UNIVERSITY OF THE AEGEAN

SCHOOL OF ENGINEERING

DEPARTMENT OF INFORMATION AND COMMUNICATION SYSTEMS ENGINEERING

POSTGRADUATE STUDIES PROGRAM

INFORMATION AND COMMUNICATION SYSTEMS

Development of a multi-cloud Simulation-as-a-Service application

THESIS

by

Eleni Papachristou

Supervisor:
Assoc. Prof. Dr. Kritikos Kyriakos

Examination committee members:
Prof. Dr. Kormentzas Georgios
Assoc. Prof. Dr. Skoutas Dimitrios

Samos, August 2025

This page is intentionally blank.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Dr. Kyriakos Kritikos for his guidance and feedback throughout my thesis development.

I would also like to thank my family for their encouragement and support during this whole period.

ELENI PAPACHRISTOU

Department Of Information and Communication Systems Engineering

UNIVERSITY OF THE AEGEAN

This page is intentionally blank.

Table of Contents

List of Figures

List of Tables

Acronyms

API	Application Programming Interface
CHIPS	Cookies Having Independent Partitioned State
CORS	Cross-Origin Resource Sharing
CSRF	Cross-Origin Resource Sharing
DI	Dependency Injection
DTO	Data Transfer Objects
ENV	Environment Variable
ER	Entity Relationship
GUI	Graphical User Interface
HATEOAS	Hypermedia As The Engine Of Application State
JWT	JSON Web Token
MSSQL	Microsoft SQL Server
OS	Operating System
PDES	Parallel and Distributed Discrete-Event Simulation
QoS	Quality of Service
RBAC	Role-based Access Control
REST	Representational State Transfer
SA	System Administrator
SOAP	Simple Object Access Protocol
UI	User Interface
VU	Virtual Users
WSL	Windows Subsystem for Linux
YAML	YAML Ain't Markup Language

Abstract

This thesis concerns the development of a scalable Software-as-a-Service service through modern frontend, backend and cloud computing technologies which will allow users to run simulations in different clouds. In particular, users will provide as input a configuration of the simulation to be run and then the service will take over running this simulation on the cloud of the user's choice. The execution of the simulation will be based on code whose Internet location is specified in the simulation configuration. After running the simulation, the service will produce a full report of the simulation and display it to the user. The user will also have the ability to download the report to their local system. Finally, the service should display to the user the progress of a simulation execution as well as a history of simulation executions performed by the user.

The main benefits of such a service are the independence from specific simulation tools as it supports every user's preferred simulation project hosted on GitHub, the compatibility with various cloud providers, as well as the automated orchestration, deployment and scaling of the simulation according to the current workload needs. It also offers a user-friendly simulation management experience as it does not require expertise around cloud deployment and orchestration; the only thing required from the user is the existence of an account in at least one cloud provider.

Keywords: Software-as-a-Service, Cloud computing, Simulation, Autoscaling, Docker, Kubernetes, Master/Slave architecture, Terraform, YAML, .Net, JWT, JSON, Minikube.

Περίληψη

Η παρούσα διπλωματική εργασία αφορά την ανάπτυξη μιας κλιμακώσιμης υπηρεσίας Λογισμικού ως Υπηρεσία (Software-as-a-Service) μέσω σύγχρονων τεχνολογιών frontend, backend και Υπολογιστικού Νέφους (cloud computing), η οποία θα επιτρέπει στους χρήστες να εκτελούν προσομοιώσεις σε διαφορετικά νέφη. Συγκεκριμένα, οι χρήστες θα παρέχουν ως είσοδο μια διαμόρφωση της προσομοίωσης προς εκτέλεση και στη συνέχεια η υπηρεσία θα αναλάβει την εκτέλεση αυτής της προσομοίωσης στο νέφος της επιλογής του χρήστη. Η εκτέλεση της προσομοίωσης θα βασίζεται σε κώδικα του οποίου η τοποθεσία στο Διαδίκτιο καθορίζεται στη διαμόρφωση της προσομοίωσης. Μετά την εκτέλεση της προσομοίωσης, η υπηρεσία θα παράγει μια πλήρη αναφορά της προσομοίωσης και θα την εμφανίζει στον χρήστη. Ο χρήστης θα έχει επίσης τη δυνατότητα να κατεβάσει την αναφορά στο τοπικό του σύστημα. Τέλος, η υπηρεσία θα πρέπει να εμφανίζει στον χρήστη την πρόοδο μιας εκτέλεσης προσομοίωσης καθώς και ένα ιστορικό των εκτελέσεων προσομοίωσης που εκτελούνται από τον χρήστη.

Τα κύρια πλεονεκτήματα της προτεινόμενης υπηρεσίας είναι η ανεξαρτησία από συγκεκριμένα εργαλεία προσομοίωσης, καθώς υποστηρίζει κάθε πρόγραμμα προσομοίωσης της προτίμησης του χρήστη που φιλοξενείται στο GitHub, η συμβατότητα με διάφορους παρόχους νέφους καθώς και η αυτοματοποιημένη ενορχήστρωση, ανάπτυξη και κλιμάκωση σύμφωνα με τις τρέχουσες ανάγκες φόρτου εργασίας. Προσφέρει επίσης μια φιλική προς το χρήστη εμπειρία διαχείρισης προσομοίωσης, καθώς δεν απαιτεί εμπειρία στην ανάπτυξη και ενορχήστρωση νέφους. Το μόνο που απαιτείται από τον χρήστη είναι η ύπαρξη λογαριασμού σε τουλάχιστον έναν πάροχο νέφους.

Λέξεις κλειδιά: Λογισμικό ως Υπηρεσία, Υπολογιστικό Νέφος, Προσομοίωση, Αυτοκλιμάκωση, Docker, Kubernetes, Αρχιτεκτονική Master/Slave, Terraform, YAML, Net, JWT, JSON, Minikube.

1

Introduction

In recent years, the growth of system complexity in fields, such as engineering, logistics, traffic control, and computational science has increased the need to use simulation systems for design validation, performance analysis, and decision making. However, for many users, running large-scale simulations remains a challenge as it requires expertise in technologies related to infrastructure, containers, cloud deployment, and task orchestration. These obstacles are amplified in cases where simulations must be run repeatedly with various parameters, making parallel execution and scalability critical, but difficult to manage manually.

To address these limitations, this thesis proposes the development of a cloud-native, Simulation-as-a-Service (SaaS) system, built using technologies, such as Asp.Net Core, Kubernetes, Docker, and Terraform. The system offers a lightweight, user-friendly Vue.js-based frontend for interaction that enables users to submit simulation jobs by simply providing a GitHub repository URL of the respective simulation project and their configuration in a JSON format. Behind the scenes, the system automates the provisioning of the required cloud infrastructure using Terraform, deploys simulation codes into a managed Kubernetes cluster, performs horizontal autoscaling when simulations are executed and cleans up resources after execution or in case of deployment failure, thus offering a seamless simulation-as-a-service (SaaS) experience.

This thesis contributes to the fields of simulation and cloud computing by filling the gap between simulation modeling and modern DevOps practices, offering a highly automated, cost-efficient, and scalable solution that can be used by researchers, engineers, and developers by simply having a cloud account. Unlike existing systems that are either tightly coupled to specific simulation tools or limited to fixed environments, this system supports multi-cloud deployment, horizontal autoscaling, and parameter sweep execution, all driven by user input.

Another advantage of the developed system (in the context of this thesis) is the combination of the backend automation and frontend accessibility. The frontend, built in HTML using JavaScript, Bootstrap 5, and Vue.js CDN-loaded libraries, offers an intuitive interface for users to configure simulation jobs, monitor progress, and view results. By combining cloud-native DevOps principles and tools such as Docker, Kubernetes, GitHub, Terraform with the frontend's usability, this project contributes a comprehensive solution to simulation execution at scale. It is tool-agnostic, supporting any container-based GitHub-hosted simulation code; cloud-flexible, able to deploy to any Terraform-compatible cloud provider; and user-friendly.

The rest of this report is organized as follows: Section 2 supplies useful background knowledge related to Simulation and Cloud Computing. Section 3 analyzes previous relevant work and compares it with our work. Section 4 analyses the Simulation-as-a-Service application's

development process by also supplying and commenting on its results, including application requirements, design models, and implementation details. Section 5 explicates how our application can be installed and supplies a demonstration of this application based on some usage scenarios along with respective screenshots. Section 6 deals with the evaluation of the application. Finally, Section 7 presents conclusions and basic directions for expanding or improving our work.

2

Background

This section describes the two main topics that have been studied and upon which this work is based. Firstly, the definition of simulation is provided, along with its benefits and uses. Additionally, applications and tools for running simulations are described, as well as their architecture. Secondly, a brief description of cloud computing is given, specifically, its definition, characteristics, benefits, service and deployment models and lastly, cloud technologies that are used in cloud software development and deployment. All such background knowledge is useful for completely understanding the context of our work and its main contributions.

2.1 Simulation

2.1.1 Simulation Definition

Simulation is the process of modeling the operation of a real-word system and analyzing the changes of its behavior over time [35]. The representation of the system is called a simulation model, which is a simpler version of the system that imitates its behavior and is used to study how the system works and evolves over time [36]. The model can be expressed as a set of mathematical or logical relationships between entities of the system that are of interest to be simulated and inspected. The state of the system is described by the state variables, which define its changes at a given point in time [73]. The simulation exploits this model and changes the state variables in order to make predictions and evaluations of the system's performance [35], [36].

2.1.2 Types of Simulations

Simulation is classified into the following types [35]:

- *Continuous Simulation*: In this type of simulation the system that is modeled changes over time continuously. The simulation involves variables that change according to differential equations. This means that the model can be described by differential equations as the variables, which describe the states of the system, are defined as continuous functions of time [37], [38]. This type of simulation is used in electrical circuits, fluid dynamics [75], population growth [76].
- *Discrete-Event Simulation*: This type of simulation simulates systems whose state changes at discrete times rather than in a continuous manner as in continuous simulation [35]. As such, in this case the system's state changes in response to discrete events that occur at a specific time [36].

- *Stochastic Simulation (Monte Carlo)*: Stochastic Simulation is a simulation type that generates multiple hypothetical outcomes by running a model with random input variables in each run and analyzes the distribution of those outcomes in order to compute the probability of an event [46]. Monte Carlo is an example of stochastic modeling technique that uses randomly generated numbers to simulate and analyze probabilistic outcomes in systems or problems. It is used to solve problems with uncertain inputs or outcomes, often by modeling static scenarios where the passage of time is not a critical factor. This method contrasts with deterministic models by incorporating randomness to reflect real-world variability [46], [47].
- *Deterministic Simulation*: Deterministic is the type of simulation where given inputs always lead to the same outputs [48], making the outcome and the model's behavior predictable.

The first two categories map to the time evolution dimension (e.g., whether time is discrete or continuous) while the last two to the presence of randomness (e.g., where the same input leads to different outputs so output is actually uncertain). This means that it is possible to combine simulations of different categories if these categories map to different dimensions (e.g., we can combine continuous simulation with stochastic simulation, such as in the case of chemical reaction rates with noise).

2.1.3 Advantages of Simulation

- *Risk-free experimentation*: as simulation does not lead to real-world consequences [73].
- *Testing of rare events*: allows checking how rare/risky/edge situations are handled, situations that might be hard to observe in real life [73].
- *Visualizes complex systems*: Helps visualize and comprehend how various components and interactions work within complicated or nonlinear systems (e.g., manufacturing, healthcare, networks) [36], [51].
- *Supports "What-If" analysis*: Enables analysts to evaluate the impact of different scenarios, configurations, or policies, answering "what-if" questions before committing resources [51].
- *Accelerates observation time*: Supports compression or expansion of time to observe long-term outcomes quickly or zoom into detailed events [51].
- *Delivers performance metrics*: Provides multiple performance metrics (e.g., utilization, queue length, wait times) that support objective, data-driven decision-making [36].
- *Stochastic and probabilistic models analysis*: Especially useful when real systems are stochastic or probabilistic, making analytical modeling difficult or impossible [36].
- *Cost-effective development*: Although developing a model requires time and expertise, it is often more cost effective than real-world trials or failures [51].
- *Facilitates reuse and adaptability*: Simulation models can be modified and reused to adapt to new conditions or further experimentation over time [36].

2.1.4 Simulation Application

There are many fields where simulation tools are used for validation, performance analysis, and decision-making, such as engineering, logistics, traffic control, computational science, defense, healthcare, physics, and biology [49], [52], [53], [54], [58]. The application of simulation in some of these fields is now explained below.

- *Healthcare education*: Simulation is globally recognized as an effective teaching method for promoting patient safety, as it enables students to apply their knowledge and practice skills in realistic clinical situations. [49], [50].
- *Education/Training*: The strengths of simulation in training include its capacity for repetition with focused objectives, extended practice time, trial-and-error exploration, exposure to rare or risky events, and accurate outcome measurement using proven evaluation methods [50].
- *Traffic control*: By modeling transport processes and traffic systems on highways and determining parameters, such as traffic intensity and delays, simulation can help in transport infrastructure design and optimization in order to address traffic problems [52].
- *Biology*: Simulation can help biologists and medical researchers to study and understand biological building blocks of living systems [54].
- *Defense*: Combat simulations in projected future scenarios produce results whose analysis can help in decision making regarding military future investments for the right preparation against potential real war scenarios [58].

2.1.5 Simulation Tools and Their Architecture

2.1.5.1 Introduction

Simulation tools typically follow and support an architecture that bases on the stages of a simulation process, such as model development, execution, output analysis and conclusion, that lead to decision making. Modern simulation tools often employ specialized simulation packages that reduce the programming burden and offer features, such as a simulation modeling framework, statistical tools, a graphical representation of models and a GUI (stands for Graphical User Interface) for communication and animation.

These tools can be broadly organized into two main categories: *simulation languages* and *application-oriented simulators* [36]. Simulation languages are flexible, low-level simulation environments that demand programming or scripting skills. They enable more control over model behavior and are able to support more complex or custom simulations. On the other hand, application-oriented simulators offer a more user-friendly GUI as well as domain-specific templates and tools. They can be used even by inexperienced users in terms of programming as they are easy to learn and exploit. However, they are less flexible, especially in terms of highly customized model support [36].

2.1.5.2 Tools Analysis

Some simulation tools which are popular in use are presented and briefly described below:

- *OMNeT++* is a discrete event simulator used for network simulations that models communication networks, multiprocessors and other distributed or parallel systems [62]. It is a modular, component-based simulation framework with an architecture that allows users to define models as hierarchical compositions of reusable modules, which communicate via message passing. Modules (components) are built by the simulation kernel and are programmed in C++ while the models are specified in a high-level language (NED) [63]. OMNeT++ also includes a graphical tool provided by the user interface libraries [62].

- *MATLAB*: MATLAB is a programming platform to create models, popular in engineering, education, research and development. It also offers a wide range of real-time engineering implementations [61]. Specifically, MATLAB is a high-level technical computing environment and programming language developed by MathWorks. Combined with Simulink and the MATLAB Compiler, it creates a modular platform of shared components, such as memory management, function execution, language processing, and graphic object creation, called the MATLAB Component Runtime (MCR). Simulink, an add-on product, provides a graphical interface for model-based design and simulation of dynamic systems, while the MATLAB Compiler converts MATLAB scripts, so called the M-files, into binary form in order to be executed outside of MATLAB. MATLAB's architecture also supports data visualization and GUI development [64]. Among its numerous capabilities is the performance of large-scale computations while running parallel simulations [118].
- *AnyLogic*: Supports discrete-event, agent-based, and system dynamics modeling in one platform [56]. AnyLogic's architecture is built around modular, object-oriented principles. The components that constitute the environment of AnyLogic are a graphical model editor and code generator that facilitates model construction, the hybrid simulation engine upon which the models run, the interface of the running models that is exposed over TCP/IP and UML as a modeling language for defining model structure and behavior. It also gives the user the ability to work with simulation models using Java code [65]. AnyLogic is used in many industries, such as supply chain, warehouse operations, healthcare, and logistics [119].
- *Arena*: A discrete event simulation tool, widely used in industrial engineering, operations research, and logistics [55], [59]. The architecture of Arena is modular and employs a flowchart-based modeling methodology, allowing users to represent complex processes using a library of predefined modules. It supports a great variety of statistical distributions that can be used for accurate modeling of process variability. It also offers statistical analysis and automated report generation. Built-in performance metrics and dashboards are also included providing real-time feedback. Finally, Arena features realistic 2D and 3D animation capabilities that help visualize simulations' results [66].
- *Simul8*: Designed for discrete event simulations, with use in fields, such as business, logistics and healthcare [60]. It uses 2D animation for the representation of a process. The user can create visual models of the analyzed system directly on the computer's screen. Each simulation model is built around Work Items, which represent dynamic entities (e.g., customers, products) that move through the system, change attributes, and use resources. These entities are generated by Work Entry Points based on specified arrival distributions. Storage Bins act as queues or buffers where the entities wait before next processes. Work Centers define the main object serving for the activity description which defines duration, resources usage during the activity, changes of entities' attributes and flow control. Work Exit Points mark the end of the modeled system. Finally, resources model the limited availability of workers, machines, or materials used during processing [60]. First, the system is modelled, then the simulation runs and the animation shows the entities' movement through the system. After the model's verification, various trials can run under different conditions which finally result in statistical analysis.
- *NetLogo*: A Java-based tool for agent-based simulations used in education and research [57]. In NetLogo simulations, the user specifies rules by giving instructions to hundreds

of agents that operate concurrently, and the simulations will then run according to these rules [77].

- *SUMO*: An open source, multimodal, continuous traffic simulation software. It is a modular simulation tool combined by two modules, sumo and sumo GUI. It has been written in C++ [20], [120].

The above tool analysis is summarized and shown in Table 1:

Table 1. Simulation tools

Tool	Type of Simulation	Application Domains	Open Source	Programming Language(s)	Deployment / Architecture Type
OMNeT++	Discrete-Event	Communication networks, multiprocessors, distributed/parallel systems	Yes	C++	Modular, component-based, hierarchical compositions of components communicating via message passing.
MATLAB	All (Plus Parallel simulations)	Engineering, education, research and development	No	MATLAB language	Modular platform (MATLAB+Simulink+ MATLAB Compiler+GUI), component based (MATLAB Component Runtime (MCR)
AnyLogic	All (Plus Agent-based)	Supply chain, warehouse operations, healthcare, logistics	No	Java, UML (for modeling)	Modular, object-oriented, components (graphical model editor and code generator)
Arena	All except Continuous	Industrial engineering, operations research, and logistics	No	SIMAN	Modular, flowchart-based; statistical analysis tools; 2D/3 animation
Simul8	All except Continuous	Business, logistics, healthcare	No	Visual Logic	2D animation for the representation of a process, model is built around dynamic entities
NetLogo	All except Continuous (Plus Agent-based	Education and research	Yes	Java	Agent-based, execution of rules by agents
SUMO	All but partially Continuous	Transportation	Yes	C++	Modular (sumo and sumo-gui)

2.1.5.3 Common Architecture

The above description of simulation tools can lead to the conclusion that in principle a modular, layered architecture is followed by most of the tools. An architecture that supports reusability, extensibility, and visual modeling. Firstly, a model composition component is responsible for the creation of models, then, a simulation engine handles the model execution and finally, statistical analysis takes place, along with data visualization, and animation, providing feedback through metrics and graphical outputs. A graphical modeling interface is also incorporated that allows users to define the simulation logic visually.

2.2 Cloud Computing

2.2.1 Definition of Cloud Computing

According to NIST (National Institute of Standards and Technology), cloud computing is a model for enabling available, on-demand network access to a shared pool of configurable computing resources. These resources can include networks, servers, storage, applications, services, and databases. They can be provisioned and released with minimal management effort or service provider interaction [1]. In other words, cloud computing is the delivery of computing services over the internet by a cloud provider with pay-as-you-go pricing, so that users do not need to purchase, operate and maintain on-premise physical data centers and servers [2].

2.2.2 Characteristics of Cloud Computing

The main characteristics of cloud computing are described below:

- *Self-Service*: Users must have the ability to provision cloud services according to their needs, almost instantly, without any human operator's intervention [4].
- *Pay-as-you-go pricing*: Cloud Computing gives consumers the ability to use only the necessary amount of resources [4] and, by extension, to pay the provider only for the actual resource usage incurred [3].
- *Elasticity*: Cloud services can be dynamically configured and elastically provisioned giving the ability to scale up and down on demand and tailor the resources to the customer's needs [1], [3].
- *Multi-tenancy*: Customers use resources from cloud providers' resource pools and they do not have the knowledge of the location of these resources [2], [3]. They can also share the same physical resources in a private and secure manner such that their workloads cannot interfere. [2]. This enables better resource usability and economy of scale, increasing the actual profits of the cloud provider.
- *Ubiquitous access*: Cloud resources can be reached from different platforms like mobile phones, tablets, laptops, and workstations [1], such that the users can access them at any time and from any device connected to the Internet [2].
- *Measured service*: The utilized cloud resources are monitored and charged according to their usage. This enables customers to control the resource usage and cost. It also enables the provider to scale and fail over these resources as needed, when the respective need arises [1].

2.2.3 Benefits of Cloud Computing

As we can see from the characteristics that were previously described, Cloud Computing offers numerous benefits like efficiency in cost management, scalability, agility, flexibility, availability, and advantages as far as disaster recovery concerns [2], [6]. These benefits are now analyzed in detail.

- *Cost management*: Firstly, the primary advantage of migrating to the Cloud is the cost reduction. As Cloud Computing is a subscription and rent based model [6], there is no need for investing in equipment, hardware and infrastructure [3]. Users rent the resources they need from Cloud providers and pay only for the used ones, reducing in this way the IT operations costs. Apart from the pay-as-you-go benefit, the on-demand use of cloud resources offers *scalability* and *elasticity* as the users/customer can attain and release resources as needed on-demand based on the current workload.

  This rent-based model also leads to the elimination of infrastructure maintenance [5] as it burdens the cloud service provider who is responsible for all the upgrades and updates, thus saving IT administration effort from the customer [2].

- *Availability*: Another notable aspect to consider is the way of gaining access to the stored information in the Cloud. Users can access information whenever required, from any location and device, provided that the device is connected to the Internet [2], [6]. Moreover, storing data to the Cloud provides high availability as the data can always be accessed, even in cases when certain devices are out of service. High Availability can be achieved through mechanisms such as [81]:
  - *fault tolerance*: while a failure occurs, it has to be detected and the system to be transferred to a healthy state through the use of suitable fault-tolerance mechanisms (e.g., fail-over, replication) in order to continue its normal operation.
  - *overload protection*: The mechanism of autoscaling is applied in order to scale up and down resources on demand providing in this way protection against resource overload.

  Concerning Service Availability, Cloud Providers ensure it through an SLA (stands for Service Level Agreement) issued between the Provider and the customer. This SLA specifies the quality of the service that is delivered to the customer by the Cloud Provider. In an SLA the outage time and how it is measured by the Cloud Provider is specified in order to meet at least 99.95% availability [81].

- *Disaster recovery*: Very important is also the fact that data can be quickly recovered in case of a natural disaster [2]. Specifically, disaster recovery can be achieved through strategies like [80]:
  - *data backup* in data centers across multiple geographic locations,
  - *site recovery* to restore services using the techniques of *hot site* (when the primary server is down, the production is loaded into already equipped and configured backup/fail-over servers), *warm site* (the production is loaded by redirecting the Domain Name System (DNS) to the backup site in pre-configured servers but it requires the manual restart of services), *cold site* (full setup and data restoration are needed before the system becomes operational).

Cloud Providers offer their clients the recovery approaches/techniques in order to create their own disaster recovery plan [82], [83]. Data backup and recovery are automated processes with minimal human intervention most of the time [80].

- *Flexibility*: Cloud Computing enables developers to deploy and run applications more easily and rapidly, without having to buy and set up hardware that is costly and time consuming [2].
- *Security:* As for security, there are some challenges regarding security risks of cloud computing over the traditional IT systems, such as cross-tenant attacks (exploiting resource isolation vulnerabilities related to resource pooling that can lead to unauthorized access to a tenant's resources) and infected or vulnerable VM images in shared image repositories, i.e., risks to data privacy and integrity that can lead to data breach of cloud customers [84]. However, with proper planning and security control, cloud computing can offer better security for confidentiality, integrity, and data availability [18].

2.2.4 Cloud Service Models

There are three main service (delivery) models in which cloud services can be classified according to the abstraction level of the respective capabilities provided. These service models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [4]. The difference among them lies not only on the abstraction level but also on the types of responsibilities the cloud user and cloud provider undertake as it is shown in Figure 1 [5]. Apart from the theem main delivery models, there is a fourth service model called Function as a service (FaaS), also known as serverless computing, which applies an event-driven architecture based on functions [7].

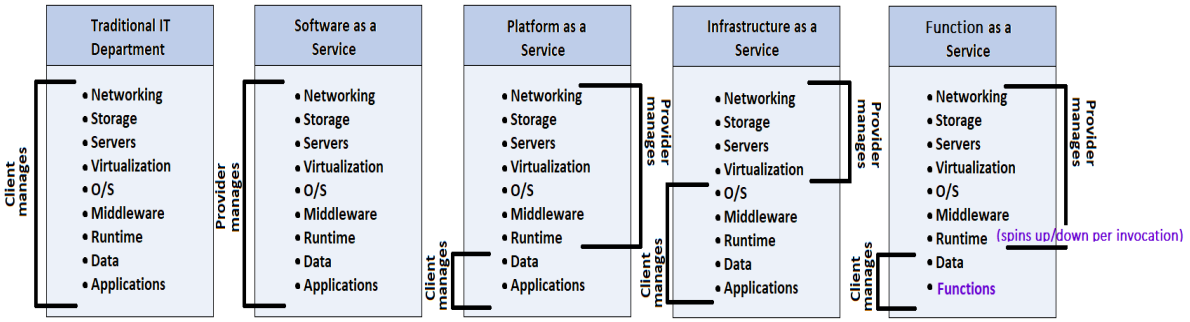


Figure 1. Comparison of Traditional IT and Cloud Service Models, based on the image of [5]

All these four service models are shortly analysed below.

- **Infrastructure as a Service (IaaS):**

  The IaaS service model is close to managing physical servers. The Cloud provider delivers the hardware and the associated services (wrapping this hardware) needed to store and run applications as a service [11]. The users do not know the exact physical location like data center room, rack of the rented hardware [14]. However, they can often choose or see the geographic region or availability zone within a specific region in which their resources are hosted.

The maintenance of physical data centers and infrastructure like servers, data storage, networking, and virtualization, as we can see in Figure 1, are abstracted from the consumer's responsibilities as they are provisioned as a service by the provider [12]. So, the cloud provider is responsible for hosting, running and maintaining the infrastructure [11], while the client has control of the operating system, network configuration and applications [13] within virtualized resources offered on top of that infrastructure.

The IaaS offers resource, network infrastructure and data management and the ability to gain information about servers, storage, and network [14]. It provides access to services on-demand. The type of resources offered by this service model are virtualized resources (e.g., VMs, virtual storage) and logical resources (load balancers, network throughput/bandwidth, information security) [14].

Some examples of IaaS service models are Amazon Web Services (AWS)'s Elastic Compute Cloud (EC2), AWS's Secure Storage Service (S3) [11], GoGrid, Rackspace Cloud [13] and Google Compute Engine (GCE) [17].

- **Platform as a Service (PaaS):**

  PaaS is a cloud platform offered as a service by the cloud provider on which developers can develop, deploy and run their applications. This model offers a higher level of abstraction compared to the IaaS model [4] as the cloud provider is strictly responsible for both the underlying infrastructure, specifically for networking, servers, operating systems, storage and virtualizations, and the respective development environment on top of this infrastructure, enabling the client to focus on pure application development, configuration and deployment [4], [11]. In particular, PaaS vendors manage the application (development) platform/environment while they provide development tools, services (language runtimes, frameworks, libraries, middleware) and capabilities like caching, asynchronous messaging, database scaling, as a service so that the developers can focus on their core business of building applications without worrying about installing, updating and managing the application development environment [12], [13].

  Some examples of PaaS service models are Google AppEngine, Microsoft Azure, Yahoo developer Network, MSFT, Heroku, Engine Yard,force.com [13].

- **Software as a Service (SaaS):**

  The SaaS is the delivery of an application as a service by the cloud provider [12]. The provider's application runs on a cloud infrastructure for which the provider is responsible. In this case, the cloud provider manages all aspects of the application environment, such as the network, servers, operating systems, storage, runtime, software [13] and the client may only need to slightly configure the software [11] as he/she needs or prefers. The provider is also responsible for scaling the application as well as updating it when the respective need arises.

Some examples of SaaS service models are Gmail, Hotmail, Office 365 and Salesforce CRM [13].

- **Function as a Service (FaaS) (Serverless Computing):**

  Function as a Service (FaaS) is basically a part of Serverless Computing along with another service model called Backend as a Service (BaaS) [7], [9]. The FaaS model relies on functions which the cloud provider operates, after their deployment, so as to provision, execute and scale them. The BaaS model consists of services like cloud storage and messaging and provides backend support to the FaaS model [7]. An application is deployed as a set of functions and FaaS is responsible for the invocation of these functions as well as for the provisioning of the underlying resources on top of which these functions run. FaaS relies also on extra services that are related to common functionalities needed for information storage and function triggering, especially as functions are usually stateless [7].

  The physical servers exist in the background, hidden from the developers and as a result, developers do not have to deal with the infrastructure, so they are free of server management and configuration as well as any other operational aspect, and can focus on pure application development [7], [8]. The execution of the functions is on-demand; this means that underlying resources are used only when a function is invoked and during its execution [10]. This leads to charging the functions only based on their actual/real usage in contrast to other delivery models, like IaaS, where the charging takes place based on usage but according to a specific unit of time (usually per hour). For example, if a function call lasts for 10 minutes, the user is charged only for that amount of time. However, in case of an IaaS, if it is used for 70 minutes, it will be charged for 120 minutes (i.e., 2 hours).

  Some examples of FaaS service models are Amazon web services (AWS) Lambda, Google Cloud Function, and Microsoft Azure Function.

2.2.5 Cloud Deployment Models

Cloud Computing can be classified into four main deployment models according to who owns or manages the cloud and to its actual level of access.

- **Public Cloud Deployment Model:**

  A *public cloud* is hosted on the premises of the cloud service provider [1] and is offered to the general public and any organization via the internet on a pay-per-use basis. The Cloud service provider owns and operates the underlying cloud resources. It allows customers to scale their workloads on demand without the need of buying and owning the hardware [5], [15].

  The public cloud offers several advantages, including high data availability and continuous uptime, ensuring services remain accessible. Its on-demand scalability enables users to easily adjust resources as needed. Its pay-per-use model ensures efficient resource utilization, minimizing waste. Finally, it offers easy and low-priced setup and also provides access to 24/7 technical expertise for support and issue resolution [15].

Its main drawback is that of security: due to its multi-tenant nature as well as potential resource isolation issues plus the possibility of non-alignment of security architecture and mechanisms between cloud provider and client, the security risk is higher than other deployment models, like the private cloud one.

- **Private Cloud Deployment Model:**

  On the other hand, a *private cloud* is not available to the general public. It can either be physically located at an organization's datacenter and be managed and maintained by that organization, or it can be hosted, controlled and maintained by a third party provider [5]. In any case, its services are offered only to users inside the owning organization. Its main benefits, compared to the public cloud, are that of data privacy and security as it is not accessible to the general public, but serves only internal users or trusted partners, and employs a uniform security architecture aligned with the organisation's main security policies.

  Its main drawback is that of higher upfront cost for hardware, software and staffing [15] as well as less scalability/elasticity depending also its size, which is usually limited compared to a public cloud.

- **Hybrid/Multi Cloud Deployment Model:**

  Another deployment model is the combination of two or more clouds (private, public or community) which is called *hybrid cloud*. In this case, the combined clouds should rely on proprietary or standard mechanisms that enable the portability and migration of customer data and applications. An example of such a model concerns the case where an organization stores sensitive data on the private cloud which is more secure, and less critical data into the public cloud [5].

  The advantages of hybrid cloud include reduced capital expenses by outsourcing infrastructure needs to public cloud providers, improved cost-efficiency for temporary projects, and optimized spending across the application lifecycle. It enables organizations to use public cloud for development, testing, or retiring outdated applications, while keeping critical workloads in private cloud environments. Hybrid cloud also offers the control of private clouds with the scalability of public clouds, supports cloud-bursting (case where a private cloud is mainly used and when its load is quite high, the public cloud is used to attain more resources), and enhances overall organizational agility and flexibility [15].

  Its main drawback is higher complexity due to the need to manage multiple clouds and difficulties in identifying root-causes and assigning accountabilities in the context of security incidents and other problematic situations.

- **Community Cloud Deployment Model:**

  A community cloud is a shared cloud infrastructure designed for use by multiple organizations that have common objectives, such as similar privacy, security, or regulatory requirements. It lies between public and private clouds in terms of accessibility and control. Access is limited to authorized members of the community, such as a group of banks or financial institutions. A community cloud can be jointly

managed and maintained by the participating organizations or operated by a third-party cloud provider. It aims to combine resource sharing and distributed control with sustainability and self-management, potentially leveraging underutilized user resources, where nodes act as consumers, producers, and coordinators [5], [15].

A community cloud is less secure than a private cloud and less scalable that a public cloud.

2.2.6 Cloud Technologies

With the increased use of Cloud Computing in recent years, virtualization technologies have been developed and used by IT Industries in order to enable multi-tenant use of physical resources. *Physical machine* or *hardware virtualization* is a technique of building multiple Virtual Machines (VMs) on a single physical machine allowing the deployment of many applications on it [22]. Each VM on that machine can run any OS of choice as well as any kind of application on top of it. In this kind of virtualisation, multiple resources are needed due to the execution of extra OSs on top of the physical machine OS.

Containerization is the next level of OS Virtualization which provides an isolated environment for each application to run so that many applications can run on the same OS kernel (either of a physical or virtual machine). Containers, in contrast to VMs, are much less resource-intensive and much faster to create and scale.

One of the most common containerization technologies is Docker, which will be analysed below along with the Kubernetes, which is a container orchestration engine that automates deployment, scaling, and management of containerized applications [22], [24]. Lastly, the Microservices architecture is analyzed as it is tightly linked to cloud computing and fits well with containerization concepts [85].

2.2.6.1 Docker

Docker is a platform which gives the ability to package and isolate an application (or one of its main components) in a container and run it on any environment [22]. A container is an isolated userspace via which applications can run as OS processes on any environment/platform. A container is created from a specific filesystem called an image, which includes both an application and its associated files, (required to run it), including dependencies, configurations, scripts and binaries. As such, a container image contains all the needed information to run an application (or an application's main component), with all its required dependencies, in the form of a container on any platform/OS.

Docker follows a client-server architecture which consists of the following components:

- *Docker daemon*: This component has the role of a server and is responsible for running, managing and distributing Docker containers. It communicates with the Docker client through a REST API via a unix socket or TCP in order to process the client's requests [23]. Docker daemon and Docker client can run on the same machine or they can communicate remotely. Furthermore, Docker daemon can also communicate with other daemons too in order to manage Docker services [21].

- *Docker client*: Docker client communicates with one or more Docker daemons and is the component through which the user interacts with Docker using commands that Docker client passes to the daemon through the aforementioned REST API [21].
- *Docker registry*: Docker registry is the component where Docker images are stored. While there are public registries, stored and managed by online services like Docker Hub, also private registries can exist that a user can run on his/her own. Images can be pushed to and pulled from the preferable registry by using the appropriate Docker commands [21], [23].
- *Docker desktop*: Docker desktop is a GUI-based application which encompasses a whole Docker (component) stack and can be used to build, deploy and share containerized applications [21]. Docker Desktop includes the Docker daemon, the Docker client, Docker Compose, Docker Content Trust, Kubernetes, and Credential Helper. Docker Compose is a tool for running multi-container applications. It combines services, networks, and volumes in a single YAML configuration file giving the ability of creating and starting them all at once [21].

The architecture described above is depicted in Figure 2.

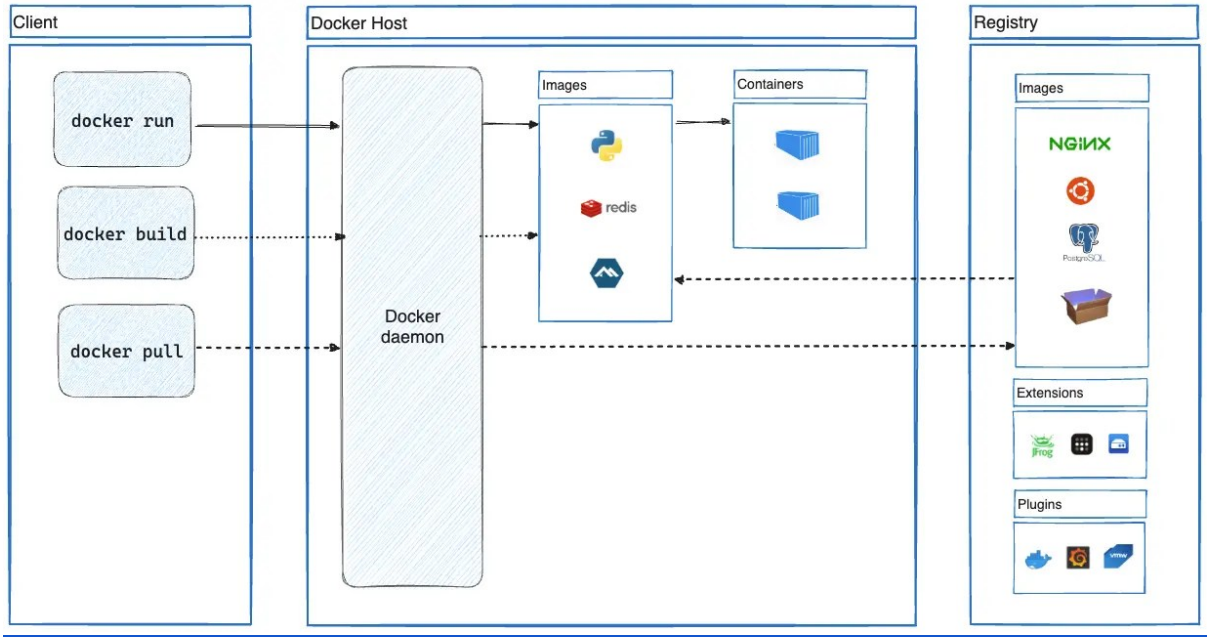


Figure 2. Docker Architecture. Source 'https://docs.docker.com/get-started/docker-overview/'

Please note that apart from Docker, Docker Compose also exists. The latter is a Docker technology, which enables the orchestration of the containers of an application within one machine (physical or virtual). This is useful for containerised applications comprising components, which map to different container images. Docker Compose lies on top of the Docker architecture and enables through single commands to manage all the containers of an application as long as its deployment architecture has been modelled within a file called docker-compose.yml.

Apart from the Docker architecture (presented in Figure 2), it is worth providing a description of the main Docker Objects that are created while using Docker.

- *Docker images*: A Docker image is a filesystem, a read-only template, with instructions for creating a Docker container out of it [21]. It is created by a Dockerfile, which is a normal textual file that encompasses suitable image generation instructions that can be utilized to construct the image of a whole application or one of its main components [21], [23]. When the 'docker build' command is executed for a specific Dockerfile from the terminal, the image will be created with all the dependencies contained in the Dockerfile.

  An image can be built based on another image called a 'base image'. For example, the base image can be an OS like Ubuntu, and another image, e.g. an image of Apache web server, can install the web server on the Ubuntu OS when the container is created [21], [22].

- *Docker containers*: A Docker container is an instance of an image and runs on a host machine (either physical or virtual) isolated from other containers and processes. Additionally, due to the isolation capability of containers, there is the ability to run multiple of them at the same time on a single host [21]. These containers can be created individually through the execution of respective commands by utilising the Docker CLI (Command Line Interface) or in one shot by executing a single command in the Docker Compose CLI.

2.2.6.2 Kubernetes

Kubernetes is an open-source platform for managing containerized applications. It is an orchestration tool for containers able to automate their deployment, management, scaling and networking [24], [25]. Kubernetes allows the deployment and scaling of containerised applications across multiple resources, which form a cluster. These resources usually originate from the same cloud while in some scenarios cluster resources could come from both public and on-premise/private clouds. On the contrary, Docker Compose orchestrates containerised applications on a single host.

It was first developed by an engineering team at Google in an attempt to keep up with rising cloud technologies like the aforementioned virtualisation and containerisation ones. In 2014 Google open-sourced Kubernetes and donated it to the Cloud Native Computing Foundation [26].

Kubernetes has many advantages, such as load balancing for distributing traffic among containers, when needed, in order to maintain deployment stability and ensure high availability. It also gives developers the ability to mount cloud or local storage. Another benefit is that Kubernetes can automatically restart a container in case of failure, replace it or even take it down if it stops responding. It also enables developers to achieve high scalability and availability as it can scale containers within the same resource and across multiple resources [24].

Kubernetes forms and manages clusters of hosts, i.e., virtual or physical machines, and containers deployed on those clusters. A cluster consists of one or more physical or virtual machines called nodes. There are two main types of nodes in Kubernetes, the control plane and

worker nodes. In the Control Plane node[1], the components included are: scheduler, etcd, API & controller manager. On the other hand, in the worker nodes, we have kubelet, kube-proxy and container runtime. Kubelet handles pod deployment and execution. Kube-proxy is a network proxy that provides support to Kubernetes services and enables the communication between pods. The container runtime is responsible for executing the containers of pods. A Kubernetes cluster must have at least one worker node[2] in order to run Pods.

The main components of Kubernetes cluster architecture are described in more detail below as shown in Figure 3:

- *Pods*: A pod is the smallest deployable unit of computing in Kubernetes [30]. It is a group of containers that all share the pod's execution environment, meaning network and storage resources, memory, hostname and filesystem volumes [30], [26]. A pod may contain one single container or it can run multiple containers that need to share resources [30]. Usually, the containers of an application are deployed on one pod, so we have a single application-to-pod mapping. However, there are cases when it is better to put multiple containers in one pod. Such a case is when an application follows a microservice architecture where each container has its own responsibility linked to one distinct microservice; in this case we can claim that the pod maps to an application component/microservice [26].
- *Node*s: A node is a virtual or physical machine in which pods run. There are two types of nodes in a Kubernetes cluster, the Master (or Control Plane) and the Worker nodes, managed by the Master node. The Master node consists of several sub-components that are responsible for controlling and managing the Kubernetes cluster as well as scheduling and scaling pods in that cluster [26], [31]. It is responsible for distributing load across the Worker nodes while each Worker node contains the appropriate environment and services to run pods on it [29].
- *API Server*: This component is part of the control plane node and it functions as the Kubernetes' front end. Its job is to expose the Kubernetes API over HTTP(S), to process, authenticate and authorize REST requests and update the state of the cluster [26], [31]. The API Server is responsible for the communication between Control Plane and the nodes, as well as the communication among the Control Plane's components [89].
- *Scheduler*: This is a control plane component that automatically selects the most suitable node for deploying to it a newly created pod. It evaluates various factors for this selection, such as resource requirements, constraints, affinities, and data locality, so as to identify feasible nodes, score them, and assign the pod to the best match [92].
- *Controller-manager*: Is the Control Plane's component that runs controller processes. A controller is a control loop that watches the shared state of a cluster through the API Server and requests changes in order to bring the current cluster's state closer to the desired one [93], [94].
- *Cloud Controller Manager:* Is a control plane component that links the cluster into a cloud provider's API, in cases when the Kubernetes runs in a Cloud platform, and manages only controllers specific to the cloud provider [117].

[1] Usually it is one node that plays this role. However, it is recommended in production environments to have multiple control plane nodes to avoid single point of failure situations.
[2] This covers even cases where the cluster includes a single machine, which can play the role of both a data plane and worker node.

- *Etcd*: This component is a consistent, distributed, highly-available key-value store that provides a reliable way to store all data that needs to be accessed by the Kubernetes cluster [91] representing its intended/desired and current state.
- *Kubelet*: This component runs on worker nodes within a Kubernetes cluster. Its job is to retrieve the information about a pod and make sure that the containers on the node are healthy and running. Kubelet is provided with the information about the pod through a YAML or JSON object called PodSpec which describes the pod and how it should run [28], [90]. Kubernetes actually manages workloads over pods which are resources that manage sets of pods instead of one single pod [110]. Workloads include pod templates, which are specifications for pod creation that incorporate the PodSpec [30], [128]. Kubelet administrates a whole workload including pod creation and supervision, interacts with the container runtime, and reports node status to the control plane components [90]. This means that Kubelet can be used to manage the desired state of a workload, which might include the availability of one or more pods.*Container runtime*: Is the software that is needed in order for the containers to run. Some examples of container runtimes that are supported by the Kubernetes are the Docker Engine, CRI-O and Containerd [95].
- *Kube Proxy:* Is a network proxy running on each node inside the Kubernetes cluster and handles network communication from outside the cluster to the pods. It is optional though, as equivalent functionality can be provided through a network plugin [115] (a software component that allocates IP addresses to pods in a cluster establishing communication between them [116]).

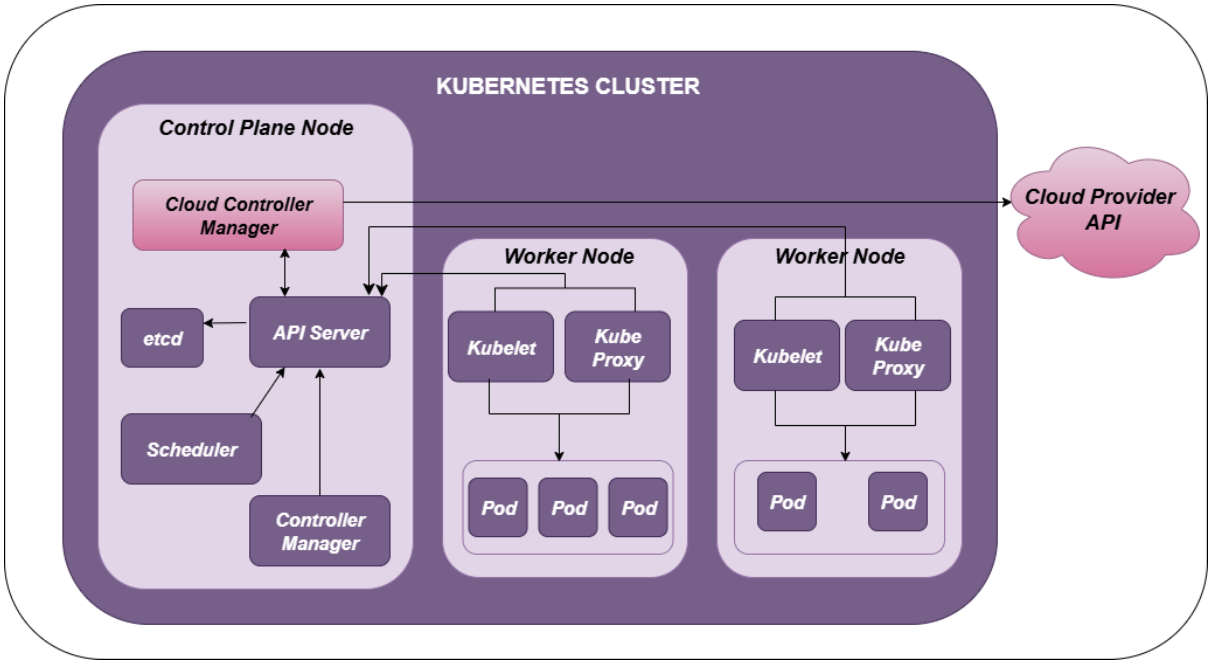


Figure 3. Kubernetes Cluster Architecture

2.2.6.3 Microservices

The Microservice architecture is an approach to developing software applications not as a single service, like monolithic applications, but as small parts called microservices, each of which is autonomous, independent and executes its own process [33]. These microservices communicate with each other over a network via message passing or RESTful APIs [33], [34]. Each service can be built using a different programming language, has its own technology and functionality, its own database and can be considered as a small, independent application itself while it can be deployed using lightweight container technology [33]. Specifically, microservices are independently developed and deployed while it is a common practice to use cloud-based containers for deploying them [87]. Docker is used as the main virtualisation technology for microservice applications while these are usually deployed and scaled in Kubernetes environments. The advantages of such environments are that they can scale microservices on demand, they offer load balancing services and they can replace failing microservices enabling the respective applications to be still fully operational. [86]

Applications built according to this technology have many benefits including:

- *Scalability*: Each microservice can be scaled independently based on demand, unlike monolithic systems that require scaling the entire application (this can take more time and is costlier) [33].
- *Flexibility*: The fact that each microservice is an independently deployable unit, offers the ability to use different technologies to develop each one according to the functionality it needs to implement and offer instead of having to follow strict, uniform development requirements for all application components [34], [86].
- *Faster delivery*: teams can manage services they own independently without being constrained by an organization's technology stack which leads to faster features' delivery [33].
- *Easy maintenance*: Smaller, isolated services with independent codebases are easier to maintain and update [33].
- *Faster Implementation*: Microservices enable parallel development, testing, and deployment of new functionalities, allowing quicker feature releases [33.

E-commerce applications can benefit from following a microservice architecture as described in [33]. Such applications can comprise the following microservices: an order service, a payment service, a shopping cart service, a product service, a client service, an inventory service. As each microservice is independent, if client requests rise during the sales season, the order service and the payment service could scale up according to CPU and memory needs in order to handle the increased load. Similarly, if the cost-related microservice is receiving several requests that require more computational resources, it could be scaled individually. If the cart service needs some changes, those changes can be applied and the cart microservice can be redeployed without modifying the rest of the microservices (so the rest of the application).

Examples of well-known companies, which have finally adopted the microservices architecture over the initial choice of a monolithic architecture are Amazon and Netflix [33].

3

Related Work

This section presents the related work and outlines its primary contributions to the current objectives/goals of this thesis. It further analyzes the strengths and limitations of each work and compares them to the work conducted in the context of this thesis.

3.1 Analysis of Related Work

3.1.1 Petri Net Simulation as a Service

There is previous work that studies simulations and how they can be executed remotely as cloud-based processes using cloud resources in order to reduce the cost and computing power needed for a simulation to run. Petri Net Simulation as a Service [45] uses SOAP (stands for Simple Object Access Protocol) in order to establish communication between services. According to that work, the client communicates with the simulation service and requests the start of the simulation of a model where this communication is conducted via SOAP. Furthermore, the integration of a simulation engine with external clients through standardized APIs, which are presented in [45], decouples the simulation from the user's machine. Thus, it can be easily observed from the currently analysed paper on Petri Net Simulation as a Service, that the idea of executing simulations as a service has been already adopted by the literature.

The work under analysis wraps the Petri Net tool called Renew as a simulation service, allowing for modular, reusable, and remotely accessible simulations. This makes it well-suited for complex, distributed systems. The service is also scalable, supporting easy deployment and configuration through pre-built virtual machine images. The analysed work's systematic breakdown of service categories, the integration of services for model manipulation, service recommendation, logging, and simulation monitoring, reinforces its practical utility for developers, making the integration of various tools easier and leading to the creation of an easy-to-use simulation environment.

Despite its strengths, the work under analysis has some limitations. The reliance on the Renew tool and a Java-based infrastructure may limit the service's portability to other simulation platforms or programming environments. The narrow scope of the case study limits generalizability, while the paper does not also deeply address issues like scalability under real-world workloads or robust fault tolerance mechanisms, which are essential for production-level deployment.

3.1.2 Parallel and Distributed Simulation in the Cloud

The paper on Parallel and Distributed Simulation in the Cloud [19] introduces the Aurora system, a custom-built framework for PDES (stands for Parallel and distributed Discrete-Event Simulation) in cloud environments. Its architecture follows a master/worker pattern to accomplish separation of coordination logic (master) from execution logic (workers) where the master coordinates simulation tasks by distributing logical processes (LPs) across worker nodes. LPs are sequential discrete event simulations that model the systems evolution over time by exchanging time-stamped messages.

One of the strongest aspects of the currently analysed paper is its exploration of leveraging cloud computing for conducting parallel and distributed simulations, a domain traditionally tied to high-performance on-premise computing infrastructure. The general benefits of cloud environments, like cost efficiency, scalability and accessibility and the advantages in the field of parallel and distributed simulations specifically, like eliminating the hardware demand and hiding the execution complexity from non-expert users, are discused supporting the argument for shifting simulation workloads to cloud platforms. The master/worker architecture applied by the Aurora system offers a practical and structured framework capable of addressing the challenges of cloud environments. The Aurora system uses message bundlin**g** to reduce network latency in high-latency cloud environments. It also performs the computations in the data level by cashing the LPs state, making it work successfully in cloud environments that follow the MapReduce paradigm in order to process large datasets in parallel. It is designed for scalable cloud infrastructures, provides high availability by offering support for fault tolerance, also supports load balancing and finally, executes computations at the data level, a feature very beneficial in cloud computing as it enables processing big data in parallel plus to reduce cost as the need to move big data is minimised.

However, the paper has some limitations. The focus on a custom-built simulation system (Aurora) may limit generalizability. The Aurora system, a single, self-contained simulation environment, is not designed as a general-purpose simulation platform where users can plug in their own simulations. Its architecture is tightly coupled to its internal logic for parallel discrete event simulation (PDES), and it assumes users will write simulations specifically for the Aurora's execution model. Finally, although scalability is claimed, there is limited exploration of how the employed architecture would behave under more complex, heterogeneous workloads.

3.1.3 A Simulation as a Service Cloud Middleware

The simulation as a service cloud middleware paper [20] introduces the SIMaaS cloud-based simulation as a service system, which promotes the idea that simulation should be abstracted from the user and delivered as a service, regardless of its underlying execution complexity. The paper also strongly advocates for container-based virtualization (Docker) as a superior alternative to heavy VMs in terms of startup time, resource usage, and scalability. By wrapping simulations in containers and deploying them on-demand, the SIMaaS approach ensures environmental consistency and isolation for each simulation job.

One of the key strengths of SIMaaS is its dynamic resource management algorithm, which elastically scales simulation workloads based on user-defined deadlines while minimizing resource costs. The system is especially well-suited for scenarios requiring the execution of a

large number of short-duration simulations, such as stochastic model checking, Monte Carlo methods or parameter sweeps. As these types of simulations often need to be repeated thousands of times with slightly different input parameters to gather statistical results, the SIMaaS uses Linux containers (Docker), which are lightweight and can be instantiated almost instantly to reduce the time and resource cost per simulation instance, making it feasible to execute large volumes of jobs in parallel. Finally, the middleware includes modules for performance monitoring, which provides real-time feedback, and result aggregation, which collects and merges the outputs of many individual simulation runs.

However, the SIMaas paper has some particular limitations. The architecture, while modular, is primarily validated in a controlled private cloud setting with limited scale (up to 60 hosts), which may not reflect the challenges of public cloud environments or larger-scale deployments.

3.2 Comparison with the Thesis Work

3.2.1 Petri Net Simulation as a Service

In terms of architecture, the Petri Net Simulation as a Service paper implements a service-oriented architecture (SOA), which allows Petri net models to be simulated using SOAP-based communication through an Apache Axis2 server. This allows the described simulation tool (the Renew system**)** to be accessed and invoked remotely as a simulation service. As our proposed work also relies on SOA architecture for the backend, it offers a complete cloud-native application with a scalable architecture comprising various components, including Terraform and Kubernetes clusters.

Deployment in the Petri Net system is limited to manual or semi-automated methods, even though the service can technically be hosted on platforms like Windows Azure. Our proposed system, however, automates the entire deployment lifecycle using Terraform, including provisioning managed Kubernetes clusters, deploying services, dynamically creating pods, managing autoscaling, and fully tearing down the infrastructure after use or in case of failure, enabling efficient and repeatable cloud-native simulation workflows.

The Petri Net service's architecture is tightly bound to the Java-based Renew tool, which limits its extensibility and compatibility with other simulation frameworks while our proposed system is designed to be tool-agnostic, allowing users to submit GitHub-hosted simulation projects, provided that they follow the Docker/Kubernetes technology and structure. This makes our system more flexible.

In terms of parallelism, the Petri Net service introduces nested simulations for tasks like adaptive traffic control, but the execution model remains mostly recursive and centrally orchestrated. Our proposed system, by contrast, supports parallel execution using a master/slave approach, where the master dynamically creates slave pods, each representing a separate simulation run, enabling parallel computing capabilities.

3.2.2 Parallel and Distributed Simulation in the Cloud

The Aurora system is built around a custom master/worker framework for parallel discrete event simulation (PDES). It emphasizes performance through message bundling and logical process caching. Our proposed system shares the master/slave concept, but rather than being

tied to a specific simulation engine, it orchestrates the simulation deployment across cloud providers using standardized and widely adopted tools like Kubernetes, Docker, and Terraform.

Aurora tightly couples simulation logic with the system infrastructure, meaning that users must write simulation code that fits into Aurora's own execution model. In contrast, our system separates the orchestration logic from the simulation logic as the orchestration is handled by the encompassed .NET API and the simulation logic is hosted in user-provided GitHub repositories, offering greater flexibility and maintainability.

Aurora relies on its own internal message-passing mechanisms, which makes integration and monitoring more complex while our system uses the Kubernetes API to communicate with the master/slave simulation, while the communication between master and slave pods (in cluster communication) is Pod-to-Pod communication through the Kubernetes API using their internal IP addresses [74]; this also enables the master pod to track simulation progress and collect results more easily.

In terms of elasticity, Aurora introduces the concept of optimistic synchronization, but it lacks implementation of cloud-native autoscaling techniques. Our proposed system supports both infrastructure-level autoscaling through Terraform (e.g. through a managed Kubernetes service like AWS Cluster Autoscaler) and in-cluster autoscaling via Kubernetes' Horizontal Pod Autoscaler (HPA). This ensures better utilization of resources and robust scalability based on actual workload demands.

3.2.3 A Simulation as a Service Cloud Middleware

Both systems share the foundational idea of Simulation-as-a-Service (SaaS): they aim to abstract away the complexity of simulation execution and allow users to submit jobs and retrieve results through a cloud-hosted service. The SIMaaS middleware is built as a container-based system that handles simulation requests via a modular web interface and RESTful APIs while our proposed system takes this concept further by leveraging Docker containers, Kubernetes for orchestration, and Terraform for infrastructure provisioning.

In terms of deployment, SIMaaS is tested within a private cloud of up to 60 hosts and focuses on on-demand container creation for simulations. It includes a Quality of Service (QoS) policy, containing a scheduler that schedules a simulation's tasks in containers and an admission controller to decide how and when to launch containers. Our proposed system, however, is designed for real-time deployment and elastic scaling in public cloud environments. It dynamically generates Terraform configurations based on user-provided credentials, deploys managed Kubernetes clusters, and uses Kubernetes APIs to control job lifecycle, autoscaling, and cleanup. Specifically, through the Kubernetes API the master pod decides the creation and scaling of slave pods based on the current simulation's state, while Kubernetes decides about pod scheduling and placement.

One significant difference lies in the flexibility of simulation execution. In SIMaaS, the simulation code is known and fixed within the system while our system is designed to be tool-agnostic. Users provide a GitHub repository URL and simulation parameters in JSON format. Our system then builds and runs simulations using the Docker/Kubernetes YAML files included in the repository; this makes the system more flexible as it enables support for any type of discrete-event simulation that complies with the master/slave structure.

Finally, SIMaaS is confined to a single pre-configured cloud setup while the proposed project allows dynamic multi-cloud deployment based on user input.

4

Application Development

This project follows the 'Waterfall' development model. This model consists of seven phases as originally described by Royce [40]: "Systems Requirements", "Software Requirements", "Analysis", "Program Design", "Coding", "Testing", and "Operations", although several variations have been proposed along the way, such as the one shown in Figure 4. As its name suggests, phases are executed sequentially in this model; in this respect, only when the current phase is completed, we can move to the next one. However, based on this model, we cannot go back to a previous phase for revision purposes unless the whole current development cycle is finished. [39].

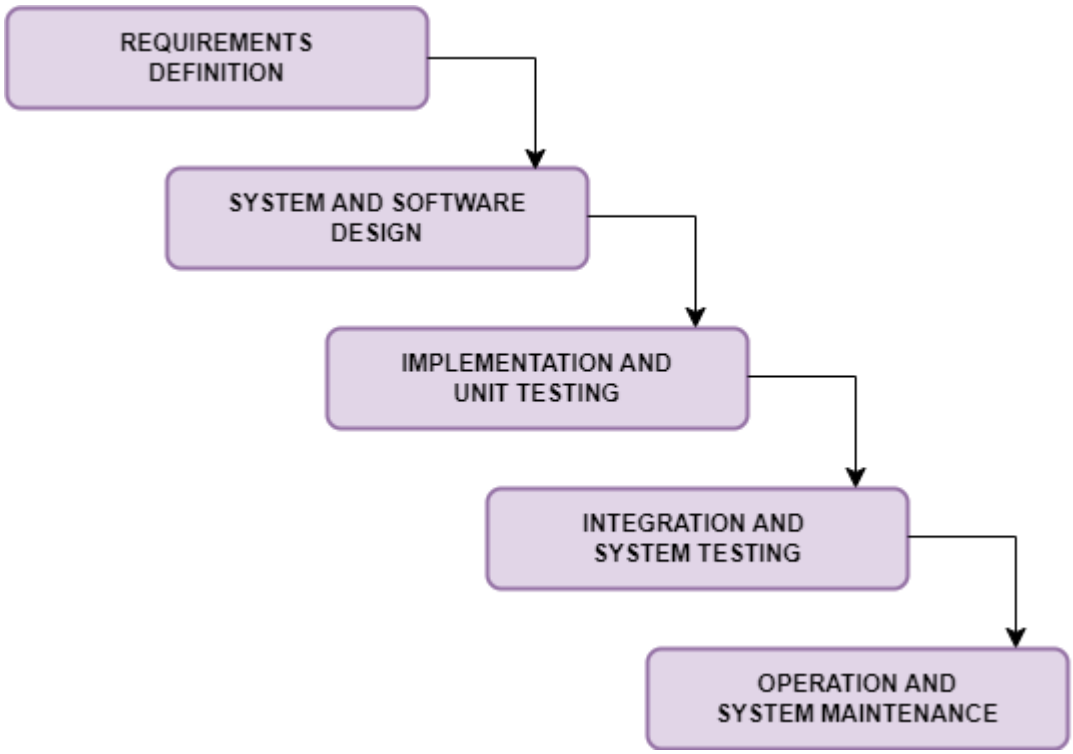


Figure 4. Waterfall software development model

For this project, we started with the requirements definition and analysis. Then the design of the application was carried out through a set of carefully-crafted UML diagrams. In particular, concerning the system design, the context diagram as well as the use case diagrams and the ER diagram were initially modelled; next, multiple sequence diagrams were modelled, each of

which covered one of the main use cases of the system. Then, when the design was completed, the code implementation phase began that was driven by the aforementioned design diagrams modelled. Finally, the system's testing phase took place and the results of this procedure are presented in Chapter 5. In the sequel, we will imprint the main results of each development phase in the following sections of this chapter. The chapter closes with the final section, dedicated to explicating the degree of satisfaction of the requirements set for the implemented system.

4.1 Software Requirements Analysis

In this section, the functional and non-functional requirements are described.

4.1.1 Functional Requirements

The system should offer three categories of management functions: user management, simulation management and simulation execution management. In the following, each category of functions is analyzed separately.

4.1.1.1 User Management

The system should be able to manage its users. The management functions include the following:

- *User registration*: a visitor should be able to register as a user in the system. During the registration request, the following information should be provided: the username, the user's full name, email address, password and optionally, job title, the organization he/she works for and his/her age. The system checks if the same name or email is already used by another user. In this case, the registration will fail and a corresponding error message should be displayed. As for the user's password, it should be hashed before being saved in the database.
- *User deletion*: the user should be able to request to be deleted. This will have the effect of deleting all data concerning the user, including his/her simulations. It is worth mentioning here the users' rights according to their role in terms of this system function. Users with the role of 'Admin' do not have the right to delete their own profile, only that of other users. Simple users, on the other hand, have the right to delete their own profile.
- *User data updating*: the user should be able to update his/her personal information, such as his/her name, age, and email address. An admin can update not only his/her own profile but also that of other users.

  For password reset purposes, it is advisable upon user data updating to also store the answer to three out of five questions which the system will display to the user. This is required for the safe continuation of the password reset process which can be guaranteed by obtaining from the user and validating one answer from these questions. This is explained in detail in the next user management function.

- *Password reset/update*:
  - In case a user wishes to reset his/her password, the system, as mentioned previously, will display a series of questions to the user. As such, the user will

have to give the answer to one of them. If the answer is correct, then a temporary secure password will be created for the user, and sent to his/her email. The user will then be able to request to change it. If the password is not changed within a certain period of time, the user's account will be deactivated.
    - In case of updating the user password, the user will have to provide both the current and the new password (the latter twice).

  Passwords will be subject to certain restrictions (e.g. at least 10 characters including letters, digits and special characters) in order to be considered valid and secure. For the update to be successful, the current password must be correct (i.e., mapping to the password that the user already has) and the new password must be valid and secure. Further, as the new password is given twice for security reasons, the two user-supplied new password values must match.

- *User Authentication*: users should be able to authenticate themselves in the system by providing their username & password. Then, a unique token with a specific validity time limit will be generated, which can be used during users' requests to authorize them.
- *User display*: the system should be able to display all information about a user based on his/her username. This will apply either in the case of an admin who requests to view the information of any user, or in the case the user him/herself requests the display of his/her own profile.
- *User search*: this functionality is accessible only to users with the role of admin. Administrators have the possibility to search for a user by his/her full name or username. The system will return a list of users matching the search terms.

4.1.1.2 Simulation Management

The management functions of a simulation include:

- *Simulation creation*: the user should be able to create a new simulation by providing configuration information for it. In this case, the following information should be specified:
    - *Simulation name*,
    - *Simulation description*: an optional textual description of the simulation,
    - *Desired cloud*: identification of a cloud to be used. It is noted that the user should also provide credentials for the cloud to be used, so that the system can operate on his/her behalf and run the simulation on that cloud. Credentials for a cloud can be provided either each time a new simulation is created, or during user's registration. The system should provide the ability to provide credentials for multiple clouds if the user has relevant accounts on them. The credential information is confidential so it should be sent and stored in a secure manner.
    - *Cloud resource configuration*: the initial amount of resources available for the simulation code's deployment in the desired cloud should be specified. In case of horizontal scaling (especially in a Kubernetes architecture), a maximum number of resources could also be specified.
    - *Simulation code*: the simulation code will either be provided in a compressed form, or its location will be specified (for example the GitHub repository path). The type of code to be used will depend on the system's architecture (serverless computing vs Kubernetes deployments).

  - *Simulation input*: input provided by the user based on which the simulation will be performed. The input may vary depending on certain cases to be supported by the system (design decision). The input affects how the simulation task is divided and this division also depends on the code delivered by the user.

If the simulation is successfully created, the simulation will be automatically associated with the date it was created by the system.

- *Simulation display*: all elements of the simulation should be displayed, along with a list of all runs of the simulation.
- *Simulation update*: the user should have the ability to modify the elements of a simulation. If the update is successful, the simulation may be related with its last modification date. Both the date of creation of the simulation and its last modification are updated automatically by the system and cannot be changed by the user.
- *Simulation deletion*: the user should have the ability to delete a simulation. In this case, all of its elements will be deleted along with all of its runs. The simulation's deletion must be confirmed by the user at the user interface level before it can proceed.
- *Simulation search*: the system should allow the user to search for simulations based on their name or description or the cloud they may relate to. The search can be simple, with the use of keywords or it can be more sophisticated with the combination of keywords and a date range that covers the date of the simulation's creation. The result of the search is a list of simulations that match the input data of the search. Please note that once the user selects one of these results, the data of the corresponding simulation will be displayed (see *simulation display* function above).
- *Simulation execution*: the user should be able to start a new execution of a specific simulation. The execution will be performed asynchronously. The user, through relevant functions/operations, should be able to manage this execution. The result of a simulation execution should be a report that can be visualized by the user and downloaded to his/her local system. In order to initiate the execution of a simulation, the user should supply the following information:
  - *constraints*: different types of constraints could be specified by the user in relation to the simulation, for example, simulation start time limit, simulation end time limit, simulation run time limit, cost. In that case, the system should try to satisfy all these constraints.

4.1.1.3 Management of Simulation Executions

Once the user requests the execution of a simulation, a page may be displayed to the user specifying the status of the simulation execution. The user should be able to refresh the page until the execution of the simulation is complete and the corresponding results of that execution will be displayed. Alternatively, the user will be able to review and manage the executions of a simulation while it is displayed. In that case, the functions of execution management that the system should offer to the user are described below:

- *Simulation execution display*: the user should be able to see general information about the simulation execution, such as simulation name, execution status (started, in progress, completed, stopped, terminated due to errors), execution start date and execution end date (if the execution has been completed or paused by the user). The results of the execution should also be shown to the user even if the simulation is paused

by the user (in this latter case the system should show partial execution simulation) results. Further, the user should be able to download a report of the results of the simulation execution in PDF format to its local system.
- *Simulation execution deletion*: the user should be able to delete a simulation execution and thus all data associated with it.
- *Simulation execution search*: the user should be able to search for a simulation execution based on its status and on a date range which covers either the start or end date (of the simulation execution) depending on the user's choice.

4.1.2 Non Functional Requirements

In this section, the non-functional requirements of the system are described.

- *Good level of usability.* The User Interface (UI) should be user-friendly and responsive, meaning it renders well on a good variety of screens and devices like PCs, Laptops, mobile phones and tablets. It should offer the user unified look-n-feel, proper user orientation, good level of navigability and dynamic experience with balanced use of Ajax & Dynamic HTML.
- *Proper Backend Development.* The backend needs to be separated from the Front-End. Further, its RESTful API must exhibit Level 3 maturity according to Richardson's maturity model [42]. The backend should also support the automatic generation and distribution of its RESTful API documentation (e.g., via swagger).
- *Platform independence*. Concerns the ability of the application/system to be deployed on any platform. This results in the following requirements:
    - The application/system must be containerised. This means that it must be mapped to one or more Docker images through which its containers can be generated. These images might be existing in public image repositories like Docker Hub or can be generated from *Dockerfile*s which must be part of the application's source code distribution.
    - The application should be able to run in clusters of cloud resources via Kubernetes. This means that in its source code appropriate Kubernetes files must exist that model its Kubernetes deployment architecture.
- *Good system performance*. Specifically, each core system function cannot take more than 5 seconds to execute. The simulation execution is performed asynchronously so the user can simply request the start of the execution. This means that this is the function that must satisfy the aforementioned constraint, not the actual execution of the simulation itself.
- *Good system scalability*. The system must have the ability to support multiple concurrent users (e.g. at least 50).
- *Good system availability*. The system must be available all the time.
- *Sufficient system reliability*. This leads to low number of errors/faults per month of operation, proper error/exception handling with self-explanatory messages and proper form validation on both client and server side.
- *Good level of security*. Once user management is implemented, then the following should be supported:
    - *Token-based authentication*.
    - *Role-Based Access Control (RBAC)* or *Attribute-Based Access Control (ABAC)*.
    - *Password hashing*.

- It is also important to implement *Cross-Site Request Forgery (CSRF) protection* as well as *Transport Layer Security (TLS)* support.

4.2 Application Design

We supply below a presentation of the design models of the thesis application along with a brief description for each one. Each type of design model is analysed in its own subsection.

4.2.1 System Environment

The context diagram presented in Figure 5 illustrates the interactions between key entities (Admin**,** User**,** Simulation System**,** Cloud Provider**)** in the simulation management environment.

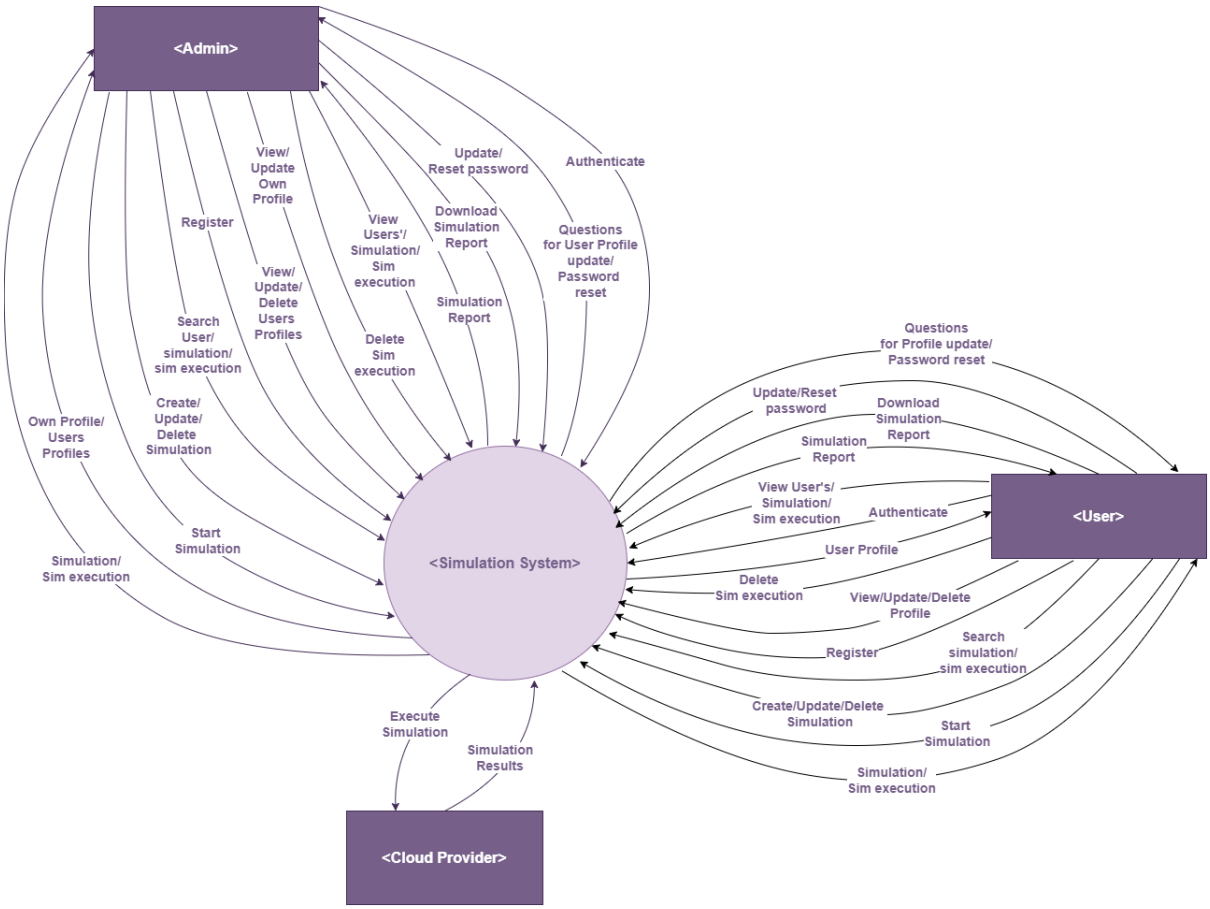


Figure 5. Context diagram of the System environment

As we can see, the admin can register, authenticate, update and reset passwords, view, update and delete other users' profiles, view, update their own profile, create, update, delete simulations, view simulations and simulation executions, search users, simulations and simulation executions, start simulations, delete simulation executions, and download simulation reports. As such, it can be considered as a super-user.

(Registered) Users have the same possibilities, but only for themselves (i.e., in terms of their profiles and their simulations). So, they do not have access to other users' profiles or simulations. They can also delete their own profile, unlike the admins.

The simulation system acts as the central processing unit which sends simulation tasks to the Cloud Provider for execution and retrieves the respective results. It also shows simulations and their executions to the user/admin as well as their profile, or other users' profiles in the case of admin. It also returns the simulation report to the user/admin with the ability to be downloaded into their system.

The Cloud Provider executes simulations as requested by the Simulation System and returns the simulation results for further processing or reporting.

4.2.2 Use Case Diagrams

The use cases and the main system actors involved in them are presented in Figure 4 below. Two use case diagrams have been created, one considering the users' management and one considering the simulations' management. Each actor has a different role to play within the system. The use cases depicted in each diagram are the basic (system) functions through which each actor/role interacts with the system.

4.2.2.1 Use case diagram for Admin and Registered User (user management)

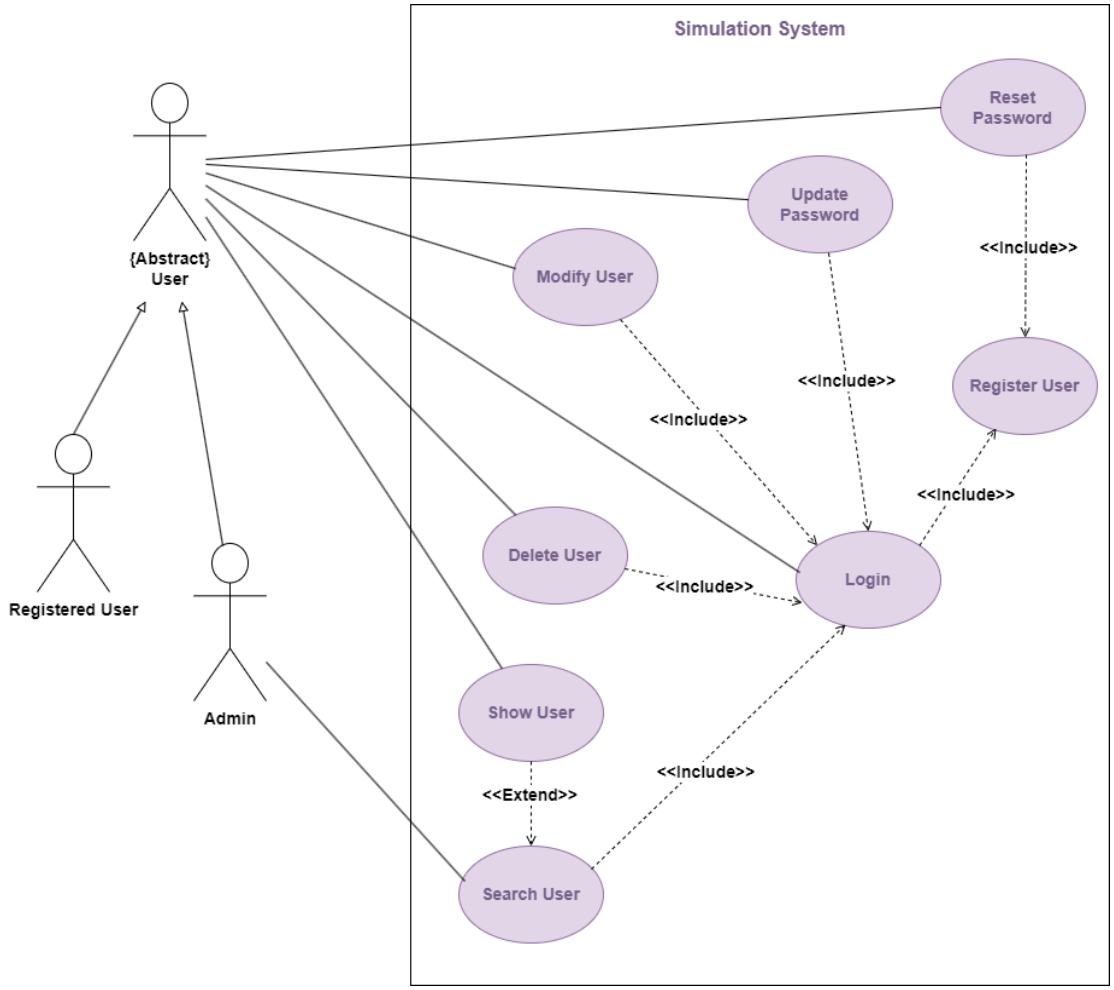


Figure 6. Use case diagram for user management

This use case diagram, shown in Figure 6s, represents the user management functionalities within the Simulation System, focusing on how different types of users interact with the system to manage user-related operations.

At the top level, the diagram introduces an abstract user actor, which specialises into two roles: Registered User and Admin. Both of these roles inherit the ability to perform actions/functions defined for the abstract user, but certain functions like deleting or modifying other users are restricted to the Admin.

The system includes several use cases such as' Register User', 'Login', 'Update Password', 'Reset Password', 'Modify User', 'Delete User', 'Show User', and 'Search User'. The 'Login' use case is central, being included in several other use cases like registration, password update, and reset, indicating that user authentication is a prerequisite for these actions. This dependency between 'Login' and the other use cases is shown by the respective <<include>> relationships.

Additionally, there is the <<extend>> relationship where 'Show User' can be optionally executed within the execution of 'Search User', suggesting that searching user profiles might involve viewing a user's profile. This is quite normal as when user profile search is conducted,

the respective results can be links towards actual user profiles. So, clicking on one of them can lead to viewing the respective user profile.

4.2.2.2 Use case diagram for User and Cloud Provider (Simulation management)

Figure 7. Use case diagram for simulation management

This use case diagram in Figure 7 illustrates the functional interactions between User/Cloud Provider and the Simulation System in the context of managing simulations. The User engages with the system to perform the functions: Create Simulation, Update Simulation, Delete Simulation, Start/Execute Simulation, Search Simulation, and the related operations for Simulation Execution.

The relationships <<include>> and <<extend>> define dependencies and optional behaviors, like 'Create Simulation', which is a foundational (system) function that is included in the 'Search', 'Update', and 'Delete Simulation' use cases, indicating that creating a simulation is a necessary component or prerequisite of these functions. Additionally, 'Search Simulation' can be extended with 'Show Simulation', meaning that viewing detailed results is an optional

follow-up to a search. A similar relationship exists between ‘Search Sim Execution’ and ‘Show Sim Execution’.

The Cloud Provider participates in the ‘Start/Execute Simulation’ use case, representing its main role in carrying out the actual simulation processing and returning the respective results. This highlights the system's reliance on an external infrastructure for simulation execution, ensuring scalability and performance.

4.2.3 System Behavior Modeling via Sequence Diagrams

Sequence diagrams show the interactions between actors and the system as well as between system components. Their horizontal axis depicts the interacting parts, in our case actors as well as the system’s frontend, backend and database, while their vertical axis shows the chronological order of the interactions (between these parts).

4.2.3.1 User Registration

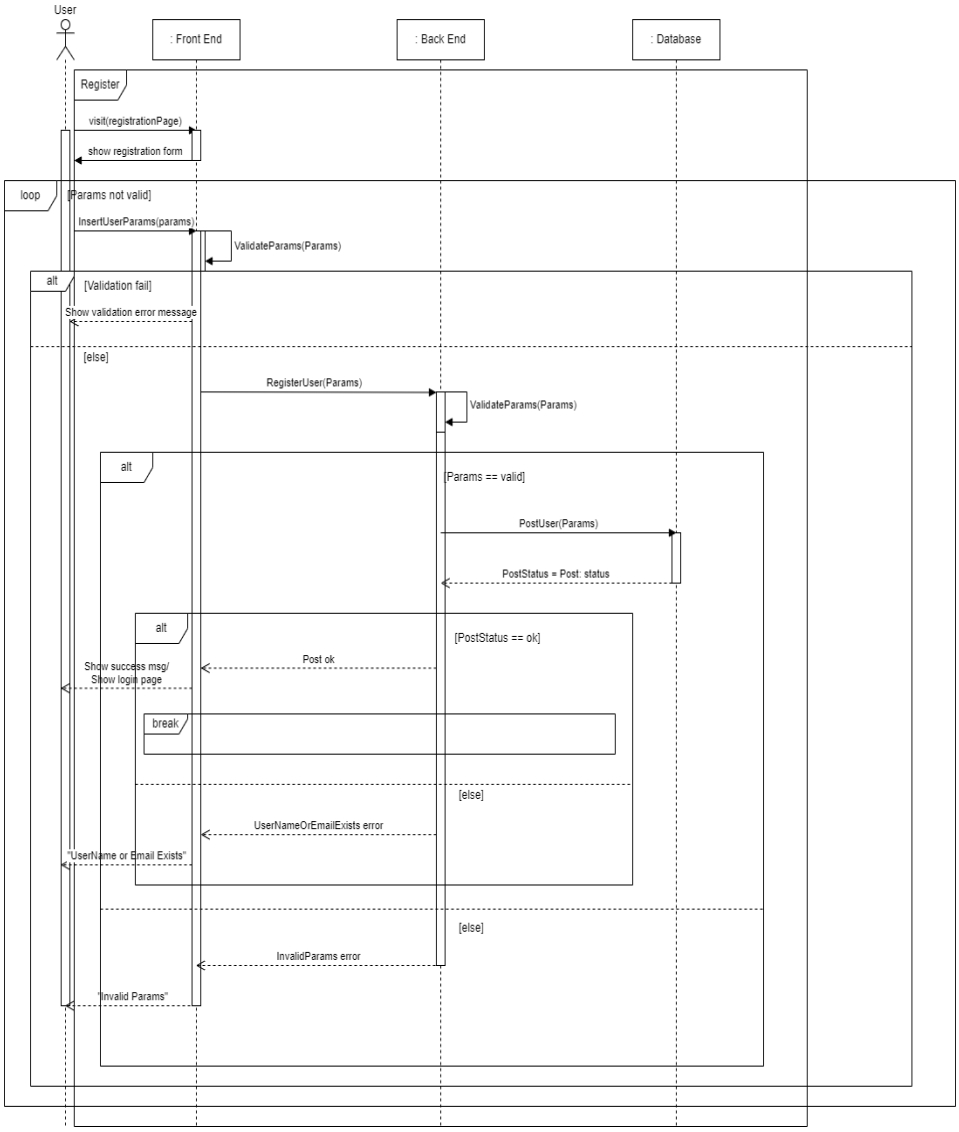


Figure 8. User Registration

The sequence diagram in Figure 8 shows the user registration process. First, the user visits the registration page, prompting the frontend to display the registration form. Upon submitting the form, the frontend checks whether the provided parameters are valid. If the parameters check fails, an error message is shown; otherwise, they are forwarded to the backend for further validation. If everything is correct, the backend attempts to store the new user data in the database. A successful response from the database results in a confirmation message and the front end redirects the user to the login page. Otherwise, an error message is returned and shown to the user. As the system enables the user to resubmit the form after a respective error is detected and shown, a loop has been modelled to indicate that the steps described above, from form submission and on, are repeatedly executed until user registration is successful.

4.2.3.2 User Login

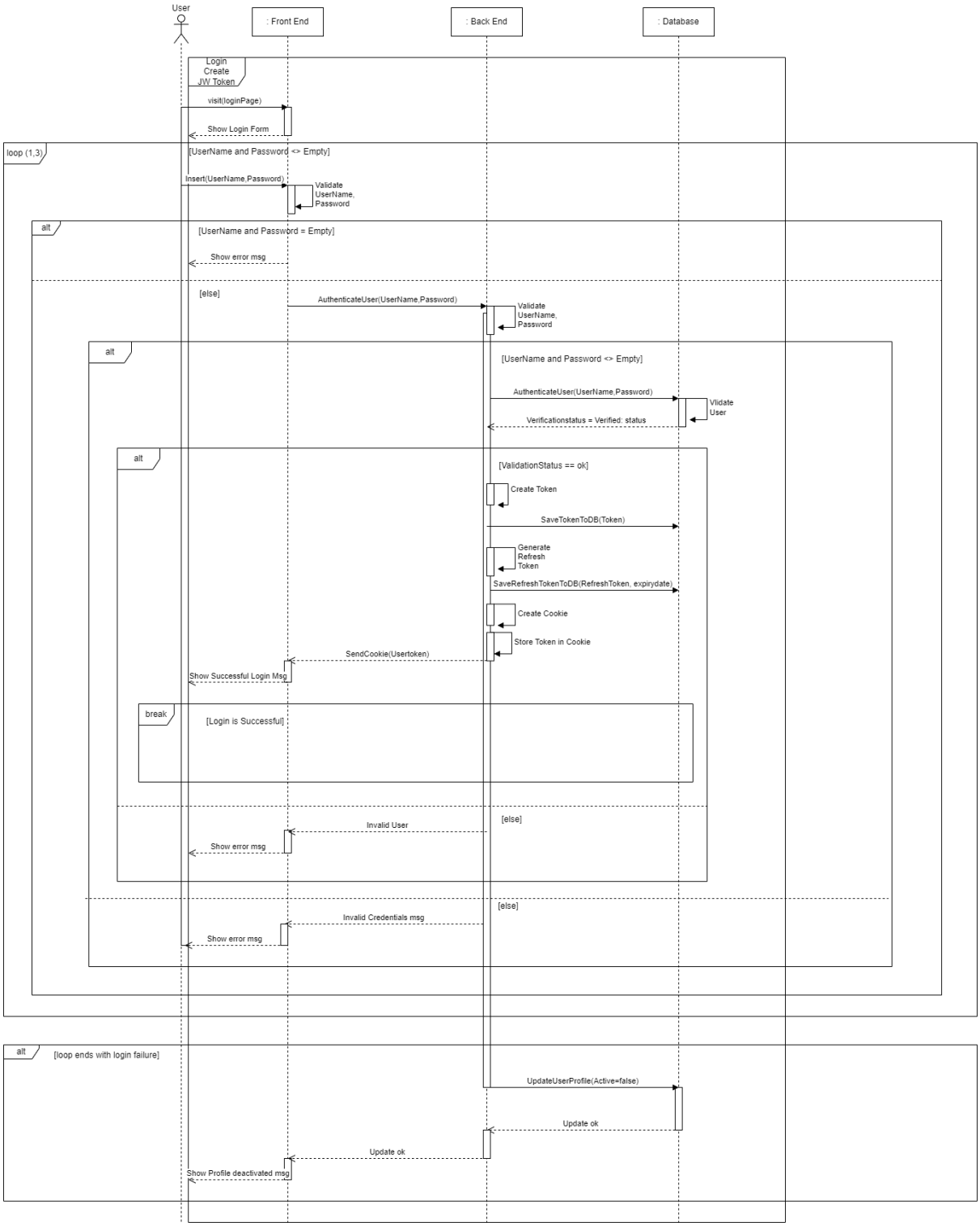


Figure 9. User Login

The login process represented in Figure 9 utilizes JWT (JSON Web Token) tokens and HTTP-Only cookies for user authentication. At first, the user visits the login page, enters his/her username and password, and submits the form. The frontend then validates the input and if it is valid, it is forwarded to the backend for authentication. The backend validates the provided credentials once again and if they are not empty, it verifies them against the stored ones in the database. If the credentials are valid, a JWT and a refresh token are generated and stored in the database, while a cookie containing each token is created and sent back to the client. A success message is then displayed to the user. If the login fails for three times in a row (see the respective loop modelled), the user becomes inactive and an appropriate error message is shown.

4.2.3.3 User Authorization

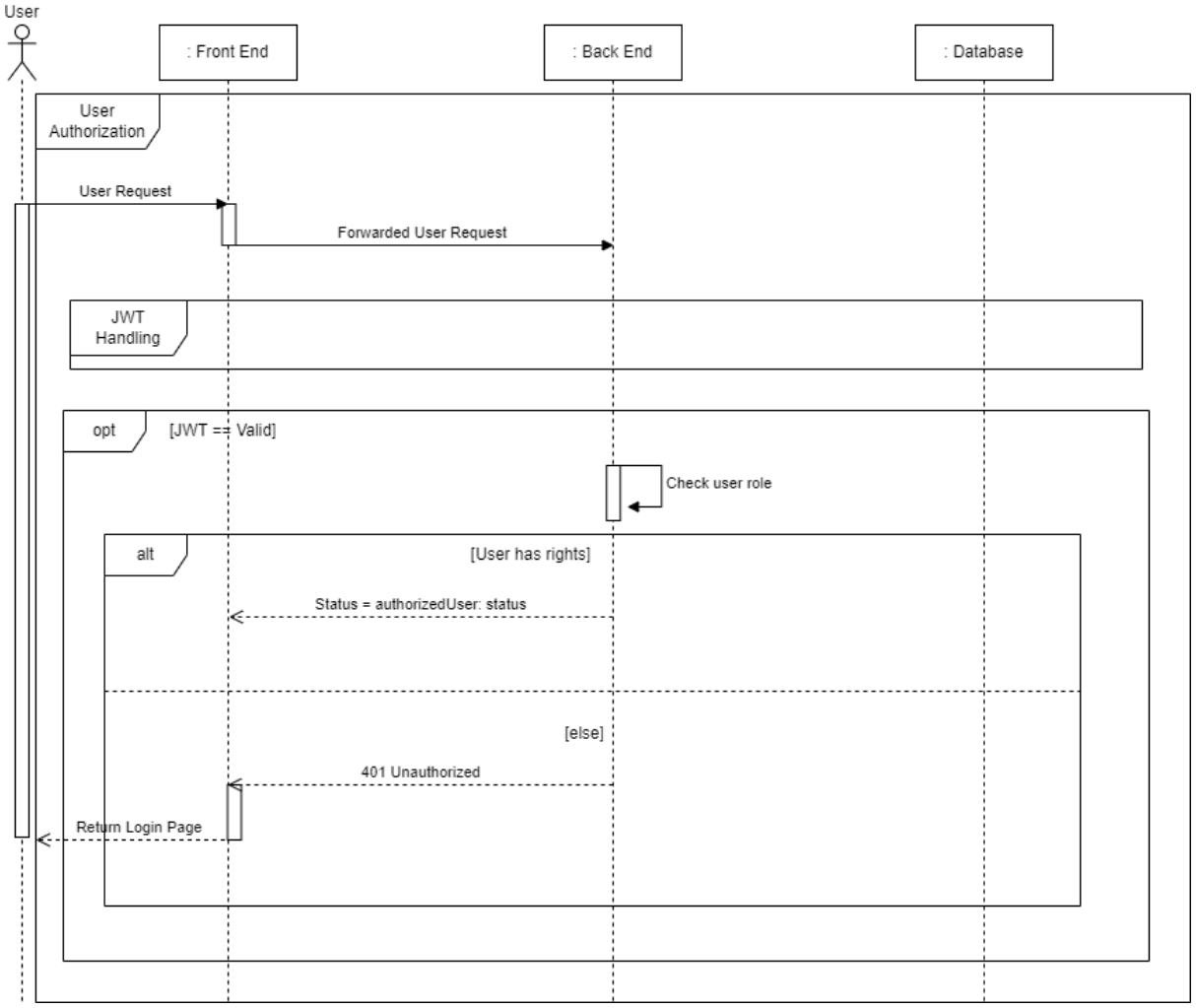


Figure 10. User Authorization

Figure 10 illustrates the user authorization process using JWT (JSON Web Token). The process begins when a user sends a request from the front end, which is then forwarded to the backend. The JWT handling process is then performed in order to validate the token. If the JWT is valid, the system proceeds to check the user's role in order to grant user permissions. If the user has the appropriate rights, the system returns a successful authorization status code; otherwise, a

401 Unauthorized response is sent, and the user is redirected to the login page. This ensures secure access control based on the role-based access control (RBAC) approach.

4.2.3.4 JWT Handling

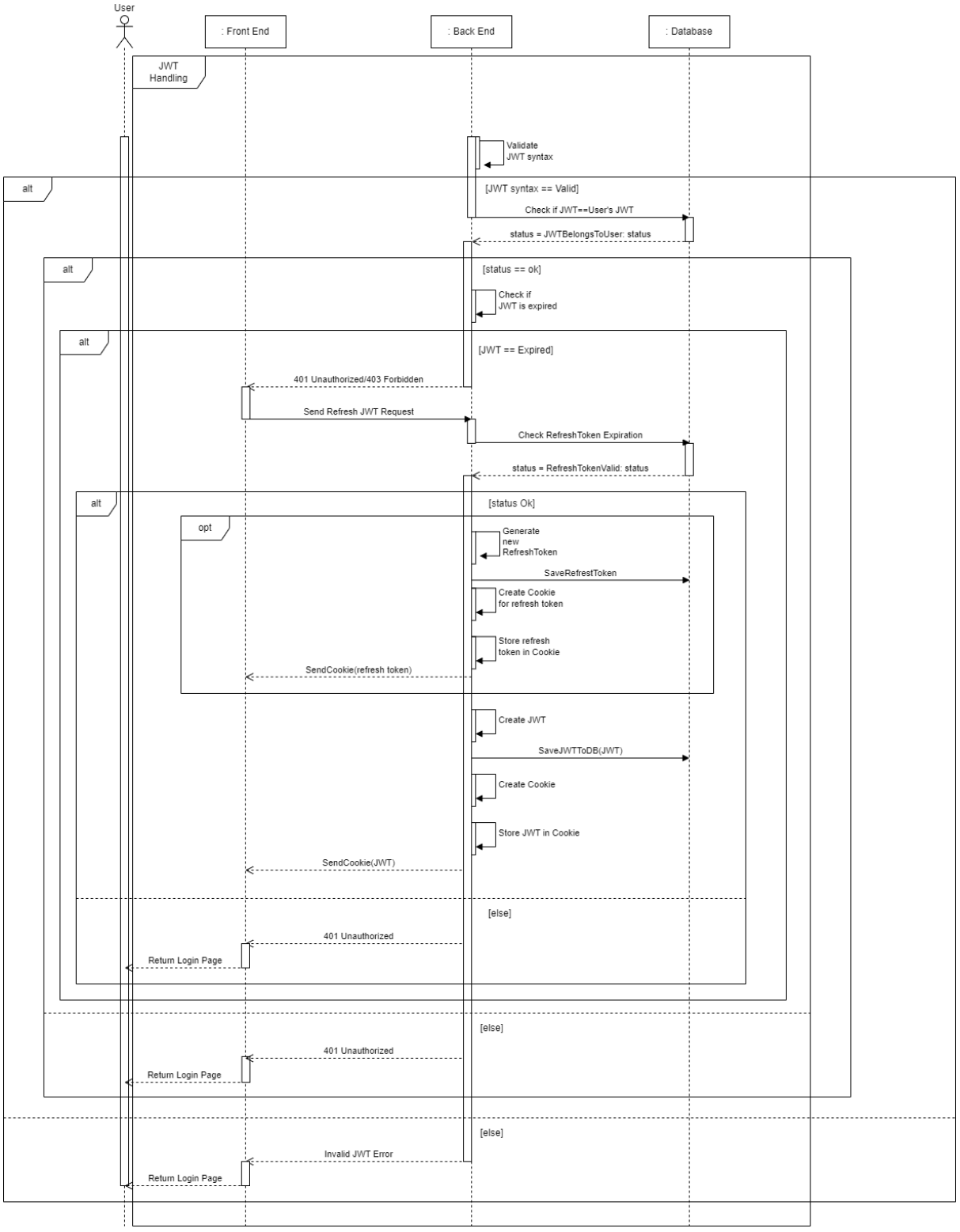


Figure 11. JWT Handling

This sequence diagram in Figure 11 outlines the JWT handling and refresh mechanism. The process begins with the JWT validation at the backend, including syntax checks and verifying that the token belongs to the user making the request. If the token is valid and not expired, the request proceeds. If the token is expired, a 401 Unauthorized status code is returned to the client and a request to refresh the JWT is triggered. The backend checks the validity of the refresh token (sent by the frontend through the user cookie); if it is valid, a new refresh token and JWT are generated, stored in the database and in HTTP-Only cookies as well, and the cookies are sent back to the client. Then, the user's request proceeds without any user's action interruption. If any step fails, such as an invalid JWT, expired refresh token, or user mismatch, a 401 Unauthorized error is returned and the user is redirected to the login page. The use of refresh token enhances the user's interaction with the system as the JWT's expiration is kept small to avoid the token been leaked and the refresh token produces new JWT without forcing the user to frequently login.

4.2.3.5 User Logout

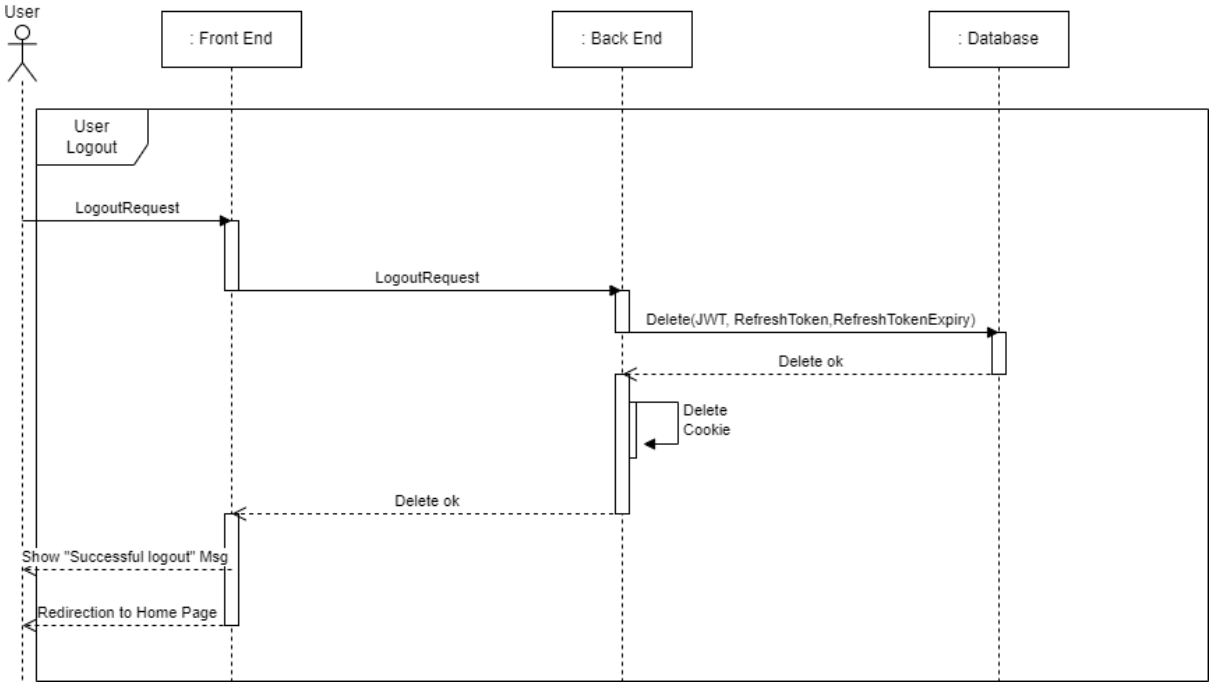


Figure 12. User Logout

Figure 12 describes the logout process. Initially, the user requests to logout from the system. The request is forwarded to the backend, which then deletes the tokens and the information related to them from the database. The respective user cookies are also deleted and a message of success is shown to the user before the redirection to Home page (which the user can view as a visitor).

4.2.3.6 User Search

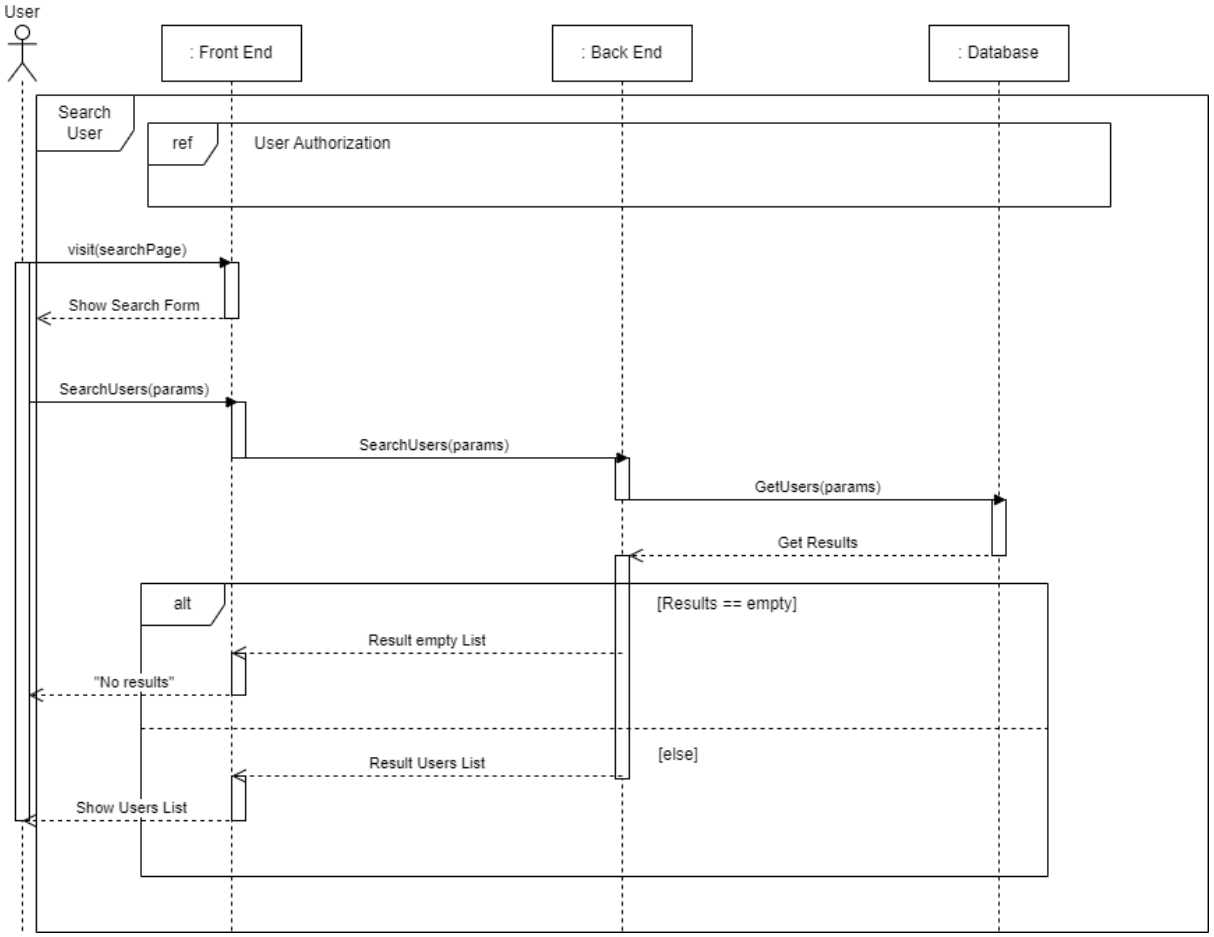


Figure 13. User Search

Figure 13 shows the user search functionality. The process begins when an authorized user (especially an admin) visits the search page and is presented with a search form. Upon entering the search parameters, the front end sends a request to the backend, which in turn queries the database for matching users using the provided parameters. The backend retrieves and returns the search results. If no users match the search criteria, a “No results” message is shown. Otherwise, the system displays the list of matched users.

4.2.3.7 User Show

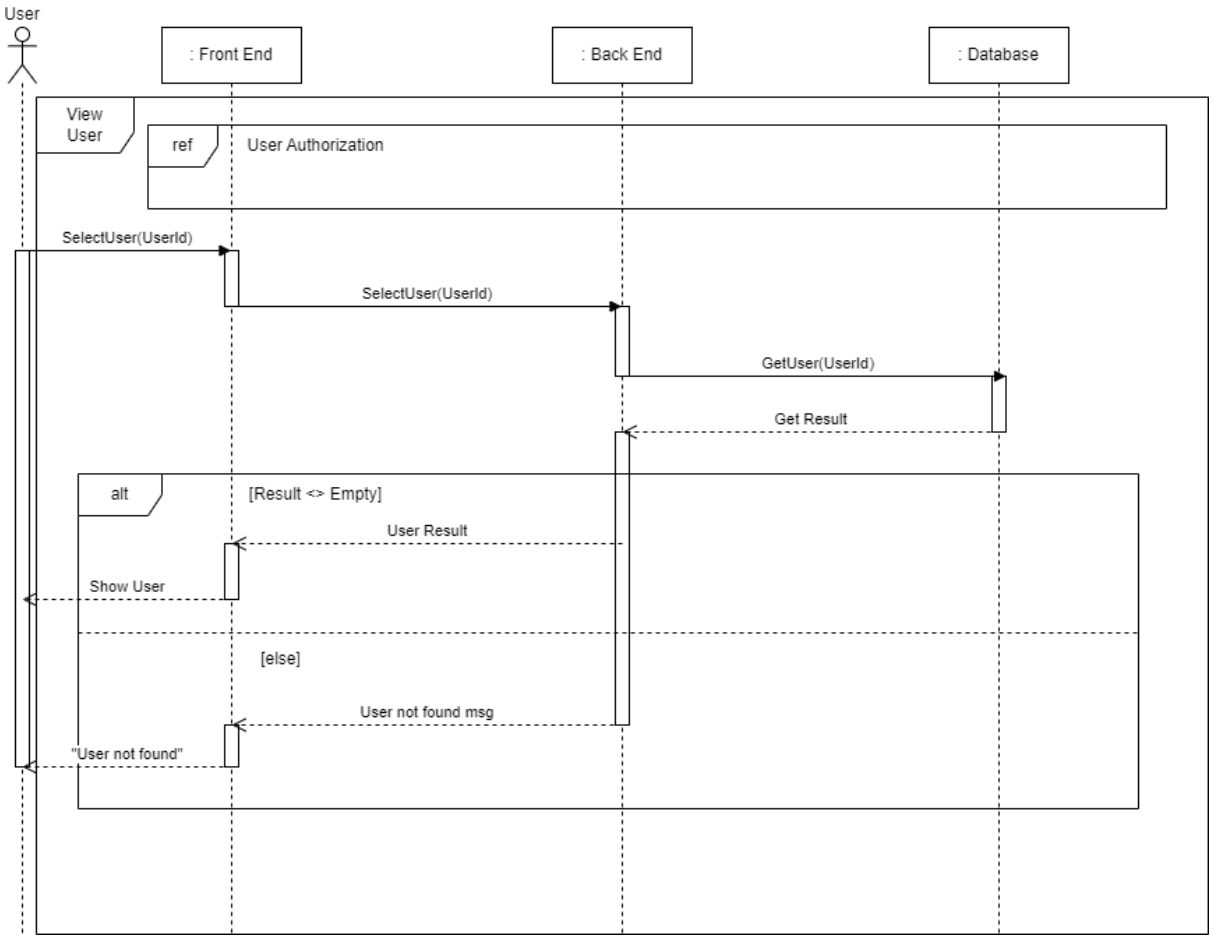


Figure 14. User Show

This sequence diagram in Figure 14 illustrates the view user functionality. An authorized user (e.g., an admin) selects a specific user from a list to view their profile. A request is then sent from the frontend to the backend, which queries the database for the user that matches the selected userId. The backend retrieves and returns the respective result. If the result is not empty, the selected user's profile is shown to the user; otherwise a 'User not found' message is returned.

4.2.3.8 User Update

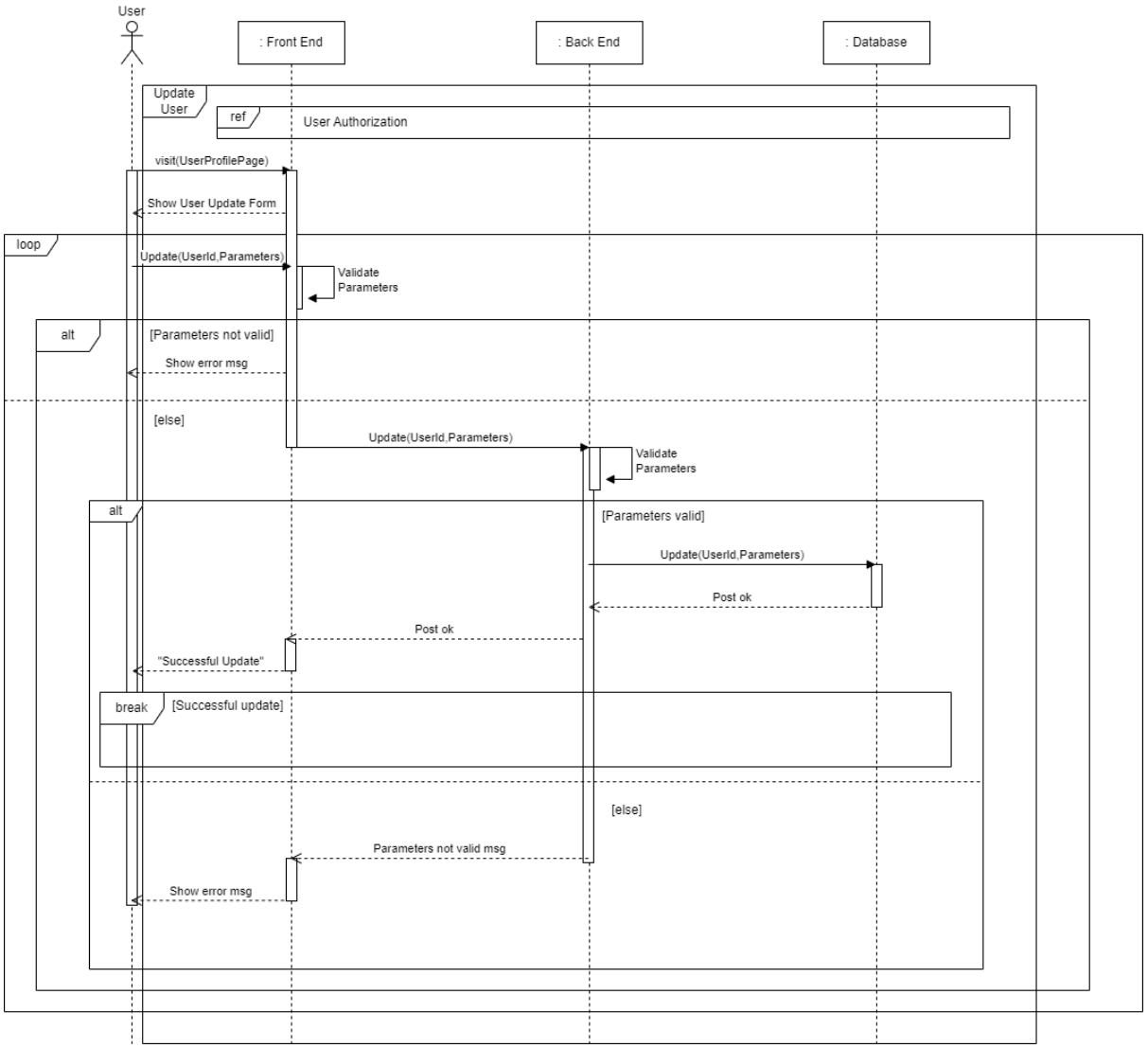


Figure 15. User Update

The sequence diagram in Figure 15 illustrates the user profile update process. It begins when an authorized user accesses their profile page and is presented with a user update form. The user submits the parameters to be updated, which the front end first validates. If the parameters are invalid, an error message is displayed, and the user is prompted to correct his/her input. If the parameters pass validation, the request is forwarded to the backend for further parameter validation and then it is sent to the database for updating the user record. Upon a successful update, the system confirms with a success message. If the update fails, an appropriate error message is returned, and the overall (user update) process may repeat.

4.2.3.9 User Delete

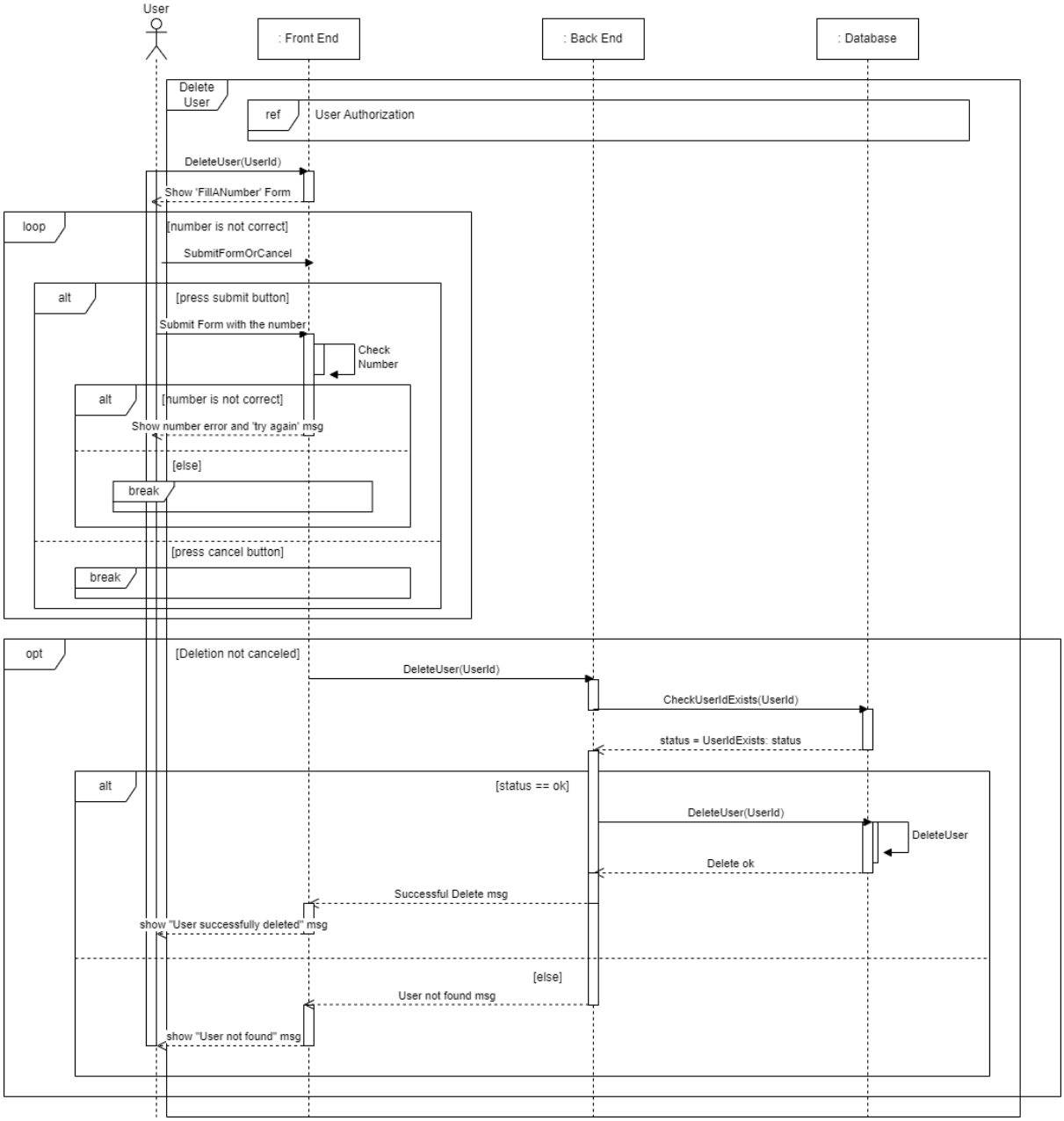


Figure 16. User Delete

The sequence diagram in Figure 16 represents the user deletion process in the system. The user (i.e., an admin), after authorization, initiates the deletion of another user by requesting to delete that user's account. The system presents a confirmation window requiring the user to input a specific number. The user can either cancel or proceed by supplying that number. If the supplied number is incorrect, an error message prompts the user to try again. Upon entering the correct number and confirming, the front end sends the deletion request to the backend which in turn, verifies whether the user exists or not. If the user is found, the deletion is executed and a success message is shown; otherwise, an error message is returned.

4.2.3.10 User Profile Delete

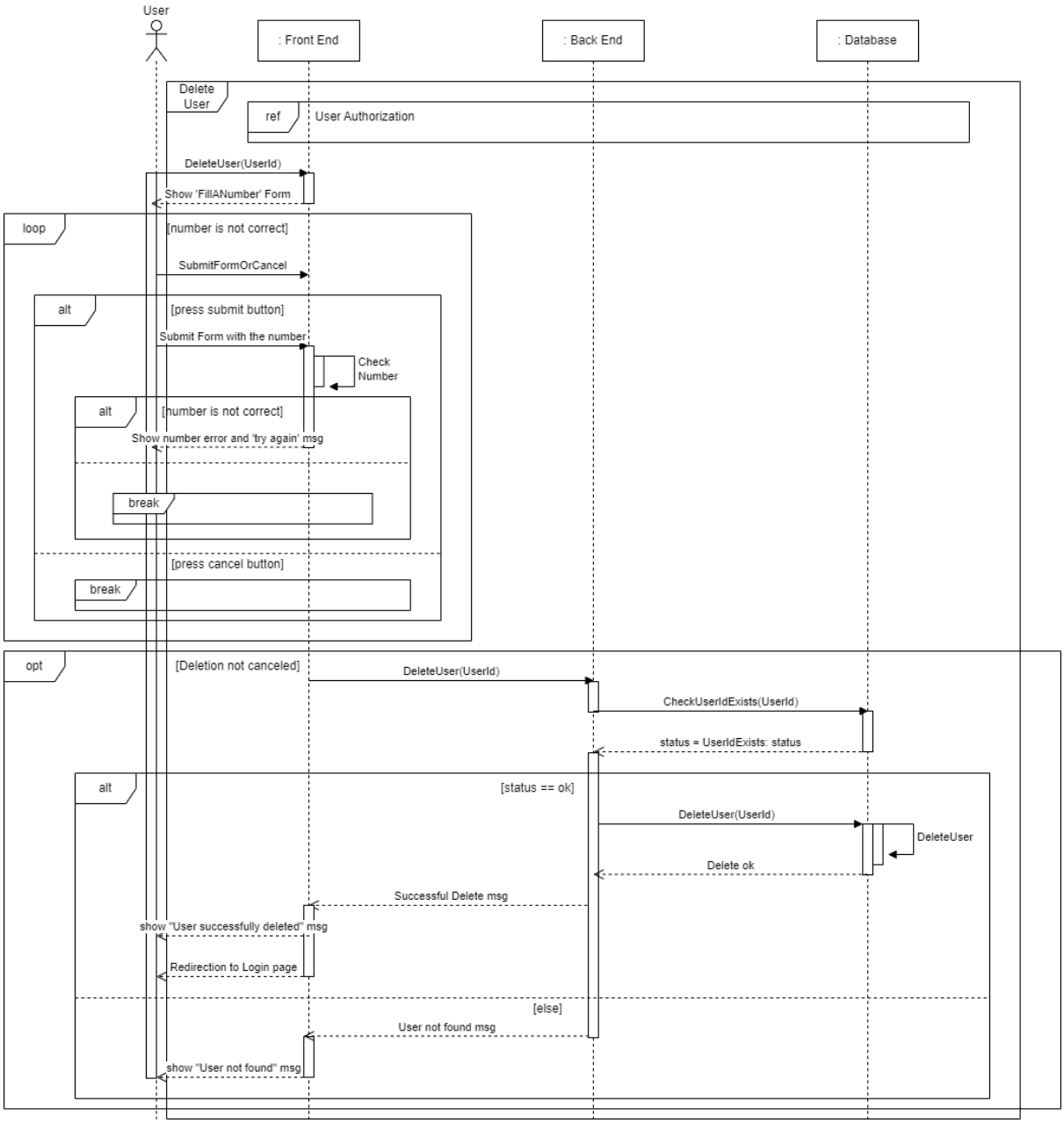


Figure 17. User Profile Delete

The sequence diagram in Figure 17 illustrates the user account/profile deletion process. The user (i.e., a normal user) begins by initiating a deletion request, triggering a frontend form requiring a number input for confirmation. If the supplied number is incorrect, an error is shown and the user may retry or cancel. If the correct number is provided, the system verifies the user's existence in the database. Upon successful validation, the backend deletes the user's profile and the frontend notifies the user of the successful deletion. At the same time, the user is redirected to the login page. If the user does not exist, an appropriate error message is displayed.

This sequence diagram (User Profile Delete) concerns the user's own profile deletion, a process which the admin does not have the right to perform. When the profile is deleted, the login page is shown as the user no longer exists in the database. The previous (User Delete) diagram depicts the deletion process of other users that only the admin has the right to perform. Upon successful deletion, the updated list of users is shown to the admin.

4.2.3.11 User Password Update

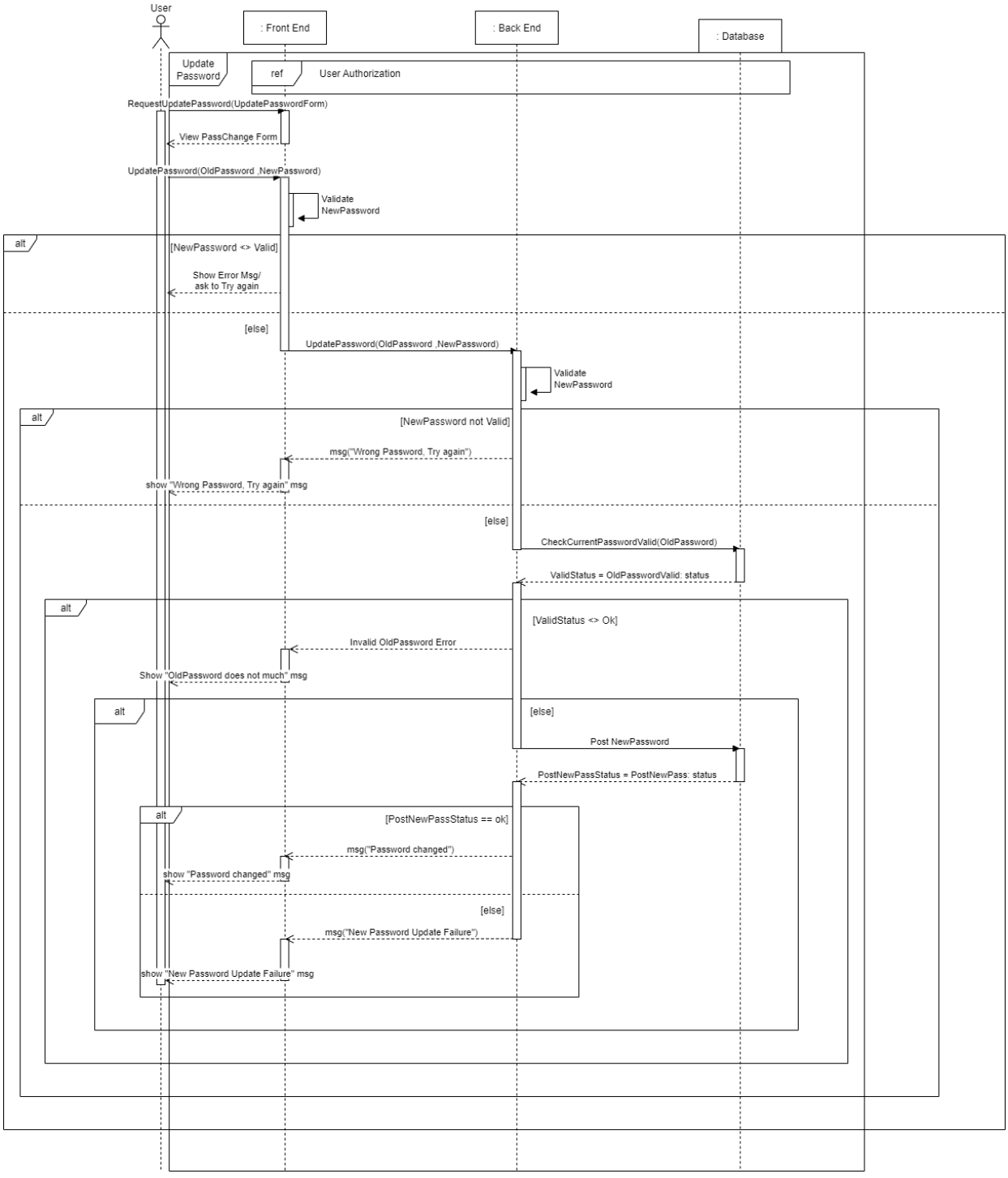


Figure 18. User Password Update

The sequence diagram in Figure 18 outlines the process of updating a user's password. The user begins by requesting the password change form. After filling out the form, the new password is validated on the front end. If invalid, an error is shown immediately; otherwise, the backend further checks the new password and validates the current (old) password via the database. If the old password does not match (i.e., the password given by the user does not match the one stored in the database), an error is returned. If both validations pass, the new password is attempted to be inserted in the database. Finally, depending on whether the update succeeds, the system either confirms the password change with a success message or returns a failure message indicating that the update was unsuccessful.

4.2.3.12 User Password Reset

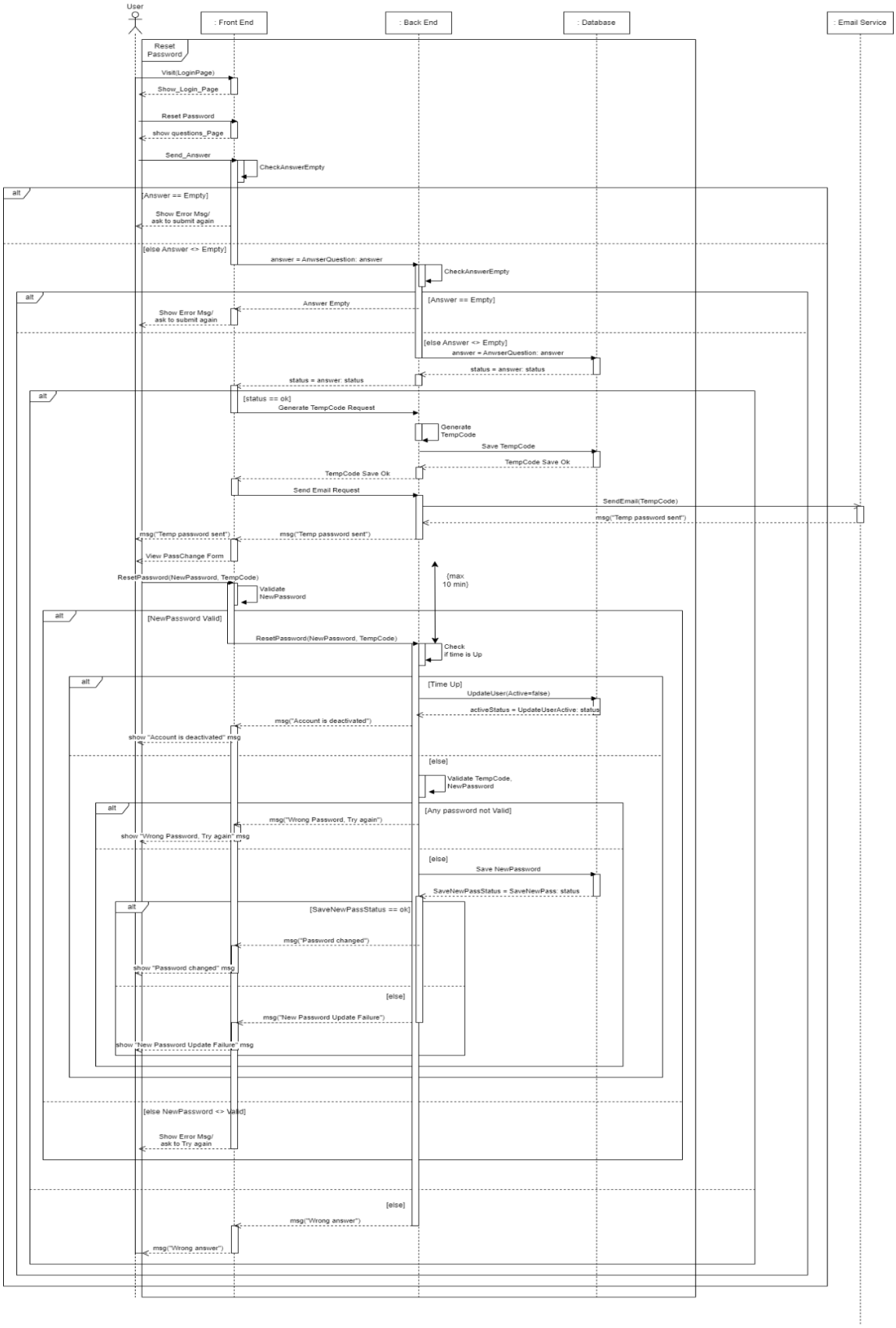


Figure 19. User Password Reset

The sequence diagram in Figure 19 illustrates the user password reset process. The user begins by visiting the login page and selecting the reset password option. A security question is then shown, and the user's answer is submitted and checked. If the answer is empty or incorrect, the user is prompted to try again. If the answer is valid, a temporary password reset code is generated, saved in the database, and sent via email. The user then accesses a password change form, where he/she enters a new password along with the temporary code. The system checks if the reset code is supplied within a 10-minute validity window and validates the new password. If it is not within 10 minutes or the new password is invalid, the reset process fails and an error message is displayed. If all checks pass, the new password is attempted to be saved. If this update is complete, a success message is shown; otherwise, an error message is displayed.

4.2.3.13 Simulation Create

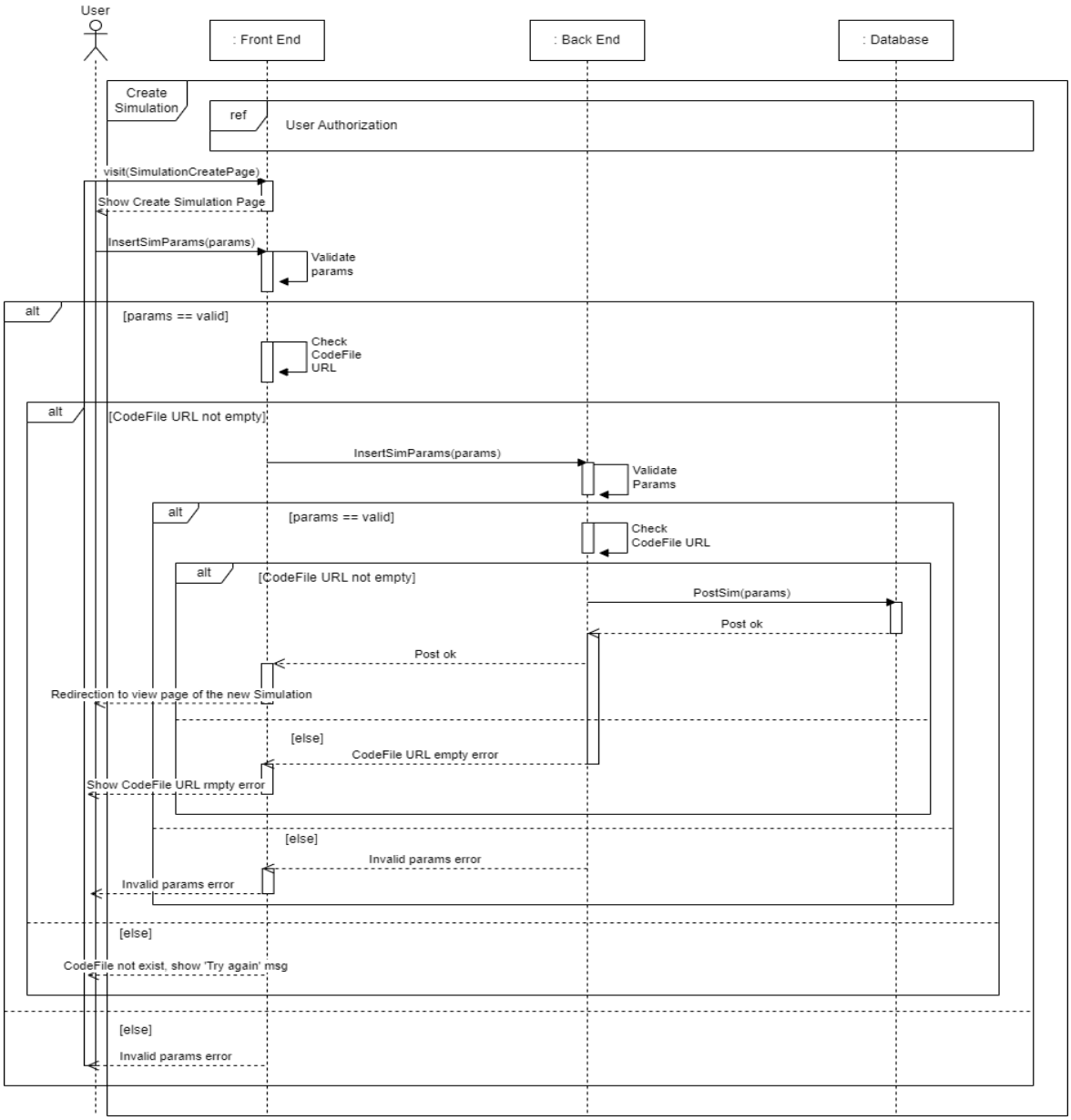


Figure 20. Simulation Create

The sequence diagram in Figure 20 illustrates the process of creating a simulation within the system. The authorised user begins by accessing the simulation creation page, where he/she inputs the necessary parameters. The front end first validates the input parameters and ensures that the CodeFile URL is not empty. If both checks pass, it sends the parameters to the backend. At the backend, the parameters are revalidated, and the CodeFile URL is checked again. If everything is valid and the URL is not empty, the backend posts the simulation data to the database and confirms successful creation. The user is then redirected to view the new simulation just created. However, if any step fails, such as invalid parameters or an empty/missing code file URL, the system returns specific error messages to guide the user to correct the input and retry the process.

4.2.3.14 Simulation Show

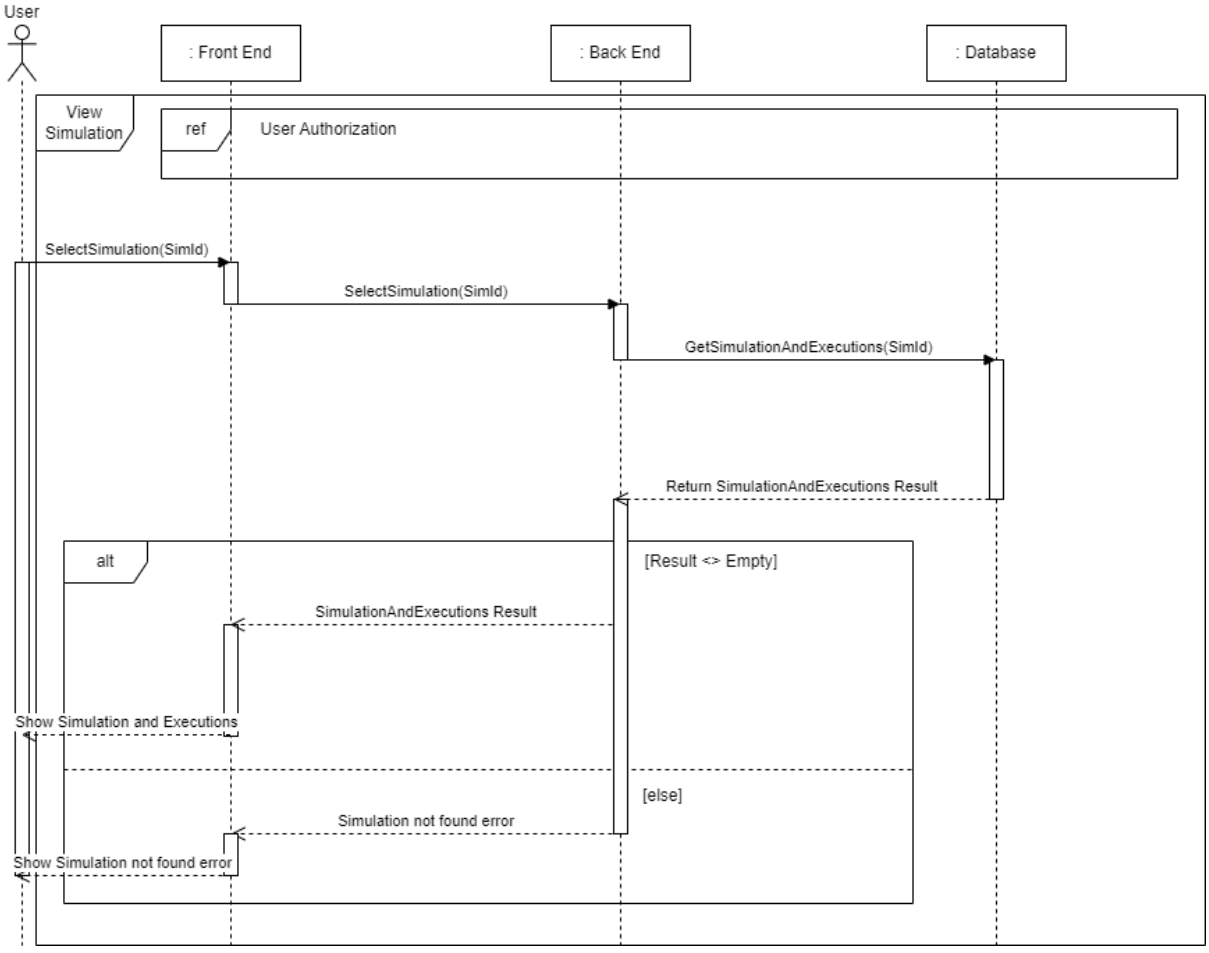


Figure 21. Simulation Show

The sequence diagram in Figure 21 illustrates the view simulation functionality. The authorised user selects a simulation from a list of simulations to view its information. In this case, a request is sent from the front end to the backend, which queries the database for the selected simulation. The backend retrieves the result and returns it to the front end. If the result is not empty, the requested simulation is shown to the user along with its executions; otherwise, a 'Simulation not found' message is returned.

4.2.3.15 Simulation Search

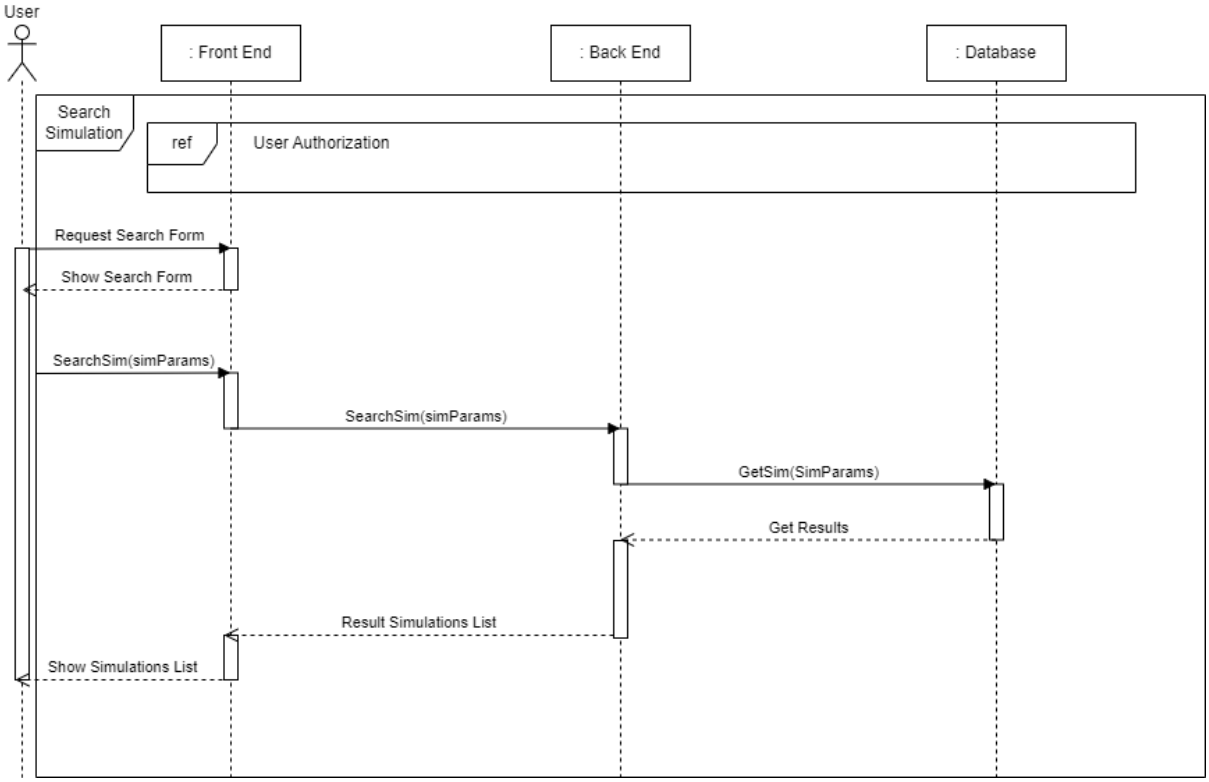


Figure 22. Simulation Search

The sequence diagram in Figure 22 illustrates the simulation search functionality. The process begins when an authorised user visits the search page and is presented with a search form. Once the user enters the search parameters, the front end sends a respective request to the backend, which in turn queries the database for matching simulations using the provided parameters. The backend retrieves and returns the search results and the system displays them in the form of a list of matched simulations.

4.2.3.16 Simulation Update

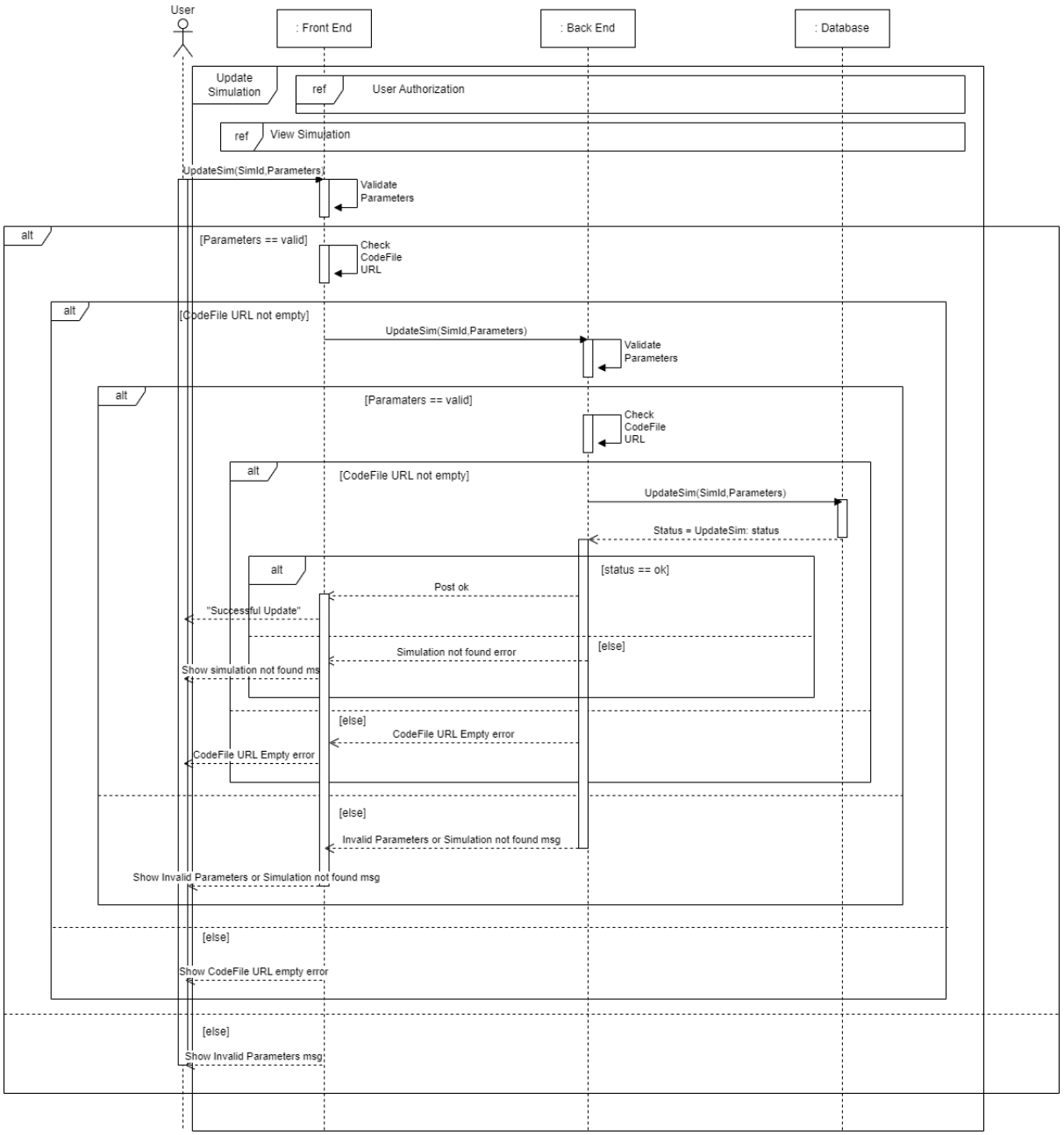


Figure 23. Simulation Update

The sequence diagram in Figure 23 depicts the process of updating an existing simulation. Initially, the system performs user authorization and displays the simulation data. The front end then validates the updated parameters (submitted by the user) and checks whether the CodeFile URL is provided. If both conditions are met, the updated parameters are sent to the backend for further validation. The backend once again checks the parameters and the CodeFile URL before attempting to update the simulation. If the update is successful, a success message is shown. Otherwise, appropriate error messages are returned depending on the issue, such as missing or invalid parameters, empty CodeFile URL, or a not found simulation.

4.2.3.17 Simulation Delete

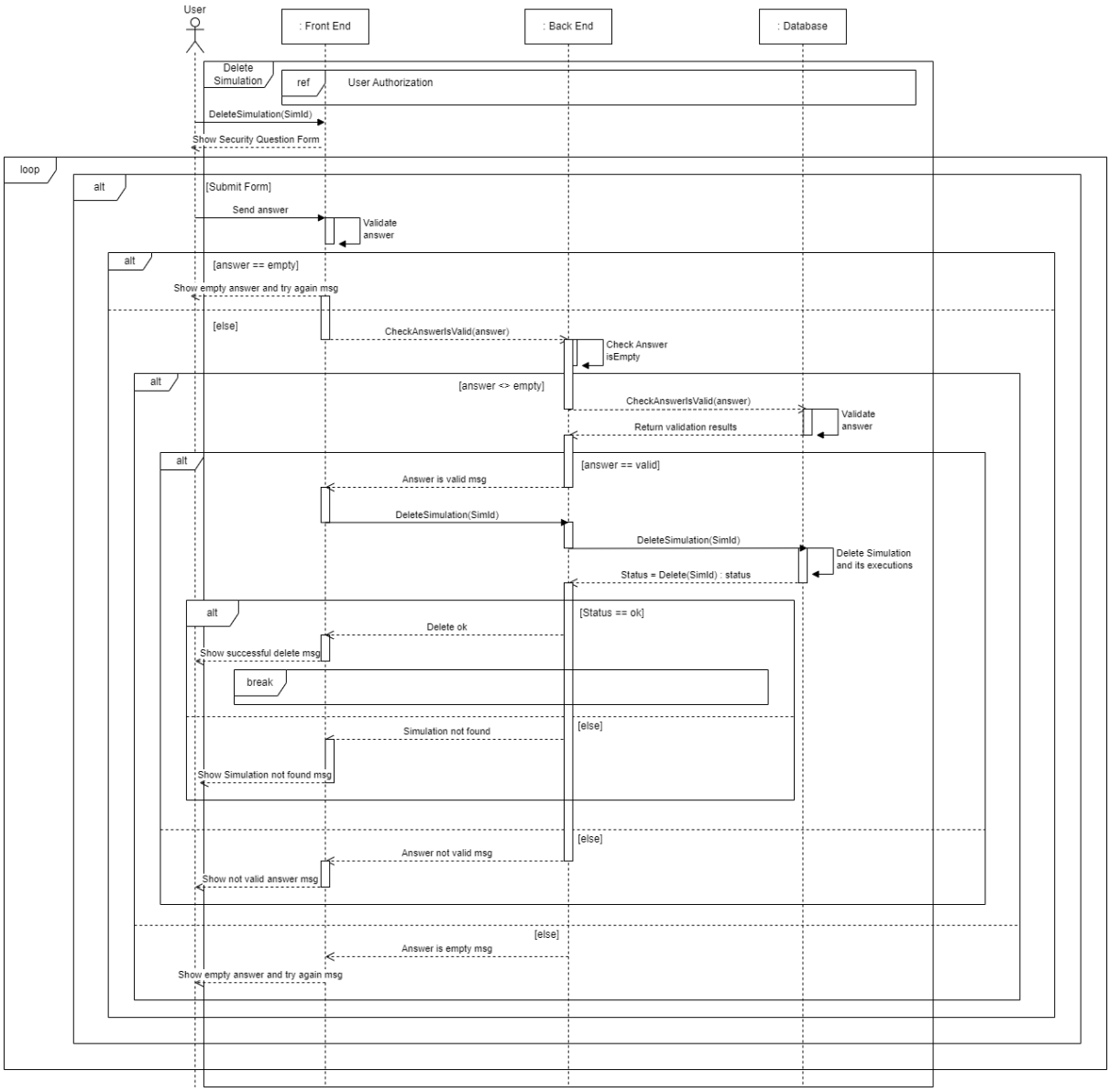


Figure 24. Simulation Delete

The diagram in Figure 24 illustrates the process of deleting a simulation. The user initiates the deletion after authorisation by answering a security question. The front end validates the provided answer, then sends it to the backend for verification. If the answer is empty, an error message prompts the user to try again. Once the answer is verified as valid, the frontend proceeds to request the deletion of the simulation from the backend. The backend then validates and deletes the simulation along with its executions in the database and a success message is shown to the user. Otherwise, appropriate error messages are returned depending on the case, e.g., due to an invalid (security question) answer or a not found simulation.

4.2.3.18 Simulation Run

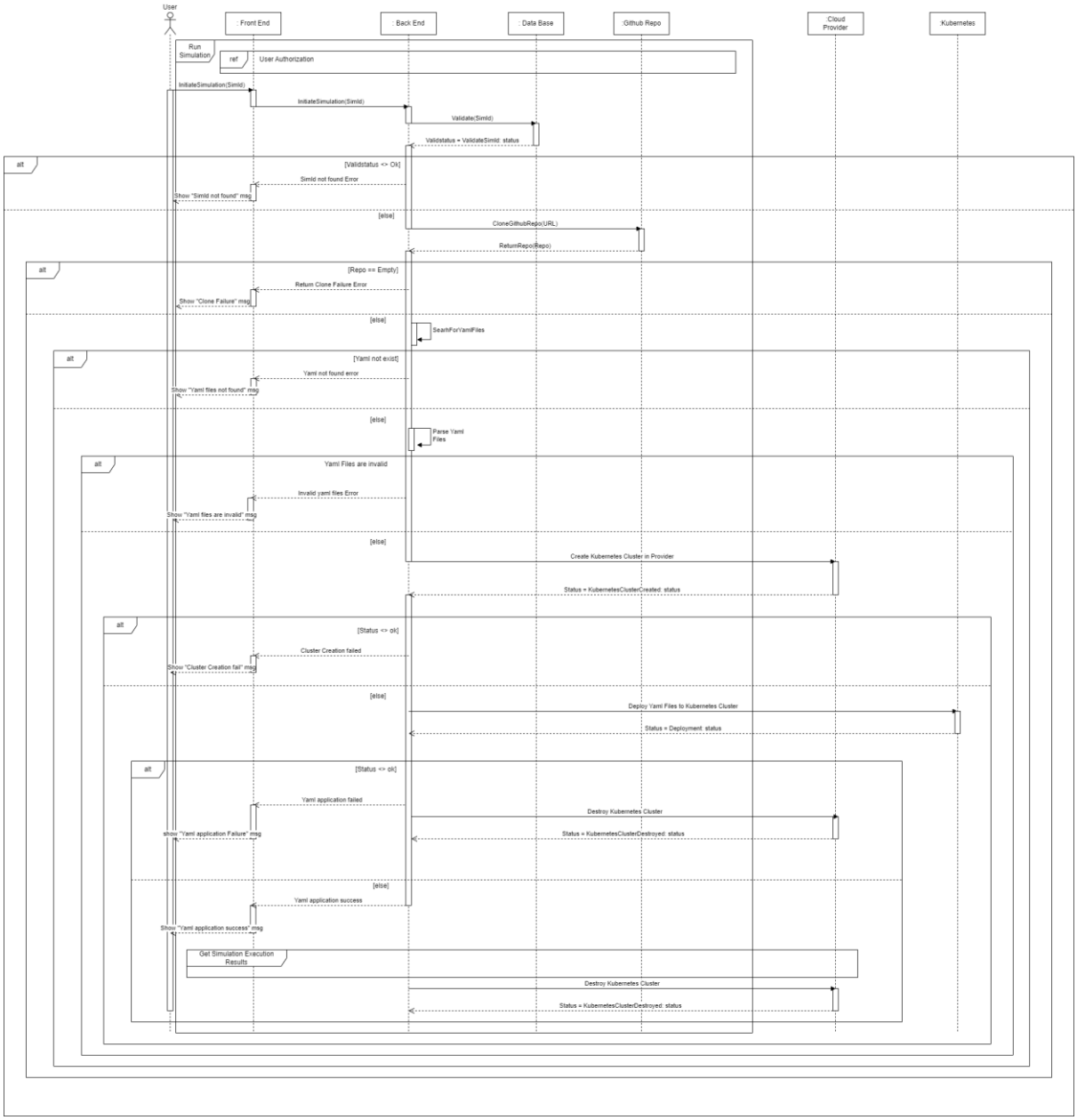


Figure 25. Simulation Run

The diagram in Figure 25 illustrates the process of running a simulation. The authorised user initiates the simulation and the front end sends the respective request to the backend, which in turn validates the simulation's id via the database. If the simulation exists, the system attempts to clone the GitHub repository from the simulation's URL and fetch the required YAML configuration files from the repository. If the YAML files are retrieved and are valid, the backend interacts with a cloud provider (the one associated with the simulation by the client) to create a Kubernetes cluster. Once the cluster is successfully created, the system deploys the YAML files to the cluster. If the deployment succeeds, the simulation executes and the results

are retrieved. In case of any failure, such as missing files, invalid YAML content, or errors during deployment, the system returns appropriate error messages. The Kubernetes cluster is destroyed in order to clean up resources either in case of deployment failure, or after the successful deployment and simulation execution.

4.2.3.19 Get Simulation Execution Results

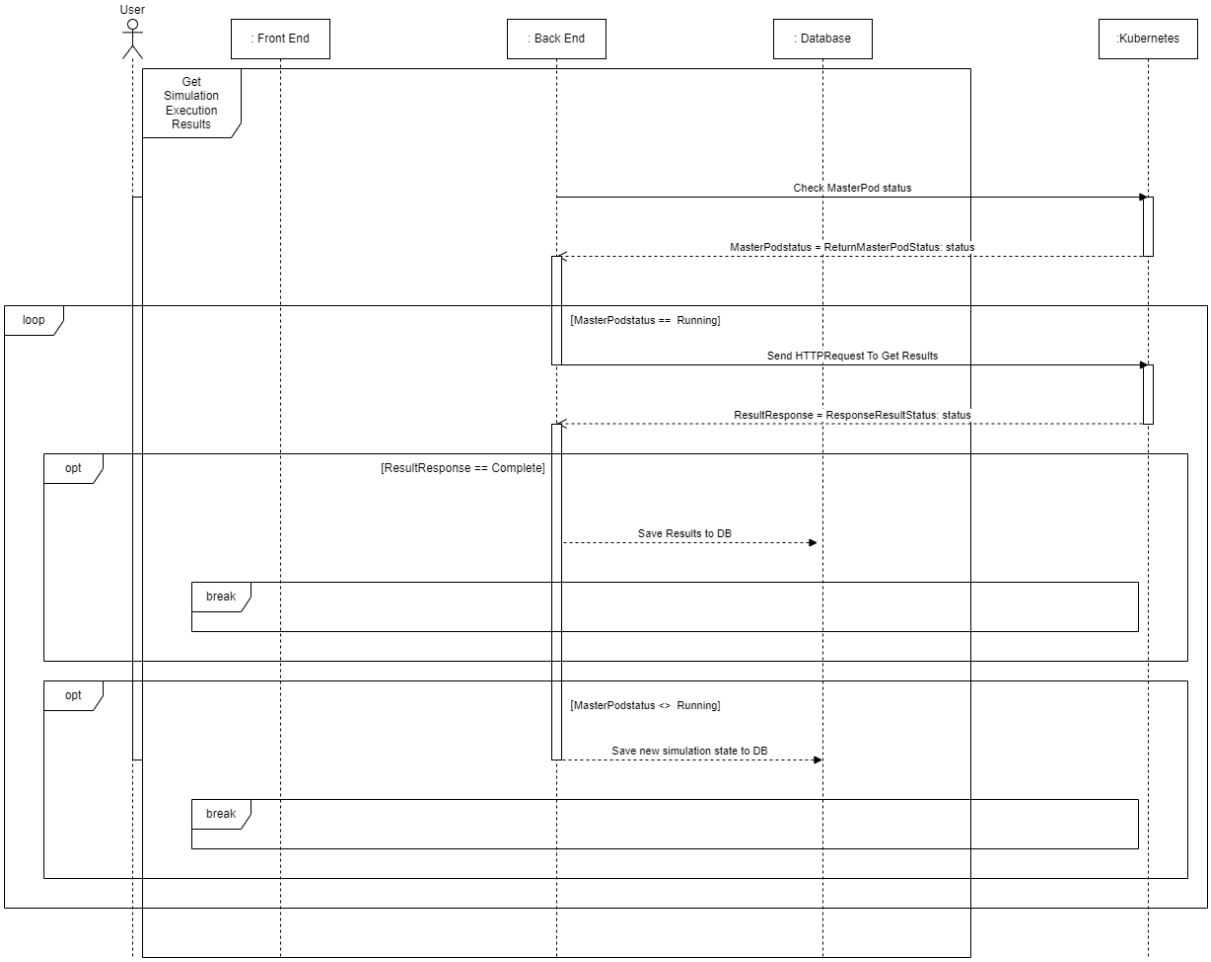


Figure 26. Get Simulation Execution Results

The diagram in Figure 26 illustrates the process of getting the simulation execution's results. The process takes place during the simulation run in a polling-based approach where the backend periodically checks the Master pod's status. While the Master pod is still running, the backend sends HTTP requests to get the results; upon receiving a complete response, it saves the results to the Database. When the Master pod stops running (in case of failure or after a successful receipt of the simulation execution results), the backend is no longer able to send HTTP requests. In this case, the simulation's state is changed accordingly at the database side.

4.2.3.20 Simulation Execution Search

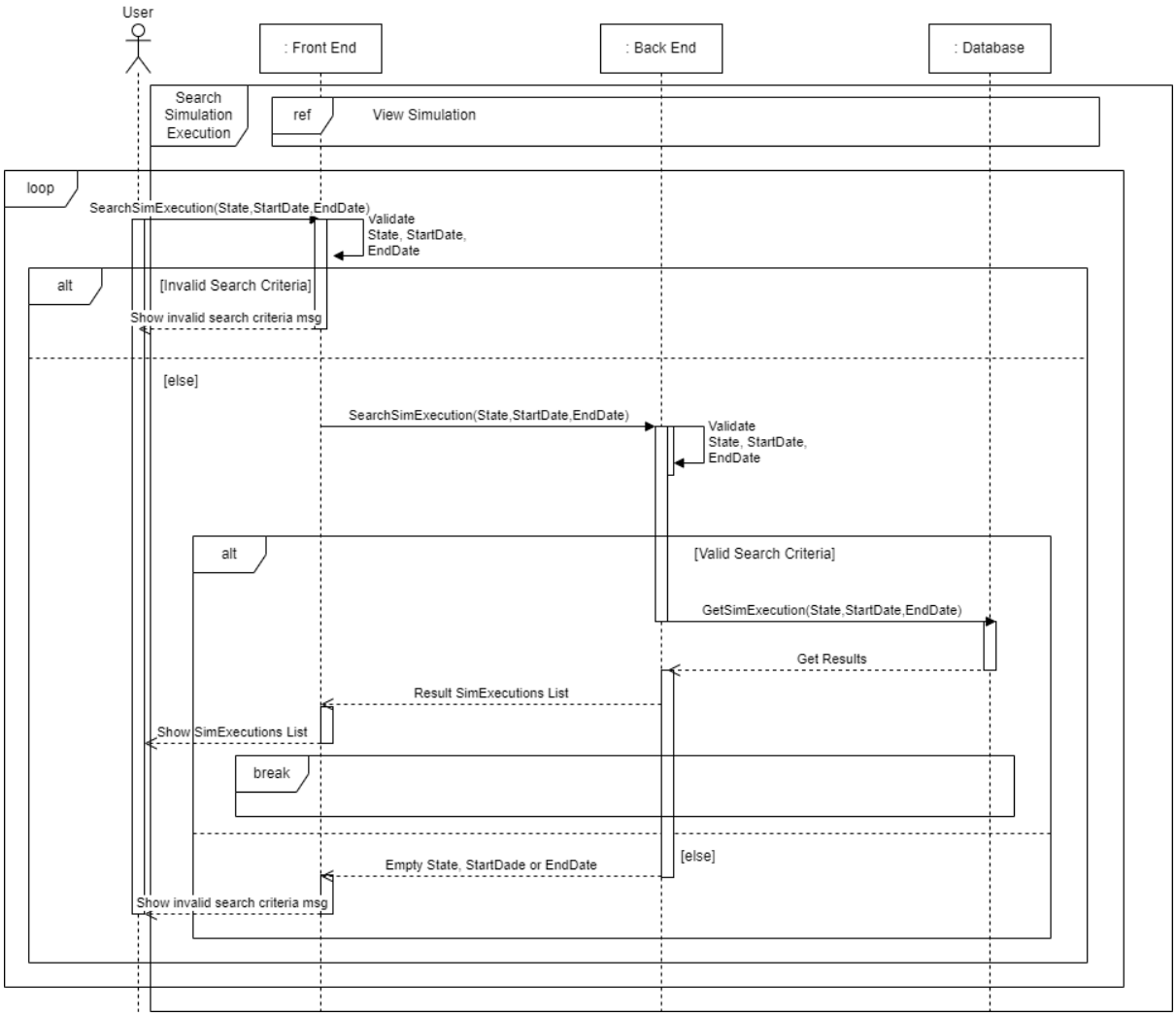


Figure 27. Simulation Execution Search

The diagram in Figure 27 illustrates the process of searching a simulation execution. In the simulation's screen, the user requests the simulation's executions that match specific criteria (the executions' state and optionally the start and end date). The front end checks the supplied criteria and if they are valid, the request is forwarded to the backend, which in turn validates the criteria once again and queries accordingly the database. Finally, the database returns the respective results, which are displayed to the user in the form of a list of matching executions. If the search criteria are not valid, an error message is shown to the user.

4.2.3.21 Simulation Execution Show

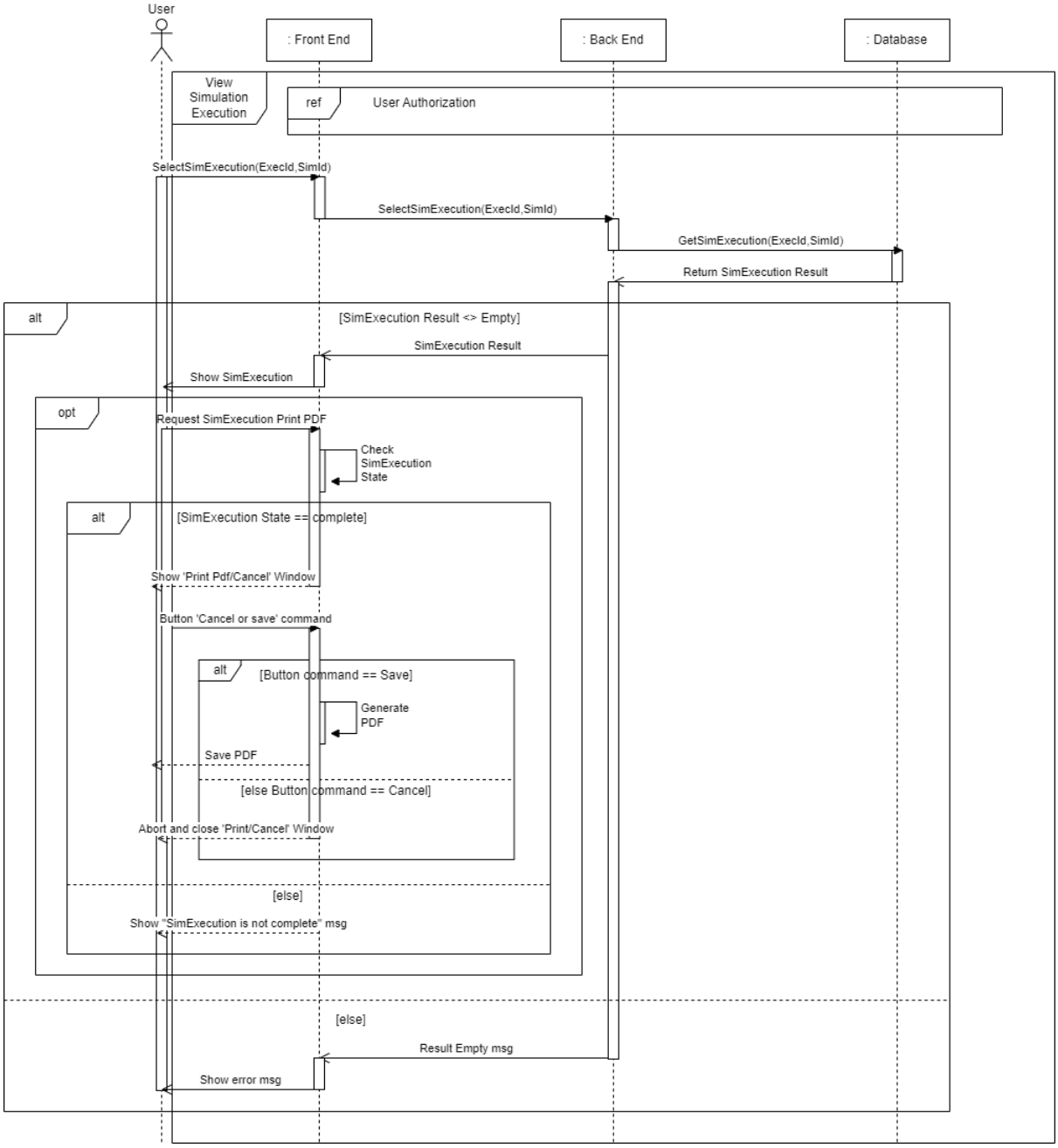


Figure 28. Simulation Execution Show

The diagram in Figure 28 describes the process of viewing the results of a simulation execution. The authorised user selects a specific simulation's execution and the front end forwards the request to the backend, which in turn queries accordingly the database. The database returns the simulation execution's data to the backend. If the result is not empty, the simulation execution's data is displayed to the user.

Optionally, the user may request to print the result in PDF format. The system first checks whether the simulation execution is complete. If it is, a dialog prompts the user to either save or cancel the PDF export. If saved, a PDF is generated and saved in the user's system.

If the execution is incomplete or the result is not found, the system displays an appropriate error message.

4.2.3.22 Simulation Execution Delete

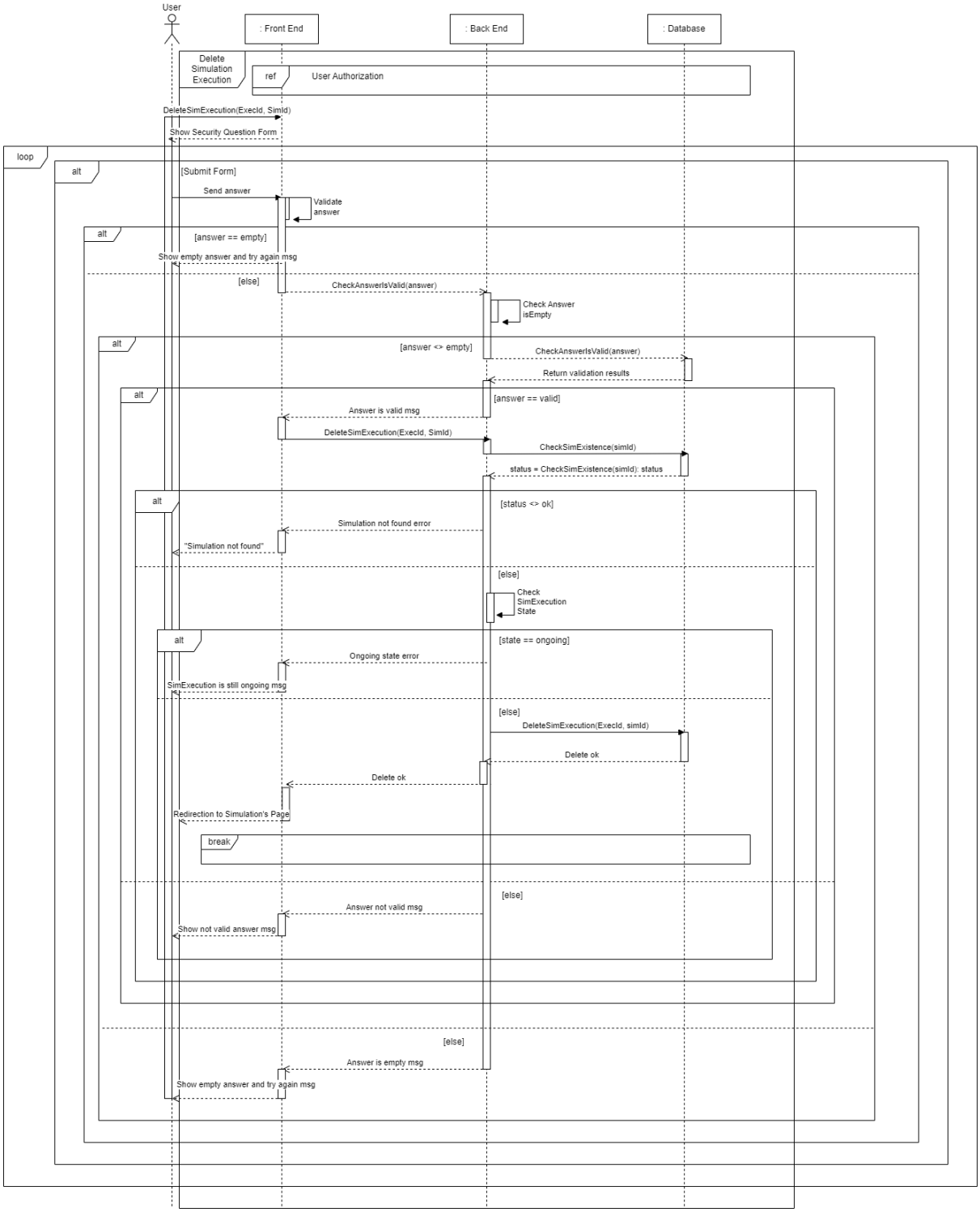


Figure 29. Simulation Execution Delete

The diagram in Figure 29 describes the process of deleting a specific simulation execution. The process begins with an authorised user's attempt to delete a simulation execution, followed by a prompt for the user to answer a security question. The frontend first validates whether the submitted answer is empty and, if not, it forwards the answer to the backend, which verifies it through the database. Once the answer is valid, the backend informs the frontend, which then proceeds to request the deletion of the simulation execution from the backend. Next, the backend verifies the existence of the simulation execution in the database. If the simulation is found, the system checks its current state. If the execution is ongoing, an error message is shown and the deletion is blocked. If the simulation is not running, it is deleted successfully and the user is redirected to the simulations page. In case the security question answer is missing or invalid, or the simulation cannot be found, appropriate error messages are displayed to the user.

4.2.4 Entity Relationship (ER) Diagram

The ER diagram presented below is created based on Chen notation style [41] and depicts the main entities of the system, their attributes as well as the relationships between them.

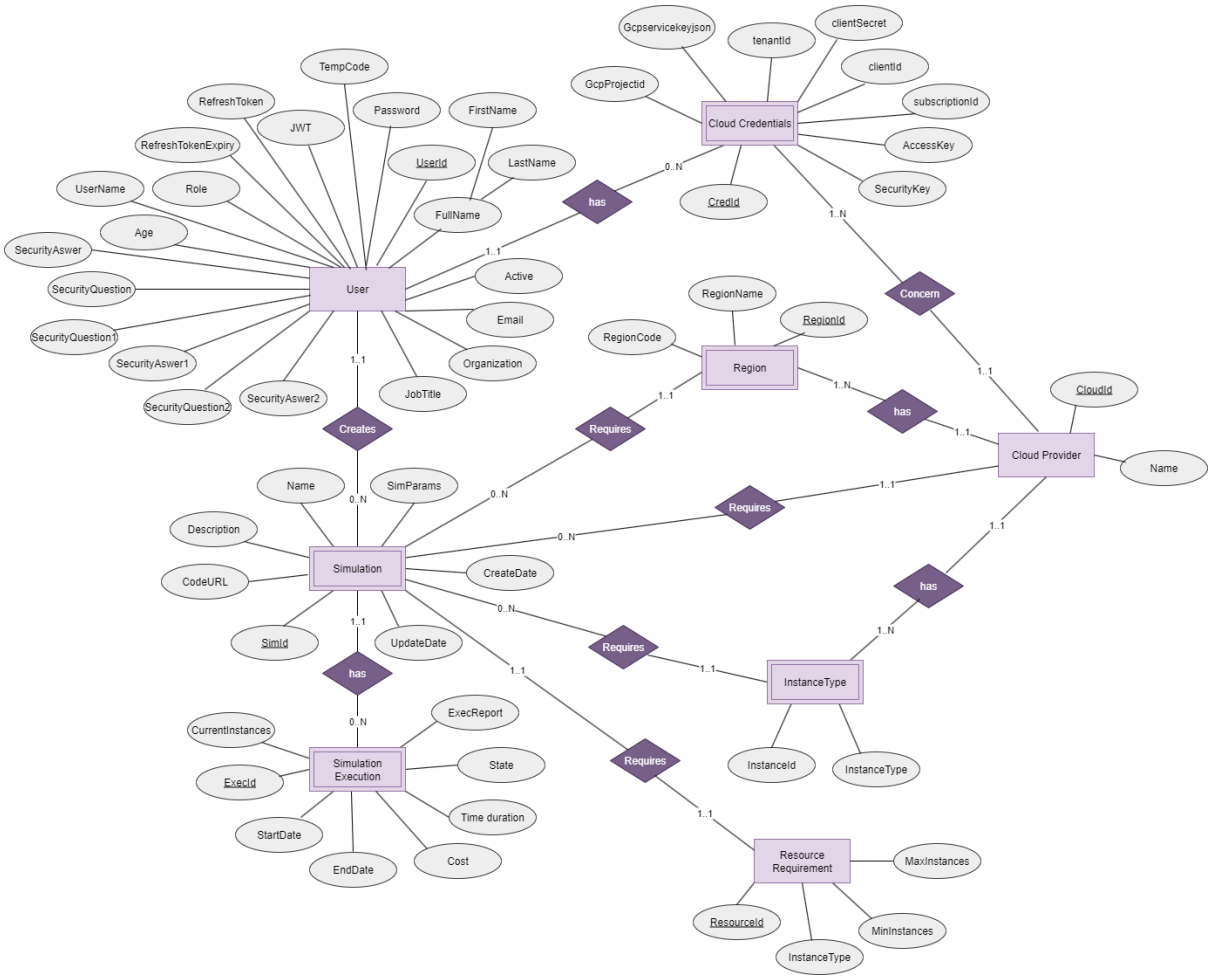


Figure 30. Entity Relationship (ER) Diagram

Two core entities depicted in ER are the 'User' entity, which includes attributes, such as username, email, password, tempcode, job title, and security details as well as the 'Simulation'

entity with attributes like name, parameters, description, and code URL. Their relationship is the act of creating a simulation as it is shown by the diamond with the label 'Create' which connects them. The cardinality of this relationship is 'one-to-many' which means that one user can create many simulations. Specifically, one user can be associated with zero or many simulations, while one simulation is associated with one and only user. The 'Simulation' entity is marked as a weak entity meaning that it existentially depends on the 'User' entity [41]. In other words, if the user that has created the respective simulation is deleted/destroyed, the same should hold for the simulation.

A similar relationship exists between 'Simulation' and 'Simulation execution' entities (with 'one-to-many' cardinality). In this respect, one simulation can have many executions or zero if it never runs. However, one execution maps to one and only simulation. The attributes of simulation execution are execution id, state, duration, cost, current instances, end date, start date and the report showing the simulation execution's results.

Each simulation requires specific resource requirements (see Resource Requirement entity), such as the (resource) instance type and minimum and maximum (resource) instances that can be created. The relationship between these two entities (Simulation and Resource Requirement) is 'one-to-one'. An extra table (InstanceType) was also added that contains the resource types per Cloud Provider and the relationship between this table and the Simulation table is 'one-to-many' as a Simulation can have only one instance type but an instance type can be used by many simulations.

A simulation also requires one cloud provider which will host the simulation. The cloud provider can be accessed through a user's authentication data (see Cloud Credentials entity). A user can have many credentials concerning different cloud providers, but specific credentials concern only one cloud provider. The cloud provider can also have many regions, but one region can belong to only one cloud provider.

4.3 Application Implementation

In this section, implementation details concerning the developed application/system, such as programming language, technologies & libraries used, the code structure and the most basic implementation classes, are specified.

4.3.1 Programming language, technologies & libraries

4.3.1.1 Backend

For the backend, the programming language used was C#. Further, the Asp.Net Core web API framework was used for the development of the backend's API in the .Net 9 platform.

The backend API (stands for Application Programming Interface) was designed by following the main principles of REST (stands for Representational State Transfer) architecture. In particular, the API uses a single URI to identify each resource, each HTTP method (GET, POST, PUT, DELETE) is used to denote a different management action over the identified resource and appropriate HTTP status codes are returned depending on the current case (e.g., 200 for successful processing and 404 if a resource was not found). Hypermedia controls or

HATEOAS (stands for Hypermedia As The Engine Of Application State) are included; this means that the responses include links to related resources, allowing the client to navigate the API dynamically without prior knowledge of all of its endpoints. The use of HATEOAS classifies the REST API as level 3 in the maturity model as it was first introduced by Leonard Richardson [42].

For the connection to the database, the Entity Framework was used, an object-relational mapper (ORM) that gives the developers the ability to work with data in the form of domain-specific objects rather than the tables of a relational database [43]. In particular, when such objects are manipulated by the application code, the framework guarantees that such manipulations are transformed into low-level SQL accesses over the respective tables of the underlying database. In order to decouple the internal data models and the API and to protect the sensitive parts of data from exposure to clients, DTOs (stands for Data Transfer Objects) are implemented, i.e., classes that are used to transfer data between layers of the application. The mapping between DTOs and domain entities was accomplished using the Mapster mapping library for .Net.

Another .Net library used in this project was FluentValidation, which uses lambda expressions for building strongly-typed validation rules in order to validate incoming data at the backend side [44].

The BCrypt.Net library was used for password hashing, ensuring user credentials are stored with industry-standard cryptographic protection.

For YAML (stands for YAML Ain't Markup Language) file manipulation (e.g., in the context of Kubernetes's application deployment architecture modelling), the YamlDotNet library was utilized, as it provides low-level support for parsing and emitting YAML data.

To interact with the Kubernetes API from the .Net environment, the KubernetesClient library was used.

4.3.1.2 Database

Data persistence is handled by Microsoft SQL Server (MSSQL), managed through SQL Server Management Studio since the type of database chosen is relational database in combination with the .Net Entity Framework. Although it is selected due to familiarity, MSSQL is a nice choice while working with Microsoft's frameworks and tools, like the .Net in our case; the main reason for that is that these tools are fully compatible with each other as they have been created by Microsoft. Another advantage of combining MSSQL with Asp.Net Core concerns the high system performance as the result of quick query execution [102]. Finally, another benefit that comes with the combination of these tools with the Entity Framework is the capability of database-first or code-first development, thus enhancing debugging and development.

4.3.1.3 Authentication/Authorization

Authentication and authorization have been implemented using JWTBearer and refresh tokens, where the latter provide a secure mechanism for session renewal. Both kinds of tokens are stored in HTTP-only cookies and in the database as well.

It is worth to mention that the HTTP-only cookies that store the access JWT and the refresh token have the following configuration: the 'partitioned' and 'secure' attributes are set, where the latter ensures that the cookie is only sent over HTTPS, the 'HttpOnly' attribute is also set, which ensures that the cookie is not accessible via JavaScript, and 'SameSite=None'; the latter is configured in order to avoid the cookies to be considered as third-party and thus be blocked by browsers in the future.

In general, the value of the 'SameSite' depends on two main cases. In the case when the frontend is co-hosted with the backend and both have the same origin, the 'SameSite' shall take the value 'Strict' so that the cookies cannot be sent with cross-site requests (requests that come from different sites from the original, having the same content, and might be malicious). The 'Strict' value is recommended in such cases so that the cookie can be sent by the browser in response to requests originating only from the cookie's origin site. In case when the frontend and the backend are of different origins, as in our development environment, the 'Strict' value will not authorise requests between the two origins. As a result, the 'SameSite' had to take the value 'None' such that the 'partitioned' attribute could undertake the protection from cross-origin attacks as it is described below. This is the recommended case as usually applications are distributed in different hosts.

The 'partitioned' attribute is the modern way of CSRF (stands for Cross-Origin Resource Sharing) protection. Typically, for CSRF protection, a non HTTP-only cookie (CSRF Cookie) containing a token (the CSRF Token) is created when a user logs in the system as well as during the backend process that refreshes the JWT and the refresh token, in order to be sent to the frontend along with the access and refresh tokens' cookies. The frontend, during every POST, PUT or DELETE request that it processes, extracts the CSRF token's value and includes it as an extra header in the HTTP Request that is sent back to the server. The server compares the CSRF token that was received through the request's header to the one in the CSRF cookie. If they are different, the request is blocked [67].

Partitioned cookies, also called CHIPS (stands for Cookies Having Independent Partitioned State), are stored in storage, partitioned by the top-level site. When a cookie is created, two keys are assigned to it, the host key, which contains the host or domain name of the site that set the cookie, and the partition key, which contains the top-level domain of the URL the browser was visiting when the request was made to the URL endpoint that set the cookie [68]. This means that if a user visits site A, which embeds content from site B, the partitioned cookie that site B sets on the user's device will have a key-pair containing information of sites A and B. If the user visits a third site C which also embeds site B, this instance of site B will no longer be able to access the cookie because the partition key does not match any more [68]. So, in this way, the CSRF protection was implemented without the need of a CSRF cookie.

CHIPS is Google's implementation in order to give the developers the ability to opt a cookie into partitioned storage, with separate cookie jars per top-level site. It has been implemented on Google's web browser Chrome since Chrome 114 release [79]. It is also supported by other browsers too, such as Firefox, Edge, Opera, Chrome Android, WebView Android [96].

Apart from the CSRF protection, either implemented through CSRF token, or through the use of the 'partitioned' attribute, it was also important to implement CORS (stands for Cross-Origin Resource Sharing) policy. CORS is a security mechanism that allows the server to bypass the 'same-origin' policy so that a web site can make AJAX calls/requests to a server of a different domain. This is important when the frontend and the backend are of different origins, like in our case [97]. So, a CORS policy has been set in our system that allows requests between the two different origins of our frontend and backend.

4.3.1.4 Frontend

At the frontend side, the application was built using HTML, CSS, and JavaScript, styled with Bootstrap 5 for responsive design and user interface consistency. Vue.js (Options API) is loaded via CDN. Fetch API was used for making asynchronous HTTP requests to the backend, facilitating dynamic user interactions, such as form submissions and data updates without full page reloads.

4.3.1.5 Cloud deployment

For cloud deployment, the system was containerized using Docker while the infrastructure can be orchestrated through Kubernetes, which manages containers, scaling and availability. These technologies can be used to deploy the system in the cloud (or in minikube for testing purposes) and also to run the simulation in a cloud provider (or in minikube for testing). For the containerization, Docker images of the backend, the frontend and the SQL server have been created and stored in DockerHub. Analytical installation instructions are provided in Chapter 5.

It must be highlighted that Terraform is employed for infrastructure-as-code (IaC) provisioning, automating the setup of the cloud resources needed for the simulation to run.

4.3.1.6 Testing

Minikube was used to test the system and its cloud deployment locally. Concerning the system's performance, the response time of requests has been tested with the use of Grafana k6 load testing tool.

4.3.2 Code Structure

In this section, the structure of the code is described and an analysis of its main classes is presented.

4.3.2.1 Backend

The backend is an Asp.Net Core web API. When such a project is created, the application's setup, like dependency Injection (DI), middleware, and security, can be defined inside the 'Program.cs' file. In general, Asp.Net Core projects configure and launch a host for the applications startup as well as its lifetime management. Further, while using the minimal hosting model (a simplified approach for building fast HTTP APIs with ASP.NET Core) the host configures a server and a request processing pipeline. In order to configure and build the host, the .Net application creates an instance of WebApplicationBuilder class with the command 'var builder = WebApplication.CreateBuilder(args);'. When the host is built, the application is created and runs [99], [100]. The 'Program.cs' file contains configuration regarding the use of JWT Bearer, the read of JWT from the HTTP-only cookie and validation of JWT through the database.

The backend is organized into seven main folders. To begin with the lowest level, the 'Data' folder contains the SimSaasContext.cs class (.cs stands for c sharp/C#) which inherits from the Entity Framework's instance 'DBContext'. DBContext is used to map classes generally called 'Entities' or 'Models' (the definition of 'Models' is used in the current work) to the actual database's tables and to query/manage the database through them [69]. The SimSaasContext.cs class introduces the 'DbSet' properties, which represent the tables in the database, and the 'OnModelCreating' method, which implements the 'UseCollation' method in order to apply the database's collation, defines the tables' primary keys with the 'HasKey' method, defines the relationships between the tables through the 'HasOne' and 'WithMany' methods and the foreign keys through the 'HasConstraintName' method.

The 'Models' folder follows that contains the classes, representing domain entities, that correspond to each table of the database. These classes are: 'Users.cs', 'Simulation.cs', 'Simexecutions.cs', 'Resourcerequirement.cs', 'Region.cs', 'Cloudprovider.cs' and 'Cloudcredential.cs'. Thus, these are the main classes that reflect and represent the main information entities of the system.

The communication between domain entities and the client is based on DTOs. DTOs are classes that work as intermediate objects so that the data can be transferred to and from the database without making changes on the actual table objects, protecting in this way the data from exposure to clients. They are internally mapped into domain entities, which are then manipulated by the system/backend such that the respective row/entries in the corresponding tables are inserted, updated or deleted (depending on the actual manipulation function/operation requested by the client).

There are several DTOs for each model, each one used for a specific use case / management operation/function on the model's objects. For example, 'UserDTO.cs' is used to get a list of users, the 'UpdateUserDTO.cs' is used to update the properties of a user, 'CreateUserDTO.cs' for creating a new user and so on. Each DTO contains only the necessary properties for the corresponding use case / management operation. DTOs are organized into seven subfolders

within the folder named 'DTO': 'UserDTOs', 'SimulationDTOs', 'SimExecutionDTOs', 'ResourceDTOs', InstanceTypeDTOs, 'RegionDTOs', 'ProviderDTOs', CredentialDTOs.

In the 'Services' folder there are classes that contain the main business logic of the application/backend. In particular, in these classes, procedures and functions used to manipulate the domain/model objects are introduced.

The 'UserService.cs' class contains functions for users' manipulation, like read, write, update, and create users. 'SimulationService.cs' and 'SimExecutionService.cs' contain business logic for manipulating simulations and simulations' executions, respectively.

The 'GMailService.cs' is responsible for the creation of the automated email, which sends the temporary password to the user who requests to reset his/her password.

The 'AthService.cs' is responsible for the user's login, logout, register and the creation/refresh of the access and refresh tokens. It also checks if a user is active or not. The JWT is validated with the use of the Microsoft.AspNetCore.Authentication.JwtBearer Nuget package and the implementation of the 'AddJwtBearer' method by configuring the signature claim (verifies the token's cryptographic signature), the issuer claim (identifies who issued the token), the audience claim (specifies who the token is intended for) and token's expiration claim (defines when the token expires) [98]. This configuration takes place into the 'Program.cs' file along with the configuration for the system to read the JWT from the HTTP-only cookie instead of the Authorization header. In particular, the authentication/authorisation is handled inside the 'Program.cs'. While configuring the host, the methods 'builder.Services.AddAuthentication' and the 'builder.Services.AddAuthorization' register the authentication and authorization services. After the configuration, the host is built, creating the application (app instance), and the 'app.UseAuthentication' and 'app.UseAuthorization' methods are implemented in order to add the authentication and authorization middleware to the HTTP pipeline.

There is also a service class for the HATEOAS links called 'LinkService.cs'. This class adds HATEOAS links to UserDTO and SimulationDTO when they are returned by the 'GET' request in UsersController and SimulationsController, respectively. For example, if an authorised user requests a list of users, the UsersController will return the users and the HATEOAS links for each one. The LinkService.cs class also returns the navigation links (links to register, login, logout, profile, simulations list and users list) according to a user's rights (if is logged in or not and his/her role).

Other service classes concern cloud management, such as the 'TerraformService,cs' class, which is responsible for the terraform application, the 'PollingService.cs', which checks the phase of the master pod in order to collect the results, and the 'SimulationRunService.cs', which implements helper classes and functions for the repository, yaml parsing and application, terraform application and resource cleanup.

The aforementioned services are called by the controllers found in the 'Controllers' folder. The controllers are at the highest level of the code's structure and responsible for the manipulation

of AJAX requests coming from the frontend. To this end, after an initial checking of the requests, they usually call the respective service (depending on what is actually requested) that exposes the right business logic and then they wrap the responses received in the form of DTOs and produce HTTP responses containing suitable messages and response status codes, such as like 200 OK for successful HTTP requests, 401 Unauthorized, 404 Not Found or 400 Bad Request, which are sent back to the client (frontend in our case).

There is also the 'Helper' folder, which contains helper classes like the 'Link.cs' class that is used by the 'LinkService.cs' class in order to construct the HATEOAS links. Other helper classes are utilised to clone Git repositories, generate temporary codes (for password reset reasons), create Terraform projects (for generating managed Kubernetes clusters), parse Yaml files (of the simulation code), generate configMaps (the configMap.yaml that is used to mount the simulation's parameters JSON in master pod's YAML) and apply RBAC (stands for Role-based access control) so that the master worker can have permission to create the slave workers.

Finally, the 'Validators' folder contains classes that validate input data in forms. Each validator class is related to the validation of instances/objects of a specific DTO class whose properties/fields are used and specified in forms. For example, the 'RegisterValidator.cs' class validates objects/instances of the 'RegisterForm.cs' DTO class whose properties are used in the User Registration form. In validator classes, rules are defined for each field/property. If at least one rule fails, then the DTO is invalid and the respective request that encompasses it does not reach the controller. In this case, a suitable response status and message is returned by the validator. Validator classes extend the AbstractValidator class of the FluentValidation C# library.

4.3.2.2 Frontend

The frontend's structure is much simpler than the backend. It takes the form of a single-page frontend that consists of just the index.html file; this means that everything is done via this page. The structure of index.html consists of the header (<head> tags) and the body (<body> tags). Inside the header there are <script> tags which import the Vue.js and the Vue Router CDN, the Bootstrap 5 CDN and the style.css file while the body contains <script> tags with the JavaScript files. The body also contains the navigation bar, which forms the page's navigation menu.

The next important file is the app.js file. In this file, specific routes have been configured to map URL paths to the components that need to be shown to the client. The Vue Router uses these routes and updates the URL paths in order to show the corresponding components to the user without the need of reloading the page from the server [70]. The app.js is the initialization of the Vue frontend app. It defines routes, components and methods and creates the Vue app instance. It also dynamically fetches navigation links from the backend. Specifically, it fetches a URL, i.e., the '/Navigation' URL, which returns HATEOAS Links from the 'NavigationController' depending on the user's rights where these links update the navigation bar. If the user is logged in, Links containing paths to the 'home', 'profile', 'simulations' and

'log out' components are returned. If the user has the 'Admin' role, he/she has access to the other users' information as well. If the user is not logged in, the 'home', 'login' and 'register' links are returned. In this way, the navigation bar is dynamically updated with the options that correspond to the users' rights.

The variables.js file has a constant variable, which contains the backend API's URL, which is called by all components through this constant variable. Specifically, the API's URL is the root endpoint that each component uses along with the relative paths in order to reach the right backend controller.

The other JavaScript files are the components that Vue Router shows to the user. The main components are the users.js, home.js, login.js, register.js, profile.js and simulations.js. Each component has an html template and methods for the encompassed functionality, which is to update the behavior of the component and to call all the backend controllers responsible for the component's functionality.

Apart from the components described above, there are also the securityquestions.js and the refreshtokens.js files. These are reusable components called by other components in special cases. The refreshtokens.js is called in every fetch when the response is 401 Unauthorized in order to refresh the JWT and resent the request. The securityquestions.js is used in some 'delete' use cases and also in the 'forgot password' use case as an extra level of security.

4.4 Requirements Satisfaction

In this section, the degree of satisfaction of each requirement of the system is presented in the table below along with a respective justification.

Table 2. Requirements Satisfaction

Requirement	**Requirement Type**	**Satisfaction (Yes/No/Partially)**	**Justification**
User registration	Functional	Yes	The requirement is fully satisfied as the application can support the registration of simple users.
User deletion	Functional	Yes	The requirement is fully satisfied as the application can support the user deletion.
User update	Functional	Yes	The requirement is fully satisfied as the application can support the user update.
Password reset/update	Functional	Yes	The requirement is fully satisfied as the application can support the password reset and update.
User login	Functional	Yes	The requirement is fully satisfied as the application identifies users through their

			username and password and gives authorization when needed.
User display	Functional	Yes	The requirement is fully satisfied as the application supports the display of user profiles.
User search	Functional	No	The requirement is not satisfied because there was not enough time to implement it.
Simulation create	Functional	Yes	The requirement is fully satisfied as the application supports the creation of a new simulation.
Simulation display	Functional	Yes	The requirement is fully satisfied as the application supports the display of a simulation's details.
Simulation update	Functional	Yes	The requirement is fully satisfied as the application supports the update of a simulation's fields.
Simulation delete	Functional	Yes	The requirement is fully satisfied as the application supports the deletion of simulations.
Simulation search	Functional	No	The requirement is not satisfied because there was not enough time to implement it.
Simulation execution display	Functional	Yes	The requirement is fully satisfied as the application supports the view of a simulation's execution details.
Simulation execution deletion	Functional	Yes	The requirement is fully satisfied as the application supports the deletion of a simulation's execution.
Simulation execution search	Functional	No	The requirement is not satisfied because there was not enough time to implement it.
API Documentation	Non-Functional	Partially	The API Documentation is generated and becomes available through Swagger and Scalar. It is partially satisfied though as the protected requests are blocked due to CSRF protection.
User-friendly & Responsive User Interface (UI)	Non-Functional	Yes	Use of Bootstrap 5 framework by including the CDN into the frontend code. Also, use of Vue.js for the creation of a single page frontend application and Vue Router that updates the URIs without reloading the page from the server leading to unified look-n-feel and improving navigability.

Level 3 REST	Non-Functional	Yes	The API uses a single URI for each operation, HTTP methods (GET, POST, PUT, DELETE) are used for resource interaction and return appropriate HTTP status codes, Hypermedia controls are included, which means that the responses include links to related resources allowing the client to navigate the API dynamically without prior knowledge of all endpoints.
Platform independence	Non-Functional	Yes	Docker images of the application are provided in order to be deployed on the Kubernetes environment (minikube or a real one).
Good system performance	Non-Functional	Partially	The response time tested in Grafana k6 is under 5s. The Requirement is partially fulfilled as not all possible scenarios have been tested.
Good system availability	Non-Functional	Partially	Only the login process was tested for 50 users in Postman.
Sufficient system reliability	Non-Functional	Yes	The application offers proper error/exception handling with self-explanatory messages and applies proper form validation on both the client and server side.

Good level of security	Non-Functional	Yes	● Authentication based on JWT and refresh tokens stored in HTTP-only cookies and in DB as well. HTTP-only cookies prevent the tokens to be accessed through JavaScript. ● RBAC: Role Based Access Control is used to distinguish Admins' actions from those of registered users'. ● Use of BCrypt.Net package to hash the passwords, cloud credentials and the answers to the security questions. ● TLS/SSL with the use of Certificates (test in development with self-signed certificate) is applied in communications between the application components and between the application and the client when needed ● CSRF: Use of 'partitioned' attribute in HTTP-only cookies that store the JWT and refresh tokens

5

System Installation and Demonstration

In this chapter, a description of how the system can be installed and run in the Kubernetes environment is provided. A demonstration of the system is also provided through images that show some specific use cases.

5.1 System Installation

The project's code can be found in GitHub in the following links:

- Frontend: https://github.com/eleni88/UI
- Backend: https://github.com/eleni88/SimAppRepo
- Master/slave simulation: https://github.com/eleni88/MasterSlaveSimulation

The respective Docker images can be found in DockerHub:

- Frontend: https://hub.docker.com/r/eleni88/simulation-front
- Backend: https://hub.docker.com/r/eleni88/simulation-app
- Master/slave simulation:
    - Master: https://hub.docker.com/r/eleni88/master-worker-sim
    - Slave: https://hub.docker.com/r/eleni88/service-worker-sim

The instructions provided for the system's installation on Kubernetes concern the installation on Kubernetes minikube. Minikube is a tool that runs a simulated Kubernetes environment locally. It runs a multi-node local Kubernetes cluster on one's personal computer making it a nice tool for experimentation, development and testing [27].

Once the application is deployed in a minikube cluster, it has to become accessible from outside the cluster. For this purpose, the Gateway API is used. The Gateway API is an official Kubernetes project that manages and configures how network traffic enters and routes within Kubernetes clusters [104]. It supports various protocols like HTTP, TLS, TCP, gRPC [105] and it consists of three types of components: The GatewayClass, which defines a group of Gateways with similar configuration, the Gateway, which handles the traffic within the cluster, and the HTTProute that maps the traffic from the Gateway to the backend services. The GatewayClass is managed by a controller that implements this class [106].

In order to deploy the system on minikube, docker images of the backend Asp.Net Core Web API, the frontend and the SQL Server had to be created. Those docker images, apart from the SQL Server image, can be found in DockerHub. The necessary YAML files for the deployment of the images are also provided and can be found in GitHub within the frontend's and backend's

folders. The SQL Server image cannot be provided due to the End-User Licensing Agreement (EULA) according to which publishing or distributing the software is not allowed. So, only the dockerfile can be provided in order to be built by the user who will accept his or her EULA while pulling the SQL Server image. The EULA can be found in [103].

The YAML files that concern the frontend's deployment in minikube are the 'simfrontend-deployment.yaml', the 'simfrontend-service.yaml' and the 'frontend-route.yaml' and can be found in the frontend's repository under the folder named 'yaml'.

YAML files that concern the backend exist in the backend's repository, also under the folder named 'yaml'. These files are the 'simbackend-deployment.yaml', the 'simbackend-service.yaml' and the 'backend-route.yaml'. This folder also contains the 'sim-gateway.yaml', which concerns both the frontend and the backend but will be applied once as it is the configuration for the Gateway.

The *-deployment.yaml creates a Deployment, which is a workload management element that runs the respective application component and manages its workload on the cluster [110], while the *-service.yaml creates a service, which is an object in the Kubernetes system that exposes an application component running in the cluster so that we can access it with relevant clients, such as browsers [111].

There are also YAMLfiles for the database containerization. These are the 'db-service.yaml' and the 'db-statefulset.yaml'. The reason why a statefulset is chosen over a deployment as a workload management API for the database is its ability to provide persistent storage in the pod so that if the container stops and restarts, the data stored in the pod are not lost. These YAMLs are located in the backend's GitHub repo inside the folder named 'Database'.

As the database had initially been created locally, it must be migrated into the container's SQL Server; for this purpose, the database's backup has to be copied in a folder inside the container and finally restored. So, in addition to the above files inside the 'Database' folder, the database's backup (.bak) file and Dockerfile can be found as well as two scripts for the database's backup restoration inside the container. The first script is called 'mssql-entrypoint.sh' and it launches the MSSQL server in the background, waits for it to be ready in order to make a connection using the database system administrator's (SA) password and finally, runs the second script, named 'restore-database.sh', which checks the database's backup file's existence inside the container and if the backup exists, it executes a SQL, restoration command. The creation of the mssql-entrypoint.sh and restore-database.sh scripts was based on the logic found in [129].

The Dockerfile contains Microsoft's SQL server 2022 image and it creates folders inside the container to copy the backup file, the mssql-entrypoint.sh and the restore-database.sh scripts in. Then, it calls the instruction 'Entrypoint' which is a command executed at the time the container starts and, in our case, specifies the script (the 'mssql-entrypoint.sh') to automatically run at that time [130].

Consequently, the Docker Desktop and minikube are prerequisites for the local installation of the application in a machine. Therefore, the Docker Desktop and minikube installation instructions are provided followed by instructions for the system's deployment in minikube.

5.1.1 Docker Desktop Installation

The instructions provided in this section concern the installation of Docker Desktop on Windows OS (Operating System). First, visit the docker.com site and under the 'Products' section, choose Docker Desktop. There is a button 'Download Docker Desktop' which enables the visitor to choose the version of Docker Desktop for the preferred OS (Windows OS in our case). When the Docker Desktop download is completed, run the Docker Desktop installer.exe file. A window with some configuration will appear asking to use WSL (stands for Windows Subsystem for Linux) 2 instead of Hyper-V. These are virtualization features used to create docker containers on Windows. Hyper-V is a feature of Windows OS, specifically of Windows 10 and 11 (Pro or Enterprise), which, while enabled, allows users to create and run VMs on the same Windows system and Docker can run on a VM and create containers [124], [125]. WSL is a feature of Windows OS which is used to create a Linux environment on the Windows System without the need of creating an entire VM [123]. According to Docker's documentation, there is no difference in Docker Desktop's functionality between those two virtualization choices, so one can make the choice that better fits to his/her architecture [122]. If the WSL 2 is chosen in the configuration window, it will be automatically installed during the Docker Desktop's installation, but in order to work successfully, the Windows Subsystem for Linux feature on Windows must be turned on. If Hyper-V is chosen, the Hyper-V and Containers Windows features must be turned on. Also, the hardware virtualization in BIOS needs to be enabled [122]. Once the Docker Desktop is installed, a username and password need to be provided while the app is running in order to gain the ability to create repositories as being logged in to docker is a prerequisite in order to push docker images to DockerHub.

5.1.2 Minikube Installation

After the installation of docker, the next step is to install minikube. Minikube can be installed on a Windows OS in three ways. First, by downloading the .exe file containing the latest release and add the minikube.exe binary to the system's PATH. The second way is to install minikube through Windows Package Manager (Winget) which is a Microsoft's tool that automates the procedures of discovering, installing, configuring, removing developer tools with the use of command line [126]. The last way to install minikube is with the use of Chocolatey Package Manager. Like Winget, the Chocolatey is also a Package Manager for Windows to automate software installation and management with the use of command line [127]. The three ways of minikube installing are described in [88].

5.1.3 System's Deployment in minikube

The procedure of the project's deployment in minikube is described below. The following steps refer to the deployment in a manual way, but the procedure can also take place automatically as it is described at the end of this section.

1. Clone the code from GitHub to local system. In order to do so, the following commands need to run in Git Bash from the location where the cloned projects will be saved:

    - **git clone https://github.com/eleni88/SimAppRepo**
    - **git clone https://github.com/eleni88/UI**

More information about GitHub repositories cloning can be found in [131].

2. Run the Docker Desktop.

3. In cmd with administrator access run the command: **minikube start --driver=docker**, in order to start the minikube with docker driver and create a cluster.

4. The third step is to install a Gateway Controller. As Kubernetes currently is not natively supporting the Gateway API's resources, the CRDs (stands for Custom Resource Definitions) for this API have to be deployed [106]. There are many projects that act as Gateway Controllers supporting the implementation of the Gateway API. In our project, the NGINX Gateway Fabric has been chosen, which uses NGINX as the data plane in order to configure an HTTP/TCP load balancer, reverse-proxy, or API gateway for applications running on Kubernetes [107]. The commands are found in [101] and have to be applied in the sequence as presented below:

    - The first command installs the Gateway API resources:

      kubectl kustomize "https://github.com/nginx/nginx-gateway-fabric/config/crd/gateway-api/standard?ref=v2.0.2" | kubectl apply -f -

    - The second command deploys the NGINX Gateway Fabric CRDs:

      kubectl apply --server-side -f https://raw.githubusercontent.com/nginx/nginx-gateway-fabric/v2.0.2/deploy/crds.yaml

    - Finally, the third command deploys the NGINX Gateway Fabric (The Controller):

      kubectl apply -f https://raw.githubusercontent.com/nginx/nginx-gateway-fabric/v2.0.2/deploy/default/deploy.yaml

The successful installation of the controller is shown in Figure 31 where NGINX is ready and running.

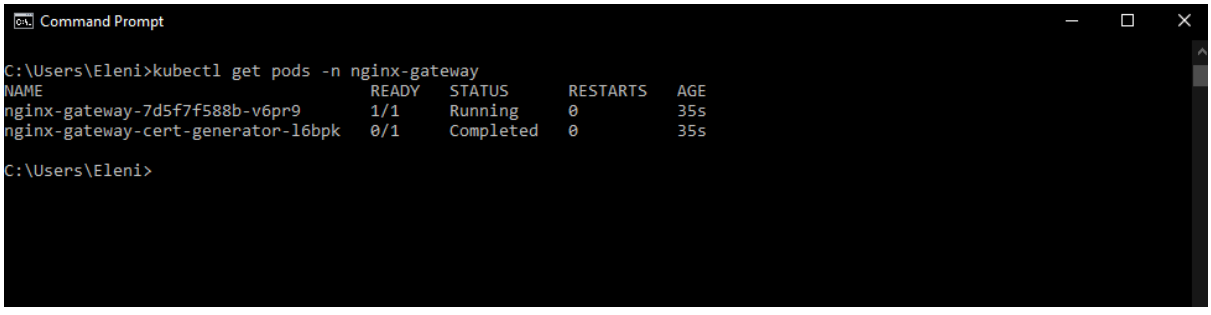


Figure 31. installation of Gateway controller

5. After the Gateway API's installation, the database and application need to be deployed in the cluster starting from the database. First, a secret to store the SA PASSWORD (the system administrator's (SA) password for logging in to SQL Server) has to be

created. This secret will be passed to the db-statefulset.yaml through an environment variable (env) in order to be consumed by the container [108].

The secret can be created with the following command by replacing the 'PassWord' with the SQL Servers admin's password:

kubectl create secret generic db-secret --from-literal=SA_PASSWORD="PassWord"

6. A secret with the database's connection string has also to be created that will be passed to the simbackend-deployment.yaml:

   kubectl create secret generic dbconnection-secret -n default --from-literal=CONNECTION_STRING="Server=mssql .default.svc.cluster.local,1433;Database=SIM_SAAS;User Id=sa;Password=PassWord;TrustServerCertificate=True;"

   Similar to the previous step, the 'PassWord' has to be replaced with the SQL Servers admin's password.

7. Once the secret is created, the database image needs to be built and then loaded to minikube environment. First, the **docker build -t simulation-db:latest .** command has to be run in cmd from the dockerfile's location. The next command is: **minikube image load simulation-db:latest** which loads the image in the container.

8. The next step is to deploy the db-service.yaml and db-statefulset.yam into the minikube cluster by running the following commands. The successful deployment is shown in Figure 32:

   - **kubectl apply -f path/db-service.yaml**
   - **kubectl apply -f path/db-statefulset.yaml**

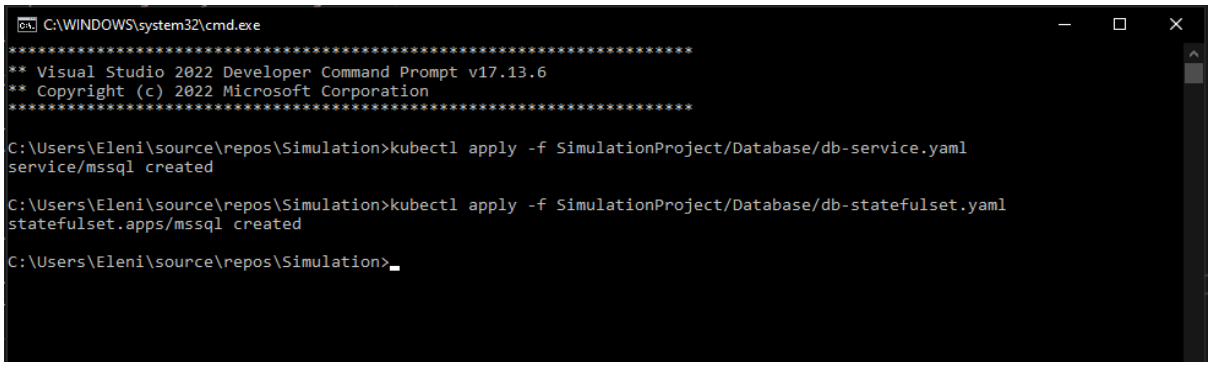


Figure 32. Database service and statefulaset applied

The 'path' in the above commands is the path where the bd-service.yaml and db-statefulset.yaml exist.

In Figure 33 we can see that the pod created is ready and running meaning that the deployment was successful. We can also get details about the mssql service and the

PVC (PersistentVolumeClaim), which is a request for storage in order to persist the data. We can also see the service's port (1433/TCP), which can be later used in a port-forward command so that the pod can be accessible from outside the cluster.

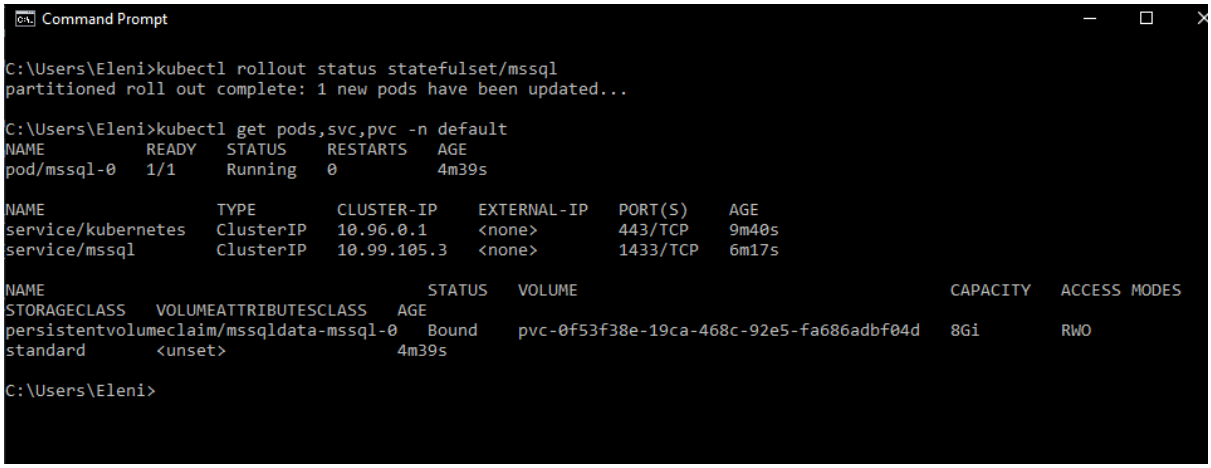


Figure 33. MSSQL runs in minikube

9. In Figure 34 we can see that the database exists in the minikube container, which means that the restoration was successful.

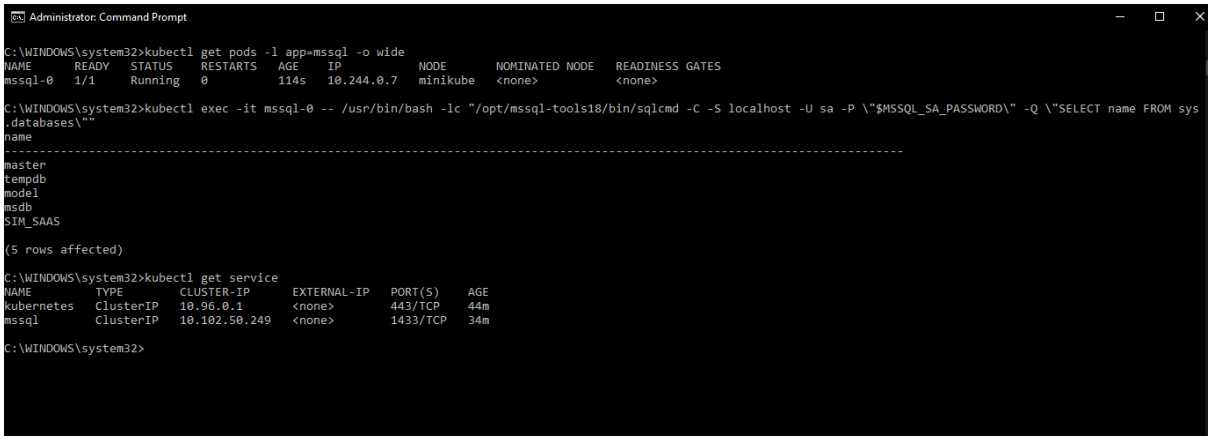


Figure 34. Database restoration success

The db-service.yaml exposes the database on the network. Once it is applied, the kubectl port-forward command can be used to forward a local port to the database's pod [109].

To find the pod's id, execute the command: **kubectl get service**.

Then, run the port-forward command: **kubectl port-forward -n default svc/mssql 1433:1433**. This is helpful if the user wants control over the database and its data through a UI. For example, a connection can be established with a tool like the

Microsoft SQL Management Studio in the address 127.0.0.1,1433 and the SA PASSWORD (Figure 35). When no longer needed, the port-forward can be stopped by pressing the keys 'Ctrl + C'.

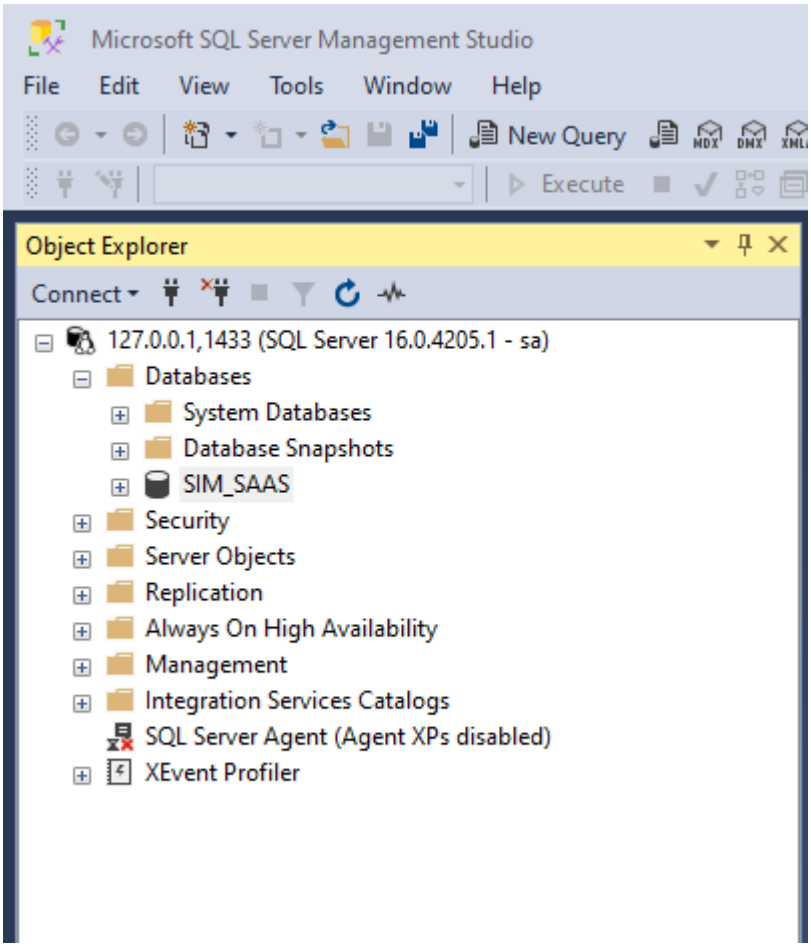


Figure 35. Connection to SQL Server

10. After the successful deployment of the database in the cluster, proceed to deploy the application's backend and frontend. First, the simbackend-deployment.yaml and the simbackend-service.yaml and then, the simfrontend-deployment.yaml and the simfrontend-service.yam need to be applied by running the following commands in terminal:

    - **kubectl apply -f path/simbackend-deployment.yaml**
    - **kubectl apply -f path/simbackend-service.yaml**
    - **kubectl apply -f path/simfrontend-deployment.yaml**
    - **kubectl apply -f path/simfrontend-service.yam**

11. After the application of the deployments and services, the TLS secret must be created. For the development purposes, a self-signed certificate has been created so that the HTTP-only cookies containing the JWT and refresh token can be sent to the browser

without being blocked. The self-signed certificate is created with the use of mkcert, a tool for creating locally-trusted development certificates without any configuration. The mkcert creates a certificate and a key in .pem format, valid for 127.0.0.1 and 'mysimapp.dev' (the application's domain) [112]; these files need to be added to a TLS secret, which is defined in the sim-gateway.yaml in order to support the HTTPS protocol [113]. The self-signed certificate can be created with the instructions provided in [112]. The TLS secret can be created with the following command [114]:

kubectl create secret tls simtls-cert --cert=path/tocert.pem --key=path/tokey.pem

The 'path/tocert.pem' and 'path/tokey.pem' are the certificate and private key returned by mkcert.

12. When the pods of the application are up and running and the TLS secret exists in the cluster, the Gateway API resources can be deployed. So, the next step is to apply the sim-gateway.yaml, the backend-route.yaml and the frontend-route.yaml:

    - **kubectl apply -f path/sim-gateway.yaml**
    - **kubectl apply -f path/backend-route.yaml**
    - **kubectl apply -f path/frontend-route.yaml**

    The results of the above commands are depicted in Figures 36 and 37.

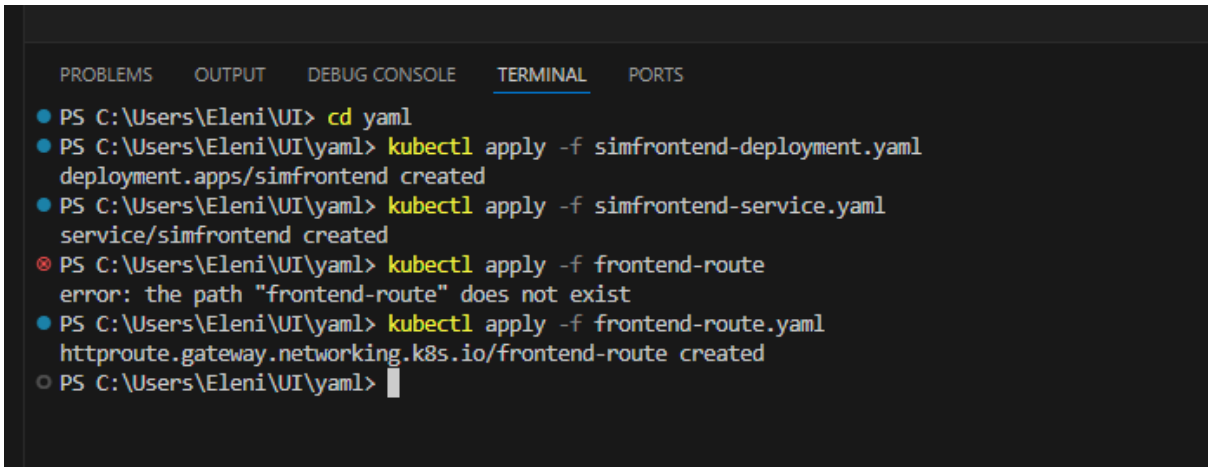


Figure 36. Application of simfrontend-deployment.yaml, simfrontend-service.yaml, frontend-route.yaml

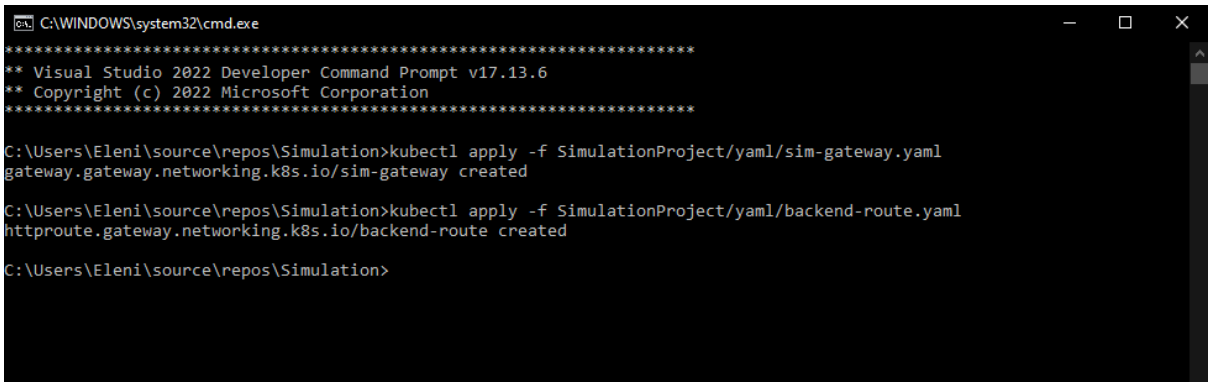


Figure 37. Application of sim-gateway.yaml and backend-route.yaml

13. The final step is to access the application from the browser. The command '**minikube tunnel**' will be used in a separate command (cmd) window so that the browser can access the Gateway's port in Minikube. After that, the application will be available in 'https://mysimapp.dev' as shown in Figure 38.

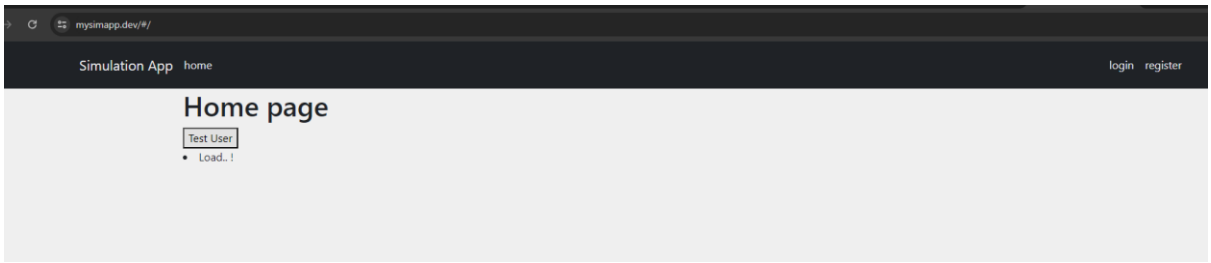


Figure 38. Application deployed

For the user's convenience, two batch scripts are provided that run in Windows cmd with Administrator's priviledges. The first script called 'minikube-init.bat' executes the minikube start and the Gateway API-relational commands. The second script called 'app-deploy.bat' builds the database's image and loads it to the minikube cluster while it also applies all the yaml files needed for the application's deployment. Specifically, the 'minikube-init.bat' script contains the commands mentioned in steps 3 and 4. The 'app-deploy.bat' script contains the commands described in steps 7, 8, 10 and 12. The provided scripts are general scripts and, as the user needs to create his/her own self-signed certificate and secrets, the corresponding commands could not be added in the first place. So, the user needs to run them manually or can add the commands for the secrets' creation in the scripts later. The command for the creation of the self-signed certificate needs to run once, so there is no need to be added in the script. The 'minikube tunnel' command is not contained in the script, but it can be added by the user, as long as the cmd window where the script was applied from is kept opened and so as the tunnel.

The scripts can run with the following commands from the directory they exist in:

.\minikube-init.bat

.\app-deploy.bat "backendpath\SimulationProject" "frontendpath\UI"

The 'backendpath' refers to the absolute path to the SimulationProject (the backend project) and the 'frontendpath' refers to the absolute path to the UI (the frontend project). The command passes these paths as variables inside the script's 'apply' commands so that the script can be independent from the project folders' paths. Before running the second script, the user needs to run the commands (in a second cmd window) to create the secrets needed according to the instructions previously provided. The two scripts can be found in the 'Database' folder of the backend's project inside a folder called 'init_deploy_scripts'.

5.2 System Demonstration

A demonstration of the system is presented in this section, describing a set of use case scenarios followed by respective screenshots.

5.2.1 User Registration

The first Use case that is described is the registration process. A new user is able to register in the system by providing his/her firstname, lastname, username, password, age, job and the organization he/she works in. It is also recommended to provide answers to the three out of five security questions as they are used in some delete cases as an extra layer of authentication/security.

The registration form implements input field validations ensuring that the inserted data are correct and follow the system's rules. For example, as we can see in Figure 39a, an error message is shown if the password does not apply the regular expression rules or a required field is empty or the security question is not unique, as shown in Figure 39b, indicating that the insertion fails.

In Figure 31c, the successful registration is depicted showing the corresponding message.

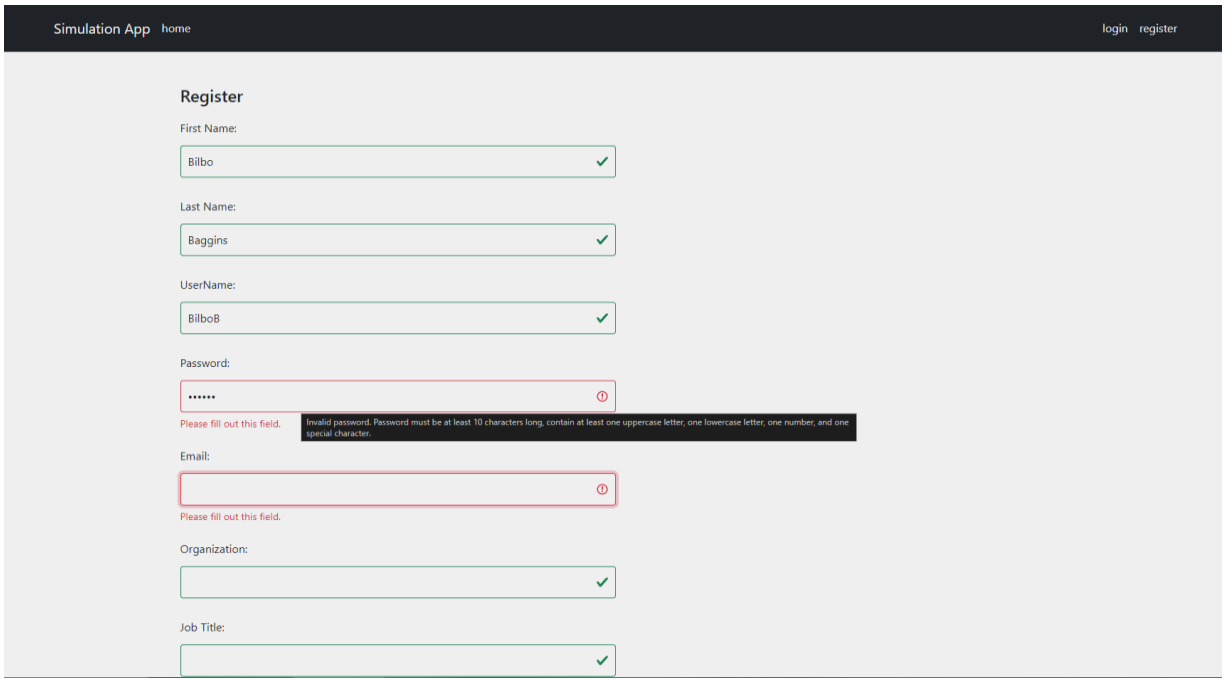


Figure 39a. The register form with error messages indicating invalid passwords and empty values for obligatory fields

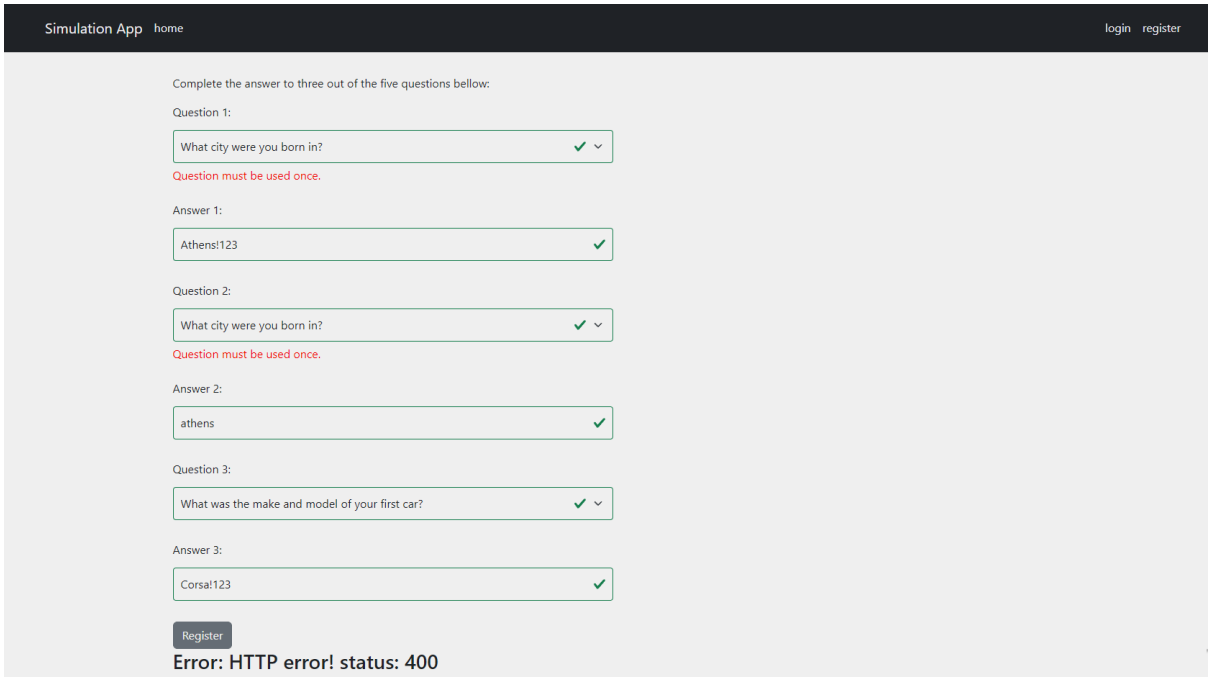


Figure 39b. The register form with error messages related to double answering of the same security question

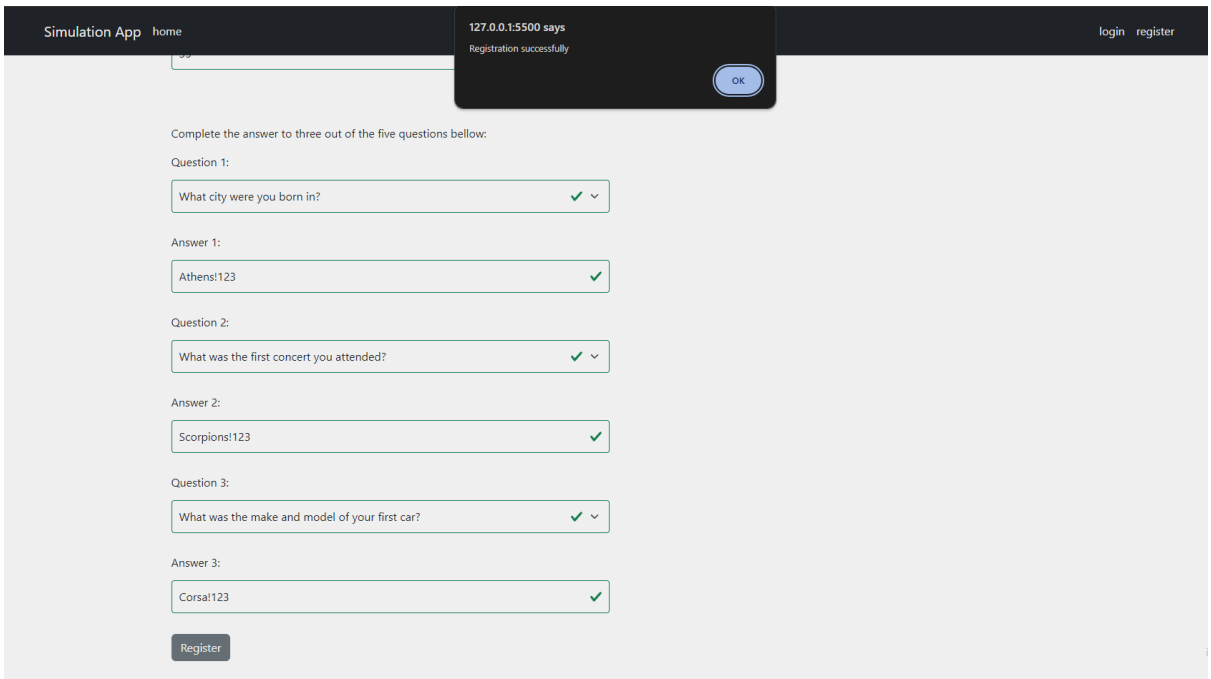


Figure 39c. The register form with all fields properly completed and a success message showing accomplished user registration

5.2.2 Login

The login form, shown in Figure 40a, requires the supply of the user credentials in the form of a username and password. It checks the validity of these fields and in case of validation failure,

a message ‘Invalid username or password’ is returned as shown in Figure 40c. For security reasons, the message does not reveal which of the two fields is incorrect.

In Figure 40b, the case of a successful login is presented. In that case, the navigation bar changes dynamically. The difference in navigation bar’s option before and after the login is shown in Figure 41 (before login) and Figure 42 (after login).

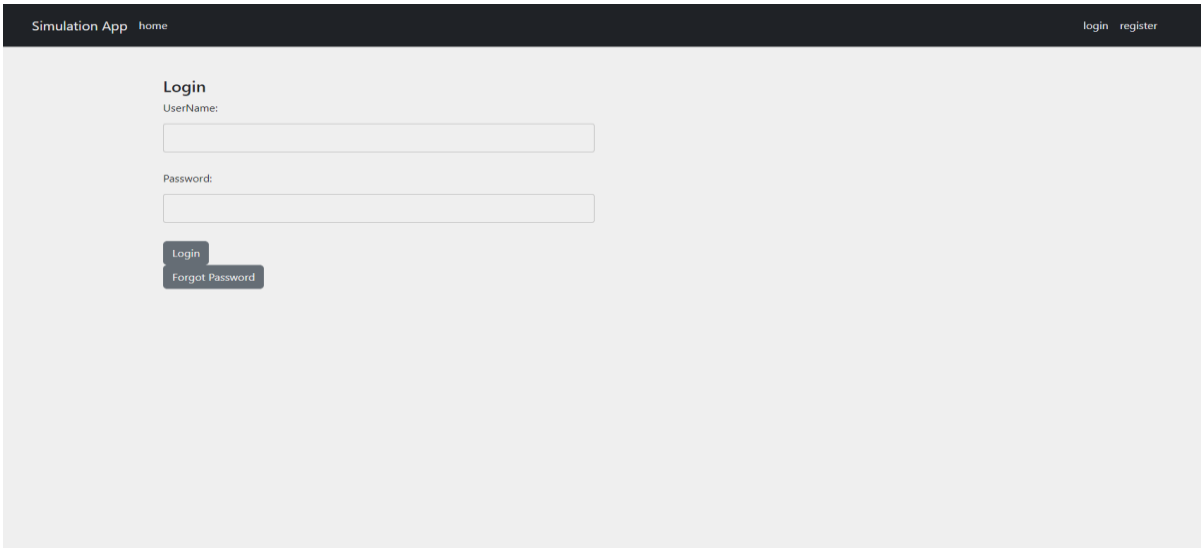


Image 40a. The login form as initially shown

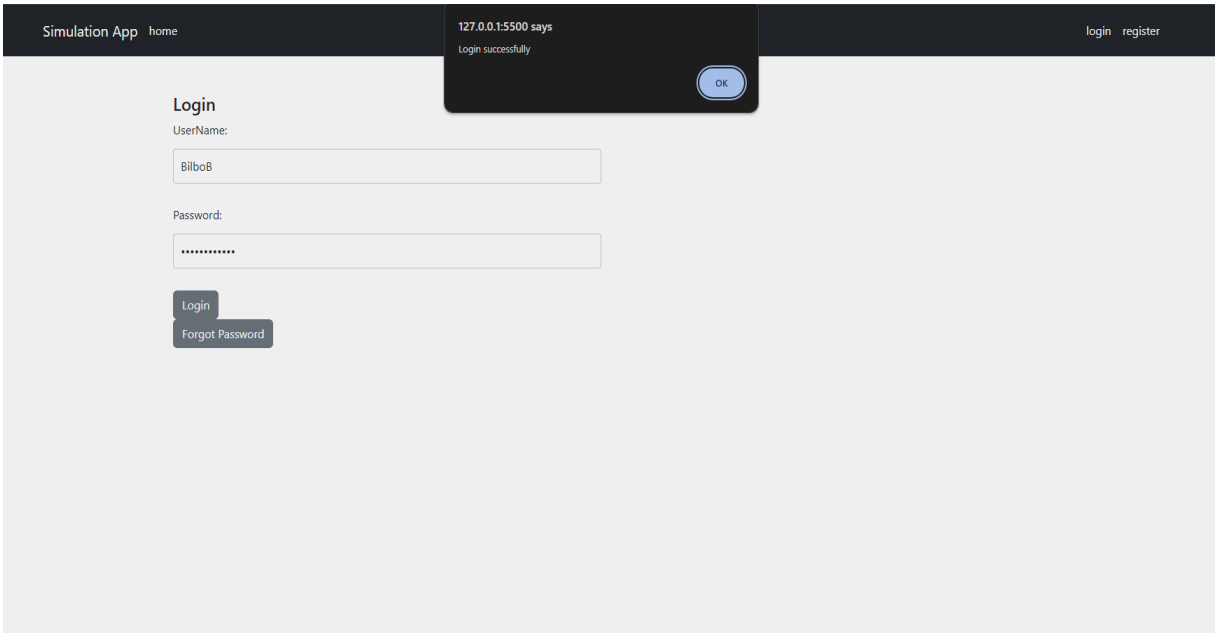


Image 40b. Successful user login.

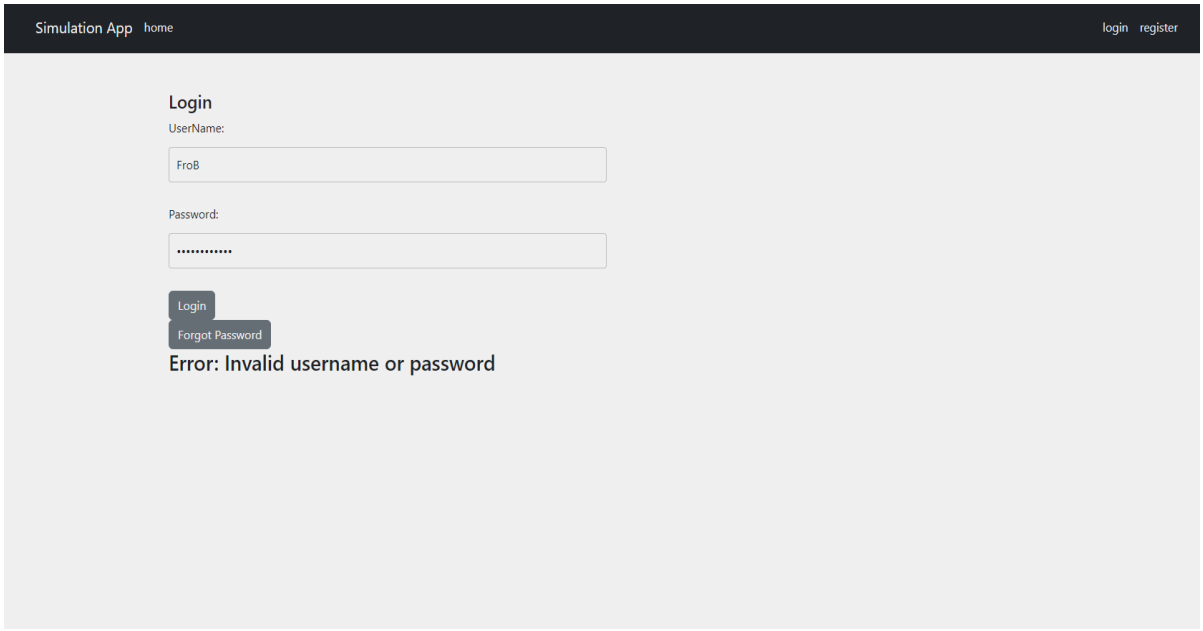


Image 40c. The login form completed with invalid user credentials.

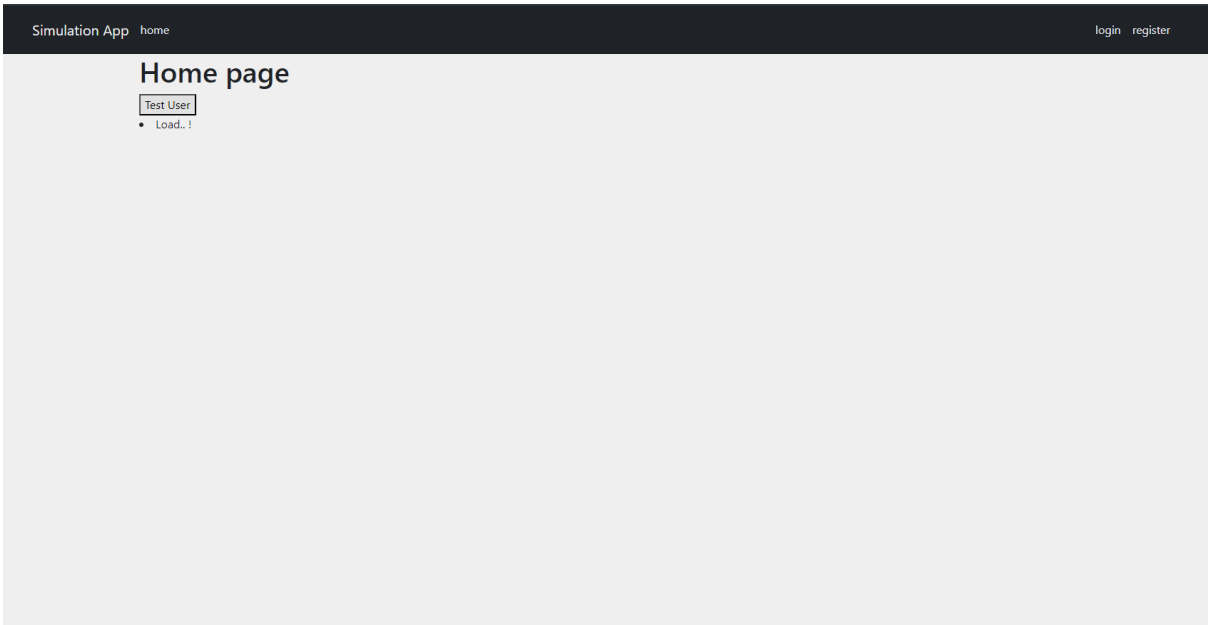


Figure 41. Home page before the login

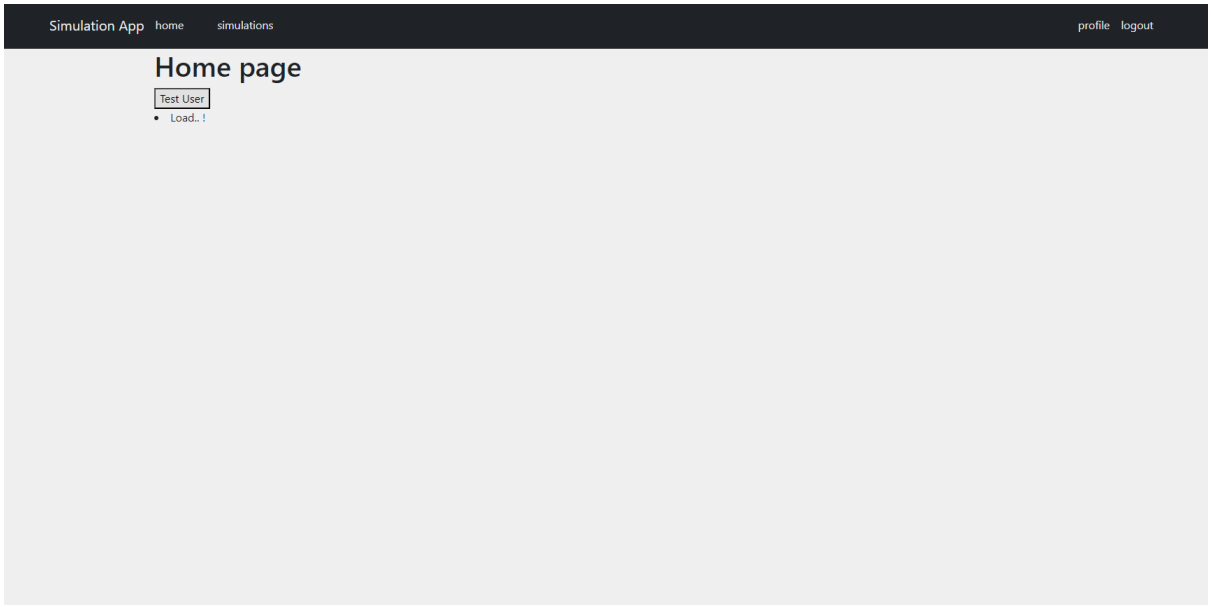


Figure 42. Home page after the login

5.2.3 Show a User's Own Profile

In the user's profile in Figure 43, details such as first and last name, username, email, age, organization and job title are depicted for the user along with the information of whether the user is active and admin. In addition, functionality regarding the profile, password and security questions update is provided through buttons as well as the ability to delete the profile, if the user is not an admin. Finally, there is a button enabling the user to add his/her credentials for a specific cloud.

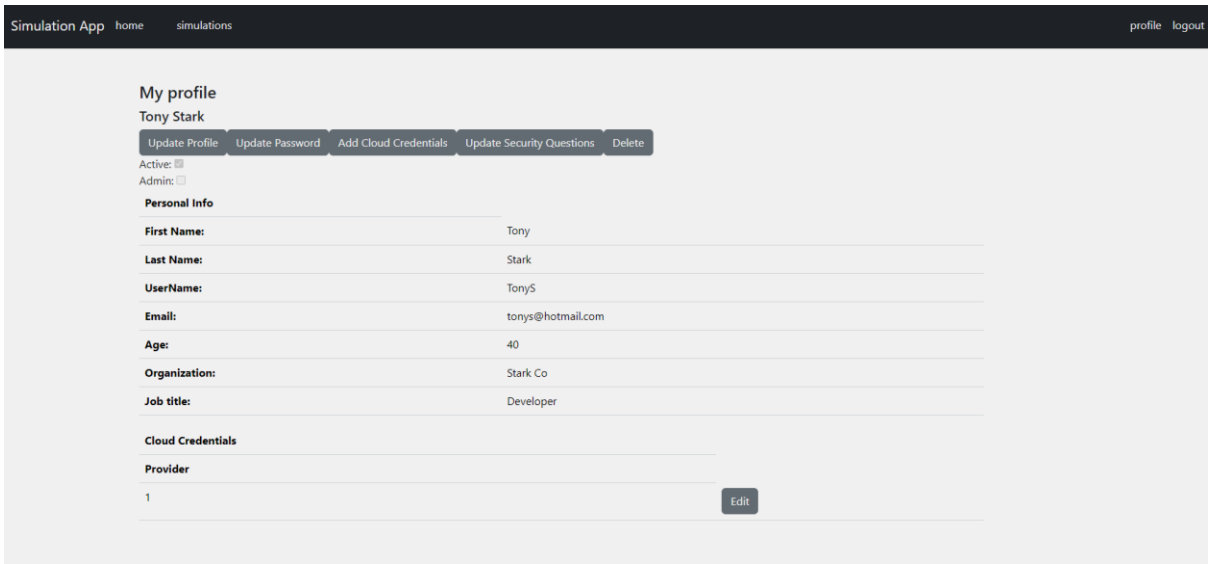


Figure 43. A user's Profile

5.2.4 Update Profile

As it was stated previously, a registered user has the ability to update his/her own profile. Specifically, through the 'update profile' button found in the user's profile screen, the form with the fields that can be updated is shown via a modal form (Figure 44).

Figure 44. Update of a user's profile

5.2.5 Update Password

The button 'Update Password' in the user profile's screen opens a modal form through which the password can be updated. In this form, the old password must be provided along with the new password, which has to be inserted twice ('NewPassword' and 'ConfirmPassword' fields) in order to be verified (Figure 45a). If the 'NewPassword' and 'ConfirmPassword' do not much or one of them is empty, an error message is shown and the form cannot be submitted (see Figures 45a and 45b). If the old password is valid and the new password meets the requirements and is verified, the update succeeds and a message of success is presented (Figure 45c).

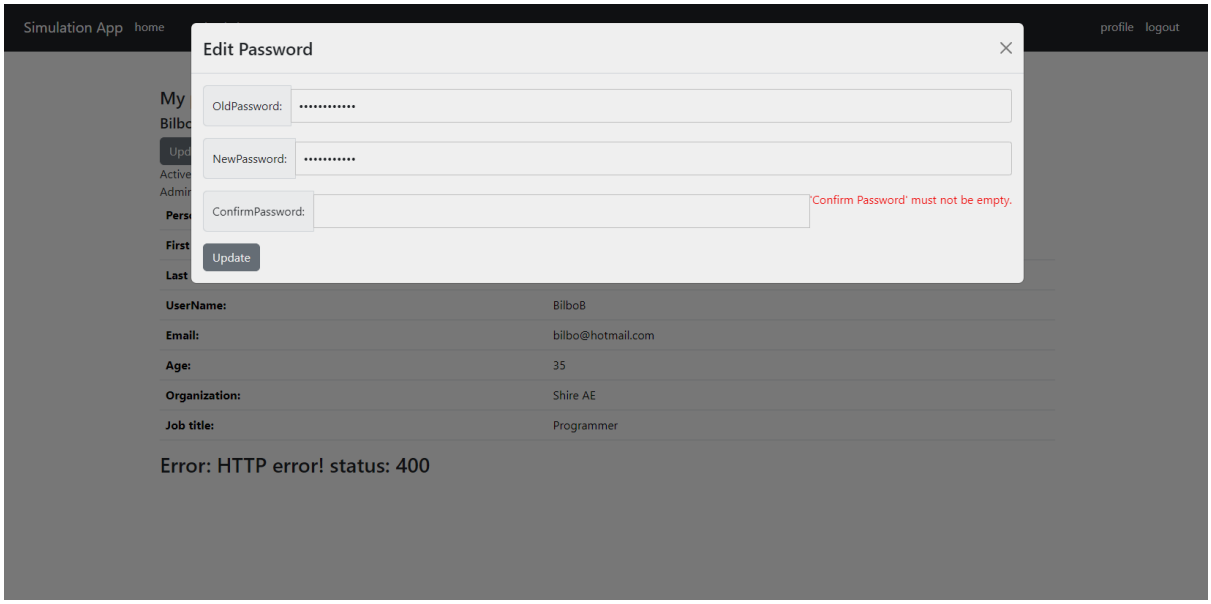


Figure 45a. Password update with empty new password

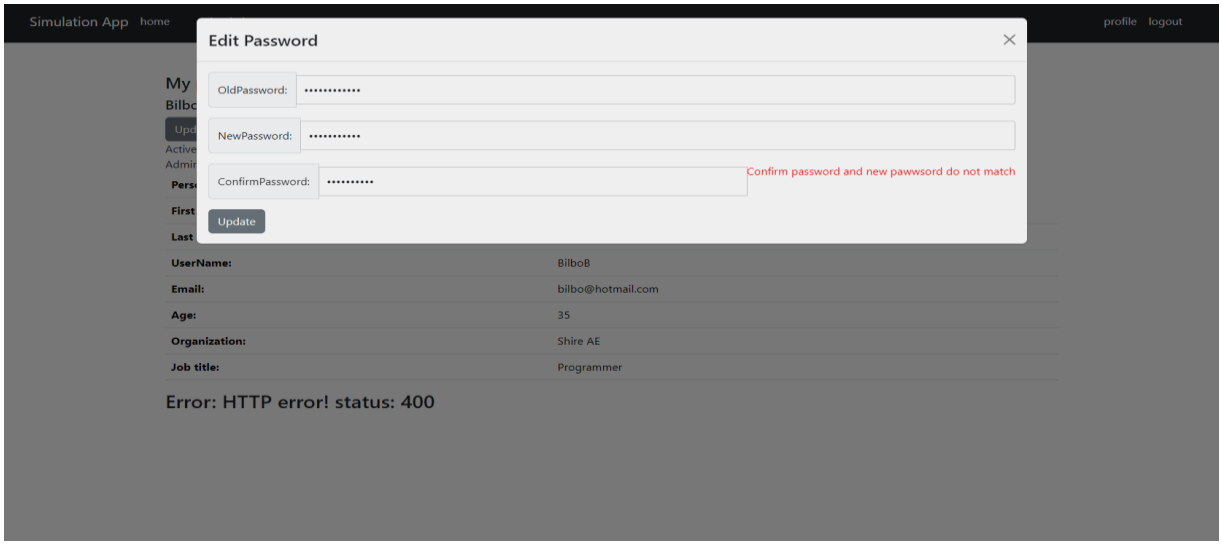


Figure 45b. Password update with new password entries not matching

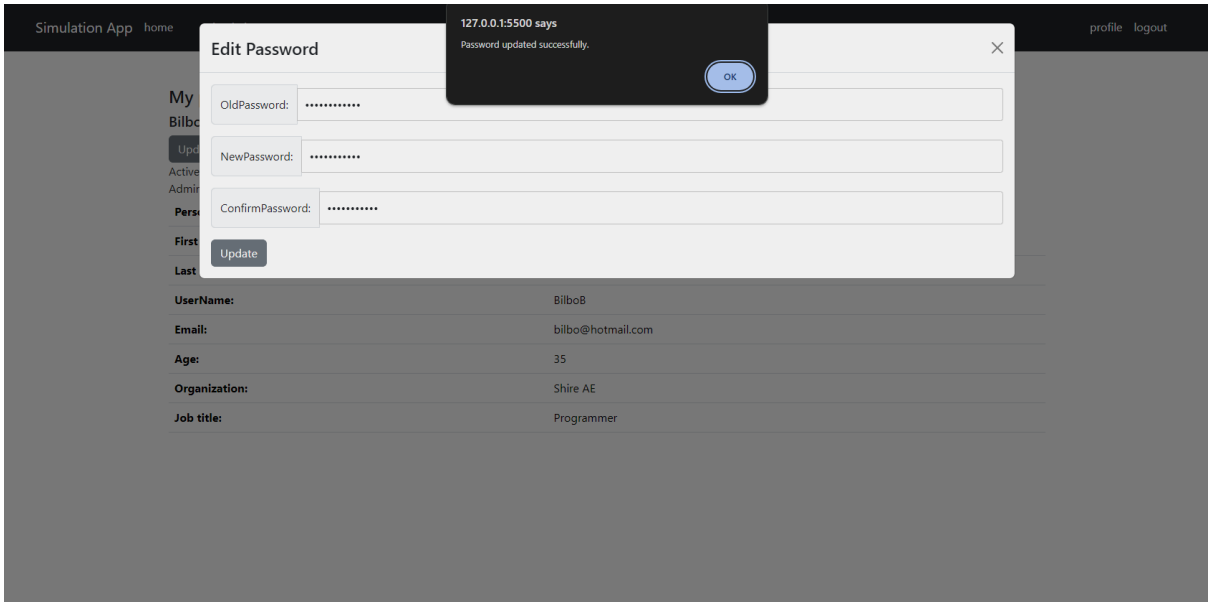


Figure 45c. Successful password update

5.2.6 Password Reset

Apart from password update, the system also supports password reset. In contrast to the password update process where the user needs to be logged in and provide the old password for authentication reasons, the reset password gives the ability to change the password in case it has been forgotten (so the user is not authenticated). The process is initialized by the 'Forgot Password' button in the login form. Upon pressing this button, a modal which prompts the user to provide the answer to a security question opens as an extra level of security in case the username has been compromised (Figure 46a). If the answer provided is valid, then a modal form opens where the 'Username', 'TmpPassword', 'NewPassword' and 'ConfirmPassword' must be provided (Figure 46b) along with a message that a temporary code is sent to the given email address (Figure 46c). If the security question validation or the password update fails, a message of 'Unauthorized' access is returned and the request fails (Figure 46d); otherwise, a message of success is shown.

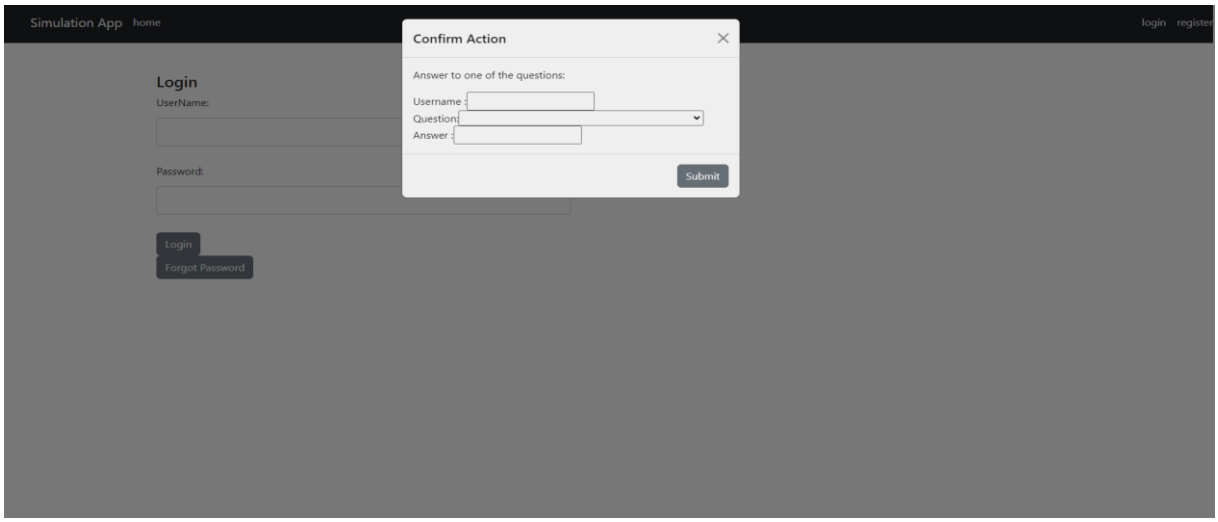


Figure 46a. Password Reset – confirmation via answer to a security question

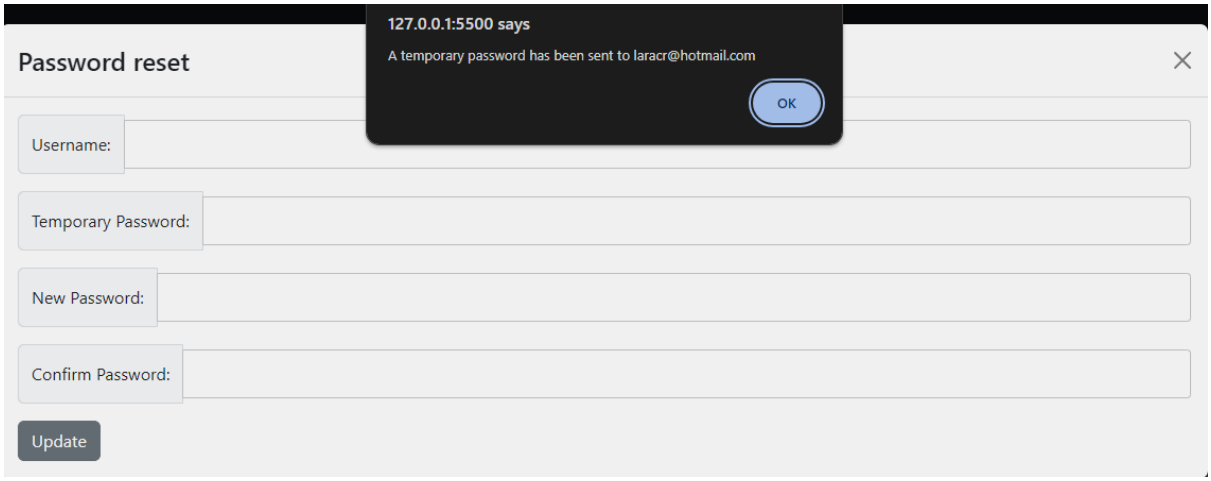


Figure 46b. Password Reset – temporary password sent to user email

Figure 46c. Password Reset – new password submission

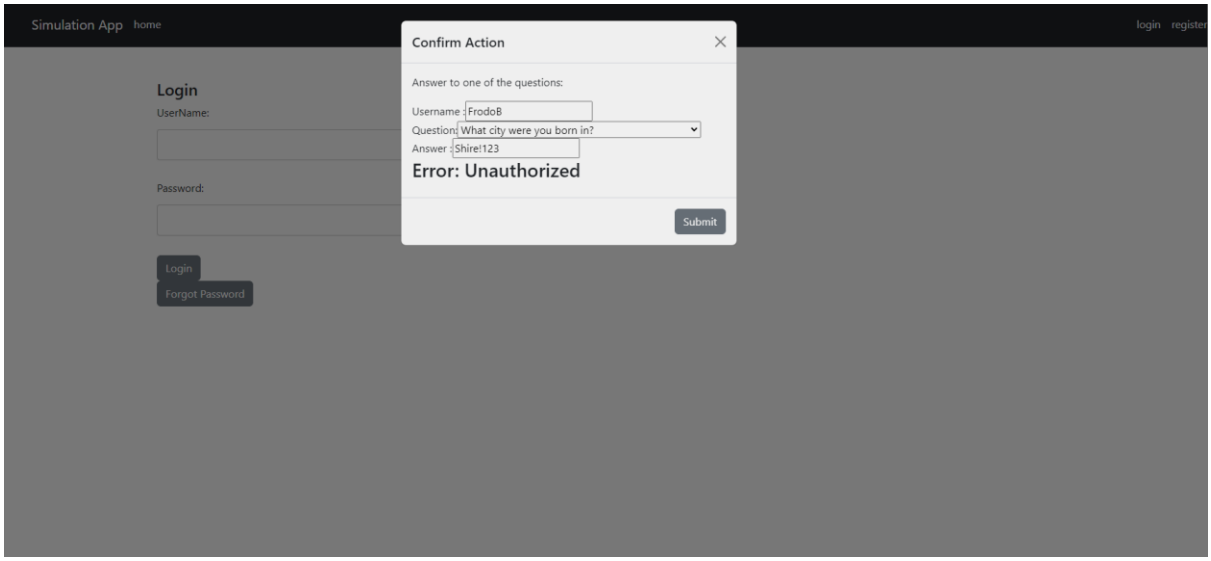


Figure 46d. Password Reset – failure of security question answering

5.2.7 Simulations List

In order to view the list of simulations, the user, after successfully logging in the system, presses the 'Simulations' option from the top menu. In Figure 47, the user simulations' list is depicted along with functionality of managing each listed simulation, such as create, view, edit, delete, execute and execute on Minikube, each of them mapping to a corresponding button. The 'Execute' button refers to the simulation execution on a specific Cloud Provider while the 'Execute on Minikube' button refers to the local execution on the Minikube environment.

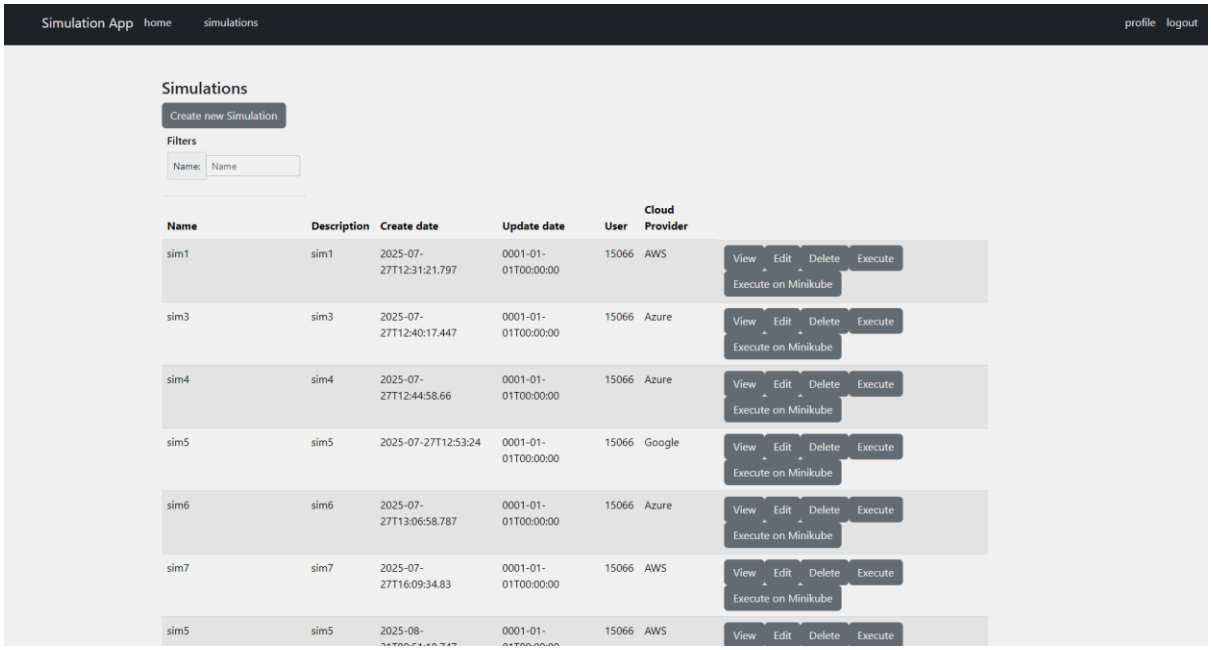


Figure 47. Simulation List

5.2.8 Simulation View

From the simulation's list screen, the user can choose a simulation and press the 'View' button in order to view its details. In the simulation's view, details, such as the simulation's name, description, creation date, update date are shown as well as the user who created the simulation, the cloud provider and region where the simulation can run, the parameters of the simulation in JSON format, the GitHub URL where the simulation's code exists, the type of resource to be used and the range of instances in which the resources can scale. In addition, a list of the simulation's executions is placed below the simulation's details as we can see in Figure 48.

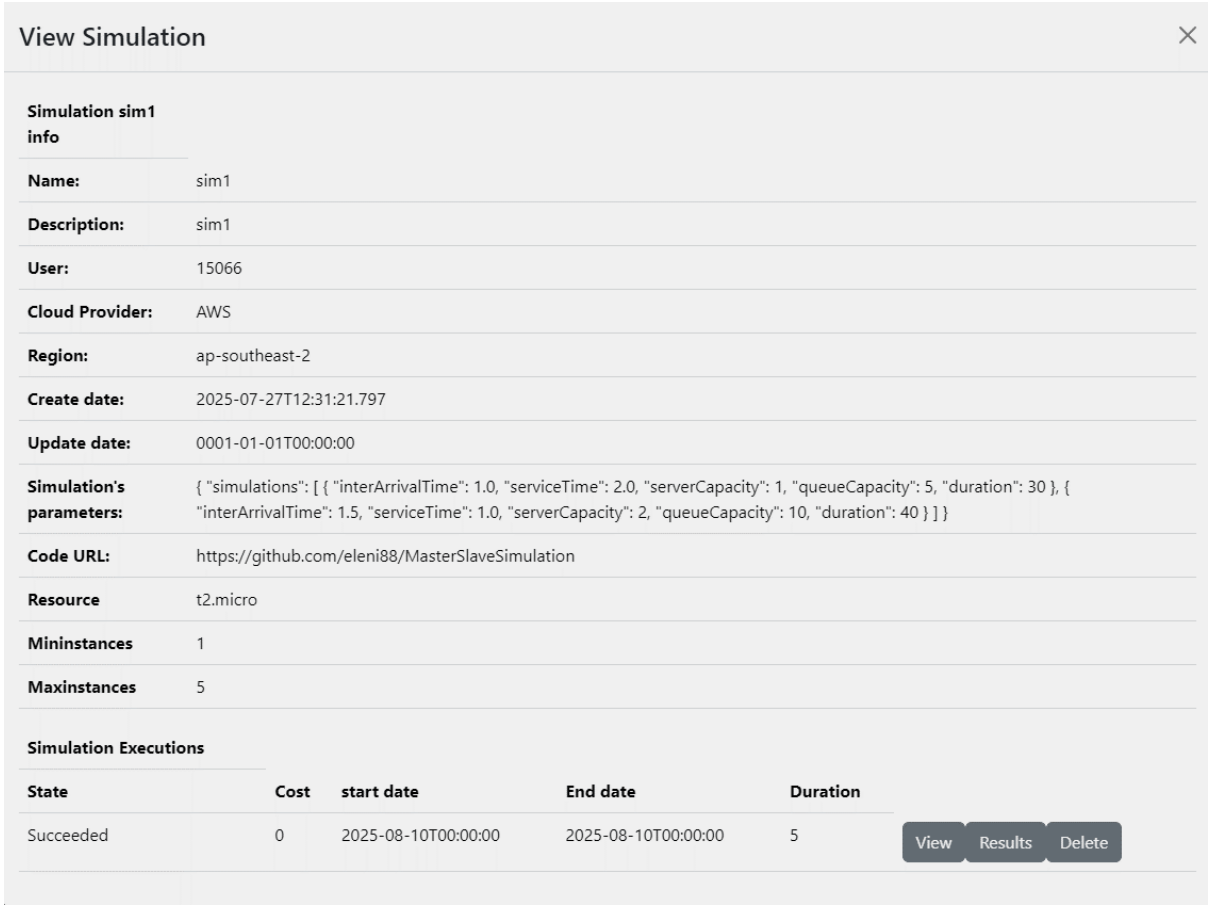


Figure 48. Simulation View

5.2.9 Simulation's Execution

In each execution of a simulation, three buttons are available. The 'View' button shows the execution's details (Figure 49), the 'Results' button shows the simulation's results (Figure 50), which can also be downloaded in PDF format by pressing the 'Export to PDF button'. When the 'Export to PDF' button is pressed, the PDF file is automatically downloaded in the user's system. Figure 51 depicts the PDF file after it is stored locally. Finally, the 'Delete' button (Figures 52 and 53) deletes the specific execution after the successful submission of the answer to a security question.

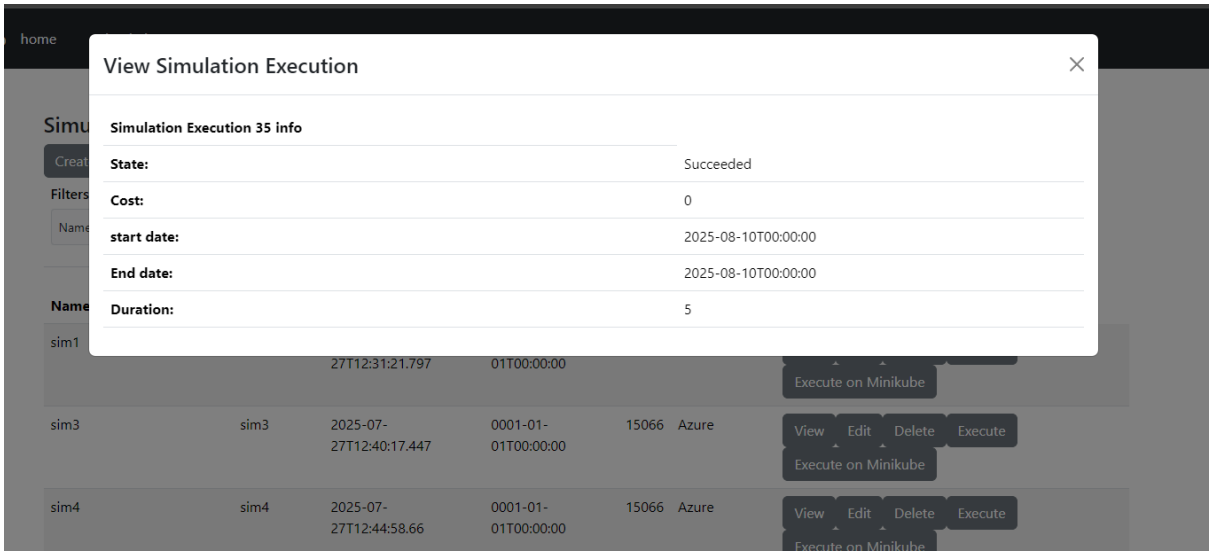


Figure 49. The simulation's execution view

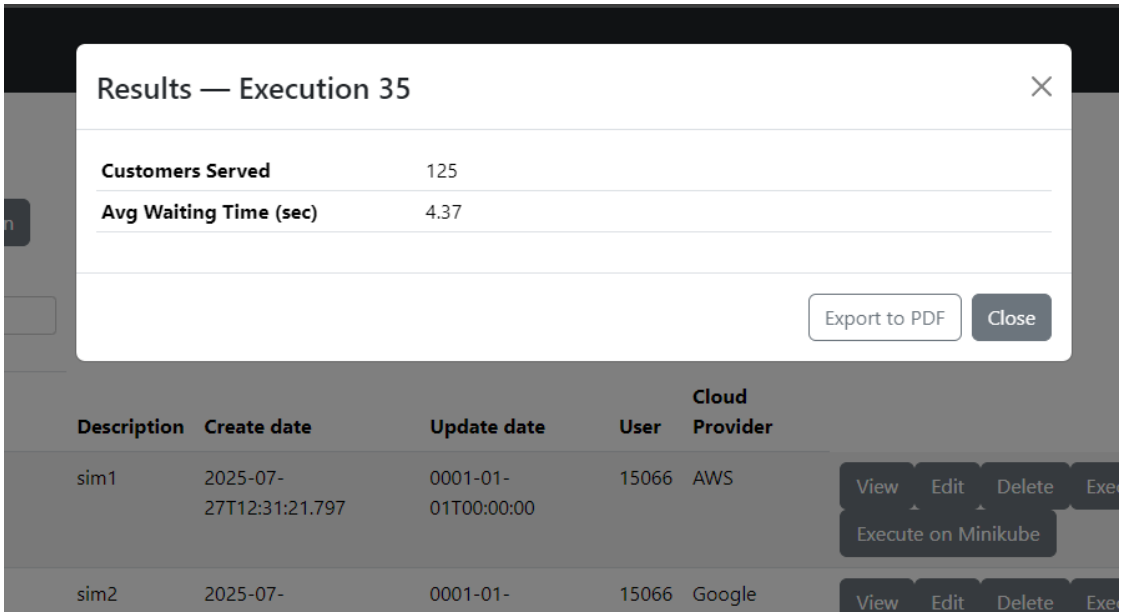


Figure 50. The simulation's execution results

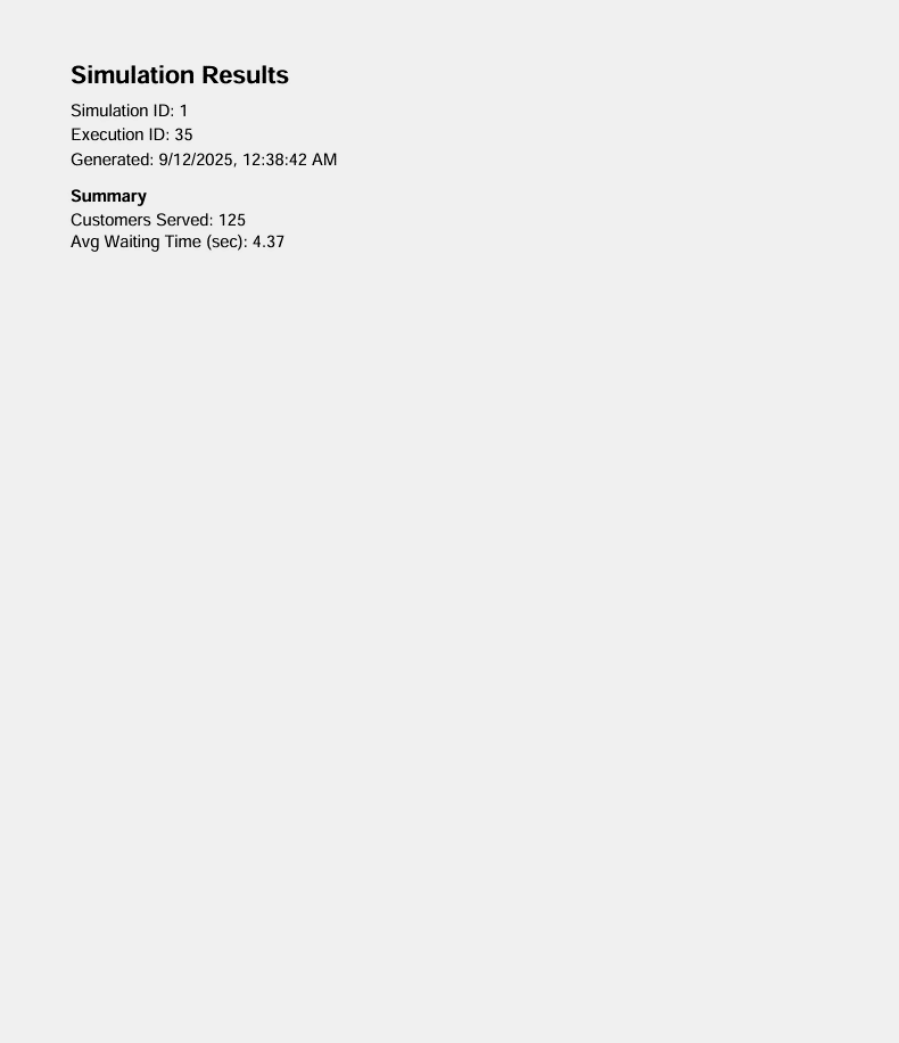


Figure 51. The PDF of the simulation's execution results

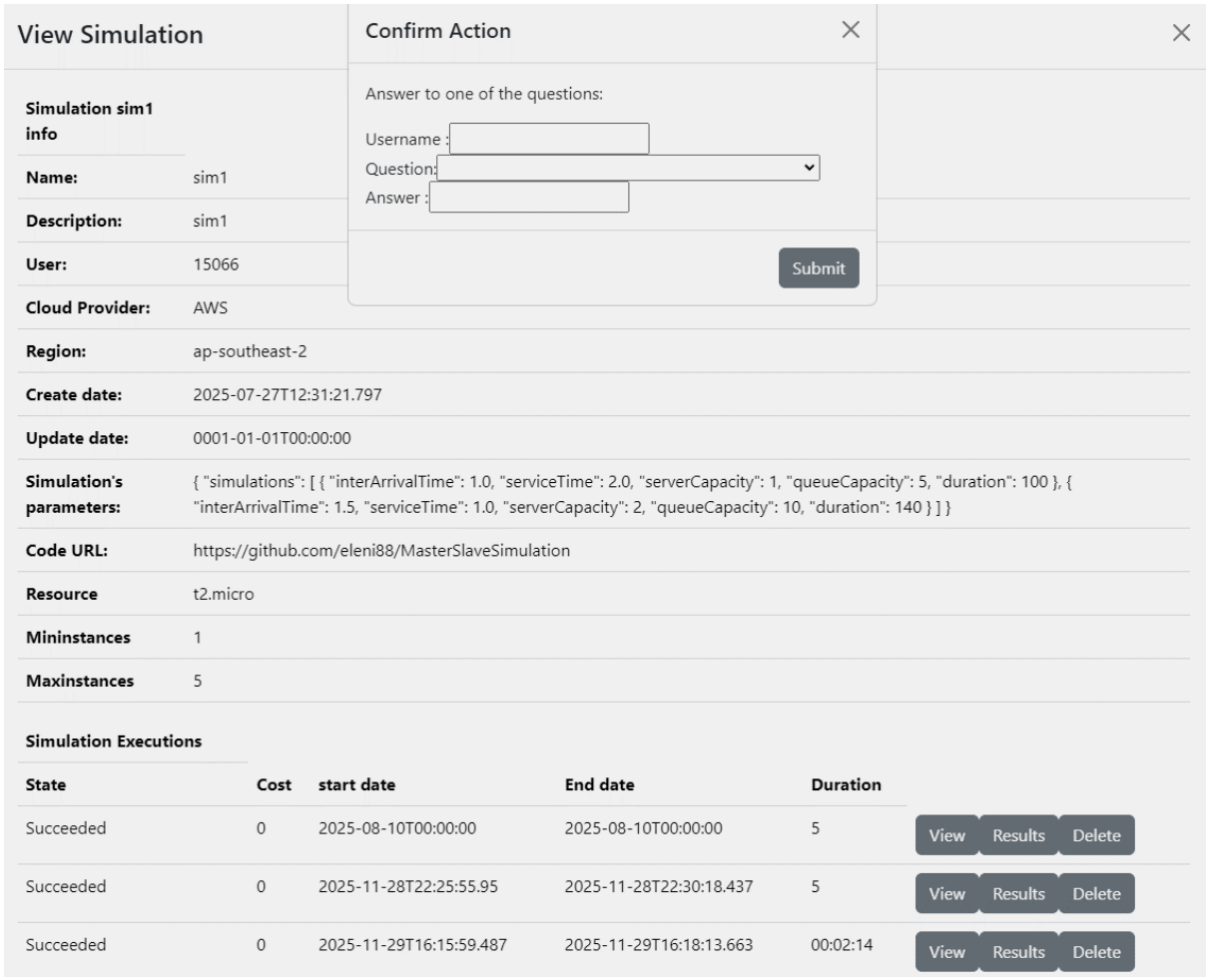


Figure 52. The simulation's execution delete – Security Questions window

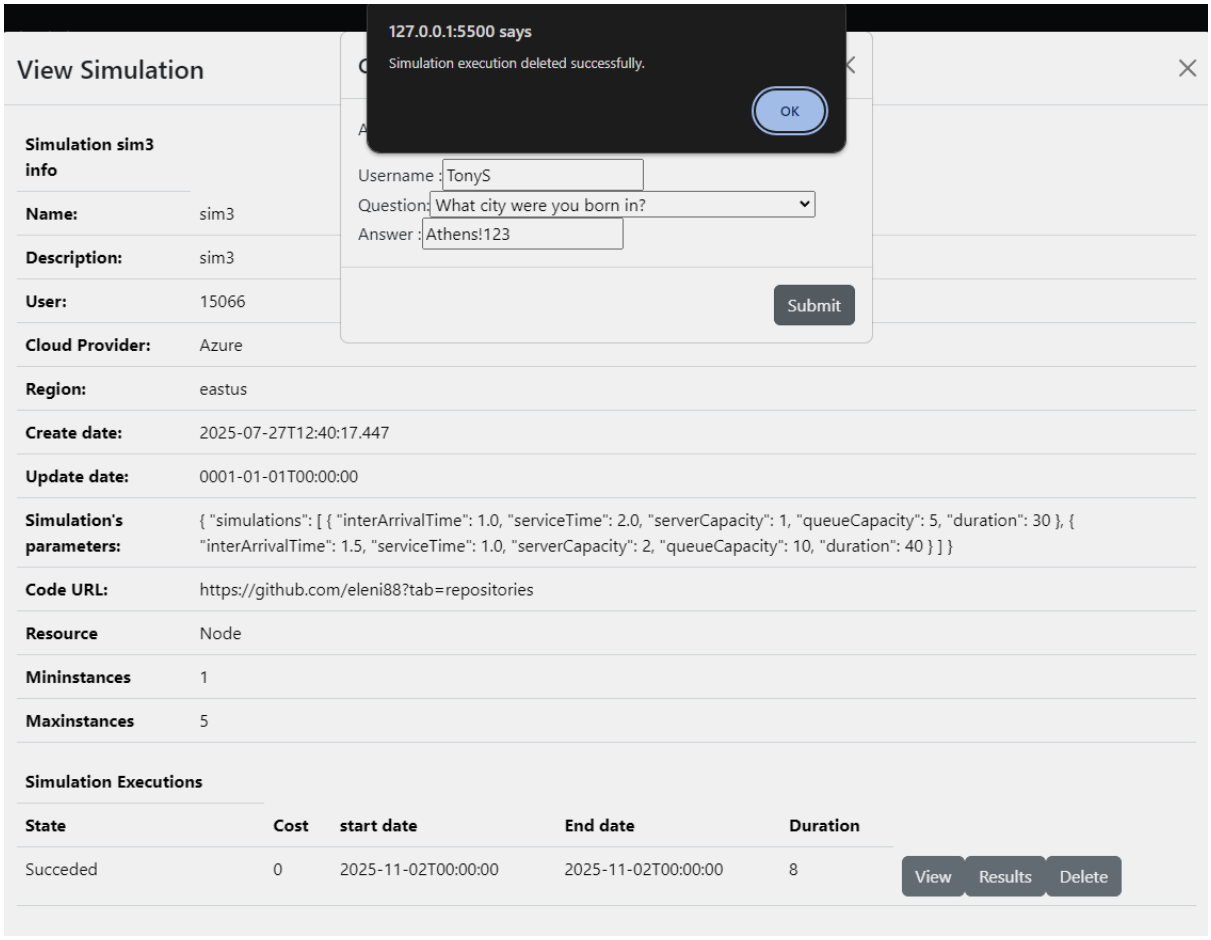


Figure 53. The simulation's execution deletion - Successful deletion

6

System Testing & Evaluation

In this chapter, our system is evaluated in order to assess the degree in which it satisfies its non-functional requirements and especially the performance ones. Further, a simulation case study is described in which a Master/Slave Simulation project is presented that is used to test the system from the cloud point of view, i.e., in a real cloud-based environment. In particular, this Master/Slave Simulation project has been developed in order to test how our system clones it from GitHub repository, creates the cloud-based Kubernetes environment and deploys and runs the simulation in it.

6.1 System Evaluation

The system's backend performance was assessed with the use of Grafana k6. Grafana k6 is an open-source, load testing tool that helps developers to run load, and performance tests in order to detect performance issues before they occur in production and make improvements that finally lead to reliable and high-performing applications [121].

The k6 tests can be written in JavaScript or Typescript and their purpose is to simulate realistic users' actions and present the results of the system's behavior.

First, the application's backend and the database were installed in the minikube environment following the installation steps described in Section 5.1. Then, the backend was exposed running the command '**kubectl port-forward svc/simbackend 8080:80'**, which makes the backend accessible from the port 8080.

According to the performance requirement, the response time for any request targeting any system function must not exceed 5 seconds. For our testing purposes, 2 scripts have been created to test the register, user login, get all users and create simulation endpoints. These scripts can be found in the backend's archives in the folder called 'Tests'.

The first script is called 'Register_test.js'; it contains a scenario in which 50 users try to register simultaneously. Specifically, the script creates 50 virtual users (VU), where each VU executes one iteration (iteration=how many times the VUs execute the script) and the maximum duration of the test, before it is forcibly stopped, is 2m. The gracefulStop is also specified at 5s; this configures the duration after the end of the test that k6 waits for the still running iterations to finish [123]. The script also specifies two thresholds containing the conditions that have to be met in order for the test to pass. The first threshold evaluates the rate of HTTP errors with the condition to be less than 1% and the other threshold checks if all requests are below 5s.

To run the Register_test.js script the command '**k6 run --insecure-skip-tls-verify --env RUN_ID=u123 Register_test.js'** needs to run from the script file's location.

The second script is called 'login_test.js' and it contains a scenario in which the 50 previously registered users try to login simultaneously. Similar to the Register_test.js script, each VU executes one iteration, the maximum duration is 2m and the gracefulStop is at 5s. It has also two thresholds, one evaluating the rate of errors and one checking the time of requests, as exactly the Rejister_test.js. The login_test.js tests the login endpoint, the get all users end point and the create simulation and point.

In order to run the loging_test.js script, the command **k6 run --insecure-skip-tls-verify --env RUN_ID=u123 Login_test.js** has to execute from the script file's location.

In the above commands, the **--insecure-skip-tls-verify** option is used to ignore the secure connection created by the backend's code [124] as the simbackend-service.yaml creates a service that listens to port 80, the port-forward command makes it available in port 8080 and as a result, the requests in the test scripts use http instead of https.

An env variable **(--env RUN_ID=u123**) is also used in the test scripts' commands. This variable is used in the Register_test.js script in order to ensure the created VU uniqueness in every script's run with different **RUN_ID**. It is used in the login_test.js as well so that the same VUs created through the register process can login the system and make requests.

The results of the Register_test.js and the login_test.js are presented below. As it can be seen, the results from the first test were successful concerning the covered registration of 50 simultaneous users. However, in the case of registration's response time, the test failed as it exceeded the 5000ms. The second test failed mainly in terms of the failure rate restriction, which was not satisfied in the login and get users cases. The failure of these tests signifies that our implementation requires some improvement, which we consider as a future work direction.

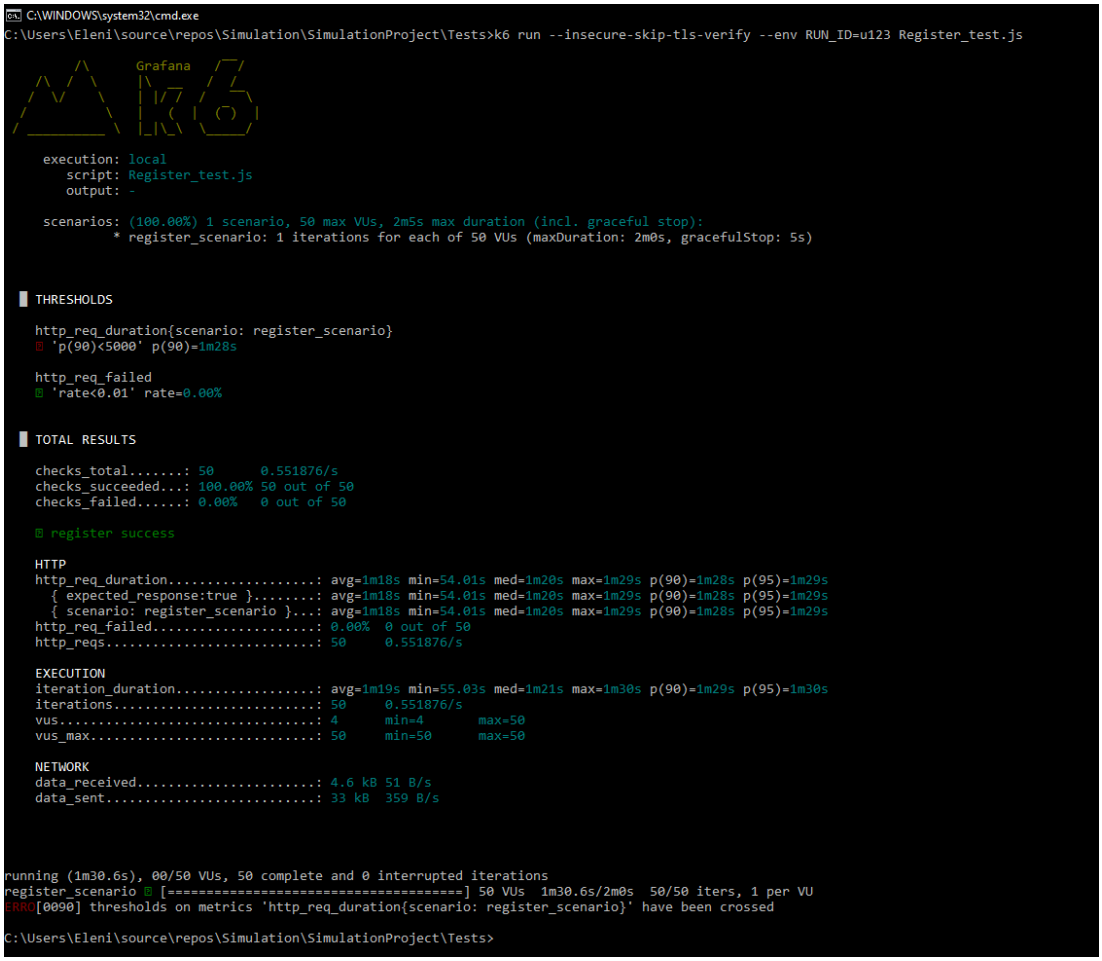


Figure 54. Register_test results

```
  THRESHOLDS

    http_req_duration{name:get_users}
    ✓ 'p(100)<5000' p(100)=71.14ms

    http_req_duration{name:simulation_create}
    ✓ 'p(100)<5000' p(100)=0s

    http_req_duration{name:user_login}
    ✓ 'p(100)<5000' p(100)=2.05s

    http_req_failed{name:get_users}
    ✗ 'rate<0.01' rate=100.00%

    http_req_failed{name:simulation_create}
    ✓ 'rate<0.01' rate=0.00%

    http_req_failed{name:user_login}
    ✗ 'rate<0.01' rate=100.00%


  TOTAL RESULTS

    checks_total.......: 100     40.742587/s
    checks_succeeded...: 0.00%   0 out of 100
    checks_failed......: 100.00% 100 out of 100

    ✗ is status 200
      ↳  0% — ✓ 0 / ✗ 50
    ✗ GET 200
      ↳  0% — ✓ 0 / ✗ 50

    HTTP
    http_req_duration..............: avg=1.01s  min=0s    med=1.03s max=2.05s   p(90)=2.02s p(95)=2.03s
      { name:get_users }...........: avg=5.01ms min=0s    med=0s    max=71.14ms p(90)=0s    p(95)=57.74ms
      { name:simulation_create }...: avg=0s     min=0s    med=0s    max=0s      p(90)=0s    p(95)=0s
      { name:user_login }..........: avg=2.02s  min=1.99s med=2.02s max=2.05s   p(90)=2.03s p(95)=2.05s
    http_req_failed................: 100.00% 100 out of 100
      { name:get_users }...........: 100.00% 50 out of 50
      { name:simulation_create }...: 0.00%   0 out of 0
      { name:user_login }..........: 100.00% 50 out of 50
    http_reqs......................: 100     40.742587/s

    EXECUTION
    iteration_duration.............: avg=2.28s  min=2.15s med=2.29s max=2.35s   p(90)=2.33s p(95)=2.35s
    iterations.....................: 50      20.371294/s
    vus............................: 50      min=50         max=50
    vus_max........................: 50      min=50         max=50

    NETWORK
    data_received..................: 0 B     0 B/s
    data_sent......................: 10 kB   4.2 kB/s



running (0m02.5s), 00/50 VUs, 50 complete and 0 interrupted iterations
user_scenario ✓ [======================================] 50 VUs  0m02.5s/2m0s  50/50 iters, 1 per VU
ERRO[0002] thresholds on metrics 'http_req_failed{name:get_users}, http_req_failed{name:user_login}' have been crossed

C:\Users\Eleni\source\repos\Simulation\SimulationProject\Tests>
```

Figure 55. Login_test results

6.2 System Testing based on a Simulation Case Study

For the case study where the system deploys and runs a simulation, a simple simulation application has been created in order to test the implemented system/application. This project can be found in the GitHub repository. The simulation application is a discrete-event simulation written in C# following a Master/Slave architecture. This simulation application makes use of the SimSharp .Net library [72] and returns the simulation results in JSON format. It represents a simple simulation model of a queue where customers arrive, wait in line, and are served one by one.

The development of the simulation application is based on concepts described in [73]. Specifically, within the queueing model implemented, the arriving customers represent the 'Entities', while the servers that handle their service requests correspond to the 'Resources', which are entities (the servers) that serve other entities (the customers). The queue models the maximum number of customers permitted to wait in line, corresponding to the 'List Processing'. The customers' arrival and the start and end of their service correspond to the 'Events' which change the state of the system. The service time for each customer's service corresponds to the 'Activity', which is a time duration known from the beginning, while there are also the 'Delays' that are time durations unknown as they may be caused by other events that may occur. The 'Events' occur as a result of the activity times and delays. Moreover, the use of distinct parameter sets in JSON format, which are deployed to individual slave pods to execute simulations under varying configurations, reflects the approach to experimentation with alternative system inputs as discussed in [73]. Finally, the overall workflow of the simulation application (problem formulation (defining the queueing system with capacity constraints and service rates), structured data input (via params.JSON), modeling (implemented using SimSharp), execution (orchestrated by the master pod through the dynamic creation of slave pods), and analysis (aggregating simulation outcomes within the master)), mirrors the structured methodology for conducting simulation studies outlined in [73].

The following are the input parameters the simulation takes in JSON format through a ConfigMap:

- Inter-arrival time: Time between customer arrivals.
- Service time: Time to serve each customer.
- Server capacity: Number of servers.
- Queue capacity: Maximum number of customers that can wait in line.
- Simulation time or stopping condition (duration): How long to run the simulation.

The Master/Slave architecture followed consists of the MasterWorker project and the ServiceWorker project, both under the same .Net Solution project called 'MasterSlaveSimulation'.

The MasterWorker project is an Asp.NET core web API that has the role of an orchestrator, as it starts the simulation, creates slave pods and collects the simulation results from the slave pods. The orchestrator class inherits from the 'BackgroundService' class, which implements the IHostedService interface [71]. Specifically, the BackgroundService gives the orchestrator the ability to run continuously in the background.

The MasterWorker creates the slave pods and collects the results. It reads the params.json, which contains the parameter set for the simulation, creates the slave pods dynamically through the kubernetes API to run the simulation jobs, it communicates with the pods through REST APIs and polls them in order to gather the simulation results. The results can then be returned by invoking a controller that wraps the master worker with the '/results' endpoint. When the polling procedure is completed, due to simulation's completion or error's occurrence, and the slave pods are no longer needed, the MasterWorker deletes them.

The SlaveWorker project is also an Asp.NET core web API which uses the SimSharp .NET library to create simulations and returns the simulations' results via the '/result' endpoint. It reads the params.json sent to it by the master pod through ENV (environment variable) and

runs the simulation. It runs a procedure in which every 'inter-arrival' seconds a new customer is created and each customer waits in the queue to be serviced for 'Service time'. The simulation runs for 'duration' seconds. The results of each simulation is returned, in a JSON format, in the '/result' endpoint and the master pod makes HTTP calls to receive them.

For demonstration and testing purposes, the case study was applied from localhost. While the 'Execute on Minikube' button was pressed, we run the command '**kubectl get pods -l app=master –w'** in order to observe the master pod's status. At the time it became 'Running', we run in a second cmd window the '**kubectl port-forward service/master 30080:80'** so that the master pod can be accessible in port 30080 and the System can communicate with the master pod through this port in order to collect the simulation's results.

Below, the steps for managing and running the simulation are shortly analysed along with screenshots that depict their results. In Figure 56, a simulation that we have just created is shown with its details. Figure 57 depicts that simulation with one execution that has been already completed. In Figure 58, the view of the simulation is shown with a new execution that has been initiated. In Figures 59 and 60 we can see in the underlying database the information concerning the executions of the specific simulation before the new execution was initiated and after the new execution's completion. Figure 61 shows the simulation's view with the new execution now completed.

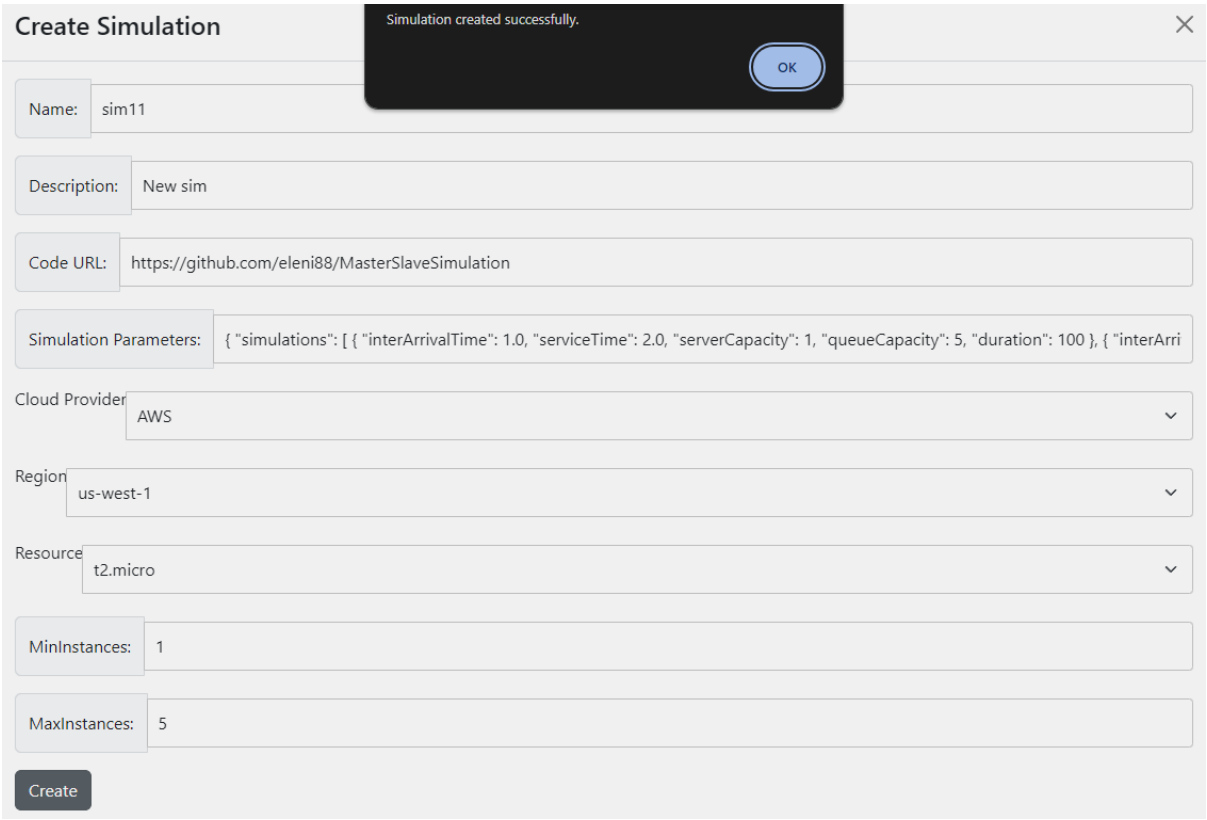


Figure 56. A new simulation to run

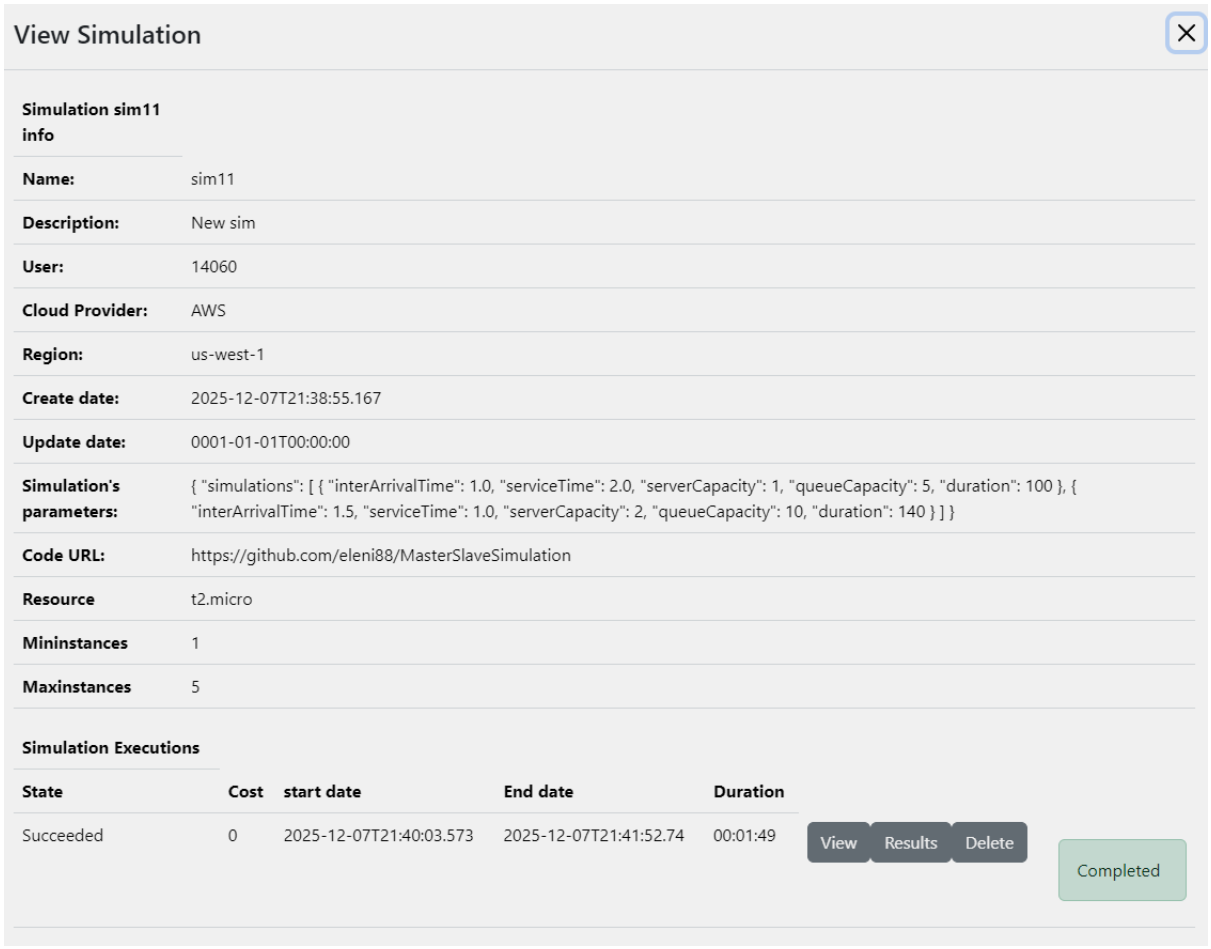


Figure 57. Simulation with one completed execution

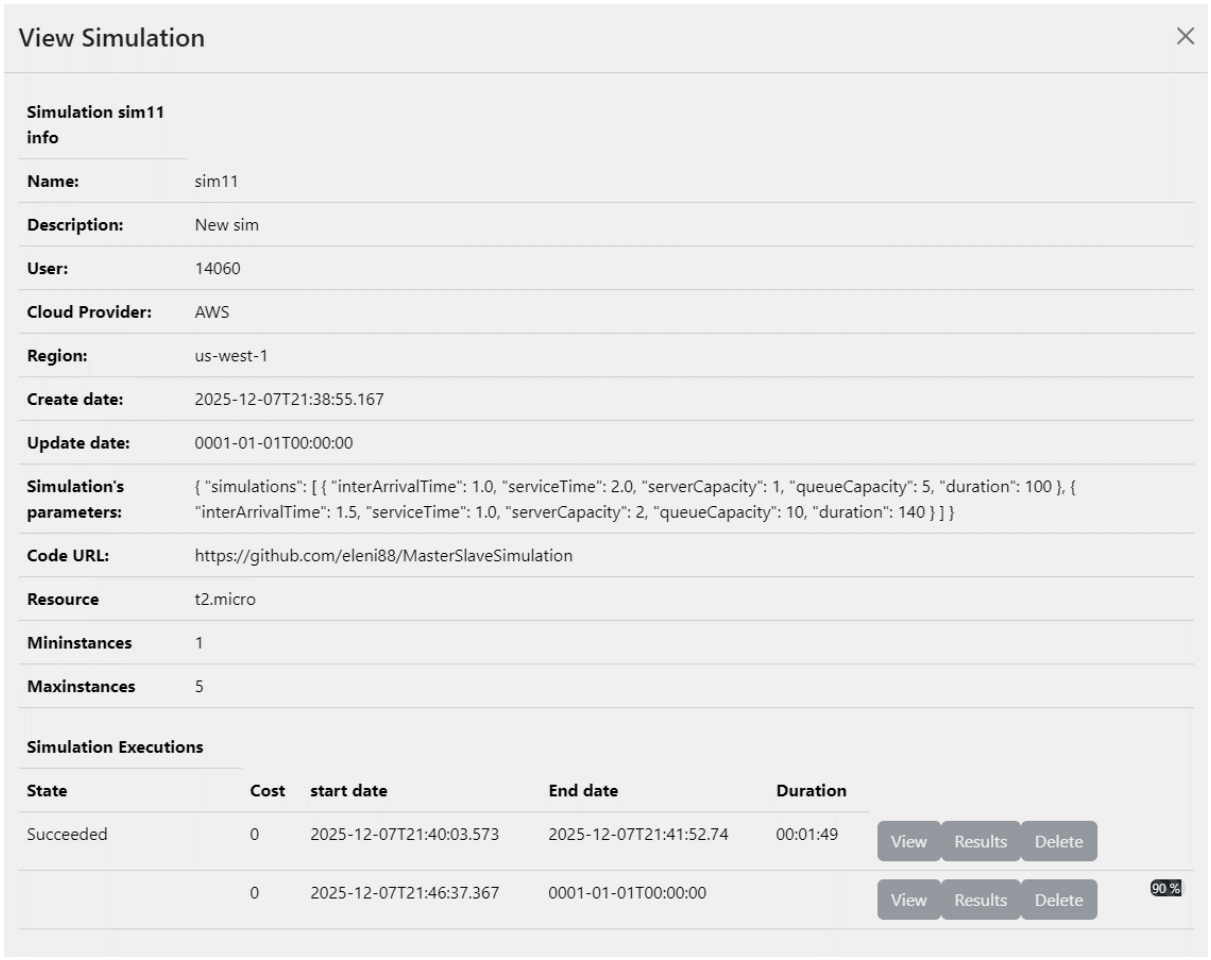


Figure 58. Simulation with new execution in progress

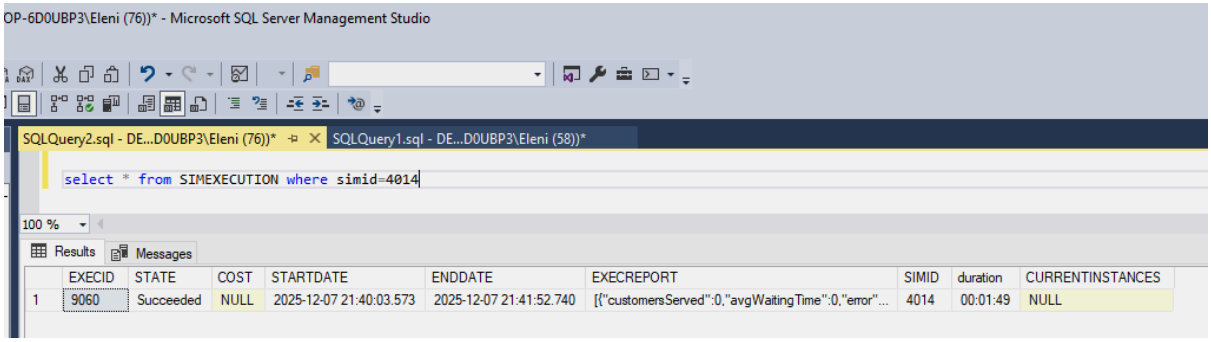
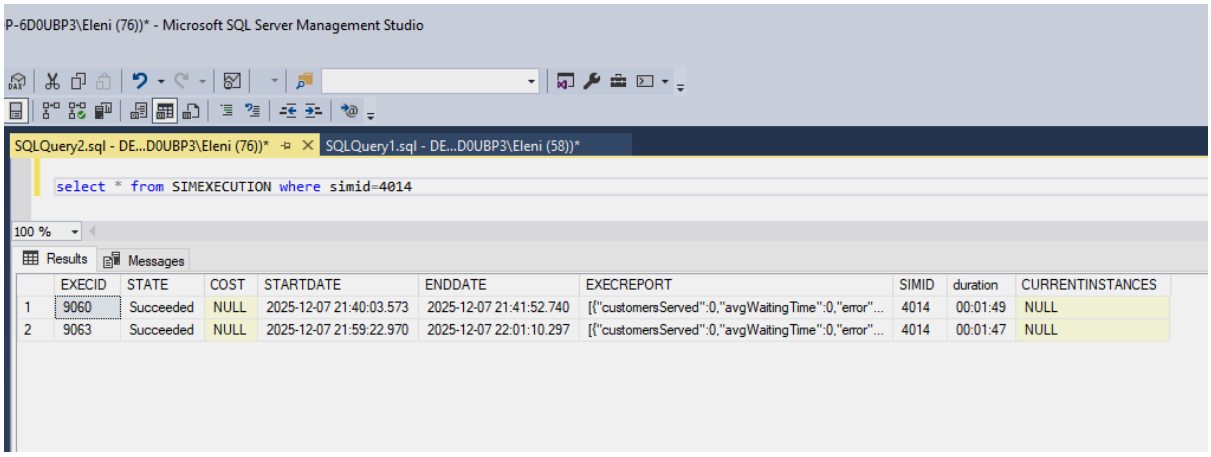


Figure 59. The simulation's executions before the new initiation



	EXECID	STATE	COST	STARTDATE	ENDDATE	EXECREPORT	SIMID	duration	CURRENTINSTANCES
1	9060	Succeeded	NULL	2025-12-07 21:40:03.573	2025-12-07 21:41:52.740	[{"customersServed":0,"avgWaitingTime":0,"error"...	4014	00:01:49	NULL
2	9063	Succeeded	NULL	2025-12-07 21:59:22.970	2025-12-07 22:01:10.297	[{"customersServed":0,"avgWaitingTime":0,"error"...	4014	00:01:47	NULL

Figure 60. The simulation's executions with the new execution completed

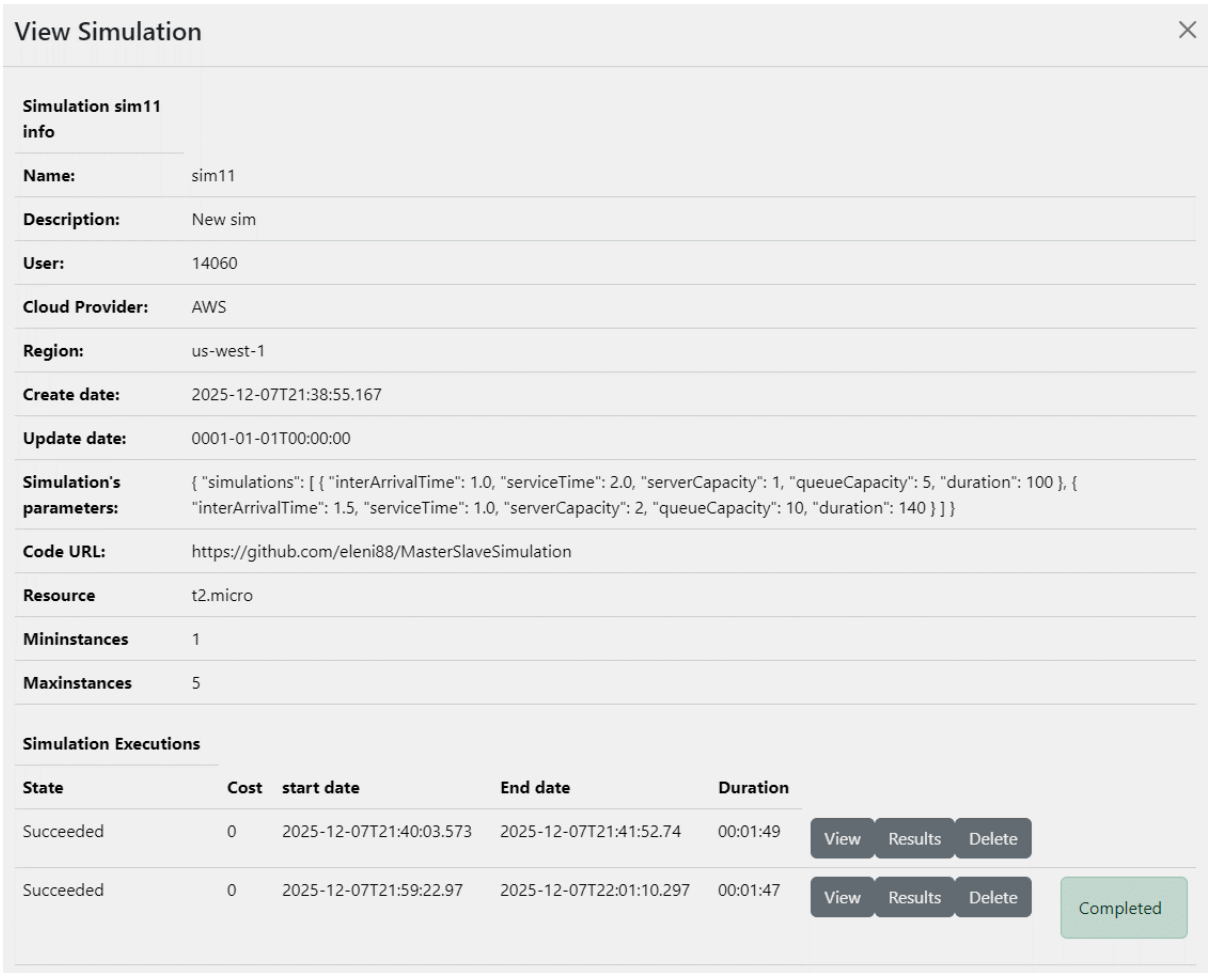


Figure 61. The simulation's new execution completed

In Figure 62, we can observe from the master pod's status that it has completed its execution (in terms of the new simulation execution). In order to do so, the command '**kubectl get pods -l app=master –w**' was run at the command prompt. This command uses the '**-w'** option which shows the pod's changes over time. So, for a single simulation's execution (e.g., the new one), the master pod was being initiated for the first 7 seconds, then it started running and in 112 seconds it had completed.

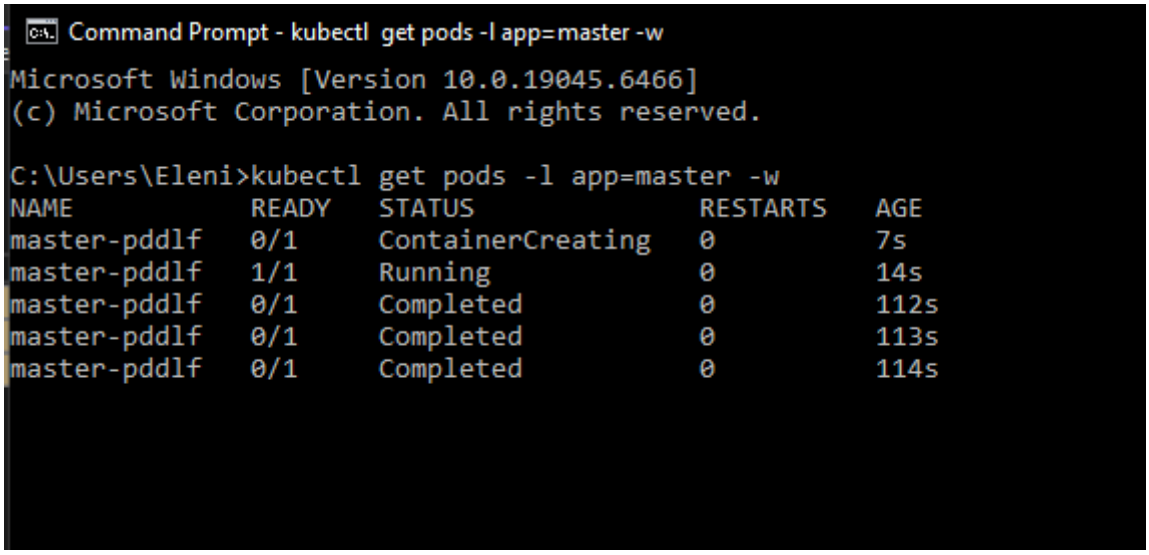


Figure 62. Master pod completed

7

Conclusions

This thesis proposes a simulation as a service application which utilizes cloud computing technologies, such as terraform, Docker, Kubernetes and GitHub, in order to enable users to run simulations in different clouds. The benefits of this system are its independence from specific simulation tools, its support of various cloud providers, the automated orchestration and deployment of Kubernetes environments and simulation code on top of them for executing simulations and its scalability along with its user-friendly interface.

During the development of this project, experience was gained upon the creation of UML diagrams, the use of C# and its combination with external tools/services like GitHub repositories and the Kubernetes API. Experience was also gained related to cloud technologies and how an application can be deployed on Kubernetes as well as the use of command prompt and PowerShell in order to run minikube commands.

There are some directions in which this project can be improved. First, the ability to pause and restart simulations' executions as well as the ability to add or remove cloud regions. As for the resources used for the simulation's run, the system could offer the option of pods' CPU, apart from their max and min instances, in order to set a limit in pod's CPU to scale vertically accordingly. Another improvement could be the automated generation of the necessary YAML files for the simulation code's deployment on Kubernetes based on the user's input instead of being provided in the GitHub repository. Furthermore, apart from giving as input the GitHub URL, the simulation's code could be uploaded in the system as well. The system's performance testing needs to be completed as not all the end points have been tested and most importantly, the system performance must be improved based on the results of the conducted performance tests in Section 6.1. In addition, the configuration for the automated horizontal scaling is missing from the part of code that creates the Terraform file as there was not enough time to implement it. Further, some changes need to take place at the code level as the simulation process cannot be completed when the system runs inside minikube cluster. The code was initially developed and tested in localhost and there are some differences between a localhost and a minikube environment. The functionality of users', simulations' and simulation executions' search can be implemented in the future as it is not currently supported. The option of changing the web-site's language can be added as well. Finally, the simulations' results can be viewed with the use of dashboards, improving in this way the results visualization and the respective decision making.

References

[1] Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing* (NIST Special Publication 800-145). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-145

[2] Yasar, K., Chai, W., & Bigelow, S. J. (2024, June). What is cloud computing? Types, examples and benefits. *TechTarget*. https://www.techtarget.com/searchcloudcomputing/definition/cloud-computing

[3] Giordanelli, R., & Mastroianni, C. (2010). The cloud computing paradigm: Characteristics, opportunities and research issues. *Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)*, 5, 11.

[4] Voorsluys, W., Broberg, J., & Buyya, R. (2011). Introduction to cloud computing. In R. Buyya, J. Broberg, & A. Goscinski (Eds.), *Cloud computing* (pp. 1 - 41). https://doi.org/10.1002/9780470940105.ch1

[5] Modisane, P., & Jokonya, O. (2021). Evaluating the benefits of cloud computing in small, medium and micro-sized enterprises (SMMEs). *Procedia Computer Science, 181*, 784–792. https://doi.org/10.1016/j.procs.2021.01.231

[6] Sether, A. (2016). Cloud computing benefits. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.2781593

[7] Hassan, H. B., Barakat, S. A., & Sarhan, Q. I. (2021). Survey on serverless computing. *Journal of Cloud Computing, 10*, 39. https://doi.org/10.1186/s13677-021-00253-7

[8] Cassel, G. A. S., Rodrigues, V. F., Righi, R. da R., Bez, M. R., Nepomuceno, A. C., & da Costa, C. A. (2022). Serverless computing for Internet of Things: A systematic literature review. *Future Generation Computer Systems, 128*, 299–316. https://doi.org/10.1016/j.future.2021.10.020

[9] Li, Z., Guo, L., Cheng, J., Chen, Q., He, B., & Guo, M. (2022). The serverless computing survey: A technical primer for design architecture. *ACM Computing Surveys, 54*(10s), Article 220, 34 pages. https://doi.org/10.1145/3508360

[10] Shafiei, H., Khonsari, A., & Mousavi, P. (2022). Serverless computing: A survey of opportunities, challenges, and applications. *ACM Computing Surveys, 54*(11s), Article 239, 32 pages. https://doi.org/10.1145/3510611

[11] Bhardwaj, S., Jain, L., & Jain, S. (2010). Cloud computing: A study of infrastructure as a service (IAAS). *International Journal of Engineering and Information Technology, 2*(1), 60–63.

[12] Kavis, M. (2014). Cloud service models. In *Architecting the cloud* (pp. 13–22). https://doi.org/10.1002/9781118691779.ch2

[13] Mohammed, C., & Zeebaree, S. (2021). Sufficient comparison among cloud computing services: IaaS, PaaS, and SaaS: A review. *International Journal of Science and Business,* 5, 17–30. https://doi.org/10.5281/zenodo.4450129

[14] Manvi, S. S., & Shyam, G. K. (2014). Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. *Journal of Network and Computer Applications, 41*, 424–440. https://doi.org/10.1016/j.jnca.2013.10.004

[15] Goyal, S. (2014). Public vs private vs hybrid vs community cloud computing: A critical review. *International Journal of Computer Network and Information Security, 6*(3), 20–29.

[16] Solanke, V., et al. (2013). Private vs public cloud. *International Journal of Computer Science & Communication Networks, 3*(2), 79.

[17] Google Cloud. (n.d.). *What is IaaS?* In *Google Cloud*. Retrieved July 26, 2025, from https://cloud.google.com/learn/what-is-iaas

[18] Google Cloud. (n.d.). *What is multicloud?* In *Google Cloud*. Retrieved July 26, 2025, from https://cloud.google.com/learn/what-is-multicloud

[19] Fujimoto, R., Malik, A., & Park, A. (2010). Parallel and distributed simulation in the cloud. *SCS M&S Magazine, 3*.

[20] Shekhar, S., Abdel-Aziz, H., Walker, M., Caglar, F., Gokhale, A., & Koutsoukos, X. (2016). A simulation as a service cloud middleware. *Annals of Telecommunications, 71*(3–4), 93–108. https://doi.org/10.1007/s12243-015-0475-6

[21] Docker. (n.d.). *What is Docker?* In *Docker documentation*. Retrieved August 01, 2025, from https://docs.docker.com/get-started/docker-overview/

[22] Potdar, A., Narayan, D. G., Kengond, S., & Mulla, M. (2020). Performance evaluation of Docker container and virtual machine. *Procedia Computer Science, 171*, 1419–1428. https://doi.org/10.1016/j.procs.2020.04.152

[23] Rad, B. B., Bhatti, H. J., & Ahmadi, M. (2017). An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS), 17(3),* 228.

[24] Kubernetes. (n.d.). *Why you need Kubernetes and what can it do*. In *Kubernetes documentation*. Retrieved August 01, 2025, from https://kubernetes.io/docs/concepts/overview/#why-you-need-kubernetes-and-what-can-it-do

[25] Red Hat. (n.d.). *What is container orchestration?* In *Red Hat*. Retrieved August 01, 2025, from https://www.redhat.com/en/topics/containers/what-is-container-orchestration#how-does-it-work

[26] Poulton, N. (2023). *The Kubernetes book*. Nigel Poulton Ltd.

[27] Kubernetes. (n.d.). *Install tools*. In *Kubernetes documentation*. Retrieved August 02, 2025, from https://kubernetes.io/docs/tasks/tools/

[28] Kubernetes. (n.d.). *kubelet*. In *Kubernetes documentation*. Retrieved July 28, 2025, from https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/

[29] Kubernetes. (n.d.). *Nodes*. In *Kubernetes documentation*. Retrieved July 28, 2025, from https://kubernetes.io/docs/concepts/architecture/nodes/

[30] Kubernetes. (n.d.). *Pods*. In *Kubernetes documentation*. Retrieved October 19, 2025, from https://kubernetes.io/docs/concepts/workloads/pods/

[31] Kubernetes. (n.d.). *Control plane components*. In *Kubernetes documentation*. Retrieved July 28, 2025, from https://kubernetes.io/docs/concepts/architecture/#control-plane-components

[32] Zhong, Z., & Buyya, R. (2020). A cost-efficient container orchestration strategy in Kubernetes-based cloud computing infrastructures with heterogeneous resources. *ACM Transactions on Internet Technology, 20*(2), Article 15, 24 pages. https://doi.org/10.1145/3378447

[33] Martínez Saucedo, A., Rodríguez, G., Rocha, F., & dos Santos, R. (2024). Migration of monolithic systems to microservices: A systematic mapping study. *Information and Software Technology, 177*, 107590. https://doi.org/10.1016/j.infsof.2024.107590

[34] Maggi, K., Verdecchia, R., Scommegna, L., & Vicario, E. (2025). Evolution of code technical debt in microservices architectures. *Journal of Systems and Software, 222*, 112301. https://doi.org/10.1016/j.jss.2024.112301

[35] Banks, J., Carson, J. S., Nelson, B. L., & Nicol, D. M. (2005). *Discrete-event system simulation* (4th ed.). Prentice Hall.

[36] Maria, A. (1997). Introduction to modeling and simulation. In *Proceedings of the 29th conference on Winter simulation (WSC '97)* (pp. 7–13). IEEE Computer Society. https://doi.org/10.1145/268437.268440

[37] Dudewicz, E. J., & Karian, Z. A. (2003). Simulation languages. In H. Bidgoli (Ed.), *Encyclopedia of information systems* (pp. 105–120). Elsevier. https://doi.org/10.1016/B0-12-227240-4/00160-X

[38] Murray-Smith, D. J. (2012). *Continuous system simulation*. Springer Science & Business Media.

[39] Saravanos, A., & Curinga, M. X. (2023). Simulating the software development lifecycle: The waterfall model. *Applied System Innovation, 6*(6), 108. https://doi.org/10.3390/asi6060108

[40] Royce, W. W. (1987). Managing the development of large software systems: Concepts and techniques. In *Proceedings of the 9th International Conference on Software Engineering (ICSE '87)* (pp. 328–338). IEEE Computer Society Press.

[41] Lucidchart. (n.d.). *ER diagrams*. Lucidchart. Retrieved April 23, 2025, from https://www.lucidchart.com/pages/er-diagrams

[42] Meshram, S. U. (2021). Evolution of modern web services – REST API with its architecture and design. *International Journal of Research in Engineering, Science and Management, 4*(7), 83–86. https://journal.ijresm.com/index.php/ijresm/article/view/970

[43] Schwichtenberg, H. (2018). *Modern data access with entity framework core*. Apress.

[44] FluentValidation. (n.d.). *Getting started*. Retrieved May 15, 2025, from https://docs.fluentvalidation.net/en/latest/

[45] Polasek, P., Janousek, V., & Ceska, M. (2014). Petri net simulation as a service. *1160*, 353–362.

[46] Pearl, J. (1988). Belief updating by network propagation. In Probabilistic reasoning in intelligent systems: Networks of plausible inference (pp. 143–237). Morgan Kaufmann. https://doi.org/10.1016/B978-0-08-051489-5.50010-2

[47] Muralidhar, K. (2003). Monte Carlo simulation. In H. Bidgoli (Ed.), *Encyclopedia of information systems* (pp. 193–201). Elsevier. https://doi.org/10.1016/B0-12-227240-4/00114-3

[48] Kleijnen, J. P. C. (2009). Kriging metamodeling in simulation: A review. *European Journal of Operational Research, 192*(3), 707–716. https://doi.org/10.1016/j.ejor.2007.10.013

[49] Saleem, M., & Khan, Z. (2023). Healthcare simulation: An effective way of learning in health care. *Pakistan Journal of Medical Sciences, 39*(4), 1185–1190. https://doi.org/10.12669/pjms.39.4.7145

[50] Kaufman, D., & Ireland, A. (2016). Enhancing teacher education with simulations. *TechTrends, 60*, 260–267. https://doi.org/10.1007/s11528-016-0049-0

[51] Banks, J. (1999). Introduction to simulation. In *Proceedings of the 1999 Winter Simulation Conference*.

[52] Dorokhin, S., Artemov, A., Likhachev, D., Novikov, A., & Starkov, E. (2020). Traffic simulation: An analytical review. *IOP Conference Series: Materials Science and Engineering, 918*(1), 012058. https://doi.org/10.1088/1757-899X/918/1/012058

[53] Imbert, C. (2017). Computer simulations and computational models in science. In L. Magnani & T. Bertolotti (Eds.), *Springer handbook of model-based science* (pp. 781–818). Springer. https://doi.org/10.1007/978-3-319-30526-4_34

[54] Brodland, G. W. (2015). How computational models can help unlock biological systems. *Seminars in Cell & Developmental Biology, 47–48*, 62–73. https://doi.org/10.1016/j.semcdb.2015.07.001

[55] Hashemi, A. S., & Sattarvand, J. (2015). Application of ARENA simulation software for evaluation of open pit mining transportation systems – A case study. In C. Niemann-Delius (Ed.), *Proceedings of the 12th International Symposium Continuous Surface Mining – Aachen 2014* (pp. 223–236). Springer. https://doi.org/10.1007/978-3-319-12301-1_20

[56] Borshchev, A. (2013). *The big book of simulation modeling: Multimethod modeling with AnyLogic 6*. AnyLogic North America.

[57] McMullen, P. (2022). A heuristic search approach to multidimensional scaling. *American Journal of Operations Research, 12*(5), 179–193. https://doi.org/10.4236/ajor.2022.125010

[58] Appleget, J. (2021). An introduction to wargaming and modeling and simulation. In C. Turnitsa, C. Blais, & A. Tolk (Eds.), *Simulation and wargaming* (pp. 1 - 22). https://doi.org/10.1002/9781119604815.ch1

[59] Natarajan, A. (2016). The design of a novel bearing manufacturing line based on simulation. *American Journal of Operations Research, 6*(6), 425–435. https://doi.org/10.4236/ajor.2016.66039

[60] Fousek, J., Kuncova, M., Fábry, J., & Zoltay, Z. (2017). Discrete Event Simulation-Production Model In SIMUL8. In *ECMS* (pp. 229-234).

[61] Horri, N., & Pietraszko, M. (2022). A tutorial and review on flight control co-simulation using MATLAB/Simulink and flight simulators. *Automation, 3*(3), 486–510. https://doi.org/10.3390/automation3030025

[62] Varga, A., & Hornig, R. (2008). An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops* (pp. 1–10).

[63] Bhardwaj, R., Dixit, V. S., & Upadhyay, A. K. (2010). An overview on tools for peer to peer network simulation. *International Journal of Computer Applications, 1*(19), 70–76. https://doi.org/10.5120/400-596

[64] Meyer, M., & Webb, P. (2005). Modular, layered architecture: The necessary foundation for effective mass customisation in software. *International Journal of Mass Customisation, 1*(1), 55–68. https://doi.org/10.1504/IJMASSC.2005.007349

[65] Borshchev, A., Karpov, Y., & Kharitonov, V. (2002). Distributed simulation of hybrid systems with AnyLogic and HLA. *Future Generation Computer Systems, 18*(6), 829–839. https://doi.org/10.1016/S0167-739X(02)00055-9

[66] Rockwell Automation. (n.d.). *Arena simulation: Discrete-event modeling*. Retrieved July 27, 2025, from https://www.rockwellautomation.com/en-gb/products/software/arena-simulation/discrete-event-modeling.html

[67] Mozilla. (n.d.). *Cross-site request forgery (CSRF)*. In *MDN Web Docs*. Retrieved July 28, 2025, from https://developer.mozilla.org/en-US/docs/Web/Security/Attacks/CSRF

[68] Mozilla. (n.d.). *Privacy sandbox: Partitioned cookies*. In *MDN Web Docs*. Retrieved July 28, 2025, from https://developer.mozilla.org/en-US/docs/Web/Privacy/Guides/Privacy_sandbox/Partitioned_cookies

[69] Microsoft. (n.d.). *DbContext (System.Data.Entity)*. In *Microsoft Docs*. Retrieved July 28, 2025, from https://learn.microsoft.com/en-us/dotnet/api/system.data.entity.dbcontext?view=entity-framework-6.2.0

[70] Vue.js. (n.d.). *Vue Router guide*. Retrieved May 15, 2025, from https://router.vuejs.org/guide/

[71] Microsoft. (n.d.). *Hosted services in ASP .NET Core*. In *Microsoft Docs*. Retrieved June 20, 2025, from https://learn.microsoft.com/en-us/aspnet/core/fundamentals/host/hosted-services?view=aspnetcore-9.0&tabs=visual-studio

[72] Heal-Research. (n.d.). *SimSharp* [Computer software]. GitHub. Retrieved June 20, 2025, from https://github.com/heal-research/SimSharp

[73] Banks, J. (2000, December). Introduction to simulation. In *2000 Winter simulation conference proceedings (Cat. No. 00CH37165)* (Vol. 1, pp. 9-16). IEEE.

[74] Extio Technology. (2023, June 16). *Mastering Kubernetes pod-to-pod communication: A comprehensive guide*. Medium. Retrieved July 21, 2025, from https://medium.com/@extio/mastering-kubernetes-pod-to-pod-communication-a-comprehensive-guide-46832b30556b

[75] Raczynski, S. (2003). Continuous simulation. In H. Bidgoli (Ed.), *Encyclopedia of information systems* (pp. 267–286). Elsevier. https://doi.org/10.1016/B0-12-227240-4/00018-6

[76] Gustafsson, L., & Sternad, M. (2016). A guide to population modelling for simulation. *Open Journal of Modelling and Simulation*, *4*(02), 55.

[77] Tisue, S., & Wilensky, U. (2004, May). Netlogo: A simple environment for modeling complexity. In *International conference on complex systems* (Vol. 21, pp. 16-21).

[78] Krajzewicz, D., Erdmann, J., Behrisch, M., & Bieker, L. (2012). Recent development and applications of SUMO-Simulation of Urban MObility. *International journal on advances in systems and measurements*, *5*(3&4), 128-138.

[79] Google. (n.d.). *Cookies and CHIPS*. Privacy Sandbox. Retrieved August 5, 2025, from https://privacysandbox.google.com/cookies/chips

[80] Abualkishik, A. Z., Alwan, A. A., & Gulzar, Y. (2020). Disaster recovery in cloud computing systems: An overview. *International Journal of Advanced Computer Science and Applications*, *11*(9).

[81] Nabi, M., Toeroe, M., & Khendek, F. (2016). Availability in the cloud: State of the art. *Journal of Network and Computer Applications, 60*, 54–67. https://doi.org/10.1016/j.jnca.2015.11.014

[82] Microsoft Azure. (n.d.). *What is disaster recovery?* In *Azure Cloud Computing Dictionary*. Retrieved July 31, 2025, from https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-disaster-recovery

[83] Amazon Web Services (AWS). (n.d.). *Disaster recovery options in the cloud*. In *Workloads on AWS: Disaster recovery whitepaper*. Retrieved July 31, 2025, from https://docs.aws.amazon.com/whitepapers/latest/disaster-recovery-workloads-on-aws/disaster-recovery-options-in-the-cloud.html

[84] Ali, M., Khan, S. U., & Vasilakos, A. V. (2015). Security in cloud computing: Opportunities and challenges. *Information Sciences, 305*, 357–383. https://doi.org/10.1016/j.ins.2015.01.025

[85] Pahl, C., & Jamshidi, P. (2016). Microservices: A systematic mapping study. In *Proceedings of the 6th International Conference on Cloud Computing and Services Science – Volume 1: CLOSER* (pp. 137–146). SciTePress. https://doi.org/10.5220/0005785501370146

[86] Velepucha, V., & Flores, P. (2023). A survey on microservices architecture: Principles, patterns and migration challenges. *IEEE Access, 11*, 88339–88358. https://doi.org/10.1109/ACCESS.2023.3305687

[87] Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. In M. Mazzara & B. Meyer (Eds.), *Present and ulterior software engineering* (pp. 195–216). Springer. https://doi.org/10.1007/978-3-319-67425-4_12

[88] Minikube. (n.d.). *Start minikube (Windows via Chocolatey)*. Retrieved October 9, 2025, from https://minikube.sigs.k8s.io/docs/start/?arch=%2Fwindows%2Fx86-64%2Fstable%2Fchocolatey

[89] Kubernetes. (n.d.). *Control plane to node communication*. In *Kubernetes documentation*. Retrieved August 02, 2025, from https://kubernetes.io/docs/concepts/architecture/control-plane-node-communication/

[90] Rivera, S. (2023, August 18). *Understanding kubelet and kubectl in Kubernetes: Key components explained*. Medium. Retrieved August 02, 2025, from https://medium.com/@sinai.rivera/understanding-kubelet-and-kubectl-in-kubernetes-key-components-explained-2d8a7e4cbcff

[91] etcd. (n.d.). *etcd*. Retrieved August 02, 2025, from https://etcd.io/

[92] Kubernetes. (n.d.). *kube-scheduler*. In *Kubernetes documentation*. Retrieved August 02, 2025, from https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/

[93] Kubernetes. (n.d.). *kube-controller-manager*. In *Kubernetes documentation*. Retrieved August 02, 2025, from https://kubernetes.io/docs/concepts/architecture/#kube-controller-manager

[94] Kubernetes. (n.d.). *Controller*. In *Kubernetes documentation*. Retrieved August 02, 2025, from https://kubernetes.io/docs/concepts/architecture/controller/

[95] Kubernetes. (n.d.). *Container runtimes*. In *Kubernetes documentation*. Retrieved August 02, 2025, from https://kubernetes.io/docs/setup/production-environment/container-runtimes/

[96] Mozilla. (n.d.). *Browser compatibility for partitioned cookies*. In *MDN Web Docs*. Retrieved August 04, 2025, from https://developer.mozilla.org/en-US/docs/Web/Privacy/Guides/Privacy_sandbox/Partitioned_cookies#browser_compatibility

[97] Mozilla. (n.d.). *CORS (Cross-Origin Resource Sharing)*. In *MDN Web Docs*. Retrieved August 04, 2025, from https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS

[98] Microsoft. (n.d.). *Configure JWT bearer authentication in ASP.NET Core*. In *Microsoft Docs*. Retrieved August 03, 2025, from https://learn.microsoft.com/en-us/aspnet/core/security/authentication/configure-jwt-bearer-authentication?view=aspnetcore-9.0

[99] Microsoft. (n.d.). *ASP.NET Core web host*. In *Microsoft Docs*. Retrieved August 04, 2025, from https://learn.microsoft.com/en-us/aspnet/core/fundamentals/host/web-host?view=aspnetcore-9.0

[100] Microsoft. (n.d.). *Configuration in ASP.NET Core*. In *Microsoft Docs*. Retrieved August 04, 2025, from https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration/?view=aspnetcore-9.0

[101] NGINX. (n.d.). *NGINX Gateway Fabric installation manifests*. Retrieved August 11, 2025, from https://docs.nginx.com/nginx-gateway-fabric/install/manifests/

[102] Intertoons. (2022, April 12). *The advantages of using ASP.NET Core with SQL Server*. Retrieved August 08, 2025, from https://intertoons.com/the-advantages-of-using-asp-net-core-with-sql-server-2.html

[103] Microsoft SQL Server Docker Image. (n.d.). *microsoft/mssql-server* [Docker image]. Docker Hub. Retrieved November 19, 2025, from https://hub.docker.com/r/microsoft/mssql-server/

[104] Kubernetes SIG Network. (n.d.). *Gateway API*. Retrieved August 11, 2025, from https://gateway-api.sigs.k8s.io/

[105] Kubernetes SIG Network. (n.d.). *Gateway API guides*. Retrieved August 11, 2025, from https://gateway-api.sigs.k8s.io/guides/

[106] Kubernetes. (n.d.). *Gateway*. In *Kubernetes documentation*. Retrieved August 11, 2025, from https://kubernetes.io/docs/concepts/services-networking/gateway/

[107] Kubernetes SIG Network. (n.d.). *NGINX Gateway Fabric implementations*. Retrieved August 11, 2025, from https://gateway-api.sigs.k8s.io/implementations/#nginx-gateway-fabric

[108] Kubernetes. (n.d.). *Define container environment variables using secret data*. In *Kubernetes documentation*. Retrieved August 11, 2025, from https://kubernetes.io/docs/tasks/inject-data-application/distribute-credentials-secure/#define-container-environment-variables-using-secret-data

[109] Kubernetes. (n.d.). *Port-forward: Access your application in the cluster*. In *Kubernetes documentation*. Retrieved August 12, 2025, from https://kubernetes.io/docs/tasks/access-application-cluster/port-forward-access-application-cluster/

[110] Kubernetes. (n.d.). *Workloads*. In *Kubernetes documentation*. Retrieved October 19, 2025, from https://kubernetes.io/docs/concepts/workloads/

[111] Kubernetes. (n.d.). *Service*. In *Kubernetes documentation*. Retrieved August 12, 2025, from https://kubernetes.io/docs/concepts/services-networking/service/

[112] FiloSottile. (n.d.). *mkcert: A simple zero-config tool to make locally trusted development certificates* [Software repository]. GitHub. Retrieved August 12, 2025, from https://github.com/FiloSottile/mkcert

[113] Kubernetes Network. (n.d.). *TLS configuration*. In *Gateway API guides – Kubernetes Gateway API*. Retrieved August 12, 2025, from https://gateway-api.sigs.k8s.io/guides/tls/

[114] Kubernetes. (2024, November 19). *Secrets*. In *Kubernetes documentation*. Retrieved August 12, 2025, from https://kubernetes.io/docs/concepts/configuration/secret/

[115] Kubernetes. (2024, October 20). *kube-proxy (optional)*. In *Cluster Architecture* (Kubernetes documentation). Retrieved August 13, 2025, from https://kubernetes.io/docs/concepts/architecture/#kube-proxy

[116] Kubernetes. (2024, October 20). Network plugins. In Cluster Architecture (Kubernetes documentation). Retrieved August 13, 2025, from https://kubernetes.io/docs/concepts/architecture/#network-plugins

[117] Kubernetes. (2024, October 20). cloud-controller-manager. In Cluster Architecture (Kubernetes documentation). Retrieved August 13, 2025, from https://kubernetes.io/docs/concepts/architecture/#cloud-controller-manager

[118] MathWorks. (n.d.). *Parallel computing with MATLAB and Simulink*. MathWorks Solutions. Retrieved August 13, 2025, from https://ch.mathworks.com/solutions/parallel-computing.html

[119] AnyLogic. (n.d.). *AnyLogic simulation software tools & solutions*. Retrieved August 13, 2025, from https://www.anylogic.com/

[120] German Aerospace Center (DLR). (2024, October 14). *Basic definition*. In *SUMO documentation*. Retrieved August 12, 2025, from https://sumo.dlr.de/docs/Simulation/Basic_Definition.html

[121] Grafana Labs. (n.d.). *k6 documentation*. Retrieved August 12, 2025, from https://grafana.com/docs/k6/latest/

[122] Docker. (n.d.). *Get Docker Desktop for Windows*. In *Docker documentation*. Retrieved October 6, 2025, from https://docs.docker.com/desktop/setup/install/windows-install/

[123] Microsoft. (n.d.). *Windows Subsystem for Linux (WSL) overview*. In *Microsoft Docs*. Retrieved October 6, 2025, from https://learn.microsoft.com/en-us/windows/wsl/about

[124] Microsoft. (n.d.). *Install Hyper-V*. In *Microsoft Docs*. Retrieved October 6, 2025, from https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/get-started/install-hyper-v?tabs=powershell&pivots=windows

[125] Microsoft. (n.d.). *Hyper-V overview*. In *Microsoft Docs*. Retrieved October 6, 2025, from https://learn.microsoft.com/en-us/windows-server/virtualization/hyper-v/overview

[126] Microsoft. (n.d.). *Windows Package Manager overview*. In *Microsoft Docs*. Retrieved October 9, 2025, from https://learn.microsoft.com/en-us/windows/package-manager/

[127] Chocolatey. (n.d.). *Getting started*. In *Chocolatey documentation*. Retrieved October 9, 2025, from https://docs.chocolatey.org/en-us/getting-started/

[128] Kubernetes. (n.d.). *PodTemplateSpec (v1 Core)*. In *Kubernetes API reference (v1.30)*. Retrieved October 9, 2025, from https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.30/#podtemplatespec-v1-core

[129] Chingono, A. (2021, April 21). *Restore database on container start up*. Retrieved November 21, 2025, from https://www.chingono.com/blog/2021/04/21/restore-database-on-container-start-up/

[130] DataCamp. (2025, May 20). *Docker ENTRYPOINT explained: Usage, syntax & best practices*. Retrieved November 21, 2025, from https://www.datacamp.com/tutorial/docker-entrypoint

[131] GitHub. (n.d.). Cloning a repository. In GitHub Docs. Retrieved August 13, 2025, from https://docs.github.com/en/repositories/creating-and-managing-repositories/cloning-a-repository?tool=webui