

**Dokumentation zum Fortgeschrittenenpraktikum**

# **ChronicleDB**

**Arbeitsgruppe Datenbanksysteme**

**Philipps-Universität Marburg**

Davit Davtyan

Lars Happel

Johannes Buder

WS 2021/2022

# Inhaltsverzeichnis

<b>1</b>	<b>Planung</b>	<b>4</b>
1.1	Kurzbeschreibung . . . . .	4
1.2	Entwicklungsprozess . . . . .	4
1.2.1	Zu erstellende Dokumente . . . . .	4
1.2.2	Rollenvergabe . . . . .	4
1.2.3	Eingesetzte Programmiersprachen . . . . .	4
1.2.4	Integrierte Entwicklungsumgebung . . . . .	4
1.2.5	Test-Framework . . . . .	4
1.3	Team . . . . .	4
1.4	Risikomanagement . . . . .	5
1.5	Zeitplan . . . . .	5
1.5.1	Zusammenfassung der Meilensteine . . . . .	5
1.5.2	Erreichte Planungsziele . . . . .	5
<b>2</b>	<b>Anforderungsanalyse</b>	<b>6</b>
2.1	Definition des Zielsystems . . . . .	6
2.2	Funktionale Anforderungen . . . . .	6
2.2.1	User . . . . .	6
2.2.2	Admin . . . . .	8
2.3	Nicht-Funktionale Anforderungen . . . . .	9
2.3.1	Qualität des Systems/Codes . . . . .	9
2.3.2	Benutzbarkeit . . . . .	9
2.3.3	Technische Anforderungen . . . . .	9
2.3.4	Anforderungen an die Dokumentation . . . . .	9
<b>3</b>	<b>Entwurf</b>	<b>10</b>
3.1	Grober Entwurf . . . . .	10
3.2	Technologien . . . . .	10
<b>4</b>	<b>Qualitätssicherung</b>	<b>11</b>
4.1	Testplan . . . . .	11
4.2	Testprotokoll . . . . .	11
4.2.1	Login . . . . .	11
4.2.2	Create Stream . . . . .	12
4.2.3	Operationen auf Streams . . . . .	14
4.2.4	Jobs und Log Files . . . . .	16
4.2.5	User Management . . . . .	17
4.2.6	Dark Mode . . . . .	19
4.2.7	Routing . . . . .	19
<b>5</b>	<b>Abschlussbericht</b>	<b>21</b>
5.1	Zusammenfassung . . . . .	21
5.2	Benutzerdokumentation . . . . .	21
5.2.1	Benötigte Abhängigkeiten . . . . .	21
5.2.2	Installationsanleitung . . . . .	24
5.3	Entwicklerdokumentation . . . . .	25
5.3.1	Kurzbeschreibung der Komponenten . . . . .	25
5.3.2	Erweiterungsszenarien . . . . .	26
5.3.3	Zu beachtende Codekonventionen . . . . .	26
5.3.4	Bekannte Fehler / fehlende Eigenschaften . . . . .	26

## *Inhaltsverzeichnis*

5.4	Erfahrungsbericht . . . . .	26
5.5	Protokolle und Entwicklungsverlauf . . . . .	27

# 1 Planung

## 1.1 Kurzbeschreibung

ChronicleDB ist ein Prototyp eines Zeitreihen-Datenbanksystems, entwickelt in der AG Datenbanksysteme, welcher einen Datenstrom von Ereignissen (Ereignisstrom) verarbeitet und persistiert. Ein Ereignis besteht aus einem Zeitstempel (timestamp) und einem Anhang (payload). Ein Beispiel für einen Ereignisstrom bietet z.B. ein Temperatursensor, welcher in bestimmten zeitlichen Abständen seine Messung (Ereignis) an das System schickt. Dabei enthält timestamp den Zeitpunkt der Messung und payload die gemessene Temperatur.

Das innerhalb eines Semesters zu erstellende Admin Panel soll dabei stets erweiterbar sein, damit es auch in Zukunft noch mit gemeinsam mit ChronicleDB einsetzbar ist.

## 1.2 Entwicklungsprozess

### 1.2.1 Zu erstellende Dokumente

Während des Entwicklungszeitraum sollen für alle Team-Treffen, mindestens einmal wöchentlich Protokolle angefertigt werden, in denen die besprochenen Themen und ermittelten Ziele festgehalten werden. Ebenso werden regelmäßig Screenshots erstellt, um den Fortschritt und den aktuellen Stand zu verdeutlichen.

### 1.2.2 Rollenvergabe

Die Protokollantenrolle in Treffen, sowie die Aufgabenverteilung bei der Softwareentwicklung werden flexibel festgelegt. Da das Projekt in erster Linie zur Übung dienen soll, ist es sinnvoll, dass jedes Teammitglied möglichst jede Aufgabe einmal übernimmt.

### 1.2.3 Eingesetzte Programmiersprachen

Das Backend, welches die Nutzerverwaltung und die Kommunikation mit Chronicle übernimmt, läuft mit Python. Das Frontend wird mit TypeScript, HTML und CSS in Angular umgesetzt.

### 1.2.4 Integrierte Entwicklungsumgebung

Visual Studio Code zusammen mit Node JS und dem Angular Framework.

### 1.2.5 Test-Framework

Als Test-Framework für die Unit Tests verwenden wir Jasmine in Verbindung mit Karma.

## 1.3 Team

Unser Team besteht aus drei Mitgliedern. Lars Happel, dem das Repository gehört, Davit Davtyan und Johannes Buder. Die Aufgabenbereiche haben wir auf alle gleichermaßen aufgeteilt, um jedem die Möglichkeit zu bieten, aus allen Bereichen zu lernen.

## 1.4 Risikomanagement

Ein nicht zu vernachlässigendes Risiko ist, dass wir mit Webentwicklung insbesondere aber mit der Programmiersprache TypeScript/JavaScript und dem Framework Angular keine bis sehr wenig Erfahrung haben. Ebenso ist ChronicleDB ein für uns zuvor unbekanntes Projekt, in das wir uns zunächst einarbeiten mussten. Da die Dokumentation des eben genannten Projekts Lücken aufweist, hatten wir zudem Schwierigkeiten, uns dort einzuarbeiten.

Da wir bereits in der Vergangenheit viel mit Java programmiert haben, haben wir uns für TypeScript entschieden, welches Java mehr ähnelt als JavaScript. Dadurch wollen wir das Risiko, dass wir nicht mit der Programmiersprache umgehen können, minimieren. Zudem haben wir uns einen Online-Angular-Kurs gekauft, der uns grundlegende aber auch tiefer gehende Themen von Angular und TypeScript näher bringt, sodass wir dieses Risiko primär behandeln können. Mit diesem Kurs können wir mit passenden Beispielen unser bisheriges Wissen erweitern. Da er ziemlich ausführlich ist, schätzen wir das Risiko, dass wir mit der Sprache und dem Framework nicht umgehen können, auf sehr gering ein. Falls doch noch Probleme auftreten, gibt es auf StackOverflow etliche Artikel über eben genannte Themen, die uns weiterhelfen.

## 1.5 Zeitplan

Durch die Leitlinie des Praktikums ist eine grobe Meilensteinplanung vorgegeben. Wir planen, uns in diesem Projekt an diesem Vorschlag zu orientieren. Da wir jedoch bisher keine Erfahrung mit der Sprache, der Library und dem Programm haben, planen wir primär für den ersten (aber auch noch für den zweiten) Meilenstein mehr Zeit ein, um uns das nötige Wissen aneignen und ausprobieren können.

Das Projekt wird dabei durchgehend für den Betreuer verfügbar sein.

### 1.5.1 Zusammenfassung der Meilensteine

#### 1. Meilenstein

- REST API / JSON verstehen
- GUI mit Button für create stream, wie pgAdmin Interface oder eigenes Interface

#### 2. Meilenstein

- Buttons für insert event, usw. alle anderen (siehe REST Spezifikation auf GitHub/dbs)

#### 3. Meilenstein

- User-Verwaltung: Admin und User Rechte - Java via Postgres oder SpringBoot oder per Hand | Rust via Postgres oder per Hand
- Authentifizierung und ggf. Anpassung der GUI, sodass nur das angezeigt wird, was auch möglich ist

#### 4. Meilenstein

- Jobs einführen, sodass eine Anfrage wiederholt automatisch ausgeführt wird
- Status visualisieren
- Tests und Dokumentationen

#### 5. Abgabe

- Software und Bericht
- Vorstellung

### 1.5.2 Erreichte Planungsziele

Es wurden alle Bestandteile der Meilensteine erreicht.

## 2 Anforderungsanalyse

### 2.1 Definition des Zielsystems

Das Zielsystem muss primär die Funktionalitäten von Angular und somit die von HTML und JavaScript unterstützen. Falls ChronicleDB lokal ausgeführt werden soll, müssen ebenso die Anforderungen an das Projekt erfüllt sein.

### 2.2 Funktionale Anforderungen

#### 2.2.1 User

##### **Stream auswählen (notwendig)**

**Vorbedingung** Der User ist in die Weboberfläche eingeloggt. Es ist noch kein Stream zur Bearbeitung ausgewählt.

**Ablauf** Der User wählt aus einer Liste der angebundenen und ihm verfügbaren Streams einen zur Bearbeitung aus.

##### **Daten einpflegen (notwendig)**

**Vorbedingung** Der User ist eingeloggt und ein zur Bearbeitung freigeschalteter Stream ist ausgewählt.

**Ablauf** Der User wählt die „Insert Data“ Funktion. Ein Dialog öffnet sich zur Auswahl einer Quelldatei. Der User gibt den Pfad zur Quelldatei an und die darin enthaltenen Daten werden der Datenbank hinzugefügt.

##### **Daten manuell einpflegen (optional)**

**Vorbedingung** Der User ist eingeloggt und ein zur Bearbeitung freigeschalteter Stream ist ausgewählt.

**Ablauf** Der User wählt die „Insert Data“ Funktion. Ein Dialog öffnet sich und der User kann die manuelle Eingabe auswählen. Der User gibt eine Eventdefinition ein und bestätigt diese. Daraufhin wird dieses Event und die darin enthaltenen Daten werden der Datenbank hinzugefügt.

##### **Backup von Daten erstellen (optional)**

**Vorbedingung** Der User ist eingeloggt und ein zur Bearbeitung freigeschalteter Stream ist ausgewählt.

**Ablauf** Der User wählt die Backup Option. Eine Backup-Datei mit den Daten des aktuellen Streams wird als Download bereit gestellt.

##### **Daten in einem bestimmten Zeitraum einsehen (notwendig)**

**Vorbedingung** Der User ist eingeloggt und ein zum Lesen freigeschalteter Stream ist ausgewählt.

**Ablauf** Der User wählt die „Time Travel“ Funktion. Ein Dialog fragt den Start- und Endzeitpunkt der einzusehenden Daten ab. Nach korrekter Eingabe werden die Daten aus diesem Zeitraum angezeigt.

##### **Daten der rechten Flanke einsehen (notwendig)**

**Vorbedingung** Der User ist eingeloggt und ein zum Lesen freigeschalteter Stream ist ausgewählt.

**Ablauf** Der User wählt die „Show Right Flank“ Funktion. Ein Dialog öffnet sich und die entsprechenden Daten werden angezeigt.

### **Min / Max / Tree Height eines Streams einsehen (notwendig)**

**Vorbedingung** Der User ist eingeloggt und ein zum Lesen freigeschalteter Stream ist ausgewählt.

**Ablauf** Der User wählt eine der oben genannten Funktionen. Ein Dialog öffnet sich und die entsprechenden Daten werden angezeigt.

### **Stream Konfiguration / Info einsehen (notwendig)**

**Vorbedingung** Der User ist eingeloggt und ein zum Lesen freigeschalteter Stream ist ausgewählt.

**Ablauf** Der User wählt die „Stream Info“ Funktion. Ein Dialog öffnet sich und die Konfiguration des Streams wird angezeigt.

### **Stream erstellen (notwendig)**

**Vorbedingung** Ein User mit der Berechtigung zum Erstellen eines Streams ist in die Weboberfläche eingeloggt. Er befindet sich auf der Seite zum Erstellen von Streams.

**Ablauf** Der User definiert die URL des Servers und spezifiziert, wie ein Event des Streams aussehen soll. Dazu wählt er zunächst aus, ob das Event nur aus einem Datenfeld (Single) bestehen soll, oder ob es aus mehreren (Compound oder VarCompound) bestehen soll. Entsprechend der Auswahl erstellt der User die gewünschte Anzahl an Komponenten und spezifiziert für jede einzelne, ob es eine Liste oder ein einzelner Wert sein soll, den Datentyp (Integer, String, etc.) und einen Subtyp (8bit Integer, 16bit Integer, 32bit Integer, etc.). Falls eine Liste oder ein String ausgewählt wurde, soll zudem die Standardgröße der Liste bzw. die Standardgröße des Strings angegeben werden.

Optional kann der User noch genauere Konfigurationen des Streams vornehmen, indem er die Standardeinstellungen zum Debugging, I/O, zu der Blockgröße, zum Cache und zum Kompressionsverfahren ändert. Sobald er alles eingestellt hat, kann er den Create-Stream-Button verwenden, um diesen zu erstellen. Nun wird dieser Stream in der Liste der Streams angezeigt und der User kann diesen auswählen.

### **Job erstellen (notwendig)**

**Vorbedingung** Ein User mit Leserechten ist in die Weboberfläche eingeloggt.

**Ablauf** Der User wählt einen Stream und eine Leseoperation aus. Nun betätigt er die Funktion „Create Job“. Es öffnet sich ein Dialog, in dem der User ein Startdatum, eine Starturzeit, einen Wiederholungszeitraum und eine Optionale Info eingeben kann. Wenn er dies bestätigt, wird ein Job für den Nutzer für diesen Stream erstellt.

Dieser wird nun in der Übersicht über Jobs angezeigt.

### **Job löschen (notwendig)**

**Vorbedingung** Ein User mit Leserechten ist in die Weboberfläche eingeloggt und befindet sich bei der Übersicht über die existierenden Jobs.

**Ablauf** Der User wählt einen Job aus und betätigt die Löschfunktion. Es öffnet sich ein Bestätigungsdialog. Wenn dieser bestätigt wurde, wird der Job gelöscht und aus der Übersicht entfernt.

### **Job manuell ausführen (optional)**

**Vorbedingung** Ein User mit Leserechten ist in die Weboberfläche eingeloggt und befindet sich bei der Übersicht über die existierenden Jobs.

**Ablauf** Der User wählt einen Job aus und betätigt die Ausführungsfunktion. Der Job wird nun ausgeführt und das Ergebnis angezeigt.

### **Jobergebnisse anzeigen (notwendig)**

**Vorbedingung** Ein User mit Leserechten ist in die Weboberfläche eingeloggt und befindet sich bei der Übersicht über die existierenden Jobs.

**Ablauf** Der User navigiert auf die Seite mit den Ergebnissen der Jobs. Dort sieht er alle Ergebnisse gemäß seiner definierten Jobs.

### **Jobergebnisse löschen (notwendig)**

**Vorbedingung** Ein User mit Leserechten ist in die Weboberfläche eingeloggt und befindet sich bei der Übersicht der Jobresults.

**Ablauf** Der User betätigt die Löschen-Funktion. Nach einer Bestätigung werden alle bisherigen Ergebnisse gelöscht.

### **Ein Jobergebniss kopieren (optional)**

**Vorbedingung** Ein User mit Leserechten ist in die Weboberfläche eingeloggt und befindet sich bei der Übersicht der Jobresults.

**Ablauf** Der User wählt ein Ergebnis aus und wählt die „Kopieren“ Funktion. Das Ergebnis ist nun in die Zwischenablage kopiert.

### **Ein Jobergebniss löschen (optional)**

**Vorbedingung** Ein User mit Leserechten ist in die Weboberfläche eingeloggt und befindet sich bei der Übersicht der Jobresults.

**Ablauf** Der User wählt ein Ergebnis aus und wählt die „Löschen“ Funktion. Das Ergebnis ist nun gelöscht und wird nicht mehr in der Liste angezeigt.

## **2.2.2 Admin**

### **User erstellen (notwendig)**

**Vorbedingung** Ein Admin ist in die Weboberfläche eingeloggt. Er befindet sich auf der Seite zur Verwaltung der Nutzer.

**Ablauf** Der Admin wählt die Funktion „Add User“ und gibt dessen Nutzernamen und Passwort ein. Zudem weist er dem neuen User entweder allgemeine Lese- und / oder Schreibrechte zu, oder nur für bestimmte Streams. Ebenso wird ausgewählt, ob der neue User über Adminrechte verfügt und ob dieser die Java-Version benutzt.

### **User löschen (notwendig)**

**Vorbedingung** Ein Admin ist in die Weboberfläche eingeloggt. Er befindet sich auf der Seite zur Verwaltung der Nutzer.

**Ablauf** Der Admin wählt einen Nutzer aus und betätigt die Funktion „Delete User“. Es öffnet sich ein Bestätigungsdialog. Wenn dieser bestätigt wird, wird dieser Nutzer aus der Datenbank gelöscht.

### **User bearbeiten (optional)**

**Vorbedingung** Ein Admin ist in die Weboberfläche eingeloggt. Er befindet sich auf der Seite zur Verwaltung der Nutzer.

**Ablauf** Der Admin wählt einen Nutzer aus und betätigt die Funktion „Edit User“. Es öffnet sich ein Dialog, in dem der Admin die Daten des Users ändern kann. Wenn dieser bestätigt wird, wird dieser Nutzer in der Datenbank aktualisiert.

### **Streams herunterfahren (notwendig)**

**Vorbedingung** Ein Admin ist in die Weboberfläche eingeloggt und ein Stream ist ausgewählt.

**Ablauf** Der Admin wählt die „Shutdown Stream“ Funktion. Ein Bestätigungsdialog wird angezeigt. Wenn dieser bestätigt wird, wird der Stream in der Datenbank heruntergefahren.



### **Stream wiederherstellen (notwendig)**

**Vorbedingung** Ein Admin ist in die Weboberfläche eingeloggt und ein heruntergefahrener Stream ist ausgewählt.

**Ablauf** Der Admin wählt die „Recover Stream“ Funktion und der Stream wird in der Datenbank wiederhergestellt.

## **2.3 Nicht-Funktionale Anforderungen**

### **2.3.1 Qualität des Systems/Codes**

Es soll eine möglichst hohe Code-Abdeckung durch Systemtests erreicht werden. Es werden die gängigen Code-Konventionen für TypeScript eingehalten.

### **2.3.2 Benutzbarkeit**

Das System soll für einen User, der damit noch nicht vertraut ist einfach verwendbar sein. Dies wird erreicht durch die Verwendung von Tooltips und gängigen Layouts wie z.B. Navbar. Der User wird vor der Eingabe ungültiger Daten abgehalten und ggfs. durch Nachrichten auf Eingabefehler hingewiesen.

Die Zielgruppe sind User, welche auf die in ChronicleDB zur Verfügung gestellten Daten zugreifen und diese ggfs. bearbeiten wollen.

### **2.3.3 Technische Anforderungen**

Die Weboberfläche muss kompatibel sein zum ChronicleDB System der AG Datenbanken. Sie muss zudem auf gängigen Browsern (Chrome, Firefox, Edge) lauffähig sein.

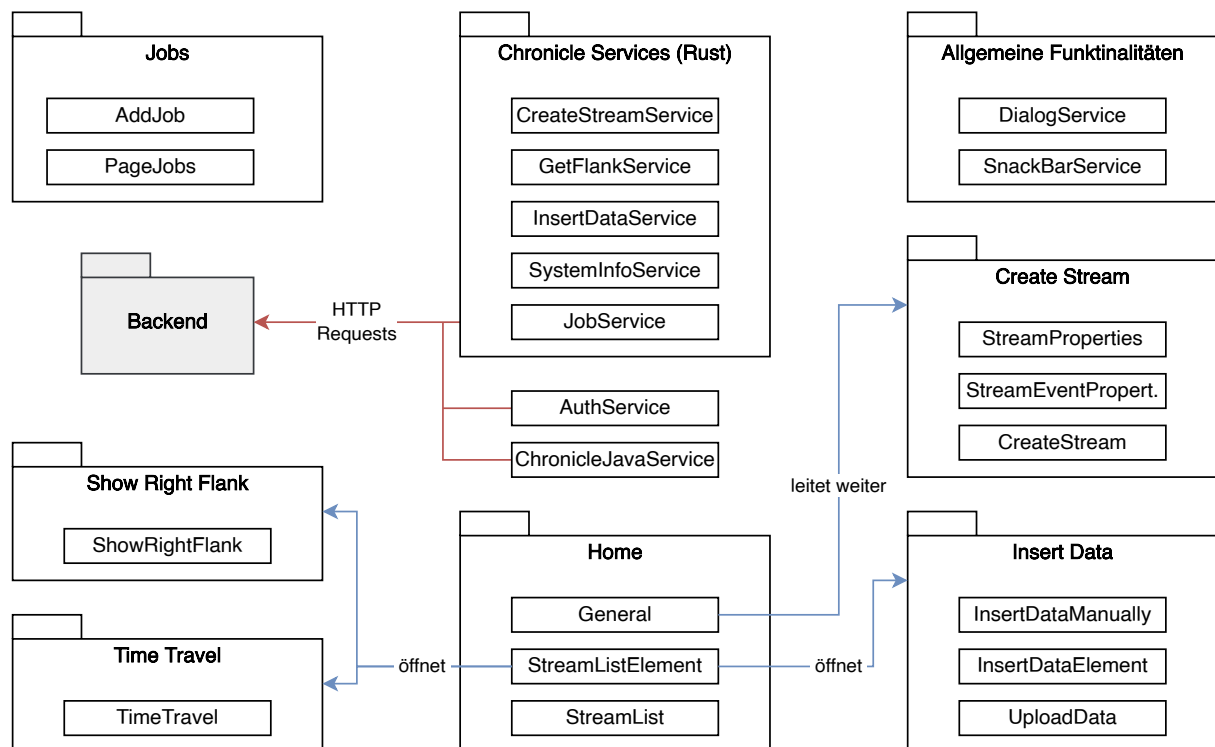
### **2.3.4 Anforderungen an die Dokumentation**

Die Anforderungen werden durch die Leitlinie des Fortgeschrittenenpraktikums vorgegeben.

## 3 Entwurf

### 3.1 Grober Entwurf

Um die Webseite umzusetzen, spalten wir die Funktionalitäten in einzelne Services (Create-Stream-Service, Auth-Service, Job-Service, ...) auf. In diesen soll mittels HTTP-Anfragen die Kommunikation mit dem Backend ablaufen. Ebenso werden unabhängige Elemente der Darstellung in einzelne Komponenten verpackt (Stream-List, Insert-Data, Upload-Data, ...), um die Wiederverwendbarkeit zu gewährleisten. Dazu dienen Services und Dialoge als Kommunikation zwischen den Komponenten.



### 3.2 Technologien

Um das Admin Panel zu erstellen, haben wir uns für das Framework Angular (Version 12.2.11) in Verbindung mit der Programmiersprache TypeScript (Version 4.2.5) und der Bibliothek Angular Material (Version 12.2.12) entschieden. Diese Material Design Bibliothek bietet uns einige Bausteine und optisch ansprechende Elemente, die wir bei unserer Gestaltung nutzen können.

Durch den Videokurs zu Angular haben wir auch manche Ideen zur Gestaltung des Codes, aber auch zu nützlichen Bibliotheken, die wir benutzen können.

Damit wir unseren bisherigen Fortschritt ausführen können, verwenden wir NodeJS. Ebenso verwenden wir Git um eine gute Versionsverwaltung zu garantieren.

Für die Tests verwenden wir Karma.

# 4 Qualitätssicherung

## 4.1 Testplan

Da wir im Frontend keine Berechnungen durchführen, sondern nur anzeigen und weiterreichen, führen wir fast ausschließlich Systemtests durch.

## 4.2 Testprotokoll

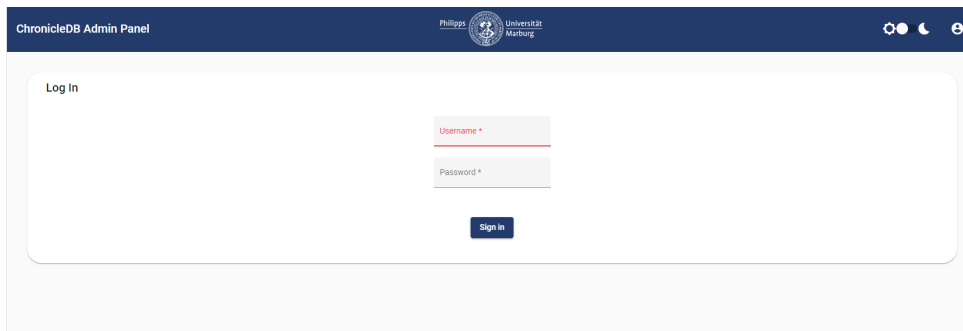
### 4.2.1 Login

#### Erfolgreicher Login

**Vorbedingung:** Der *Nutzer* öffnet die Website und befindet sich auf der Login-Seite. Der *Nutzer* ist bereits im Backend registriert.

**Ablauf:** Der *Nutzer* gibt seinen korrekten Nutzernamen und Passwort ein und klickt auf „Sign in“.

**Erwartetes & Tatsächliches Verhalten:** Der *Nutzer* wird eingeloggt und weitergeleitet.



#### Fehlgeschlagener Login

**Vorbedingung:** Der Nutzer öffnet die Website und befindet sich auf der Login-Seite. Er wird entweder falsche Daten eingeben oder nicht im Backend registriert sein.

**Ablauf:** Der *Nutzer* gibt seinen Nutzernamen und Passwort ein und klickt auf „Sign in“.

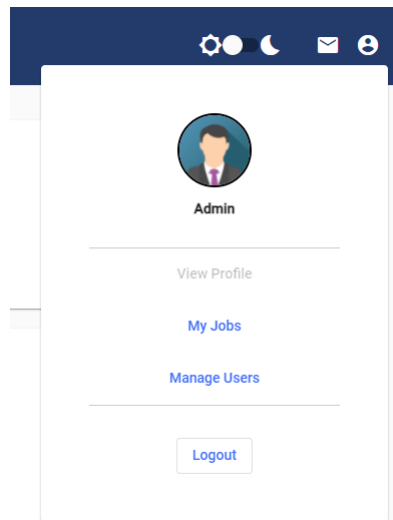
**Erwartetes & Tatsächliches Verhalten:** Der *Nutzer* wird nicht eingeloggt und es passiert nichts.

#### Eingeloggter User wird korrekt angezeigt

**Vorbedingung:** Der *Nutzer* hat sich bereits eingeloggt und befindet sich auf der Home-Seite der Website.

**Ablauf:** Der *Nutzer* klickt auf den Account-Button.

**Erwartetes & Tatsächliches Verhalten:** Der korrekte Nutzername wird angezeigt.



### Logout

**Vorbedingung:** Der *Nutzer* ist eingeloggt.

**Ablauf:** Der *Nutzer* klickt auf den Account-Button und dann auf „Logout“.

**Erwartetes & Tatsächliches Verhalten:** Der *Nutzer* wird ausgeloggt und zum Login Screen geleitet.

### Login nötig

**Vorbedingung:** Der *Nutzer* ist nicht eingeloggt.

**Ablauf:** Der *Nutzer* gibt die URL „/create\_stream“ ein.

**Erwartetes & Tatsächliches Verhalten:** Der *Nutzer* wird auf die Login-Seite weitergeleitet. Sobald er sich eingeloggt hat, wird er zurück auf die „create-stream“ Seite geleitet.

## 4.2.2 Create Stream

### Create a New Stream

**Vorbedingung:** Der *Nutzer* hat eine valide Server URL eingeben und versucht nun einen *Stream* zu erstellen.

**Ablauf:** Nachdem man auf den „Create a New Stream“ Button geklickt hat.

**Erwartetes & Tatsächliches Verhalten:** *Nutzer* wird weitergeleitet auf die „/create\_stream“ Seite und kann nun seinen *Stream* vor dem erstellen konfigurieren .

### Customize Event

**Vorbedingung:** Der *Nutzer* hat eine valide Server URL eingeben und auf „Create a New Stream“ geklickt und versucht nun Events zu konfigurieren .

**Ablauf:** Der *Nutzer* konfiguriert seine Events indem er auf den „+“-Button klickt und so welche hinzufügt. Zuvor wählt er aus ob es sich um Single oder VarCompound oder Compound handelt. Wenn er versucht weniger als 1 Event zu formulieren erhält er eine Fehlermeldung via Snackbar.

**Erwartetes & Tatsächliches Verhalten:** *Nutzer* wählt aus den Drop-Down Listen „Single or List“, „Type“ und „Subtype“ seine Konfiguration und fügt bei Bedarf mehrere Events hinzu. Weniger als 1 Event zu haben wird nicht gestattet.

## Customize Event

## Stream Properties

**Vorbedingung:** Der *Nutzer* hat StreamPoperties Drop-Down geklickt, und ist bereit diese zu konfigurieren.

**Ablauf:** Der *Nutzer* konfiguriert seinen *Stream* über multiple Drop-Downs und Input Felder .

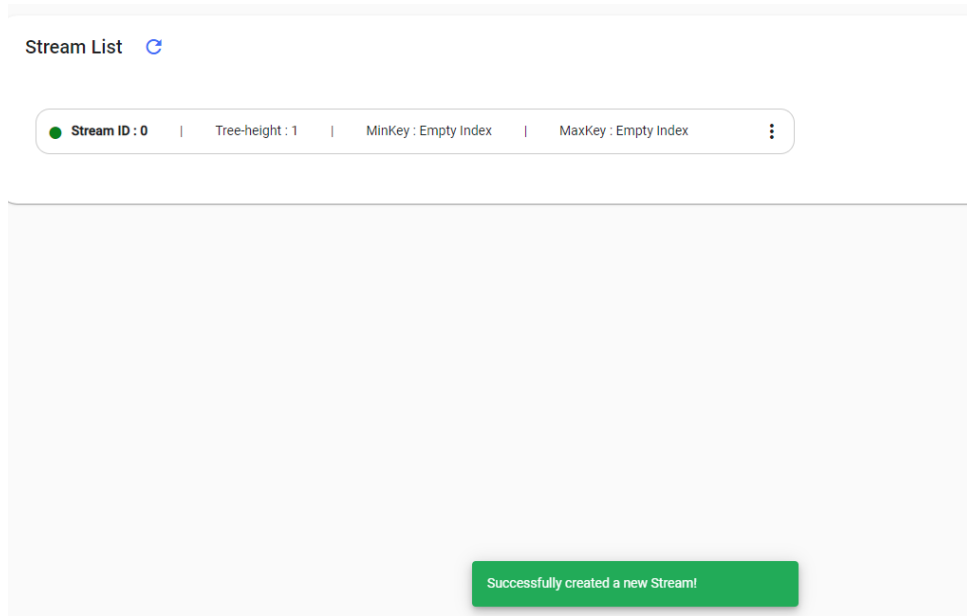
**Erwartetes & Tatsächliches Verhalten:** *Nutzer* gibt valide Eingaben ein und konfiguriert den *Stream*.

## Create a New Stream „auf /create\_stream“

**Vorbedingung:** Der *Nutzer* hat einen validen *Stream* konfiguriert und versucht nun einen *Stream* zu erstellen.

**Ablauf:** Nachdem man auf den „Create a New Stream“ Button geklickt hat wird ein Stream erstellt.

**Erwartetes & Tatsächliches Verhalten:** *Nutzer* wird weitergeleitet auf die „/home“ Seite und kann nun seinen *Stream* in der StreamList sehen. Ein *Stream* wurde erfolgreich erstellt.



### 4.2.3 Operationen auf Streams

#### Daten per Eingabe einfügen

**Vorbedingung** Ein *Stream* wurde bereits erzeugt und ist in der *Stream* Liste angezeigt.

**Ablauf** Der *Nutzer* öffnet mit einem Linksklick auf die 3 vertikalen Punkte das Kontextmenü und wählt dort den obersten Punkt "Insert Data" per Linksklick aus. Es erscheint ein Dialog der die Eingabe von Daten erlaubt. Der User wählt einen Timestamp der höher ist als der bisher höchste Timestamp des *Streams*. Unterhalb des Timestamp befinden sich Felder die das Eingeben der Daten erlauben. Diese Felder entsprechen den bei Erstellung des Streams bestimmten Datentypen. Der User füllt die Felder mit Informationen, welche den Datentypen (bsp. Int, Float, String) entsprechen und klickt "Insert Event".

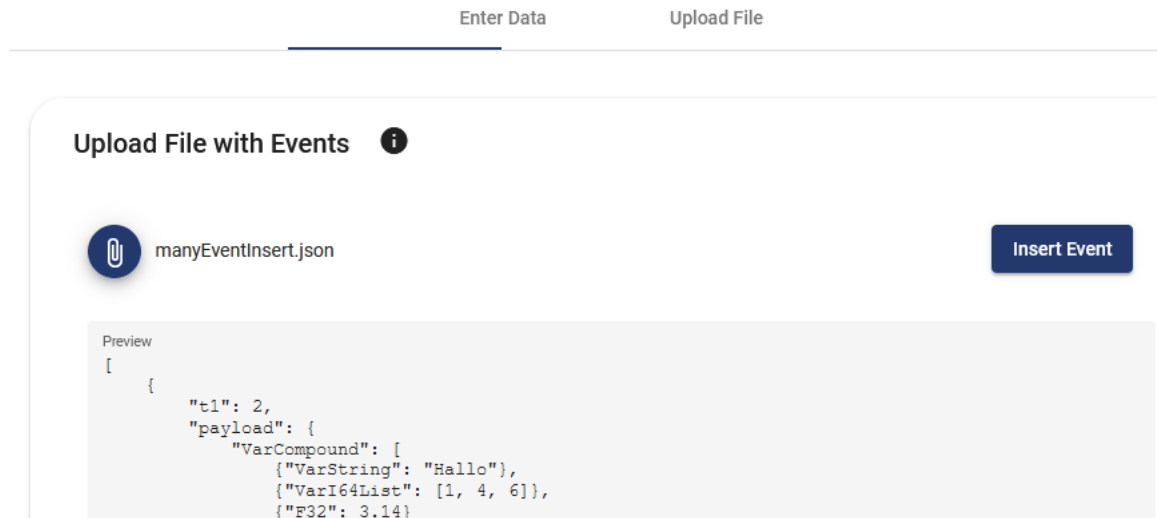
**Erwartetes & Tatsächliches Verhalten** Die Daten werden in den Stream eingefügt und dies wird über einen Dialog bestätigt. Der eingestellte Timestamp wird um 1 erhöht.

#### Daten per File Upload einfügen

**Vorbedingung** Ein *Stream* wurde bereits erzeugt und ist in der *Stream* Liste angezeigt.

**Ablauf** Der *Nutzer* öffnet mit einem Linksklick auf die 3 vertikalen Punkte das Kontextmenü und

wählt dort den obersten Punkt "Insert Data" per Linksklick aus. Es erscheint ein Dialog der die Eingabe von Daten erlaubt. Der *Nutzer* wählt die Schaltfläche "Upload File". Mit Klick auf die Büroklammer öffnet sich ein Dialog zur Auswahl einer Datei. Der *Nutzer* selektiert eine Datei mit Daten, welche dem gewählten Stream entsprechend und bestätigt den Dialog. Es wird eine Vorschau der einzufügenden Daten im Dialog angezeigt. Über den Button "Insert Event" wird der Dialog bestätigt.



**Erwartetes & Tatsächliches Verhalten** Die Daten werden in den Stream eingefügt und dies wird über einen Dialog bestätigt.

### Max/Min Key anzeigen

**Vorbedingung** Ein *Stream* wurde bereits erzeugt und ist in der *Stream* Liste angezeigt.

**Ablauf** Der *Nutzer* öffnet das Kontextmenü des Streams und wählt den zweiten Punkt "Get Keyünd anschließend *Max Key* oder *Min Key* gewählt.

**Erwartetes & Tatsächliches Verhalten** Es wird ein Dialog angezeigt der den jeweiligen Wert anzeigt.

### Tree Height anzeigen

**Vorbedingung** Ein *Stream* wurde bereits erzeugt und ist in der *Stream* Liste angezeigt.

**Ablauf** Der *Nutzer* öffnet das Kontextmenü des Streams und wählt den dritten Punkt "Get Tree Height".

**Erwartetes & Tatsächliches Verhalten** Es wird ein Dialog angezeigt der den Wert anzeigt.

### Stream Info anzeigen

**Vorbedingung** Ein *Stream* wurde bereits erzeugt und ist in der *Stream* Liste angezeigt.

**Ablauf** Der *Nutzer* öffnet das Kontextmenü des Streams und wählt den Punkt "SStream Info".

**Erwartetes & Tatsächliches Verhalten** Es wird ein Dialog angezeigt der den Wert anzeigt.

### Right Flank Info anzeigen

**Vorbedingung** Ein *Stream* wurde bereits erzeugt und ist in der *Stream* Liste angezeigt.

**Ablauf** Der *Nutzer* öffnet das Kontextmenü des Streams und wählt den Punkt "SShow Right Flank".

**Erwartetes & Tatsächliches Verhalten** Es wird ein Dialog angezeigt mit einem Textfenster, welches die Right Flank Info der Datenbank als formatierter String anzeigt.

### Query Time Travel

**Vorbedingung** Ein *Stream* wurde bereits erzeugt und ist in der *Stream* Liste angezeigt.

**Ablauf** Der *Nutzer* öffnet das Kontextmenü des Streams und wählt den Punkt "Query Time Travel". Es wird ein Dialog angezeigt, der dem *Nutzer* erlaubt die Anfrage zu spezifizieren. Der *Nutzer* wählt einen Start- und Endtimestamp sowie die Option Exclusive oder Inclusive und betätigt den "Go!"Button.

**Time Travel**

Exclusive
Inclusive

From  
0

To  
50

Go!
Close

 Create A Job

**Erwartetes & Tatsächliches Verhalten** Im unteren Bereich des Dialogs wird eine zwischen String-View und Table-View umschaltbare Ansicht der gewünschten Timestamps und deren Daten angezeigt.

### Shutdown Stream

**Vorbedingung** Ein *Stream* wurde bereits erzeugt und ist in der *Stream* Liste angezeigt.

**Ablauf** Der *Nutzer* öffnet das Kontextmenü des Streams und wählt den Punkt "SShutdown Stream".

**Erwartetes & Tatsächliches Verhalten** Der Stream wird geschlossen und die Farbe des Status Symbols ändert sich von grün zu rot.

### Recover Stream

**Vorbedingung** Ein *Stream* wurde bereits erzeugt und ist in der *Stream* Liste als offline angezeigt.

**Ablauf** Der *Nutzer* öffnet das Kontextmenü des Streams und wählt den Punkt "Recover Stream".

**Erwartetes & Tatsächliches Verhalten** Der Stream wird wiederhergestellt und die Farbe des Status Symbols ändert sich von rot zu grün.

## 4.2.4 Jobs und Log Files

*Hinweis: Nur in der Rust-Version!*

### Bestehende Jobs einsehen

**Vorbedingung** Der *Nutzer* ist eingeloggt.

**Ablauf** Durch Klick auf den Menüpunkt "Jobs" im linksseitigen Menü wird die Job Seite aufgerufen.

**Erwartetes & Tatsächliches Verhalten** Eine Liste der bestehenden Jobs wird im Hauptfenster angezeigt.

### Existierenden Job löschen

**Vorbedingung** Eine Liste mit mindestens einem existierenden Job wird angezeigt.

**Ablauf** Der *Nutzer* klickt auf das Mülltonnen-Symbol am rechten Ende des zu löschenden Job Eintrags.

**Erwartetes & Tatsächliches Verhalten** Der Job wird aus gelöscht und fortan nicht mehr ausgeführt. Die entsprechende Zeile verschwindet aus der Liste.



### Existierenden Job sofort ausführen

**Vorbedingung** Eine Liste mit mindestens einem existierenden Job wird angezeigt.

**Ablauf** Der *Nutzer* klickt auf das Play-Symbol rechts im Job Eintrag.

**Erwartetes & Tatsächliches Verhalten** Die im Job hinterlegte Aktion wird sofort ausgeführt.

### Einen neuen Job anlegen

**Vorbedingung** Der Dialog für eine beliebige Job-ermöglichte Datenbankanfrage ist geöffnet und der Button "Create A Job" wird angezeigt.

**Ablauf** Der *Nutzer* klickt auf "Create A Job" und der Dialog "Add a new Job" wird angezeigt. Der *Nutzer* wählt einen Zeitpunkt für die erste Ausführung und ein Zeitintervall für die wiederholte Ausführung. Optional kann eine Zusatzinfo angegeben werden, die den Job identifiziert. Anschließend klickt der User auf "Add".

**Erwartetes & Tatsächliches Verhalten** Der Job wird der Liste hinzugefügt und zum nächsten Zeitpunkt automatisch ausgeführt.

### Anzeigen der geloggten Jobs

**Vorbedingung** Der Nutzer ist eingeloggt.

**Ablauf** Der Nutzer klickt in der rechten oberen Ecke das Brief-Symbol

**Erwartetes & Tatsächliches Verhalten** Eine Liste der bisher durch Jobs geloggten Einträge für den aktuellen *Nutzer* wird angezeigt.

## 4.2.5 User Management

**Hinweis:** Nur in der Rust-Version möglich.

### User Management öffnen



**Vorbedingung:** Der *Nutzer* hat auf „User Management“ geklickt.

**Ablauf:** Der *Nutzer* klickt auf Button in der Navigations Bar

**Erwartetes & Tatsächliches Verhalten:** *Nutzer* wird weitergeleitet auf die „/user\_management“ Seite.

User Management

Filter

Username ↑	Admin	Java	Reading Rights	Writing Rights	Create Streams	
 Admin	✓	✗	✓	✓	✓	⋮
 Java	✓	✓	✓	✓	✓	⋮

Items per page: 5 1 ~ 2 of 2 < >

Add a New User

### Tabellen Funktionen

**Vorbedingung:** Der *Nutzer* hat „User Management“ offen.

**Ablauf:** Der *Nutzer* klickt auf die Pfeile neben den Spalten Headern. Und kann so nach Username, Admin etc sortieren.

**Erwartetes Verhalten:** Die Tabelle wird geordnet je nachdem was ausgewählt wird.

**Tatsächliches Verhalten:** Nicht immer wird perfekt sortiert.

Username ↑	Admin	Java	Reading Rights	Writing Rights	Create Streams
------------	-------	------	----------------	----------------	----------------

### Kontextmenü

**Vorbedingung:** Der *Nutzer* hat auf „3 vertikalen Punkte“ geklickt.

**Ablauf:** Der *Nutzer* klickt auf Button und es öffnet sich das Kontextmenu.

**Erwartetes & Tatsächliches Verhalten:** *Nutzer* Kontextmenu öffnet sich . Es lässt den gewählten *Nutzer* löschen oder bearbeiten.

### Add a New User

**Vorbedingung:** Der *Nutzer* hat auf „Add a New User“ geklickt.

**Ablauf:** Es öffnet sich ein Dialog Fenster die es ermöglicht einen neuen User anzulegen.

**Erwartetes Verhalten:** *Nutzer* erhält die Möglichkeit einen neuen *Nutzer* anzulegen.Dabei können verschiedenen Attribute gesetzt und verändert werden. Dabei werden einige Optionen automatisch gesetzt falls z.B Admin ausgewählt wird.

**Tatsächliches Verhalten:** Klappt alles soweit , nur wenn Java-Version gewählt wird ist die Allowed Reading/Writing Streams überflüssig.

Add a new User

Username

Password

☐ Admin

☐ Java

☐ Create Streams

☐ Full Reading Rights

☐ Full Writing Rights

Allowed Reading Streams

Allowed Writing on Streams

Add a New User

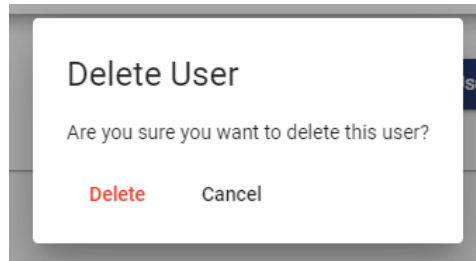
Cancel

### Delete User

**Vorbedingung:** Der *Nutzer* hat auf die drei vertikalen Punkte geklickt und das Kontextmenu geöffnet.

**Ablauf:** Es öffnet sich ein Dialog Fenster das fragt ob man sicher ist den *Nutzer* zu löschen.

**Erwartetes & Tatsächliches Verhalten:** *Nutzer* wird gelöscht falls Ja gewählt wird.Es öffnet sich eine Snackbar die dies bestätigt. Oder man wählt Cancel und es wird abgebrochen.(Snackbar öffnet sich nicht immer)

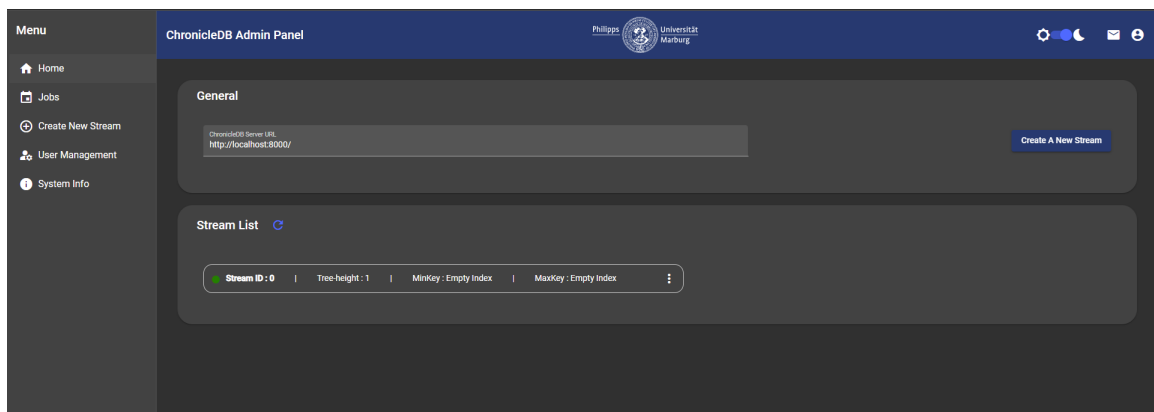


### 4.2.6 Dark Mode

**Vorbedingung:** Der Light-Mode ist an.

**Ablauf:** Der Nutzer klickt auf den Dark-Mode Toggle.

**Erwartetes & Tatsächliches Verhalten:** *Nutzer* Die ganze Applikation wird dunkel. Der Dark Mode wird angewandt



### 4.2.7 Routing

#### Java Routing

**Vorbedingung:** Der Benutzer ist in der Java-Version.

**Ablauf:** Der *Nutzer* klickt auf verschiedenen Buttons.

**Erwartetes & Tatsächliches Verhalten:** In der URL wird sichtbar das richtig geroutet wird. Zum Beispiel bei Create Stream ist der richtige Pfad „.../java/create\_stream“

#### Rust Routing

**Vorbedingung:** Der Benutzer ist in der Rust-Version.

**Ablauf:** Der *Nutzer* klickt auf verschiedenen Buttons.

**Erwartetes & Tatsächliches Verhalten:** In der URL wird sichtbar das richtig geroutet wird. Zum Beispiel bei Create Stream ist der richtige Pfad „.../rust/create\_stream“

#### Nichtexistierende URL

**Vorbedingung:** Der Benutzer ist eingeloggt.

**Ablauf:** Der *Nutzer* fügt der URL „/invalidABCD“ an.

**Erwartetes & Tatsächliches Verhalten:** Der Nutzer wird auf die Hauptseite geleitet.

**Nicht erlaubte URL**

**Vorbedingung:** Der Benutzer ist eingeloggt und hat keine Rechte, Streams zu erstellen.

**Ablauf:** Der *Nutzer* fügt der URL „/create\_stream“ an.

**Erwartetes & Tatsächliches Verhalten:** Der Nutzer wird auf die „No Access“ Seite geleitet.

# 5 Abschlussbericht

## 5.1 Zusammenfassung

Gemäß den Anforderungen kann ein *Nutzer* einerseits bei der *Rust*-Version neue Streams erstellen, Daten einfügen und auslesen, Infos anzeigen lassen und Jobs erstellen, die automatisch im Backend ausgeführt werden. Die Ergebnisse kann sich der Nutzer anzeigen lassen, kopieren und löschen. Andererseits kann er bei der *Java*-Version auch Streams erstellen, Daten einfügen und löschen.

Ebenso haben wir eine Nutzerverwaltung im Backend, in der man für jeden Nutzer genau definieren kann, bei welchen Streams er Lese- und Schreibrechte besitzt, welche Chronicle Version er benutzen soll und ob er Streams erstellen darf. Dabei unterscheiden wir auch zwischen Usern und Admins, welche entsprechend andere Nutzer verwalten dürfen.

## 5.2 Benutzerdokumentation

### 5.2.1 Benötigte Abhängigkeiten

#### Backend

Siehe *requirements.txt* im Ordner *Backend*.

- aniso8601: Version 8.0.0
- appdirs: Version 1.4.3
- APScheduler: Version 3.8.1
- attrs: Version 19.3.0
- Automat: Version 0.8.0
- backports.entry-points-selectable: Version 1.1.1
- backports.zoneinfo: Version 0.2.1
- blinker: Version 1.4
- certifi: Version 2019.11.28
- chardet: Version 3.0.4
- click: Version 7.1.2
- cloud-init: Version 21.2
- colorama: Version 0.4.3
- command-not-found: Version 0.3
- configobj: Version 5.0.6
- constantly: Version 15.1.0
- cryptography: Version 2.8
- dbus-python: Version 1.2.16
- distlib: Version 0.3.3

- distro: Version 1.4.0
- distro-info: Version =0.23ubuntu1
- entrypoints: Version 0.3
- filelock: Version 3.3.2
- Flask: Version 1.1.2
- Flask-Cors: Version 3.0.10
- Flask-Jsonpify: Version 1.5.0
- Flask-RESTful: Version 0.3.8
- Flask-SQLAlchemy: Version 2.4.3
- greenlet: Version 1.1.2
- httplib2: Version 0.14.0
- hyperlink: Version 19.0.0
- idna: Version 2.8
- importlib-metadata: Version 1.5.0
- incremental: Version 16.10.1
- itsdangerous: Version 1.1.0
- Jinja2: Version 2.11.2
- jsonify: Version 0.5
- jsonpatch: Version 1.22
- jsonpointer: Version 2.0
- jsonschema: Version 3.2.0
- keyring: Version 18.0.1
- language-selector: Version 0.1
- launchpadlib: Version 1.10.13
- lazr.restfulclient: Version 0.14.2
- lazr.uri: Version 1.0.3
- make-response: Version 1
- MarkupSafe: Version 1.1.1
- more-itertools: Version 4.2.0
- netifaces: Version 0.10.4
- oauthlib: Version 3.1.0
- pexpect: Version 4.6.0
- pipenv: Version 2021.11.9
- platformdirs: Version 2.4.0
- pyasn1: Version 0.4.2

- pyasn1-modules: Version 0.2.1
- PyGObject: Version 3.36.0
- PyHamcrest: Version 1.9.0
- PyJWT: Version 1.7.1
- pymacaroons: Version 0.13.0
- PyNaCl: Version 1.3.0
- pyOpenSSL: Version 19.0.0
- pyrsistent: Version 0.15.5
- pyserial: Version 3.4
- python-apt: Version 2.0.0+ubuntu0.20.4.5
- python-dateutil: Version 2.8.2
- python-debian: Version =0.1.36ubuntu1
- pytz: Version 2020.1
- pytz-deprecation-shim: Version 0.1.0.post0
- PyYAML: Version 5.3.1
- requests: Version 2.22.0
- requests-unixsocket: Version 0.2.0
- SecretStorage: Version 2.3.1
- service-identity: Version 18.1.0
- simplejson: Version 3.16.0
- six: Version 1.15.0
- sos: Version 4.1
- SQLAlchemy: Version 1.3.18
- ssh-import-id: Version 5.10
- systemd-python: Version 234
- Twisted: Version 18.9.0
- tzdata: Version 2021.5
- tzlocal: Version 4.1
- ubuntu-advantage-tools: Version 27.2
- ufw: Version 0.36
- unattended-upgrades: Version 0.1
- urllib3: Version 1.25.8
- virtualenv: Version 20.10.0
- virtualenv-clone: Version 0.5.7
- wadllib: Version 1.3.3
- Werkzeug: Version 1.0.1
- zipp: Version 1.0.0
- zope.interface: Version 4.7.1

## Frontend

- @angular/animations: Version 12.2.0
- @angular/cdk: Version 12.2.12
- @angular/common: Version 12.2.0
- @angular/compiler: Version 12.2.0
- @angular/core: Version 12.2.0
- @angular/forms: Version 12.2.0
- @angular/material: Version 12.2.12
- @angular/platform-browser: Version 12.2.0
- @angular/platform-browser-dynamic: Version 12.2.0
- @angular/router: Version 12.2.0
- @auth0/angular-jwt: Version 5.0.2
- @types/crypto-js: Version 4.0.2
- crypto-js: Version 4.1.1
- rxjs: Version 6.6.0
- tslib: Version 2.3.0
- zone.js: Version 0.11.4

## ChronicleDB

- Rust: Version 0.9.66
- Java: Spring Boot Version 2.4.5

### 5.2.2 Installationsanleitung

1. Repository clonen
2. Chronicle Java und Rust ausführen
3. Python installieren, falls noch nicht vorhanden und *virtual environment* erstellen
4. Dependencies aus *chronicleDB-admin/backend/requirements.txt* für Python installieren.
5. *chronicleDB-admin/backend/chroniclebackend.py* ausführen
6. Angular CLI installieren
7. Konsole in dem Ordner *chronicleDB-admin* öffnen
8. Dependencies mit `npm install` installieren lassen
9. Frontend mit `ng serve` builden
10. Die Website ist unter folgender URL erreichbar: `http://localhost:4200/`



## 5.3 Entwicklerdokumentation

### 5.3.1 Kurzbeschreibung der Komponenten

- **navigation** dient als „Oberste“ Komponente und beinhaltet die Kopfleiste, das Menü und die jeweilige Komponente, die über die URL aktuell angesteuert wird.
- **page-home/dashboard** mit den Unterkomponenten *card-general-stream* und *stream-list* ist die Start- / Hauptseite, die der Nutzer sieht. Sie beinhaltet generelle Einstellungsmöglichkeiten, wie z.B. die URL von Chronicle und eine Liste der Streams. Diese Streamliste repräsentiert die Rust-Version und wird nur dann angezeigt, wenn der Nutzer die Rust-Version benutzt.
- **page-create-stream/create-stream** ist jene Komponente, die von der Startseite aus aufgerufen wird, wenn man einen neuen Stream in der RUST-Version erstellen will. Sie beinhaltet eine Komponente, in der man allgemeine Einstellungen für den Stream vornehmen kann (*card-stream-properties*) und eine, in der man den Stream-Elemente konfiguriert (*card-stream-event-properties*).
- **page-insert-data/insert-data-tab-menu** ist die Komponente, die ausgehend von der Rust-Stream Liste der Hauptseite als Dialog geöffnet wird, wenn man Daten in ein Stream einfügen möchte. Die Komponente selber stellt ein Tab-Menü dar, wo man auswählen kann, ob man Daten manuell einfügen (*insert-data-manually*) möchte oder ob man eine Datei hochladen möchte (*upload-data*). Beim Manuellen Einfügen wird entsprechend der Streamdefinition Input-Felder Dynamisch angezeigt, welche in der Komponente *insert-data-event-element* umgesetzt sind.
- **show-right-flank** wird als Dialog angezeigt, wenn der Nutzer in der Rust-Streamliste bei einem Stream die Option „Show Right Flank“ auswählt.
- **time-travel** wird als Dialog angezeigt, wenn der Nutzer in der Rust-Streamliste bei einem Stream die Option „Time Travel“ auswählt.
- **stream-info** wird als Dialog angezeigt, wenn der Nutzer in der Rust-Streamliste bei einem Stream die Option „Stream Info“ auswählt.
- **page-jobs** ist jene Seite, die aufgerufen wird, wenn ein Nutzer der Rust-Version seine Jobs anzeigen lassen möchte. Sie beinhaltet eine Tabelle mit allen aktuellen Jobs.  
Die Komponente *add-job* wird von anderen Komponenten über den JobService benutzt (z.B. Get-RightFlankComponent), um neue Jobs zu erstellen, die dann in ebengenannter Tabelle angezeigt werden.
- **page-login** ist die Loginseite.
- **page-messages** ist jene Seite, die aufgerufen wird, wenn ein Nutzer der Rust-Version in der Rust-Version seine Job-Ergebnisse über den Message-Button anzeigen will. Die Komponente selbst ist eine Liste von *job-results*.
- **page-system-info** ist jene Seite, die aufgerufen wird, wenn ein Nutzer der Rust-Version im Menü den Button „System Info“ betätigt.
- **page-user-management/user-management** ist die Seite, die angezeigt wird, wenn ein Admin im Menü den Button „User Management“ betätigt. Es beinhaltet eine Tabelle, in der alle Nutzer mit ihrer Konfiguration dargestellt werden. Die Komponente (Dialog) *add-user* wird dabei angezeigt, wenn ein neuer Nutzer erstellt werden soll.
- **no-access** wird angezeigt, wenn ein Nutzer eine Webseite aufruft, die er nicht sehen darf!
- **java-page-create-stream** ist analog zu *page-create-stream/create-stream* jene Komponente, die angezeigt wird, wenn ein Nutzer der Java-Version einen neuen Stream erstellen will.
- **java-stream-list** ist die Java-Version der Streamliste, die im Dashboard angezeigt wird, wenn ein Nutzer die Java-Version benutzt.

### 5.3.2 Erweiterungsszenarien

#### Upload-Data

Bisher kann man nur Dateien mit einem bestimmten Format hochladen, welches dann geparsed und an Chronicle gereicht wird. Eine Erweiterung wäre, dass man hier noch weitere Formate unterstützt und die Datei bereits im Vorfeld auf Fehler untersucht und diese anzeigt.

#### Datenvalidierung

Bisher wurde noch nicht die Größe der Zahlen überprüft, ob diese z.B. in einen 8bit Integer passen. (Lediglich die Syntax wird getestet)

#### Nutzer bearbeiten

Man kann bisher nur Nutzer erstellen und löschen. Daher wäre eine Erweiterung, dass man diese auch im Nachhinein noch bearbeiten kann.

#### Java Nutzerverwaltung

Bisher gibt es nur bei der Rust-Variante Einschränkungen auf die Lese- und Schreibrechte von Streams.

#### Java Jobs

Bisher gibt es nur bei der Rust-Variante die Möglichkeit, Jobs zu erstellen.

#### Benachrichtigungen

Ebenso wäre es noch möglich, dass ein Nutzer eine Email hinterlegen muss, sodass man ihn benachrichtigen könnte, wenn Probleme mit seinem Konto aufgetreten sind (z.B. Fehlgelagener Login) oder bestimmte Jobs ein Resultat geliefert haben.

### 5.3.3 Zu beachtende Codekonventionen

Komponenten, die eine eigene URL haben, bzw. als eigenständige Seite angezeigt werden, bekommen das Präfix „page-“. Analog bekommt eine Komponente, die nur als Bestandteil einer anderen Komponente in einer Karte vorkommt das Präfix „card-“ und befindet sich in dem Ordner der Oberkomponente.

### 5.3.4 Bekannte Fehler / fehlende Eigenschaften

Bei Create-Stream (Rust-Variante) sind bei den Stream-Properties teilweise falsche Eingaben möglich, die nicht überprüft wurden (z.B. Logical Block Size, Macro Block Size).

## 5.4 Erfahrungsbericht

Wie so oft standen wir im Laufe des Projekts vor Entscheidungen die wir heute anders treffen würden. Eine die nicht dazu gehört war es Angular 12.0 als Framework zu benutzen unter TypeScript. Die vorgefertigten Angular Material Komponenten templates haben einen schnellen und gut dokumentierten Einstieg ermöglicht in die Welt des Front-End Development. Gerade die weit verbreiteten Tutorials und Anleitungen waren sehr hilfreich, um sich Wissen und Methoden anzueignen. Vor allem sollte man bevor man versucht etwas selbst per Hand zu coden, einmal schauen ob das Framework dafür nicht schon Methoden und Komponenten hat. Ein konkretes Beispiel dafür sind „Reactive Forms“ um Nutzereingaben aufzunehmen. Angular Material bietet hierfür wunderbare Werkzeuge, die uns aber leider zu Beginn des Projekts nicht bekannt waren.

Später als wir eine Nutzerverwaltung implementieren sollten, war es unabdingbar ein Backend anzuschließen. Wir entschieden uns für ein selbst geschriebenes Python Backend zur Verwaltung und Verarbeitung von Nutzer-Eingaben. Diese Entscheidung haben wir teilweise bereut, da wir im Endeffekt viel Arbeit hineinstecken mussten, dass die Kommunikation einwandfrei läuft. Selbst in kleineren Aspekten wie Urzeiten gab es (kleinere) Probleme mit z.B. den Zeitzeonen, die die eine Programmiersprache berücksichtigt

hat und die andere nicht. Andererseits hatten wir so die Möglichkeit, die Funktionen maßgeschneidert an unsere Anforderungen anzupassen und ebenso tiefer gehende Funktionen einzubauen (z.B. automatisierte Abfragen / Jobs im Backend).

Rückblickend wäre es sinnvoll gewesen, für gewisse Teile (z.B. Nutzerverwaltung) vordefinierte Sachen zu verwenden, um sich die Arbeit zu erleichtern. So hatten wir jedoch auch die Möglichkeit, in diesen Bereichen was zu lernen...

Wir können zukünftigen Frontend Entwicklern Angular ans Herz legen. Wir haben damit Großteils nur positive Erfahrungen gemacht.

### 5.5 Protokolle und Entwicklungsverlauf

Die Protokolle der Meetings sowie die Screenshots, die die Entwicklung darstellen sind in dem Ordner *Screenshots* und in der Datei *Protokolle der Treffen.md* zu finden.