Eötvös Loránd University

Faculty of Informatics

Department of Programming Languages and Compilers

Mini Ericsson Orchestrator Cloud Manager

Author:
Khanbayov Rasul
Computer Science BSc

Internal supervisor:
Dr. Zsók Viktória
Assistant Professor, Ph.D.

External supervisor:
Ákos Princzinger
Section Manager, MSc degree

Budapest, 2022

EÖTVÖS LORÁND UNIVERSITY
FACULTY OF INFORMATICS

Thesis Registration Form

Student's Data:
Student's Name: Khanbayov Rasul
Student's Neptun code: GGUSA3

Course Data:
Student's Major: Computer Science BSc

I have an external supervisor

External Supervisor's Name: *Akos Princzinger*

Supervisor's Home Institution:	*Ericsson Hungary*
Address of Supervisor's Home Institution:	*Budapest Magyar tudósok körútja 11. 1117*
Supervisor's Position and Degree:	*Section Manager, MSc degree*
Supervisor's e-mail address:	*akos.princzinger@ericsson.com*

Internal Supervisor's Name: *Dr. Zsók Viktória*

Supervisor's Home Institution:	*ELTE IK PNYF ELTE Fac of Informatics, PLC*
Address of Supervisor's Home Institution:	*H-1117 Budapest, Pázmány Péter sétány 1/C*
Supervisor's Position and Degree:	*Assistant Professor, Ph.D.*

Thesis Title: Mini Ericsson Orchestrator Cloud Manager

Topic of the Thesis:
(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis.)

Nowadays Cloud computing is quickly gaining popularity with companies and organizations as a result of its ease, convenience and efficiency. With the help of these technologies your data in applications like Facebook, Twitter and Google Drive is stored virtually. You are able to enter them with the appropriate authentication and procedures, which lets you to access from any device. Cloud technologies also provide Internet Service Provider operators and enterprises with the ability to manage Network Functions Virtualization (NFV) services and IT workloads in a more efficient manner. My thesis is about a "Mini" Ericsson Orchestrator Cloud Manager (EO CM) application, which will provide an integrated platform for managing life cycle of NFV and IT workloads. Furthermore, it will manage virtualization resources in a flexible way, allowing cloud providers to build and grow cloud infrastructure for virtualized applications and VNF-based services as well as it provides a secure GUI and open APIs for all the NFV and IT workload management services. Basically, this application offers a space where users can deploy packages for VIM Zone services. Using GUI the user will be able to operate components on the VIM Zone. The product's security protects against an unauthorized access to Mini EO CM. Provides tenant-based access control, user level authentication, and role-based access control. The product's backend will be done in Java, the container will be needed in Kubernetes, additionally a database will be used to store all cloud objects and their attributes, including those that are instantiated in the VIM zones. After some investigations, it is possible that the GUI will be done in JavaScript and YMER framework. To sum up, Mini EO CM application will handle the life-cycle management of virtual applications.

Budapest, 2021. 12. 09.

Contents

Chapter 1

Introduction

1.1 Motivation

We are living in an increasingly hyperconnected world and it is not just people that are connected anymore. The Internet of Things (IoT) is creating new connectivity requirements and scenarios for our world. This connectivity is provided by Communications Service Providers (CSP) which currently facing an exponential rise in the number of customers in their network. This leads to capital expenditures, operating expenses as well as inability to offer new services and applications for users dynamically. IT organizations use the deployment of shared and virtual infrastructure to manage the delivery of multiple networked applications and expand resource utilization in a more efficient way.

Network Functions Virtualization (NFV) applies virtualization technologies to the long-established network and service functions, and associated applications that CSPs use to reduce the cost and improve time-to-market in their network infrastructure [1]. Cloud technologies, especially Ericsson Orchestrator Cloud Manager offers services such as building cloud infrastructure for virtualized applications, secure GUI, and open API for all management services. Cloud Manager enables orchestration of resources across different Virtualized Infrastructure Managers (VIM) (OpenStack or VMware) additionally Kubernetes to 5G and IoT customers.

The main purpose of this thesis is to build an integrated platform where users are able to manage the life cycle of NFV and IT workloads. By taking advantage of this platform, they do not need to be using various applications to create packages, upload YAML format files and deploy/delete resources from Kubernetes.

1.2 Thesis structure

Chapter 2 User Documentation contains:

- A short description of the cloud and cloud computing. See section 2.1.
- A brief introduction to the tools used in the front-end and back-end developments, all the requirements for the installation steps, and running the project. See section 2.2.
- A detailed overview of the usage of the application from the client's perspective. See section 2.3.

Chapter 3 Developer Documentation contains:

- A detailed specification of the application. See section 3.2.
- A comprehensive description for the REST architectural style. See section 3.3.
- Explanation to the communication of Spring Boot with React Js and PostgreSQL database. See subsection 3.3.3 and subsection 3.3.4.
- Advantages of Java Kubernetes client library and its utilization in the application. See subsection 3.3.6.
- Usage of Jasypt java library for basic encryption and Spring boot authentication. Check subsection 3.3.7 and subsection 3.3.8 respectively.
- Testing, a short description of Spring MVC test framework, test templates for different HTTP requests, and results of integration tests. See section 3.4.

Chapter 2

User Documentation

In this chapter, Mini Ericsson Orchestrator Cloud Manager is explained from an end-user perspective. Initially, a brief introduction to the cloud concepts is detailed. Then the tools used in front-end and back-end are introduced. Afterwards, an instruction to install and run the system is provided as well as all the requirements for the usage of this system. In conclusion, explanations on how to use each part of the application are given.

2.1 Cloud

What is *the cloud*?

Due to the boom in cloud computing in recent years, *the cloud* as a term is fully integrated into the lives of ordinary people. The cloud refers to the servers that are accessed over the Internet, along with the software and databases that run on those servers [2]. By using the cloud, users and companies do not have to manage physical servers themselves or run software applications on their own machines. The cloud enables users to access the same applications and files from almost any device, because the computing and storage take place on servers in a remote data center, instead of the user device storage. For example, Gmail stores emails and attachments on google drive cloud storage allowing users to access their emails and files via any internet-connected device. Such benefits also include data security, unlimited storage capacity, backup, and data restoration.

If we look at the relationship between a user and the cloud, the examples detailed above cover the uses of the cloud. In short, it can be said that all usage is based on storing certain data.

However, if we will get a view from a different perspective, the developer's side, the field of the application will be tremendously widened. The focus shifts from storing data to processing data. In general, applications require increasing computing capacity. As a consequence of the growing demand for memory, the demand for processors has increased.

In fierce business competitions, there is a growing burden on developers. Diverting to cloud computing reduces IT and operating costs. For example, as the cloud vendor is used to update and maintain servers, they do not need to do it manually every time. This is really useful for small businesses when they are not able to afford internal infrastructure but can outsource their infrastructure needs affordably by means of the cloud. Moreover, international companies are operating easily by using the cloud, since employees and customers can access the same files and applications from any place.

Ericsson Orchestrator Cloud Manager allows cloud providers to build and grow cloud infrastructure for applications and services. It became necessary due to problems examining development opportunities. In connection with this, it is also worthwhile to review what is cloud computing.

2.1.1 Cloud computing

Humans use cloud computing without even realizing it. Using an online service to send an email, watch TV or movies, edit documents, or store pictures and other files, it is likely that it is cloud computing that makes it all possible in the background. So cloud computing is the delivery of computing services — including servers, storage, databases, networking, and software to offer faster innovation, and flexible resources. Customers usually pay for the cloud services they use, which helps to lower operating costs, run infrastructure more efficiently, and scale as their business needs change [3].

Benefits of cloud computing other than eliminating capital expenses are productivity, speed, and performance. Cloud companies have their own data centers where servers and/or data storage are located. Cloud data centers are regularly upgraded to the latest generation of fast and efficient computing hardware. The advantage is reduced network latency for applications and greater economies of scale over a single data center. Vast amounts of computing resources can be provisioned in minutes, which gives businesses a lot of flexibility and decreases pressure on capacity planning. When it comes to performance, it is about the time saved by not setting up hardware, software patching, and other time-consuming IT management tasks. Cloud computing takes away the need for many of these tasks, so developers can spend time on achieving more important business goals [4].

The initial cloud computing services are barely a decade old, nevertheless, small start-ups to international companies are embracing the technology for all kinds of reasons. By utilizing cloud computing services, users can create cloud-native applications, store, recover and analyze data as well as deliver software on demand.

2.1.2 Docker Container

A container is a standard unit of software package that includes the software code and all its dependencies, so the application runs quickly and reliably regardless of the computing environment [5].

Docker containers can be stored in image files and they are easy to build on their own executable software packages that contain everything you need to run an application: code, system tools, required system libraries, and settings.

Container image files become containers at runtime and in the case of Docker containers – images become containers when they run on the Docker Engine. The Docker Engine is available for Linux, MacOS, and Windows operating systems. In such containers, stored software is always the same and can run regardless of the infrastructure. The containers isolate the software from its environment and ensure that the software works uniformly. I am using the docker container to set up the PostgreSQL database. This is done by building an image from a Dockerfile. Dockerfile is a text file, it consists of all the commands needed to build a given image.

2.1.3 Kubernetes

Container orchestration is the automation of much of the operational effort required to run containerized workloads and services. This includes a broad range of tasks software teams need to manage a container's lifecycle, including deployment, provisioning, networking, scaling, load balancing, and so on.

If we are thinking about load-based handling of containers, Kubernetes can be considered one of the most common choices. It allows developers to easily build containerized applications and services, as well as scale, schedule, and monitor those containers. It is a great choice to have an already proven container for an orchestration solution need. Because of this, Kubernetes is currently the most popular container for orchestration platforms. While there are other options for container orchestration, such as Docker Swarm or Apache Mesos, Kubernetes has become the industry standard. Several of the world's largest companies use it, and a significant portion of them is also involved in its development. Kubernetes contributes large-scale container capabilities, a dynamic contributor community, the growth of cloud-native application development, and the widespread availability of Kubernetes tools [6]. So, I used Kubernetes to provide me with the platform where the user is able to deploy and delete several resources such as *ReplicaSet*, *ControllerRevision*, and more.

Pods

The smallest execution units are called Pods in a Kubernetes cluster. One or more containers are enclosed in a pod instead of one cluster running on cluster nodes. The common resources and local network are shared in a pod by applications which makes it easy to communicate between applications. Pods use a kubelet on each node to communicate with Kubernetes API and the rest of the cluster. Kubernetes can automatically recreate the pod to attain the desired scalability as the load increases.

Nodes

Node is the smallest unit of the compute hardware in a Kubernetes cluster. Nodes form a layer of abstraction like containers. One can think of a node as a basic resource with power and memory, then they can be identified with each other. All of them form the Kubernetes cluster. As needs alter, the Kubernetes cluster automates distributing of workloads. Nodes are immediately removed from the cluster when they are failed. In case of failure, new nodes replace them.

Clusters

A collection of nodes called Kubernetes cluster. They can be both physical servers or virtual machines. The User is always managing a cluster when utilizing Kubernetes. At least one instance of the Kubernetes control plane needs to run on a node, and at least one node for pods to execute on. On the condition that nodes are added or removed from the cluster, the cluster itself will automatically redistribute the load as necessary.

2.1.4 Cloud Management Database

Cloud Management Database stores the records for resources and services. It is a central repository that reflects the presence and state of objects in all infrastructure managers managed by Ericsson Orchestrator Cloud Manager. In other words, the Cloud Management Database stores all cloud objects and their attributes, including those that are instantiated in the VIM zones. PostgreSQL is used in the project as the cloud management database to be in charge of storage services.

2.2 Install Guide

2.2.1 Back-End Tools

Back-End development of the project was built up by Java Spring Boot with Spring Framework as the main programming language and PostgreSQL as a database tool. The followings explain each one of them in a brief description.

Spring Framework

Spring is the world's most popular Java framework for its speed, productivity, and simplicity. With the help of the Spring framework, Java programming becomes faster, easier, and safer for everyone.

"We use a lot of the tools that come with the Spring framework and reap the benefits of having a lot of the out of the box solutions, and not having to worry about writing a ton of additional code—so that really saves us some time and energy." Sean Graham, Application Transformation Lead [7].

Spring has lots of trusted libraries that have been proven by developers in the whole world. Tech companies like Google, Microsoft, and many others have given their contributions to this framework. Spring brings solutions to everyday activities such as online shopping, streamings, or an immense amount of other innovations. The framework is able to create almost any application client can imagine by reason of third-party libraries and extensions. Spring is capable of building both secure cloud-based microservice for the web and complex flowing data streams for the companies.

Java Spring Boot

Java Spring Boot is a tool that develops web applications and microservices with Spring Framework quicker and more simple [8]. Spring Framework offers built-in assistance for common tasks that an application needs to perform. Here include data binding, type conversion, validation, exception handling, and more. Spring Boot is really useful when developers need to deploy microservices-based architectures into the cloud since Spring Cloud has supporting libraries and patterns. This transforms how you approach programming tasks, especially, Java related. Once you work on Spring Boot, you will notice fast start-up, shutdown, and optimized execution. Representational State Transfer (REST) APIs are developed by Spring Boot in Mini Ericsson Orchestrator Cloud Manager to establish communication between client and server over HTTP.

PostgreSQL

PostgreSQL is an advanced, enterprise-class open-source relational database that supports both SQL (relational) and JSON (non-relational) querying. It is a highly stable database, backed by more than 20 years of development which has contributed to its high levels of integrity, resilience, and correctness. PostgreSQL is used as a primary database for many web applications as well as mobile and analytics applications [9]. It supports the most popular programming languages such as Python, Java, C/C+, JavaScript (Node.js), and Go.

The benefits of PostgreSQL are abundant features and extensions, reliability and standards compliance, and open source license. PostgreSQL owns robust feature sets including point-in-time recovery, tablespaces, nested transactions, a refined query planner/optimizer, and write ahead logging. It also supports international character sets, multi-byte character encodings, Unicode, and it is locale-aware for sorting, case-sensitivity, and formatting. PostgreSQL is a highly fault-tolerant database because of its write-ahead logging. It has full support for foreign keys, views, joins, and stored procedures, in many different languages. PostgreSQL source code is accessible under an open-source license, allowing you to use, modify, and implement it as you want at no charge [10]. PostgreSQL provided me with storage where all the details about the users, created packages, and deployed resources into Kubernetes are kept.

2.2.2 Front-End Tools

Front-End development of the project has been prepared by React JS, React-Bootstrap, Material UI, HTML, and CSS are used. The followings explain each one of them in a brief description.

React JS

React what is also known as React.js or ReactJS is a free and open-source front-end JavaScript library for creating user interfaces based on UI components. It is sustained by Meta and a group of individual developers and companies. React can be used as a fundamental in the development of single-page, mobile, or even server-rendered applications with frameworks. Nevertheless, React is only troubled with state management and rendering that state to the DOM. Consequently, React applications usually requires the use of supplementary libraries for routing and other client-side functionalities [11].

React-Bootstrap

React-Bootstrap replaces the Bootstrap JavaScript. Each one of the components has been created from scratch which converts into a true React component. In this case, dependencies like jQuery and others are unnecessary. React-Bootstrap has improved alongside React JS which is making it the perfect choice for UI design. It is well-suited to all types of Bootstrap themes, which gives a consistent appearance and maintains uniformity in the application [12].

Material UI

Material UI is an open-source, front-end framework for React components. This framework is built using Leaner Style Sheets (LESS) which is a backward-compatible language extension for CSS. While developing front-end graphics of the application Material UI provides a high-quality digital experience based on Google's Material Design. Its main focuses are on providing bold and crisp designs, for example, it creates text fields by focusing on how components reflect the light and cast shadows [13].

HTML

HTML which stands for Hyper Text Markup Language is the standard markup language for creating Web pages. It describes the formation of a Web page and consists of a sequence of components. Each component of the HTML helps to label the content which makes it easy for the user to understand the interface [14].

CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in HTML or in other markup languages. CSS provides the layout, fonts, and color which improve content accessibility by providing more flexibility and control in the specification of presentation characteristics. It can control the layout of multiple web pages all at the same time [15].

2.2.3 System Requirements

The requirement of the system to run this web application consists of:

- Operating System: Windows 10
- RAM: 16.0 GB
- CPU: 2.6GHz
- Java: 17.0.2

2.2.4 Environment preparation

Here are the instructions on how to install the Java Development Kit, PostgreSQL, React JS library, Docker Desktop, Minikube, Node Js, and integrated development environment: IntelliJ IDEA. If you already have relevant versions installed on your device, this section can be skipped. Installation links are given in the references.

1. Install Java Development Kit
   The latest version of Java Development Kit (JDK) can be installed from the official oracle website [16]. In order to make sure JDK is installed successfully, open the command prompt and use the command 'java -–version' to check.
2. Install IntelliJ IDEA [17]
3. Install Docker Desktop [18]
4. Install Node Js [19]
   In order to make sure it is installed successfully, open the command prompt and execute *npm -v* command to check.
5. Install React Scripts
   React Scripts can be installed with *npm install react-scripts* command.
6. Install Minikube [20]
7. Install PostgreSQL [21]

2.2.5 Install and run the project

All the necessary tools to run the project have been installed, and here an introduction to how could we install and run the project for users is explained.

1. Install the project [22]
2. Create docker image on `src/main/docker/docker-compose.yml` file. Open PowerShell on this `src/main/docker` folder and run the following command:

```
    docker-compose up --build
```

Code 2.1: Command to create docker image

3. Start a cluster with 2 nodes.

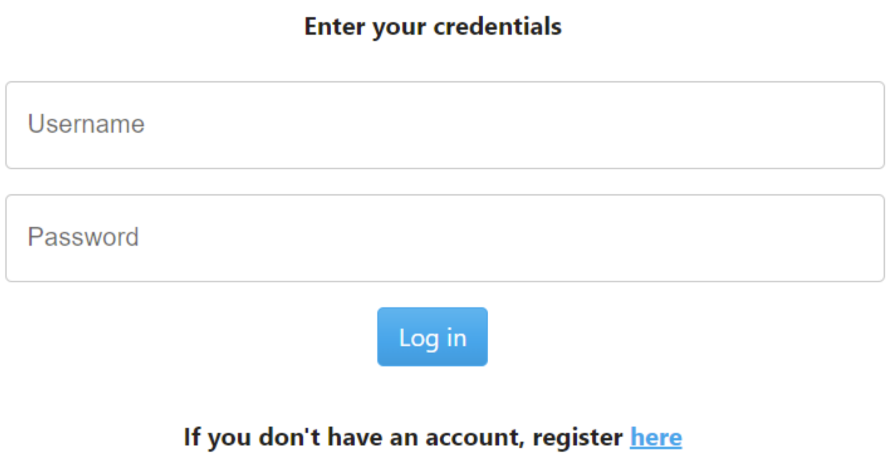
```
    minikube start --nodes 2 -p multinode-demo
```

Code 2.2: Command to start a cluster with 2 nodes

4. Prepare pgAdmin from PostgreSQL database for the project
   Create server and fill following parameters:
   - Name: Mini EO-CM
   - Host name/address: localhost
   - Port: 5454
   - Maintenance database: postgres
   - Username: postgres
   - Password: postgres
5. Open the project in IntelliJ IDEA and run main method from `src/main/java/com/ericsson/packageManager/PackageManager.java`
6. Open `packagefrontend` in Command Prompt and execute *npm start* command. The website with URL - `http://localhost:3000/` will be opened.

2.3 Usage of the system

If the above conditions are fulfilled then the login page will be opened automatically.



Enter your credentials

Username

Password

Log in

If you don't have an account, register here

Figure 2.1: Login page

The user should log in to the system with the correct username and password. Otherwise, error messages will be displayed by explaining what is incorrect with the credentials. If the user does not have an account, he should click on the *here* link which will forward him to the registration page.

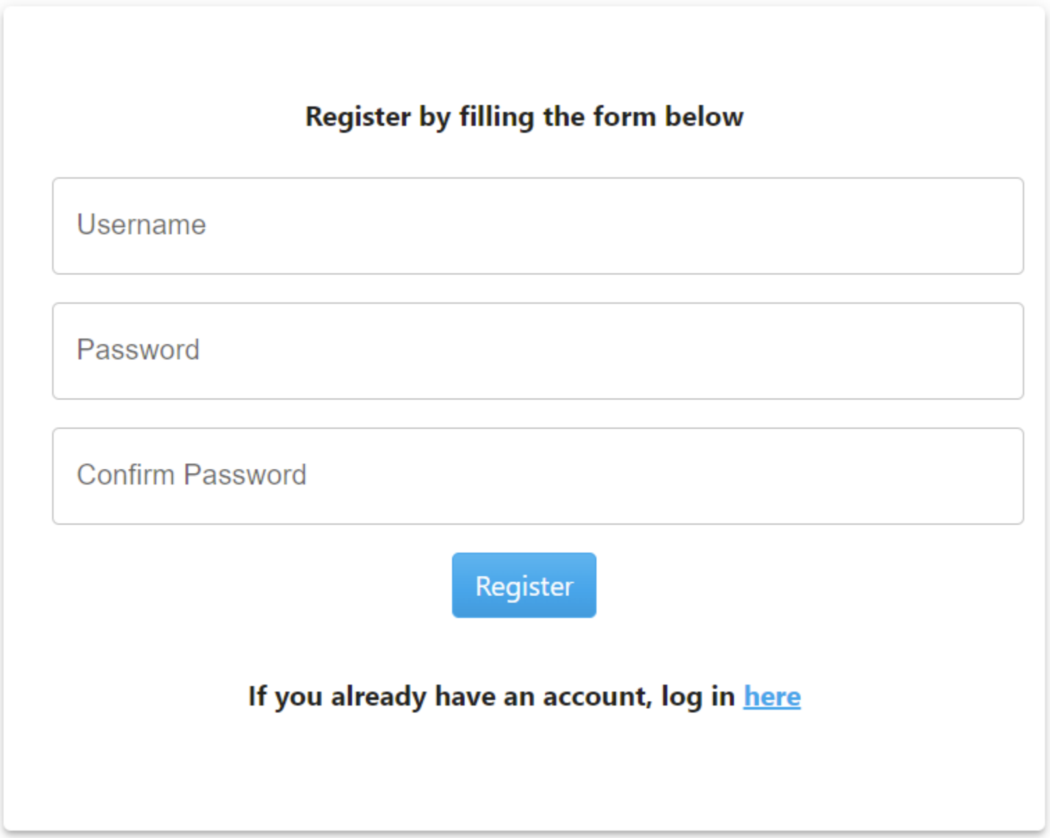
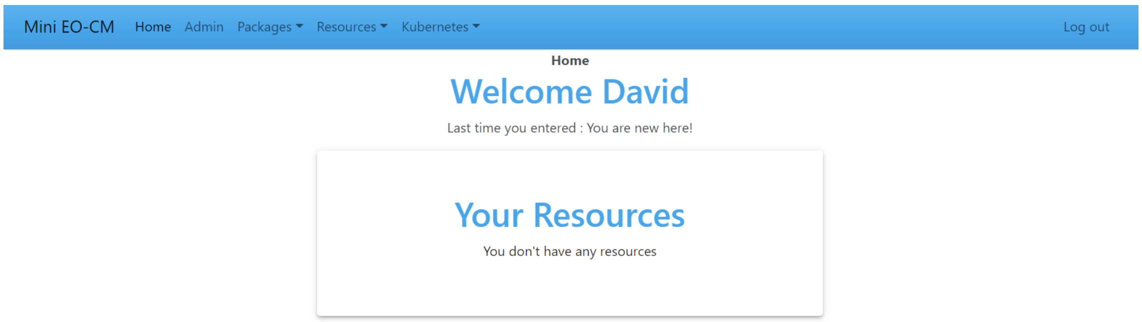

Register by filling the form below
Username
Password
Confirm Password
Register
If you already have an account, log in here

Figure 2.2: Registration page

When creating a new account, the user must follow the rules specified below:

- *Username* must be unique, size must be between 5 and 15.
- *Password* and *Confirm Password* must match, and the size must be between 5 and 15 as well.

If the above conditions are not satisfied, error messages will pop up and explain why the user was not been able to create an account. After successful registration, the application will redirect the user to the login page. The user should enter the credentials that were registered previously. Succeeding with logging in will be proceeded by redirecting to *Home Page*.



Home

Welcome David

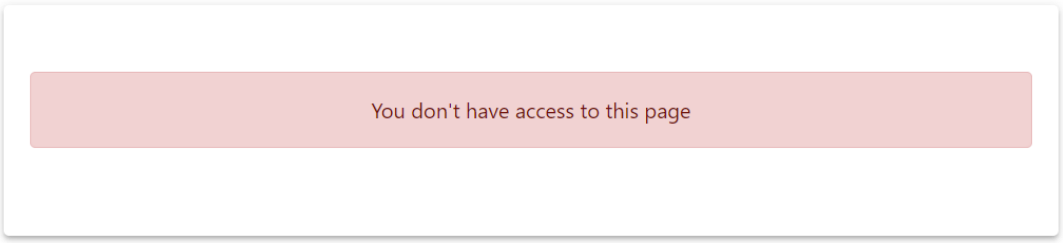
Last time you entered : You are new here!

Your Resources

You don't have any resources

Figure 2.3: Home page

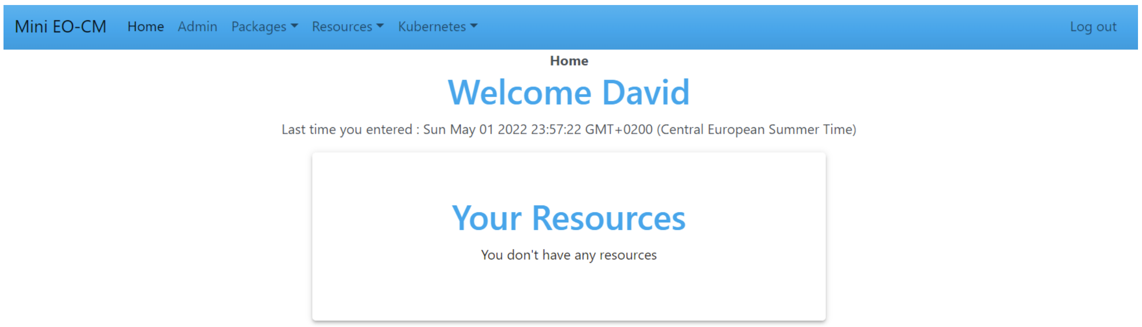
In the header of *Home Page*, there are several menu items arranged. *Home* menu item is the current place of *Home Page*. That is followed by *Admin* page, which is only accessible by Admin users. If the user wants to access *Admin* page, an error message will be displayed.



You don't have access to this page

Figure 2.4: Not authorized Admin page

If it is the first time user entered the application, *You are new here!* information will be written in *Last time you entered* field. When the user logs out, the application will keep the date and save it in the database. Next time, that date will appear as demonstrated in Figure 2.5.



Home

Welcome David

Last time you entered : Sun May 01 2022 23:57:22 GMT+0200 (Central European Summer Time)

Your Resources

You don't have any resources

Figure 2.5: Home page with last entered time

2.3.1 Packages

The next header element is *Packages* menu item, which has 4 operations.

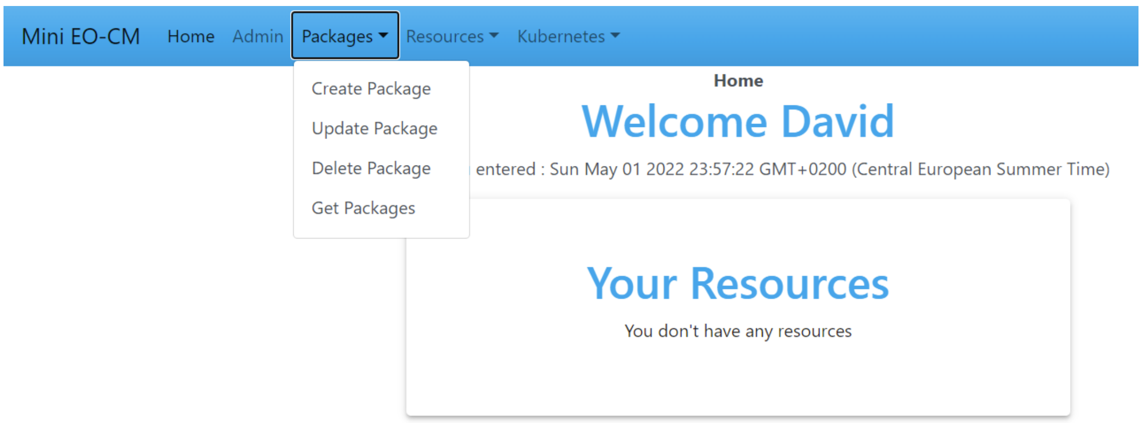


Figure 2.6: Operations on Packages

Creating a package

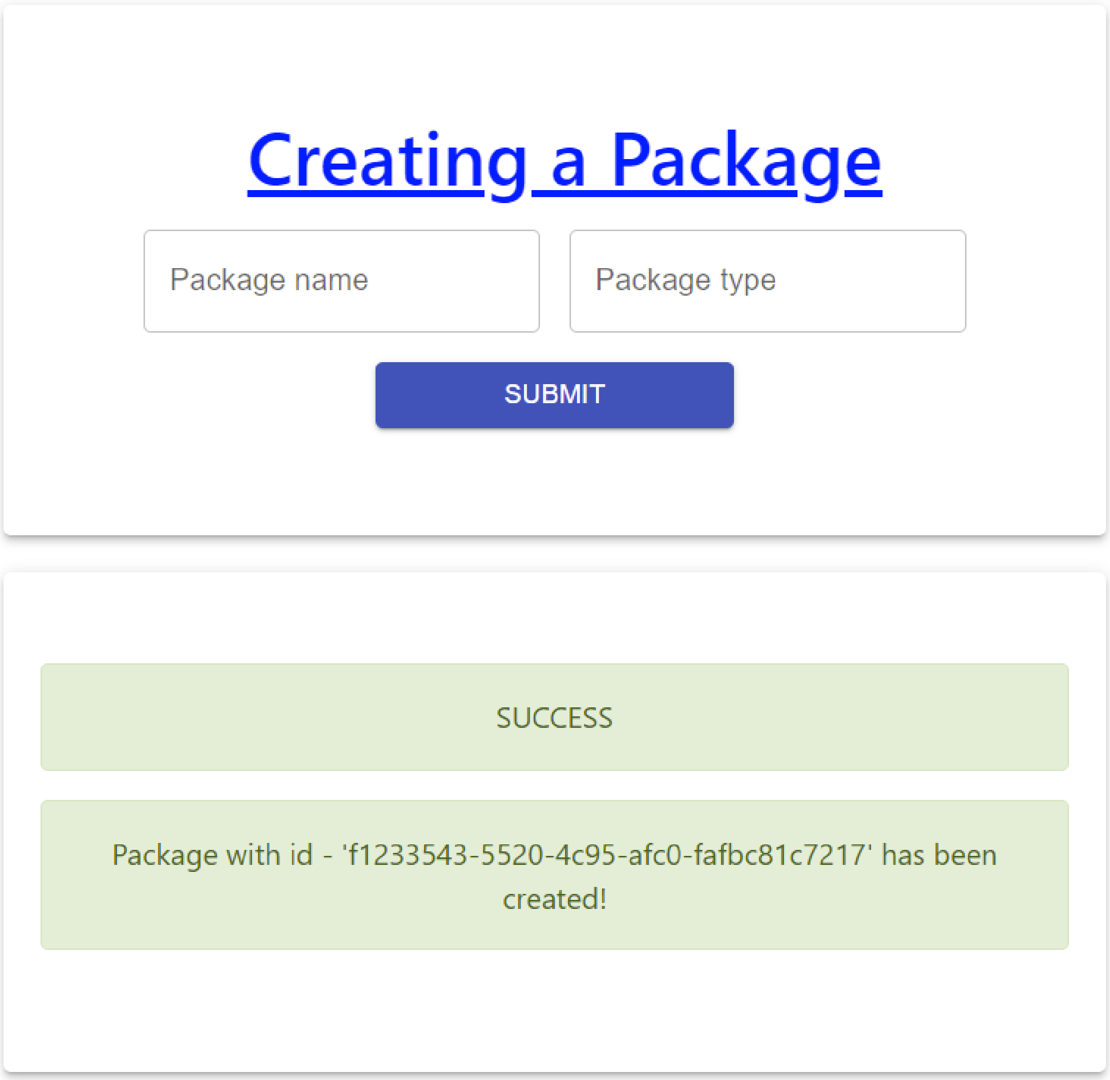
The user is able to create packages. He should enter the name and type for packages. Application supports 1 type of package for now:

1. CNF - used to deploy Container-Based Network Functions into Kubernetes clusters(the Container Infrastructure Service Management (CISM) cluster)

During Package creation rules specified below must be obeyed:

- *Package name* size must be between 5 and 15.
- *Package type* must be *CNF* package.

After the successful creation of the package, the application shows the package ID generated based on the following parameters as illustrated in Figure 2.7. By using this ID, the user is able to update and delete this package.



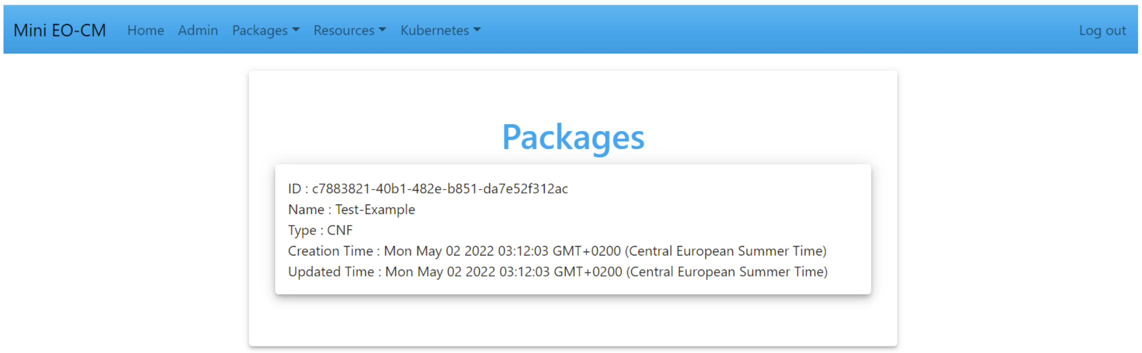
SUCCESS

Package with id - 'f1233543-5520-4c95-afc0-fafbc81c7217' has been created!

Figure 2.7: Creation of a Package

List of Packages

The user is able to see a list of packages created. Here the information about Package ID, name, type as well as creation, and updated time are displayed.



ID : c7883821-40b1-482e-b851-da7e52f312ac
Name : Test-Example
Type : CNF
Creation Time : Mon May 02 2022 03:12:03 GMT+0200 (Central European Summer Time)
Updated Time : Mon May 02 2022 03:12:03 GMT+0200 (Central European Summer Time)

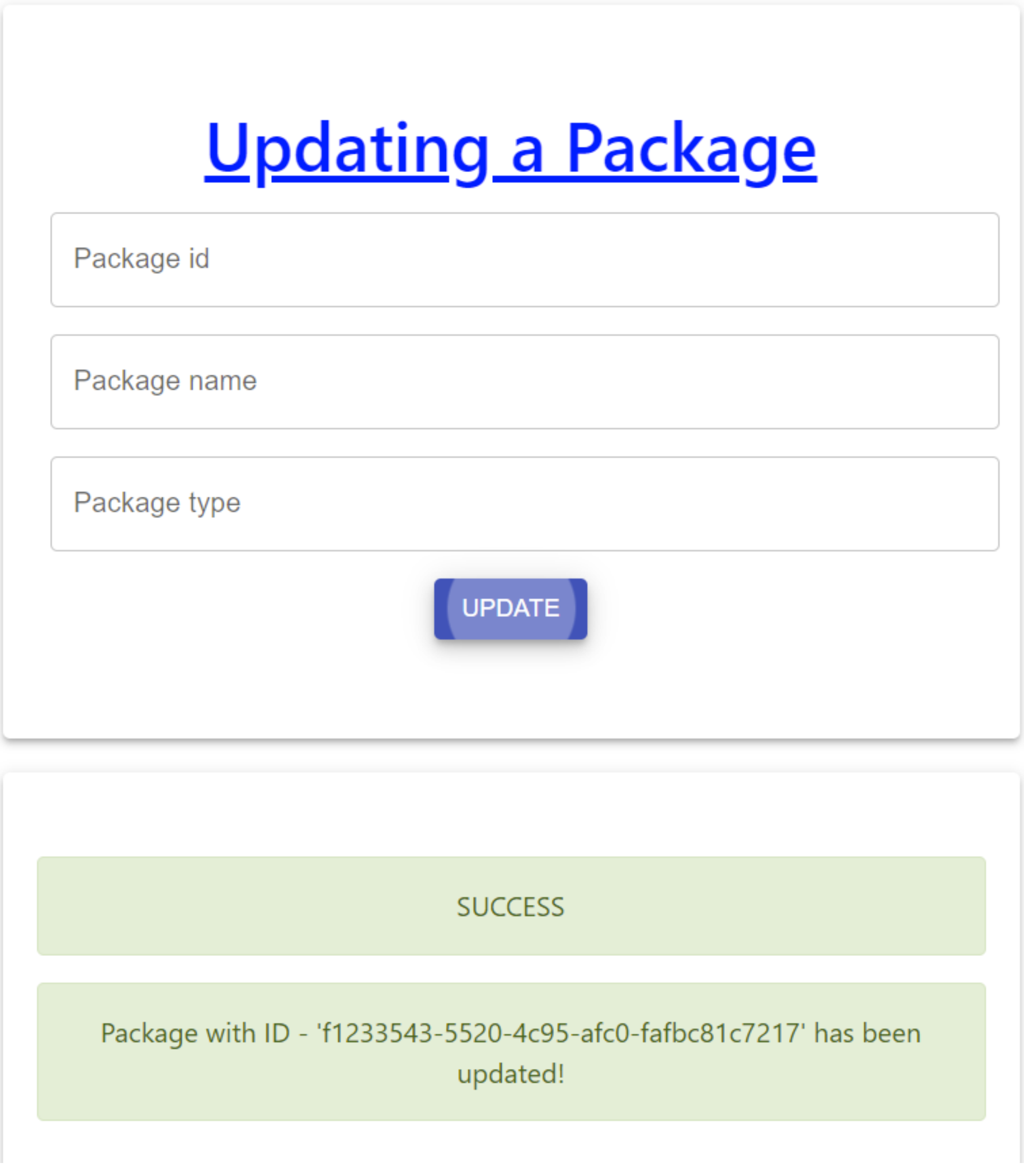
Figure 2.8: List of Packages

Updating a package

Packages can be updated if the name or type attributes of the package have been created inaccurately. The rules shown below must be followed:

- Package ID must exist.
- Package name size must be between 5 and 15.
- Package type must be *CNF* package.
- There must not be deployed *Resource* that related to this Package.

After a successful update of the package, a message will appear describing the package with the following ID that has been updated.


Updating a Package

Package id

Package name

Package type

UPDATE

SUCCESS

Package with ID - 'f1233543-5520-4c95-afc0-fafbc81c7217' has been updated!

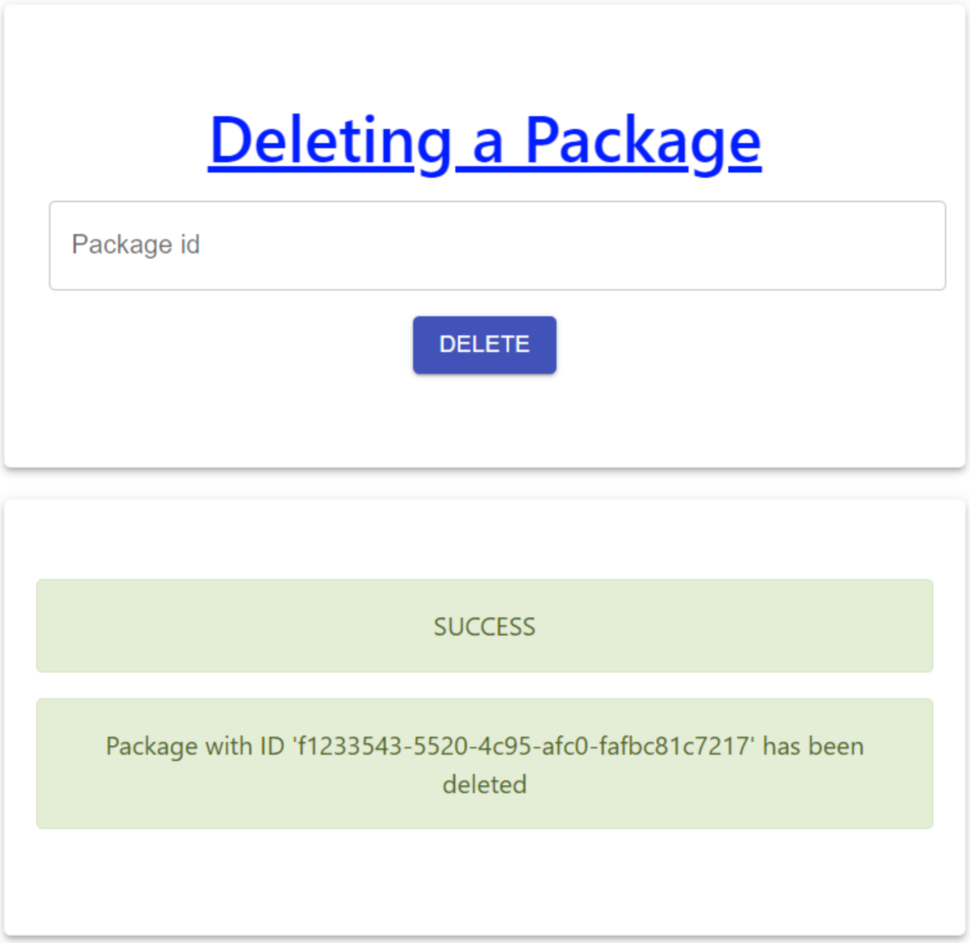
Figure 2.9: Successful update of a Package

Deleting a package

Packages can be deleted as well by entering Package ID. The rules shown below must be followed:

- Package ID must exist.
- There must not be deployed *Resource* that related to this Package.

The message, as described in Figure 2.10, will appear by the continuation of successful package deletion. On the condition that the user also had undeployed *Resources* related to this package, they will be deleted as well.



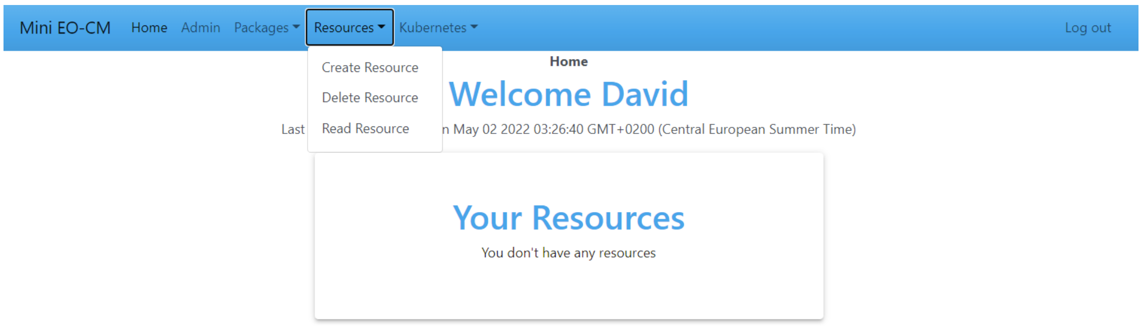
SUCCESS

Package with ID 'f1233543-5520-4c95-afc0-fafbc81c7217' has been deleted

Figure 2.10: Deletion of a Package

2.3.2 Resources

After creating packages and some other operations, based on these packages, Resources can be created which later will be deployed to the Kubernetes. Besides creating a Resource, reading and deleting that resource is also possible.



Home

Welcome David

Last … n May 02 2022 03:26:40 GMT+0200 (Central European Summer Time)

Your Resources

You don't have any resources

Figure 2.11: Operations on Resource

Create a Resource

Resources can be created based on the Package ID and YAML file. The user generates a YAML file and sends it alongside kind and Package ID. Kind represents the type of Kubernetes objects to be created while using the YAML file. The application supports 6 kinds:

1. **Service** - a REST object in Kubernetes which is similar to a Pod. Like all of the REST objects, a POST HTTP request can be sent for a Service definition to the API server in order to create a new instance. Service object name must be a valid RFC 1035 label name [23].

2. **Deployment** - provides declarative updates for Pods and ReplicaSets. Deployments can be defined to create new ReplicaSets or to detach existing Deployments and adopt all their resources with new Deployments.

3. **Job** - can create several Pods. This will be continued until a specified number of retry execution of the Pods successfully terminated. Once pods are complete, the Job tracks the successful completions. The Job is considered complete when a specified number of successful completions is reached

4. **ReplicaSet** - The purpose of ReplicaSet is to maintain a steady set of replica Pods running at any given time. In essence, it is frequently used to guarantee the availability of a specified number of identical Pods.

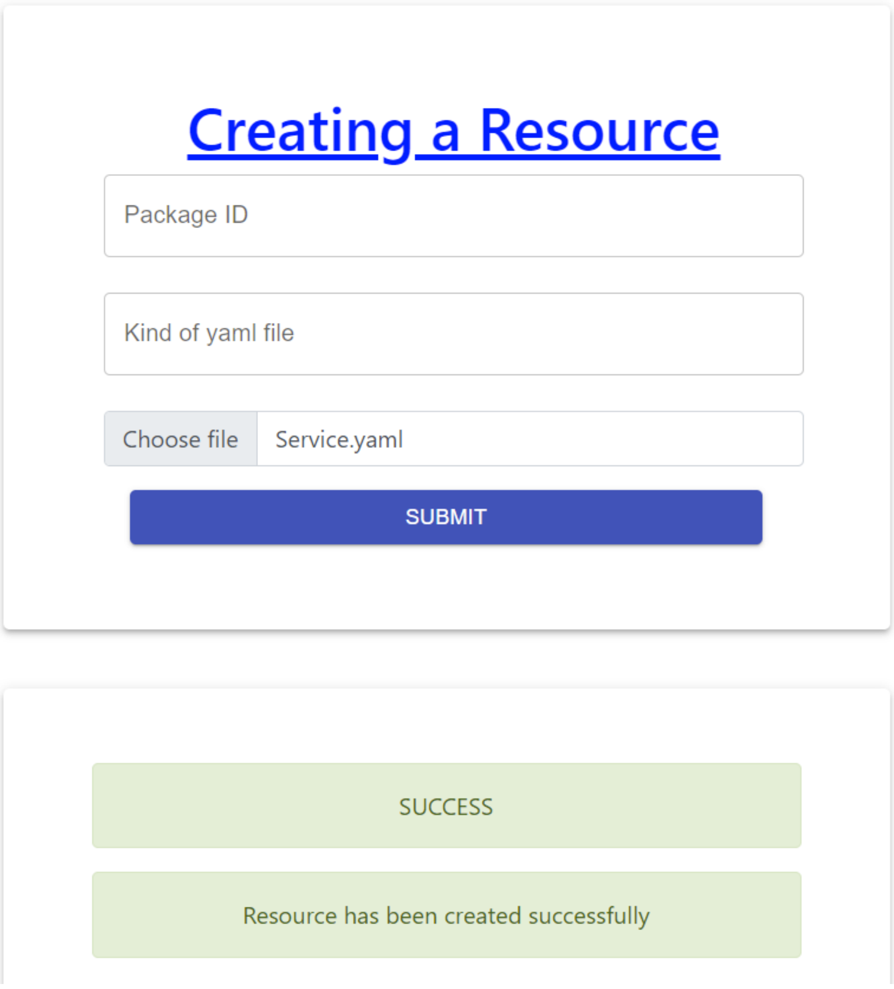
5. **StatefulSet** - workload API object used to manage stateful applications. Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.

6. **ControllerRevision** - implements an immutable snapshot of state data. Users are responsible for serializing and deserializing the objects that contain their internal state. ControllerRevision can not be updated once it is created.

Example YAML file for each Kubernetes object can be found in `exampleResources` folder. The rules shown below must be followed when creating a Resource:

- Package ID must exist.
- Kind must be supported.
- If there is already a Resource based on this Package ID, this will replace an already existing resource, nevertheless, the resource must be available.

If the above-mentioned conditions are satisfied, then Resource is created and the file is uploaded to the system. On the condition that the kind is not supported or the file is syntactically wrong, then detailed error messages will be displayed.

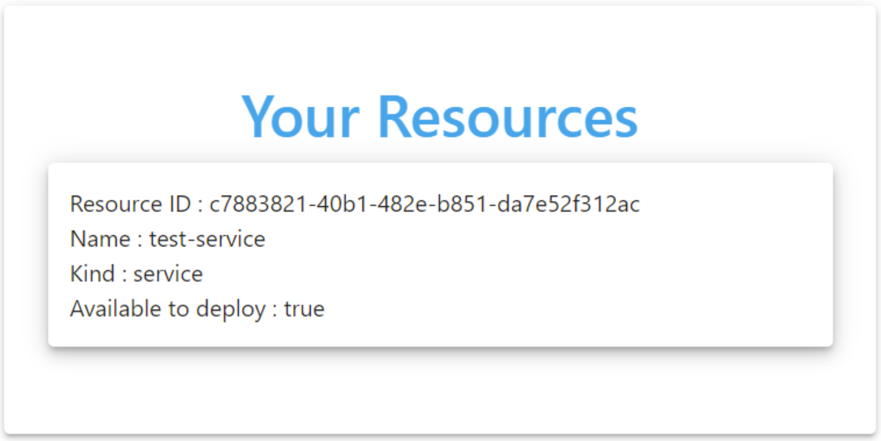


SUCCESS

Resource has been created successfully

Figure 2.12: Creation of Resource

After the creation of the Resource, it will be listed on *Home* page, and its relevant Resource ID, Name, Kind, and availability are illustrated for each resource.

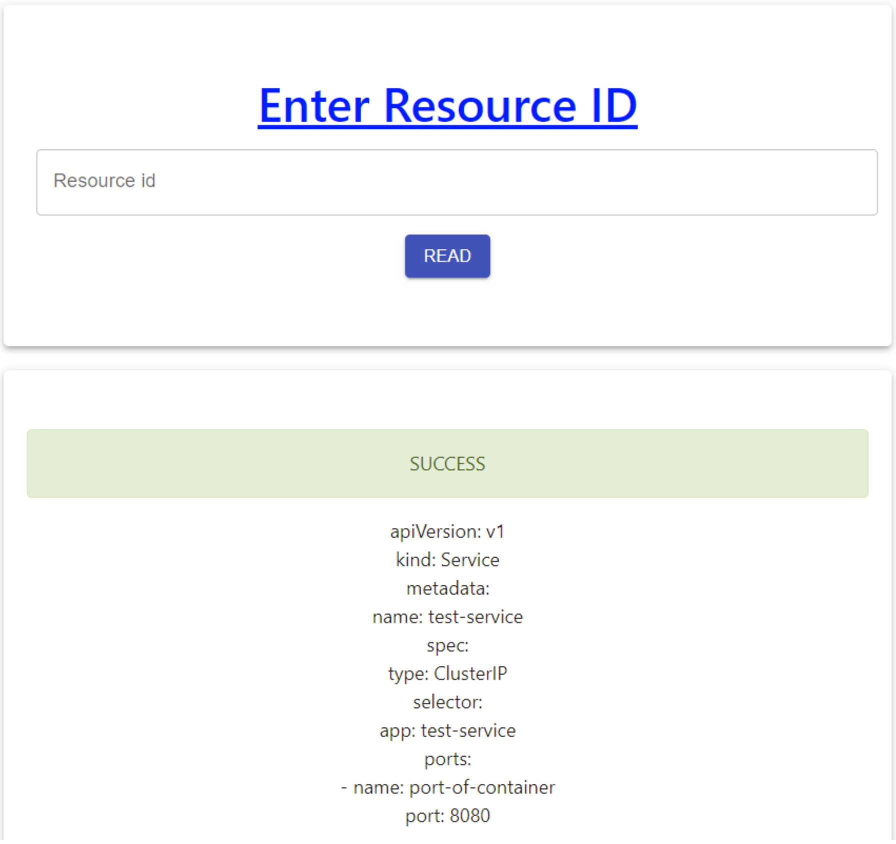


Resource ID : c7883821-40b1-482e-b851-da7e52f312ac
Name : test-service
Kind : service
Available to deploy : true

Figure 2.13: List of Resources

Read Resource

File that has been uploaded to the system by creating a Resource can be viewed by entering relevant Resource ID.



```
apiVersion: v1
kind: Service
metadata:
name: test-service
spec:
type: ClusterIP
selector:
app: test-service
ports:
- name: port-of-container
port: 8080
```

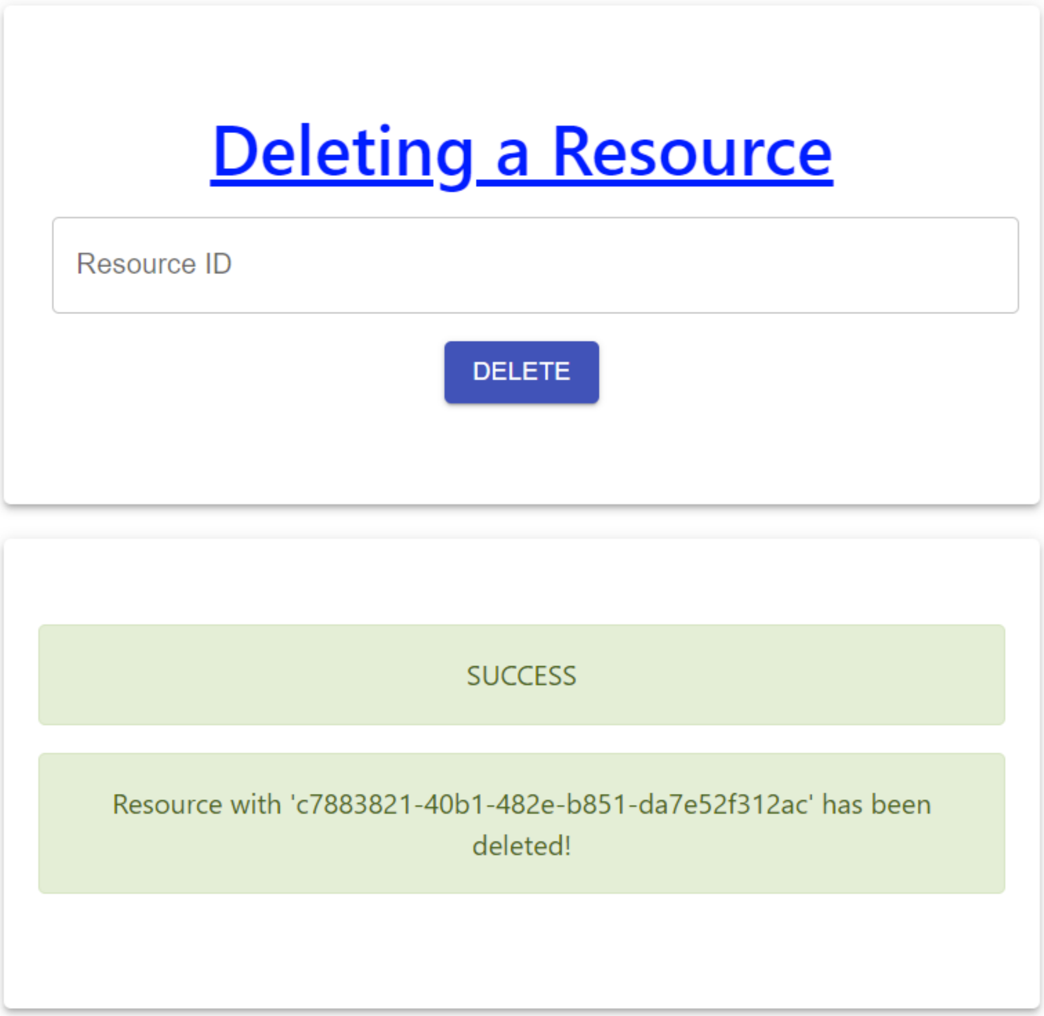
Figure 2.14: Reading a Resource

Delete Resource

A Resource can be deleted by entering the relevant Resource ID. The rules shown below must be followed:

- Package ID must exist.
- The Resource must be available, it must not be deployed to Kubernetes.

The message, as described in Figure 2.15, will appear by the continuation of successful Resource deletion. The relevant file that has been uploaded will be deleted as well.



SUCCESS

Resource with 'c7883821-40b1-482e-b851-da7e52f312ac' has been deleted!

Figure 2.15: Deleting a Resource

2.3.3 Kubernetes

Kubernetes is the next header element. Users can list all the pods and nodes, can deploy Resources that have been created previously, and delete them. The following subsections explain every operation in a more detailed version.

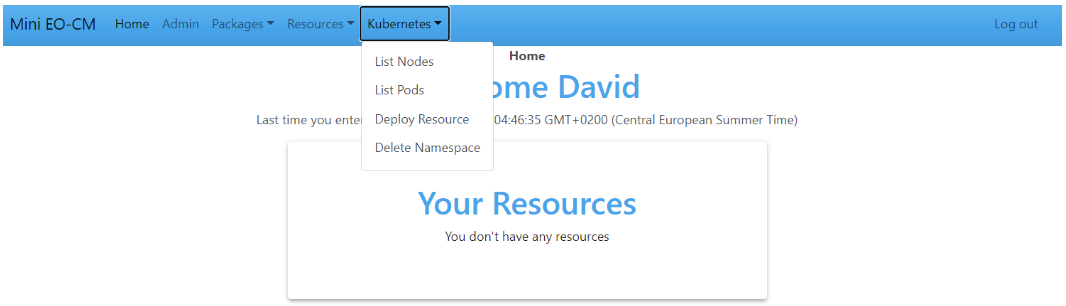
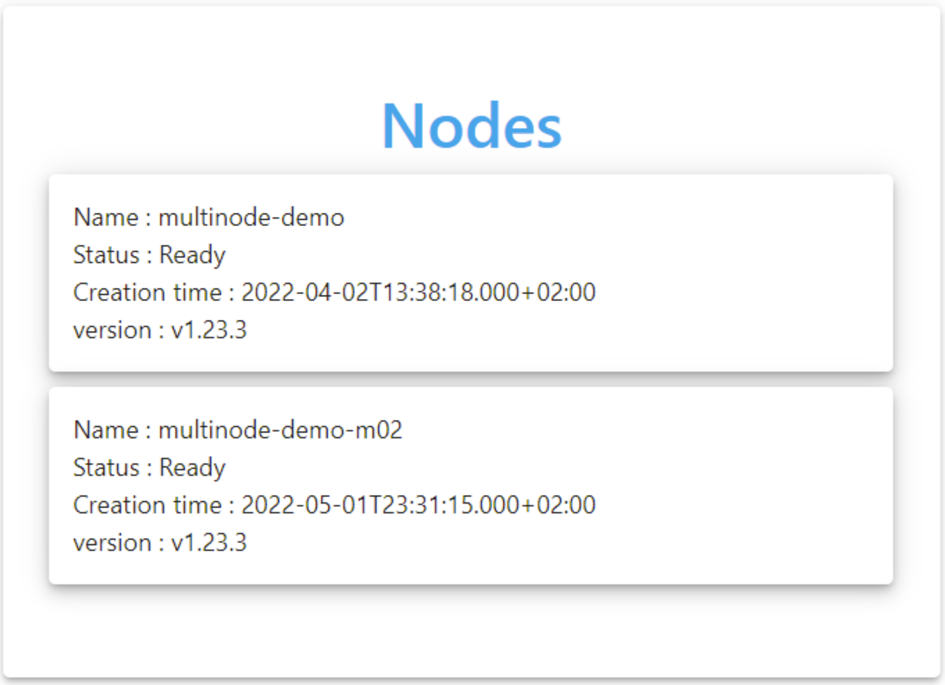


Figure 2.16: Kubernetes Operations

List Nodes

Nodes together form a powerful machine. When programs are deployed onto the cluster, the work is distributed to the individual nodes. Name, status, creation time, and version are displayed for each of the nodes.



Nodes

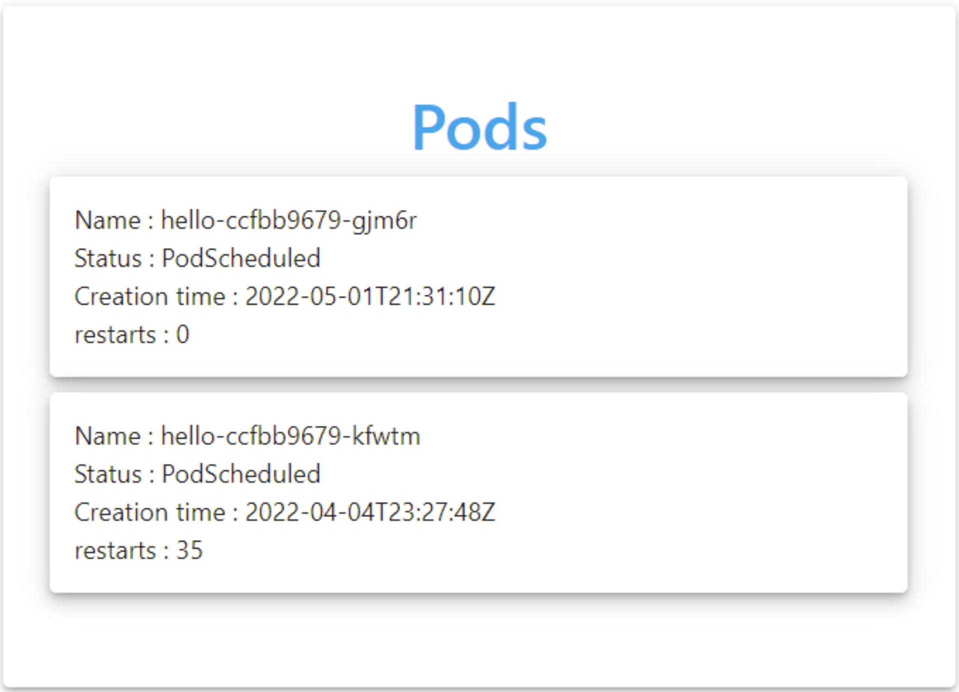
Name : multinode-demo
Status : Ready
Creation time : 2022-04-02T13:38:18.000+02:00
version : v1.23.3

Name : multinode-demo-m02
Status : Ready
Creation time : 2022-05-01T23:31:15.000+02:00
version : v1.23.3

Figure 2.17: List of Nodes

List Pods

Kubernetes creates a Pod to host the application instance when a Resource is deployed. Name, status, creation time, and the number of restarts are listed for each of the Pods.


Pods

Name : hello-ccfbb9679-gjm6r
Status : PodScheduled
Creation time : 2022-05-01T21:31:10Z
restarts : 0

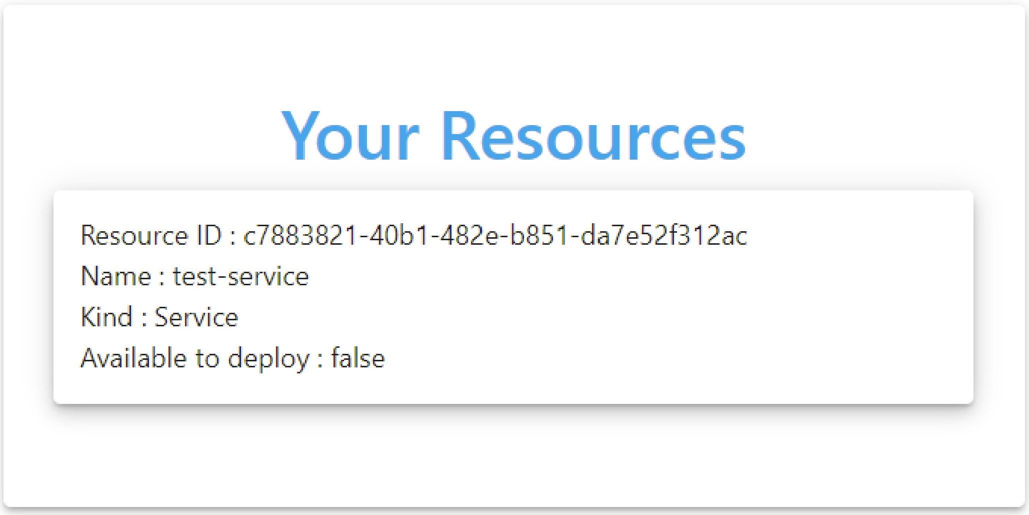
Name : hello-ccfbb9679-kfwtm
Status : PodScheduled
Creation time : 2022-04-04T23:27:48Z
restarts : 35

Figure 2.18: List of Pods

Deploy Resource

The user is able to deploy a Resource based on the following rules specified below:

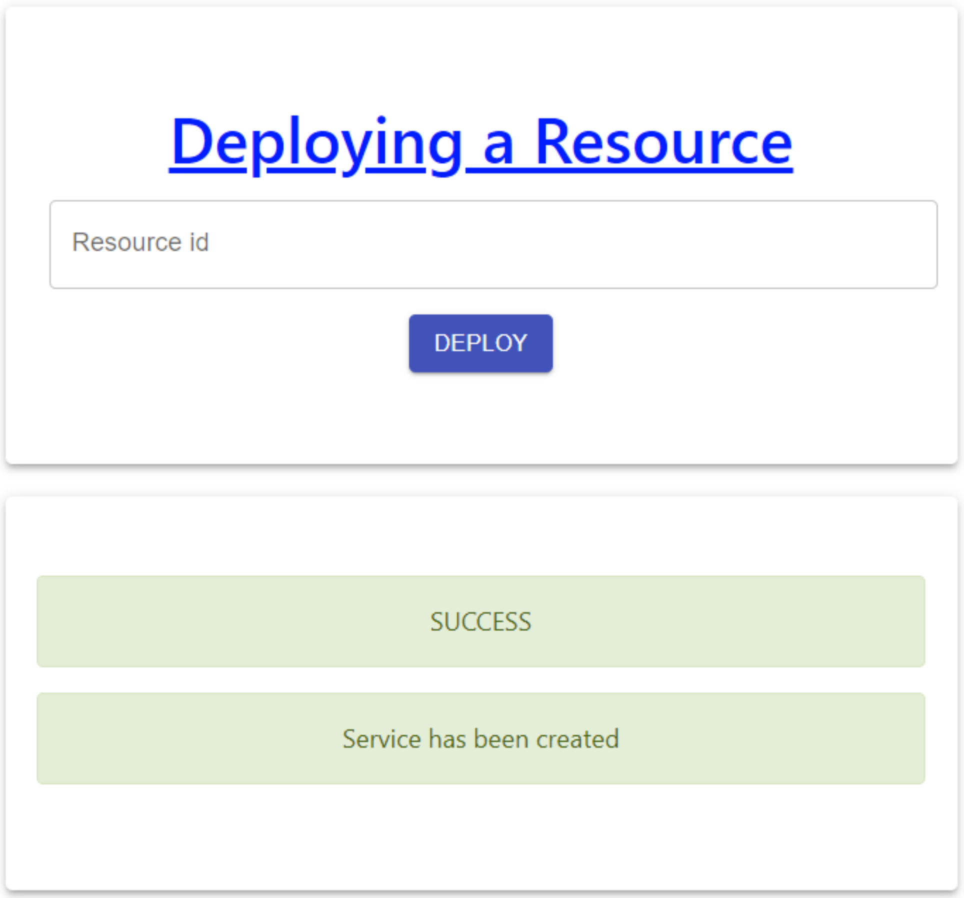
- Resource ID must exist.
- Resource ID must be available to deploy.


Your Resources

Resource ID : c7883821-40b1-482e-b851-da7e52f312ac
Name : test-service
Kind : Service
Available to deploy : false

Figure 2.19: Resource after deployment

If the deployment was successful, the availability of the resource will change from True to False as it is indicated in Figure 2.19.



SUCCESS

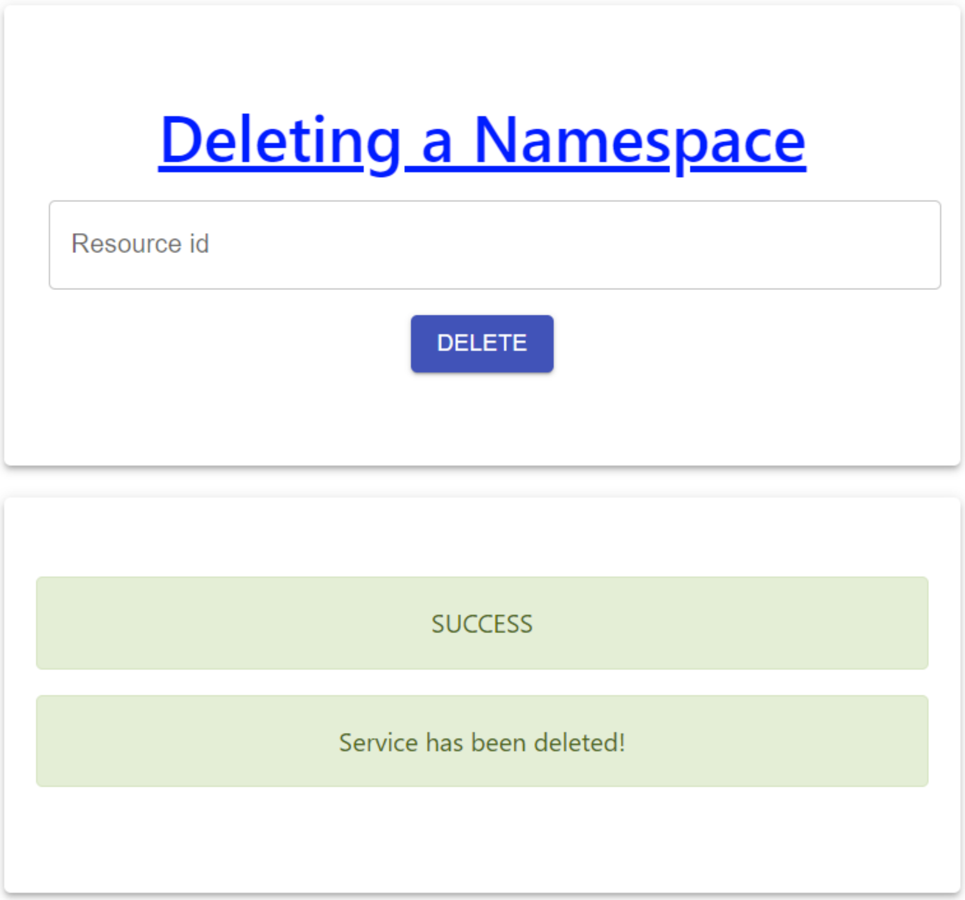
Service has been created

Figure 2.20: Deploying a Resource

Delete Namespace

As the user is able to deploy a Resource, deleting that is also possible if the following rules specified below are obeyed:

- Resource ID must exist.
- Resource ID must be already in deployment state, availability must be False.



SUCCESS

Service has been deleted!

Figure 2.21: Deleting Namespace

2.3.4 Admin

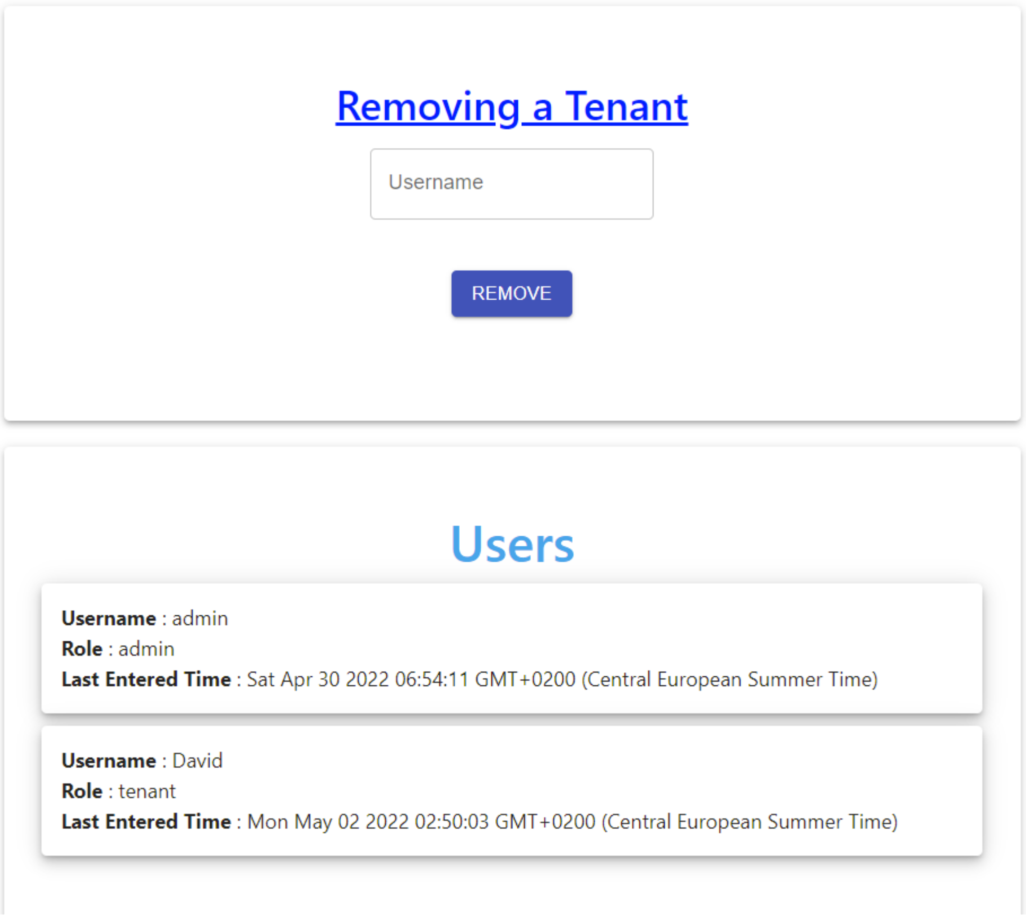
As discussed earlier in Figure 2.4, *Admin* page is designed for administrators to manage the users by listing all information about the users and able to remove them. Admin can log in to the application with the following credentials:

- *Username*: admin
- *Password*: admin

Username, role and last entered time is displayed for each user in *Admin* page. In order to remove a user, there are some restrictions that exists:

- The user must exist
- User must not have any *Resources* or *Packages*
- Admin cannot remove Admin.

If these conditions are fulfilled, the user is deleted successfully, otherwise an error message will be displayed.



Username : admin
Role : admin
Last Entered Time : Sat Apr 30 2022 06:54:11 GMT+0200 (Central European Summer Time)

Username : David
Role : tenant
Last Entered Time : Mon May 02 2022 02:50:03 GMT+0200 (Central European Summer Time)

Figure 2.22: Admin page

This component allows the admin to maintain stability in the application. Most importantly, they ensure a safe and efficient platform where unknown users can be deleted immediately.

In this chapter, we have introduced the components and usage of the project. Creating packages, uploading YAML files that connect relevant packages, and deploying those resources into Kubernetes is the main accomplishment of the user. Various applications are combined in one GUI and with the help of a few clicks, all the life cycle of network functions virtualization can be done in a second. In the next chapter, we present the models, technology, and implementation of this application.

Chapter 3

Developer Documentation

This chapter covers the developer's Manual, which contains the basics and process of development. Initially, a detailed specification of the project is explained. In order to present the usage of the APIs, a brief overview of HTTP requests, their URI locations, and response structures are described. Afterwards, the developer is introduced to main components, storing, and updating models in PostgreSQL, and communication with Kubernetes. Finally, the testing plan and results of the tests are given.

3.1 Prerequisites

Before writing the code, the developer must have a few necessary tools. The installation links can be found in references.

- Project available from GitHub [22]
- PostgreSQL [21]
- Postman [24]
- IntelliJ IDEA development environment [17]

3.2 Specification

This section will specify all the features of the project. Since the unknown user cannot enter the application, the use case diagram in Figure 3.1 is explained from the user and admin point of view.

If the user does not have an account, he must register to the system. After registering the user is able to do 3 kinds of management. These are Package, Resource, and Kubernetes management. After creating a package the user is able to create the resource. By creating a resource, the YAML file is uploaded to the application which later on can be deployed into Kubernetes. Packages can be deleted and updated, YAML files that have been uploaded can be read and deleted, and lastly deployed namespace can be deleted from Kubernetes.

Admin has special credentials to log in to the system. After being able to log in, the list of users is visible. The users can be deleted by the admin as well.

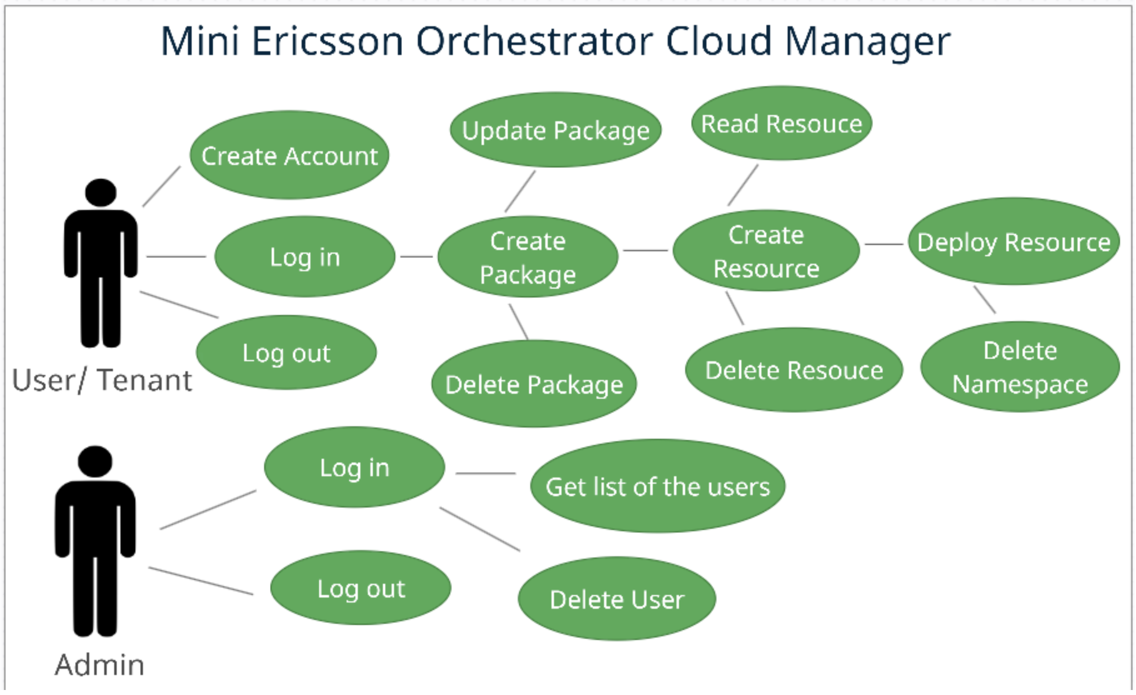


Figure 3.1: Use case diagram of the application

So far we have seen the overview of what our application is capable of doing. In the next section, all of these components are explained in detail.

3.3 Implementation

This section describes how HTTP requests, which are developed by Spring boot, communicate with the front-end tool and database, including how to manipulate data and use models to update the database. Implementation of the Kubernetes API with the usage of the client library is detailed. Then the background of the encryption of the passwords and authentication of the REST APIs are presented from the developer's point of view.

3.3.1 API overview

Mini Ericsson Orchestrator Cloud Manager employs the Representational State Transfer (REST) architectural style. All the API operations use the JavaScript Object Notation (JSON) lightweight data-interchange format. The operations use these HTTP verbs:

1. GET
   Queries for a list of items or for details on a single item. For example, `getUserPackages`, `getUserResources` or `getUsers`.

2. POST
   Requests that create or logging in, out to the system. For example, `createPackage`, `createResource` or `login`.

3. PUT Request that modify a package, resource. For example, `updatePackage`.

4. DELETE Requests that delete a service such as package, user. For example, `deletePackage`, `deleteResource` or `deleteUser`.

URI Location

The Uniform Resource Identifier (URI) location resides in the `packageManager` directory on the Mini Ericsson Orchestrator Cloud Manager host URL. For example to access the order operations enter the URL: http://localhost:8080/packageManager. The verb and the values in the URL determine which operation is executed. This example shows `packageList` operation.

```
GET
http://localhost:8080/packageManager/packageList
```

Code 3.1: Mini Ericsson Orchestrator Cloud Manager API URL

General Request Headers

All inbound requests include a general-header section. For authentication, the header must have an Authorization. The system validates Authorization before it processes a request.

If a POST or PUT operation requires a request body, the header must also include the *Content-Type* property. The *Content-Type* property must be set to *application/json*

Authorization

To send the username and password authentication for every request, use the `Authorization` property. In this case, the user credentials are Base63 encoded.

```
Authorization: Basic <encoded credentials>
Content-Type: application/json
```

Code 3.2: Request Header

- <encoded credentials> - ASCII Base64 encoding of the credentials in the format "username:password".

Response Body Structures

The response includes a status and entity schema. If the response is successful, then the `status` and `requestStatus` is equal to `200` and `SUCCESS` respectively. Otherwise `404` and `ERROR` are the values for `status` and `requestStatus`.

```
{
    "status": 200,
    "entity": {
        "requestStatus": "SUCCESS",
        "messages": [
            {
                "msgText": "Username with 'admin' is authorized!"
            }
        ]
    }
}
```

Code 3.3: Success response example

```
{
    "status": 404,
    "entity": {
        "requestStatus": "ERROR",
        "messages": [
            {
                "msgText": "We couldn't find username with 'Kanan'
                    in the system!"
            }
        ]
    }
}
```

Code 3.4: Error response example

3.3.2 Models

There are SQL scripts `data.sql` and `schema.sql` where tables are created and objects can be inserted or deleted when the application starts running.

```
CREATE TABLE package(
 id varchar(100) NOT NULL,
 name varchar(100) NOT NULL,
 type varchar(100) NOT NULL,
 creationTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
 updatedTime TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
 username varchar(100) NOT NULL,
 PRIMARY KEY (id));

CREATE TABLE users(
 username varchar(100) NOT NULL,
 password varchar(100) NOT NULL,
 role varchar(100) NOT NULL,
 lastEnteredTime TIMESTAMP,
 PRIMARY KEY (username));

CREATE TABLE resources(
 id varchar(100) NOT NULL,
 name varchar(100) NOT NULL,
 kind varchar(100) NOT NULL,
 available boolean NOT NULL,
 username varchar(100) NOT NULL,
 PRIMARY KEY (id));
```

Code 3.5: `schema.sql` file

```
insert into users(username, password, role, lastEnteredTime) values
    ('admin','
    Qr6AgNbWILBmh7UBoHkEJHUH7TVd2eCpgFtB7csx7ABOW2NbOh4xUOWoAHp0oFN5
    ','admin', null);
```

Code 3.6: `data.sql` file

Because `data.sql` file contains an insertion of `admin` user, we can enter an admin role to the application without even registering.

`src/main/java/com/ericsson/packageManager/entity` is a folder that contains all the necessary model classes to make these objects capable of interacting with a database. As we know, each model is represented by a Java class.

User

User is a model in Mini EO-CM application for the authentication system. Basically, it represents the tenants and admins who interact with our project and handle user accounts and authentications. Each user has a username, password, role, and last entered time.

Package

Packages can be created by providing Package type and Package name. There are 5 fields for Package to cooperate with other models. These are `ID`, `Name`, `Type`, `CreationTime`, `UpdatedTime` and `Username`. Here `ID` is the primary key element. Universal Unique Identifier (UUID) is used to uniquely identify a package. Information about the UUID can be found on the website [25].

Resource

The Resource model has the following attributes: `ID`, `Name`, `Kind`, `Available` and `Username`. `Resource ID` describes which package it belongs to. `Kind` shows what type of YAML file has been uploaded while `Available` field indicate whether resource has been deployed to Kubernetes or not.

3.3.3 React Js - Spring

React Js communicates with the Spring Boot application with the help of `fetch` methods. By using different methods of request (POST, PUT, DELETE, GET), URL and content, Spring Boot gets the request, executes the command accordingly, and returns the appropriate response. This response is rendered in React Js and printed out to the screen.

POST request below for creating a Package is described below. Name and type variables are called from input fields. Based on them, a new attribute containing both is initialized. This attribute is converted into JSON format, and it is included in content when fetching that request. URL and request type is specified as well. The response transformed into a readable JSON file. By using new `res` attribute, we can get `responseCode`, `requestStatus` and `message`.

```
        const pac={name, type}
      fetch("packageManager/createPackage",{
        method:"POST",
        headers:{"Content-Type":"application/json"},
        body:JSON.stringify(pac)
      }).then((response) => response.json()).then((res) => {
          let requestStatus = user.entity.requestStatus
          let message = user.entity.messages[0].msgText
      })
```

Code 3.7: POST request via React Js

Another example illustrates a GET request to obtain all the packages as a list. For this `useState` is used in order to store those elements. As usual, URL, HTTP request type is included for `fetch` request. The result was first converted into readable JSON, afterwards, it was assigned to `packages` variable using `setPackage` method. This `packages` will be used subsequently in displaying. `useEffect` hook allows you to perform side effects in your components. Here, fetching the data, and directly updating the DOM are all included.

```
    const[packages,setPackages] = useState([])
    useEffect(()=>{
      fetch("packageManager/userPackageList").then(res=>res.json())
          .then((result)=>{
        setPackages(result);
      })
    },[])
```

Code 3.8: GET request via React Js

3.3.4 Interaction with Database

Spring Boot project connects to a data source using configurations to the one shown below in the project's `application.properties` file. These configurations give Spring Boot access to a PostgreSQL database.

```
spring.datasource.url=jdbc:postgresql://localhost:5454/postgres
spring.datasource.username=postgres
spring.datasource.password=postgres
```

Code 3.9: `application.properties`

The next part is about how Packages can be inserted, updated, listed, or deleted from the database using SQL statements in Java.

Insertion

The JDBC template is the core API through which accessing most of the functionalities is possible such as running statements, procedure calls, and iterating over the `ResultSet` and returning objects. Since Package has named parameters, the `NamedParameterJdbcTemplate` framework for JDBC template can be used.

Initially, an SQL statement is created with column names. Using the `MapSqlParameterSource` we can provide all the values for the named parameters. Afterwards, `update` method is executed.

```
public String insertPackage(Package aPackage) {
    final String sql = "insert into package(id, name, type,
        creationTime, updatedTime, username) values(:id,:name,:type
        ,:creationTime,:updatedTime,:username)";

    KeyHolder holder = new GeneratedKeyHolder();
    SqlParameterSource param = new MapSqlParameterSource()
        .addValue("id", aPackage.getId())
        .addValue("name", aPackage.getName())
        .addValue("type", aPackage.getType())
        .addValue("creationTime", new Timestamp(System.
            currentTimeMillis()))
        .addValue("updatedTime", new Timestamp(System.
            currentTimeMillis()))
        .addValue("username", aPackage.getUsername());
    template.update(sql,param, holder);
}
```

Code 3.10: `insertPackage` method

Modification

`updatePackage` is a bit different from `insertPackage`, because Package with `ID` already exists. Therefore just changing SQL statement is enough.

```
public void updatePackage(Package aPackage) {
        final String sql = "update package set name=:name, type=:
            type, updatedTime=:updatedTime, username=:username where
             id=:id";

        KeyHolder holder = new GeneratedKeyHolder();
        SqlParameterSource param = new MapSqlParameterSource()
                .addValue("id", aPackage.getId())
                .addValue("name", aPackage.getName())
                .addValue("type", aPackage.getType())
                .addValue("updatedTime", new Timestamp(System.
                    currentTimeMillis()))
                .addValue("username", aPackage.getUsername());
        template.update(sql,param, holder);
    }
```

Code 3.11: `updatePackage` method

Deletion

By providing Package ID and creating a SQL statement will be followed by `executeUpdate` method. This should be done `PreparedStatementCallback` because a single `executeUpdate` call or repeated `executeUpdate` calls with varying parameters must be executed there.

```
public void deletePackage(Package aPackage){
        final String sql = "delete from package where id=:id";

        Map<String, Object> map = new HashMap<String, Object>();
        map.put("id", aPackage.getId());

        template.execute(sql, map, new PreparedStatementCallback<
            Object>() {
            @Override
            public Object doInPreparedStatement(PreparedStatement
                ps)
                    throws SQLException, DataAccessException {
                return ps.executeUpdate();
            }
        });
    }
```

Code 3.12: `deletePackage` method

Get

`RowMapper` interface is used for querying results to Java objects. For every row returned by the query, Spring uses the row mapper to generate the Java bean.

```
public List<Package> findAll() {
        return template.query("select * from package", new
            PackageRowMapper());
    }
```

Code 3.13: `findAll` method

```
public Package mapRow(ResultSet rs, int arg1) throws SQLException {
        Package aPackage = new Package();
        aPackage.setId(rs.getString("id"));
        aPackage.setName(rs.getString("name"));
        aPackage.setType(rs.getString("type"));
        aPackage.setCreationTime(rs.getTimestamp("creationTime"));
        aPackage.setUpdatedTime(rs.getTimestamp("updatedTime"));
        aPackage.setUsername(rs.getString("username"));

        return aPackage;
    }
```

Code 3.14: `mapRow` method

The same logic is also applied to Resource and User models. So far we have learned how packages are stored in the database, and how SQL statements are sent to the PostgreSQL to update or delete the package. Now we can dive into the implementation of the file uploading.

3.3.5 File management

In this section, we aim to observe how YAML files are getting uploaded and stored. First of all, when the application starts running, `resources` folder gets created in the root directory.

```java
    @Bean
    CommandLineRunner init(StorageService storageService) {
        return (args) -> {
            storageService.init();
        };
    }
    //
    public void init() {
        Files.createDirectories(rootLocation);
    }
```

Code 3.15: Bean to create `resources` folder

We need to register a `MultipartConfigElement` class to upload files with Servlet containers. It is auto-configured in Spring Boot.

```java
@PostMapping("/{id:.+}")
public Response createResource(@PathVariable String id,
    @RequestParam("file") MultipartFile file, @RequestParam("kind")
    String kind) {
    //
    storageService.store(id, file, user.getUsername());
    //
}
```

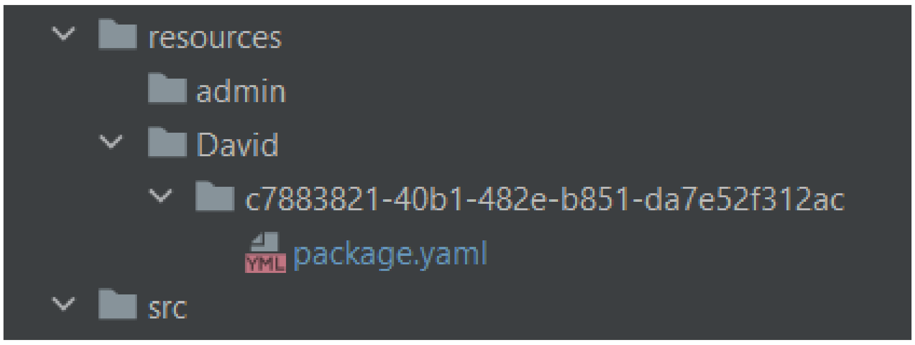
Code 3.16: `createResource` method

Based on username and Package ID nested folders are created, if there exists one already it will be replaced. Default name for the file will be `package.yaml`. After the directories and file is created, the content of the original file is copied into the new one.

```java
    try {
        new File(this.rootLocation.toString() + "\\" + username + "
            \\" + id).mkdirs();
        Path destinationFile = this.rootLocation.resolve(username).
            resolve(id).resolve(Paths.get("package.yaml")).normalize
            ().toAbsolutePath();
        try (InputStream inputStream = file.getInputStream()) {
            Files.copy(inputStream, destinationFile,
                StandardCopyOption.REPLACE_EXISTING);
        }
    } catch (IOException e) {
        throw new StorageException("Failed to store file.", e);
    }
```

Code 3.17: Storing a file

As you can see in the figure below `package.yaml` has been created in nested David's folder and Package ID folder. If David creates a new Resource with a different Package ID, a new folder under the David folder will be added.



```
resources
    admin
    David
        c7883821-40b1-482e-b851-da7e52f312ac
            package.yaml
src
```

Figure 3.2: Example Resources Folder

Files can be deleted as well, in this case, both relevant `package.yaml` and the Package ID folder are deleted. When the admin removes the tenant, the folder related to that user is deleted automatically.

3.3.6 Kubernetes-Spring

The API server is the core of the Kubernetes control plane. Clusters and external components communicate with each other using HTTP API. Pods, Nodes, and Namespaces are queried and their status can be changed. The kubectl command-line interface is a classic way for operating such requests. However, the possibility of accessing API directly using REST calls leads us to client libraries.

Officially-supported `Java Kubernetes client library` [26] often handle common tasks. The library added to the project as well. It can be found in `lib/java` folder. The following part describes deploying and deleting namespace APIs.

Initially, we should create the `ApiClient` and then the API stub instance. `Service` resources are part of the `CoreV1 API`, hence `CoreV1Api` instance is created, which will be used to invoke the cluster's API server. As a next step, we instantiate a `V1Service` instance based on the YAML file. Consequently, the process of filling all the properties is done.

Lastly, the API stub that is created should be called. This serializes our resource object and POST the request to the server. `V1Service` object contains `metadata` and `status` fields. Its `status` field can be checked when it is finished.

```java
ApiClient client = Config.defaultClient();
Configuration.setDefaultApiClient(client);
//
Yaml yaml = new Yaml(new Constructor(V1Service.class));
V1Service yamlSvc = (V1Service) yaml.load(inputStream);

CoreV1Api api = new CoreV1Api();
V1Service createResult = api.createNamespacedService("default",
    yamlSvc, null, null, null);
//
```

Code 3.18: Create Service Namespace

Regarding the deletion, `deleteNamespacedService` method is used to remove the service using default options for this specific kind of resource. The developer can use the last parameter - `V1DeleteOptions` to specify a grace period and cascading behavior for any dependent resources.

```java
ApiClient client = Config.defaultClient();
Configuration.setDefaultApiClient(client);
//
try {
    CoreV1Api api = new CoreV1Api();
    V1DeleteOptions body = new V1DeleteOptions();
    api.deleteNamespacedService(getName(id), "default", null,
        null, 56, true, null, body);
} catch (ApiException e){
    return ErrorResponse("Service deletion was not succeed");
}
//
```

Code 3.19: Delete Service Namespace

Job workload resources are part of the `Batch API` while the rest (`Deployment, ReplicaSet, ControllerRevision, StatefulSet`) are using `Apps API`.

Here the manipulation between Kubernetes resources and the Java Kubernetes API library has been explained.

3.3.7 Jasypt

`Jasypt` is a java library that allows the developer to add basic encryption capabilities to projects with minimum effort. It is also useful for people who do not have deep knowledge of cryptography. The benefits of Jasypt are the followings:

- standard encryption techniques with high security able to encrypt both bidirectional and unidirectional.
- Suitable for integration into Spring Boot applications and transparently capable of being integrated with Spring Security.
- Having specific features for high-effective encryption in multi-processor/multicore systems.

To use `Jasypt`, include dependency into your `pom.xml` file as follows:

```
<dependency>
    <groupId>org.jasypt</groupId>
    <artifactId>jasypt</artifactId>
    <version>1.9.3</version>
    <scope>compile</scope>
</dependency>
```

Code 3.20: Jasypt dependency

To encrypt the password, you can use `StrongPasswordEncryptor` method:

```
    public static String encryptPassword(String inputPassword) {
        StrongPasswordEncryptor encryptor = new
            StrongPasswordEncryptor();
        return encryptor.encryptPassword(inputPassword);
    }
```

Code 3.21: `encryptPassword` method

Note: The encrypted password is changed every time `encryptPassword` is called. The `checkPassword` method can still check whether the unencrypted password still matches each of the encrypted passwords. In order to inspect the input password against the encrypted password, `checkPassword` method should be used:

```
public static boolean checkPassword(String inputPassword, String
   encryptedStoredPassword) {
        StrongPasswordEncryptor encryptor = new
           StrongPasswordEncryptor();
        return encryptor.checkPassword(inputPassword,
           encryptedStoredPassword);
    }
```

Code 3.22: `checkPassword` method

3.3.8 Basic Authentication

Various API requests can be sent using Postman. This is not possible in GUI as the user needs to log in to the system before handling packages or resources. Basic authentication is needed to secure REST APIs created inside a Spring boot application. Thus, the secured REST APIs will ask for authentication details before giving access to the data.

Spring security related jar files are included `spring-boot-starter-security` dependency and simplest way to connect is adding it to `pom.xml`

```
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
```

Code 3.23: `spring-boot-starter` dependency

As a next step, we need to configure `WebSecurityConfigurerAdapter` utility class to allow authorization support in all spring boot REST APIs which will require the user to be authenticated prior to accessing any configured URLs within the application.

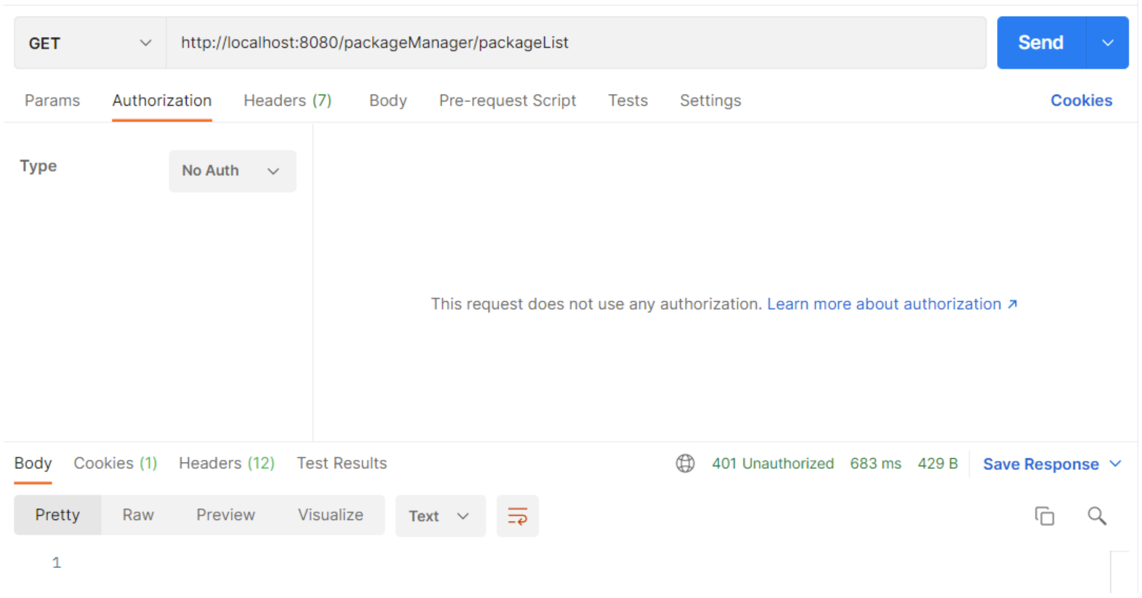
```java
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable().authorizeRequests().anyRequest().
            authenticated().and().httpBasic();
    }

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth)
        throws Exception {
        auth.inMemoryAuthentication()
                .withUser("admin")
                .password("{noop}password")
                .roles("USER");
    }
}
```

Code 3.24: `SecurityConfig` class

Upon sending an API request `basic-auth` must be attached to the header with the username and password combination specified in `SecurityConfig` class. By this user can access the rest API response. Otherwise `401 Unauthorized` response will appear.

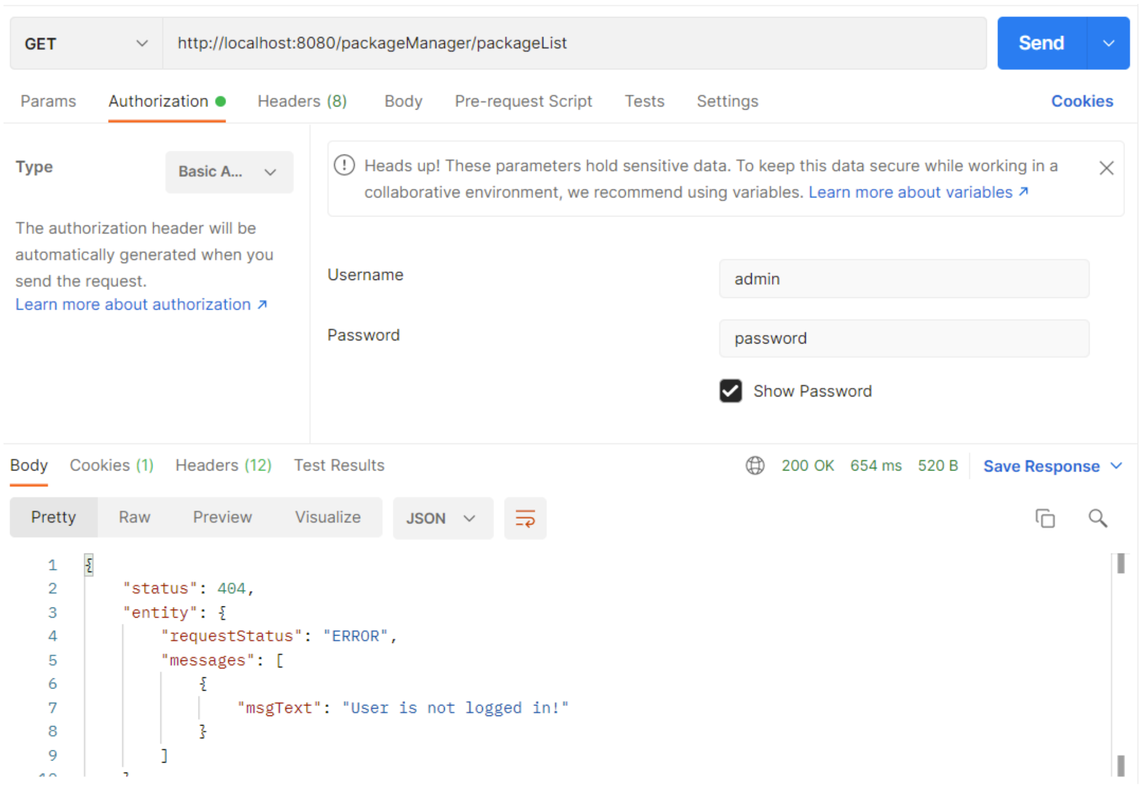

GET http://localhost:8080/packageManager/packageList

Type: No Auth

This request does not use any authorization. Learn more about authorization ↗

401 Unauthorized 683 ms 429 B

Figure 3.3: Sending API request without auth



http://localhost:8080/packageManager/packageList

Type: Basic A...

The authorization header will be automatically generated when you send the request.
Learn more about authorization ↗

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about variables ↗

Username: admin

Password: password

Show Password

200 OK 654 ms 520 B

```
{
    "status": 404,
    "entity": {
        "requestStatus": "ERROR",
        "messages": [
            {
                "msgText": "User is not logged in!"
            }
        ]
```

Figure 3.4: Sending API request with auth

3.4 Testing

Testing is one of the essential parts of the application to prove its correctness against multiple inputs for various API requests. Testing has been divided into 3 files under `src/test/java/com/ericsson/packageManager` folder. 72 integration tests have been written in order to check if each of the Package, Resource, and User models work correctly and synchronously. Integration tests are written with the Spring MVC framework.

3.4.1 Spring MVC

Integration testing plays a crucial role in the application development cycle by confirming the end-to-end behavior of a system. Without explicitly starting a Servlet container, the Spring MVC test framework allows the developer to write and run integration tests that test controllers. `MockMVC` class is part of the Spring MVC test framework. It will be used alongside Spring boot's `WebMvcTest` class to execute Junit test cases which check REST controller methods.

3.4.2 Test configuration

First and foremost, the dependency must be included in the `pom.xml`.

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
</dependency>
```

Code 3.25: Spring Boot Test dependency

A `JUnit` test class can be utilized with the below-given configuration to test the Spring MVC controller requests and responses.

```java
@RunWith(SpringRunner.class)
@WebMvcTest
@AutoConfigureMockMvc
public class UserApiTest {

    @MockBean
    private UserService userService;

    @MockBean
    private StorageService storageService;

    @MockBean
    private PackageService packageService;

    @MockBean
    private ResourceService resourceService;

    @Autowired
    PackageManagerController packageManagerController;

    @Autowired
    private MockMvc mockMvc;
    //
}
```

Code 3.26: `UserApiTest` class configuration

- `SpringRunner` is an alias for the `SpringJUnit4ClassRunner`. This is used for providing `Spring TestContext Framework` functionality which will support annotations and classes. It is a custom extension of JUnit's `BlockJUnit4ClassRunner`.

- `@WebMvcTest` annotation is used for Spring MVC tests. It impairs full auto-configuration, but applies configuration relevant to MVC tests.
- The `WebMvcTest` annotation auto-configure `MockMvc` instance as well.
- `@MockBean` annotation add mock objects to the Spring application context. This mock is replacing any existing bean of the same type in the application context

Post and Put requests testing

In this code snippet below, a POST request with URL has been performed which has content JSON. After request performed out, `Matchers` are checked based on the parameter given such as `code`, `requestStatus` and `messages`. Whole data as JSON parameter is assigned to `result` variable. This helps us to check all the messages one by one. When it comes to the PUT request, it is the same as POST, `MockMvcRequestBuilders.put(url)` is the only difference that needs to be changed.

```
MvcResult result = mockMvc.perform(MockMvcRequestBuilders.post(url)
    .header(HttpHeaders.AUTHORIZATION,"Basic " + Base64Utils.
       encodeToString("admin:password".getBytes()))
    .content(content)
    .contentType(MediaType.APPLICATION_JSON))
    .andExpect(MockMvcResultMatchers.status().isOk())
    .andExpect(MockMvcResultMatchers.jsonPath("$.status", Is.is(
       code)))
    .andExpect(MockMvcResultMatchers.jsonPath("$.entity.
       requestStatus", Is.is(requestStatus)))
    .andExpect(MockMvcResultMatchers.jsonPath("$.entity.messages").
       isArray())
    .andExpect(MockMvcResultMatchers.jsonPath("$.entity.messages",
       hasSize(messages.size())))
    .andExpect(MockMvcResultMatchers.content().contentType(
       MediaType.APPLICATION_JSON)).andReturn();

List<String> expectedMessages = new ArrayList<>();
```

```
for(int i = 0; i < messages.size(); i++){
    String path = "$.entity.messages[" + i + "].msgText";
    expectedMessages.add(JsonPath.read(result.getResponse().
       getContentAsString(), path));
}
for (String s : messages) {
    assertThat(expectedMessages, hasItem(s));
}
```

Code 3.27: POST Request Testing

Delete Requests Testing

Testing the DELETE requests is the same as the POST request mentioned above. The only difference is that there is no JSON content to add to the header.

```
mockMvc.perform(MockMvcRequestBuilders.delete(url)
    .header(HttpHeaders.AUTHORIZATION,"Basic " + Base64Utils.
       encodeToString("admin:password".getBytes()))
    .andExpect(MockMvcResultMatchers.status().isOk())
    .andExpect(MockMvcResultMatchers.jsonPath("$.status", Is.is(
       code)))
    .andExpect(MockMvcResultMatchers.jsonPath("$.entity.
       requestStatus", Is.is(requestStatus)))
    .andExpect(MockMvcResultMatchers.jsonPath("$.entity.messages").
       isArray())
    .andExpect(MockMvcResultMatchers.jsonPath("$.entity.messages",
       hasSize(1)))
    .andExpect(MockMvcResultMatchers.jsonPath("$.entity.messages
       [0].msgText", Is.is(message)))
    .andExpect(MockMvcResultMatchers.content().contentType(
       MediaType.APPLICATION_JSON));
```

Code 3.28: DELETE Request Testing

Get Requests Testing

As we already know, the response coming from GET request is divided into 2 types. When it is successful, the response is the list of requested objects such as Packages, Users. After getting resulted variable, objects are examined. Otherwise, the usual error response which is mentioned in section 3.3.1 appears and it is handled as the previous requests.

```
MvcResult result = mockMvc.perform(MockMvcRequestBuilders.get(url)
    .header(HttpHeaders.AUTHORIZATION,"Basic " + Base64Utils.
       encodeToString("admin:password".getBytes())))
    .andExpect(MockMvcResultMatchers.status().isOk())
    .andExpect(MockMvcResultMatchers.content().contentType(
       MediaType.APPLICATION_JSON)).andReturn();
        //
```

Code 3.29: GET Request Testing Success Response

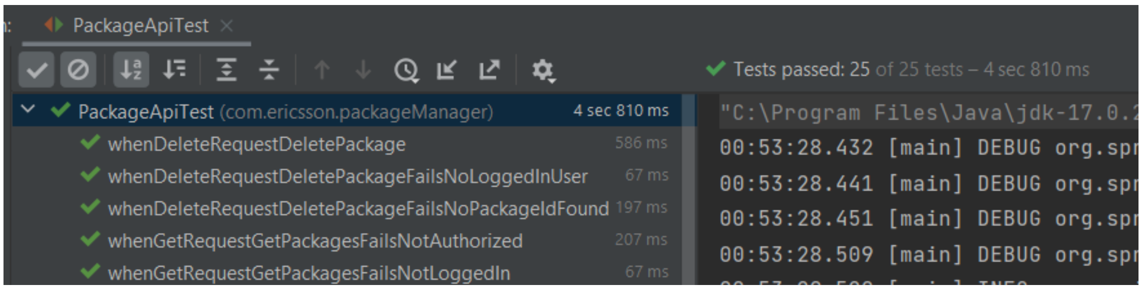
```
mockMvc.perform(MockMvcRequestBuilders.get(url)
    .header(HttpHeaders.AUTHORIZATION,"Basic " + Base64Utils.
       encodeToString("admin:password".getBytes())))
    .andExpect(MockMvcResultMatchers.status().isOk())
    .andExpect(MockMvcResultMatchers.jsonPath("$.status", Is.is(
       code)))
    .andExpect(MockMvcResultMatchers.jsonPath("$.entity.
       requestStatus", Is.is(requestStatus)))
    .andExpect(MockMvcResultMatchers.jsonPath("$.entity.messages").
       isArray())
    .andExpect(MockMvcResultMatchers.jsonPath("$.entity.messages",
       hasSize(1)))
    .andExpect(MockMvcResultMatchers.jsonPath("$.entity.messages
       [0].msgText", Is.is(message)))
    .andExpect(MockMvcResultMatchers.content().contentType(
       MediaType.APPLICATION_JSON));
```

Code 3.30: GET Request Testing Failure Response

3.4.3 Testing results

In conclusion, with the help of the Spring MVC test framework, all the edge cases of possible API requests have been checked. All of the 72 integration tests in 3 different test classes passed the verification. See Figure 3.5, Figure 3.6, and Figure 3.7.

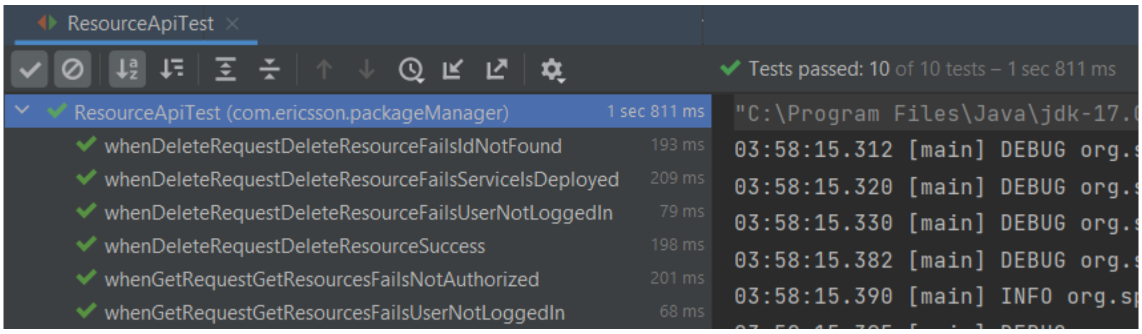
Using template methods for different HTTP request types made it easier and quicker to write effective test cases. All of the successful tests proved that our application can run smoothly without any errors and exceptions.



```
PackageApiTest
✓ Tests passed: 25 of 25 tests – 4 sec 810 ms
✓ PackageApiTest (com.ericsson.packageManager)                4 sec 810 ms
    ✓ whenDeleteRequestDeletePackage                               586 ms
    ✓ whenDeleteRequestDeletePackageFailsNoLoggedInUser             67 ms
    ✓ whenDeleteRequestDeletePackageFailsNoPackageIdFound          197 ms
    ✓ whenGetRequestGetPackagesFailsNotAuthorized                  207 ms
    ✓ whenGetRequestGetPackagesFailsNotLoggedIn                     67 ms

"C:\Program Files\Java\jdk-17.0.2
00:53:28.432 [main] DEBUG org.spr
00:53:28.441 [main] DEBUG org.spr
00:53:28.451 [main] DEBUG org.spr
00:53:28.509 [main] DEBUG org.spr
```

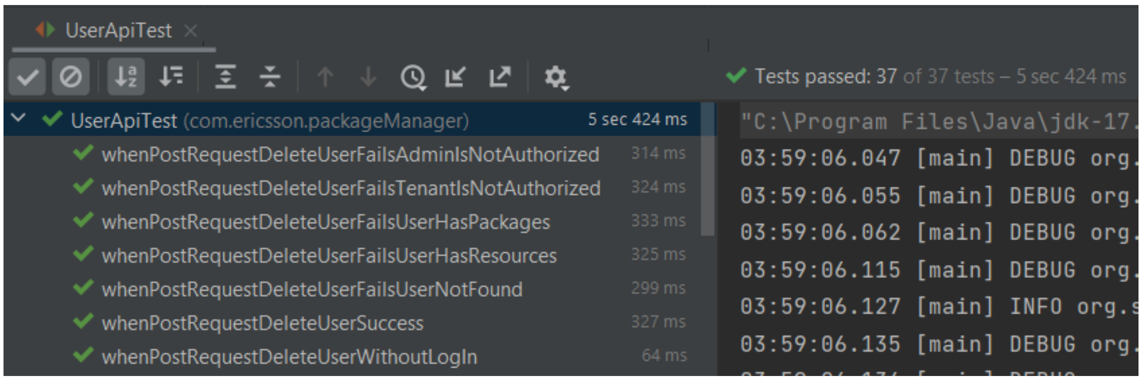
Figure 3.5: `PackageApiTest` test class result



```
ResourceApiTest
✓ Tests passed: 10 of 10 tests – 1 sec 811 ms
✓ ResourceApiTest (com.ericsson.packageManager)               1 sec 811 ms
    ✓ whenDeleteRequestDeleteResourceFailsIdNotFound               193 ms
    ✓ whenDeleteRequestDeleteResourceFailsServiceIsDeployed        209 ms
    ✓ whenDeleteRequestDeleteResourceFailsUserNotLoggedIn           79 ms
    ✓ whenDeleteRequestDeleteResourceSuccess                       198 ms
    ✓ whenGetRequestGetResourcesFailsNotAuthorized                 201 ms
    ✓ whenGetRequestGetResourcesFailsUserNotLoggedIn                68 ms

"C:\Program Files\Java\jdk-17.0
03:58:15.312 [main] DEBUG org.s
03:58:15.320 [main] DEBUG org.s
03:58:15.330 [main] DEBUG org.s
03:58:15.382 [main] DEBUG org.s
03:58:15.390 [main] INFO org.sp
```

Figure 3.6: `ResourceApiTest` test class result



```
UserApiTest
✓ Tests passed: 37 of 37 tests – 5 sec 424 ms
✓ UserApiTest (com.ericsson.packageManager)                   5 sec 424 ms
    ✓ whenPostRequestDeleteUserFailsAdminIsNotAuthorized           314 ms
    ✓ whenPostRequestDeleteUserFailsTenantIsNotAuthorized          324 ms
    ✓ whenPostRequestDeleteUserFailsUserHasPackages                333 ms
    ✓ whenPostRequestDeleteUserFailsUserHasResources               325 ms
    ✓ whenPostRequestDeleteUserFailsUserNotFound                   299 ms
    ✓ whenPostRequestDeleteUserSuccess                             327 ms
    ✓ whenPostRequestDeleteUserWithoutLogIn                         64 ms

"C:\Program Files\Java\jdk-17.
03:59:06.047 [main] DEBUG org.
03:59:06.055 [main] DEBUG org.
03:59:06.062 [main] DEBUG org.
03:59:06.115 [main] DEBUG org.
03:59:06.127 [main] INFO org.s
03:59:06.135 [main] DEBUG org.
```

Figure 3.7: `UserApiTest` test class result

The implementation and testing phases were explained in this chapter. Now it is time to move to the conclusion to explain what I have gained through this project and possible additional work in the future.

Conclusion and future work

This thesis aimed at presenting creating an integrated platform where the life cycle of Network Functions Virtualization is handled in a flexible way by providing a secure user interface and open APIs for all management services.

One of the biggest improvements that can be made in the project is to add OpenStack VIM alongside Kubernetes. Openstack is a cost-effective extension of the public cloud infrastructure which enables organizations to optimize maintenance costs and service providers to create an infrastructure competitive to hyperscalers. In this case *HOT* type packages will be supported by the project as well. *HOT* packages can be deployed to OpenStack infrastructure and operations on those packages can take place.

From the developer's point of view, one of the most significant advantages of using Spring boot is that it can quickly set up and run standalone web applications or microservices. Spring boot also removes the necessary code and provides us with a built-in server to make the implementation smooth. Free dependent management helped me to add libraries such as `Kubernetes Java client library` and `Jasypt`.

This project achieves the initial specifications and works as expected. By using this application users are able to create and manage CNF packages, upload YAML files and operate components on VimZone services by deploying resources. Admin can control tenants and remove them when it is needed. The product is secured from any unauthorized access.

To conclude, this thesis is not only my favorite but also a great opportunity to learn fundamentals of the cloud computing and the technologies that are used such as Docker, and Kubernetes. I was able to learn how to build and share containerized applications and microservices by using Docker. Furthermore, I learned several functionalities of Kubernetes which manages easier to have multiple, independent services, and to host them at scale than with a traditional infrastructure

Acknowledgements

I would like to thank the following people, without whom I would not have been able to complete this thesis!

First and foremost, I would like to thank my supervisor, Prof. Viktória Zsók, for assisting and guiding me through the process of writing my thesis work.

I would like to extend my deepest gratitude to Tamás Balogh (Ericsson) and Dávid Kovács (Ericsson) for helping with the design of the application, as well as dedicating time to the meetings, answering all of my questions during the development phase, debugging the code with me, and giving advice on how to further polish the project for the submission.

Special thanks to Ericsson's Discovery team for their support during the semester. And to my line manager at Ericsson, Ákos Princzinger, for giving me the flexibility and time needed to work on this thesis.

Bibliography

[1] Balamurali Thekkedath. *Network Functions Virtualization for dummies*. John Wiley and Sons, Inc, 1972.

[2] *What is the cloud?* URL: `https://www.cloudflare.com/en-gb/learning/cloud/what-is-the-cloud/` (visited on 05/13/2022).

[3] Sai Vennam. *Cloud Computing*. URL: `https://www.ibm.com/au-en/cloud/learn/cloud-computing/` (visited on 05/13/2022).

[4] *What is Cloud Computing?* URL: `https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/` (visited on 05/13/2022).

[5] *Use containers to Build, Share and Run your applications*. URL: `https://www.docker.com/resources/what-container/` (visited on 05/13/2022).

[6] *What is Kubernetes?* URL: `https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/` (visited on 05/13/2022).

[7] *Why Spring?* URL: `https://spring.io/why-spring` (visited on 05/13/2022).

[8] *Java Spring Boot*. URL: `https://www.ibm.com/cloud/learn/java-spring-boot/` (visited on 05/13/2022).

[9] *About PostgreSQL*. URL: `https://www.postgresql.org/about/` (visited on 05/13/2022).

[10] *What is the PostgreSQL?* URL: `https://aws.amazon.com/rds/postgresql/what-is-postgresql/` (visited on 05/13/2022).

[11] *React (JavaScript library)*. URL: `https://en.wikipedia.org/wiki/React_(JavaScript_library)/` (visited on 05/13/2022).

[12] *React Bootstrap*. URL: `https://react-bootstrap.github.io/` (visited on 05/13/2022).

[13] *What is Material UI in React?* URL: https://www.educative.io/edpresso/what-is-material-ui-in-react/ (visited on 05/13/2022).

[14] *HTML Introduction.* URL: https://www.w3schools.com/html/html_intro.asp (visited on 05/13/2022).

[15] *CSS.* URL: https://en.wikipedia.org/wiki/CSS (visited on 05/13/2022).

[16] *Java Development Kit.* URL: https://www.oracle.com/java/technologies/downloads/ (visited on 05/13/2022).

[17] *IntelliJ IDEA.* URL: https://www.jetbrains.com/idea/download/ (visited on 05/13/2022).

[18] *Docker Dekstop.* URL: https://docs.docker.com/desktop/windows/install/ (visited on 05/13/2022).

[19] *Node Js.* URL: https://nodejs.org/en/download/ (visited on 05/13/2022).

[20] *Minikube.* URL: https://minikube.sigs.k8s.io/docs/start/ (visited on 05/13/2022).

[21] *PostgreSQL.* URL: https://www.postgresql.org/download/ (visited on 05/13/2022).

[22] *Mini EO-CM.* URL: https://github.com/rasulkhanbayov/Thesis/tree/master (visited on 05/13/2022).

[23] *RFC 1035 Label Names.* URL: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#rfc-1035-label-names (visited on 05/13/2022).

[24] *Postman.* URL: https://www.postman.com/downloads/ (visited on 05/13/2022).

[25] *Universal Unique Identifier.* URL: https://www.techtarget.com/searchapparchitecture/definition/UUID-Universal-Unique-Identifier (visited on 05/13/2022).

[26] *Java Kubernetes Client library.* URL: https://github.com/kubernetes-client/java/ (visited on 05/13/2022).

List of Figures

List of Codes