*
 - *Conditions:* The frontend calls the function to create an URL to initiate an authorization process by sending a request to the authorization server,
 - *Expected result:* The URL is valid.
- *Positive scenario.* This scenario is used for the implementation of four tests for each user’s role: admin, guarantor, teacher and student

- *Conditions:* The frontend calls the function to process the access JWT token it got from the server for authorization of a user with a certain role.
- *Expected result.* Successfully parsed the token and the data in the user store matches the user’s identity sent in a token including the role.

The results of unit testing. Frankly speaking, in the beginning I underestimated the importance of the unit tests implementation as I was very confident about my code as it was structured and clear for me to work correctly. Moreover, multiple times I have successfully tested all the functionalities manually myself.

Unexpectedly, The implementation of the unit tests process helped me to uncover and correct some weaknesses in the implementation of some functions. But most importantly, thanks to the creation of tests with also negative scenarios I have found some unhandled errors and was able to test the implementation in the various conditions.

The full set of tests is added to the CI pipeline and executed by every change pushed to the repository. The duration of tests as you can see in Figure 6.1 is less than four seconds, which is a remarkably good result.

```
$ yarn run test
yarn run v1.22.19
$ vitest
RUN v0.30.1 /builds/dbs/dbs-frontend
✓ test/auth/authorizationService.test.ts (5 tests) 45ms
✓ test/auth/generateTokenService.test.ts (3 tests) 67ms
✓ test/auth/refreshTokenService.test.ts (3 tests) 71ms
✓ test/auth/rejectTokenService.test.ts (1 test) 16ms
Test Files 4 passed (4)
Tests 12 passed (12)
Start at 17:49:16
Duration 3.53s (transform 151ms, setup 1ms, collect 2.86s, tests 199ms, environment 2.54s, prepare 591ms)
Done in 4.60s.
```

Figure 6.1: CI tests

6.2 Manual testing

I have asked four people to do the manual user acceptance tests of the functionalities I have implemented. The implementation does not contain many

UI components, but is focused on the proper functioning. As it was very important for me to make sure, that I get all the features tested I have offered each tester a scenario, which is shown in Table 6.1. Moreover, I have created a list of the requirements shown below, which they also must use for testing the features in random order and also estimate the look for a few UI components. Besides, I have defined the roles that testers should use for testing and the conditions they need to establish.

Roles. The test must be provided for the role I have assigned to the tester for checking the correctness of the access control system. Each tester has one of the next four roles assigned: admin, guarantor, teacher, and student. All testers have different roles assigned.

Conditions.

- Client and Server applications are running.
- Course was imported, the user's username exists in the database for that course and has an assigned role for the imported course.
- The expiration time of the access token is recommended to be mocked for five minutes.

Requirements for the application.

- No access to the application for non-authorized users.
- Clear and fast login procedure.
- Access control responds to the role permissions.
- Application allows persistent authorization across the page reloads.
- Application does not require repeated authorization after the access token expiration (five minutes in a case of a mocked expiration time instructed in the conditions section or sixty minutes if not mocked).
- Logout procedure includes the modal window for submitting the option to log out.
- Logout process is clear and fast.
- Login and logout UI components are good-looking.

Offered scenario.

	Test Step:	Expected result:
1.	Try to access the home page of the application.	Redirection to the login page.
2.	Clicks on the login button.	Redirected to the login form.
3.	Submitted login form with valid credentials.	Redirected to the home page.
4.	Control all the displayed navigation tabs to be correct for your role.	Tabs are displayed correctly.
5.	Refresh the page.	Page was successfully refreshed, user still has access to the application.
6.	Leave a browser page with an application open for 5 minutes and then click to one of the tabs from the navigation bar	Component loaded by the path of navigation tab.
7.	Click the logout button.	The modal window displayed and requires the submission to log out.
8.	Submit.	Redirected to the login page.

Table 6.1: Scenario for manual testing.

Results.

- *Tester 1. Anonymous with the admin role*
 - *Scenario:* passed
 - *Requirements:* passed
 - *Feedback:* "Page reload takes longer from time to time. Access control works good as well as login and logout."
- *Tester 2. Bc. Darya Litvinenka with the guarantor role*
 - *Scenario:* passed
 - *Requirements:* passed
 - *Feedback:* "Everything works fine. "
- *Tester 3. Bc. Adam Vonašek with the teacher role*
 - *Scenario:* passed
 - *Requirements:* passed

- *Feedback:* "I didn't find any problem and would rate the experience usage as positive. The only thing I'm not sure about is the access control for my role as the application doesn't have many tabs and functionalities yet."
- *Tester 4. Anonymous with the student role*
 - *Scenario:* passed
 - *Requirements:* passed
 - *Feedback:* "The speed of the login was not very fast. Otherwise i didnt have a problem with anything."

Conclusion

First of the thesis goals was contributing to the project by improving the development efficiency and maintenance, which was achieved by introducing the DevOps methodology and also automating such processes as testing and development. The next goal was defined as adjusting the project to make it more structured and secure. Therefore, I have analyzed the current access management structure and came to the conclusion that it is unnecessarily too complicated. I have managed to logically simplify it and provide a design of a new access management system. Moreover, based on the clear permissions system I have strengthened the security by providing validation before every user's request. Finally, I planned to implement the authentication and authorization features. These functionalities were created using the OAuth 2.0 protocol and authorization microservice. Furthermore, I have reduced security vulnerabilities by providing a suitable way of storing sensitive data. In my opinion, this thesis will definitely be useful for developers and managers of the BI-DBS portal. It can be used as a part of introduction materials to DevOps methodology for new students before getting to the application development as well as the documentation of the access management system for managers and developers.

From the analysis of the current and planned application state, I have outlined three challenges, which we are facing in the new application state due to the microservices architecture. I have provided the solutions for two of them. The deployment problem was resolved by automation of this process. Possible security vulnerabilities were decreased by creating an access management system. The testing challenge still remains unresolved. The application needs a strong set of integration tests. Ideally, they should be a part of CI/CD pipeline to reduce the probability of deploying errors. This is an idea for further improvement which can become a topic for a thesis or a set of tasks for the teams of BI-DBS developers.

An integral part of working on the development of the BI-DBS portal is working in a team. I am endlessly grateful for the collaboration with such amazing

CONCLUSION

and skillful people and the experience I have gained. Thanks to this thesis and the BI-DBS project, I have learned a lot about application security and efficient development. Additionally, through the development of various features, I have gained experience working with modern technologies such as Vue.js, TypeScript, Pinia, and others.

Sources

1. PLYSKACH, Ing. Andrii. *Modernization and Migration of DBS portal*. Master thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.
2. *OAuth 2.0 - OAuth* [online]. 2023. [visited on 2023-04-23]. Available from: <https://oauth.net/2/>.
3. MALEC, Ing. Oldřich. *Project and infrastructure management of BI-DBS teaching portal*. Bachelor thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.
4. MIKE, Potel. *MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java* [online]. 1996. [visited on 2023-04-04]. Available from: <https://www.wildcrest.com/Potel/Portfolio/mvp.pdf>.
5. HARRIS, Chandler. *Microservices vs. monolithic architecture* [online]. 1996. [visited on 2023-04-08]. Available from: <https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith>.
6. *What is PHP?* [online]. 2023. [visited on 2023-04-05]. Available from: <https://www.php.net/manual/en/intro-what-is.php>.
7. *HTML: HyperText Markup Language* [online]. 2023. [visited on 2023-04-06]. Available from: <https://developer.mozilla.org/en-US/docs/Web/HTML>.
8. *Doctrine* [online]. 2022. [visited on 2023-04-05]. Available from: <https://github.com/doctrine/orm>.
9. *Nette* [online]. 2023. [visited on 2023-04-05]. Available from: <https://nette.org/en/>.
10. *Latte* [online]. 2023. [visited on 2023-04-07]. Available from: <https://latte.nette.org/en/>.

11. *AdminLTE* [online]. 2022. [visited on 2023-04-08]. Available from: <https://github.com/ColorlibHQ/AdminLTE>.
12. *The Progressive JavaScript Framework. Vue 2* [online]. 2023. [visited on 2023-04-07]. Available from: <https://v2.vuejs.org/>.
13. ZAKELŠEK, Haidi. *Options API vs. Composition API* [online]. 2022. [visited on 2023-04-08]. Available from: <https://medium.com/codex/options-api-vs-composition-api-4a745fb8610>.
14. *JavaScript* [online]. 2023. [visited on 2023-04-08]. Available from: <https://en.wikipedia.org/wiki/JavaScript>.
15. *Webpack concepts* [online]. 2023. [visited on 2023-04-06]. Available from: <https://webpack.js.org/concepts/>.
16. *PHP OOP - Object-Oriented Programming in PHP* [online]. 2022. [visited on 2023-04-08]. Available from: <https://www.phptutorial.net/php-oop/>.
17. *What is Symfony* [online]. 2023. [visited on 2023-04-06]. Available from: <https://symfony.com/what-is-symfony>.
18. *What's new in Vue 3 — a roundup* [online]. 2023. [visited on 2023-04-08]. Available from: <https://medium.com/front-end-weekly/whats-new-with-vue3-5b6562d3898b>.
19. *Composition API FAQ — Vue.js* [online]. 2023. [visited on 2023-04-08]. Available from: <https://vuejs.org/guide/extras/composition-api-faq.html>.
20. *Why Vite* [online]. 2023. [visited on 2023-04-08]. Available from: <https://vitejs.dev/guide/why.html>.
21. *TypeScript is JavaScript with syntax for types.* [online]. 2023. [visited on 2023-04-08]. Available from: <https://www.typescriptlang.org>.
22. *Why Quasar?* [online]. 2015. [visited on 2023-04-08]. Available from: <https://quasar.dev/introduction-to-quasar>.
23. *Pinia* [online]. 2023. [visited on 2023-04-09]. Available from: <https://en.wikipedia.org/wiki/Pinia>.
24. *Microservices and DevOps: Better together* [online]. 2023. [visited on 2023-04-15]. Available from: <https://www.mulesoft.com/resources/api/microservices-devops-better-together>.
25. *DEvOps* [online]. 2023. [visited on 2023-04-11]. Available from: <https://en.wikipedia.org/wiki/DevOps>.
26. *What Is DevOps?* [online]. 2023. [visited on 2023-04-11]. Available from: <https://www.atlassian.com/devops>.

27. *6 Principles of DevOps – DevOps Agile Skills Association (DASA)* [online]. 2023. [visited on 2023-04-12]. Available from: <https://www.devopsagileskills.org/dasa-devops-principles>.
28. *7 Principles of DevOps for Successful Development Teams* [online]. 2021. [visited on 2023-04-15]. Available from: <https://blog.hubspot.com/website/devops-principles>.
29. *DevOps: Principles, Practices, and DevOps Engineer Role* [online]. 2021. [visited on 2023-04-15]. Available from: <https://www.altexsoft.com/blog/engineering/devops-principles-practices-and-devops-engineer-role/>.
30. *DevOps: Principles, Practices, and DevOps Engineer Role* [online]. 2023. [visited on 2023-04-16]. Available from: <https://appinventiv.com/blog/devops-adoption-and-implementation/>.
31. *Role-Based Access Control* [online]. 2023. [visited on 2023-05-02]. Available from: <https://auth0.com/docs/manage-users/access-control/rbac>.
32. *Course - KOSapi* [online]. 2015. [visited on 2023-04-25]. Available from: <https://kosapi.fit.cvut.cz/projects/kosapi/wiki/Course>.
33. *TeacherRole KOSapi* [online]. 2015. [visited on 2023-05-02]. Available from: <https://kosapi.fit.cvut.cz/projects/kosapi/wiki/TeacherRole>.
34. HAŠEK, Bc. Radoslav. *db.s.fit.cvut.cz – tests*. Master thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.
35. *Authentication vs. authorization* [online]. 2023. [visited on 2023-05-02]. Available from: <https://auth0.com/docs/get-started/identity-fundamentals/authentication-and-authorization>.
36. *The Progressive JavaScript Framework* [online]. 2023. [visited on 2023-05-02]. Available from: <https://vuejs.org>.
37. *Introduction — Pinia* [online]. 2023. [visited on 2023-05-02]. Available from: <https://pinia.vuejs.org/introduction.html>.
38. *The official Router for Vue.js* [online]. 2023. [visited on 2023-04-09]. Available from: <https://router.vuejs.org>.
39. *Promise based HTTP client for the browser and node.js* [online]. 2023. [visited on 2023-05-02]. Available from: <https://github.com/axios/axios>.
40. *Access Tokens* [online]. 2023. [visited on 2023-05-02]. Available from: <https://auth0.com/docs/secure/tokens/access-tokens>.
41. *JSON Web Tokens - jwt.io* [online]. 2023. [visited on 2023-05-02]. Available from: <https://jwt.io>.

42. *Refresh Tokens* [online]. 2023. [visited on 2023-05-02]. Available from: <https://auth0.com/docs/secure/tokens/refresh-tokens>.
43. *OAuth 2.0* [online]. 2017. [visited on 2023-05-02]. Available from: <https://rozvoj.fit.cvut.cz/Main/oauth2>.
44. *APPS MANAGER BETA* [online]. 2014. [visited on 2023-05-02]. Available from: <https://auth.fit.cvut.cz/manager/app-types.xhtml>.
45. *XSS vs CSRF — Web Security Academy* [online]. 2023. [visited on 2023-05-02]. Available from: <https://portswigger.net/web-security/csrf/xss-vs-csrf>.
46. *Local Storage vs Session Storage vs Cookie* [online]. 2022. [visited on 2023-05-02]. Available from: <https://www.xenonstack.com/insights/local-vs-session-storage-vs-cookie>.
47. *LocalStorage vs Cookies: All You Need To Know About Storing JWT Tokens Securely in The Front-End* [online]. 2020. [visited on 2023-05-02]. Available from: <https://dev.to/cotter/localstorage-vs-cookies-all-you-need-to-know-about-storing-jwt-tokens-securely-in-the-front-end-15id>.
48. *OWASP Foundation* [online]. [visited on 2023-05-02]. Available from: <https://owasp.org>.
49. *Local Storage Versus Cookies: Which to Use to Securely Store Session Tokens* [online]. 2023. [visited on 2023-05-02]. Available from: <https://www.pivotpointsecurity.com/local-storage-versus-cookies-which-to-use-to-securely-store-session-tokens/>.
50. *Cookie Security Flags* [online]. 2023. [visited on 2023-05-02]. Available from: <https://www.invicti.com/learn/cookie-security-flags/>.
51. *requests - IBM Documentation* [online]. 2021. [visited on 2023-05-02]. Available from: <https://www.ibm.com/docs/en/cics-ts/5.3?topic=protocol-http-requests>.
52. *Cross-Origin Resource Sharing (CORS)* [online]. 2023. [visited on 2023-05-02]. Available from: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>.
53. *HTTP authentication* [online]. 2023. [visited on 2023-05-02]. Available from: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication>.
54. *Docker: Accelerated, Containerized Application Development* [online]. [visited on 2023-05-03]. Available from: <https://www.docker.com>.
55. CASTRO, Santiago. *8 Reasons Why Docker Matters For Dev* [online]. 2023. [visited on 2023-05-03]. Available from: <https://www.jobsity.com/blog/8-reasons-why-docker-matter-for-devs>.

56. *What is DevSecOps?* [online]. 2023. [visited on 2023-05-03]. Available from: <https://www.ibm.com/topics/devsecops>.
57. *The DevSecOps Platform* [online]. 2023. [visited on 2023-05-03]. Available from: <https://about.gitlab.com>.
58. *GitLab documentation* [online]. 2023. [visited on 2023-05-03]. Available from: <https://docs.gitlab.com>.
59. *CloudFIT* [online]. 2023. [visited on 2023-05-03]. Available from: <https://help.fit.cvut.cz/cloud-fit/index.html>.
60. *What is Buildah?* [online]. 2023. [visited on 2023-05-03]. Available from: <https://www.redhat.com/en/topics/containers/what-is-buildah>.
61. *Deployment — Vue CLI* [online]. 2022. [visited on 2023-05-03]. Available from: <https://cli.vuejs.org/guide/deployment.html#docker-nginx>.
62. *What is NGINX?* [online]. 2023. [visited on 2023-05-03]. Available from: <https://www.nginx.com/resources/glossary/nginx/>.
63. *Yarn - Package Manager* [online]. [visited on 2023-05-04]. Available from: <https://yarnpkg.com>.
64. KRISHNA, Ashutosh. *Yarn vs NPM: Which One is Best to Choose?* [online]. 2023. [visited on 2023-05-04]. Available from: <https://www.knowledgehut.com/blog/web-development/yarn-vs-npm>.
65. *Vitest Blazing Fast Unit Test Framework* [online]. 2023. [visited on 2023-05-05]. Available from: <https://vitest.dev>.

Acronyms

BI-DBS	Database systems
BI-SP1	Team software project 1
BI-SP2	Team software project 2
CI	Continuous integration
CD	Continuous development
CSRF	Cross-site request forg
DevOps	Development and Operations
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IT	Informational technology
JS	JavaScript
MVP	Model View Presenter
OAuth	Open authorization
ORM	Object Relational Mapping
PHP	Hypertext Preprocessor
SPA	Single-page application
SQL	Structured Query Language
URL	Uniform Resource Locator
XSS	Cross-site scripting

Contents of enclosed media

BT_Chukava_Volha.pdf	the file with the thesis in pdf format
README.md.....	the file with media contents description
src	the directory of source codes
├─ access_control.....	the directory with services for access control
├─ api.....	the directory with prepared API requests
├─ authorization	the directory with components and services
├─ automation.....	the directory with files used for automation
├─ language_config	the directory with translation configuration
├─ model.....	the directory with interfaces and enums
├─ tests.....	the directory with unit tests
text	the thesis text directory
├─ pdf.....	the directory with media of pdf format used in thesis
├─ png.....	the directory with media of png format used in thesis
├─ template.....	the directory with LaTeX template
├─ tex.....	the directory with LaTeX source files