



BACHELOR'S THESIS

WK2: BLOCKCHAIN: EXAMPASS
(MULTICHAIN-BASED)

Author

Roman Karaba

Pursued academic degree

Bachelor of Science (BSc)

Vienna, 2018

Degree Programme Code: A 033 521

Field of study: Computer Science - Medieninformatik

Supervisor: Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas

Contents

1	Motivation	3
1.1	The Blockchain Technology	3
1.2	Permissionless and Permissioned Chains	4
1.2.1	Permissionless (Public) Chains	4
1.2.2	Permissioned (Private) Chains	4
1.3	Goal of the Thesis	5
2	Related Work	5
2.1	Similar projects	5
2.2	Why not to use blockchain as database	6
2.3	How it relates to this thesis	6
3	Design	7
3.1	Technologies	7
3.2	Architecture overview	8
3.3	Functionality	9
3.3.1	User: Student	9
3.3.2	User: Teacher	11
4	Implementation	13
4.1	Front-end	13
4.2	Back-end	15
4.2.1	Express/Node server	15
4.2.2	MultiChain	16
4.2.3	Utilities	17
4.3	Deployment	17
4.3.1	Installation (local)	17
4.3.2	Installation (global)	18
5	Evaluation and Discussion	18
6	Conclusions and Future Work	19
7	Acknowledgements	20

Abstract

In this thesis I discuss the advantages and disadvantages of using a blockchain as a database solution. The core consists of an implementation of a course management system with the utilization of the MultiChain blockchain framework for data storage of the application as a permissioned (private) chain using the Proof-of-Authority consensus mechanism. The system is implemented as a single page application (SPA) using a combination of Facebooks React for the front-end of the application and a Node.js/Express server as the back-end handler. I draw conclusions on the viability and use case scenarios about the implementation of this system.

1 Motivation

1.1 The Blockchain Technology

The blockchain technology has come a long way since its invention by Satoshi Nakamoto in 2008[1], with the intention to serve as a public distributed ledger for trading assets in the form of digital tokens, Bitcoin being the first such a token. It started a technological revolution[2] with many industries looking into the technology and its uses. The offer of transaction data and meta-data immutability with full transparency was attractive to the public, as it promised almost full control of assets to the individual.

A blockchain in its essence is a decentralized peer-to-peer insert only database with a consensus mechanism that eliminates the need of trusting a centralized entity for the management of the ledger. It was a natural next step to investigate the possible usage of this technology as a distributed database solution, since it shows potential in solving problems at a inter-organizational data communication level.

The standard approach in the past to this problem having organizations have a internal centralized system managed by each organization separately. Each organization on its own was responsible for validation, integrity and security of data. They had to deal with disagreements between the final version of the data between the multiple systems.

The current generation approach was to hire an external intermediary company as an entity that would manage a central database to represent a final truth of the data. This meant that the organizations no longer communicated between each other directly but with this central entity e.g. SWIFT in the financial industry or Amadeus in the travel industry.

Blockchain proposes a new approach to this problem, similar to the first mentioned approach, by connecting the organizations again, where each organization has its own blockchain node deployed. Because of the Peer-to-Peer nature of blockchain and its consensus mechanism, it presents a certainty in the validity and truth of data when querying the chain as well as being up to date.

A problem arose because the original consensus mechanism relies on the proof of work consensus[1] which heavily impacts performance of the system[3].

New consensus mechanisms such as Proof-of-Stake (PoS), Proof-of-Activity (PoW / PoS-hybrid), Proof-of-Capacity (PoC or Proof-of-Storage), and many more have been proposed as a more efficient alternative to PoW with some success.

1.2 Permissionless and Permissioned Chains

1.2.1 Permissionless (Public) Chains

Blockchain systems like Bitcoin or Ethereum are so called *permissionless* (public) systems where any node on the Internet can join and compute or *mine* blocks typically in accordance with the Proof-of-Work consensus mechanism[4]. In brief, the Proof-of-Work consensus mechanism dictates that every block in the blockchain contains a hash of the previous block alongside the data that's to be stored in the chain. The hash is generated by inputting the hash of the previous block, the data you want to store and a proof of work value X as input to a hash function predetermined by the specific blockchain variant. The resulting hash of a predetermined length, has to fulfill a condition, typically it has to start with N (mining difficulty) zeroes or ones, if it does, the miner doing the hashing has successfully found a block and it can be appended to the chain storing the data in it with X as proof. If not, the miner has to increment the X value and repeat the hashing process until he or some other miner participating in the blockchain network succeeds.

To put this into perspective, the probability to succeed when using this technique to generate a SHA256 hash with a difficulty set to 30, in other words where the first 30 characters are successive zeros, for a random message is one in 2^n with $n = 30$ or one in over 1 billion. Because this function is cryptographic, there is no other way of finding such hash apart from repetitive trial and failure, but verifying the validity is very simple as it takes only to executing the hash function with the provided proof of work value. As a blockchain matures and the difficulty increases, this process becomes very demanding on computing performance.

1.2.2 Permissioned (Private) Chains

The idea behind *permissioned* (private) blockchains is that, as its name suggests, they are private and only a selected group of individuals are granted a permission to access the private chain, from which only a subset is allowed to mine blocks and add them to the chain. One of such systems is called Multichain[5]. It utilizes a Proof-of-Authority consensus mechanism where the selected few who can mine blocks become *Authorities* who sign blocks and add them to the chain without having to perform heavy computation like with the Proof-of-Work consensus. Becoming an authority is dependant on the criteria a system establishes, but generally speaking the Authorities guarantee with their reputation and identity as leverage for the privilege. If one of the authorities acts in a malicious manner, its privilege is taken and a new authority is chosen in to replace them.

This approach offers substantial improvements in the performance of the chain, by up to over 1000 transactions per second at 1 000 000 total transactions[6], in comparison to Bitcoins Proof-of-Work approach that pales in comparison with a theoretical max of 7 transactions per second[7]. Also the privacy aspect can be appealing for private organizations as data confidentiality is typically a required functionality.

1.3 Goal of the Thesis

The goal is to implement a course management system utilizing MultiChain as a blockchain database for storing user and course data. Immutability being one of the core strengths of a blockchain offers data integrity for the students and teachers as users, which ensures re-callable assignment and test evaluations with their respective feedbacks.

Students are supposed to be able to find courses, enroll into courses, withdraw from courses as well as view course details and their own performance evaluation at that course. It is also expected that they have the option to view a general summary of their overall grades and achievements.

Teachers will utilize the application to create courses, write feedbacks, evaluate assignments and tests. After a student has been fully evaluated according to the set criteria of the course, the teacher may grade the student which completes the students course participation.

The development process of this app comprised of tools research, simple iterative programming and consulting with experts professionals in the field and I was able to use the MultiChain blockchain as the core database for this web app to utilize its benefits.

My development strategy initially was utilizing a SCRUM[8] inspired approach, with 4 planned sprints with issues assigned to each sprint. Because of unexpected events this SCRUM inspired approach started to resemble more to the Kanban[9] approach with having a stack of issues on a Kanban board and solving them one by one, independent from the initial time plan.

2 Related Work

2.1 Similar projects

Research in utilizing the blockchain technology as a distributed database has been done with varying degrees of success. ModelChain[10] is a framework with blockchain and machine learning as the core technologies. The distributed nature of a blockchain database enabled a rich data set for a model used to train and implement any online learning algorithm like EXPLORER or Distributed Autonomous Online Learning in the proposed framework. The Proof-of-Work consensus is utilized at the core of this framework with a newly developed Proof-of-Information consensus system. The report states that using the Proof-of-Work consensus at its core did limit the performance of the framework. One of

the given examples for a use case of this framework was to create a model that would predict the risk of re-admission for a particular set of patients.

A more in depth analysis of a blockchain database with the focus on the Bitcoin infrastructure showed the problem of transaction uncertainty and transaction conflicts on a proof-of-work based chain, as well as the problem of issuing transactions that are or are not mutually consistent with a given subset of pending transactions[11].

The satellite chain infrastructure[12] promises a more industry compliant approach to privacy, scalability and to the lack of governance by presenting the possibility of designing a network with separate regions. Introducing also a regulator entity chosen by the participants of the network to for governance and enforcement of mutually agreed policies to increase mutual trust of the network participants. The network regions would also have the possibility of using diverse consensus systems which offers flexibility to the requirements of potential participants of the system.

2.2 Why not to use blockchain as database

While it is exciting to use the cutting edge to develop a new system, it certainly can bring more negatives into the system than the positives. Using a blockchain as database makes sense in quite specific scenarios. If the system consists of entities, that would use the data storage solution, are all trusted parties, with known writers the utilization of a blockchain is not advised. Unless mistrust exists between the parties and an always online third trusted party is not viable for usage, it is recommended to not use a blockchain[13].

There is an argument to be made here from the perspective of a user. This functionality might be desired and increase the attractiveness of such a system which could increase its popularity.

2.3 How it relates to this thesis

Both of these research papers[10][11] are working with the underlying concept of using a system based on Proof-of-Work which coincidentally causes the mentioned performance issues and conflict problems. Utilizing a Proof-of-Authority consensus system should inherently eliminate these issues. The consensus system used by MultiChain works on the basis that no node can issue transaction onto the chain more than once in a row, if more than one node is present in the network. Combining this with the performance of MultiChain and a accordingly designed data storage strategy could avoid such problems. Scalability of the MultiChain system is also according to its developers unlimited[14] which resolves one of the issues tackled by the satellite chain infrastructure. Also the governance is not an issue in the proposed system of this thesis since it is designed to be used by a single entity.

3 Design

Two main approaches were considered for development.

1. Desktop application that would communicate with a remote endpoint to access data from the MultiChain server.
2. Web application implemented as a distributed system.

To develop the desktop application, the Electron.js[15] framework was considered. Electron combines the Chromium web browser and Node.js[16] into a single run-time that can be packaged for the MacOS, Windows and Linux platforms. This offers the possibility to develop a cross platform desktop application using a web stack architecture, HTML and CSS for the user interface and JavaScript for the logic of the application.

The second approach, building a web application in a distributed system, presented more options in the choice of technologies. For example, Back-end programming done in JavaEE on top of a Apache Tomcat web server and a traditional SQL database, or a purely JavaScript oriented stack, utilizing Node.js as the back-end, and any one of the modern front-end JavaScript frameworks like Vue, Angular or React.

JavaScript, also called, ECMAScript defined by the ECMA-262 8th edition standard[17] introduced the `async/await` keywords which present a significant improvement in handling asynchronous function calls and code readability. Using ES would help implement the majority of the project using almost only a single programming language that could help a someone inexperienced to understand the code base faster, which is also why I decided to implement this system on a JavaScript focused web stack.

3.1 Technologies

- Express.js - A lightweight back-end framework build on top of Node.js
- React - Front-end component driven framework developed by Facebook
- MongoDB - Simple and powerful document database
- MultiChain - Blockchain solution
- Webpack - Modules bundler, build preloader

Other notable libraries that we used to implement this application:

- bcrypt - Cryptographic library used for password encryption using a hashing function proposed by Neils Provos and David Mazires[18]
- Material UI - React component library that implements Google's Material Design[19]
- Material Kit React - React component library based on Material UI[20]

- jsonwebtoken - library implemented according to the RFC-7519[21] Standard

3.2 Architecture overview

The front-end of the application is served by a static server that upon accessing its URL delivers the user the React single page application (SPA). Unless the user requests the home URL intentionally, the SPA doesn't communicate with the static server. Any further requests from the client are directed to the Express.js/Node.js back-end server. The back-end server handles all of the remaining functionality such as login, signup, creating courses, enrolling into courses etc. per request from the client interface.

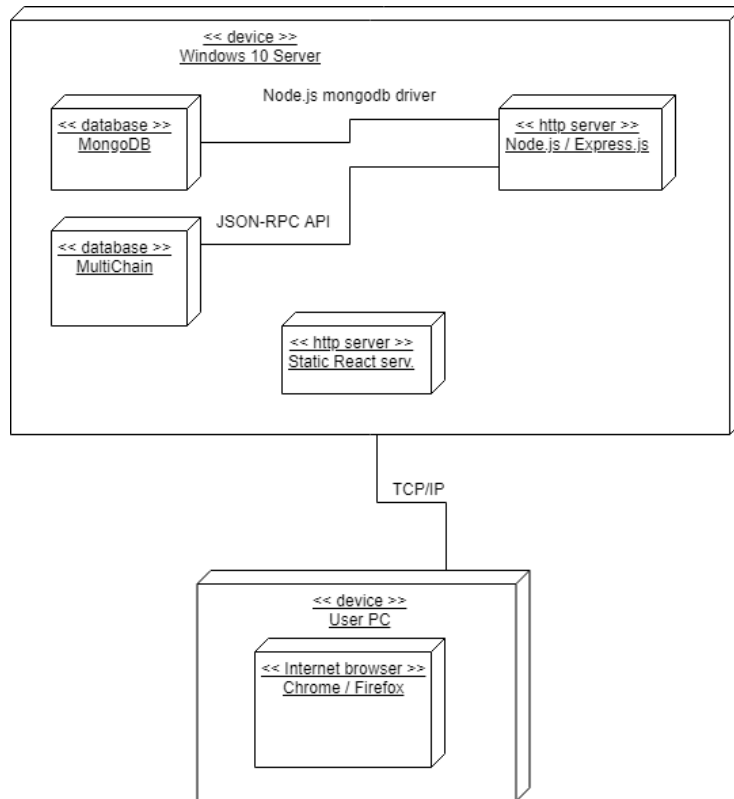


Figure 1: System Overview

Figure 1. shows the physical representation of how the system is deployed. The nature of splitting the static front-end server and the back-end server enables us to deploy the front-end on different hardware entirely, which would reduce load on the machine. The SPA does not utilize Reacts ability to render the page server-side. This is because the front-end is quite simple and is not

expected to under-perform, even on lower-end hardware.

3.3 Functionality

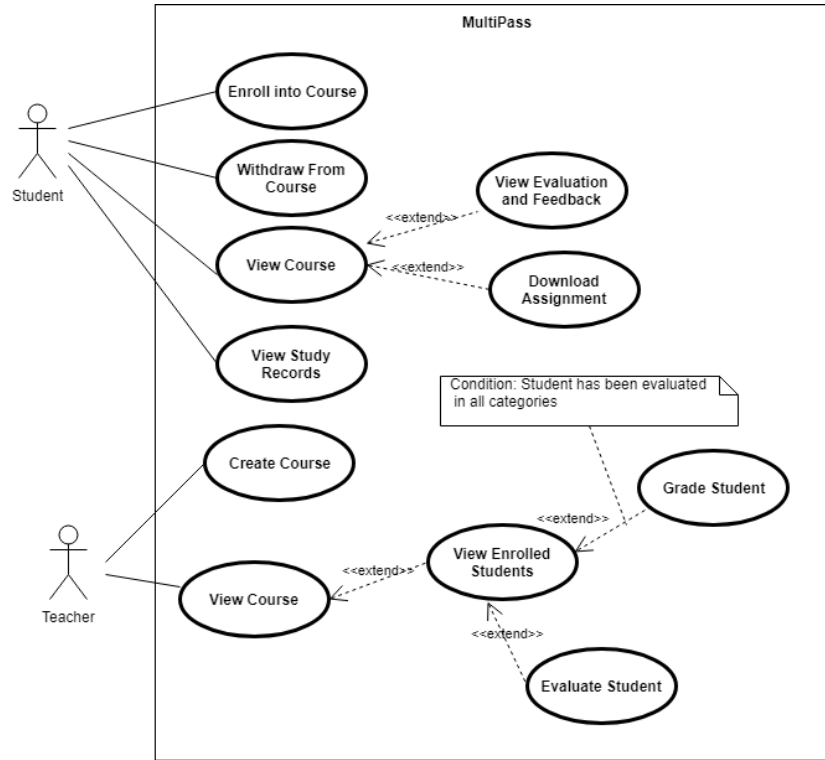


Figure 2: Use Cases

Users considered in this application are students and teachers. Figure 2. describes how the users may interact with the system.

Login and signup are not part of the use case diagram, but the system does support it, and we will suppose in the following sections that the user is signed up and logged in into the system.

While on the signup page, the user can signup as a teacher or student by clicking either the *Signup (Student)* or *Signup (Teacher)* button.

3.3.1 User: Student

Students are offered the functionality to manage and reflect on their performances. They can enroll into a courses, withdraw from courses, download assignments and view their achieved grades and evaluation.

To enroll into a course, a student must click on the *Find Course* button on top of the navigation bar, this redirects him inside the SPA to the find-course

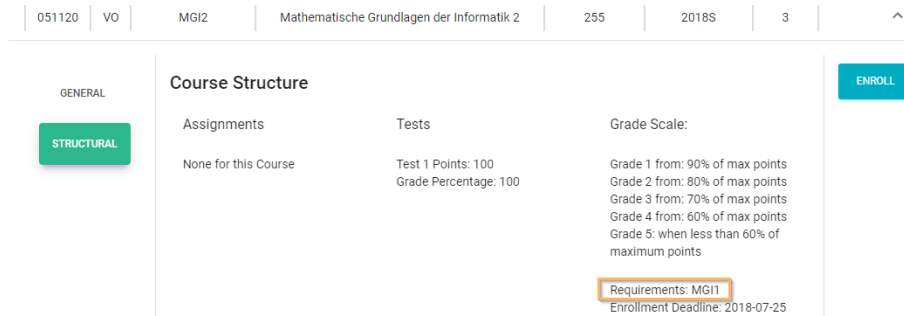


Figure 3: Structural course information

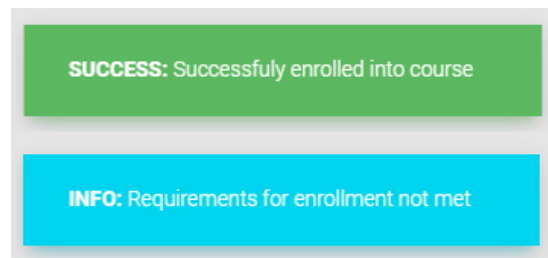


Figure 4: Information Snackbars

view with a list of available courses. After expanding a course card of the students choice they can view the general and structural course information as well as an *Enroll* button (Figure 3).

When attempting to enroll into a course, the student has to fulfill some mandatory and optional requirements. These requirements are the enrollment deadline and optional prerequisites, being other completed courses. For example a student can enroll into the *Programming 2* course only if he/she meets the requirement to have already completed the *Programming 1* course. The requirements can be found in the structural information tab of the course card as can be seen in. Figure 3.

After attempting to enroll in a course the student receives visual feedback in the form of information snackbars (Figure 4). Depending on the success of the enrollment request the message and color of the snackbar change.

Withdrawing from a course takes a similar sequence of actions. The difference is in clicking on the *My Courses* tab on the top navigation bar instead of *Find Course*. Course cards change according to the view they are being displayed on and the user type, so in this case the enrollment button changes to a *Withdraw* button, and the contents of the card includes a personal tab for the student. It is possible to withdraw from a course only until the enrollment deadline which is considered as the starting date for the course.

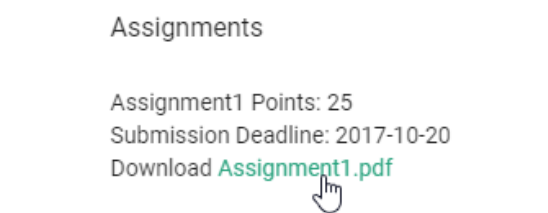


Figure 5: Assignment download link

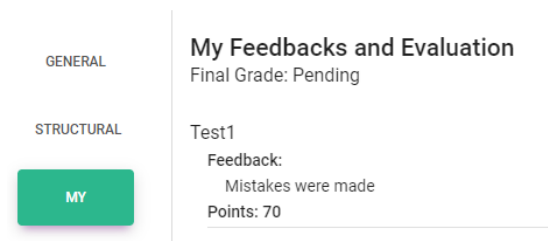


Figure 6: Feedback and evaluation

To download a specific assignment for a course, the user can navigate to either of the two main views. In the structural information tab of a course card is an Assignment section with download links (Figure 5) to assignment pdf files, unless the course does not have any assignments set by the course creator.

Viewing evaluation and feedback is possible via the *My* tab of the course card in the *My Courses* view as can be seen on Figure 6. When the student receives a grade for the course, it is displayed under the course name of the course card in the Structural information course card tab and in the *Study Records* view accessible via the top navigation bar. The *Study Records* view displays a table with all of the study performances of the student in a simple tabular form with crucial course descriptors as table column headings.

3.3.2 User: Teacher

Functionality for the teacher user type consists of creating courses, evaluating student assignments and tests with the option to submit feedback to with each evaluation category and grade students under the condition the student has received all of the required evaluations predetermined at course creation.

Upon clicking on the *Create Course* button on the top navigation bar, the teacher is shown a create course form containing various fields to define the course (Figure 7). The form is divided into two main sections, the General section and the Grading Scheme section. Most of the fields in both sections have to be filled out, with the exception of the Course Description and the Requirements fields of the create course form. Each field has its own description to be easily distinguishable and error checked, e.g. the teacher will not be able to

General

Course ID

Abr.

Course Name

Student Count

Course Semester

ECTS

Course Description

Grading Scheme

Assignment Count

2

Grade Percentage

50

1. Points:

Submission Date

2. Points:

Submission Date

Test Count

2

Grade Percentage

50

1. Points:

2. Points:

Grade 1 Percentage:

90

Grade 2 Percentage:

80

Grade 3 Percentage:

70

Grade 4 Percentage:

60

Requirements

Enrollment Deadline

SUBMIT

Figure 7: Create Course form

ERROR: Check Fields for errors: Course ID, Course Type, Course Name Abreviation, Course Name, Student Count, Course Semester, ECTS, Course Enrollment Deadline,

Figure 8: Error information snackbar

enter characters into the Student Count and ECTS fields of the General section.

The teacher can create a custom grading scheme in the appropriately named Grading Scheme section. By increasing the assignment count, additional fields appear allowing the teacher to upload assignment files via an upload button, assign the maximum points to be earned for the assignments and pick a submission date from a graphical date picker when clicking on the submission date field. Upon choosing an assignment file to be uploaded its name is shown when the teacher hovers over the upload button and the delete button next to the upload button enables, which removes the assignment file. Increasing the test count field demonstrates a simpler behaviour, by only adding fields with the maximum points achievable for each test.

After filling out the form with all of the necessary data, the teacher can click on the *Submit* button. The form inputs are validated and a information snackbar is displayed. In case some of the fields contain errors an error notification is displayed with the information about which fields are either missing or contain errors (Figure 8).

Upon a successful creation of the course, the course is added into the system,


GENERAL	Students in Course		
STRUCTURAL	Matriculation ID	First Name	Last Name
STUDENTS	01301624	Roman	Karaba 

Figure 9: Student tab of in a course card

stored into the chain and the teacher can view it in the *My Courses* view. The teachers version of the course cards contains a *Students* tab which displays all of the enrolled students in the course with an evaluation button on the right side of each student entry (Figure 9).

Clicking on the evaluation button opens up the evaluation and feedback dialog with fields corresponding to the respective assignments and tests of the course (Figure 10). The teacher can then proceed to fill out the feedback text field and point field. All of the past evaluations and feedbacks of the student in this course are automatically filled out in the fields. This dialog serves also the purpose for submitting the final grade of the student. Unless all of the evaluation categories are filled out and submitted the teacher wont be allowed to grade the student and a error snackbar is displayed informing the teacher to evaluate the student in all categories first. When all of the evaluations are filled out and submitted, the grade field calculates the final grade depending on the grading criteria set up at the creation of the course enabling the submission of the grade. If the student has been already graded an error snackbar appears with the information about the student already receiving a grade in the past. Its not possible to change the grade via this dialog and has to be changed manually.

4 Implementation

4.1 Front-end

Because of the decision to implement the front-end with React a builder and code transcompiler was necessary. React uses its own JSX syntax which is in its essence identical to JavaScript, with the exception of being able to write HTML code without the use of string quotations and the ability to perform logical operations within the HTML code as can be seen on Figure 11

Current browsers don not support JSX, nor most of the newest ECMAScript standards natively and therefore all of the JSX and javascript source files have to be transcompiled into basic JavaScript. Webpack with the Babel compiler are used for this purpose. To properly set up the Webpack configuration for this task the create-react-app tool developed by Facebook has been used. It sets up the Webpack configuration with all necessary Babel polyfills, creates a basic folder structure and a basic hello react web app.

Evaluate Student

Test 1

Feedback

Mistakes were made

Points (Max: 100)

70

Calculated from assessed Test and Assignment points

Final Course Grade

3

SUBMIT GRADE

BACK

SUBMIT (POINTS AND FEEDBACK)

Figure 10: Feedback, point evaluation and grading dialog

```

render(){
  const { classes, courseData } = this.props;
  return (
    <GridContainer spacing={16} className={classes.content}>
      <GridItem xs={12}>
        <Typography variant="display1">My Courses</Typography>
        <Divider />
      </GridItem>
      <CourseCard
        course={tooltipCard}
        tableDescription
        style={{backgroundColor: "rgb(53, 224, 172)}}/>
      {
        courseData !== undefined ?
        courseData.map((obj, i) =>
          <CourseCard
            course={obj.course} key={i} courseCardType={this.props.courseCardType}
          />
        ) : null
      }
    </GridContainer>
  );
}

```

Figure 11: MyCourses component render function (JSX syntax)

Our single page application has 6 pages in total. The Login, Signup, Create Course, Find Course, Study records, and My Courses page. Routing between those internal pages is achieved with help of the ReactRouter from the react-router-dom package.

After the user logs in, the front-end recognizes what type of user is logged in and uses the user type to render content appropriately which results in the student and teacher being able to see only their relevant content, and being able to only send requests appropriate to their role. This functionality is most apparent in the CourseCard component thanks to which the course card contents differ with the individual student data being displayed in his/her course details and the students list displayed to the teacher. The user type also modifies the navigation bar with a different version displayed for the student and teacher.

All of the requests to the back-end server are defined in the appService.js script file. All of the requests are HTTP POST requests, upon a successful login, the user receives a JSON Web Token (JWT) from the back-end. This JWT is then stored in the local storage of the browser which results in a auto-login functionality upon opening the app after closing the browser window. The JWT is also used to authenticate for any request by sending the JWT in the post request payload. This token has a 120 minutes expiration duration after which the token gets deleted from the local storage and the user is redirected to the login page.

Component styling of the application was implemented with the help of using the pre-made Material UI components library and the Material Kit React library, as well as writing custom JavaScript Style Sheets (JSS) which are then parsed through a sass compiler to apply color themes and color gradients to the components. Some inline styling is also present in cases where using a class oriented approach would be inefficient.

4.2 Back-end

The back-end of this application consists of multiple components. The Express.js/Node.js server, MultiChain server and the MongoDB server. Code of the back-end is written according to the ECMAScript 2017 standard and utilizes its functional additions that are not present in standard JavaScript, e.g. object deconstruction, async/await, const/let variable declaration, arrow function syntax, import/from statements and exporting modules. Webpack and Babel had to be again used for transcompilation since, Node.js version 8 LTS does not support the ECMAScript 2017 standard.

4.2.1 Express/Node server

REST API endpoints are defined in the server.js script file which defines the server run-time. The server handles all of the requests from the client with the requests split to two handlers, an authentication handler and a data handler. Three endpoints are assigned to the authentication handler, being login,

signup and a general authentication endpoint. This handler is responsible for communication with the MongoDB database and user verification.

To increase the security of this app, passwords are not stored in any database. The received password from the signup request is hashed using the bcrypt algorithm with 10 rounds of salting and the resulting hash stored in the database. In case of an unauthorized access to the database, an attacker would only see the hashes which are unusable for verification. The signup creates the user and initializes the user object which is subsequently then stored into the chain.

When a user logs in, his credentials are again hashed and compared to the hash stored in the MongoDB database. Upon a successful verification a JWT is generated containing the users name, user type, user ID, generation time stamp, expiration time stamp and is signed by the server. The servers signature is then used for authentication when the user requests or submits any further data from the client.

User data retrieval and data update contained in the chain are handled purely by the data handler. In total eleven REST API endpoints were implemented for this purpose. The first step of every function handling the respective endpoint is verifying the existence and validity of the JWT contained in the request. If the JWT is invalid, the server responds with a HTTP 401 code for the unauthorized attempt.

The data handler also handles file storage and retrieval. When a teacher submits assignment files when creating a course in the course management system, the handler creates directories in which to store the assignment files if they are not present as well as a directory for the specific course in the file system at `C:/MultiPassData/Assignments/`. The assignments files are also retrieved from the respective directory upon download request from the client.

4.2.2 MultiChain

The MultiChain node, multipass, utilized for immutable data storage contains two data streams, a user data stream and a course data stream. Data stored in these streams is stored in a base64 format as the 1.0 version of MultiChain does not support JSON string data storage. This is handled by encoding and decoding the base64 format for every interaction with the chain by the backend server. By default its configuration is a permissioned (private) chain and accepts JSON-RPC API calls only locally, thus remote rpc calls are not possible in this configuration for improved security.

Every new data entry is stored in its respective data stream with the user ID as key for users and a combination of the course ID and semester for the courses. Data manipulation is possible by retrieving the whole object, modifying it and republishing it into the chain under the same key, which guarantees the integrity of change history of data and presents the possibility to view the data history.

The configured maximum data entry size is 4MB which should suffice for any data entry executed in the system, but in case more would be needed MultiChain 1.0 supports up to 64MB for a data entry.

4.2.3 Utilities

During the development utility scripts have been written to extend and simplify the implementation and deployment of the system. Helper scripts for JWT generation, verification and its decoding when processing client requests.

The `initApp.js` which is used at the deployment stage loads users from the `users.JSON.js` file and utilizes the implemented signup functionality to create the users and save them into the chain. Initializing the by running this script is recommended when deploying and running the app for the first time, as it handles the creation of the data streams used in the system. The script is runnable by executing `npm run initapp` in the Backend directory via the CLI.

Another useful script developed for debugging the system is the `chainInfo.js` script that queries the MultiChain blockchain used in the system to log the contents of the chain. The script is runnable by executing `npm run chaininfo` in the Backend directory.

4.3 Deployment

The project repository can be found at <https://git01lab.cs.univie.ac.at/a1301624/MultiPass>.

Prerequisites for the local demo deployment:

- Node.js v8 LTS
- MultiChain v1.0
- MongoDB v4.0
- Git

4.3.1 Installation (local)

1. After installing MongoDB, it is recommended to add its installation location to your PATH, which enables the usage of `mongod` and `mongo` CLI commands for controlling and querying the database. The `C:/data/db` directories have to be created manually. After creating the directory run the MongoDB database by executing `mongod` in the CLI or by running the MongoDB executable
2. After installing the MultiChain server, it is necessary to add its directory into the environment PATH for the next steps. Execute `multichain-util create testchain` and `multichaind testchain -daemon` in the CLI to initialize MultiChain
3. Terminate the testchain instance and navigate into the MultiChain folder at `C:/Users/%USERNAME%/AppData/Roaming/MultiChain` and add the line `autosubscribe=streams` into the `multichain.conf` file.

4. Create the multipass MultiChain instance by executing `multichain-util create multipass`, and run it by executing `multichaind multipass -daemon`. Notice the API request port in the console.
5. Install Webpack globally by executing `npm install -g webpack`
6. Execute `npm install` in the Frontend and Backend directories via the CLI
7. Copy the generated MultiChain password from `C:/Users/%USERNAME%/AppData/Roaming/MultiChain/multipass/multichain.conf` and set it in the `Multipass/Backend/src/multichainOptions.js`
8. Run the back-end and front-end by executing `npm run frontend` in the Frontend directory and `npm run backend` in the Backend directory
9. Execute `npm run initapp` in the Backend directory

The app should now be running on your machine with the front-end on `localhost:8080`

4.3.2 Installation (global)

The installation and deployment process for a global deployment, where the app is reachable from any network, follows almost the same deployment process as the local installation. The main difference is in adjusting the front-end `appService.js` script file located in `MultiPass/Frontend/src/scripts/` by changing the IP address at the beginning of the file to the IP address of the back-end Node.js/Express server.

It is also recommended to execute `npm run build` on the front-end to build an optimized front-end bundle. The optimization runs the source file through a minification process which improves the performance of the client user interface. Afterwards just serve the bundle via any static web server, I used the `serve` npm package for this purpose.

5 Evaluation and Discussion

I evaluated the result of this project by manual testing and using the provided MultiChain API to query the chain and view the saved data history and its recallability. I conducted a small sample user case study and let my peers test the application to gather feedback.

Evaluation categories consisted of interface design, application responsiveness, confusion and attractiveness of the application. Interface design evaluating the overall looks of the app, application responsiveness evaluating the feedback of the application, confusion evaluating the difficulty of understanding the UI and attractiveness in interest of using the app in the future. The general consensus about the implementation was rather positive with an average score of 7/10 for

the interface design, 8/10 in application responsiveness, 4/10 in confusion and 7/10 in the attractiveness category.

The application performed well at least when judged by empirical experience. In theory it should perform well with a larger amount of data when deployed because of the design and functionality offered by MultiChain. Using a Proof-of-Work consensus system in a private organization, while securing the validity of data without having to trust a certain authority does offer a certain positive aspect, but the performance trade-off is not worth it, especially when a malicious and misbehaving authority can be easily stripped of status and access into the network.

6 Conclusions and Future Work

In this thesis I proposed a course management system to be used by teachers and students. The implementation successfully utilized a permissioned (private) blockchain platform with the positives of data validity and integrity. Arguments were made about why Proof-of-Authority and the utilization of a blockchain as a distributed database in a system is viable.

Work that could be done on this particular project in the future would consist of adding more functionality to the front end like sorting courses in the different views according to some predefined filters, conduct load testing on the system and add more management functionality for the teacher.

It would be also interesting to implement various versions of this app by utilizing different blockchain frameworks like Hyperledger. Hyperledger, a permissioned (private) blockchain system developed by the Linux Foundation offers the functionality of splitting up the network into multiple regions to uphold to stricter privacy and visibility rules of the chain. Hyperledger also shows higher performance, up to 3500 transactions per second, which scaled up to 100 nodes on the network[22].

One of the approaches to further research the viability of a blockchain as database would be implementing a similar or the same application using the newer version of MultiChain(2.0), which is currently in the prerelease phase, and deploying multiple nodes on a network to test the behaviour of such app on a network.

Using blockchain as a database does make sense in a network of multiple nodes, but using it as a database with only a single node would miss most of the advantages that it can offer. A standard relational database or a document database will with certainty perform better in such scenarios. The advantage of data immutability and native content updates across the network would not be utilized. Guaranteeing the immutability of the data is not possible in such a circumstance since the entity controlling the node does not answer to any peers on the network and can freely read, delete, modify and rewrite the contents of the chain without anyone noticing and holding this entity responsible for the malicious intent, and having only one instance of the chain does not in any way take the advantage of updating the database across the network since there is

no real network.

7 Acknowledgements

I would like to express my sincere gratitude to my supervisor and advisor Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas for the continuous support and guidance during the work on this thesis. My gratitude also goes to my work colleagues and peers for helping me with brainstorming and their feedback on the implementation of this system.

References

- [1] Satoshi Nakamoto. “Bitcoin: A peer-to-peer electronic cash system”. In: (2008).
- [2] Don Tapscott and Alex Tapscott. *Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world*. Penguin, 2016.
- [3] Arthur Gervais et al. “On the Security and Performance of Proof of Work Blockchains”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: ACM, 2016, pp. 3–16. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978341. URL: <http://doi.acm.org/10.1145/2976749.2978341>.
- [4] Stefano De Angelis et al. “PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain”. In: (2018).
- [5] Gideon Greenspan. *MultiChain private blockchain-White paper*. 2015. URL: <http://www.multichain.com/download/MultiChain-White-Paper.pdf>.
- [6] Gideon Greenspan. *MultiChain Roadmap*. URL: <https://www.multichain.com/blog/2017/06/multichain-1-beta-2-roadmap/>.
- [7] Kyle Croman et al. “On scaling decentralized blockchains”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2016, pp. 106–125.
- [8] Ken Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.
- [9] David J Anderson. *Kanban: successful evolutionary change for your technology business*. Blue Hole Press, 2010.
- [10] Tsung-Ting Kuo and Lucila Ohno-Machado. “ModelChain: Decentralized Privacy-Preserving Healthcare Predictive Modeling Framework on Private Blockchain Networks”. In: *arXiv preprint arXiv:1802.01746* (2018).
- [11] Sara Cohen and Aviv Zohar. “Database Perspectives on Blockchains”. In: *arXiv preprint arXiv:1803.06015* (2018).
- [12] Wenting Li et al. “Towards scalable and private industrial blockchains”. In: *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM. 2017, pp. 9–14.
- [13] Karl Wüst and Arthur Gervais. “Do you need a Blockchain?” In: *IACR Cryptology ePrint Archive 2017* (2017), p. 375.
- [14] Coin Sciences Ltd. URL: <https://www.multichain.com/qa/5043/multichain-scalability>.
- [15] GitHub Inc. URL: <https://electronjs.org>.
- [16] Node.js Foundation. URL: <https://nodejs.org/en/>.

- [17] ECMA International. *ECMAScript 2017 8th edition*. URL: <https://www.ecma-international.org/publications/standards/Ecma-262-arch.htm>.
- [18] Niels Provos and David Mazières. “A Future-adaptive Password Scheme”. In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. ATEC '99. Monterey, California: USENIX Association, 1999, pp. 32–32. URL: <http://dl.acm.org/citation.cfm?id=1268708.1268740>.
- [19] Google Inc. *Material Design*. URL: <https://material.io/>.
- [20] Creative Tim. *Material Kit React*. URL: <https://www.creative-tim.com/product/material-kit-react>.
- [21] Michael Jones, John Bradley, and Nat Sakimura. *JSON Web Token (JWT)*. RFC 7519. May 2015. DOI: 10.17487/RFC7519. URL: <https://rfc-editor.org/rfc/rfc7519.txt>.
- [22] Elli Androulaki et al. “Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains”. In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys '18. Porto, Portugal: ACM, 2018, 30:1–30:15. ISBN: 978-1-4503-5584-1. DOI: 10.1145/3190508.3190538. URL: <http://doi.acm.org.uaccess.univie.ac.at/10.1145/3190508.3190538>.