



universität
wien

BACHELORARBEIT / BACHELOR'S THESIS

Titel der Bachelorarbeit / Title of the Bachelor's Thesis

„uChain - Blockchain-powered study performance and
evaluation pass“

verfasst von / submitted by

Tamás Révész

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of
Bachelor of Science (BSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt /
degree programme code as it appears on
the student record sheet:

UA 033 521

Studienrichtung lt. Studienblatt /
degree programme as it appears on
the student record sheet:

Bachelorstudium Informatik

Betreut von / Supervisor:

Univ. Prof. Dr. Wolfgang Klas

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas, head of the research group Multimedia Information Systems at the University of Vienna. His guidance, encouragement, and support throughout the course of my research have been invaluable. I am also grateful for his invaluable feedback and suggestions that greatly improved the quality of my work.

I would also like to thank my parents for their constant support and understanding throughout my studies. Their encouragement and belief in me have been a source of inspiration and motivation.

I would like to acknowledge the support of the Computer Science faculty and the Multimedia Information Systems research group at the University of Vienna for providing me with the resources and facilities necessary to complete my research.

Finally, I would like to extend my appreciation to all those who have supported me in any way during the course of this research.

Abstract

This research project aimed to explore the feasibility of using blockchain technology as an alternative to conventional university administration systems. Our goal was to build a study performance and evaluation pass that is operated by smart contracts. The pass records individual, required study achievements like milestones, tests, etc. during a course, the final grading of a course, and the collection of gradings of courses during the entire study. A private, Ethereum-based blockchain was created with proof-of-authority mining consensus, and a set of smart contracts were deployed on the blockchain to handle the administrative logic and data storage for the platform. A web application (dApp) which allows the integration of a cryptocurrency wallet browser extension was developed for user interaction, along with off-chain file storage and user registration services. The prototype was tested both programmatically and by human testers on a series of well-defined use cases. The results indicate that blockchain technology has the potential to serve as a viable alternative to traditional administration systems, but user education on the use of smart contract-powered systems is crucial.

Kurzfassung

Dieses Forschungsprojekt zielte darauf ab, die Machbarkeit der Verwendung von Blockchain-Technologie als Alternative zu konventionellen Universitätsverwaltungssystemen zu untersuchen. Unser Ziel war es, einen Studienleistungs- und Evaluationspass zu erstellen, der von Smart Contracts betrieben wird. Der Pass dokumentiert die individuellen, erforderlichen Studienleistungen wie Meilensteine, Tests usw. während eines Kurses, die endgültige Bewertung eines Kurses und die Sammlung von Kursbewertungen während des gesamten Studiums. Eine private, auf Ethereum basierende Blockchain mit Proof-of-Authority-Mining-Konsens wurde erstellt, und eine Reihe von Smart Contracts wurden auf der Blockchain bereitgestellt, um die administrative Logik und Datenspeicherung für die Plattform zu verwalten. Eine Web-Anwendung (dApp), die die Integration eines Kryptowährungsbrieftaschen-Browser-Erweiterung ermöglicht, wurde für die Benutzerinteraktion entwickelt, zusammen mit Off-Chain-Dateispeicher- und Benutzerregistrierungsdiensten. Das Prototyp wurde sowohl programmgesteuert als auch von menschlichen Testern auf einer Reihe gut definierter Anwendungsfälle getestet. Die Ergebnisse zeigen, dass Blockchain-Technologie das Potenzial hat, als alternative zu traditionellen Verwaltungssystemen zu dienen, aber eine Bildung der Benutzer über die Verwendung von smart contract-gestützten Systemen ist von entscheidender Bedeutung.

Contents

Acknowledgements	i
Abstract	iii
Kurzfassung	v
List of Tables	xi
List of Figures	xiii
List of Algorithms	1
1. Introduction	1
1.1. Problem statement	1
1.2. Goals of the work	2
1.3. Process of the work and methods	2
1.4. Contribution	3
2. Related work	5
2.1. Inspiration from existing study performance and examination pass systems	5
2.1.1. u:find	5
2.1.2. Moodle	5
2.1.3. u:space	5
2.2. Inspiration from similar research projects	6
3. System requirements	9
3.1. Use cases	9
3.2. Considered use cases	10
3.3. Quality related requirements	10
4. System design	11
4.1. Network	11
4.1.1. Blockchain type	11
4.1.2. Consensus mechanism	11
4.1.3. Type of nodes	11
4.2. Smart contracts	12
4.2.1. Model	12
4.2.2. View	13

Contents

4.2.3. Controller	13
4.3. Supporting servers	13
4.3.1. File server	13
4.3.2. Registrator server	14
4.4. Web application	15
4.4.1. Login page	15
4.4.2. Registration	16
4.4.3. Registration (after the registration request is sent)	16
4.4.4. Pending registrations	16
4.4.5. Home	16
4.4.6. Profile	16
4.4.7. All courses	16
4.4.8. My courses	17
4.4.9. All study programs	17
4.4.10. My study programs	17
4.4.11. Add student to course	17
4.4.12. Create new study program	17
4.4.13. Your study performances	17
4.4.14. Assessment performances	18
4.4.15. Grading	18
4.4.16. Create new course	18
5. Implementation	19
5.1. Overview of technology stack	19
5.2. Blockchain network	21
5.2.1. Account generation	21
5.2.2. Creating genesis.json	21
5.2.3. Setting up bootnode	21
5.2.4. Network deployment	22
5.3. Smart contracts	22
5.3.1. Stages of input validation	22
5.3.2. Access validation	23
5.3.3. Challenge of contract dependencies	24
5.3.4. Automatic grade calculation	25
5.3.5. Deployer	25
5.4. Registrator server	26
5.5. File server	26
5.6. Frontend	26
5.6.1. Web3	26
5.6.2. Component library	27
5.6.3. User authorization hook	27
5.6.4. Displaying error messages	27
5.6.5. Dynamic loading of accordion content	28

5.7. Overview of the deployed system	28
5.8. How to run the prototype	28
6. Evaluation and discussion	29
6.1. Sample data	29
6.2. Programmatic testing	30
6.2.1. Test setup and test cases	30
6.2.2. Running the tests	31
6.2.3. Results	31
6.3. User testing	31
6.3.1. Selection of testers	31
6.3.2. Test setup and user tasks	31
6.3.3. Results and feedback	32
6.4. Discussion of results	32
7. Conclusion and future work	33
Bibliography	35
A. Appendix	39
A.1. Use cases	39
A.2. Diagrams	53
A.3. Test output	56
A.4. Puppeth config	59
A.5. MetaMask explanation protocol	60
A.6. User test instruction protocol	60
A.7. User interface	62

List of Tables

A.1. Course registration	39
A.2. Course de-registration	40
A.3. Assessment registration	41
A.4. Assessment de-registration	42
A.5. Submission upload	43
A.6. Test attendance	44
A.7. Assessment evaluation	45
A.8. Grading	46
A.9. Modification of evaluation	47
A.10.Modification of grade	48
A.11.User registration	49
A.12.Creation of new course	50
A.13.Creation of new study program	51
A.14.Course registration of student by SPM	52

List of Figures

2.1. Screenshot of the u:find platform displaying the course selection of study program Bachelor Computer Science of 2023 summer semester [Unic] . . .	6
2.2. Screenshot from a Moodle tutorial showing how students can upload their solution for an assignment [Unib]	7
2.3. Grading tutorial on u:space for course instructors [Unid]	8
4.1. UML activity diagram of the registration process	15
5.1. Execution flow of the used technologies based on Figure 4 of [ACB20] . . .	20
5.2. Excerpt from Figure A.2 (Appendix), showing the data manager, controller and view contracts	24
A.1. UML class diagram of the smart contracts	53
A.2. UML object diagram of the deployed smart contracts	54
A.3. UML deployment diagram of system components	55
A.4. Login page	62
A.5. Home page	62
A.6. Registration page	63
A.7. Registration page (after request is sent	63
A.8. Pending registrations page	64
A.9. Profile page	64
A.10.All courses page	65
A.11.My courses page - student	65
A.12.My courses page - lecturer	66
A.13.All study programs page	66
A.14.My study programs	67
A.15.Add student to course page	67
A.16.Create new study program page	68
A.17.Your study performances page	68
A.18.Your study performances page - cont.	69
A.19.Assessment performances page	70
A.20.Assessment performances page - submission	70
A.21.Assessment performances page - exam	71
A.22.Grading page	71
A.23.Grading page - cont.	72
A.24.Create new course page	72
A.25.Create new course page - cont. 1	73

List of Figures

A.26.Create new course page - cont. 2	73
---	----

1. Introduction

Blockchain technology and smart contracts are among the most promising technological advancements in recent years. Blockchain is a distributed digital ledger that records transactions across a network of computers creating a decentralized system. It is the technology that underpins the digital currency, Bitcoin [Nak09].

In the current state, blockchain technology is being tested and implemented in various use cases across different industries, from supply chain management, healthcare, finance, real estate, and education.

The Ethereum blockchain platform, created by Vitalik Buterin, was designed to be more versatile than Bitcoin, since it would allow developers to build decentralized applications (dApps) on top of a blockchain. This is achieved by deploying smart contracts on the network [But13].

Interestingly, the term "smart contract" was invented well before Ethereum or Bitcoin. It was a concept first created by Nick Szabo [Sza94], who introduced the concept of smart contracts, which are digital contracts (programs) that are automatically executed when the predetermined conditions are met [sma].

1.1. Problem statement

The current university administration systems are implemented using conventional technologies, which handle all the administrative logic and data storage on a centralized server (e.g. [B.S18]). The platform provided by the Ethereum network could be suitable to house both the logic and storage functionality of such systems with the benefits of higher transparency for the users - thanks to the data immutability on the blockchain.

The usage of blockchain technology, which is inherently designed for decentralization, could greatly enhance the stability of the system, which is a critical point for systems like the one used for a university's administration. Being decentralized means that the network is not controlled by a single entity, but rather distributed among a network of nodes. This decentralization can be leveraged to increase the system's stability by allowing new machines to be added to the network at any time and with relatively low effort. Additionally, the decentralization of the network also ensures that the system is not reliant on a single point of failure, making it much more robust and resilient to potential issues. Therefore, the implementation of blockchain technology can greatly increase the stability of the system by providing a decentralized infrastructure that is adaptable and resilient.

1. Introduction

1.2. Goals of the work

The goal of this project is to implement an application for a digital "Studienleistungs und Prüfungspass" (study performance and examination pass) based on blockchain technology. The pass records individual, required study achievements like milestones, tests, etc. during a course, the final grading of a course, and the collection of gradings of courses during the entire study. The implementation is based on Ethereum blockchain technology, using smart contracts. The focus of the project is on the proper design and implementation to capture most of the functionality of the application.

The system provides administrative functionalities such as enrolling in courses, registering to and deregistering from courses within a defined timeframe, submitting assignments, taking tests, grading tests and assignments, generating final evaluations based on rules defined by course instructors, and assigning grades based on established criteria.

The smart contracts are deployed on a private Ethereum network that uses the proof-of-authority consensus mechanism.

In order to determine the effectiveness of the application that was developed, it is necessary to evaluate its ability to successfully execute the use cases A.1 that were provided. These use cases represent specific scenarios in which the application should be able to function as intended, and serve as a means of measuring the application's ability to meet its goals. By examining the application's performance in executing these use cases, it is possible to gain insight into its overall capabilities, and identify areas where improvements may be necessary. Overall, the ability of the application to successfully execute the provided use cases is a key metric for evaluating its effectiveness and determining if it has achieved its goals.

1.3. Process of the work and methods

The process of completing this project began with the writing of a project conception based on the project requirements. This served as a foundation for the project and helped to define the overall goals and objectives. Next, we studied similar projects in order to gain a better understanding of the current state of the field and to identify potential challenges. After that, we decided on the technology stack that would be used for realization of the project.

The next step was to design the system architecture, which involved determining the overall structure and layout of the system, as well as the specific components that would be required. This was followed by the building of the private blockchain and the smart contracts, which were the foundation of the system. Additionally, we built supporting subsystems that were required to make the system function properly in cases which could not be covered with smart contract functionality only.

The final stages of the project involved creating a user interface and testing the smart contracts programmatically. Usage scenarios were tested by a selection of users in order to evaluate if the use cases can be executed and ensure that the system was functioning as intended.

1.4. Contribution

The main contribution of my project is that it explores the potential of applying smart contract technology in a university setting. By creating a private Ethereum blockchain that uses the proof-of-authority consensus mechanism and implementing a platform for handling university administration tasks, my project demonstrates the feasibility of using blockchain technology in a real-world scenario.

Furthermore, my project could serve as a reference for other university-related use cases in the field of blockchain technology. By providing a working implementation of a smart contract-based administration system, other developers and researchers can use my project as a starting point for similar projects and as a source of inspiration for new ideas. Additionally, it could also be used by universities as a reference for potential implementation of such a system in their own institution.

2. Related work

2.1. Inspiration from existing study performance and examination pass systems

The University of Vienna currently employs a number of different systems to manage the administrative needs of its students, particularly in terms of tracking and recording study performance and examination results. These platforms include u:find, u:space, and Moodle [Unie]. My intent when creating my platform was to combine parts of these platforms' functionality into a single platform.

2.1.1. u:find

u:find is a directory platform offered by the University of Vienna that provides students and staff with a comprehensive overview of all courses and staff members at the university. The platform is organized by semesters and allows users to search for courses (see Fig. 2.1.1), examinations, persons, or organizations. Users can also check course and examination dates, register for courses and examinations, and export calendar information. In addition, u:find also provides building plans of the University of Vienna via Google Maps, making it easy for students and staff to navigate the campus [Unie].

2.1.2. Moodle

Moodle is an e-learning platform used by some courses at the University of Vienna. It allows instructors to upload materials, communicate with students, conduct tests, and share recordings. Student can also upload their solutions for assignments here (see Fig. 2.1.2). Not all courses use Moodle and those that do will have a link to the Moodle course in u:find. A u:account registration is required to access Moodle [Unif].

2.1.3. u:space

u:space is a portal for students and staff at the University of Vienna to manage the administration of university-related activities. It provides central access for students to all services required during their studies, such as applying for admission to a degree program, ordering student ID cards, paying tuition and fees, registering for and de-registering from courses and exams, checking grades and academic progress, downloading and printing study documents, and finding information about campus rooms. Its features also include the ability for staff to enter course descriptions in u:find. It provides easy management of the registration process for each course, including adding or removing students. Course

2. Related work

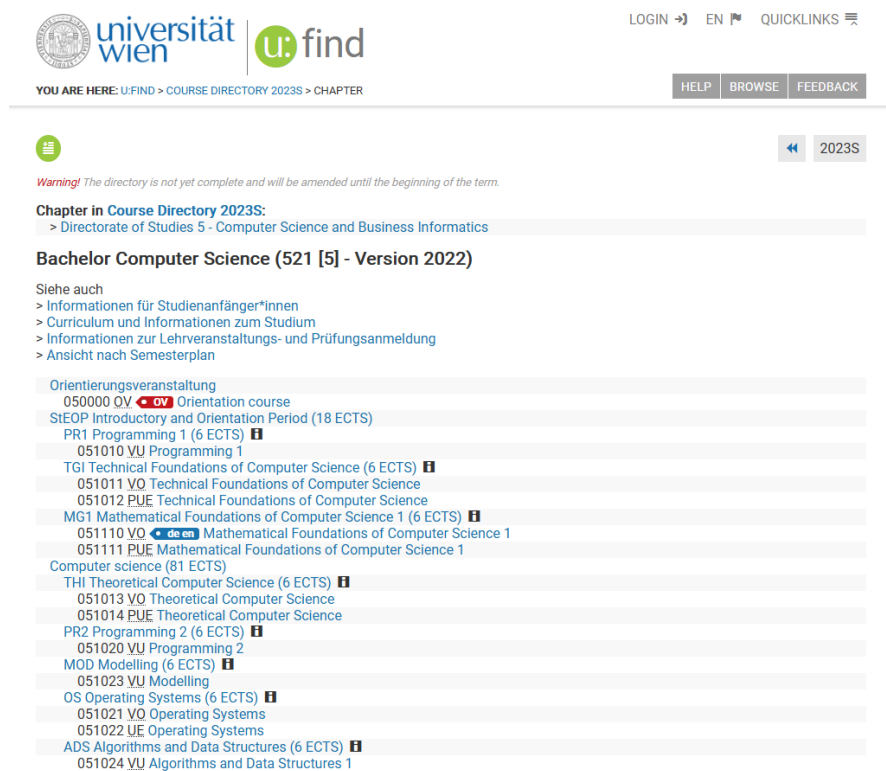


Figure 2.1.: Screenshot of the u:find platform displaying the course selection of study program Bachelor Computer Science of 2023 summer semester [Unic]

instructors can also enter grades for students of their courses (see Fig. 2.1.3). The same u:account registration that is required for logging in Moodle is required on this platform as well [Unie].

2.2. Inspiration from similar research projects

In this section, we present an overview of previous researches in the field of administration systems utilizing blockchain technology. This includes studies and projects that have been developed to address the specific challenges and limitations of traditional administration systems, and how blockchain technology can be applied in these scenarios. We will examine the different approaches and methodologies used in these works in order to gain a comprehensive understanding of the possibilities these systems can achieve.

Abreu et al. [ACB20] proposed a reference architecture for using blockchain technology to register and consult student degree certificates issued by higher education institutions. The authors argued that blockchain technology offers several advantages, such as document authentication, transparency, immutability, and trust, which make it an attractive solution for higher education. They also mentioned that most higher education institutions have

2.2. Inspiration from similar research projects

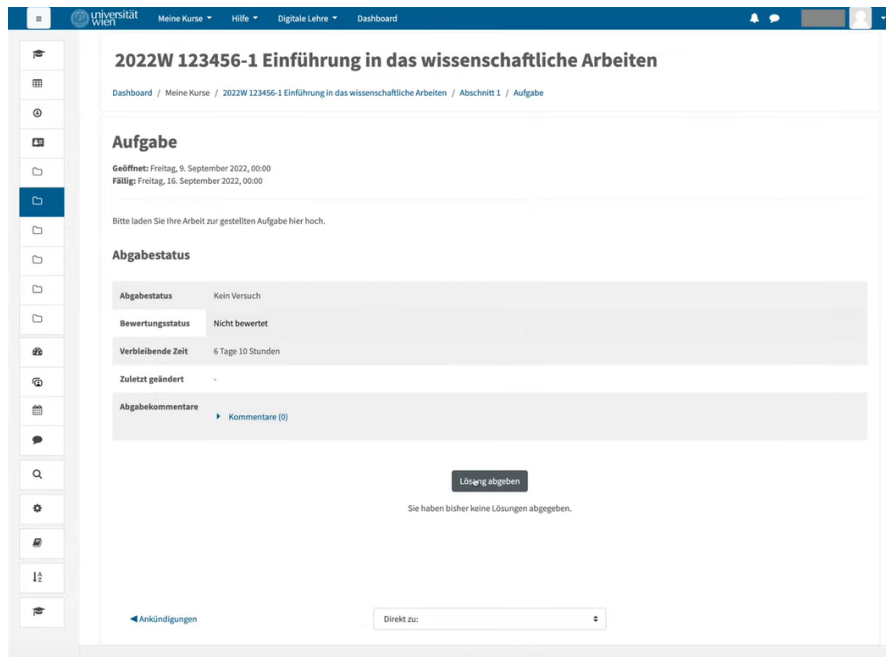


Figure 2.2.: Screenshot from a Moodle tutorial showing how students can upload their solution for an assignment [Unib]

their own specialized systems for maintaining student records, which are hosted in-house and have little or no interoperability. The methodology used in the study includes proposing a reference architecture for blockchain, developing a prototype for an application in the educational domain, designing a usage scenario, conducting technical validation of the prototype, validating the prototype for the applicability of the blockchain, and validating the prototype with a user. The Ethereum platform was chosen to create the proof-of-concept, using smart contracts to store and share data, and executing smart contracts. The proposed reference architecture for blockchain is based on previous research and includes an application layer, an API layer, a blockchain layer, a conventional database, consensus algorithms and a network layer. The prototype was tested on the Ropsten public testing network and allows entities to send and retrieve data through the use of a front-end application. The study found that the proposed approach based on blockchain technology could provide a viable alternative for storing and controlling access to student certificate data, increasing trust and transparency in the validation of diplomas.

Mjoli et al. [MD21] explored the integration of blockchain technology and IoT in a smart university application architecture. The authors proposed the use of a blockchain-based architecture model, specifically the concept of the Ethereum blockchain combined with the proof-of-authority consensus mechanism, as a potential solution to developing a data-centric architecture model. The proposed model was divided into three layers: the network layer, the interaction layer, and the application layer. The design considerations included that the components of the model must be plug and play and that the communication

2. Related work

Noten eingeben

Filtern: ☒ angemeldet ☒ bewertet ☐ abgemeldet ☐ nicht erschienen ☒ abgeschlossen

Suchen: [Zurücksetzen](#)

NACHNAME, VORNAME	MATR.NR.	BEWERTUNG	DATUM	BEURTEILT DURCH	STATUS	BEGRÜNDUNG	GESPEICHERT
Blau, Marie	3456706	1	30.08.2020	Meisinger, Eva	bewertet		✓
Einstein, Alfred	3456710	4	30.08.2020	Meisinger, Eva	bewertet		✓
Fleißig, Carina	3456714	2	30.08.2020	Meisinger, Eva	bewertet		✓
Freud, Siegfried	3456705	3	30.08.2020	Meisinger, Eva	bewertet		✓
Jahoda, Marina	3456707	4	30.08.2020	Meisinger, Eva	bewertet		✓
Meier, Herbert	3456713	3	30.08.2020	Meisinger, Eva	bewertet		✓
Muster, Benni	3456711	1	30.08.2020	Meisinger, Eva	bewertet		✓
Sorglos, Andrea	3456708	1	30.08.2020	Meisinger, Eva	bewertet		✓

< Zurück 1 Weiter >

[Zurück](#) [Noten aus Lernplattform importieren](#) [Bewertung freigeben](#) [zum Seitenanfang](#)

Figure 2.3.: Grading tutorial on u:space for course instructors [Unid]

protocol in the network layer must be standardized. The system was tested in a simulated environment within the context of a smart university campus that is subsumed by a smart city. The goal of this research is to address the complexities and lack of control that current IoT systems present by introducing a data-centric approach that utilizes blockchain technology to provide increased security, transparency, and user control over the usage of their data.

Pongnumkul et al. [PKLP20] discussed a proof-of-concept of integrating blockchain technology in the land registration system in Thailand to improve the reliability of storage and reduce the steps in the process. The authors of the paper used the process of loan contracts with banks as a use case for the PoC. The model was composed of three main components: private blockchain, smart contracts, and web application. The private blockchain is developed using Ethereum as basis and is controlled by the Department of Land and banks. Smart contracts are used to manage the title deeds and record and verify transactions related to land deeds. The web application serves as an interface for the Department of Land, banks, and the public to access the blockchain network and carry out various activities with the title deeds. They conducted performance evaluations of the system. The authors suggest that the results show promising potential for reducing process and improving the performance of blockchain in land registration systems.

3. System requirements

In our research project, we aimed to explore the potential of applying blockchain technology in the field of university administration systems. A study performance and evaluation pass consists of a number of complex functionalities and operations. As highlighted in the "Related work" section, the University of Vienna currently employs multiple different services, such as u:find, u:space, and Moodle, to manage the various requirements of such a system. However, our goal was to create a single, blockchain-powered platform that could serve as a viable solution for the university's administrative needs. Therefore, it is important to express that the scope of this project was not to produce a production-ready prototype, and thus, the system requirements were narrowed down to a manageable coverage.

3.1. Use cases

In the early stages of our research project, a set of specific use cases were identified and agreed upon as the functional requirements for the prototype system. These use cases served as the foundation for the development of the prototype and were refined and extended as the project progressed and a better understanding of the technological capabilities and user experience needs were gained. The full definition of these use cases can be found in Appendix A.1. In the subsequent section, a summary of the use cases will be presented.

In general, the platform was expected to determine if all prerequisites for executing a use case are met, and only allow actions if all checks passed. The project's use cases included functionality for course registration and de-registration, where a student can select a course and register for it if the course has available places and the student is eligible for participation, and subsequently de-register if needed. Additionally, the system included functionality for assessment registration and de-registration, allowing students to register for assessments that require separate registration and optionally de-register from them.

The system should provide the feature of uploading submissions, where students can submit documents within a course before the submission deadline. The functionality for test attendance was needed as well, where course instructors can confirm whether a student attended a test. For assessments, course instructors can submit evaluations for students' performance. The option for an automatic grading was expected, which operates based on the students' performance throughout the semester.

The course instructor should have the ability to modify evaluations and grades retrospectively. The system also included functionality for user registration, where students

3. System requirements

and course instructors can register in the system and provide their personal information for review and acceptance by the university administration or study program management. Course and study program creation were another important capabilities of the platform, where a course instructor can create a new course and study program manager can create a new study program.

3.2. Considered use cases

There were a selection of functionalities that were taken into consideration for potential features but could not be implemented due to time constraints. These functionalities may serve as reasonable extension of this research project given the possibility for a follow-up work.

One such functionality is the ability for students to obtain a PDF version of their performances, including grades and evaluation of tests and submissions. Another feature could be the handling of cases where more students are registered for a course than the participation limit. Additionally, a teacher could create courses with multiple groups, allowing students to register to multiple groups simultaneously and a preference ranking would be used to decide which group the student gets accepted into. Other potential features include the ability for administrators to add more admins to the system, change the wallet address connected to users' information in case of loss of access, revoke user access and adding grades to students.

3.3. Quality related requirements

The system was expected to achieve a number of quality related requirements:

- The system was expected to be able to process and consistently store all transactions of the actors.
- It should grant students access to their own and their own only performance records.
- Course instructors should only be able to input evaluations and define evaluation criteria for their own courses.
- The system should provide an easy-to-use web interface for interacting with smart contract interfaces.
- The system should also provide test data with users, courses and study programs.

4. System design

In this section of this research paper, the architecture of the system that was implemented to realize the use cases will be presented. The system is comprised of four main components: the blockchain network, the smart contracts, the supporting servers and the web application. The system design incorporated key considerations from the work of Mjoli et al. [MD21] with an emphasis on modularity. This ensures that individual components can be easily replaced without disrupting the functionality of other components. For example, the system should be able to handle a change in the blockchain network without requiring modifications in the rest of the system components, allowing for a seamless deployment. The design of each component will be discussed in further detail in this section.

4.1. Network

Our system utilizes the Ethereum blockchain technology as its foundation, allowing for the secure execution of smart contract transactions.

4.1.1. Blockchain type

In order to maintain control and exclusivity over our network, we opted for a private blockchain network [Inv]. A private blockchain can only be joined through invitation from the network operator. This allows us to benefit from the capabilities of blockchain while maintaining full control over its participants.

4.1.2. Consensus mechanism

To ensure the security and efficiency of our network, we have chosen to implement the proof-of-authority consensus mechanism [Coi]. This mechanism assigns authority to specific nodes, known as validators or signers, who are pre-selected and possess a reputation for trustworthiness. These validators are responsible for validating transactions and creating new blocks on the blockchain. As PoA does not rely on computational power, it is a cost-effective solution for our private network [Coi].

4.1.3. Type of nodes

Our network has three types of nodes [MD21]:

- **Bootnode:** A bootnode is a special type of node that is responsible for maintaining a list of all the active nodes on the network. It acts as a directory service and helps new nodes to find and connect to the network.

4. System design

- **Signer node:** A signer node, also known as a validator node, is a node that is responsible for validating transactions and creating new blocks on the blockchain. These nodes are pre-selected by the network operator. They are responsible for maintaining the integrity of the network.
- **Client node:** A client node, also known as a non-validating node, is a node that is connected to the network but does not participate in the validation of transactions or the creation of new blocks. These nodes can only read data from the blockchain and can only send transactions to be included in blocks.

4.2. Smart contracts

The smart contracts are deployed in our private blockchain, they contain the business logic of the system and are responsible for handling the different use cases.

In a blockchain network that supports smart contracts, such as Ethereum, gas is a unit of measurement that represents the computational effort required to execute a specific operation on the blockchain. When a smart contract is executed, the network's nodes use computational power to process the contract's code and the transaction associated with it. The cost of this computation is measured in gas, and the user must provide this required amount of gas to cover the gas costs. The gas cost of a smart contract is determined by the number of operations required to execute the contract and the current gas price set by the network [ethb]. This price increases on higher network load, hindering overload induced denial-of-service cases.

In general, it is more efficient to have a fewer number of longer smart contracts that are optimized for gas usage (because gas costs money). However, in the context of our private network, where the tokens paid for gas do not hold any real monetary value, we have made the decision to prioritize ease of maintenance and modification. To achieve this, we have employed a greater number of smart contracts that effectively apply the principle of separation of concerns. To accomplish this goal, we have chosen to implement the Model-View-Controller architecture model.

The Model-View-Controller (MVC) architecture model is a design pattern used to organize code in a way that separates the data model (the "Model"), the user interface (the "View"), and the control flow (the "Controller") [Moz]. The MVC architecture helps to keep the code organized and easier to understand, and also allows for easy modification of the application. It also enables separating the concerns of the different parts of the application, allowing for a more maintainable and testable codebase [Evg]. On the other side, it produces an inflated codebase which leads to more gas-intensive code execution. The overview of the developed contract can be seen in the Appendix on Figure A.2.

4.2.1. Model

The Model represents the data of the application. It contains the data structures and methods that are used to manipulate the data. Our adapted Model consisted of a two-layered architecture, with the "storage" layer as base layer and the "data manager" layer

4.3. Supporting servers

as top layer. The storage layer was created to function like a database of a conventional application, providing only basic create-read-update-delete methods. The smart contracts of the storage layer communicate directly only with smart contracts of the data manager layer. The data manager layer was responsible for offering more complex methods for manipulating the storage layer that can be used by the View and Controller. Since the stored data structures are handled by the data manager layer, the View and Controller contracts do not require intimate knowledge about the way the data is stored or how the stored data structures are internally formatted.

4.2.2. View

The View represents the user interface of the application. It is responsible for displaying the data from the Model. Our modified View operates similarly to the GET methods of a REST API application. It offers specific methods required to showcase the data stored in the system. The contracts of View reads directly from the data manager contracts of the Model.

4.2.3. Controller

The Controller is responsible for handling user input and updating the Model accordingly. In our system, the Controller operates similarly to the POST/PUT/DELETE methods of a REST API application. When a user executes a state-changing transaction, the Controller receives the input and updates the Model. The contracts of the Controller manipulate the Model's state via the data manager contracts.

4.3. Supporting servers

The supporting servers, the file server and the registrator server, are system components that possess specific functionalities that cannot be fulfilled by smart contracts alone.

4.3.1. File server

Use case A.5 requires the functionality of uploading documents into the system. While it is technically possible to store the bytes of a multimedia file on the blockchain, it would be highly computationally expensive and would rapidly inflate the size of the blockchain. It is important to note, it would not be possible to reduce the blockchain's size by deleting the file, as the blockchain stores the data of every transaction that has ever been executed on it, thus the old transaction containing the uploaded file would still be intact.

To address this issue, we have implemented a dedicated file server running off-chain, which is responsible for the storage and retrieval of multimedia files. This server offers an API endpoint for file upload and retrieval. This way, instead of storing the files themselves, only their identifying hash values would be recorded on the blockchain. Further details on the implementation can be found in the "Implementation" section.

4. System design

4.3.2. Registrar server

As outlined in use case A.11, the system must have the capability to handle user registrations, as it is a prerequisite for access to the majority of the system's functionality. In a smart contract application, a user must possess a wallet address that can be used to identify them, and they must have network tokens in that wallet to send state-changing transactions due to the gas costs associated with these transactions. This presented a chain of challenges for us.

We started at the issue of users wanting to register to the platform by interacting with the appropriate smart contract method, but not having any tokens to cover the gas costs associated with this transaction. A possible solution to this problem could be to allow transactions on the network that have 0 gas price. However, as previously discussed in subsection "Smart contracts", gas prices are set dynamically by the network based on network load, and even though a transaction may be allowed with 0 gas price, it may still become unfulfilled, preventing the user from registering.

Another solution could be the integration of a concept known as a "faucet" [Cry], which is a smart contract containing a large number of tokens that can be allocated to other wallets on the network upon request. This would allow users to request tokens from the faucet contract in order to use the network. However, this request would also require a transaction that costs gas, making it impossible for the user to execute without tokens.

To address this issue, we decided to implement a faucet contract while also finding a way for the user to initiate the token request off-chain. A server that has access to its own wallet loaded with tokens to cover transaction costs served as a solution by providing an API endpoint that executes a transaction signed by the server's wallet. This way, the user could request tokens by calling the server's API endpoint and receive the allocated tokens on-chain. However, this approach also presented a potential problem in that a malicious actor could drain the faucet's token reserve by repeatedly requesting tokens through the server.

Our final solution was to restrict the faucet's token allocation by tying it to a registration request and only performing the allocation if the wallet connected to the user's registration had been reviewed and accepted by a staff member at the university. The process is depicted in Figure 4.3.2. The diagram illustrates the registration process for a student, but the same protocol applies for any user attempting to register to the system.

It is important to mention that after a user's registration is accepted and tokens are transferred to their wallet, they have the ability to request additional tokens if the initial allocation is depleted. However, to prevent excessive draining of the faucet, a cooldown period has been implemented for each individual user that resets after each token request. This ensures a sustainable distribution of tokens while also providing necessary resources for users to interact with the platform.

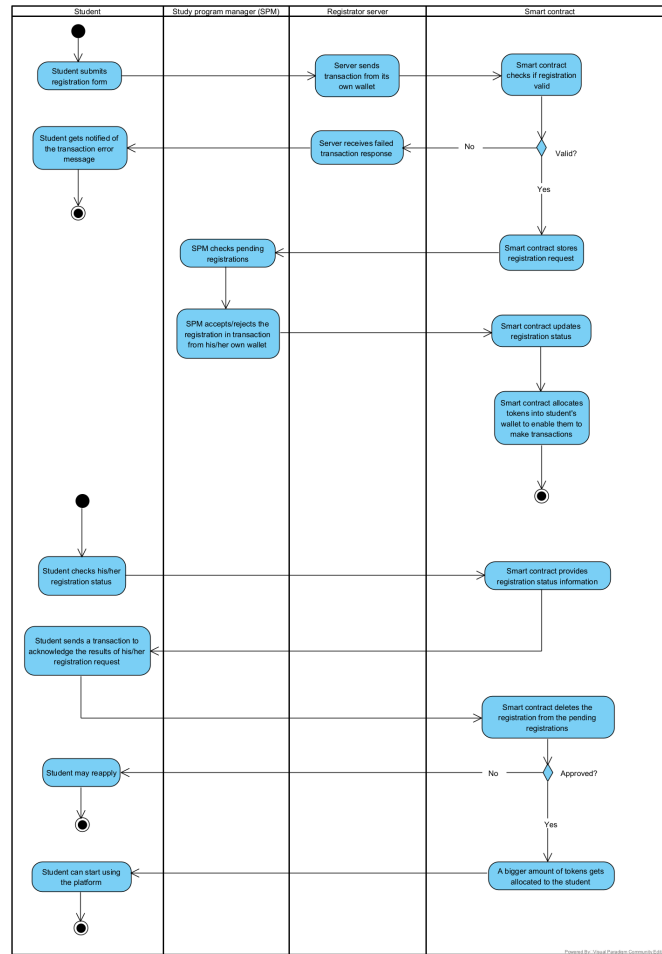


Figure 4.1.: UML activity diagram of the registration process

4.4. Web application

The web application serves as the primary means of interaction for users of the system, providing a user-friendly interface for interacting with the underlying smart contracts. To facilitate this interaction, a server must also be incorporated into the system that the web application can be served from. To enable users to communicate with the smart contracts through their own wallets, a browser plugin is required that is capable of seamlessly integrating with the web application.

The subsequent parts will delve into the structure and design of the user interface.

4.4.1. Login page

Accessible for: users without a connected wallet

Showcase: A.7

4. System design

The page prompts users to log into their MetaMask wallet, as the platform requires a connected wallet to communicate with smart contracts and identify users.

4.4.2. Registration

Accessible for: unregistered users

Showcase: A.7

After logging into the wallet, unregistered users will be presented with a registration form.

4.4.3. Registration (after the registration request is sent)

Accessible for: unregistered users with a pending registration

Showcase: A.7

This page allows users to follow the status of their registration as it is reviewed by an administrator. Once reviewed, users can finalize their registration on this page.

4.4.4. Pending registrations

Accessible for: study program managers

Showcase: A.7

Study program managers can review and accept or reject pending registrations.

4.4.5. Home

Accessible for: registered users

Showcase: A.7

This is the home page for registered and logged-in users, displaying their basic profile information.

4.4.6. Profile

Accessible for: registered users

Showcase: A.7

This page displays detailed profile information and provides users with the option to request more tokens if needed.

4.4.7. All courses

Accessible for: users with a connected wallet

Showcase: A.7

Courses are grouped by study program and study programs are grouped by semester. By opening a course, users can view its details, including multiple assessments, which can be either exams or submissions. For students: students can register for or deregister from courses and register for or deregister from assessments (if the assessments belong to a course that does not have continuous assessment).

4.4.8. My courses

Accessible for: students and lecturers

Showcase: A.7, A.7

Courses are grouped by study program and semester in the same way as in the All Courses page. For students: de-registration for courses and assessments is possible here. For lecturers: the courses the lecturer is teaching, with a list of participants available in the "Registered People/Places" field.

4.4.9. All study programs

Accessible for: users with a connected wallet

Showcase: A.7

A list of all study programs available on the platform.

4.4.10. My study programs

Accessible for: students

Showcase: A.7

A list of study programs that the student is enrolled in.

4.4.11. Add student to course

Accessible for: study program managers

Showcase: A.7

Study program managers have the ability to add students to courses, even if the students cannot register themselves due to violations of requirements (e.g. the course is full or the student is not enrolled in the correct study program).

4.4.12. Create new study program

Accessible for: study program managers

Showcase: A.7

Study program managers can add new courses to the system.

4.4.13. Your study performances

Accessible for: students

Showcase: A.7, A.7

Students can view their courses with the assessments they are registered for (for continuous assessment courses, all assessments; for non-continuous assessment courses, only the assessments they have explicitly registered for). Students can view a performance summary for a given course and their course grade. By opening an exam assessment, the attendance status and exam evaluation are displayed. By opening a submission assessment, users can view the evaluation and upload new documents or access previously uploaded submission documents.

4. System design

4.4.14. Assessment performances

Accessible for: lecturers

Showcase: A.7, A.7, A.7

Courses the lecturer is teaching are grouped the same way as on other pages. Within a course, assessments are listed and in an assessment, the participants are displayed. By opening a participant's information, the evaluation and attendance/submission data is presented. The lecturer can confirm the student's attendance or non-attendance in an exam and download submitted files of a submission. Finally, the lecturer can create or modify the evaluation of the assessments.

4.4.15. Grading

Accessible for: lecturers

Showcase: A.7, A.7

In this section, courses taught by the lecturer are grouped in the same manner as other pages. Within each course, a list of participants can be viewed in an accordion format. By examining a participant, their performance summary is displayed, including total points and grade. Additionally, to provide the lecturer with a comprehensive understanding of the student's performance in the course, all assessments taken by the student are aggregated here. Both the Assessment Performances and Grading pages have a button for updating the student performances of the selected course. This feature automatically evaluates late or unuploaded submissions and missed exams with 0 points, and then calculates a grade based on the course's grading criteria and minimum assessment requirements.

4.4.16. Create new course

Accessible for: lecturers

Showcase: A.7, A.7, A.7

Lecturers are provided with an in-depth form to enable them to create new courses.

5. Implementation

In this section, we will discuss the implementation of our proposed system and highlight some of the key challenges and their solutions. We will also provide a detailed overview of the technologies used and explain how the prototype can be started.

5.1. Overview of technology stack

The blockchain network for this prototype was implemented using **Geth**, an open-source Ethereum execution client written in the Go programming language that is capable of handling transactions, deployment and execution of smart contracts and contains an embedded computer known as the Ethereum Virtual Machine (EVM) [get]. Geth offers a command-line interface and software development kit, enabling developers to create, manage, and interact with Ethereum accounts, as well as send and receive transactions and interact with smart contracts on the Ethereum blockchain. It also includes a built-in miner for creating new blocks and adding them to the blockchain, as well as APIs for developers to interact with the blockchain, including JSON-RPC, which allows for communication with the Geth client via HTTP.

The smart contracts were written in **Solidity**, which is a contract-oriented programming language for writing smart contracts on the compatible blockchains. Solidity contracts are compiled into bytecode, which is then deployed to the blockchain and executed by the EVM.

The smart contract development environment was provided by **Hardhat**, an open-source development environment that offers a set of tools and utilities to help developers test, deploy, and debug their smart contracts on the Ethereum blockchain in a local and safe environment [hara]. The scripts for managing the smart contract were written in JavaScript and utilized the libraries offered by Hardhat, such as ethers.js.

The frontend, registrator, and file server were developed using the Typescript programming language and **Node.js** runtime environment. To facilitate management and maintenance of these codebases, **Webpack** [web] was utilized as a JavaScript (TypeScript) module bundler. Each module contains a piece of code that can be imported and used by other modules. Webpack takes all of these modules and their dependencies and creates a bundle, which is a single file or a few files that contain all of the code needed to run the application. Webpack also provides a powerful plugin system that allows developers to add additional functionality to the bundling process. It is usually utilized for client-side applications but can be used for server-side applications as well. This made working with and compiling Typescript code much more streamlined by employing the plugin **ts-loader** (for the server-side apps) and **babel-loader** (for the frontend app).

5. Implementation

The frontend application was built using the JavaScript library **React**, which enables the user interface to be broken down into small components, such as buttons, forms, and sections. Each component is a self-contained piece of code that handles its own state and logic, and can be easily reused and composed to build a larger user interface. React uses a virtual DOM to efficiently update the user interface, only changing the parts of the page that need to be updated when the component's state or data changes.

The **ethers.js** [etha] library was utilized for interaction between the applications and the blockchain. This library provides a simple and user-friendly API for developers to interact with smart contracts and perform common tasks such as sending transactions, querying the blockchain, and interacting with smart contracts. The use of TypeScript definitions in the library allows for straightforward integration into a Typescript project.

The **npm** package manager was employed for the management of project dependencies.

The **MetaMask** browser extension was used to facilitate user interaction with the blockchain through their web browser, acting as a bridge between the browser and the network [meta].

The configuration of system components was managed through the use of **.env** files located in the root directory of each project.

In order to establish a reliable and easy-to-use deployment process, **Docker** was utilized to containerize the system components. The blockchain, file server, registrator server, and frontend server were developed with their own Docker setup that allowed us to test them separately and integrate them into the overall system.

The development of this project was carried out using the **Visual Studio Code** IDE, with the use of a code formatter plugin, **prettier** [Pre], to ensure code consistency and readability.

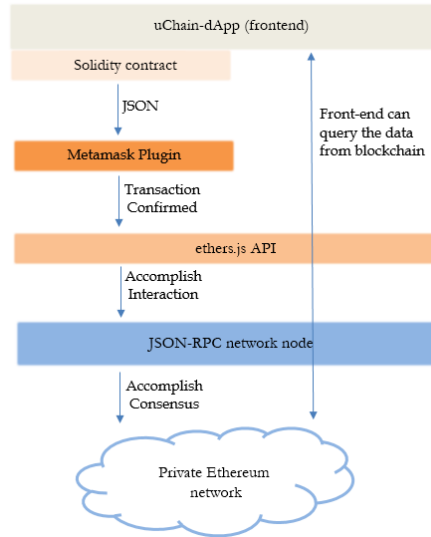


Figure 5.1.: Execution flow of the used technologies based on Figure 4 of [ACB20]

5.2. Blockchain network

The project can be found under `uchain/` in the source code repository.

Geth provides the tools for us to create our own private blockchain. The plan was to set up two signer nodes for validating transactions and adding new blocks, a client node for receiving and propagating transactions from outside of the network towards to signer nodes, and a bootnode that helps the nodes finding each other. We also had to keep in mind that a wallet for the admin account (used for contract deployment) and a wallet for the registrator server (used to make transactions for the users) were required to be prefunded with tokens.

5.2.1. Account generation

The first step was the generate the signer accounts and accounts for prefunding. This was executed using the `geth -datadir . account new` command. The created accounts' passwords, public and private keys were stored.

5.2.2. Creating genesis.json

The next step was to generate a `genesis.json` file that allows the configuration of the network via its inclusion into the genesis block. The process of creating a genesis specification file was facilitated through the use of the Geth tool `puppeth`, which offers a command-line interface for ease of configuration. The specific CLI commands utilized in the creation of the `genesis.json` file is presented in Appendix A.4, the following section adds explanations for the configuration steps:

- Geth offers for private networks the consensus mechanisms Ethash and Clique out of the box, with the latter being the PoA consensus we chose to apply in our network.
- Since the block period length correlates with the waiting time for a transaction to be validated, we wanted it to be reasonably short, although the shorter the block period, the faster the size of the blockchain increases.
- The two sealer addresses are the addresses of the two signer accounts.
- The two pre-funded are the addresses of the admin and registrator accounts.

5.2.3. Setting up bootnode

Geth also provides preconfigured bootnodes that can be accessed via the `bootnode` command. In order to start the signer and client nodes, the enode ID of the bootnode is required. The bootnode requires the `boot.key` file that was generated with the command `bootnode -genkey boot.key`. The bootnode was started and the printed enode ID was recorded in the `.env` file. As long as the bootnode is started with the same `boot.key` file, the enode ID remains the same.

5. Implementation

5.2.4. Network deployment

The necessary commands to start the different types of nodes were defined in Dockerfiles (`uchain/Dockerfile-rpcnode`, `uchain/Dockerfile-bootnode`, `uchain/Dockerfile-signernode`), and a docker-compose file (`uchain/docker-compose.yml`) was utilized to assemble the network. The use of a "volume" [doc] provided by Docker enabled the persistence of data between network restarts, allowing for the blockchain network to be started and restarted without loss of previously generated blockchain data.

5.3. Smart contracts

As presented in section "System design", the smart contracts are designed using the MVC architecture. The Model component was further divided into a data manager and storage layer. The flow of data can be understood in terms of a hierarchical structure, with three distinct layers: the top, middle and bottom layers. The top layer, which the user directly transacts with, is represented by smart contracts on the View-Controller axis. Then the data flows between the top layer and the middle layer which is the layer of the Model's data manager contracts. Finally, there is a connection between the middle and bottom layer, the layer of storage contracts.

5.3.1. Stages of input validation

To ensure the principle of separation of concerns, user input validation is performed at each of these hierarchical layers. The top layer, which focuses on the logic of the system, validates input from a business logic perspective. The middle layer, which mediates the flow of data between the top and bottom layers, checks for type correctness of input. Finally, the bottom layer, responsible for data storage, validates input with a focus on maintaining data consistency, similar to how a conventional system would use a database.

The example of adding a submission demonstrates this concept with comments:

1. Top layer - logic checks

```
PerformanceController.addSubmission(uint256 assessmentId, string[]  
calldata documentHashes) external onlyStudent
```

- the sender must be a student
- student must be registered for the assessment they are trying to add the submission to
- the assessment they are trying to add the submission to must be of type submission (not exam)
- the time of the transaction cannot be after the submission deadline

2. Middle layer - data correctness checks

```
PerformanceDataManager.setOrOverrideSubmission(uint256 uId,
```

```
uint256 assessmentId, uint256 timestamp, string[] calldata
documentHashes) external onlyWhitelisted
```

- sender must be a whitelisted address
- provided list of hashes must not be empty
- none of the list elements are allowed to be "" string

3. Bottom layer - consistency checks

```
SubmissionStorage.storeSubmission(uint256 uId, uint256
assessmentId, PerformanceDataTypes.Submission calldata submission)
external onlyWhitelisted
```

- the uID must be valid (not 0)
- the assessment ID must be valid (not 0)
- submission for the given assessment ID should not be set

5.3.2. Access validation

An emphasis on validation, especially on access validation was deemed necessary due to the direct accessibility of smart contracts to tech-savvy users, meaning transactions can be sent without the mediation of the frontend, which could potentially lead to unauthorized access (e.g. student tries to submit a better grade for herself/himself).

The access validation has two aspects:

- **User role validation** was implemented in the top layer of the contracts to ensure that only authorized users, based on their registered role (e.g. student, lecturer), could access the system.
- **Whitelist validation** was implemented in the middle and bottom layers to restrict accessibility to only contracts within the top layer, given the fact that only the contracts of the top layer are meant to be directly communicated with by users.

User role validation

User role validation is realized by the contract `UserAccessController` (`smartcontracts/contracts/logic/UserAccessController.sol`). It defines the modifiers `onlyRegistered`, `onlyLecturer`, `onlyStudent`, `onlySPM`, `onlyLecturerOrSpm` that validate based on the sender of the transaction. These modified are used in the Controller and View contracts.

Whitelist validation

The whitelist validation system operates on two layers: data managers are only accessible to controller and view contracts, while storage contracts are only accessible to data managers.

5. Implementation

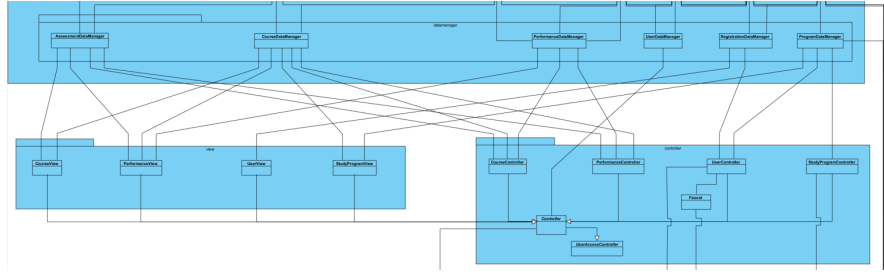


Figure 5.2.: Excerpt from Figure A.2 (Appendix), showing the data manager, controller and view contracts

One approach to implementing a whitelist for access would have been to provide a list of permitted access contracts for each contract requiring restricted access. However, this method would quickly become impractical with a large number of contracts (e.g. 6 data managers, 6 storages results in $6*6=36$ dependencies). To address this issue, a dedicated contract called **AccessWhitelist** (`smartcontracts/contracts/accesscontrol/AccessWhitelist.sol`) was created for each layer (one for the data managers, one for the storages). These contracts are responsible for keeping track of all the contract addresses that the given layer should provide access to (see dependency arrows starting from the **AccessWhitelist** instances in the Appendix on Figure A.2).

By utilizing the **AccessWhitelist** contracts, contracts only need to receive the address of the corresponding **AccessWhitelist** contract. In the **AccessController** (`smartcontracts/contracts/accesscontrol/AccessController.sol`) abstract contract, that the data manager and storage contracts inherit from, the modifier **onlyWhitelisted** is used to retrieve from the provided **AccessWhitelist** contract whether the caller is whitelisted according to its records. This approach effectively addresses the scalability issues associated with maintaining a whitelist for access in a large-scale contract system.

5.3.3. Challenge of contract dependencies

In addition to the scalability challenges encountered in access validation, the contract deployment process also presented a need for an efficient solution. Specifically, the requirement for providing each contract of the top layer with the addresses of the appropriate contracts of the data manager layer presented a scalability issue. As seen on Figure 5.2, each Controller and View contract must be aware of the addresses of multiple data manager contracts that they intend to use, which results in a large number of dependencies.

To address this problem, a contract called **AddressBook** (`smartcontracts/contracts/addressbook/AddressBook.sol`) was developed. This contract is responsible for maintaining a record of the addresses of the data manager contracts. By using the **AddressBook** contract, instead of providing multiple data manager addresses for each Controller and View contract, providing only the address of the **AddressBook** contract is sufficient. The abstract contract **AddressBookUser** (`smartcontracts/contracts/addressbook/`

`AddressBookUser.sol`), that the Controller and View contracts are inheriting from, implements a method that allows for the retrieval of the selected contract's address from the `AddressBook` contract. This resolves the high level of direct connectedness between the contracts (see dependency arrows starting from the `AddressBook` instance in the Appendix on Figure A.2)). Additionally, this system allows easy changing of contracts since the new contract's address should only be registered at the `AddressBook` contract.

5.3.4. Automatic grade calculation

To address the challenge of automatically grading courses, a solution was implemented in the form of the `updateCourseParticipantPerformances` function in contract `smartcontracts/contracts/logic/PerformanceController.sol`. This function is responsible for updating assessment evaluations and calculating grades for students in a course. However, a unique aspect of this function is its ability to handle the different grade calculation protocol of a course without continuous assessment (course type VO - "Vorlesung"). Normally, a course requires all of its assessments to be evaluated in order to calculate the grade but a VO allows students to register for the assessments (exam appointments) at their discretion, and only the last exam that the student registered for is considered when calculating their grade. This necessitates a specialized logic for grade calculation within the `updateCourseParticipantPerformances` function. Further information on the implementation of this function can be found in the source code comments.

5.3.5. Deployer

In order to streamline the deployment process of the smart contracts, a `Deployer` (`smartcontracts/contracts/Deployer.sol`) contract was developed that minimizes the number of transactions required to deploy and configure the system. To maintain a manageable size for the `Deployer` contract, it does not import all of the contracts that should be deployed. Instead, it expects the bytecode of the contracts to be provided as function arguments (see example function interface 5.3.5). Using these bytecodes, the `Deployer` contract performs the deployment and stores the address of the created contracts for future reference. This approach allows for a more efficient deployment process while minimizing the size of the `Deployer` contract.

```
1 function deployControllers(  
2     bytes memory courseControllerContract ,  
3     bytes memory performanceControllerContract ,  
4     bytes memory studyProgramControllerContract ,  
5     bytes memory userControllerContract ,  
6     bytes memory faucetContract  
7 ) external onlyOwner;
```

5. Implementation

5.4. Registrator server

The registrator server is created with the Express.js framework. It is able to access its own wallet by providing it with the private key of the wallet. It utilizes ethers.js to communicate with the `UserController` contract, which offers a function for registering users by their provided wallet addresses and profile data.

The registrator server has a single API endpoint `/registration` that accepts the data required for registration. It converts the received data into the format requested by the smart contract, sends the transaction to the network's client node, and waits for the response from the blockchain. If the transaction is successful, it returns a response with status 200. If there is an error signaled by the blockchain, such as a reverted function call, it returns a response with status 500 along with the error message from the smart contract.

5.5. File server

The file server is implemented using the Express.js framework and utilizes the `multer` library [Exp] as middleware for handling uploaded files. Multiple files can be uploaded at once at the `/uploads` API endpoint. Multer is configured to store the data on the server machine's disk. The stored files are collected in a directory that is made available through its own endpoint `/files/:filename`. The uploaded files can be retrieved by calling this endpoint with the requested file's name. In order to prevent file with the same name from being overridden and to ensure privacy for the uploaded files, a dedicated function is used to generate a new name for the uploaded file before it is stored. This function employs the `crypto` library to generate the md5 hash of the uploaded file's name combined with the time of upload. The generated names of the stored files are returned in the response, allowing for easy retrieval and identification of the stored files.

5.6. Frontend

The frontend of the system was developed using the React framework. A single-page application architecture was implemented, which utilizes client-side routing. In order to realize the application's components, functional components that incorporate hooks for state management were utilized.

5.6.1. Web3

To bootstrap to implementation of the web3-specific requirements, the frontend is implemented using the `web3-react` framework library developed by Uniswap [Unia]. Web3-react provides a set of higher-order components and hooks that handle connecting to an Ethereum node, managing accounts, and interacting with smart contracts. The library also includes features for handling common tasks such as querying the blockchain for contract state, sending transactions, and handling errors. It offers easy integration with web3

libraries such as `ethers.js` that our project utilized. The use of `web3-react` allowed us to focus on building the application's specific functionality, without having to worry about the details of connecting to the Ethereum network. Additionally, it also provides a way to handle web3 injection from browser extensions like MetaMask, making it easy to interact with the blockchain from a web app.

5.6.2. Component library

The `material-ui` library [mui] was employed as the web application's component library. This library allowed the use of Material-UI components with React. `material-ui` is a popular UI library for building user interfaces that are consistent with the Material Design guidelines developed by Google.

The library `react-hook-form-mui` helped managing forms throughout the application. It provides integration between the library `react-hook-form` and `material-ui`. `React-hook-form` is a library that provides a way to manage form inputs and validation in React by offering Material-UI form components with the added benefit of validation and error handling provided by `react-hook-form`.

5.6.3. User authorization hook

The user interface employs a centralized approach for user authentication throughout the platform. To achieve this, a custom hook called `useAuthStore` was developed using the `zustand` library [Poi]. `zustand` is a state management tool for React that allows for easy sharing of state across components without the need for a context provider. This way, the shared state can be managed by a single component called `UserAuth` via the `useAuthStore` hook, and the same hook provides the necessary data for authorization for the other components throughout the application. This eliminates the need for reauthorizing the user's wallet account in each self-contained component. Additionally, the hook includes a function for reauthorization, which can be called from any component in the application to update the user's authorization status. This approach allows for a seamless user experience and efficient management of user authentication.

5.6.4. Displaying error messages

A centralized approach, based on the same concept used for user authorization, is utilized for displaying error messages. The `zustand` library is applied in this case as well, by creating a hook called `useErrorStore`, which stores the latest error message in its state. The `ErrorAlert` component utilizes this hook and displays the error message when a new message is set through the hook. Other components in the application can use the `setErrorMessage` function of the hook to update the error message if an error occurs, providing a consistent and centralized approach for handling and displaying error messages throughout the application.

5. Implementation

5.6.5. Dynamic loading of accordion content

The user interface frequently employs accordions in its layout. During the implementation of these accordions the optimization of the accordion's content loading process was also considered. The `CustomAccordion` component was created to handle this optimization by only loading the content of the accordion once it is opened for the first time.

This is achieved through the use of a parameter called `signalLoad` in `CustomAccordion`, which is a function of type `() => void`. This function is called when the accordion is opened for the first time.

To provide the function for the `signalLoad` parameter, a custom hook called `useLoadSignal` was created. This hook returns a boolean state variable `loaded` and a function of type `() => void` that is used to set the `loaded` variable to true once invoked. The `useLoadSignal` hook is utilized in the dedicated accordion components, such as in the `StudyProgramAccordion`, and the generated `loaded` variable is then utilized to render a placeholder component in the accordion until the `loaded` value changes to true, at which point the actual components are rendered. This allows for efficient loading and improves the overall user experience.

5.7. Overview of the deployed system

The deployment of the system components in the prototype setup is depicted in Figure A.2 (Appendix). Since the prototype utilizes Docker containerization, the artifacts on the diagram represent the deployed containers and their corresponding ports for communication.

The system comprises of four Geth-powered containers running the blockchain, one serving as a bootnode, and two others as signer nodes, while the remaining one serves as the client (RPC) node. By connecting to the bootnode, the signer and client nodes can establish a peer-to-peer network with each other. The client node is the only node that can be accessed externally through a JSON-RPC port.

Additionally, the diagram includes the servers for registrator, file and frontend, all of which are powered by Node.js. The registrator server has a direct connection to the client node of the blockchain. These servers interact with the users through HTTP ports.

5.8. How to run the prototype

For details see `README.md` in the source code repository's root directory.

6. Evaluation and discussion

During the evaluation of the system, the ability to execute the defined use cases was used as a measure for determining if the goals of the project had been met and if the system could be suitable for implementation in real-world settings. These use cases, previously outlined in section "System requirements", were tested in two ways. Firstly, the system was tested programmatically through the use of test cases, which allowed for a systematic evaluation of the system's functionality and performance. Additionally, user testing was conducted to evaluate the system's performance in real-world usage scenarios. This approach allowed for an assessment of the system's ability to handle the unpredictable nature of human interaction and to identify any issues that may arise in actual usage.

6.1. Sample data

With the goal of demonstrating the system's ability to perform in a wide range of situations outlined in the system requirements, a set of sample data and test wallets were created and made available in the source code repository under `smartcontracts/samples` and `smartcontracts/secrets/testwallets.json`. The data entries are completely fictional, with names generated by a random name generator [rana] and phone numbers generated by a phone number generator [ranb]. The rest of the data was created with the goal of providing comprehensive demonstration scenarios. To restrict certain actions within specific time frames, the dates were defined as relative values in days rather than absolute dates. This allows for the responsible script to convert these relative values into absolute dates by adjusting the current date by the relative value (e.g. relative value of +5 results in a date 5 days after than the current date). This way, the intended time periods are always up-to-date.

To load this sample data into the system, a script located at `smartcontracts/scripts/loadSampleData.js` was created. This script interacts directly with the smart contract data managers, bypassing the more complex controller and view level of the smart contract architecture. It is important to note that the script had to be granted access to the admin wallet that deployed the smart contracts in order to function properly. Given the number of transaction that are required to correctly set each piece of data, the process of uploading the sample data may take a significant amount of time - multiple minutes, depending on the block period configuration.

6.2. Programmatic testing

In order to systematically evaluate the functionality of the system, programmatic testing was conducted on the smart contracts that form the core logic of the system. Since the smart contracts were developed using the hardhat development environment, the testing framework provided by hardhat was an obvious choice for creating test cases. The hardhat network, a local Ethereum network, is facilitated by the testing framework and the ethers.js library was utilized for communication with the network [Harb]. The Mocha test framework provided by hardhat was employed for running the tests [Harb]. The test cases are located in the `smartcontracts/test` directory of the source code repository.

Since these tests were meant to evaluate the smart contracts' performance on a high level, the system's capabilities were tested through the surface level components only which are the controller and view contracts.

6.2.1. Test setup and test cases

As previously discussed in the "Implementation" section, a specialized deployer smart contract was created to facilitate the deployment process and ensure correct configuration of the smart contracts. The same deployer smart contract was intended to be used during this test deployment as well. Before deploying the system, it was crucial to test the deployer smart contract to ensure that it was capable of performing its intended task. To accomplish this, a dedicated testing script was designed and executed. This script specifically targeted the deployer smart contract and evaluated its ability to successfully deploy the smart contract system. It utilized a function `deploySystem` that initiated the deployment process via the deployer contract. Through this process, we were able to confirm that the deployer smart contract was fully functional and ready for use in the deployment of the system.

Since we already made sure that the deployer contract was working as intended, we could utilize it for the deployment of the rest of the smart contracts. With the help of hardhat's `loadFixture` function, we were able to define a starting state for the network and its smart contracts, which was then restored before executing each test case. This starting state consisted of performing the deployment via the function `deploySystem`.

The test cases were organized according to the use cases they were testing, ensuring a logical flow and clear distinction between different test scenarios. The testing process consisted of several stages, starting with the preparation stage where a specific state was established by loading a set of data into the system. This was followed by the execution of the main state alteration, which was the primary action being tested. Our testing approach followed this flow of execution with test subcases, which were used to test each step along the way and ensure the overall integrity of the test process.

We tested not only the ideal scenarios of each use case but incorporated a selection of edge cases that the system was expected to handle.

6.2.2. Running the tests

The tests can be run by navigating into the `smartcontracts` directory of the source code repository and executing the `npm hardhat test` command. This command invokes Hardhat's testing framework, which automatically compiles the smart contracts if they weren't compiled before.

6.2.3. Results

The output of the test cases is available in Appendix A.3. The system was able to pass all 68 test cases, including the tests dealing with edge cases.

6.3. User testing

User testing is an essential part of evaluating the overall performance and usability of the system. In order to gain a comprehensive understanding of how the system behaves from the perspective of the end-user, user testing was conducted. This method of evaluation allows us to assess the system's capability as a whole, evaluating the integration of the different system components such as smart contracts, frontend, file server, and registrator server. Furthermore, user testing also puts the system under pressure in terms of usability and robustness, allowing us to identify any potential issues or areas for improvement. The testing process involved recruiting a sample of users who were representative of the target user group and providing them with tasks to complete using the system. The users were then observed and their feedback was collected to evaluate the system's overall performance and usability.

6.3.1. Selection of testers

The testing process was conducted with the participation of three individuals from diverse backgrounds. The testers were selected to represent a range of potential age demographics of the platform's users. The first tester, referenced as 24F, was a female studying pharmacy at the University of Vienna in her first semester of a master's program. The second tester, 36F, was a working professional in the logistics industry with a degree in bachelor of arts. The third tester, 60M, was a course instructor at the University of Szeged.

6.3.2. Test setup and user tasks

The tests were performed in 1-on-1 sessions with 3 test subjects. The test subjects were provided with a computer running the system's components locally, with the MetaMask extension [Metb] pre-installed and 8 test wallet accounts pre-imported and named according to their naming in `testwallets.json`. The MetaMask wallet contained a ninth account as well which was the default account. The uChain platform's home page was opened in the browser.

Before each session, the system was restarted, freshly deployed and sample data was loaded into the system. Because this testing was not performed with the intention of

6. Evaluation and discussion

evaluating the test subject's ability to operate a cryptocurrency wallet, to ensure that all subjects had a basic understanding of the used wallet extension, a brief introduction on how to interact with the MetaMask wallet was provided. The textual content of this introduction can be found in Appendix A.5.

The subjects were then instructed to carry out the defined use cases in sequential order, including the necessary switching between the MetaMask wallet's accounts. The protocol of the user tasks can be found in Appendix A.6.

6.3.3. Results and feedback

During the testing phase, all of the required test cases were successfully executed by the test subjects. At first, operating the platform felt unusual for the testers but after executing the first few transactions, they got used to it and were able to use it by their own.

6.4. Discussion of results

The results of the programmatic testing indicate that the core system component, the smart contracts, are able to effectively meet the requirements of the system and handle edge cases.

User testing revealed that the platform is usable by a wide range of users, with the provision of appropriate instructions. This suggests that the platform could serve as a viable alternative to the conventional, non-blockchain platforms currently used by the university, such as Moodle or u:space.

However, it is important to note that the project heavily relies on direct communication between the user and the contracts, which requires a level of familiarity with interacting with such systems, such as operating a digital wallet like MetaMask. In a real-world scenario, it would not be reasonable to assume this level of familiarity among all users, thus a more comprehensive educational process with appropriate tutorials would be necessary.

Overall, the system demonstrated its ability to successfully execute all required test cases and provide a satisfactory user experience.

7. Conclusion and future work

This project was aiming for exploring the capabilities and applicability of a blockchain-based study performance and evaluation pass in the university administration.

We focused on developing and deploying a private blockchain network that uses the proof of authority consensus mechanism, and designing a system of smart contracts that represent the core of the system. Through experimentation, we were able to gain insights on how to effectively assemble a system using a multitude of smart contracts, and how to organize the code within these contracts. Additionally, we explored data validation concepts to ensure the integrity of the information stored on the blockchain. We solved blockchain specific challenges by designing and integrating off-chain system components into the project. Ultimately, we were able to build a web3 web application that offers a reasonable user experience for interacting with the system.

In terms of future work, there are several avenues that could be explored to enhance the capabilities of the platform developed in this project. One potential area of focus could be expanding the use cases of the system beyond the scope of this project, in order to discover new and innovative ways in which it can be utilized.

Additionally, investigating a large scale deployment of the system could provide valuable insights into the real-world applicability of the platform.

Another important aspect to consider is evaluating the scalability requirements and capabilities of the blockchain technology used in this project, specifically in terms of maximum number of processed transactions per second.

Finally, further research could be conducted to explore ways in which the privacy of the data stored on the blockchain could be improved. A study by Hongzhi et al. [LHLC19] proposed a blockchain-based storage and sharing scheme for educational records scheme that utilizes a combination of blockchain technology, storage servers, and cryptography techniques to create a secure and reliable environment for the storage and sharing of educational records. The blockchain is employed to ensure the security and reliability of data storage, while smart contracts are utilized to regulate the process of storage and sharing. As an extension of our own research, it would be worthwhile to investigate the potential benefits of integrating a similar concept into our system, with the goal of increasing data privacy and security.

Overall, the findings of this project demonstrate the potential for blockchain technology to be applied in the field of university administration, and provide valuable insights for future developments in this area.

Bibliography

- [ACB20] Antonio Wellington S. Abreu, Emanuel F. Coutinho, and Carla I. M. Bezerra. A blockchain-based architecture for query and registration of student degree certificates. SBCARS '20, page 151–160, New York, NY, USA, 2020. Association for Computing Machinery.
- [B.S18] Chaithra B.S. An innovative information system for college management. *International Journal of Management, Technology, and Social Sciences*, pages 140–145, 06 2018.
- [But13] Vitalik Buterin. Ethereum white paper: A next generation smart contract & decentralized application platform. 2013.
- [Coi] CoinDesk. What is proof-of-authority? <https://www.coindesk.com/learn/what-is-proof-of-authority/>. Accessed: 2022-01-25.
- [Cry] Crypto Market Pool. Create a crypto faucet using a solidity smart contract. <https://cryptomarketpool.com/create-a-crypto-faucet-using-a-solidity-smart-contract/>. Accessed: 2022-01-25.
- [doc] Volumes - docker. <https://docs.docker.com/storage/volumes/>. Accessed: 2022-01-26.
- [etha] Ethers v5 documentation. <https://docs.ethers.org/v5/>. Accessed: 2022-01-26.
- [ethb] ethereum.org. Gas and fees. <https://ethereum.org/en/developers/docs/gas/>. Accessed: 2022-01-25.
- [Evg] Evgenia Kuzmenko. Model-view-controller architecture pattern: Usage, advantages, examples. <https://hackernoon.com/model-view-controller-a-architecture-pattern-usage-advantages-examples>. Accessed: 2022-01-25.
- [Exp] ExpressJs. Multer. <https://www.npmjs.com/package/multer/>. Accessed: 2022-01-26.
- [get] Geth - private networks. <https://geth.ethereum.org/docs/fundamentals/private-network/>. Accessed: 2022-01-26.
- [hara] Hardhat documentation. <https://hardhat.org/docs>. Accessed: 2022-01-26.

Bibliography

- [Harb] Hardhat. Hardhat - contract testing. <https://hardhat.org/tutorial/testing-contracts>. Accessed: 2022-01-22.
- [Inv] Investopedia. Public, private, permissioned blockchains compared. <https://www.investopedia.com/news/public-private-permissioned-blockchains-compared/>. Accessed: 2022-01-25.
- [LHLC19] Hongzhi Li, Dezhi Han, Kuan-Ching Li, and Arcangelo Castiglione. Edurss:a blockchain-based educational records secure storage and sharing scheme. *IEEE Access*, PP:1–1, 11 2019.
- [MD21] Siphamandla Mjoli and Nomusa Dlodlo. Exploring the integration of blockchain technology and iot in a smart university application architecture. In *2021 International Symposium on Electrical, Electronics and Information Engineering*, ISEEIE 2021, page 301–306, New York, NY, USA, 2021. Association for Computing Machinery.
- [meta] Metamask docs. <https://docs.metamask.io/guide/#why-metamask/>. Accessed: 2022-01-26.
- [Metb] MetaMask. Metamask installation. <https://chrome.google.com/webstore/detail/metamask/nkbihfbeogaeaoehlefnkodbefgpgknn?hl=en>. Accessed: 2022-01-22.
- [Moz] Mozilla. Mvc. <https://developer.mozilla.org/en-US/docs/Glossary/MVC/>. Accessed: 2022-01-25.
- [mui] material-ui. <https://github.com/mui/material-ui/>. Accessed: 2022-01-27.
- [Nak09] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. May 2009.
- [PKLP20] Suporn Pongnumkul, Chanop Khonnasee, Swiss Lertpattanasak, and Chantri Polprasert. Proof-of-concept (poc) of land mortgaging process in blockchain-based land registration system of thailand. In *Proceedings of the 2020 The 2nd International Conference on Blockchain Technology*, ICBCT'20, page 100–104, New York, NY, USA, 2020. Association for Computing Machinery.
- [Poi] Poimandres. zustand. <https://github.com/pmndrs/zustand/>. Accessed: 2022-01-27.
- [Pre] Prettier. Prettier - code formatter. <https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode/>. Accessed: 2022-01-26.
- [rana] Random name generator. <https://randomwordgenerator.com/name.php>. Accessed: 2022-01-15.
- [ranb] Random phone number generator. <https://randommer.io/Phone>. Accessed: 2022-01-15.

- [sma] What are smart contracts on blockchain? <https://www.ibm.com/topics/smart-contracts>. Accessed: 2022-01-26.
- [Sza94] Nick Szabo. Smart contracts. <https://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/L0Twinterschool2006/szabo.best.vwh.net/smart.contracts.html>, 1994. Accessed: 2022-01-19.
- [Unia] Uniswap. web3-react. <https://github.com/Uniswap/web3-react/tree/v6/>. Accessed: 2022-01-27.
- [Unib] University of Vienna. Moodle - upload tutorial. <https://youtu.be/0dNEJlCaDyM?t=24>. Accessed: 2022-01-22.
- [Unic] University of Vienna. u:find - bachelor computer science 2023ss. https://ufind.univie.ac.at/en/vvz_sub.html?semester=2023S&path=292229. Accessed: 2022-01-21.
- [Unid] University of Vienna. u:space - grading tutorial. https://wiki.univie.ac.at/download/attachments/101505045/Noten%C3%BCBernahme_uspace_202004.pdf?version=1&modificationDate=1669993373823&api=v2. Accessed: 2022-01-22.
- [Unie] University of Vienna. Web services. <https://studieren.univie.ac.at/en/web-services-ospace-ufind-moodle-webmail/>. Accessed: 2022-01-21.
- [Unif] University of Vienna. Web services - moodle. <https://zid.univie.ac.at/en/e-learning/>. Accessed: 2022-01-21.
- [web] Webpack documentation. <https://webpack.js.org/concepts/>. Accessed: 2022-01-26.

A. Appendix

A.1. Use cases

Table A.1.: Course registration

ID	1
Name	Register for course
Level	Primary
Brief description	A student selects a course and attempts to register for it. The logic checks if all requirements are met and allows the transaction or denies the transaction.
Prerequisites	<ul style="list-style-type: none">• Being within the registration deadline• Check for the presence of completed prerequisite courses (if required)
Actors	<ul style="list-style-type: none">• Student• Smart contract
Triggering event	A student selects a course and attempts to register for it.
Description	<ol style="list-style-type: none">1. Student initiates registration for a course.2. A smart contract checks if the student meets the necessary requirements and allows registration or provides an error message.3. If no error message is provided, the new block is validated by one or more signing nodes and added to the blockchain.
Postcondition	The transaction was successfully added to the blockchain.

A. Appendix

Table A.2.: Course de-registration

ID	2
Name	De-register from course
Level	Primary
Brief de- scription	A student de-registers from a course that they have previously registered for. The logic checks if all requirements are met and allows the transaction or denies the transaction.
Prerequisites	<ul style="list-style-type: none"> • Student is registered for the course • Being within the de-registration deadline
Actors	<ul style="list-style-type: none"> • Student • Smart contract
Triggering event	A student selects a course that they are already registered for and attempts to de-register from it.
Description	<ol style="list-style-type: none"> 1. Student initiates de-registration from a course. 2. A smart contract checks if the student is registered for the course and is within the unregistration deadline. If one of these two conditions is not met, an error message is provided and no transaction takes place.
Postcondition	The transaction was successfully added to the blockchain.

Table A.3.: Assessment registration

ID	3
Name	Register for assessment
Level	Secondary
Brief description	A student selects an assessment that requires separate registration and attempts to register for it. The logic checks if all requirements are met and allows the transaction or denies the transaction.
Prerequisites	<ul style="list-style-type: none"> • Being within the registration deadline • Assessment requires separate registration • Student is registered for the course
Actors	<ul style="list-style-type: none"> • Student • Smart contract
Triggering event	A student selects an assessment and attempts to register for it.
Description	<ol style="list-style-type: none"> 1. Student selects a course they are registered to. 2. Student selects an assessment that requires separate registration. 3. A smart contract checks if the student meets the necessary requirements and allows registration or provides an error message. 4. If no error message is provided, the new block is validated by one or more signing nodes and added to the blockchain.
Postcondition	The transaction was successfully added to the blockchain.

A. Appendix

Table A.4.: Assessment de-registration

ID	4
Name	De-register from assessment
Level	Secondary
Brief de- scription	A student de-registers from an assessment that they have previously registered for. The logic checks if all requirements are met and allows the transaction or denies the transaction.
Prerequisites	<ul style="list-style-type: none"> • Being within the de-registration deadline • Assessment requires separate registration • Student is registered for the assessment
Actors	<ul style="list-style-type: none"> • Student • Smart contract
Triggering event	A student attempts to register from an assessment.
Description	<ol style="list-style-type: none"> 1. Student selects the assessment they want to de-register from. 2. A smart contract checks if the student meets the necessary requirements and allows de-registration or provides an error message. 3. If no error message is provided, the new block is validated by one or more signing nodes and added to the blockchain.
Postcondition	The transaction was successfully added to the blockchain.

Table A.5.: Submission upload

ID	5
Name	Submit a document
Level	Primary
Brief description	Within the framework of a course, submission deadlines are pre-defined for students to submit documents.
Prerequisites	<ul style="list-style-type: none"> • Student is registered for the submission • Being within the submission deadline
Actors	<ul style="list-style-type: none"> • Student • File server • Smart contract
Triggering event	The student submits a document to be evaluated by the course instructor.
Description	<ol style="list-style-type: none"> 1. The document is uploaded via the web interface and the document gets uploaded to a file server, with only a proof (hash) stored on the blockchain. 2. After a document is submitted, the course instructor has the option to see all information about the submission (e.g. submission date) and download the document.
Postcondition	The student has a recorded submission in the blockchain.

A. Appendix

Table A.6.: Test attendance

ID	6
Name	Confirm test attendance
Level	Primary
Brief description	Throughout the semester, students take tests and the course instructors can confirm whether the student attended the test.
Prerequisites	<ul style="list-style-type: none"> • Student is registered for the test • Course instructor is lecturing at the course
Actors	<ul style="list-style-type: none"> • Student • Course instructor • Smart contract
Triggering event	The course instructor selects a student for attendance confirmation.
Description	<ol style="list-style-type: none"> 1. Course instructor selects the student in the list of test participants on the web interface. 2. The instructor decides if the attended or not attended status should be confirmed for the student. 3. The confirmation transaction is submitted.
Postcondition	Status of attendance / not attendance is recorded in the blockchain.

Table A.7.: Assessment evaluation

ID	7
Name	Evaluate student's assessment
Level	Primary
Brief description	Course instructors can submit evaluations for students' assessment performances.
Prerequisites	<ul style="list-style-type: none"> • Course instructor is lecturing at the course • Student is registered for the assessment
Actors	<ul style="list-style-type: none"> • Student • Course instructor • Smart contract
Triggering event	The course instructor selects a student and an assessment for evaluation.
Description	<ol style="list-style-type: none"> 1. Course instructor selects the student in the list of assessment participants on the web interface. 2. The instructor submits the achieved points and feedback optionally via a form. 3. The evaluation transaction is submitted. This evaluation will be used for determining the final grade.
Postcondition	Evaluation is recorded in the blockchain.

A. Appendix

Table A.8.: Grading

ID	8
Name	Generate overall grade
Level	Primary
Brief description	An automatic evaluation based on the performance throughout the semester takes place. This evaluation is recorded in the student's transcript.
Prerequisites	<ul style="list-style-type: none"> • Course instructor is lecturing at the course • The student is registered for the course
Actors	<ul style="list-style-type: none"> • Student • Course instructor • Smart contract
Triggering event	The course instructor initiates the automatic grading of a course's participants.
Description	<ol style="list-style-type: none"> 1. Course instructor starts the process of automatic grading of course participants on the web interface. 2. The smart contract's grading logic checks the students' assessment performances and generates a grade according to the predefined grading criteria.
Postcondition	The generated grade is recorded in the blockchain.

Table A.9.: Modification of evaluation

ID	9
Name	Modify evaluation
Level	Secondary
Brief description	The course instructor has the ability to modify the student's evaluations retrospectively.
Prerequisites	<ul style="list-style-type: none"> • Existing evaluation of the student
Actors	<ul style="list-style-type: none"> • Student • Course instructor • Smart contract
Triggering event	The course instructor selects a student's evaluation and edits it.
Description	<ol style="list-style-type: none"> 1. Selection of an evaluation that is to be modified. 2. Assigning the new number of points, option to add textual comment. 3. Execution of the smart contract to make the change.
Postcondition	Modified evaluation is stored in the blockchain.

A. Appendix

Table A.10.: Modification of grade

ID	10
Name	Modify grade
Level	Secondary
Brief description	The course instructor has the ability to modify the student's grade retrospectively.
Prerequisites	<ul style="list-style-type: none"> • Existing grade of the student
Actors	<ul style="list-style-type: none"> • Student • Course instructor • Smart contract
Triggering event	The course instructor selects a student's evaluation and edits it.
Description	<ol style="list-style-type: none"> 1. Selection of an grade that is to be modified. 2. Assigning the new number of points, option to add textual comment. 3. Execution of the smart contract to make the change.
Postcondition	Modified grade is stored in the blockchain.

Table A.11.: User registration

ID	11
Name	Register students/course instructors into the system
Level	Primary
Brief description	Students/course instructors register in the system to be able to use its functionalities. The registrations need to be reviewed and accepted by the university administration. The registration includes students/course instructors providing their personal information (name, birthdate, address, etc.).
Prerequisites	<ul style="list-style-type: none"> • The user has created a wallet on the network (e.g. with MetaMask). • The user does not have an existing registration.
Actors	<ul style="list-style-type: none"> • Student/course instructor • Study program manager/administrator • Smart contract
Triggering event	Student/course instructor submits a registration request with their information via the web interface.
Description	<ol style="list-style-type: none"> 1. The student/course instructor fills out the registration form using the web interface and submits it to the smart contract. 2. The smart contract adds the wallet address to the data and the request is stored in the smart contract. 3. An administrator reviews and accepts or rejects the registration. If accepted, the smart contract generates a universal ID and saves it along with the data.
Postcondition	The student/course instructor's status is saved and they can use the system using the registered wallet address/they cannot use the system if the registration was rejected.

A. Appendix

Table A.12.: Creation of new course

ID	12
Name	Create a new course
Level	Primary
Brief description	Course instructor creates a new course.
Prerequisites	<ul style="list-style-type: none"> • Instructor is registered in the system.
Actors	<ul style="list-style-type: none"> • Course instructor • Smart contract
Triggering event	Course instructor opens the form for creating a course on the web interface.
Description	<ol style="list-style-type: none"> 1. The course instructor inputs the descriptions (as in the university's course catalog) of the course using the web interface. 2. The course instructor can add other registered colleagues as instructors. 3. The course instructor defines the registration criteria (if applicable), requirements, grading scale, class dates, number of participants, etc. 4. The transaction is submitted and the smart contract stores this data.
Postcondition	The new course is visible to students through the appropriate smart contract method.

Table A.13.: Creation of new study program

ID	13
Name	Create a new study program
Level	Secondary
Brief description	Study program manager creates a new study program.
Prerequisites	<ul style="list-style-type: none"> • Study program manager is registered in the system.
Actors	<ul style="list-style-type: none"> • Study program manager • Smart contract
Triggering event	Study program manager opens the form for creating a study program on the web interface.
Description	<ol style="list-style-type: none"> 1. The Study program manager inputs the name of the study program using the web interface. 2. The transaction is submitted and the smart contract stores this data.
Postcondition	The new study program is visible to students through the appropriate smart contract method.

A. Appendix

Table A.14.: Course registration of student by SPM

ID	14
Name	Register student for course
Level	Secondary
Brief description	The study program manager selects a student and attempts to register him/her for a course.
Prerequisites	<ul style="list-style-type: none"> • Student is not registered for the course yet
Actors	<ul style="list-style-type: none"> • Student • Study program manager • Smart contract
Triggering event	Study program manager selects a student for course registration.
Description	<ol style="list-style-type: none"> 1. Study program manager initiates a student's registration for a course. 2. A smart contract checks if requirements are met and allows registration or provides an error message. 3. If no error message is provided, the new block is validated by one or more signing nodes and added to the blockchain.
Postcondition	The transaction was successfully added to the blockchain.

A.2. Diagrams

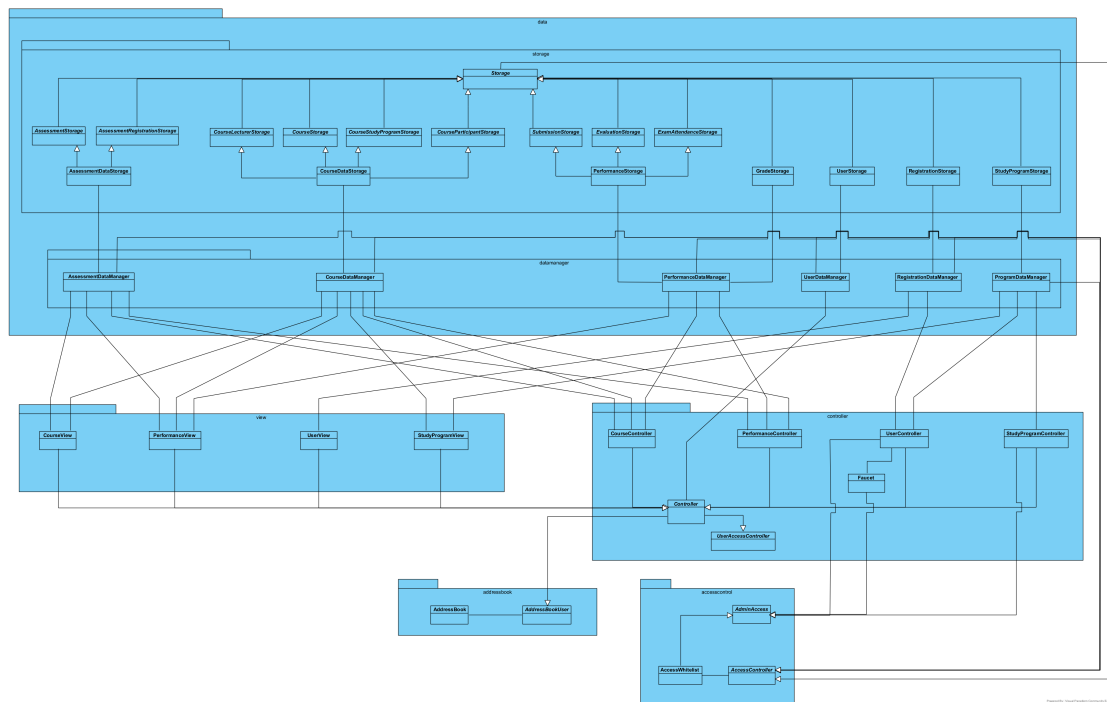


Figure A.1.: UML class diagram of the smart contracts

A. Appendix

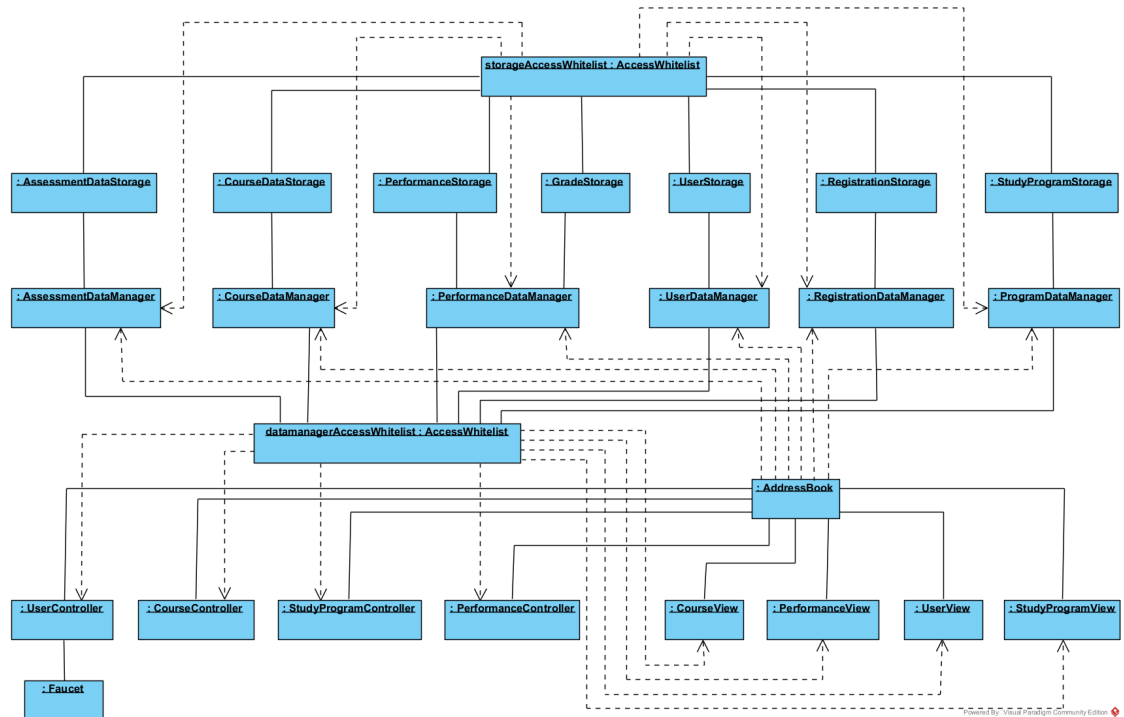


Figure A.2.: UML object diagram of the deployed smart contracts

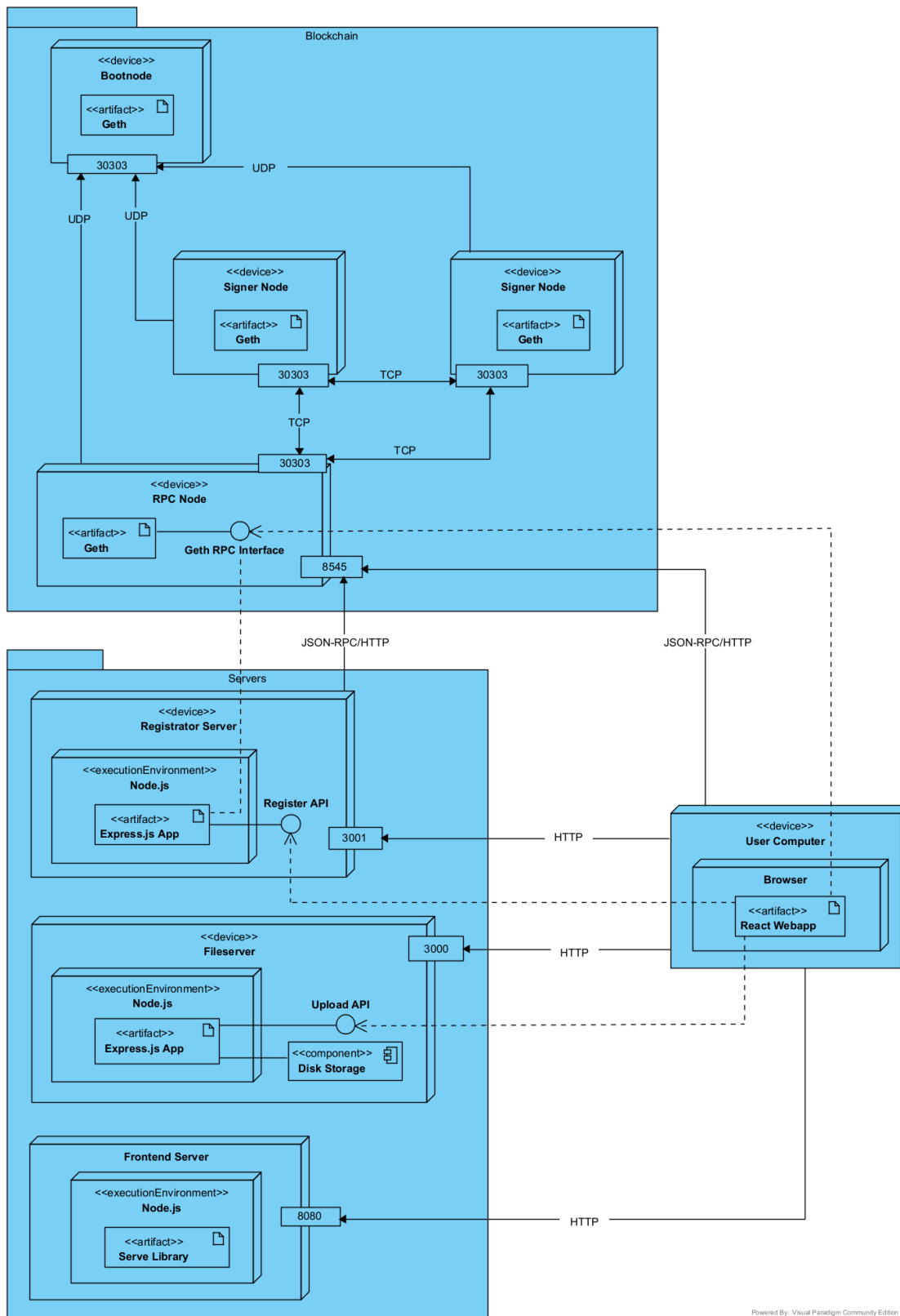


Figure A.3.: UML deployment diagram of system components

A. Appendix

A.3. Test output

```
1 Registration
2   Dependency checks
3     ✓ Should set automatic acceptance (3457ms)
4   Use case: Register user
5     ✓ Should registration result in 'under review' pending registration
      (140ms)
6     ✓ Should registration acceptance result in 'accepted' pending
      registration (201ms)
7     ✓ Should registration rejection result in 'rejected' pending
      registration (215ms)
8     ✓ Should acknowledgement of accepted registration result in
      registered user (286ms)
9   Edge cases
10    ✓ Should rejected user be able to register again (280ms)
11    ✓ Should accepted but not acknowledged registration result in not
      registered user (168ms)
12    ✓ Should registered user not be able to register again (280ms)
13    ✓ Should user with pending registration request not be able to
      register again (170ms)
14    ✓ Should user with 'under review' registration not be able to
      acknowledge result (110ms)
15    ✓ Should registration with not existing study program revert (57ms)
16    ✓ Should registration with date of birth in the future revert (58ms
      )
17
18 Study program creation
19   Use case: Create study program
20     ✓ Should add study program (80ms)
21
22 Course creation
23   Dependency checks
24     ✓ Should lecturer register (262ms)
25   Use case: Create course
26     ✓ Should add course (477ms)
27     ✓ Should add course with assessment (492ms)
28   Edge cases
29     ✓ Should not add course if sender is not lecturer (324ms)
30     ✓ Should not add course with assessment with higher min points than
      max points (406ms)
31
32 Course registration
33   Dependency checks
34     ✓ Should user register (249ms)
35     ✓ Should add study program (66ms)
36     ✓ Should add course (389ms)
37   Use case: Register for course
38     ✓ Should student register for course (937ms)
39     ✓ Should course registration revert if course is full (866ms)
40     ✓ Should course registration revert if registration deadline has
      passed (825ms)
41     ✓ Should course registration revert if not enrolled in required
```



```

study program (732ms)
42 Use case: Register student for course
43   ✓ Should SPM register student for course (978ms)
44 Use case: Deregister from course
45   ✓ Should student deregister from course (1076ms)
46 Edge cases
47   ✓ Should course registration revert if student is already
registered (1077ms)
48   ✓ Should course deregistration revert if deregistration deadline
has passed (1194ms)
49   ✓ Should course deregistration revert if student is not registered
for course (753ms)
50
51 Assessment registration
52   Dependency checks
53     ✓ Should student register for course (940ms)
54   Use case: Register for assessment
55     ✓ Should student register for optional assessment (1069ms)
56     ✓ Should student be automatically registered for not optional
assessments of course (952ms)
57     ✓ Should assessment registration revert if registration deadline
has passed (999ms)
58   Use case: Deregister from assessment
59     ✓ Should student deregister from assessment (1181ms)
60     ✓ Should course deregistration deregister from its assessments as
well (1222ms)
61   Edge cases
62     ✓ Should assessment registration revert if student is already
registered (983ms)
63     ✓ Should assessment registration revert if student is not
registered for the course (707ms)
64     ✓ Should assessment deregistration revert if student is not
registered for the assessment (681ms)
65
66 Deployment
67   ✓ Should deploy Storages 1 (339ms)
68   ✓ Should deploy Storages 2 (137ms)
69   ✓ Should deploy Datamanagers (265ms)
70   ✓ Should deploy controllers (181ms)
71   ✓ Should deploy views (144ms)
72   ✓ Should deploy system and configure deployments (843ms)
73
74 Assessment evaluation
75   Dependency checks
76     ✓ Should student be registered for assessment (1012ms)
77   Use case: Evaluate student's assessment
78     ✓ Should lecturer evaluate student's assessment (1104ms)
79   Use case: Modification of evaluation
80     ✓ Should lecturer modify the evaluation of a student's assessment
(1198ms)
81   Edge cases
82     ✓ Should evaluation revert if lecturer attempting to evaluate does
not lecture at the course (1164ms)

```

A. Appendix

```
83
84 Grading
85   Dependency checks
86     ✓ Should lecturer evaluate student's assessment (1089ms)
87     ✓ Should student submit a document for assessment (1134ms)
88     ✓ Should lecturer confirm student's test attendance (1138ms)
89   Use case: Generate grade
90     ✓ Should auto-evaluate missing submission correctly (1346ms)
91     ✓ Should auto-evaluate not attended exam correctly (1440ms)
92     ✓ Should summarize student's evaluation points correctly - VO (1958
ms)
93     ✓ Should summarize student's evaluation points correctly - not VO
(1827ms)
94     ✓ Should grade course performance correctly - grade 4 (1735ms)
95     ✓ Should grade course performance correctly - not achieved min
points (1590ms)
96   Use case: Manual evaluation
97     ✓ Should lecturer manually grade a student's course performance
(1105ms)
98   Use case: Modification of evaluation
99     ✓ Should lecturer modify the grade of a student's course
performance (1185ms)
100
101 Submission
102   Dependency checks
103     ✓ Should student be registered for assessment (959ms)
104   Use case: Submit a document
105     ✓ Should student submit a document for assessment (1133ms)
106   Edge cases
107     ✓ Should submission revert if deadline has passed (1136ms)
108     ✓ Should submission revert if assessment is not of submission type
(1001ms)
109
110 Test attendance
111   Dependency checks
112     ✓ Should student be registered for assessment (950ms)
113   Use case: Confirm test attendance
114     ✓ Should lecturer confirm student's test attendance (1098ms)
115   Edge cases
116     ✓ Should confirmation of exam attendance revert if student is not
registered for the assessment (503ms)
117     ✓ Should confirmation of exam attendance revert if assessment does
not exist (492ms)
118
119
120 68 passing (55s)
```

A.4. Puppeth config

```

1 puppeth --network=uchain
2
3 What would you like to do? (default = stats)
4   1. Show network stats
5   2. Configure new genesis
6   3. Track new remote server
7   4. Deploy network components
8 > 2
9
10 What would you like to do? (default = create)
11   1. Create new genesis from scratch
12   2. Import already existing genesis
13 > 1
14
15 Which consensus engine to use? (default = clique)
16   1. Ethash - proof-of-work
17   2. Clique - proof-of-authority
18 > 2
19
20 \begin{lstlisting}[breaklines]
21 How many seconds should blocks take? (default = 15)
22 > 4
23
24 Which accounts are allowed to seal? (mandatory at least one)
25 > 0x9c3ca6a0a2a5ea1d25c896da9679e97629804284
26 > 0x4b6497cf99294547dd458d939b27c471f53dba29
27 > 0x
28
29 Which accounts should be pre-funded? (advisable at least one)
30 > 0x65e1fc4b8fbc4e90aa7feb60b9b6e5b224841aae
31 > 0xd7485a0358d585c7611321b8267172f69091dbff
32 > 0x
33
34 Should the precompile-addresses (0x1 .. 0xff) be pre-funded with 1 wei? (
    advisable yes)
35 > yes
36
37 Specify your chain/network ID if you want an explicit one (default =
    random)
38 > 8105

```

A.5. MetaMask explanation protocol

"The MetaMask wallet is a digital wallet that allows you interact with the platform. There will be cases during our tests when you are instructed to click on a button that changes some data on the website, which causes a small window to pop up in the top right corner requesting confirmation from you. This small window is your MetaMask wallet, and you just need to click on "Confirm" to resume the action. After this, you need to wait for another small notification to pop up in the bottom right corner informing you about the successful transaction.

A MetaMask wallet is able to contain multiple accounts, just like a real wallet can hold your debit cards. If you are instructed to switch between the wallet accounts, you have to click on the MetaMask icon in the browser's toolbar, then by clicking on the account avatar you open the list of the accounts stored in the wallet. Select the one specified by the instructions and you successfully switched to the new account."

A.6. User test instruction protocol

1. Make sure that "Account 1" is selected in your wallet.
2. Fill out the registration form on the starting page. Register as a student who wishes to enroll in computer science. Submit your registration request. Now you can see that your registration request is "under review".
3. Switch to account "spm1" in your wallet. Now you operate the account of a study program manager.
4. Navigate to page "Registrations". Find your own registration in the list of pending registrations that you submitted earlier and review it. Accept the registration and confirm your first transaction in the MetaMask popup.
5. Switch back to "Account 1".
6. Navigate to page "Registration". You can see that your registration's status changed to "accepted". Now finalize your registration.
7. Navigate to page "All courses" from the dropdown menu "Courses". Register for the course "Advanced Mathematics" of computer science in winter semester 2022. Register for "Exam option 2" of the course as well.
8. Switch to account "student1". Now you operate the account of a student called "Valerie Cooper".
9. Navigate to page "My courses" from the dropdown "Courses". Deregister from the course "Statistics" of winter semester 2022 at computer science.
10. Switch to account "student3". Now you operate the account of another student called Joey Davis.

A.6. User test instruction protocol

11. Navigate to page "Study performances". Open the course "Programming" of summer semester 2023 at computer science. Select the assessment "Milestone 1" and upload the two prepared textfiles you find on the computer.
12. Switch to account "lecturer2". Now you operate the account of a lecturer called "Joseph Chambers".
13. Navigate to page "Assessment performances" from the dropdown menu "Study performances". Open course "Programming" of summer semester 2023 at computer science. Select the assessment "Milestone 1" and inspect the participant "Joey Davis" in the "Registered participants" list. Open the student's submitted files. Now evaluate the student's work with arbitrary amount of points.
14. Switch to account "lecturer1". Now you operate the account of another lecturer called "Dora Mann".
15. Stay on the same page "Assessment performances". Open the course "Statistics" of winter semester 2022 at computer science. Select assessment "Test 2" and find the student "Joey Davis" in the "Registered participants" list. Now confirm that Joey Davis has attended the test.
16. In the same course, find "Lee Singleton" in the participant list and edit his evaluation to another arbitrary amount of points.
17. Switch to account "lecturer2".
18. Navigate to page "Grading" from the dropdown menu "Study performances". Open course "Programming" of summer semester 2023 at computer science. Now update the performances. Inspect "Lee Singleton" and review his generated grade. Edit his grade to another arbitrary grade.
19. Navigate to page "Create new" from the dropdown "Courses". Fill out the form and create a course of your liking with type VU, 2 classes and 2 assessments (1 exam and 1 submission).
20. Switch to account "spm1".
21. Navigate to page "Create new" from the dropdown "Studies". Give an arbitrary name to the study program and create it.
22. Navigate to page "Add student to course" from the dropdown "Courses". Select the course that you've just created from the list, find the user in the student's list that you registered in the beginning.

A. Appendix

A.7. User interface

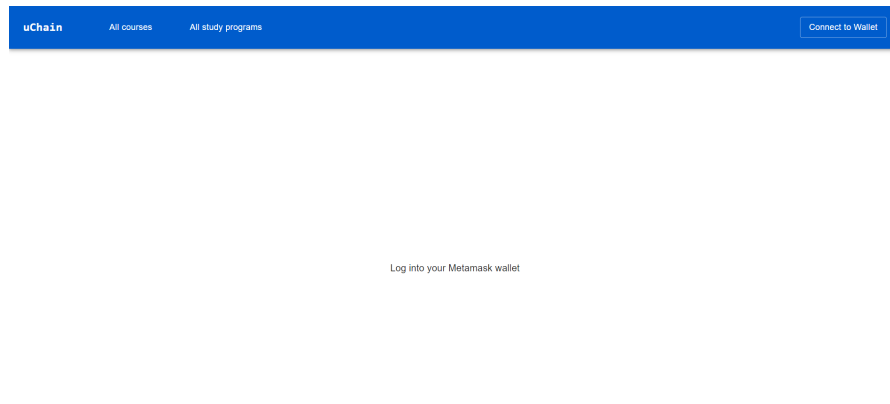


Figure A.4.: Login page

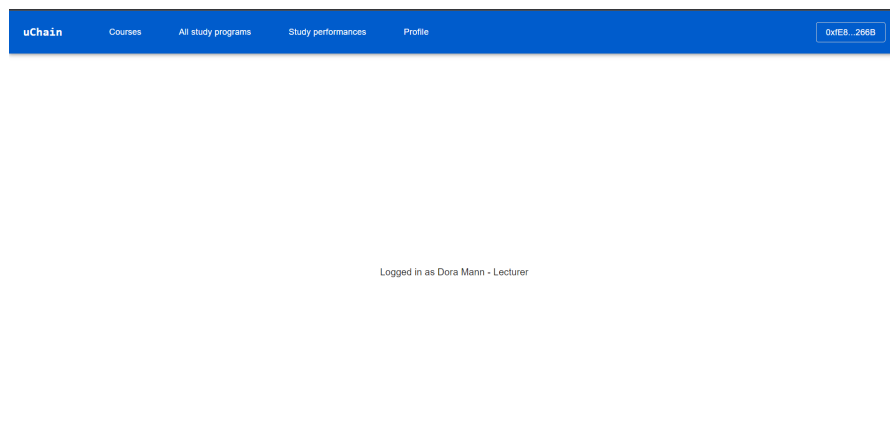
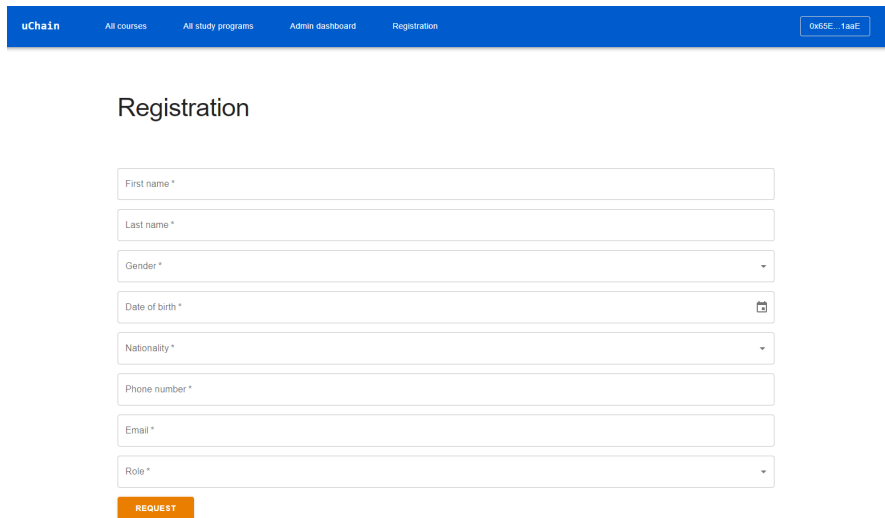


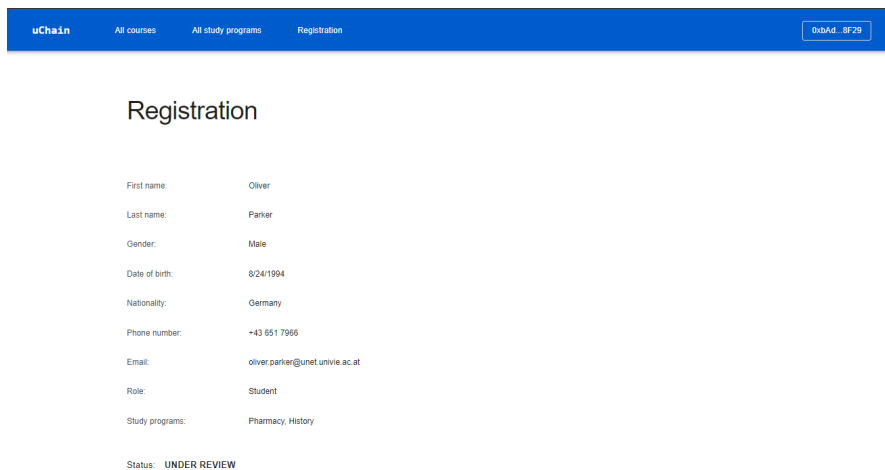
Figure A.5.: Home page

A.7. User interface



The screenshot shows the 'Registration' page of the 'uChain' application. The top navigation bar is blue with the 'uChain' logo and links to 'All courses', 'All study programs', 'Admin dashboard', and 'Registration'. A 'Logout' button is in the top right. The main heading is 'Registration'. Below it is a form with the following fields: 'First name *', 'Last name *', 'Gender *' (a dropdown menu), 'Date of birth *' (with a calendar icon), 'Nationality *' (a dropdown menu), 'Phone number *', 'Email *', and 'Role *' (a dropdown menu). At the bottom of the form is an orange 'REQUEST' button.

Figure A.6.: Registration page



The screenshot shows the 'Registration' page after a request has been sent. The top navigation bar is the same as in Figure A.6. The main heading is 'Registration'. Below it, the form fields are now populated with the following data: First name: Oliver, Last name: Parkner, Gender: Male, Date of birth: 9/24/1994, Nationality: Germany, Phone number: +43 651 7966, Email: oliver.parkner@unet.univie.ac.at, Role: Student, Study programs: Pharmacy, History. At the bottom, the status is displayed as 'Status: UNDER REVIEW'.

Figure A.7.: Registration page (after request is sent)

A. Appendix

uChain

CoursesStudiesProfileRegistrations

0x3d1...a478

Pending registrations

▼ Oliver Parker

ACCEPT

REJECT

First name:

Oliver

Last name:

Parker

Gender:

Male

Date of birth:

8/24/1994

Nationality:

Germany

Phone number:

+43 651 7966

Email:

oliver.parker@unet.univie.ac.at

Role:

Student

Study programs:

Pharmacy, History

Figure A.8.: Pending registrations page

uChain

CoursesStudiesProfileRegistrations

0x3d1...a478

Profile

First name:	Heather
Last name:	Harper
Gender:	Female
Date of birth:	1/18/1997
Nationality:	Austria
Phone number:	+43 699 423209
Email:	heather.harper@unet.univie.ac.at
Role:	Study program manager
Study programs:	-
Wallet address:	0x3d1d42dCA0853a81E5373E6794F3AA6162fa478
Balance:	0.9999999999999999121379 UToken

REQUEST TOKENS

Figure A.9.: Profile page

A.7. User interface

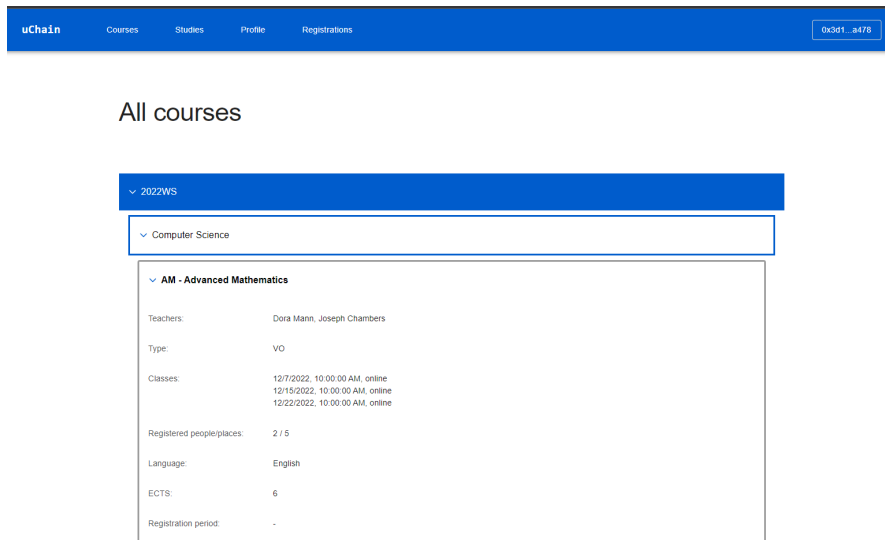


Figure A.10.: All courses page

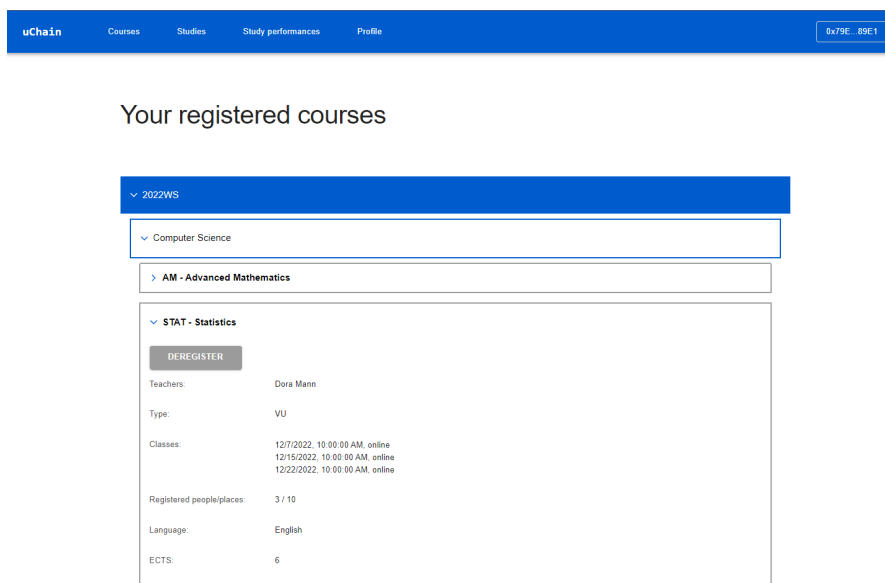


Figure A.11.: My courses page - student

A. Appendix

▼ 2022WS

► Computer Science

▼ Physics

► AM - Advanced Mathematics

▼ STAT - Statistics

Teachers: Dora Mann

Type: VU

Classes: 12/7/2022, 10:00:00 AM, online
12/15/2022, 10:00:00 AM, online
12/22/2022, 10:00:00 AM, online

Registered people/places: 3 / 10

Participants:

- Valerie Cooper - 2
- Lee Singleton - 3
- Joey Davis - 4

Language: English

ECTS: 6

Registration period: 1/20/2023, 11:58:00 AM - 2/4/2023, 11:58:00 AM

Deregistration period: 1/20/2023, 11:58:00 AM - 2/9/2023, 11:58:00 AM

Description: Lorem ipsum dolor sit amet

Examination online: Exam incum

Figure A.12.: My courses page - lecturer

uChain Courses Studies Profile Registrations

0x3d1 - 3476

▼ Computer Science

Program identifier: 1

► Physics

► Pharmacy

► History

Figure A.13.: All study programs page

A.7. User interface

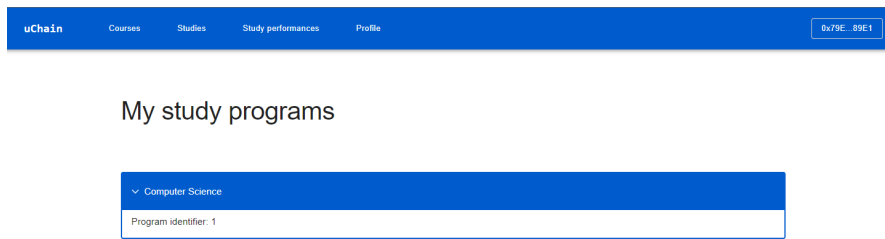


Figure A.14.: My study programs

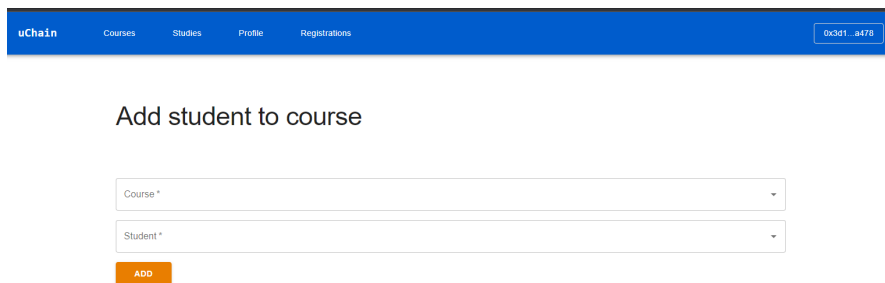


Figure A.15.: Add student to course page

A. Appendix

The screenshot shows the 'Create new study program' page. At the top is a blue navigation bar with the 'uChain' logo and links for 'Courses', 'Studies', 'Profile', and 'Registrations'. A user ID '0x3d1...a478' is displayed on the right. The main heading is 'Create new study program'. Below it is a text input field labeled 'Enter study program name *' and an orange 'ADD' button.

Figure A.16.: Create new study program page

The screenshot shows the 'Your study performances' page. The top navigation bar is blue with 'uChain' and links for 'Courses', 'Studies', 'Study performances', and 'Profile'. A user ID '0x4e0...4331' is on the right. The main heading is 'Your study performances'. Below it is a blue bar with a dropdown arrow and '2022WS'. A white box contains a dropdown for 'Computer Science'. Inside this box, another dropdown shows 'STAT - Statistics'. Below this, details for the course are listed: Teachers: Dora Mann, Type: VU, Registered people/places: 3 / 10, and Grading criteria: 4: >= 50%, 3: >= 65%, 2: >= 75%, 1: >= 85%. A table shows 'Total evaluated points' as 30 / 30 (100 %) and 'Grading' as '-'. At the bottom of the box are expandable sections for 'Test 1', 'Milestone 1', and 'Test 2'. Below the box is a link for 'Physics'. At the very bottom is a blue bar with a dropdown arrow and '2023SS'.

Figure A.17.: Your study performances page

Total evaluated points	Grading
30 / 30	-
100 %	

Test 1

Attendance

Time, place: 1/15/2023, 11:58:37 AM, Audimax

Type: Exam

Max. points: 30

Min. required points: 10

Requires separate registration: No

Confirmation: attended

Last modified: 1/15/2023, 12:01:01 PM

Evaluation

Points: 30

Last modified: 1/20/2023, 12:01:43 PM

Feedback: -

Evaluated by: Dora Mann

Milestone 1

Uploads

Deadline: 1/29/2023, 11:58:41 AM

Type: Submission

Max. points: 50

Min. required points: 10

Requires separate registration: No

Documents: [non_existent_example.txt](#)

Last modified: 1/28/2023, 12:01:22 PM

SELECT

SAVE

Evaluation

-

Test 2

Figure A.18.: Your study performances page - cont.

A. Appendix

uChain

CoursesAll study programsStudy performancesProfile

0x0e5...6890

Assessment performances

> 2022WS

< 2023SS

Computer Science

PROG - Programming

UPDATE PERFORMANCES

Teachers: Joseph Chambers

Type: VU

Registered people/places: 2 / 2

Grading criteria:

4:	>= 50%
3:	>= 65%
2:	>= 75%
1:	>= 85%

> Test 1

> Milestone 1

Figure A.19.: Assessment performances page

< Milestone 1

Deadline: 2/4/2023, 11:58:54 AM

Type: Submission

Max. points: 30

Min. required points: 10

Requires separate registration: No

Registered participants

Lee Singleton - 3

Uploads		Evaluation	
Documents:	non_existent_example.txt	Points:	20
Last modified:	1/30/2023, 12:01:26 PM	Last modified:	1/30/2023, 12:01:51 PM
		Feedback:	-
		Evaluated by:	Joseph Chambers

EDIT

> Joey Davis - 4

Figure A.20.: Assessment performances page - submission

A.7. User interface

3: >= 65%

2: >= 75%

1: >= 85%

Test 1

Time, place:

1/10/2023, 11:58:50 AM, Audimax

Type:

Exam

Max. points:

10

Min. required points:

5

Requires separate registration

No

Registered participants

> Lee Singleton - 3

Joey Davis - 4

Attendance

Confirmation: not attended

Last modified: 1/11/2023, 12:01:14 PM

Evaluation

Points: 0

Last modified: 1/31/2023, 1:24:10 PM

Feedback: Automatic: Exam was not attended

Evaluated by: System

EDIT

> Milestone 1

Figure A.21.: Assessment performances page - exam

uChain

Courses

All study programs

Study performances

Profile

index_6880

Grading

> 2022WS

> 2023SS

Computer Science

PROG - Programming

UPDATE PERFORMANCES

Teachers:

Joseph Chambers

Type:

VU

Registered people/places:

2 / 2

Grading criteria:

4: >= 50%

3: >= 65%

2: >= 75%

1: >= 85%

Registered participants

> Lee Singleton - 3

Total evaluated points

Grading

Figure A.22.: Grading page

A. Appendix

Registered participants

Lee Singleton - 3

Total evaluated points
29 / 40
72.5 %

Grading

Grade: 3 (automatic)
Last modified: 1/31/2023, 1:24:10 PM
Feedback: -
Graded by: System

EDIT

> Test 1

> Milestone 1

> Joey Davis - 4

Figure A.23.: Grading page - cont.

uChain Courses All study programs Study performances Profile 0vIEA...266B

Create new course

Course title *

Course code *

Course type *

Year * Season *

Description

Examination topics

Language *

ECTS *

Max. number of participants *

Requirement course codes

Lecturers *

Study programs *

Figure A.24.: Create new course page

☐ Registration period required

Grading criteria

Grade 1 *	min. % (>=)
Grade 2 *	min. % (>=)
Grade 3 *	min. % (>=)
Grade 4 *	min. % (>=)

Classes

Time *	Place *	REMOVE
01/31/2023 09:23 pm		

ADD

Figure A.25.: Create new course page - cont. 1

Assessments

Type *

Exam

Title *

Time *

01/31/2023 09:22 pm

Place *

Max. points *

0

Min. points (if not achieved, the course is failed automatically.) *

0

REMOVE

ADD

CREATE

Figure A.26.: Create new course page - cont. 2