University of Basel

Decentralized Kanban board using Secure Scuttlebutt and Conflict-Free Replicated Data Types

Bachelor Thesis

Faculty of Science
Department of Mathematics and Computer Science
Computer Networks Group
https://cn.dmi.unibas.ch

Examiner: Prof. Dr. Christian Tschudin
Supervisor: Fabrizio Parrillo, MSc.

Jannick Heisch
jannick.heisch@stud.unibas.ch
19-065-720

17.07.2022

Acknowledgments

I would like to thank Prof. Dr. Christian Tschudin and Fabrizio Parrillo, who made it possible for me to work on this project and gave me dedicated support and guidance during this time. I am particularly grateful for the constructive feedback, which was very helpful and motivated me.

Abstract

In the beginning, the World Wide Web was characterized by a decentralized structure, but it became increasingly centralized over time. Almost all applications we encounter in everyday life are based on a central server-client architecture and require a continuous connection to the Internet. These applications are user-friendly and easy to implement, but user data is stored on individual servers usually maintained by companies that profit from this data.
Well-known digital Kanban board applications, which are very popular in the business world for organizing or optimizing workflows in the form of lists and cards, are also built on such centralized structures.
This thesis examines how digital Kanban boards can be realized using an alternative approach, namely in the form of a decentralized structure in which users host their own data and are independent of the Internet. For this purpose, the existing android app Tremola, a Secure Scuttlebutt implementation, was extended by the prototype of a decentralized Kanban board. The main goal of this project is to ensure an eventual consistency between the board states of the individual users, so that the same board is displayed to everyone. Therefore, the board modifications performed by the users are saved as Conflict-free Replicated Data Types in an append-only log and shared with all participants using the Secure Scuttlebutt protocol.

Table of Contents

1 Introduction

Since Tim Berners-Lee invented the World Wide Web in 1989, it has rapidly gained importance. In 2021, an estimated 63% of the world's population used the Internet, with rising tendency [20]. It provides easy access to information on numerous topics and allows us to communicate with our friends and family via social media. For many people, the Internet has become an integral part of daily life.
For Tim Berners-Lee, it was important that everyone should be able to use the Web without being dependent on a central authority. In his book "Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor" (1999), in which he describes the development of his invention, he wrote the following:

> *I made it clear that I had designed the Web so there should be no centralized place where someone would have to "register" a new server, or get approval of its contents. Anybody could build a server and put anything on it.* [10]

As Tim Berners-Lee points out, he sees the decentralized structure as the central component of the World Wide Web. Considering his basic idea, the current concept of the Internet needs to be questioned.
Although the World Wide Web is decentralized by design, it has become increasingly dominated by centralized structures. This development has been significantly influenced by a few large companies, especially social media platforms such as Facebook, Twitter and YouTube that hold a quasi-monopoly status.
Nowadays, most of the services we use every day are based on a centralized approach: the data is stored on the providers' central servers. Even though these services bring many advantages, they share some crucial disadvantages. All user data is stored on the servers of the providers. The user himself has lost control over his own data and can no longer decide how it will be stored and to what extent it may be used. In consequence, individual companies are gaining more and more power, as they can decide how to use their customers' data.
An example of the abuse of this power is the Facebook-Cambridge Analytica data scandal, which became public in 2018. A former Cambridge Analytica employee revealed that data from 50 million Facebook users had been misused without the users being aware of it. The

data was mainly used for political purposes, for example for Donald Trump's presidential campaign in 2016 [15].
Furthermore, the centralized structure provides an easy way, especially for governments, to censor the web and restrict access to certain content. Internet censorship threatens the right to free expression and the opinions of minorities or independent media. The "Great Firewall of China" is one of many examples of governments restricting access to the Internet. It is used by the Chinese government to block access to certain foreign internet services in order to "protect" its citizens, which is controversial because this process tends to control and suppress other opinions that are not compatible with the government's way of thinking [8].
The growing awareness of these issues caused by the current structure of the World Wide Web is leading to the emergence of new initiatives that attempt to "re-decentralize" the current Web and thus return power to the users.
One of these approaches is Secure Scuttlebutt (SSB) [4]. It is a decentralized protocol where each user hosts their own data and replicates it over the network via peer-to-peer connections without the intervention of a central authority. Since the user is in control of their own data, they can decide how to share it and are not dependent on providers' servers or a connection to the Internet.
This thesis presents a decentralized Kanban board application based on the Secure Scuttlebutt protocol and Conflict-free Replicated Data Types. The prototype allows multiple users to collaborate on the same board. The main goal of this work is that all participants, after making various changes to the board, will have a consistent board displayed again.

1.1 Contribution

Many known Kanban boards are offered as cloud services, where users can register and work together on a board. The data is stored on a central server that also coordinates the communication between the clients. If the connection to this server is interrupted or restricted by a third party, it is no longer possible to continue working on the board.
A solution to this issue offers a Kanban board based on a decentralized structure. Secure Scuttlebutt provides a secure and decentralized platform on which various applications can run. Since no decentralized board based on the Secure Scuttlebutt protocol is known yet, this thesis presents such a decentralized digital Kanban board application. It is implemented in the Android Secure Scuttlebutt implementation Tremola [7].
The prototype supports all major Kanban board operations and enables multiple users to collaborate on the same board. The data is hosted by the users and can be shared through several possibilities, such as peer-to-peer connections via Wi-Fi or Bluetooth, so that users do not have to rely on a continuous connection to the Internet.
To store changes, Conflict-free Replicated Data Types allow the different board states of several participants to be merged into a consistent state. As a result, each user will see the same board after the data exchange. The Secure Scuttlebutt Protocol is used to store and exchange board operations-data in a decentralized way.

With the help of these technologies, no central server is needed to store data and coordinate the communication between the users. The board can be modified offline, these changes will later be exchanged with all participants.

1.2 Thesis Outline

This thesis is structured as follows:
The next chapter gives an overview of the technologies used in this project and presents their main concepts.
The features and implementation of the decentralized Kanban board is described in chapter 3. The focus is particularly on how these methods realize the decentralized structure.
Finally, chapter 4 summarizes the results and provides an outlook on future projects and tasks which are opened up by this thesis.

2 Background

To realize a decentralized Kanban board, several technologies were used. As mentioned in the introduction, two main functionalities, Secure Scuttlebutt and Conflict-free Replicated Data Types, are used to realize a decentralized Kanban board. In addition, topological sort of graphs is a useful method that was implemented in the application to help sorting board operations.
The following chapter provides an overview of these fundamental concepts and technologies.

2.1 Secure Scuttlebutt

The Secure Scuttlebutt protocol was invented by Dominic Tarr in 2014. At the time, he lived in a sailboat, which significantly shaped the characteristics of SSB. During his stay on the sailboat, he had no permanent connection to the Internet. This situation resulted in the idea of developing a decentralized protocol that would allow data to be exchanged independently of the Internet [9]. Furthermore, the applications running on SBB handle the transmission of data without the need for a central server.
The main features of Secure Scuttlebutt are described below.

2.1.1 Append-only log

Each user of the Secure Scuttlebutt protocol owns an Ed25519 key pair. In contrast to usual applications, a user is not identified by a name or an e-mail address, but by his public key. The data is stored in append-only logs, the so-called feeds. Each user has a feed in which he can add new messages, besides he receives the feeds that have been shared by other peers. In addition to the actual content, each message of a feed also contains a timestamp of its creation, the author's public key, a sequence number, the hash value of the previous message and the signature of the message [13].
The first entry of a feed always starts with the sequence number 1, each following message is assigned a next higher number (e.g. first message is assigned to sequence number 1; second message is assigned to sequence number 2 and so on). A message is signed by the author's private key, which enables the other peers to verify if this message actually originates from

the indicated author and if the content has not been changed subsequently.
In addition, a cryptographical hash value of the previous message is calculated and stored. These so-called backlinks create a Linked List-like data structure. Since the backlink is also signed as part of the message, it cannot be subsequently modified. Therefore, it is only allowed to append a new message at the end of a feed, but not to insert or delete entries in the log. Furthermore, you can use the backlinks to determine a total order within a feed. Since messages can only be appended to the end of a log and the sequence cannot be changed, the order of the messages is clearly determined and fixed.
The additional information stored in each message creates a single-writer append-only log: A user can only append a message to the end of his own feed. Modifying or deleting existing messages or creating a backlink to another feed can be easily detected and will result in an invalid log. Such feeds are rejected by the other Secure Scuttlebutt peers and are not replicated further. Figure 2.1 shows an example of such an append-only log, where the individual log entries are chained together by backlinks.

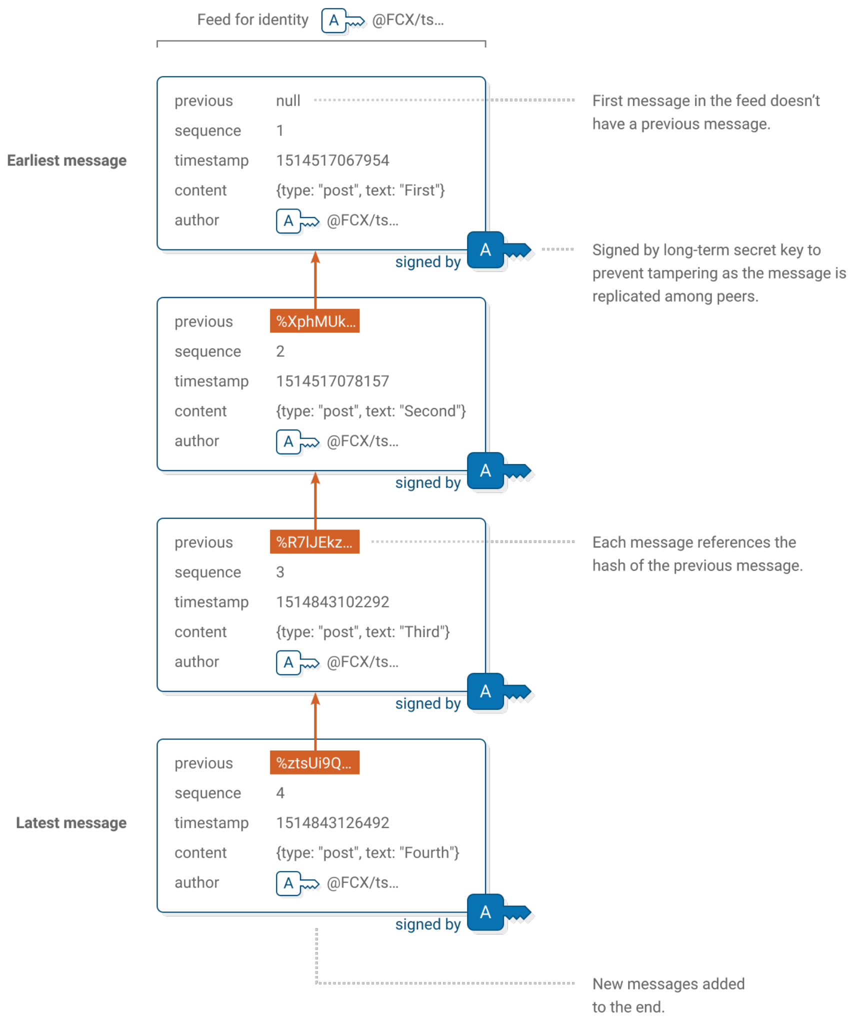

Feed for identity A @FCX/ts...
Earliest message
previous null
sequence 1
timestamp 1514517067954
content {type: "post", text: "First"}
author A @FCX/ts...
signed by A
First message in the feed doesn't have a previous message.
Signed by long-term secret key to prevent tampering as the message is replicated among peers.
previous %XphMUk...
sequence 2
timestamp 1514517078157
content {type: "post", text: "Second"}
author A @FCX/ts...
signed by A
previous %R7lJEkz...
sequence 3
timestamp 1514843102292
content {type: "post", text: "Third"}
author A @FCX/ts...
signed by A
Each message references the hash of the previous message.
Latest message
previous %ztsUi9Q...
sequence 4
timestamp 1514843126492
content {type: "post", text: "Fourth"}
author A @FCX/ts...
signed by A
New messages added to the end.

Figure 2.1: Example of a feed in Secure Scuttlebutt [3].

2.1.2 Replication

While most applications use a central server to coordinate communication between clients, the Secure Scuttlebutt protocol is based on a gossip protocol. Assuming that the user is not interested in all global data, but only in a certain part of it, everyone can determine which content from which peers are relevant to them. This creates a social graph (Figure 2.2). Therefore, not every client in Secure Scuttlebutt has to handle the entire global data pool, but only from those people the user follows. To replicate new messages, a secure peer-to-peer connection is established between clients, through which updates are exchanged. By comparing the sequence numbers of the individual messages in the feeds, new messages can be efficiently identified, which are then replicated.

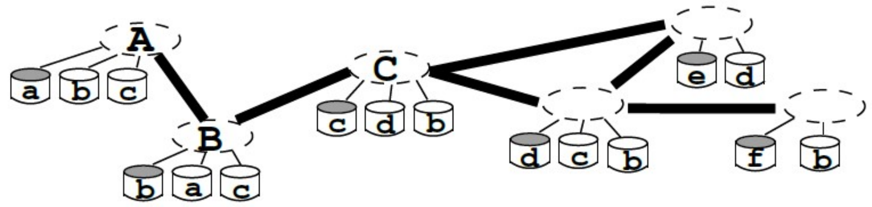


Figure 2.2: Social graph: A and B follow each other so A contains the log b. A does not have a log c, because A does not follow C. If A and C follow each other, A will receive the log c via B [13].

2.2 Conflict-free Replicated Data Types

Collaborative editing systems allow multiple users to work on the same document simultaneously. But concurrent updates of multiple users without coordination can lead to inconsistencies between the local replicas of the participants.There are several approaches to resolve this problem:
The first option provides the possibility to serialize the updates to guarantee strong consistency. All replicas execute the operations in a single global order. While one user is editing the document, a lock is set so that the other participants have to wait until it is released. Afterwards, another user can continue working on the document. This method prevents concurrent modifications and therefore leads to a poor user experience and does not scale well.
Another approach is based on the Operational Transformation (OT), which is used by the most common Collaborative editing systems like Google Docs. The idea behind Operational Transformation is shown in Figure 2.3. Two users perform an operation (O1, O2) on the same document simultaneously. Since the two users have now a different local state, the execution of the other user's operation would lead to an inconsistency between the two local states, because "the actual effect of an operation at the time of its execution may be different from the intended effect of this operation at the time of its generation" [14]. To avoid inconsistencies, the user's operation must first be transformed (O1', O2') before it is

executed.
In contrast to the first approach, no resource has to be locked and the concurrent modification of the same content is possible. But on the other hand, the transformation process requires additional calculations. Furthermore, a decentralized implementation of OT is very complex, so that a central server is often used to compute these transformations [14].

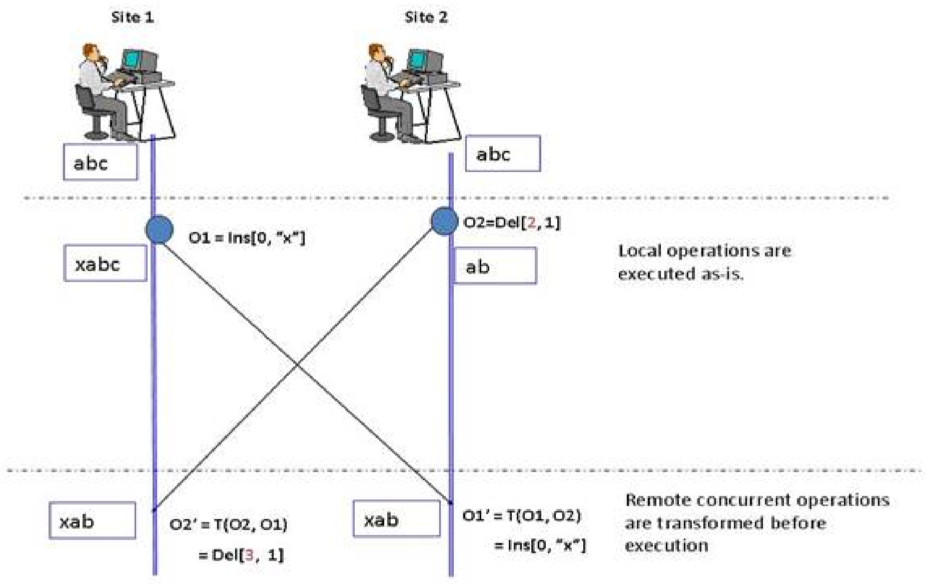

Site 1
Site 2
abc
abc
O1 = Ins[0, "x"]
O2=Del[2, 1]
xabc
ab
Local operations are executed as-is.
xab
O2' = T(O2, O1) = Del[3, 1]
xab
O1' = T(O1, O2) = Ins[0, "x"]
Remote concurrent operations are transformed before execution

Figure 2.3: Example of two users using Operational Transformation to achieve consistency [18].

The latest approach are Conflict-free Replicated Data Types. This data structure enables several users to work on a document without coordination. The replicas may have different states, but after all operations are finally executed in each replica, they converge to a consistent state without concurrency control. There are two methods of CRDTs:

2.2.1 State-based Convergent Replicated Data Type (CvRDT)

In CvRDTs, each replica performs updates to its local state independently of the other replicas. Replicas transmit their complete local state to other replicas, which combine the received state with their own using a merge function (Figure 2.4) .
This merge function needs to fulfill the Least Upper Bound properties: The merge must be commutative and associative, whereby the order of the received updates is irrelevant and leads to the same result. In addition, the function must be idempotent, which means that the result is not affected if the same update occurs several times. This merge function combined with the set of all states forms a Semi-lattice. Furthermore, the state must be non-decreasing across updates. A state-based object which fulfils these requirements satisfies the monotonic semilattice property and guarantees strong eventual consistency [17].

CvRDTs only require a reliable transmission channel, the order in which the updates are received does not affect the result. But each update covers a complete state of a remote replica, which means that more data has to be exchanged.

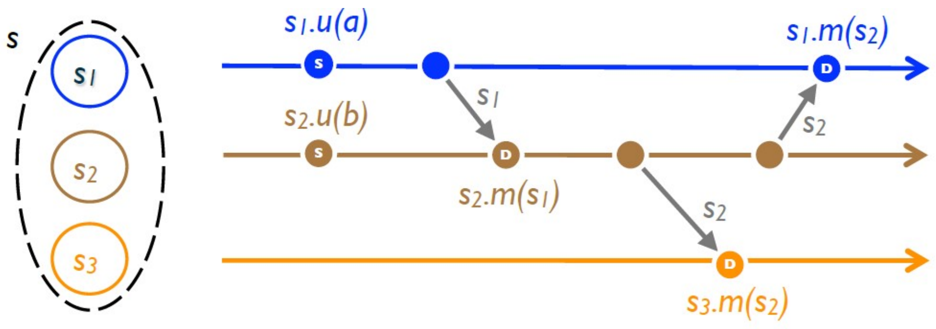


Figure 2.4: State-based replication [17].

2.2.2 Operation-based Commutative Replicated Data Type (CmRDT)

Instead of transmitting the entire local state as in the state-based approach, CmRDTs only transmit individual update operations (Figure 2.5). CmRDTs do not require a merge function, yet certain conditions are necessary for the transmission of updates. One important requirement is a causally ordered broadcast communication protocol to ensure that operations are transferred in causal order. Any duplication or loss of data must be prevented so that each update is transmitted exactly once to each replica [17].
As a result, operation-based replication is more efficient than state-based replication since only small updates, but not entire states, need to be transmitted. On the other hand, however, this faces complex requirements on the communication protocol.

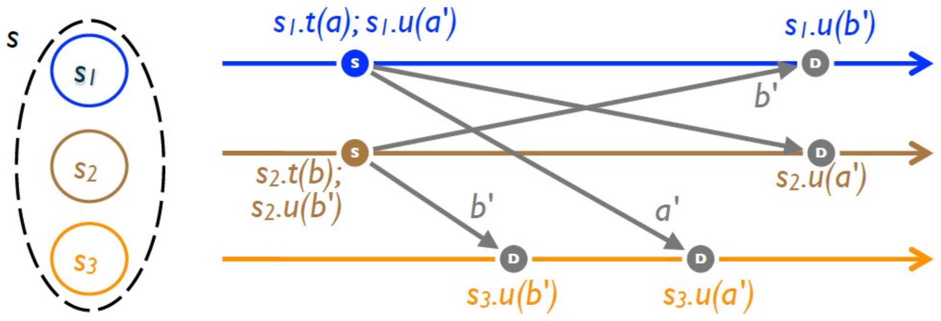


Figure 2.5: Operation-based Replication [17].

2.3 Topological Sort

The Topological sort of a directed acyclic graph G is a linear order of all its vertices in such a way that for every directed edge (u,v), u is placed before v. A graph is a directed acyclic graph if all its vertices are directed edges and if there exists no path where the start vertex is the same as the end vertex. Directed graphs which contains cycles cannot be ordered by topological sorting.
Any direct acyclic graph has at least one linear order, but can also have more [12]. Figure 2.6 shows a directed acyclic graph with multiple topological orderings.

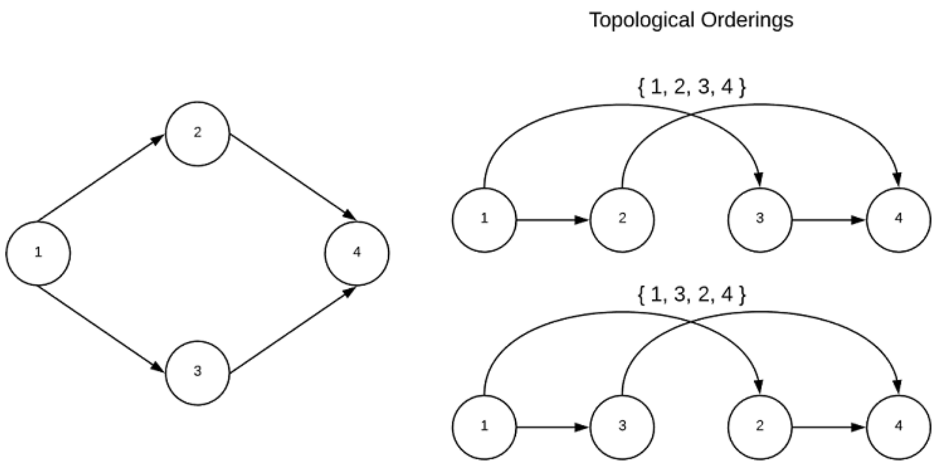


Figure 2.6: Topological sorting of a Directed acyclic graph [5].

2.3.1 Kahn's algorithm

Kahn's algorithm performs a topological sorting by finding a list of ”start nodes” that have no incoming edges and inserts them in the list S. When sorting a directed graph, at least one such “start node” must exist, otherwise it is a directed graph with a cycle which cannot be sorted topologically.
The following pseudo code describes the Kahn's algorithm [16]:

```
$L$ ←Empty list that will contain the sorted elements
$S$ ←Set of all nodes with no incoming edges

while S is non-empty do
    remove a node n from S
    insert n into L
    for each node m with an edge e from n to do
        remove edge e from the graph
        if m has no other incoming edges then
            insert m into S

if graph has edges then
    return error (graph has at least one cycle)
else
    return L (a topologically sorted order)
```

Kahn's sorting algorithm returns one non-unique result because directed acyclic graphs can have multiple linear orderings. Since there can be several nodes at the same time that have no incoming edges, the final result changes depending on which of these nodes is chosen next in the algorithm.
For a directed acyclic graph G = (V,E), this sorting algorithm has a linear running time of $O(|V| + |E|)$, because in order to find all "start nodes" each vertex and edge need to be checked [16]. When a new vertex is appended to the graph, the algorithm has to start all over again to find a valid order.

2.3.2 ScuttleSort

In the case of tangled append-only logs such as those used in Secure Scuttlebutt, it would result in poor performance if the entire topological sorting algorithm had to be run from scratch for each new event that arrives.
Scuttle Sort is a topological sort algorithm adapted for append-only logs. It uses the advantageous properties of an append-only log, namely that new entries can only be inserted at the end of the log and that no entries can be moved or deleted. As a result, several tangled attachment-only logs are more efficiently merged into one linearization [19].
Scuttle Sort is based on an Incremental Topological Sort algorithm, in which each sorted element is assigned a rank. In contrast to classical sorting methods, the linear order is maintained, such that the entire graph does not have to be re-sorted when a new event is received.

3 Implementation

The Kanban board was implemented in the Secure Scuttlebutt App Tremola [7]. The backend of Tremola is written in Kotlin. It implements the Secure Scuttlebutt protocol by providing a database for the feeds and securely replicating the feeds to other peers. Received events are forwarded to the frontend, which is implemented in JavaScript and HTML. The frontend is responsible for the user interface as well as the processing of user input.
The application logic of the Kanban board is implemented in JavaScript and prepares the data provided by the backend for display on the user interface, which is written in JavaScript and HTML. User input is returned to the application logic that prepares the data and passes it to the backend. The backend packages the data into a valid log entry and replicates it.

3.1 Kanban board

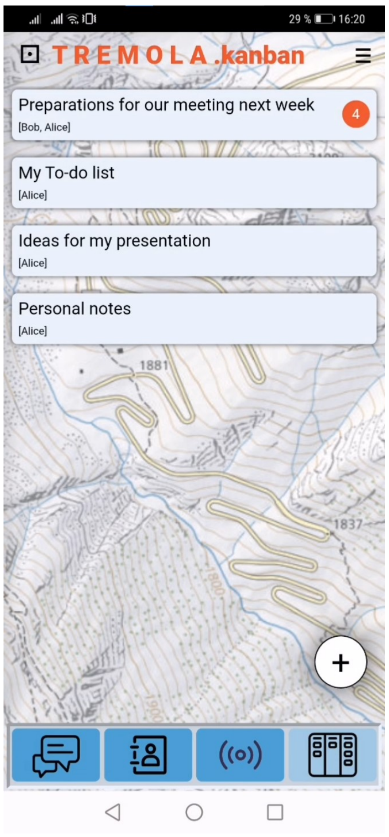
Kanban boards are used to manage and visualize workflows to make them more efficient and to detect bottlenecks. These boards are based on cards organized in different columns. The columns represent the different phases of the work progress. Each card includes the individual tasks or work steps and can be moved between the columns depending on the progress.
There are many different use cases: For personal use, the kanban board helps to structure one's own work and organize daily tasks. In companies, they are often used for improving the management of team projects e.g. in the field of software development, marketing, accounting etc.
Many different applications for digital Kanban boards are available on the market. They differ mainly in various user interfaces as well as some advanced features of the board. But almost all applications have one aspect in common: they are cloud applications based on a client-server architecture. A central server coordinates the communication between the users of a board.
In the following, an alternative Kanban board is introduced, which, in contrast to other applications, is characterized by a decentralized structure. Its concept is based on Secure Scuttlebutt and Conflict-free Replicated Data Types.

3.2 Features

The decentralized Kanban prototype of this thesis enables several users to work on one board without being coordinated by a central server. In addition, all functions of the application can be used offline. It is possible to edit the board without having a connection to the Internet or to other users. The modifications can then be shared with other participants later.
The application supports all basic features of a Kanban board and is particularly designed for mobile devices. Below you will find a brief overview of the most important functionalities:

3.2.1 Board list

A user can be a member of several boards at the same time. The board list gives a clear overview of all boards on which the user is working (Figure 3.1). The list is sorted chronologically, i.e. the board with the most recent modifications appears at the top. To better identify new changes that have not been seen yet, the number of the latest updates are displayed on the board list.
To create a new board, click on the plus icon in the lower left area. After specifying the board's title, you can invite people from the contact list to join this board.
If a Kanban board is no longer relevant to the user, he or she can hide it. It is then no longer displayed in the list.

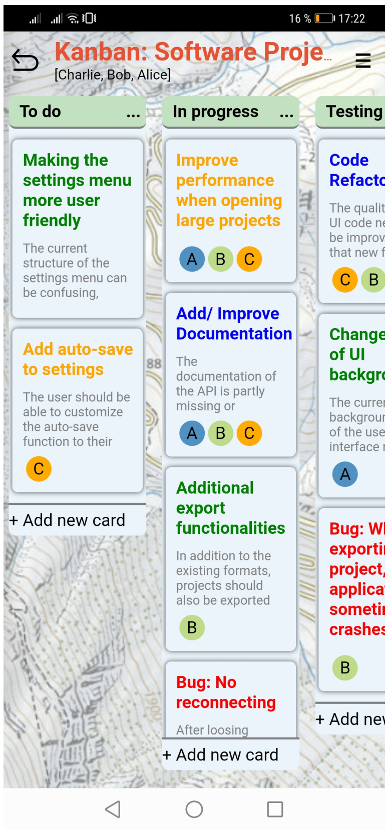


Preparations for our meeting next week 4
[Bob, Alice]
My To-do list
[Alice]
Ideas for my presentation
[Alice]
Personal notes
[Alice]

Figure 3.1: Board list.

3.2.2 Cards and lists

As mentioned above, the two main elements of a Kanban board are the columns and its cards (Figure 3.2). New columns can be created via the options menu in the upper right corner and then appear on the right side of the board. It is possible to rename and delete columns. Furthermore, the order of the columns can be individualized via drag and drop. Cards can be directly assigned to a column. Clicking on a card opens a menu that allows further customization (Figure 3.3): You can change the card's title and description, users can be assigned to the card and exchange further information with each other via the comment function. In addition, the color of the card's title is changeable, for example to highlight certain cards (or tasks) and indicate their priority. By using drag and drop, the cards can be reordered within the column or assigned to another column.



Kanban: Software Proje...
[Charlie, Bob, Alice]

To do ...

Making the settings menu more user friendly
The current structure of the settings menu can be confusing,

Add auto-save to settings
The user should be able to customize the auto-save function to their
C

+ Add new card

In progress ...

Improve performance when opening large projects
A B C

Add/ Improve Documentation
The documentation of the API is partly missing or
A B C

Additional export functionalities
In addition to the existing formats, projects should also be exported
B

Bug: No reconnecting
After loosing

+ Add new card

Testing

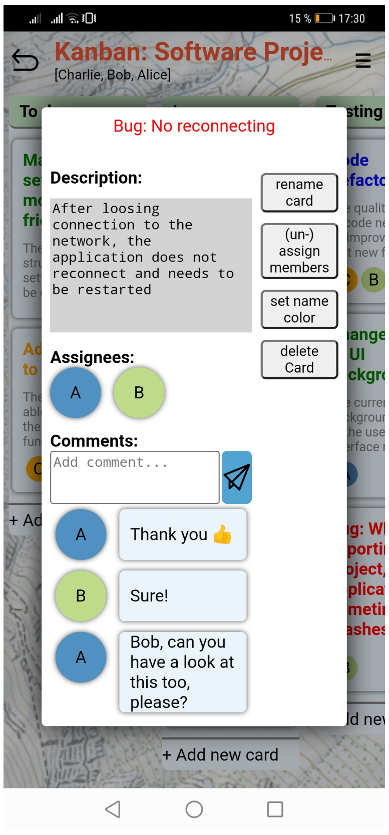
Code Refacto
The qualit UI code ne be improv that new f
C B

Change of UI backgro
The curren backgrour of the use interface i
A

Bug: Wh exportin project, applica sometin crashes
B

+ Add nev

Figure 3.2: Cards and lists of the board.



Bug: No reconnecting

Description:

```
After loosing
connection to the
network, the
application does not
reconnect and needs to
be restarted
```

A: Thank you 👍

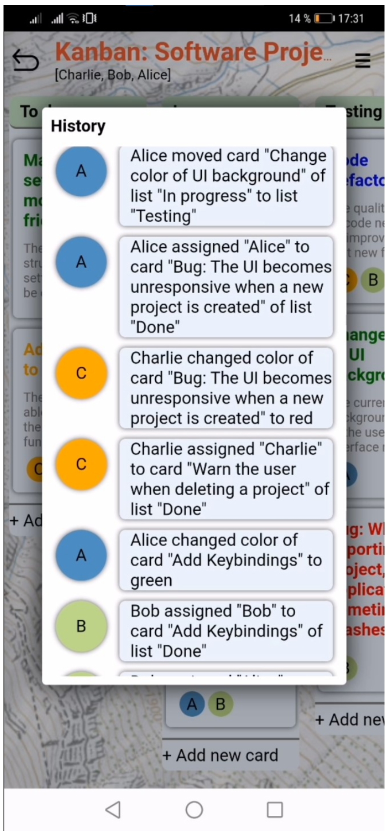
B: Sure!

A: Bob, can you have a look at this too, please?

Figure 3.3: Customization via the card menu.

3.2.3 Other features

A history window (Figure 3.4) can be opened via the options menu in the upper right corner. This overview lists chronologically all changes that have been made to the board. It describes who has performed which operation on which element. In this way, the user has the possibility to follow how the Kanban board has been modified.
If the Tremola app crashes and the Kanban app enters an undefined state, e.g. if it was interrupted while sorting operations, a reload function comes into play. This can be accessed by the user via the options menu. The reload function deletes all local data about the Kanban board and re-executes all operations. This ensures that in such unfortunate cases the user can continue to work on the Kanban board without any further consequences.
The prototype contains an additional debug function which can be called via the menu. A window opens with the display of a graph formatted in DOT [1]. It represents the causality graph of the board operations which will be explained later in this thesis. The individual vertices of the graph represent the board operations and the directed edges are the pointers between these operations.
The debug function is very helpful in analyzing the functionality of the prototype and gives interested users an insight into how this application works behind the user interface.



Alice moved card "Change color of UI background" of list "In progress" to list "Testing"

Alice assigned "Alice" to card "Bug: The UI becomes unresponsive when a new project is created" of list "Done"

Charlie changed color of card "Bug: The UI becomes unresponsive when a new project is created" to red

Charlie assigned "Charlie" to card "Warn the user when deleting a project" of list "Done"

Alice changed color of card "Add Keybindings" to green

Bob assigned "Bob" to card "Add Keybindings" of list "Done"

Figure 3.4: History menu.

3.3 Data Structure

This section gives an overview of all data structures used in the implementation. The first part describes how the board operations are stored via Secure Scuttlebutt messages. The second part outlines the process that combines these stored messages to eventually achieve a consistent board state.

The data structure used in this Kanban board application is based on an operation-based, Conflict-free Replicated Data Type. Instead of sending the entire state of the board to all users, this data type requires only the transmission of the executed modifications of the board.
The transmission process, i.e. the exchange of updates between users, is performed by the Secure Scuttlebutt protocol. Thereby every change to the board is saved as a SSB message and appended to the end of the user's feed. These changes are then replicated to other peers.
All the board operations together form a “recipe” for one unique Kanban board. The basic idea is that each user receives all the performed operations after a certain time, such that everyone will end up“cooking” the same board.

The prototype currently supports the following 14 different operations:

1. Board Operations:
    - create
    - rename
2. Column Operations:
    - create
    - rename
    - remove
3. Card Operations:
    - create
    - rename
    - remove
    - move to column
    - set description
    - post comment
    - assign user
    - unassign user
    - set title color

Each operation contains additional parameters. For example, the board rename operation requires the new name of the board.
The operation performed on the board is saved as a SSB event, which contains, in addition to the metadata described in Chapter 2, the actual content of the message. The latter is stored in a Json dictionary format that contains following elements:

1. The type of the message that allows to identify if this message is relevant for the Kanban board application.
2. The board operation that is stored in an additional dictionary. It includes the ID of the board on which this operation was performed, a list storing the type of operation with its parameters and a previous pointer.

The board ID is the cryptographic hash value of the operation that created this board. The identification is independent of user input, thus two boards could have the same name and still be distinguished. The cards and columns are identified by the board ID and the hash value of their creation message.
Since the backlinks of the Secure Scuttlebutt protocol only determine causality within one feed, the previous pointers are needed to cross-reference the feeds of other users who also perform operations on the board, in order to obtain global causality across all feeds.

The previous pointer contains the cryptographic hash values of the last received operation of each participating feed. If this operation is a board creation operation the previous pointer is assigned to NULL.
These pointers then act as a logical clock: it describes that the operation was performed after the one referenced by the pointer, i.e. $a \leftarrow b \Rightarrow time(a) < time(b)$. The pointers form a directed acyclic graph in which the vertices represent the individual operations of the same board and the directed edges represent a "Happened-Before" relation in opposite direction (Figure 3.5). This graph, also called "tangles" [13], determines a global partial ordering of board operations.
Since the pointers would grow linearly with the number of participants, an optimization was implemented based on the transitivity of causal connections. In case of the related events $a \leftarrow b$ and $b \leftarrow c$, the relation $a \leftarrow c$ is given by transitivity and must not be stored additionally.

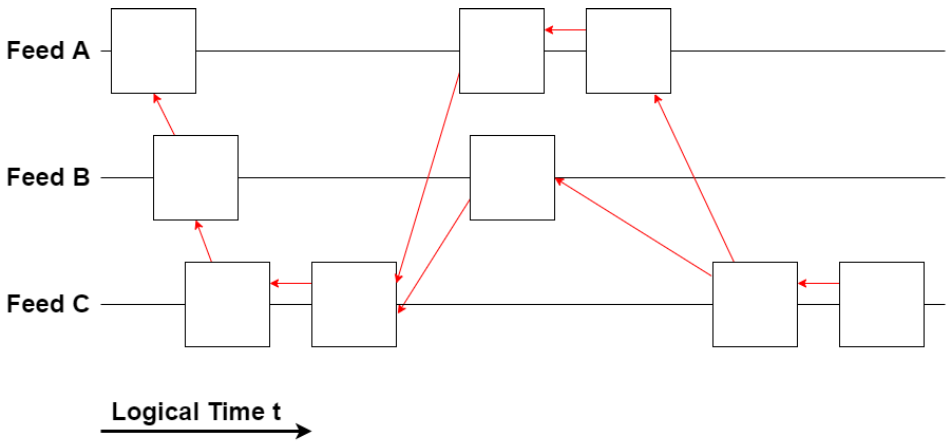


Figure 3.5: Directed acyclic graph: each box represents a board operation.

The order in which the operations are executed is important for the consistency between replicas, since not all operations are commutative. For example, if you color a card red and then blue instead of blue and then red, you will get a different color for the card.
To be sure that each replica performs the operation in the same order, one could use the timestamp sent by the Secure Scuttlebutt protocol. However, this has some disadvantages: On the one hand, the system clocks are inaccurate, which could lead to a wrong order, and on the other hand, this could be exploited by deliberately setting the system time in the future, so that one's own operations are always executed last and are thus preferred.
To solve these issues, each replica should order the operations based on a logical clock. In order to achieve this, all replicas need to perform a linearization of the directed acyclic graph given by the previous pointers to generate the same Kanban board.
In the first versions of this prototype, Kahn's algorithm was used to perform topological sorting. Since the graph represents only a partial sequence and not a total order, some operations are executed concurrently and thus cannot be uniquely linearized. This leads to several possible linear sequences, which in turn cause inconsistencies. Therefore, a global tie breaking rule is required to be executed in each replica in order to obtain only one ex-

act linearization: since the cryptographic hash value of a message is ”pseudorandom” and cannot be specifically influenced by the user, a lexical comparison is performed between the hash values of the concurrent operations.
Even though this first version of the Kanban board worked, it had significant drawbacks: every time a new operation was executed on the board, the sorting algorithm had to sort a new linearization from scratch. Furthermore, the runtime complexity is linear to the number of operations and previous pointers, i.e., Kanban boards on which many operations have already been executed would perform poorly and negatively impact the user experience.
Therefore, in the final version, Kahn’s algorithm was replaced by the ScuttleSort algorithm. It stores the rank of each operation from previous runs, such that a new operation does not have to sort the entire graph, but only the affected part. This leads to better performance than classical topological sorting algorithms.
Due to the eventual consistency of the data structure achieved in this way, every user who has received all updates after some time obtains the same state of the Kanban board.

The logical part of the Kanban application is responsible for processing the data in such a way that it can be visualized by the user interface. For this purpose, a complete Kanban board must be generated from all the individual performed operations.
The messages received by the backend are forwarded to the Kanban application and stored locally in a dictionary. If the new message is a Kanban board operation, it is inserted into a linear timeline using the ScuttleSort algorithm. The linear ordering represents a sorted list that contains the hash values of each operation. The first entry in this list is the operation that must be executed first.
To display a Kanban board, a snapshot of a board is created from all available operations. The operations are executed one after the other in the previously defined order. When the operations are executed, auxiliary data structures are generated that describe the state of the Kanban board and are used by the graphical user interface.
Since the order of operations affects the resulting board, they must be re-executed each time a new operation is inserted into the linear ordering.
To improve performance, some optimizations have been made: if the new operation is appended to the end of the sorted list it is applied directly to the existing snapshot without the need to recalculate a new snapshot from scratch. This is possible because the operation at the last position of the list cannot be overwritten by previous operations. The same optimization can be used for operations where the order does not matter, such as the create and remove operation.

3.4 User Interface

The user interface is inspired by the Kanban board application Trello [6], one of the market leaders in the digital sector. It is particularly designed for mobile devices, such as smartphones and tablets.
The auxiliary data structure contains all the information about the Kanban board: the current title, columns and cards.
For the cards, all attributes such as title, description, color, assignees and comments are stored, as well as the column to which this card is assigned. Local information is also stored, including the personally selected positions of the cards and columns.
During the development of the user interface, special emphasis was placed on performance. It is not necessary to re-render the entire board with all its contents when a new operation has been performed on the board, but only the affected elements. If only one operation is executed at the end of a snapshot, it is immediately applied by the user interface. When a new snapshot needs to be created, it is compared to the previous snapshot to identify the relevant changes to the UI. Only the modified elements are then updated.
Furthermore, the user interface is optimized for mobile devices. Despite the small screen area of smartphones, the Kanban board can be displayed clearly and is intuitive to use.

4 Conclusion

This thesis shows the feasibility of a decentralized Kanban board based on the Secure Scuttlebutt protocol and Conflict-free Replicated Data Types. An operation on the board is stored as a single message and contains a pointer to the previous received operation. By using these cross-referencing pointers, causality can be determined between operations across all users' feeds. Topological sorting with ScuttleSort puts the operations in a linear order. This ensures that the displayed Kanban board is consistent across all replicas. Due to the chosen data structure, no additional coordination is required and no central authority is needed to guarantee consistency between replicas.
This prototype shows that collaboration-based applications, such as the Kanban board, can also be implemented in a decentralized manner using Secure Scuttlebutt. This does not mean that functionality or user-friendliness has to be sacrificed. On the contrary, the prototype supports all essential functionalities, as well as some additional features such as the comment or drag and drop function of the cards. The decentralized structure allows the Kanban board application to function independently of an Internet connection. Users can work offline on the board and later exchange the modifications with other participants without the need of a central server.

Tim Berners-Lee's vision of a decentralized web is not impossible to achieve. He founded the World Wide Web Foundation, an organization that engages in social projects and tries to "re-decentralize" the web. In his blog post to the 30$^{\text{th}}$ anniversary of the World Wide Web, he published following comment:

> *The web is for everyone and collectively we hold the power to change it. It won't be easy. But if we dream a little and work a lot, we can get the web we want.*[11]

With regard to the problems caused by the current, mainly centralized World Wide Web – as mentioned in the introduction of this thesis – it is increasingly important for each user to express his own opinions without being censored, to be independent of third parties and to keep control over his own data.

The presented prototype of a digital decentralized Kanban board shows, as Tim Berners-Lee writes in his comment, that we can find and realize approaches to change or improve the web step by step, namely to “re-create” a decentralized structure of the Internet.

4.1 Future Work

An outlook on further possible improvements and extensions of the Kanban board is given below.

4.1.1 Additional Features

One characteristic of Kanban boards is that they can be used for many different purposes. A personal Kanban is used to organize private tasks, in the corporate sector Kanban boards, for example, support the management of large software projects.
The application must therefore offer many functionalities so that users can adapt their board to their particular needs. The prototype already offers many features, but additional functions allow the user to customize the board even better to their preferences. For example, a tag system could be implemented to assign cards to different tags and group them within a column. In addition, a deadline-function could be added to indicate when the task represented by the card must be completed.
There is also the possibility to implement different views of the Kanban board into the application, such as the chronologically sorted representation of the cards on a timeline.

4.1.2 Dynamic Groups

Currently, the prototype does not support dynamic groups. That means that other users can only be added at the beginning, when the board is created. Subsequently adding new members or removing users is not possible. However, the members of a Kanban board are not always known at the time of creation and can change constantly.
To solve this problem, an invitation system is needed that enables members to subsequently invite new people to join the board. A possible implementation of such an invitation system is shown in the project ”private-group-spec” [2]. Each board contains a symmetric group key that encrypts the board operations. To invite a new user, the symmetric key for the respective board is sent to him. In this way, he has access to the board and can collaborate with the other members.

4.1.3 Optimizations

During the implementation process, some optimizations were made to the logic and the user interface. However, the prototype still has further potential for optimizations.
For example, the way snapshots of the Kanban board are created is still inefficient. If an operation is not attached at the end of the linear ordering, the entire sorted list of operations is run to create a new snapshot for the user interface. Creating another auxiliary data structure, for example, would help to optimize this process. It stores for each element of the

panel the index of the last operation which modified that element. When a new operation is added to the sorted list via ScuttleSort, the indices can be compared. If the index of the new operation is smaller than the stored value, the latest operation does not have to be executed because later it would be overwritten anyway. If the index is larger, the operation can be executed directly on the existing snapshot, since it will not be overwritten by any other operation.

As you can see, there are no limits to the possibilities for extending Kanban boards. The thesis' prototype of a digital decentralized Kanban board provides a good working basis for further projects, such as improving the functionalities or adding new features.

Bibliography

[1] DOT Language. URL https://graphviz.org/doc/info/lang.html. Retrieved: 10.07.2022.

[2] Private-Groups Spec. URL https://github.com/ssbc/private-group-spec. Retrieved: 10.07.2022.

[3] Scuttlebutt Protocol Guide. URL https://ssbc.github.io/scuttlebutt-protocol-guide/index.html. Retrieved: 10.07.2022.

[4] Secure Scuttlebutt. URL https://scuttlebutt.nz/. Retrieved: 10.07.2022.

[5] Topological sorting. URL https://hideoushumpbackfreak.com/algorithms/images/top-ordering.png. Retrieved: 10.07.2022.

[6] Trello, . URL https://trello.com/. Retrieved: 10.07.2022.

[7] Tremola, . URL https://github.com/cn-uofbasel/tremola. Retrieved: 10.07.2022.

[8] China zieht Mauer der Internet-Zensur weiter hoch, Jan 2015. URL https://www.sueddeutsche.de/wirtschaft/internet-china-zieht-mauer-der-internet-zensur-weiter-hoch-dpa.urn-newsml-dpa-com-20090101-150126-99-04899.

[9] George Anadiotis. Manyverse and Scuttlebutt: A human-centric technology stack for social applications, Oct 2018. URL https://www.zdnet.com/article/manyverse-and-scuttlebutt-a-human-centric-technology-stack-for-social-applications/. Retrieved: 10.07.2022.

[10] Tim Berners-Lee. *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*, page 99. Harper San Francisco, 1999.

[11] Tim Berners-Lee. 30 years on, what's next #ForTheWeb?, Mar 2019. URL https://webfoundation.org/2019/03/web-birthday-30/. Retrieved: 10.07.2022.

[12] Thomas H. Cormen. *Introduction to algorithms*, pages 612–615. MIT Press, Cambridge, Mass, 3rd ed. edition, 2009.

[13] Aljoscha Meyer Dominic Tarr, Erick Lavoie and Christian Tschudin. Secure Scuttlebutt: An Identity-Centric Protocol for Subjective and Decentralized Applications. In *Proceedings of the 6th ACM conference on information-centric networking*, pages 1–11, 2019.

[14] drstarry. A Gentle Introduction of Collaborative Editing, Dec 2016. URL https://drstarry.github.io/collebertive-editing/. Retrieved: 10.07.2022.

[15] Issie Lapowsky. Cambridge Analytica took 50m facebook users' data—and both companies owe answers, Mar 2018. URL https://www.wired.com/story/cambridge-analytica-50m-facebook-users-data/. Retrieved: 10.07.2022.

[16] Renkun Liu. A low complexity topological sorting algorithm for directed acyclic graph. *International Journal of Machine Learning and Computing*, 4(2), 2014-4.

[17] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free Replicated Data Types. Research Report RR-7687, INRIA, July 2011. URL https://hal.inria.fr/inria-00609399.

[18] Chengzheng Sun. Operational Transformation Frequently Asked Questions and Answers. URL https://www3.ntu.edu.sg/scse/staff/czsun/projects/otfaq/. Retrieved: 10.07.2022.

[19] Christian Tschudin. ScuttleSort: Incremental and Convergent Linearization of tangles, May 2022. URL https://github.com/tschudin/scuttlesort/blob/main/doc/README.pdf. Retrieved: 10.07.2022.

[20] International Telecommunication Union. Statistics: Individuals using the Internet. URL https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx. Retrieved: 10.07.2022.

University of Basel

Faculty of Science

Declaration on Scientific Integrity
(including a Declaration on Plagiarism and Fraud)
Translation from German original

Title of Thesis:

Name Assesor:	Prof. Dr. Christian Tschudin
Name Student:	Jannick Heisch
Matriculation No.:	19-056-720

With my signature I declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Place, Date: Wehr, 17.07.2022 Student: J. Heisch

Will this work be published?

○ No

[✓ Yes. With my signature I confirm that I agree to a publication of the work (print/digital) in the library, on the research database of the University of Basel and/or on the document server of the department. Likewise, I agree to the bibliographic reference in the catalog SLSP (Swiss Library Service Platform). (cross out as applicable)]

Publication as of: 17.07.2022

Place, Date: Wehr, 04.01.2023 Student: J. Heisch

Place, Date: ______ Assessor: ______

Please enclose a completed and signed copy of this declaration in your Bachelor's or Master's thesis .