Politechnika Łódzka
Wydział Elektrotechniki Elektroniki Informatyki i Automatyki

Cyprian Szczepański
235988

PRACA DYPLOMOWA
~~magisterska~~/inżynierska/~~licencjacka~~
na kierunku …………Informatyka……………………………

Distributed system to promote student activities

Katedra Mikroelektroniki i Technik Informatycznych
………
(nazwa instytutu/katedry)

Promotor: dr inż. Przemysław Sękalski …………………………………………………………………………………
(*tytuł/stopień naukowy, imię i nazwisko*)

Opiekun pomocniczy*) ……………………………………………………………………………………………………
(*tytuł/stopień naukowy, imię i nazwisko*)

Promotor uczelni partnerskiej)** ………………………………………………………………………………………
(*tytuł/stopień naukowy, imię i nazwisko*)

ŁÓDŹ 2024

* jeśli został powołany
** w przypadku procedury uznania

Abstract

The primary purpose of this thesis is to create a distributed system to promote and settle activities by students in an academy. This application allows students to show their abilities during their studies through academic activity, which the academy might reward them or be evidence of gaining additional knowledge for their careers. The project uses blockchain technology by showing new information technology solutions in areas like verifying students' abilities and securing the data for gaining competencies. Still, the project also creates an internal nodes on the network only seen by people related to the academy.

Streszczenie pracy

Głównym celem tej pracy dyplomowej jest stworzenie rozproszonego systemu do promowania i rozliczania działań przez studentów na uczelni. Dzięki aplikacji, studenci są w stanie wykazać się zdolnościami poznanymi podczas trwania swojej nauki poprzez aktywność akademicką, która może być wynagrodzona przez uczelnię lub też może stanowić dowód nabytych dodatkowych umiejętności dla swojej kariery zawodowej. Projekt wykorzystuje technologie blockchain w celu przedstawienia nowych rozwiązań informatycznych w obszarze weryfikacji umiejętności studentów oraz bezpieczeństwa danych nabywanych kompetencji. Mimo to w ramach projektu tworzone są również wewnętrzne węzły w sieci, które widzą tylko osoby powiązane z uczelnią.

Keywords

Blockchain, Microservices, CI/CD, Golang, NextJS

Słowa kluczowe

Blockchain, Mikroserwisy, Ciągła integracja/Ciągłe dostarczanie, Golang, NextJS

Table of Contents

List of Figures

List of Listings

1 Introduction

1.1 Problem definition

In today's reality, more and more people are committing to studying at universities. The purpose of this trend is so that people can have better possibilities in their future work; employees need academic education. Within this fact, there are two main questions:

- Is the academy diploma, course certificate, and other activities an applicant provides to an employee not defrauded?
- Can the faculty program fulfill employees' requirements to recruit a student?

In the first question, people need to be convinced to trust the description in the CV fully; they need proof, but it is hard to create a document with appropriate approval for every student's activity; it can take too much time and effort for academy workers.

Universities host diverse academic activities, from traditional coursework to research projects, internships, and extracurricular involvement. Documenting these activities is demanding due to the myriad forms they can take. Hence, the next problem is standardized documentation for academic achievements, and even individual professors may have distinct methods of recording and certifying student accomplishments.

Students might be tempted to mispresent their achievements because of the lack of foolproof documentation university can create from their system.

To overcome these challenges, a solution must streamline the verification process and ensure its efficiency and accuracy. Technology-driven solutions and standardized frameworks can alleviate the administrative burden, enhance reliability, and contribute to a more transparent and trustworthy verification system for academic credentials.

In the second question, the reality of today's market is changing more rapidly than the program of each university faculty. Also, academies provide a basis for each lecture that a student can develop on his/her own in the future. Employees want specialists, so naturally, students should dive into subjects that are in demand. What has stayed the same is that employees always look forward to students, voluntary work, and working with people on something that could solve specific problems. Academies are open for students to help them advance their soft skills, but this community rarely takes up this activity. Thanks to investigators from managements studies at the University of Technology in Lodz, who surveyed people currently studying, we can notice why students do not accept this action; most are unaware.

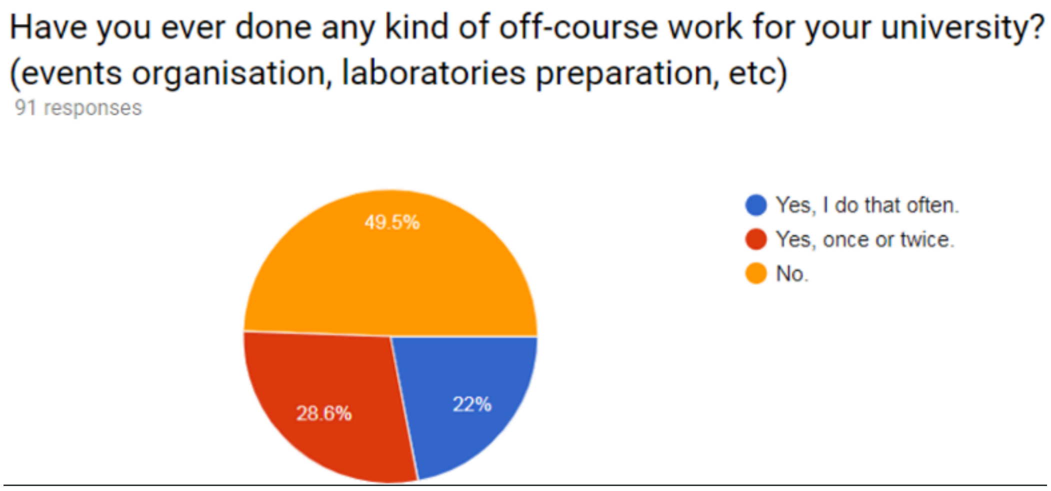


Figure 1. How many students work for the university according to the survey

In the following survey question, we can better understand why students do not contribute to academic work. This question can be divided into three parts:

- Students are focused on other activities.
- Students need more motivation to do voluntary work.
- There needs to be more information for people who could take additional work for the academy.

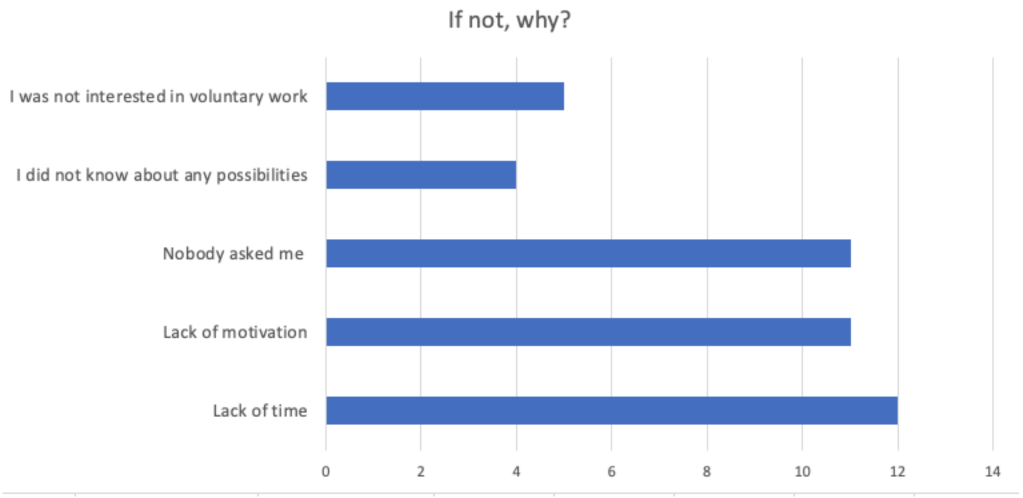


Figure 2. Reasons why students do not work for the university

The challenge for solving this issue could be creating a platform that enables everyone in the university, for students who can see all possible activities in the tutor's projects and the other activities, but also for academy workers where they can create tasks for students.

Moreover, this platform needs gamification methods for students to create better perspective for themselves. For example, students can get points and rewards for completing a new activity that ends successfully and is accepted by the tutor.

Unfortunately, there are no suitable sources to show all student's trustworthy activities and motivate this community to consider another type of experience employees seek during recruitment. This paper will introduce a solution that can solve these two questions provided at the start of this section.

1.2 Objectives

The following outlines the objectives of this thesis that will, step by step, make a move for solving the placement in the problem definition section.

The first objective is to develop entirely scalable backend services for this product. We want our services to continuously work on the server regardless of service updates and scale our services when there is significant usage. This aspect could be resolved using microservice architecture. The provided architectural pattern is an approach for creating loosely coupled, fine-grained services that communicate using lightweight protocols like REST API.

The secondary objective responds to the challenges posed by the problem definition; the proposed solution is a system that addresses academic verification student activity recording and introduces a token-based incentive mechanism. We can divide the solution into three parts:

1. Implementing a blockchain system using Ethereum and ERC-20 standards for immutable and decentralized verification. The framework involves recording academic achievements securely on the blockchain, providing transparent and tamper-resistant proof of qualifications.
2. Extending the blockchain framework to capture diverse student activities, such as voluntary work, collaborative projects, and academy contributions. The system ensures transparent accountability and enables a broad-based view of a student's capabilities.
3. Introducing a token-based incentive system where students earn tokens for their various achievements. While devoid of external value, these tokens can be exchanged within the application for rewards. The framework fosters well-rounded development and complements academic qualifications by showcasing a student's proactive involvement.

Also, a third objective will handle the user interface for pointed challenges. We will achieve a fully responsive, user-friendly web application connected with blockchain environments.

1.3 Structure

This thesis will be structured as follows. Chapter 2 will provide a theoretical discussion of blockchain aspects of the project that will be covered after this chapter. They will appear with a comprehensive explanation of Ethereum and Substrate as potential solutions to utilize blockchain in the project. Also, how we use blockchain to store data and essential information about the ERC20 protocol.

Chapter 3 is the central chapter and shows an innovative approach to solving this problem provided in the Introduction. The first part shows the abstraction of the solution, like the specification of requirements, and architecture of the solution. The second part focuses on presenting backend services in action and client-side applications.

Chapter 4 will evaluate the tools needed to distribute a fully working product. It focuses on continuous integration and continuous development (CI/CD) that I use to maintain my solution.

Chapter 5 will conclude this work by discussing the results and the relevance of the contribution of the problem provided in the Introduction.

2 Fundamentals

In this chapter, we will explain the essential aspects of blockchain that are used in creating this project.

2.1 Blockchain basics

Blockchain is a decentralized, distributed ledger technology that securely records transactions across multiple computers, making data immutable and transparent. Blockchain is ideal for delivering information because it provides immediate, shared, and utterly transparent information stored on a immutable ledger that can be accessed only by permissioned network members. We can share a single view of the truth; you can see all the details of a transaction end to end; this gives confidence that we deliver, not faucet value. (1)

Blockchain is not only changing the face of finance but also significantly impacting a wide range of sectors such as healthcare, logistics, education, and many others. Creating data registries that do not require the trust of intermediaries opens the door to new business models, streamlined processes, and increased transparency and trust.

We highlight a few key concepts to understand the next sub-chapters better.

2.1.1 Decentralization

There is no central authority controlling the blockchain system. Instead, transactions are verified and recorded across a distributed network of nodes. These nodes are individual computers that communicate with each other through a peer-to-peer (P2P) network.

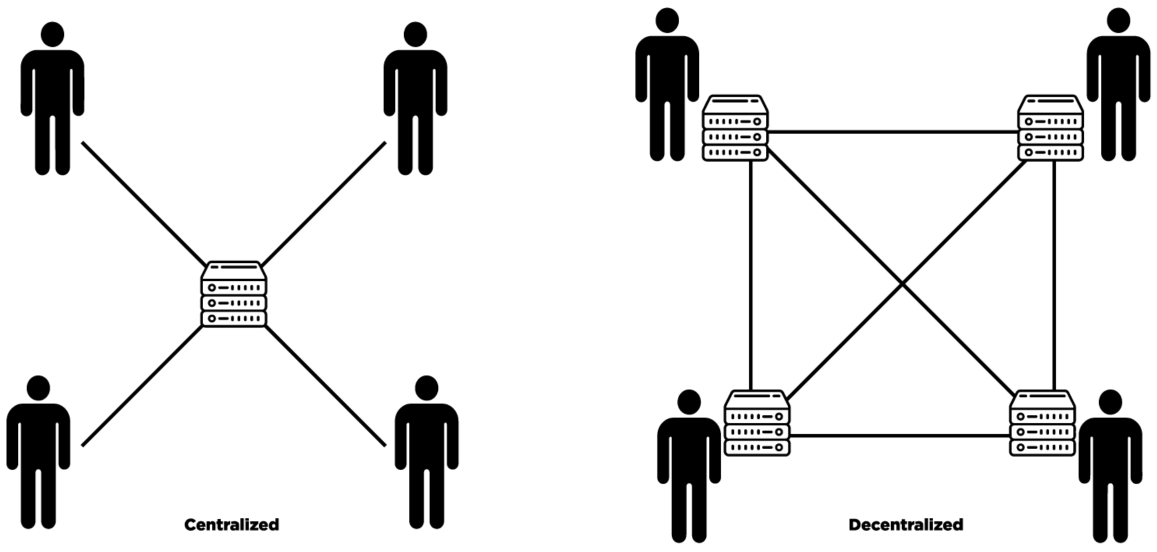


Figure 3. Centralized vs decentralized network

2.1.2 Consensus

Consensus mechanisms are integral to blockchain networks, ensuring agreement among participants regarding the validity of transactions. Blockchain achieves consensus without intermediaries. Through consensus algorithms such as Proof of Work (PoW) or Proof of Stake (PoS). The primary purpose of consensus algorithms is to secure the blockchain while keeping control over the system decentralized and diffused across as many participants as possible. (2)

Proof of Work works on the fact that reward of newly minted currency is an incentive to those who contribute to the system's security; this energy cost is required to participate in mining. Proof of Stake works on the risk of validators' cryptocurrency, meaning that validators take turns proposing and voting for the next valid block with the amount of holding the associated cryptocurrency. Validators lose their deposit of cryptocurrencies if most validators reject the block they staked it on. (2)

2.1.3 Tokens

Tokens represent digital assets or utility within a blockchain ecosystem. These tokens can represent cryptocurrencies, assets, or rights and are stored and transferred securely on the blockchain. (2)

For now, in this work, we define two types of tokens: fungible and non-fungible. The first one is divisible and non-unique assets like 1 BTC (cryptocurrency used in Bitcoin), which have the same value everywhere, no matter where it is used. Non-fungible tokens (NFT) represents unique assets like digital content.

2.1.4 Smart Contracts

Smart contracts are self-executing contracts with predefined conditions written in code. These contracts automatically enforce and execute terms when predefined conditions are met, facilitating trustless and tamper-proof agreements. (2)

2.2 What is Polkadot?

Polkadot is a multi-chain blockchain platform designed to facilitate interoperability between different blockchains. It was created by Dr. Gavin Wood, one of the co-founders of Ethereum (for now, one of the most popular blockchains). It is often called a multi-chain; this means that unlike previous blockchain implementations, which have focused on providing a single chain like Bitcoin or Ethereum, Polkadot itself provides the bedrock for a large number of dynamic data structures, so-called parachains, which means we can set in parallel different independent chains like Bitcoin or Ehtereum. (3)

Compared with Ethereum, the benefit of choosing Polkadot is that we do not limit our need to specific hosts that can run, for example, smart contracts. In other words, we can run EVM, which is the environment of Ethereum for smart contracts on any parachains on Polkadot. (4)

2.3 What is Substrate?

Substrate is a Software Development Kit that allows building applications that use blockchain with Polkadot ecosystem. This solution gives us control over creating predefined application logic that is usually too complex to create everything at zero. The modules that are predefined are called pallets. Substrate has pallets required in modern blockchains like staking, fungible tokens, or non-fungible tokens. One of these pallets is ready-to-use for smart contracts called pallet contracts, which will be used in this projects. (5)

Substrate is a composable and adaptable tool designed to be upgradeable for our needs if we need different network stacks, transaction formats, etc., that are not composable in a single classic blockchain solution. Removing limits for developers allows us to create more suited products for our needs; that is why we consider Polkadot with Substrate more innovative in the blockchain sector.

2.4 ERC-20

The ERC20 standard (Ethereum Request for Comments) is a standard for fungible tokens. It defines a common interface for contracts implementing a token.

```
    contract ERC20 {
        function totalSupply() constant returns (uint theTotalSupply);
        function balanceOf(address _owner) constant returns (uint balance);
        function transfer(address _to, uint _value) returns (bool success);
        function transferFrom(address _from, address _to, uint _value) returns
           (bool success);
        function approve(address _spender, uint _value) returns (bool success);
        function allowance(address _owner, address _spender) constant returns
           (uint remaining);
        event Transfer(address indexed _from, address indexed _to, uint _value);
        event  Approval(address  indexed  _owner,  address  indexed  _spender,  uint
_value);
     }
```

Listing 1. The ERC20 interface defined in Solidity

An ERC20-compliant token contract must provide at least the following functions and events:

- totalSupply: Returns the total units of token that currently exist.
- balanceOf: Returns the balance of tokens that currently have the owner.
- transfer: Transfer specified number of tokens to specified address from specified address.
- transferFrom: Delegate transferring a specified number of tokens from a specified address.
- approve: Allows spender to withdraw specified number of tokens from specified account.
- allowance: Returns tokens from spender to the owner.
- Transfer: Trigger when a transfer is successful.
- Approval: Log an approved an event.

ERC20 allows for two different workflows. The first is a single-transaction, straightforward workflow using the transfer function. Wallets use this workflow to send tokens to other wallets. The second workflow is a two transaction workflow that uses approve followed by transferFrom. This workflow allows a token owner to delegate their control to another address. It is often used to delegate control to a contract to distribute tokens. (2)

3 Implementation

This chapter shows how to solve the problem, which was placed in the Introduction.

3.1 Requirements Analysis

3.1.1 Functional Requirements

Figure 3 omits login and register a user and updates personal data use cases for simplicity. Nowadays, these features are considered foundational functionality. This diagram illustrates the most crucial functionalities for each actor, like admin, teacher, and student. Omitted functionalities will be exposed in chapter 3.5 Frontend side solution.

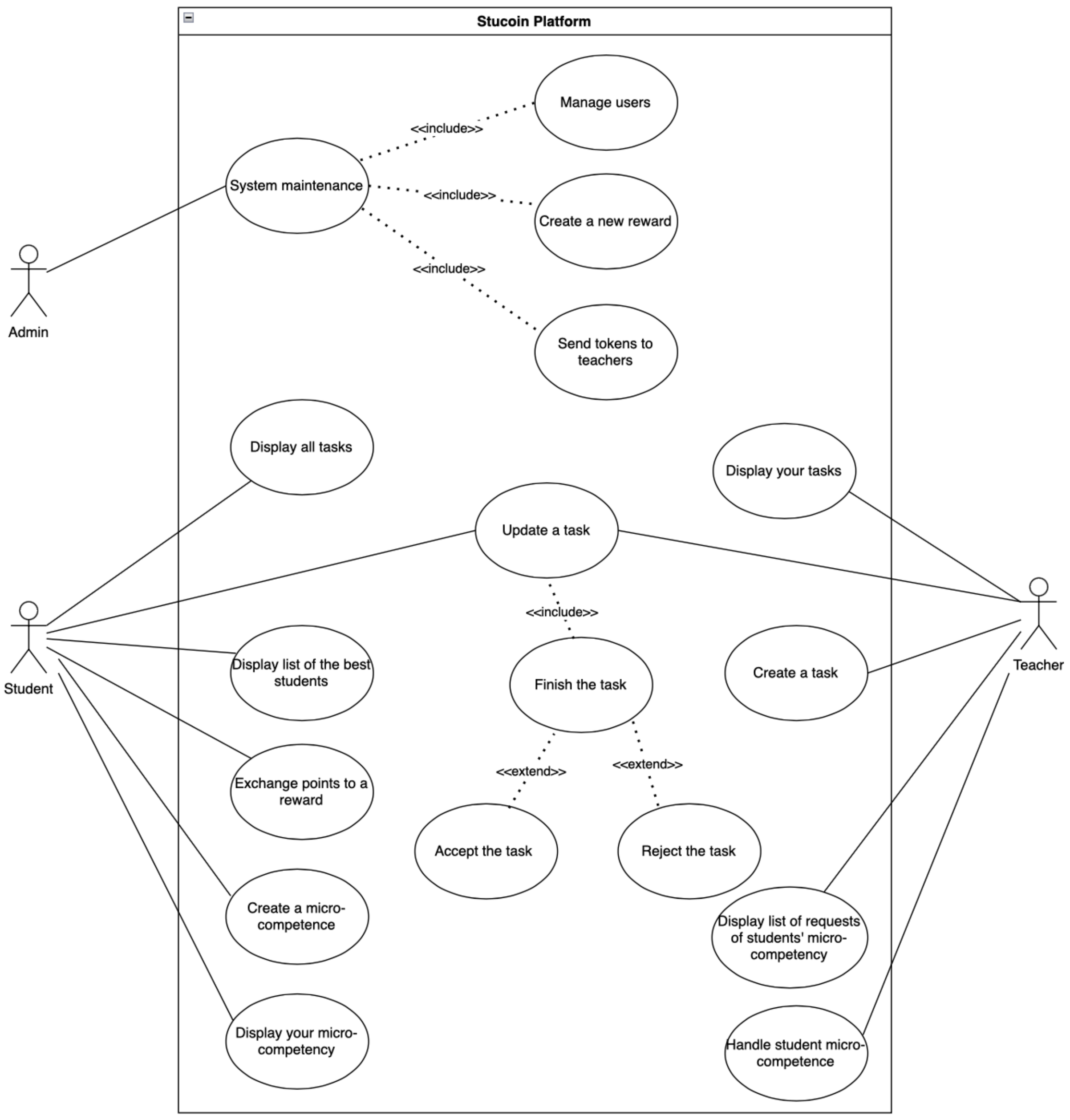


Figure 4. Requirements analysis with use cases

3.1.1.1 System maintenance

The System maintenance module encompasses three primary functionalities: Manage users, Create a new reward, and Send tokens to teachers.

1. Manage users
    a. Main case
        i. Administrators can manage user accounts, including creating new accounts, updating existing accounts, and deactivating or deleting accounts.
        ii. Creation involves specifying necessary details likes email, password, and user role.
        iii. Updating allows modification of user information such as email, password, or profile details such as name of university or faculty.
    b. Alternative cases
        i. Required fields for creating or updating user accounts must not be blank or contain invalid data.
        ii. Warning prompts for deactivating or deleting accounts with associated content or transactions.
    c. Initial condition
        i. The user must log in to the platform.
        ii. The user acting must have administrative privileges.
2. Create a new reward
    a. Main case
        i. Administrators can create new rewards based on specified criteria (e.g., name, description, amount of points student exchange).
        ii. The created reward becomes available for students on page Rewards.
    b. Alternative cases
        i. Required fields for creating a reward must not be blank or contain invalid data.
        ii. Warning prompts for creating rewards with conflicting criteria or conditions.
    c. Initial condition
        i. The user must log in to the platform.
        ii. The user acting must have administrative privileges.
3. Send tokens to teachers
    a. Main case
        i. Administrators can send tokens to teachers' wallets that they specify in their data on the Stucoin Platform.
        ii. Administrators set the teacher(s) token amount. The university should specify how many tokens teachers can earn in faculty or departments.
    b. Alternative cases
        i. If the administrator lacks sufficient tokens, the system prevents the transaction and notifies the administrator.
        ii. If the recipient teacher(s) are inactive or non-existent, the system prompts the administrator to select the valid teacher(s).

   c. Initial condition
      i. The user must log in to the platform.
      ii. The user acting must have administrative privileges.
      iii. The admin has a connected wallet to the plaform.

3.1.1.2 Display all tasks

The Display all tasks module shows students' tasks on the Stucoin Platform. Tasks have various statuses that inform students that they are able to pick, are reserved, or have ended.

1. Main case
   a. Upon accessing the designated page, the system shall present the student with a comprehensive list of tasks.
   b. Each task entry in the list shall include essential details such as the task title, description, email of the teacher, and task status.
   c. The task list shall be paginated, displaying a predefined number or tasks per page.
2. Alternative cases
   a. If there are no tasks, the system shall display a message indicating no available tasks.
   b. The system shall support searching and filtering options to enable students to organize and customize the displayed task list according to their preferences (e.g., searching by task title, filtering by task status).
   c. Students can view all details of a particular task by choosing a specified task.
   d. Students can filter tasks they are assigned by themselves.
3. Initial condition
   a. Students must log in to access the functionality.
   b. The user must have student role to view all tasks.

3.1.1.3 Display your tasks

The Display your tasks module shows teachers' tasks on the Stucoin Platform. Tasks have various statuses informing teachers that they can pick; students pick their task or are waiting to accept or reject it.

1. Main case
   a. Upon accessing the designated page, the system shall present the teacher with a comprehensive list of teacher tasks.
   b. Each task entry in the list shall include essential details such as the task title, description, email of the teacher, and task status.
   c. The task list shall be paginated, displaying a predefined number of tasks per page.
2. Alternative cases
   a. If there are no tasks, the system shall display a message indicating that the teacher has no tasks and encourage them to create one.
   b. The system shall support searching and filtering options to enable teachers to organize and customize the displayed task list according to their preferences (e.g., searching by task title, filtering by task status).

    c. Teachers can view all details of a particular task by choosing a specified task.
3. Initial condition
    a. A teacher must log in to access the functionality.
    b. The user must have the teacher's role to view created tasks.

3.1.1.4 Create a task

The create a task module is dedicated to teacher functionality where the teacher can write content and send appropriate files that are included to achieve the task by the student.

1. Main case
    a. The system shall provide a feature for teacher to create tasks or assignments for their students.
    b. Upon accessing the designated page, teachers can initiate the task-creation process.
    c. The system provides initial data to the teacher; for example, the title and the things said immediately are sent the task to the database.
    d. Teachers shall specify essential details for the task, including title, description, and send files such as documents, reports, and multimedia files to enhance task comprehension.
    e. The system enables constant updating. The task is during task creation, so the teacher can come back later to add additional information to this task.
    f. The teacher must specify how many tokens students can earn by completing it.
2. Alternative cases
    a. If a title or amount of points is left blank during task creation, the system shall prevent this by adding an example value to each field.
    b. The system rejects creating another task if the teacher has no minimum tokens specified by the system.
    c. Suppose the teacher needs more tokens set during task creation. The system will change it to the lowest that can be drawn from the teacher's wallet.
3. Initial condition
    a. The user must be logged in to the system with a teacher role to create tasks.
    b. The teacher has a wallet connected to the system.

3.1.1.5 Update a task

The update a task module is a combined responsibility of the teacher with student. It is an extended Create a task module, where the teacher can add new content, but the student can only update the task status and add files.

1. Primary actor: student
2. Secondary actor: teacher
3. Main case
    a. Teacher update task case
        i. The teacher selects the task to be updated.

        ii. The teacher modified task details such as title, description, or available files. The teacher can delete student's files.
        iii. If the student sets the task status to completed, the teacher can confirm it or reject it.
    b. Student update task case
        i. The student accesses the task list.
        ii. Students can assign to the specified task.
        iii. Student can add files to the task.
        iv. Students may finish the task by setting the status of the finished task.
        v. After finishing it, the student has no access to update the task.
4. Extensions
    a. Accept the task
        i. If the task requires acceptance by the teacher, the system will display appropriate components that the teacher can choose to finish the task.
        ii. If the teacher admits that the student finishes the task correctly, the teacher sets the task status as accepted.
        iii. Student earns points when a teacher sets the task's status by acceptance.
    b. Reject the task
        i. If the task requires acceptance by the teacher, the system will display appropriate components that the teacher can choose to finish the task.
        ii. If the teacher admits that the student failed the task, the teacher sets the status as rejected.
        iii. Students do not get any points.
5. Initial condition
    a. The task must exist in the system.
    b. The teacher and the student must have a connected wallet to the platform.
    c. The teacher and the student have an appropriate role in accessing and updating the task.

3.1.1.6 Display list of the best students

The display list of the best students module shows the best students by the total points they gained during their studies.

1. Main case
    a. The system shall provide a feature to generate and display a ranking of best-performing students based on predefined criteria.
    b. The ranking shall consider the total points students achieved by finishing tasks.
    c. The ranking should display students sorted by the total points descending.
    d. Each student entry in the list shall include essential details such as student name, surname, email, and total points.
2. Alternative cases

    a. If no students meet the predefined criteria for inclusion in the ranking, the system shall display a message indicating that the ranking is currently unavailable.
    b. The system shall support customization of ranking criteria by sorting students ascending or displaying current user places on the ranking with students near them on the list by the total points.
3. Initial condition
    a. Users must log in to the platform.

3.1.1.7 Exchange points to reward

The exchange points to a reward is a module where students can spend the earned tokens by finishing tasks.

1. Main case
    a. The student accesses the rewards page of the system.
    b. The student selects the desired reward for which they wish to exchange points.
    c. The system checks the student's balance to ensure sufficient points for the selected reward.
    d. If the students has enough points, proceed and send the reward by the student's email.
    e. The system updates the student's wallet by drawing several tokens.
2. Alternative case
    a. If the student does not have enough points to exchange for the selected reward, the system informs the student and prompts them to accumulate more points or choose a different reward.
3. Initial conditions
    a. The user must have a student role to process.
    b. Students must have a connected wallet to the platform.
    c. The user must log in to the platform.

3.1.1.8 Create a micro-competence

Create a micro-competence is a module where students can specify the knowledge they achieved during their studies, and it is not for passing subjects.

1. Main case
    a. The system shall provide a feature for students to create their micro-competencies.
    b. A student can initiate the micro-competence creation process upon accessing the designated page.
    c. Students shall specify essential details for the micro-competence, including title and description, and send files such as documents, reports, and multimedia files enhance micro-competence comprehension.
    d. Students must set a supervisor, a teacher who verifies their knowledge.
    e. At the end, students should confirm that they have completed creating micro-competencies.
    f. The system saves the data in the database and displays a new entry on the micro-competency teacher board.

2. Alternative cases
    a. During micro-competence creation, the student left. The system will not save the data.
    b. If some fields are left blank, the system rejects to confirm a new micro-competence.
3. Initial condition
    a. Students must be logged in on the platform.
    b. The user must have a student's role to perform this activity.

3.1.1.9 Display your micro-competency

Display your micro-competency shows students their micro-competency, which is accepted or rejected by the supervisor, which is a teacher.

1. Main case
    a. The system shall provide a feature to generate and display a list of all student micro-competencies. The system will encourage them to create one.
    b. Each micro-competency entry in the list shall include essential details such as the title, description, email of the teacher, and status that could be accepted, rejected, or not verified.
    c. The micro-competencies list shall be paginated, displaying a predefined number of tasks per page.
2. Alternative cases
    a. If there are no student micro-competencies, the system shall display a message indicating no available micro-competencies.
    b. The system shall support searching and filtering options to enable students to organize and customize the displayed micro-competencies list according to their preferences (e.g., searching by title, filtering by status).
    c. Students can view all details of a particular micro-competence by choosing a specified list entry.
3. Initial condition
    a. Students must log in to access the functionality.
    b. The user must have student permission to view all tasks.

3.1.1.10 Display list of requests of students' micro-competence

Display list of requests of students' micro-competence module is dedicated only to teachers where the teacher can see upcoming new micro-competencies that arrived to verify.

1. Main case
    a. The system shall provide a feature to generate and display a list of all students micro-competencies that they assigned their micro-competencies to the teacher to verify.
    b. Each micro-competency entry in the list shall include essential details such as the title, description, email of the teacher, and status that could be accepted, rejected, or not verified.

   c. The micro-competencies list shall be paginated, displaying a predefined number of tasks per page.
2. Alternative cases
   a. If there are no student micro-competencies, the system shall display a message indicating no available micro-competencies to verify.
   b. The system shall support searching and filtering options to enable students to organize and customize the displayed micro-competencies list according to their preferences (e.g., searching by title, filtering by status).
   c. Teachers can view all details of a particular micro-competence by choosing a specified list entry.
3. Initial condition
   a. Teacher must be logged in to the system with a teacher role to create tasks.
   b. The user must have a teacher role to perform this activity.

3.1.2 Non-functional Requirements

- The system should have an intuitive and user-friendly interface accessible to users of all roles.
- The system should provide timely responses to user interactions, with a maximum acceptable delay of 2 seconds for page loading.
- Error messages should be clear and informative.
- The system should be available for use all the time.
- The system should gracefully handle errors and failures, ensuring minimal disruption to user activities.
- The system should scale seamlessly to accommodate increasing numbers of users, tasks, and rewards without significant degradation in performance.
- The system should implement secure authentication mechanisms (e.g., password hashing, multi-factor authentication) to verify the identity of users.
- Access to system features and data should be restricted based on user roles.
- The system should be responsive and function properly across various devices and screen sizes.

3.2 Architecture

The primary objective of this chapter is to provide a comprehensive overview of the products involved in the Stucoin Platform. Also, there is an overview of microservice architecture, including its benefits and challenges. Also, all technologies used to create each product will be listed. In the following chapters, we will extend this knowledge of why these solutions were used.

The final application is divided into two products: Stucoin-frontend and Stucoin-backend. We will look at each in the upcoming chapters, but for now, some core concepts of each will be provided.

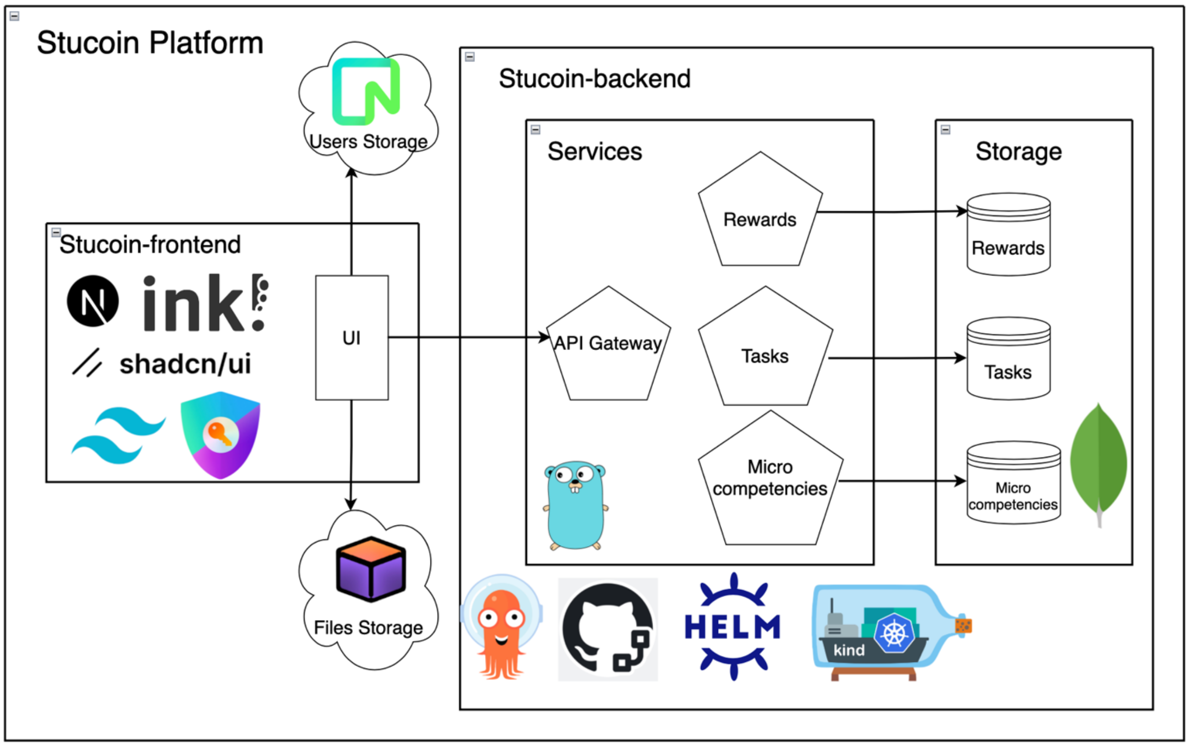


Figure 5. Architecture of the application divided by products

Our Stucoin-backend solution follows the microservices pattern. That means the server is decomposed into a set of loosely coupled and independently deployable services, each responsible for the particular business logic of the application. We can quickly scale our services depending on the usage by users, thanks to isolating each service from the rest. Also, this fact is essential in two aspects:

- This modular approach enables independent development of each service or adding a new one without redeploying every piece of the product.
- If some service fails, this does not cascade into the rest of the services. So we can quickly find bugs and repair service with the working system; what is essential is that we can roll back to the last version of the not working service so the users do not see the difference. This will be explained in detail in Chapter 3.4 Backend side solution.

By providing critical aspects of why we choose microservice architecture, we are now aware of why we have yet to decide on the classic approach, which is the monolithic approach. In the monolithic architecture, for example, if the service fails, everything stops working, which means we must stop the application to find and fix bug; nowadays, this situation is unacceptable because we lose our users because of dissatisfaction with our services. (6)

In Figure 4, we can see that some services interact with a dedicated data storage service. This Storage solution serves as a centralized storage solution for all application data. This centralized approach ensures data consistency and integrity across the application, enabling seamless data sharing and access among microservices. This scenario simplified our architecture, and in the future, we could consider moving each database storage to each microservice based on the business logic that could solve it, which could be more isolated from any security issues or errors.

On the other hand, the Stucoin-frontend is responsible for the user interface layer, which presents information the Stucoin-backend handles to users. This product is designed to be modular, responsive, and adaptable to various devices and screen sizes.

As we can see in Figure 4, this part of the Stucoin Platform uses third-party solutions that are external to the boundaries of this project. They are responsible for storing users' data and files; for this work, we are not focused on making solutions for this business logic, so we can adapt our Stucoin-frontend to use professional providers available for our data and store them by additional frameworks. More will be exposed in Chapter 3.5, Frontend side solution; for now, it is essential to note that our Stucoin-frontend is not only connected and handled by the Stucoin-backend server.

At the end of this chapter, all technologies will be listed in this project, which will be explained in detail in the following chapters.

Technologies used in Stucoin Platform:

1. Stucoin-frontend
    - Core
        - NextJS
    - Design
        - Tailwind css
        - Shadcn ui
        - BlockNote JS
    - Data providers
        - Prisma ORM
        - Neon/PostgreSQL
        - Edge Store
    - User authentication
        - AuthJS
    - Blockchain
        - Ink!
2. Stucoin-backend
    - Core
        - Golang
        - Gorilla Mux
    - Storage
        - MongoDB

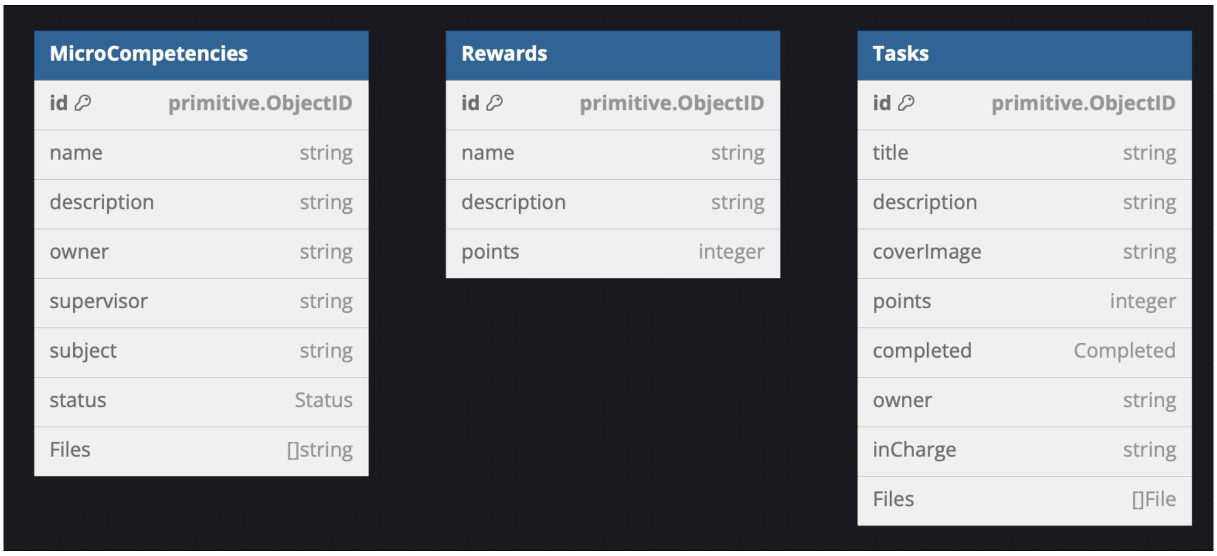
- CI/CD
    - Kubernetes
    - Helm
    - ArgoCD
    - Github Actions
    - Kind

3.3 Databases and providers

In this chapter, we dive into the practical implementation of databases in the Stucoin Platform. We explore how databases have been tailored to suit specific needs and requirements. Moreover, this chapter examines the external providers of databases that have been considered for our projects.

3.3.1 MongoDB

MongoDB was chosen to store data that students and teachers can create, so there are other micro-competencies and tasks. Also, space is shown for storing rewards that students can exchange for points.



MicroCompetencies	
id	primitive.ObjectID
name	string
description	string
owner	string
supervisor	string
subject	string
status	Status
Files	[]string

Rewards	
id	primitive.ObjectID
name	string
description	string
points	integer

Tasks	
id	primitive.ObjectID
title	string
description	string
coverImage	string
points	integer
completed	Completed
owner	string
inCharge	string
Files	[]File

Figure 6. MongoDB collections in Storage service

```
enum Status {
  Waiting
  Accepted
  Rejected
}
```

```
enum Completed {
  Open
  Completed
  InCompleted
  Aborted
  Accepted
}

struct File {
  name string
  path string
  size string
}
```
Listing 2. Structures used in MongoDB

This NoSQL solution was the first choice because of the JSON document type for storing and retrieving data in collections. However, MongoDB also has a great developer user experience, which means a rich query language that offers a wide range of operators and expressions, enabling developers to express complex criteria for retrieving data. (7) Also worth pointing out is that various programming languages have written libraries to handle this database. (8)

3.3.2 Neon – Serverless PostgreSQL

Neon Tech utilizes serverless PostgreSQL primarily for its scalability, cost-efficiency, and what is essential for this project – ease of management. By adopting a serverless approach, Neon Tech handles varying workloads without requiring manual intervention. (9) This allows us to focus on solving two primary questions pointed out in Chapter 1.1, Problem Definition.

Our Stucoin-frontend client is connected to this by Prisma ORM, which helps us model our schemas quickly, connect to Neon, and migrate our schemas by a single command. (10)

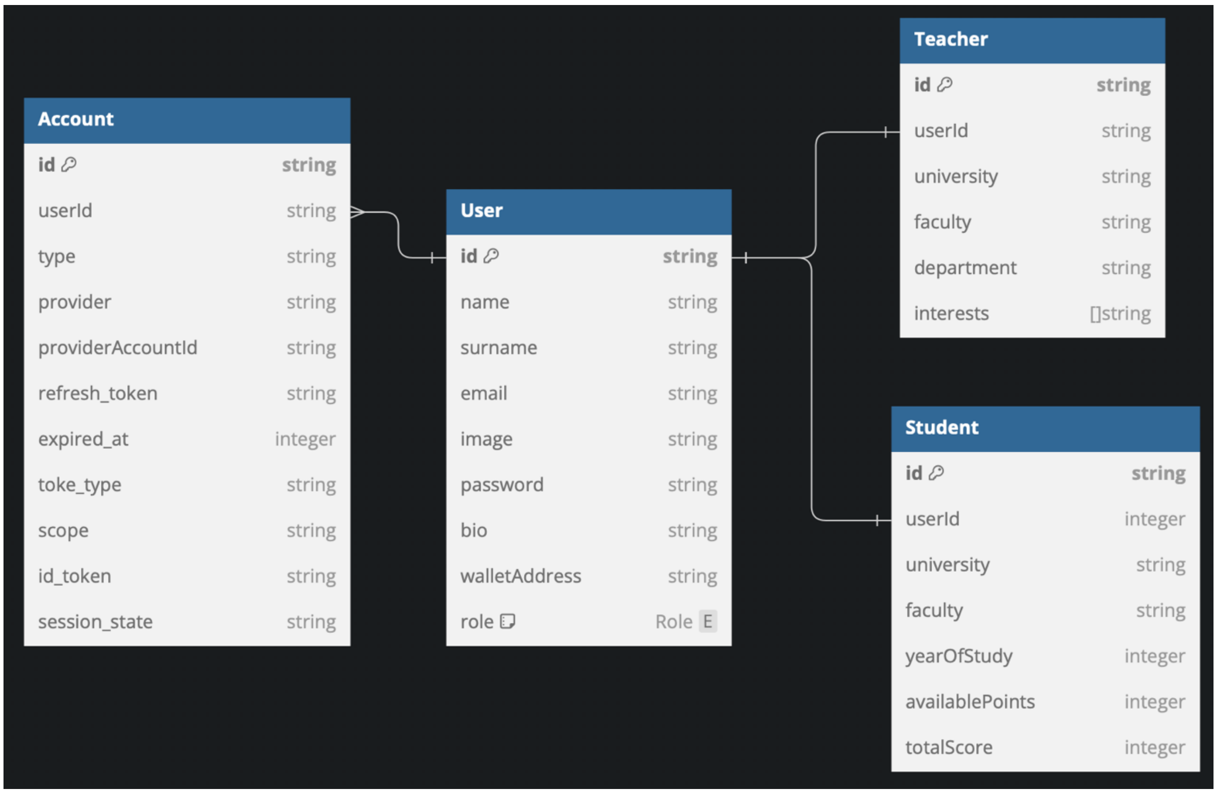


Figure 7. Relation in tables PostgreSQL

```
enum Role {
  Admin
  Student
  Teacher
}
```

Listing 3. Enum Role used in User table

Tables like User, Teacher, and Student store basic information about our users. AuthJS adapters strictly specify the Account table to enable connection to the Stucoin Platform by Google and Github providers. (11) More about AuthJS will be explained in Chapter 3.5, Frontend side solutions.

3.3.3 Edgestore

The second external solution is Edge Store. This product focuses on storing and handling files in cloud storage like AWS, Azure, or simply Edge Store. Like with Neon Tech, choosing this tool enables us to ease the creation of robust components for saving and retrieving files in tasks and micro competencies modules. This simplifies our architecture maintenance in the Stucoin-backend to only storing references to the particular file as a string in the Storage service. (12)

3.4 Backend side solution

This chapter explains how services work in our microservice architecture; we look at the technologies used and their benefits to the project.

First, the connection between the API gateway and each service, which contains separated business logic in two pieces:

- For the process of decoding JSON body from request and encoding response.
- For operating with a connected database.

By creating those layers between the API gateway and the database, we secure data from potential issues like inappropriate actions on a database by a user during an application.

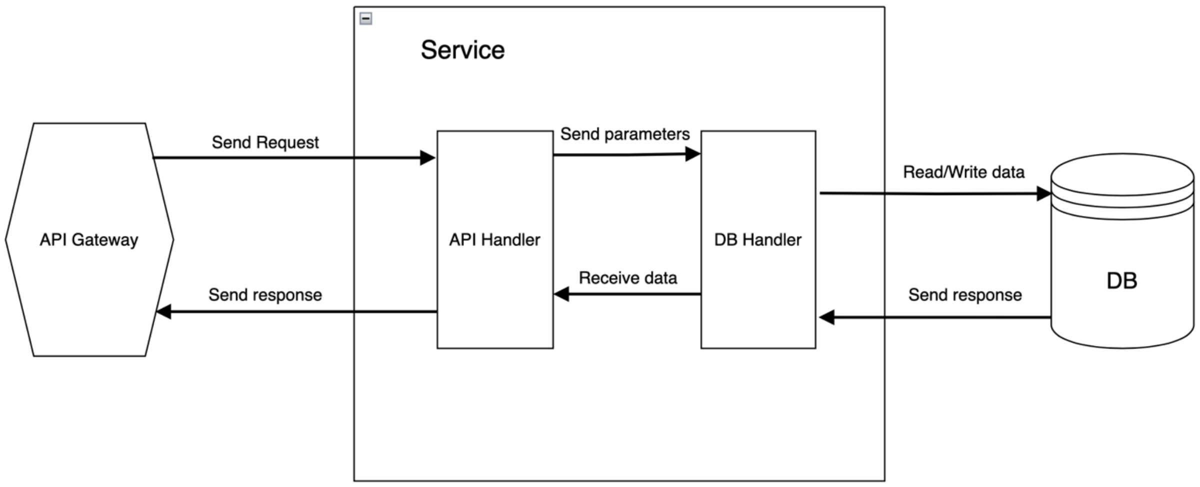


Figure 8. Data flow of example service in Stucoin-backend

The API Gateway service has its counterpart in the codebase as BFF (Backend For Frontend), reflecting our plans for further utilization of the latter and the benefits it brings. As a dedicated service tailored for specific frontend clients, BFF allows for enhanced customization and optimization of backend functionality to serve better the requirements and performance needs of individual frontend applications or interfaces. In the future, it will be considered to increase user experience by writing frontend clients for specific platforms; that is why the BFF pattern is essential; for now, it only works as an API gateway because of only available the web client.

Services connected to databases implement CRUD (Create Read Update Delete); we skip explaining this part because of its simplicity. API gateway is a service responsible for routing requests to the appropriate service. In future development, make this service more secure by implementing authentication and authorization with the role; this aspect is only implemented on the frontend side.

All the services are written in Golang with the help of Gorilla Mux. The main benefits of creating a backend solution in Golang include its efficiency in terms of performance, its built-in concurrency support, and simple and concise syntax facilitating rapid development.

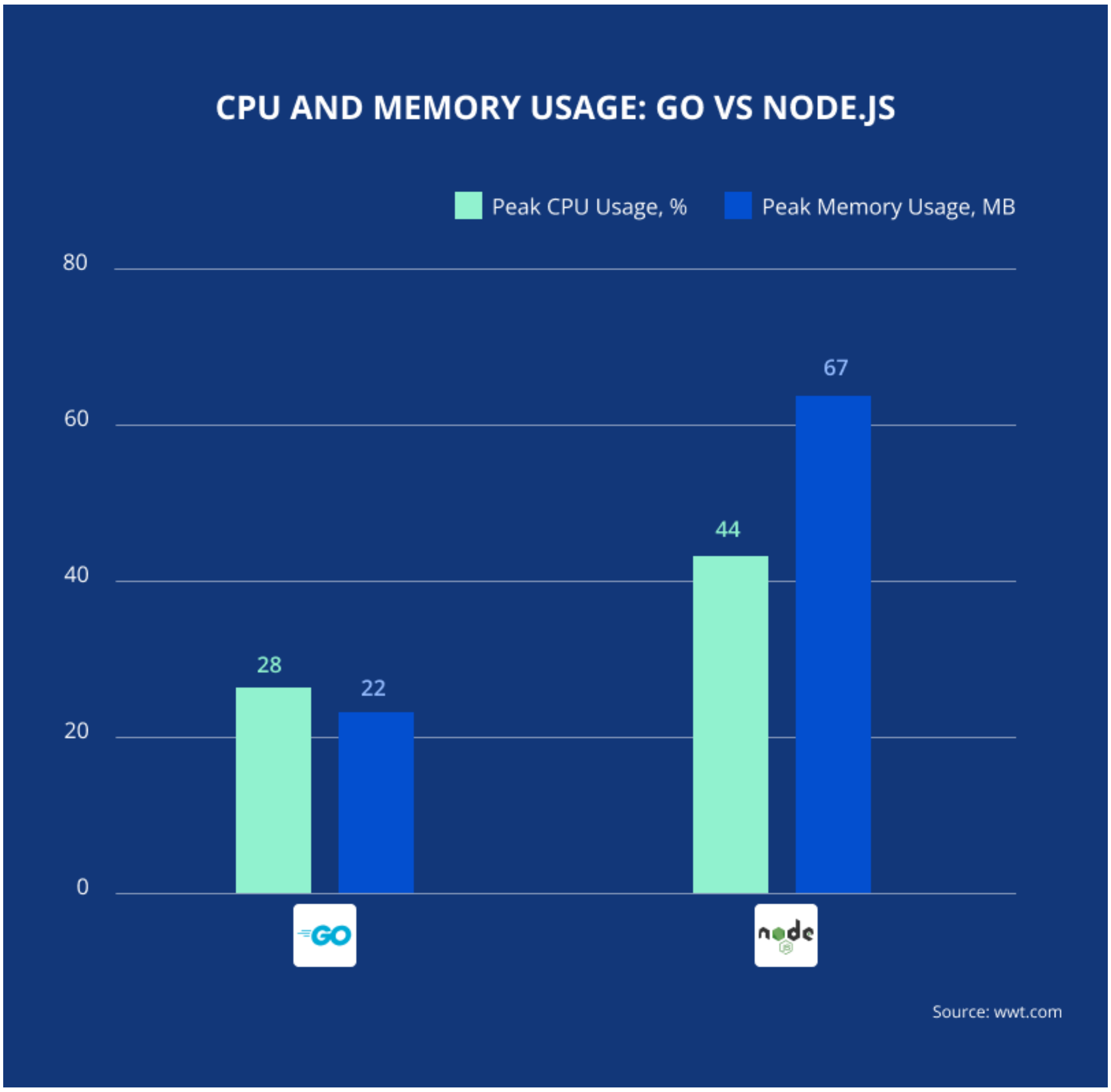


Figure 9. Cpu and memory usage: go vs nodejs

As we can see in Figure 8. Golang outperforms the popular server-side solution, NodeJS, regarding requests per second, CPU load, and memory consumption. (13) We chose Golang to create a robust and fast solution for fetching data and processing requests in our microservice architecture. (14) For handling REST API, Go has a mature, robust standard library, “net/http”, which perfectly fits into our requirements – fast and efficient. We have added the Gorilla Mux package, which implements a requests router and dispatcher to match incoming requests in our services. (15)

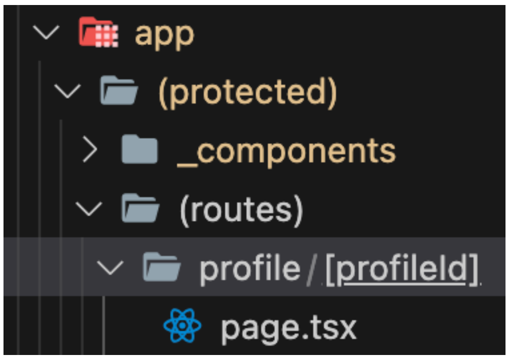
3.5 Frontend side solution

This chapter will explain technologies used to create Stucoin-frontend products and show essential features of this part of the Stucoin Platform.

3.5.1 Technologies

The core of building this solution was NextJS. This React framework enbables us to have features that help automate configuration to focus on developing applications. Now there will be listed some of these features:

- Routing: A file-system-based router that supports layout, nested routing, loading states, or error handling.



```
app
  (protected)
    _components
    (routes)
      profile / [profileId]
        page.tsx
```

Figure 10. Example route for open a specific user profile based on profileId argument

- Styling: NextJS supports predefined styling methods configured for us, like Tailwind CSS, used in this product.
- Data fetching: NextJS delivers simplified data fetching with async/await.

To construct reusable React components for building a modern, responsive user interface, we chose Tailwind CSS and Shadcn UI. The first is a utility-first CSS framework that streamlines the process of building user interfaces by providing a comprehensive set of utility classes. It offers a wide range of utility classes that can be applied directly to HTML elements to style them. This approach offers unparalleled flexibility and customization, allowing developers to create highly tailored designs without writing custom CSS. On the other hand, the second one is a component library built on top of Tailwind CSS, offering pre-designed components like buttons, forms, navigation bard, modals, and more.

AuthJS is an authentication library for React applications. It provides a simple way to implement authentication, including features like sign-in with various providers (such as social media accounts like Facebook, Github, etc.), JSON Web Token (JWT) authentication, and more. AuthJS is easy to integrate with NextJS and customizable to create protected routes pages and middleware for authenticating users.

Blocknote is a block-based rich-text editor for React. It solves challenges like allowing users to create content with various blocks that increase the readability of end documents, in our case, tasks and micro-competencies. Block is a paragraph, heading, image, or some other content predefined or defined by developers.

Blocknote offers more features like:

- Nesting and indentation in text
- Customizable slash (/) menu, but pressing this key while being on the editor, opens a pallet of all defined blocks that we can use

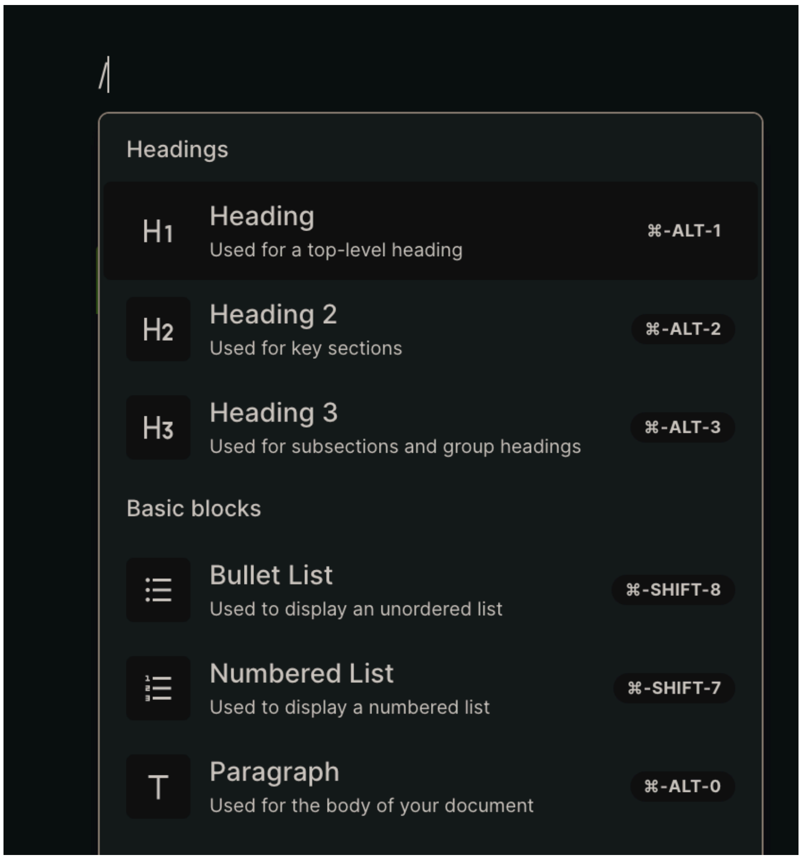

/
Headings
H1 Heading
Used for a top-level heading
⌘-ALT-1
H2 Heading 2
Used for key sections
⌘-ALT-2
H3 Heading 3
Used for subsections and group headings
⌘-ALT-3
Basic blocks
Bullet List
Used to display an unordered list
⌘-SHIFT-8
Numbered List
Used to display a numbered list
⌘-SHIFT-7
T Paragraph
Used for the body of your document
⌘-ALT-0

Figure 11. Blocknote slash command

- Drag and drop blocks
- Format menu enables changing the currently selected block to another type, giving color, or making text bold.

In Stucoin-frontend, we use ink! An embedded domain-specific language was created on top of the programming language Rust that helps to create smart contracts in Substrate with Polkadot. This solution gives some freedom by not worrying about the governance aspect or crypto-economics during development. (16)
To connect ink! with our user interface, we chose useink, a React library containing everything to handle ink! portfolio.

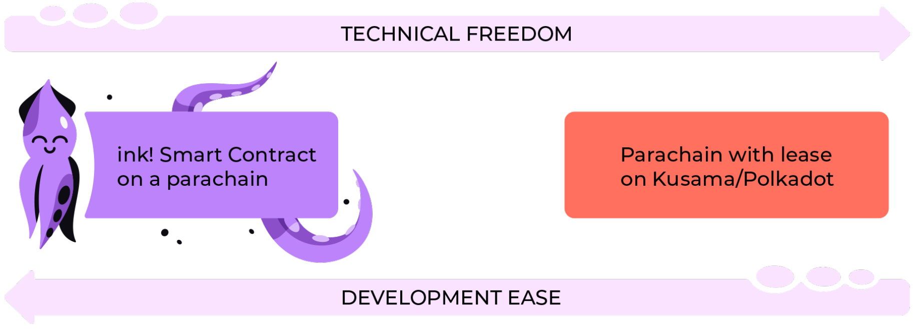

TECHNICAL FREEDOM
ink! Smart Contract on a parachain
Parachain with lease on Kusama/Polkadot
DEVELOPMENT EASE

Figure 12. Compare ink! with pure Polkadot parachains

The web client uses images that come from Storyset. It is free to use illustrations for creating frontend side applications; the only thing that is needed is to attach owners or Storyset pages to the website we are trying to create that is used as a link.

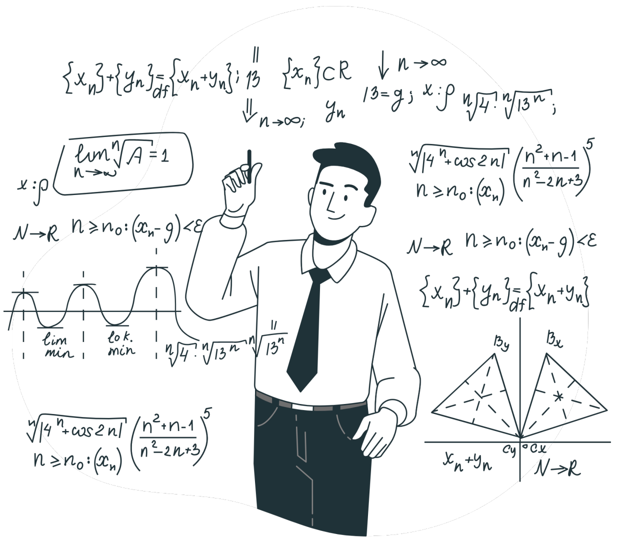


Figure 13. Example illustration that is used in project from Storyset

3.5.2 Implemented features

3.5.2.1 Task creator

It enables the teacher to create a task, a document consisting of a title, BlockNote editor, space for drag & drop files, adding cover Images, and setting points for completing the task for a student. Also, it displays information about the teacher and a student assigned to the task.

3.5.2.2 List of tasks

Displays a paginated list of tasks for students and for teachers, only their tasks. Also, each role can search tasks by title, sort by points, or filter by task status.

3.5.2.3 List of rewards

Displays a slide of rewards that student can exchange their points.

3.5.2.4 Settings

Users can change their personal data, passwords, or university details, they attend.

3.5.2.5 See profile

Enables to see user profiles is useful when we want to see teacher or student details from the specific task.

3.5.2.6 List of top students

Displays a list of students descending with a positive number of total scores, which means total points gained from completing tasks.

3.5.2.7 Login/register a user

We can register users with passwords and set appropriate roles. In the login section, we can log in by password and email or by Github and Google providers. This feature uses AuthJS.

3.5.2.8 Connect web3 wallet

Users can connect their wallets with applications to receive tokens or spend them for rewards (students). The application shares the possibility of connecting to wallet that support Polkadot environments like Nightly, Polkadot JS, and Subwallet. Users without connecting wallets can not see tasks or create micro-competencies.

4 Deployment

In this chapter, we dive into tools that streamline the deployment workflows. In Figure 9, we expose a single iteration CI/CD process. There is a need to use software development practices; with continuous deployment, we solve the challenges of manually integrating our Stucoin Platform with upcoming changes in code repositories. Upcoming sub-chapters will explain continuous integration and delivery and the tools we used to achieve our goals of automating the deployment of our system.

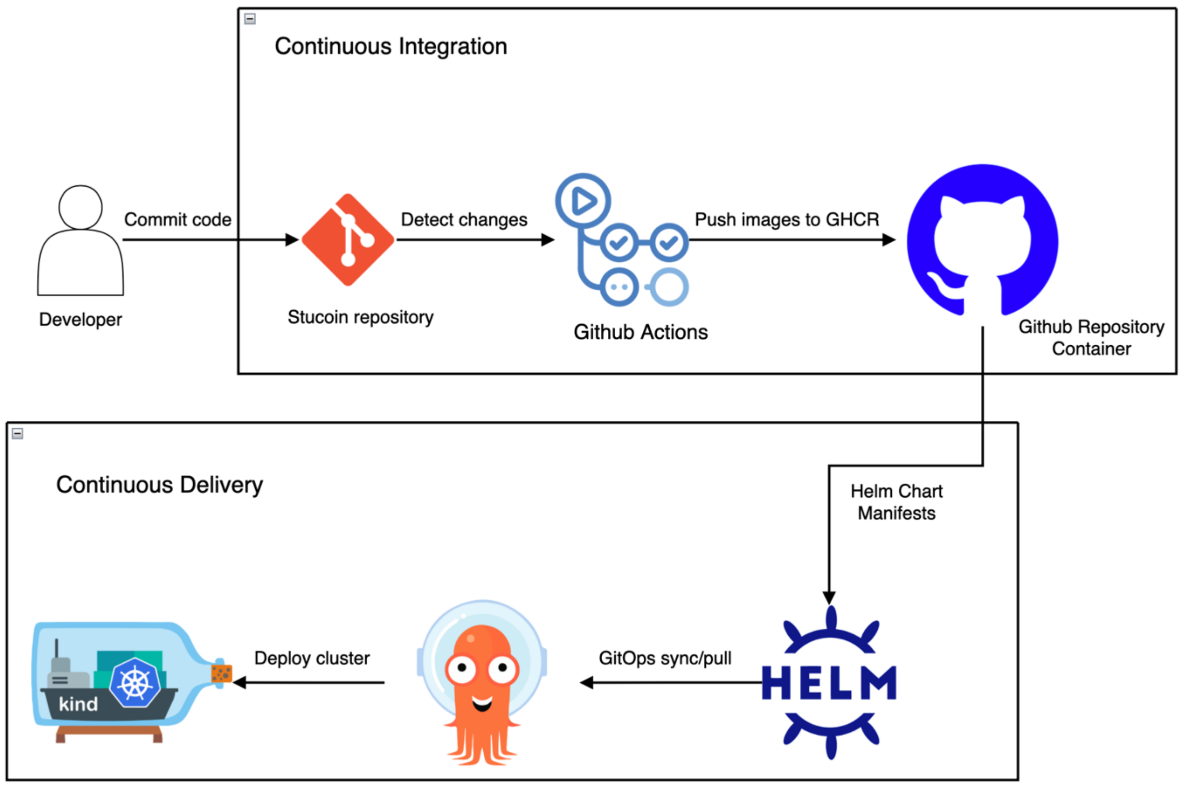


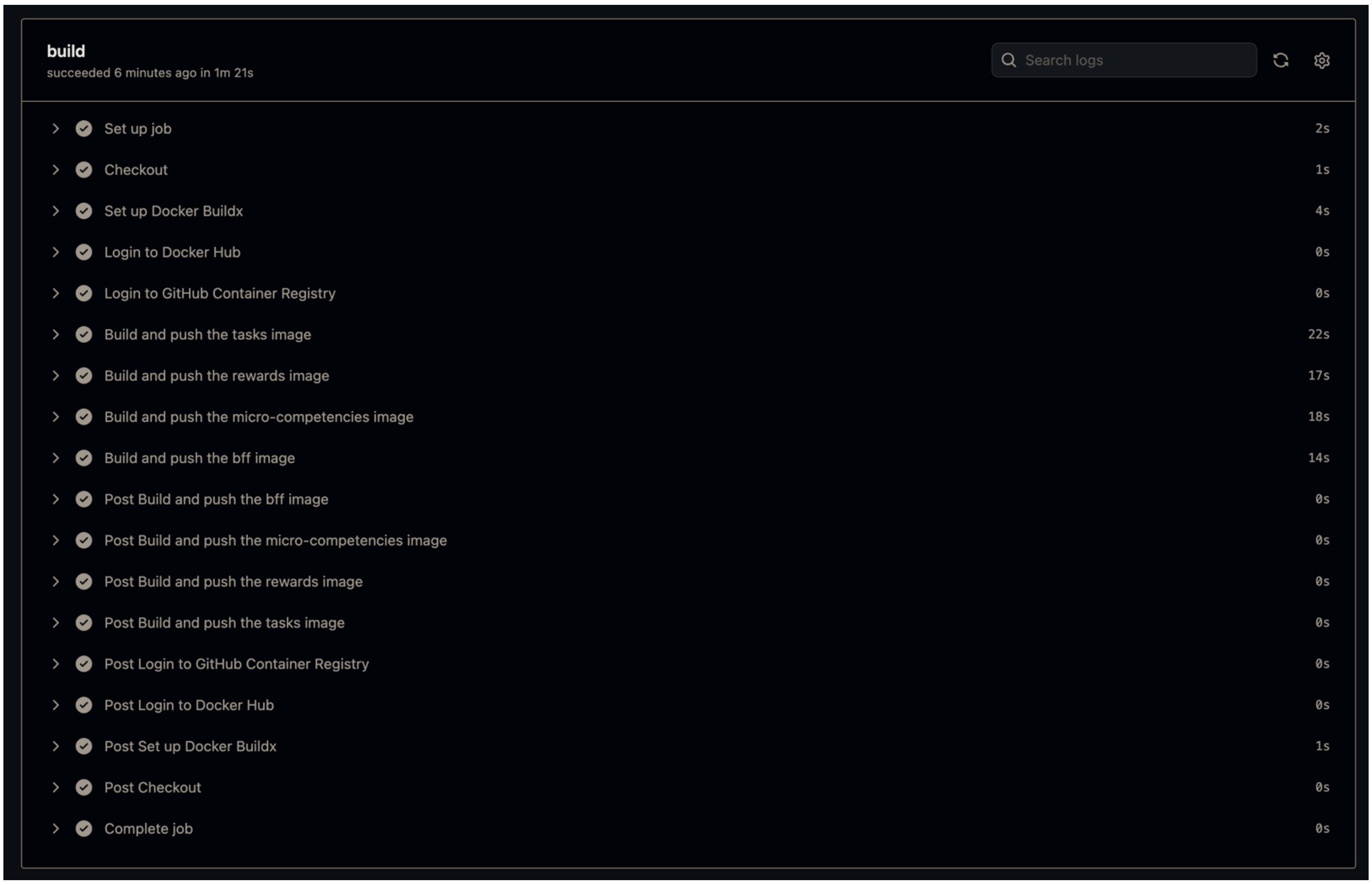
Figure 14. CI/CD diagram

4.1 Continuous Integration

Continuous integration (CI) enables integration code to verify its correctness frequently through automated tests. With CI, errors and security issues can be identified and fixed more efficiently and much earlier in the development process. (16) Our continuous deployment starts here; with CI, it is possible to do continuous delivery. So after commits changes in the Stucoin-backend repository, there is a triggered Github Actions pipeline.

Thanks to Actions, we can define custom workflows to test our code and build images of each microservice to the Github Repository Container where they are stored.

An example of a successful Github Actions pipeline in Stucoin-backend is shown in Figure 10.



```
build
succeeded 6 minutes ago in 1m 21s

Set up job                                              2s
Checkout                                                1s
Set up Docker Buildx                                    4s
Login to Docker Hub                                     0s
Login to GitHub Container Registry                      0s
Build and push the tasks image                         22s
Build and push the rewards image                       17s
Build and push the micro-competencies image            18s
Build and push the bff image                           14s
Post Build and push the bff image                       0s
Post Build and push the micro-competencies image        0s
Post Build and push the rewards image                   0s
Post Build and push the tasks image                     0s
Post Login to GitHub Container Registry                 0s
Post Login to Docker Hub                                0s
Post Set up Docker Buildx                               1s
Post Checkout                                           0s
Complete job                                            0s
```

Figure 15. Stucoin-backend build pipeline

As an example pipeline shows, many processes are required to successfully deploy images to the GitHub Container Registry, like logging in to appropriate platforms or building Dockerfiles. Developers, for now, do not have to worry about these aspects, and those pipelines save them time and allow them to focus on production.

Our Stucoin-backend services are ready to deliver to the next step, continuous delivery. Also, we can easily reuse our services by pulling images.

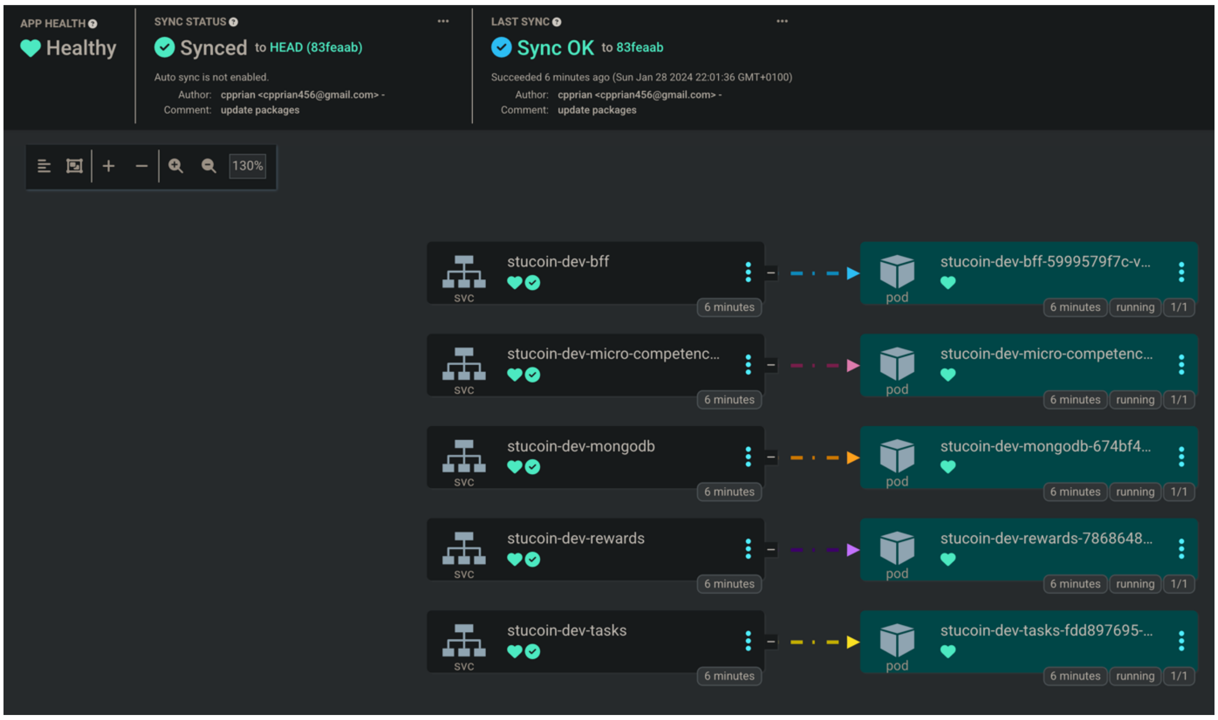
Example command to pull tasks service:

```
$ docker pull ghcr.io/cpprian/stucoin-tasks:latest
```

4.2 Continuous Delivery

Continuous Delivery (CD) is a follow-up after CI to automate the infrastructure provision and application release process after the code was testes and built. (16) This also facilitates rolling updates, which means new code changes are gradually roll out to production, ensuring minimal downtime and reducing the risk of service disruptions.

We approach this challenge using ArgoCD with Helm as a Kubernetes manifest. Both automate the deployment of applications to Kubernetes clusters by using git repositories as the source of truth for application configurations. Apart from that, ArgoCD has features like rollback to the last commits in a git repository, which could be helpful if something goes wrong in new changes that break our code; health status analysis that helps detect application errors, we can easily see the status of each service in web UI that ArgoCD provides to developers.


APP HEALTH
Healthy

SYNC STATUS
Synced to HEAD (83feaab)
Auto sync is not enabled.
Author: cpprian <cpprian456@gmail.com> -
Comment: update packages

LAST SYNC
Sync OK to 83feaab
Succeeded 6 minutes ago (Sun Jan 28 2024 22:01:36 GMT+0100)
Author: cpprian <cpprian456@gmail.com> -
Comment: update packages

Figure 16. ArgoCD in action

Helm is a package manager for Kubernetes applications, simplifying the deployment and management of complex applications on Kubernetes clusters. Helm charts provide a reusable and consistent way to deploy applications across different environments, promoting efficiency and reliability in the deployment process.

When used together, ArgoCD and Helm charts provide a powerful combination for managing and deploying cloud-native applications. ArgoCD automates the deployment process and the lifecycle management of applications, while Helm charts provides a declarative way to define and package applications.

At the end of whole process, there is a ready-to-use Kubernetes cluster that we can expose port to our API gateway and connect to the Stucoin-backend. Cluster work locally on KIND (Kubernetes in Docker), a tool for simulating nodes as Docker containers. In the future, cloud solutions like AWS or Azure will be considered for storing and managing products on the cloud, which is enabled outside of the local machine.

5 Conclusion

5.1 Results

Due to the project's lack of time and complexity, some features still need to be completed. They are listed as follows:

- They are ink! Providers follow ERC20 standards, but functions in web applications still need to be finished.
- The micro-competency module has yet to be created in the Stucoin-frontend, but on the Stucoin-backend, it is ready to use in the Kubernetes cluster.
- Add the admin interface in Stucoin-frontend.

However, without these aspects, the Stucoin Platform is a production-ready solution.

5.2 Related work & contributions

For now, there are no projects that solve the problem in this work, but some solutions have a similar idea to using blockchain as a source of truth to verify documents in terms of academic needs.

DoxyChain is one of them that suggests solving the authentication of certificates or any document using blockchain as a source of truth. (18) Regarding our micro-competency module, this could be the same idea to verify student achievement. Still, in the Stucoin Platform, students can create a new micro-competence at zero and then occasionally generate a pdf file that consists of all micro-competencies that teachers earlier accepted, and then anyone could verify.

5.3 Plans for future development

After finishing all modules, the following steps of this project will be to enhance CD aspects by deploying Stucoin-backend to the cloud like AWS or Azure. Still, we could also consider adding open-source solutions like Prometheus for monitoring systems and possible issues and Grafana for visualizing all the metrics that could be useful while maintaining and providing user support for the application.

To serve better our users, we can add multi-factor authentication, which means that after adding a phone number, they can verify their identity next time by providing a code displayed on their phone device to the application. Also, we could add a confirmation of the identity of the users after registering them to send an email with appropriate details.

We can enhance the system's functionality by adding a new module named Notification System. It could work like sending a notification to the user on the Notifications page, where the system informs users about any changes made to a task (new content, new files, or changing status) or micro-competency (accepted or rejected by a teacher). Adding a Rabbit MQ message broker in the Stucoin backend could be more attractive, which could exchange messages with the notification service with the rest of the services during some events, like changing the state of the task a student is assigned.

Also, we could enrich the task module experience by adding the following:

- Comments section, where the teacher and student who own the task could share thoughts.
- They were displaying task change history. This features enables viewing all users' chronological log of modifications, including who made the changes and when they were made.
- We could set up sever students to join the task as a group project.

Bibliography

1. **IBM.** ***Blockchain Overview*** ***.*** **[Online] https://www.ibm.com/topics/blockchain.**
2. Andreas M. Antonopoulos, Gavin Wood. ***Mastering Ethereum.*** **s.l. : O'Reilly Media.**
3. POLKADOT: VISION FOR A HETEROGENEOUS MULTI-CHAIN FRAMEWORK. [Online] https://assets.polkadot.network/Polkadot-whitepaper.pdf.
4. Polkadot wiki. ***Polkadot vs. Ethereum.*** **[Online] https://wiki.polkadot.network/docs/learn-comparisons-ethereum-2.**
5. Welcome to Substrate. ***substrate.*** **[Online] https://docs.substrate.io/learn/welcome-to-substrate/.**
6. Newman, Sam. ***Building Microservices.*** **1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reily Media, 2015.**
7. Why Use MongoDB and When to Use It? ***MongoDB.*** **[Online] https://www.mongodb.com/why-use-mongodb.**
8. MongoDB Drivers. ***MongoDB.*** **[Online] https://www.mongodb.com/docs/drivers/.**
9. What is Neon? ***NEON.*** **[Online] https://neon.tech/docs/introduction/about.**
10. What is Prisma? ***Prisma.*** **[Online] https://www.prisma.io/docs/orm/overview/introduction/what-is-prisma.**
11. adapters - Models. ***Auth.js.*** **[Online] https://authjs.dev/reference/core/adapters#models.**
12. ***Edge Store .*** **[Online] https://edgestore.dev/.**
13. Fetisov, Evgeniy. Node.js vs Golang: Which Technology is Best for Your Projects. ***Jay.devs.*** **[Online] https://jaydevs.com/nodejs-vs-golang/.**
14. Go for Web Development. ***go.dev.*** **[Online] October 4, 2019. https://go.dev/solutions/webdev.**
15. gorilla/mux. ***github.com.*** **[Online] https://github.com/gorilla/mux.**
16. gitlab.com. ***What is CI/CD?*** **[Online] https://about.gitlab.com/topics/ci-cd/#What%20is%20continuous%20delivery%20(CD)?.**